Christophe Bobda
Senem Velipasalar *Editors*

Distributed Embedded Smart Cameras

Architectures, Design and Applications

Springer

Distributed Embedded Smart Cameras

Christophe Bobda · Senem Velipasalar
Editors

Distributed Embedded Smart Cameras

Architectures, Design and Applications

Springer

Editors

Christophe Bobda
University of Arkansas
Fayetteville, AR
USA

Senem Velipasalar
Syracuse University
Syracuse, NY
USA

ISBN 978-1-4614-7704-4 ISBN 978-1-4614-7705-1 (eBook)
DOI 10.1007/978-1-4614-7705-1
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2014941695

Preface

Driven by technological improvements, the noninvasive nature of cameras, affordable prices, security concerns, and federal government grants to aid fighting terrorism, camera utilization has become an integral component of our daily lives. Cameras are pervasively used for surveillance and monitoring applications such as traffic monitoring, monitoring commercial vehicles, and surveillance at schools and parks.

Current video surveillance systems operate in Close-Circuit Television (CCTV), where data collected by camera are analyzed by operator or stored on a central server for further processing. While capabilities of video-processing and communication systems have significantly increased in the recent years due to advances in video sensing technologies, the amount of data being produced by such systems is becoming increasingly difficult to manage. For instance, video sequence of HDTV format (1920×1080 pixels) at 30 frames per second with 24 bits depth per pixel requires 0.5 gigabytes per second in uncompressed form. The department of defense estimates a 5,000 times increase in the amount of sensor data for future assets in Theater and the amount of data produced by development systems like the DARPAs Autonomous Real-Time Ground Ubiquitous Surveillance Imaging System (ARGUS-IS) will be comparable to the 72 gigabytes that the human vision sends to the brain. Current and future systems will not be able to provide the bandwidth required to transport increasingly higher amounts of data. Furthermore, analyzing and understanding terabytes of data in real-time require computer architectures with capability far beyond the currently available systems and processing power of backend servers. Current and future communication systems, even with the most advanced video compression architecture, will not be able to provide the required bandwidth to transport such a datastream.

Smart cameras are capable of analyzing video data in the camera, close to the sensor, thus limiting the amount of data to be transported. Enhancing smart cameras with communication allows for collaborative scene and event analysis, usually in real-time, which further reduces the need for a central server. Events detected in one camera can be transmitted to a surrounding camera for contextual and geographical-related interpretation. Smart cameras can operate in stationary or mobile mode.

The benefit of this approach is illustrated in the following two real-life case studies. In the first example (Fig. 1a) a set of cameras are used for fall detection.

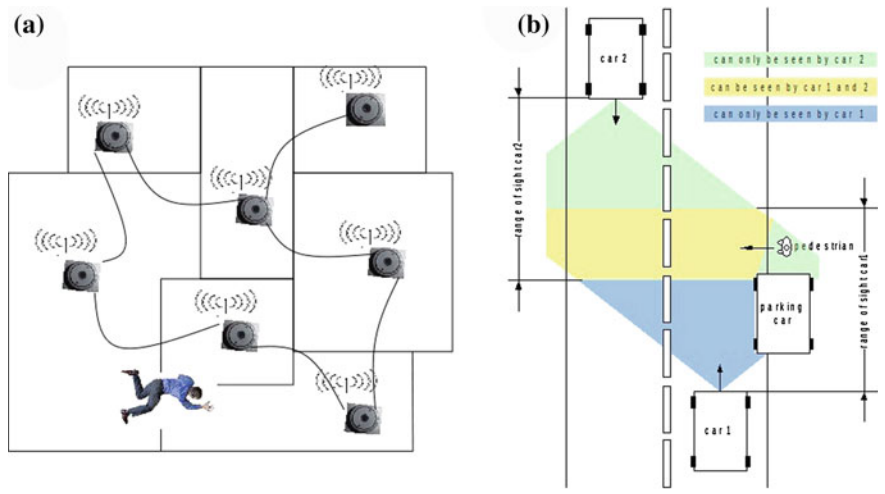

(a)
(b)
car 2
can only be seen by car 2
can be seen by car 1 and 2
can only be seen by car 1
range of sight car2
pedestrian
parking car
range of sight car1
car 1

Fig. 1 Efficient visual perception with cooperative cameras **a** Cooperation detection of emergency situations in nursing homes **b** Occluded pedestrian becomes visible through cooperation

Each camera can cover only a partial area of the entire scene. To detect a fall the entire body must be available. In our example, only part of the body can be seen by each camera, none of which can infer a fall with this partial information. The collaboration will allow the camera to exchange partial information, which can be combined to infer the fall.

In the second example, consider cameras mounted on cars, with car-to-car communication capabilities. Figure 1b shows a traffic situation in which a pedestrian crossing the street cannot be seen by the driver of an incoming car. The pedestrian is occluded by a stationary car, which has the pedestrian in its field of view. This information can be sent to surrounding cars, so that their drivers can avoid an accident.

The design and deployment of collaborative smart cameras is a difficult task, which can be tackled only with multidisciplinary expertise. At the lowest level within single camera, VLSI expertise is required for designing optimal and smart CCD that can provide low-level operation on pixels. Computer architects are needed to design dedicated systems that reflect the behavior of most image understanding algorithms. The image processing expert will provide sound image understanding algorithms and machine learning method for information fusion. Communication expertise is required to design better protocols and paradigms for real-time information exchange.

Despite its relatively young age, research in multi-camera networks is becoming increasingly popular. The ACM/IEEE international conference on "Distributed Smart Cameras" is established and is the venue for researchers and developers to convene and discuss the most recent advances in the field. A comprehensive survey of the activities in this field has been provided in the book by Aghajan and Cavallero [1].

In this book, emphasis has been placed on embedded architecture for smart cameras and mobility, thus complementing the books by Aghajan et al. [1] and Bhanu et al. [2], whose emphasis has been mostly on fundamentals, algorithm and software, and stationary systems.

This book is organized into three main parts that better categorize the contribution. The first part deals with architecture and design flow, the second part handles smart cameras in the mobile environment, and the last part applications.

Part I consists of four chapters and covers architectures of smart cameras and their design flow. In Chap. 1, Wolf provides an overview of platforms and architectures for embedded smart cameras. The contribution analyzes algorithmic needs at high-level and presents various platforms with their expected performance. In Chap. 2, Ahmadinia and Watson give an overview of System-on-Chip solutions for smart camera. Chip miniaturization and increased density is leading the way to system integration in a single chip with the advantage of performance, robustness, miniaturization, and power saving. Ahmadinia and Watson investigate the landscape and present the recent development in System-on-Chip solution for imaging. Emphasis is put on processors, communication infrastructure, and memory. In Chap. 3 Bobda et al. present the benefit of using reconfigurable devices such as FPGA in embedded smart camera. Their smart camera is presented as prototyping platform, along with the hardware/software partitioning for vision application. A hardware middleware architecture is introduced, which has the capability of reducing communication delay among smart cameras, while reducing design burdens and increasing interoperability. Part I is completed with the contribution of Mefenza et al., which deals with design and verification of embedded smart cameras. An integrated design and verification based on OpenCV and SystemC is presented, which allows entire hardware/software systems to be captured at a high level of abstraction. Subsequent refinements are then performed until the final implementation, with the possibility of prototyping the entire design in the RazorCam, an FPGA-based camera designed at the University of Arkansas.

Part II deals with smart cameras in mobile environment. In Chap. 5, Martinel et al. give an overview of the landscape and recent developments in distributed mobile computer vision. They use case studies (augmented reality and surveillance) to illustrate the advantages of mobile and distributed computer vision. Future challenges on the integration of mobile devices as node in visual sensor networks are discussed. In the seventh chapter, Velipasalar et al. discuss the use of wearable cameras for automatic fall detection and activity classification. Methods for event detection and classification are explained and case studies conducted with the embedded CITRIC platform. In Chap. 6, Almagambetov and Velipasalar use embedded smart cameras mounted on vehicles to detect and track taillights of vehicles in front, recognize common alert signals, and counting the cars passing on both sides of the vehicle. They present the design and implementation of a robust and computationally lightweight algorithm for a real-time mobile vision system. Their emphasis is on low-power, with process scenes entirely on the microprocessor of an embedded smart camera.

Applications of distributed smart cameras are the subject of Part III. In Chap. 8, Wang and Aghajan first review detection and tracking approaches using single camera. Thereafter, they explore methods to combine images from multiple sources to enable tracking in a distributed smart camera network. As application, room occupancy estimation is used. In Chap. 9, Lee et al. discuss a self-organized and scalable multiple-camera system for tracking across cameras with nonoverlapping field of views. Using GPS locations of uncalibrated cameras, detection of camera link relationships is done based on routing information provided by Google Maps. Unsupervised learning is used here to compute network properties such as transition time and brightness transfer function. Self-organization is enforced by unsupervised learning, which allows to improve the scalability of the system. In Chap. 10, Chen et al. discuss the use of soft-biometric features for person identification. Such features, which are invariant to illumination and view changes, are integrated into the feature representation of a target. A reference set is used for track association, instead of the appearance of targets in different cameras. The reference set acts as a basis to represent a target by measuring the similarity between the target and each of the individuals in the reference set. In Chap. 11, Talla et al present a new technique for processing large and complex images, especially SAR images. The method performs computation in parallel and is based on a new modeling of textural parameters of a generic order $n > 1$ equivalent to the classical formulation, but which is no longer based on the co-occurrence matrix of order $n > 1$. While the work at first glance seems not connected to the topic of the book, a closer look shows that the presented parallel approach could be used for distributed processing and fusion of large images taken from different satellites or different UAVs with different perspectives. This extends the applicability field of multi-camera network to distributed remote sensing.

In the last chapter, Banerjee et al. present the use of smart camera as part of a set-up for distracted driver avoidance. The camera is used to analyze the pose and behavior of car occupancy to infer dangerous driving situations.

Fayetteville, AR, March 2014 Christophe Bobda
Syracuse, NY Senem Velipasalar

References

1. Aghajan H, Cavallaro A (2009) Multi-camera networks: principles and applications. Academic Press, London
2. Bhanu B, Ravishankar C, Roy-Chowdhury A, Aghajan H, Terzopoulos D (2011) Distributed video sensor networks. Springer, New York. http://books.google.com/books?id=ToPU3XFV9gIC

Contents

Contributors

Hamid Aghajan Department of Electrical Engineering, Stanford University, Stanford, USA

Ali Ahmadinia School of Engineering and Built Environment, Glasgow Caledonian University, Glasgow, UK

Akhan Almagambetov Norwich University, Northfield, VT, USA

Le An University of California, Riverside, CA, USA

Nilanjan Banerjee University of Maryland Baltimore County, Baltimore, MD, USA

Bir Bhanu University of California, Riverside, CA, USA

Christophe Bobda Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR, USA

Xiaojing Chen University of California, Riverside, CA, USA

Chun-Te Chu Department of Electrical Engineering, University of Washington, Seattle, WA, USA

Amol Deshpande University of Maryland Baltimore County, Baltimore, MD, USA

Albert Dipanda Electronique, Computer and Image Engineering Laboratory (Le2i), University of Burgundy, Burgundy, France

Zhijun Fang School of Information Technology, Jiangxi University of Finance and Economics, Nanchang, Jiangxi, China

Janvier Fotsing Electronique and Signal Processing Laboratory (LETS), National High School of Engineering, University of Yaounde 1, Yaounde, Cameroon

Jenq-Neng Hwang Department of Electrical Engineering, University of Washington, Seattle, WA, USA

Kuan-Hui Lee Department of Electrical Engineering, University of Washington, Seattle, WA, USA

Younggun Lee Department of Electrical Engineering, University of Washington, Seattle, WA, USA

Anvith Mahabalagiri 4-206 Center for Science and Technology, Syracuse University, Syracuse, NY, USA

Niki Martinel Univeristy of Udine, Udine, Italy

Michael Mefenza Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR, USA

Christian Micheloni Univeristy of Udine, Udine, Italy

Koray Ozcan 4-206 Center for Science and Technology, Syracuse University, Syracuse, NY, USA

Andrea Prati IUAV University of Venice, Venice, Italy

Mahbubur Rahaman University of Maryland Baltimore County, Baltimore, MD, USA

Ryan Robucci University of Maryland Baltimore County, Baltimore, MD, USA

Narcisse Talla Tankam Automatic and Computer Engineering Laboratory (LAIA), Fotso Victor University Institute of Technology Bandjoun, University of Dschang, Dschang, Cameroon; Electronique and Signal Processing Laboratory (LETS), National High School of Engineering, University of Yaounde 1, Yaounde, Cameroon

Emmanuel Tonyé Electronique and Signal Processing Laboratory (LETS), National High School of Engineering, University of Yaounde 1, Yaounde, Cameroon

Senem Velipasalar 4-206 Center for Science and Technology, Syracuse University, Syracuse, NY, USA

Zixuan Wang Department of Electrical Engineering, Stanford University, Stanford, USA

David Watson School of Engineering and Built Environment, Glasgow Caledonian University, Glasgow, UK

Marilyn Wolf School of ECE, Georgia Institute of Technology, Atlanta, GA, USA

Franck Yonga Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR, USA

Ali Akbar Zarezadeh Dspace GmbH, Paderborn, Germany

Acronyms

ACE	Adaptive Communication Environment
ADAS	Advanced Driver-Assistance Systems
API	Application Programming Interfaces
ASIC	Application-Specific Integrated Circuit
BTF	Brightness Transfer Function
CCD	Charge-Coupled Device
CCTV	Closed-Circuit Television
CIDA	Component Interconnect for Data Access
CMOS	Complementary Metal Oxide Semiconductor
CORBA	Common Object Request Broker Architecture
CPP	Central Purpose Processing
CPU	Central Processing Unit
DCT	Discrete Cosinus Transform
DSP	Digital Signal Processor
EM	Expectation Maximization
FOV	Field of View
FPGA	Field Programmable Gate Array
GIOP	General Inter ORB Protocol
GMM	Gaussian Mixture Model
GPIO	General Purpose Input Output
GPS	Global Positioning System
GPU	Graphics Processing Unit
HardORB	Hardware Object Remote Broker
HOG	Histogram of Oriented Gradients
HPD	High-Performance Device
IDCT	Inverse Discrete Cosinus Transform
IDL	Interface Definition Language
IOR	Interoperable Object Reference
ITS	Intelligent Transportation Systems
LBP	Local Binary Patterns
LPD	Low-Performance Device
LVDS	Low-Voltage Differential Signaling
MAC	Media Access Control
MonoSLAM	Monocular Simultaneous Localisation And Mapping
MPI	Message Passing Interface

MPSoC	Multiprocessor System on Chip
nAUC	Normalized Area Under Curve
NoC	Network on Chip
ORB	Object Remote Broker
PDA	Personal Digital Assistant
PHOG	Pyramid of Histogram of Oriented Gradients
PICSY	Potsdam Intelligent Smart Camera
PIR	Passive Infrared
PTZ	Pan Tilt Zoom
RADAR	RAdio Detection And Ranging
LiDAR	LIght Detection And Ranging
RANSAC	RANdom SAmple Consensus
RHT	Randomized Hough Transform
RISC	Reduced Instruction Set Computer
RPR	Relevant Painting Region
RTL	Register Transfer Level
SDI	Streaming Data Interface
SDK	Software Development Kit
SDR	Software-Defined Radio
SIFT	Scaled-Invariant Feature Transform
SIMD	Single Instruction Multiple Data
SoC	System on Chip
SURF	Speeded Up Robust Features
SVM	Support Vector Machines
SWAP	Size, Weight and Power
TAO	The ACE ORB
TLM	Transaction-Level Modeling
TTF	Tangent Transfer Function
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
UVM	Universal Verification Methodology
VLIW	Very Long Instruction World
VLSI	Very Large-Scale Integration
VO	Vehicle Object
VSP	Video Signal Processor

Part I
Architectures and Technologies for Embedded Smart Cameras

Chapter 1
Platforms and Architectures for Distributed Smart Cameras

Marilyn Wolf

Abstract Embedded computer vision places huge computational demands on smart cameras; in addition, these systems must often be designed to consume very low power and be inexpensive to manfuacture. In this chapter, we consider computational platforms for both smart cameras and networks of smart cameras. A platform is a combination of hardware and software that provides a set of features and services for an application space. We first compare a broad range of computing fabrics suitable for embedded computer vision: FPGAs, GPUs, video signal processors, and heterogeneous multiprocessor system-on-chip. We then look at approaches to the design of a platform for distributed services in a smart camera network.

1.1 Introduction

Computer vision is a highly computationally-intensive task; there is effectively no limit to the amount of computation that one can reasonably use in complex vision tasks. Smart cameras, as platforms for embedded computer vision, must be designed to provide high-performance computing. And like other embedded computing systems, they must be designed to meet other constraints as well: low energy and power utilization, real-time performance, and low cost.

In this chapter, we will examine platforms for embedded computer vision. A platform is a combination of hardware and software that provides operations and services. Platform-based design is widely used in the semiconductor industry, with smart phone platforms being a prime example. Some of the platforms used for smart cameras are based on platforms originally designed for multimedia computing. Computer vision and multimedia share some common characteristics since both deal with

M. Wolf (✉)
School of ECE, Georgia Institute of Technology, 777 Atlantic Drive NW,
Atlanta, GA 30332, USA
e-mail: wolf@ece.gatech.edu

image sequences. However, many multimedia compression algorithms are designed to require only integer arithmetic whereas many computer vision algorithms require floating-point arithmetic.

We can extend the notion of a platform beyond a single camera to a network of cameras. A uniform platform that provides abstract services, such as data communication, allows algorithm designers to develop applications efficiently. Not only does a distributed computing platform abstract away details of network interfaces, but it can also provide higher-level services such as data synchronization protocols that are non-trivial to correctly implement.

We will start with a brief discussion of the embedded computer vision design space. Section 1.3 describes the variety of computational fabrics available for smart cameras. Section 1.4 expands on the discussion of fabrics to consider the design of heterogeneous multiprocessors, notably multiprocessor system-on-chip. Section 1.5 describes platforms and techniques for distributed smart camera networks.

1.2 The Design Space

We can define several traditional computer system metrics relevant to embedded vision systems. The term *performance* is traditionally used in computer system design to refer to execution speed. Throughput and latency are both important computing performance metrics that are relevant to embedded vision. Throughput measures the number of computations finished per unit time; frames per second is a simple example of a vision-oriented throughput metric. Relating algorithm-oriented throughput metrics to CPU-level metrics such as clock speed is challenging due to the complexity of both the algorithms and of processor architectures. Latency measures the time required to finish a given calculation. Modern processors provide higher throughput by pipelining several operations [25]. Latency at the algorithmic level often has to do with the temporal relationships between inputs and outputs. For example, a filter that compares successive frames will have a latency of at least two frame periods independent of the capabilities of the underlying platform.

Energy and power are important metrics for virtually all modern computer systems. Energy consumption determines battery life of battery-powered systems; power consumption determines heat generation, which affects the system cost and can even determine whether the processor can operate [61]. These metrics are very important to embedded computer vision systems. The cost of delivering power to a smart camera is a significant component of installation cost. Computer vision may be performed using cell phones that are battery powered. Physically small smart cameras can be deployed in many interesting applications and provide new views of the subject that provide better results than the traditional ceiling-mounted cameras. High power consumption causes the camera to generate heat, which may not be compatible with many operating environments. At higher heat levels, active cooling may be required, increasing the camera's size, cost, and noise.

A proper understanding of the characteristics of applications is the key to both performance and energy optimization in software design. The operations and memory characteristics of the application need to be properly matched to the computing platform to meet performance requirements without wasting either hardware resources or energy. Application characteristics are frequently measured using simulation studies; simulation provides the most accurate measurement of both operator and memory parallelism. We do not know of a detailed study of the characteristics of computer vision algorithms. However, we can learn something from studies performed on multimedia algorithms such as video compression. Fritts [17, 18] performed a set of experiments on the original MediaBench benchmark suite. Their experiments showed that the benchmarks exhibit about 10 iterations per loop for video algorithms. Fritts also introduced path ratio as a metric to measure the control content of a loop body, $PR = \frac{n_{exec}}{n_{body}}$, where n_{body} is the total number of instructions in the loop's body and n_{exec} is the number of loop body instructions actually executed. Experiments showed that video benchmarks exhibited a path ratio of about 0.8, indicating that many of these loops contain conditional evaluations. Tallu et al. [53] performed a series of experiments to evaluate the available parallelism in multimedia applications. They created several configurations of the SimpleScalar CPU simulator with different numbers of available resources. The largest CPU had extensive microarchitectural resources, including an issue width of 16, 16 integer ALUs, 8 multipliers, and a 16-kB data cache. These configuraitons allowed them to evaluate the amount of CPU resources that each program could absorb. They measured nine benhmark programs, most of which showed a value of instructions per cycle (IPC) of less than four.

Numerical characteristics are critical to the success of embedded vision systems. Computer vision often uses performance to describe metrics related to accuracy. Numerical methods link the algorithmic and computational views of performance: insufficient numerical accuracy causes vision algorithms to become unacceptably inaccurate; numerical computations performed to unnecessary degrees of accuracy consume energy, chip area, and sometimes execution speed. When vision algorithms are implemented on general-purpose CPUs, IEEE Standard 754 provides several levels of precision [15] and algorithm designers may simply perform all computations at a precision that eliminates accuracy problems. Embedded vision systems, because they must meet multiple metrics such as power consumption, often require more careful attention to the required accuracy of algorithms.

Soderquist and Leeser [50] describe in detail numerical algorithms for division and square root, two closely related operations. They identify several metrics for the evaluation floating-point division: latency, area, throughput, complexity, and interactinos with other operations. (We would also add energy per operation as another important metric that has become increasingly important since the 1997 publication of that article.) They identify two major types of algorithms: multiplicative and subtractive. Based on a series of experiments, they conclude that subtractive algorithms, based on the Sweeney/Robertson/Tocher or SRT algorithms, provide superior overall performance. SRT-based algorithms also make less use of other floating-point resources, making it easier to pipeline operations. Wang and Leeser [59] developed the VFloat library which generates floating-point units for reconfigurable systems. VFloat allows designers to specify the bitwidths for operations and to mix fixed-point and floating-point in a system through the use of conversion modules.

Given these requirements, embedded vision system designers have several choices for the computing platform used at a vision node. We can identify two axes for the implementation space. One, which is often referred to as a fabric, relates to the design and manufacture of digital components. Field-programmable gate arrays (FPGAs), programmable processors, graphics processing units (GPUs), and semicustom VLSI are all fabrics that provide very different benefits and costs. The other major axis relates to architecture, namely the degree of homogeneity or heterogeneity of the computing platform. Heterogeneous multiprocessors are used in many applications because they provide high-levels of performance at lower energy/power and cost. We will see that many embedded computing platforms make use of heterogeneous architectures.

1.3 Fabrics

VLSI technology provides system designers with a variety of computational fabrics, resulting in a wide range of performance/energy/cost trade-offs. The fabric categories described here are not entirely mutually exclusive. For example, FPGAs often make use of small embedded processors and some platform FPGAs include large, semicustom embedded CPUs.

This section describes the varieties of computational fabrics for smart cameras. The next section describes FPGAs, followed by GPUs in Sect. 1.3.2, programmable processors in Sect. 1.3.3, and video signal processors in Sect. 1.3.4.

1.3.1 Field-Programmable Gate Arrays

Modern field-programmable gate arrays (FPGAs) provide not only a great deal of logic, but also on-chip memory and specialized structures for DSP. For example, the Xilinx 7 series family [9] has several features: blocks of 32K-bit dual-ported RAM, ranging from 13 M-bit for the Artix-7 family to 68 M-bit for the Virtex-7 family; and DSP slices with a 25×18 multiplier, 48-bit multiplier, and pre-adder, ranging from 740 DSP units for the Artix-7 family to 3,600 for Virtex-7. The Altera Stratix V family [8] includes embedded memory blocks as well as variable-precision DSP blocks ranging from 9×9 to 54×54, a 64-bit accumulator, internal coefficient memory, and pre-adder/subtractor. Many FPGAs also include specialized components for high-performance networking.

Gudis et al. [22] describe a framework for FPGA-based heterogeneous multiprocessors for computer vision. The FPGA contains a set of accelerators connected by a crossbar. A memory controller connects the accelerators to bulk memory through video DMA controllers; the bulk memory is shared with an ARM host processor. The ARM processor runs a vision service framework that provides an API for video services. The framework provides C++ classes for video drivers, resource management, and vision services. Farabet et al. [16] developed the NeuFlow architecture for FPGA-based computer vision. They use a dataflow style for accelerators; the accelerators are connected in a 2-D mesh. A smart DMA unit connects the mesh to bulk DRAM. The smart DMA provides multiple ports, each of which can operate independently.

1.3.2 Graphics Processing Units

Graphics processing units (GPUs) are widely used in desktops and laptops to perform graphics functions; they are also increasingly used in mobile devices. GPU architectures are still evolving as VLSI device densities increase, but broadly speaking we can identify some common characteristics of GPUs. They make use of single-instruction multiple data (SIMD) architectures; SIMD is well-suited to graphics algorithms because the screen can generally be divided into independent regions. They make use of small, localized memories near the processing elements to provide large memory bandwidth to the function units. They also provide limited-precision floating-point arithmetic.

Relatively little work has been done to implement computer vision algorithms on GPUs. Fung and Mann [19] described the OpenVIDIA library for computer vision algorithms on GPUs. For example, they used the vertex processor, rasterizer, and fragment processor to compute a Hough transform forl line detection. They also implemented a Harris corner detector that could be used for feature tracking. In a separate article [20] they discussed the use of GPUs for computer vision algorithms. They identified several features of GPUs that are useful for computer vision: local caching well-suited to spatially-coherent accesses; on-the-fly integer/floating-point conversions, non-pageable pinned memory, and streams for paralel processing. Li et al. [35] implemented a face detection algorithm based on the Haar transform on an Intel platform that contained both a GPU and CPU. Wang et al. [58] implemented object removal using OpenCL.

Nagendra described a method for analyzing the performance of automotive image sensors and demonstrated results from GPUs [42]. Those experiments benchmarked an object detection algorithm by Viola and Jones [57] using an Nvidia 7800GTX GPU and a 3 GHz Xeon processor. Although this GPU is more powerful than those typically found in mobile applications, they reported speedups of several hundred times over the CPU.

1.3.3 Programmable Processors

Programmable von Neumann processors fetch instructions from a memory, then execute the instruction. Computer architects have devised a wide range of machines that fit the von Neumann model. Computer vision systems often employ three variations: the central processing unit (CPU), digital signal processor (DSP), and video signal processor (VSP). The term *CPU* is a broad brush term for a processor that is not optimized for the application at hand. CPU and reduced instruction set computer (RISC) processor are often treated as synonymous, although not all CPUs are RISC. CPUs are often used to run operating systems and user interfaces; they are also often used to perform tasks that are not amenable to processor optimizations. DSP is also a term that has been used without regard to particular meaning, but the term originally applied to processors with two features: a hardware multiplier and separate memories for instructions and data. Hardware multipliers, although less expensive today, were considered a novelty for quite some time; the AT&T DSP-16 [5], widely regarded as the first DSP, included a multiplier configured for use in multiply-accumulate operations. The DSP-16 also provided separate instruction and data memories, known as a Harvard architecture; this architecture is strictly different from the von Neumann model, which combines instructions and data in a single memory. However, machines with separate instruction and data caches are often classified as DSPs. We will discuss VSPs below; these machines combine some aspects of DSPs with mechanisms for highly parallel instruction execution. After discussing VSPs, we will briefly touch upon enhancements to programmable processors.

Processors can be supplemented by a variety of mechanisms to enhance important operations. Specialized instructions, co-processors, and vector units are all examples of common enhancements to processors.

Many early operations in computer vision systems exhibit regular access patterns and relatively simple numeric properties. For example, the Haar transform is a wavelet transform that computes sums and differences of numbers and division by $\sqrt{2}$. These operations can be efficiently mapped onto loops on integer processors. Given the prevalence of loops in digital signal processing, DSPs typically provide support for efficient loop operation. The TI 674x VLIW DSP [29] uses the SPLOOP mechanism to implement software pipelining. In software pipelining, instructions are generated that perform different phases of several loop operations simultaneously; the code is generated so that each iteration uses a different part of the CPU resources. This technique normally requires recoding the loop into three sections: prolog, kernel, and epilog. The SPLOOP mechanism generates the required sequence of instructions for software pipelining based upon a single iteration of the loop that is kept in an internal buffer. The loop iteration buffer also reduces memory bandwidth and energy requirements.

Many general-purpose processors now include multimedia operations; the Intel MMX instruction set is one such example [44]. These instructions take advantage of the fact that the CPU datapath's carry chain can be split to perform several simultaneous operations on subwords. Such instructions are limited to integer operations on shorter word lengths.

A co-processor is closely tied to the execution unit of the processor. A co-processor responds to an instruction opcode and is dispatched by the execution unit. (In contrast, the accelerators we will discuss below appear more as I/O devices and are not associated with opcodes.) Floating-point operations are often organized as co-processors but more complex operations are also performed in this way. For example, the TI C55x DSP [28] provides processors for pixel interpolation, motion estimation, and for DCT/IDCT.

Vector units were originally developed for scientific computation. Many numerical algorithms are expressed as matrix and vector operations. Vector units directly perform operations such as vector addition and multiplication. Such units are infrequently used in computer vision platforms. However, they are available on other embedded platforms. For example, the Quovira MPC5676R [49], designed for automotive engine control, includes two Power Architecture CPUs and a floating-point vector processing unit.

1.3.4 Video Signal Processors

Video signal processors (VSPs) were originally developed for multimedia applications such as video/audio compression and decompression. VSPs are at the core of embedded computer vision systems.

VSPs are organized as very-long instruction word (VLIW) processors. A VLIW processor has a staticially-scheduled instruction stream; in comparison, superscalar processors use hardware in the CPU's execution unit to dynamically schedule instructions. VLIW instructions that are similar in scope to a RISC instruction are organized into packets. At run time, the processor fetches a packet and executes the instructions, usually simultaneously but in some cases over a few cycles. The compiler's job is to generate instructions and packets that are optimized for performance (and perhaps energy consumption) while adhering to two forms of correctness constraints: the ordering of instructions into packets must not violate the data dependencies implied by the program; and the packets must not violate the allocation constraints imposed by the limitations on available hardware in the CPU.

Several parts of the VSP microarchitecture have received attention: the allocation of PEs, register file organization, crossbars, and off-chip *vs.* on-chip memory. The number of processing elements puts one upper bound on the number of parallel operations that can be performed by the machine. Another upper bound is given by the number of ports on the register file: a binary operator would require three ports, two for the operands and one for the result. The delay of a register file increases superlinearly with the number of ports [14]. The number of processing elements required depends on the balance of other architectural parameters, such as the number of ports in the register file and the clock rate [13]. A VLIW machine relies on the compiler's ability to combine registers and function units in a variety of combinations places stress on the interconnect network between the registers and function units. Crossbars are typically used to connect the registers, function units, and memory in a VSP [12]. VSPs tend to have flatter memory hierarchies than do general-purpose CPUs because multi-level caches tend to provide smaller benefits for the access patterns of multimedia applications [14].

Modern VSPs such as the TI 320C6727B [30] illustrate these principles:

- The CPU contains a total of eight function units organized into two clusters. Each cluster contains a data addressing unit for data transfers, a multiplier, and arithmetic, logical, and branch operators.
- Each cluster has a 32-entry register file that is accessible to all the function units in the cluster.
- The processor supports floating-point multiplication, addition, and subtraction.
- On-chip memory provides RAM for program and ROM for program and data. A program cache is also available.

1.4 Heterogeneous Architectures

Multiprocessors are often used in high-performance computing, including embedded computer vision. A single processor, even a VLIW machine, often does not provide enough computational throughput for complex vision applications. Multiprocessors, beyond providing sheer computational horsepower, allow a complex vision pipeline to be spread across multiple processors, providing an efficient mapping from application to architecture. Multiprocessors may be implemented in custom VLSI or FPGAs.

Multiprocessors are now common in desktop and laptop computers. These processors are known as multicores because they use several identical CPUs, each known as a core, connected by a network to a set of caches. This homogeneous architecture allows operating systems to migrate tasks and manage performance, power consumption, and thermal behavior. However, multicores are not widely used in embedded applications. Austin et al. [3] evaluated a workload consisting of applications that included speech recognition, computer vision, video compression, graphics, and communications; based on trends in 2004, they showed that this workload would require computation rates about 16 times that provided by a 2 GHz Intel Pentium 4 processor. Multicores are also less energy-efficient than specialized alternatives. As a result, many embedded computing platforms are heterogeneous multiprocessors.

We first introduce multiprocessor system-on-chip platforms as examples of heterogeneous architectures. We then discuss accelerators for computer vision applications. We close with a brief discussion of interconnect for single-chip multiprocessors.

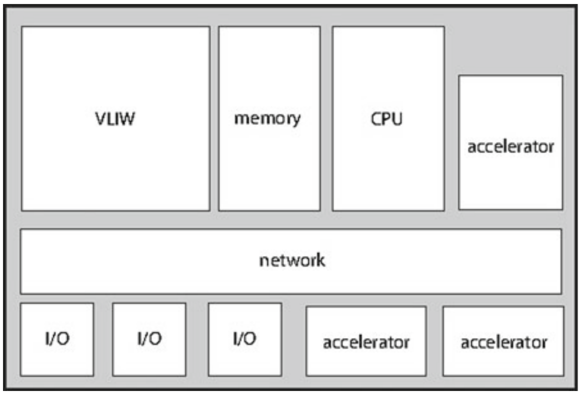


Fig. 1.1 Organization of a typical heterogeneous MPSoC

1.4.1 Multiprocessor System-on-Chips

A multiprocessor system-on-chip (MPSoC), as shown in Fig. 1.1, is a single-chip system including multiple programmable processors, memory, and I/O. Wolf et al. [64] discuss the history of MPSoCs in general; single-chip multiprocessors gained commercial acceptance in the embedded space well before their appearance in general-purpose systems. Given the aggressive performance and energy requirements of multimedia applications, embedded multimedia has been a driving force in the development of MPSoCs. While some MPSoCs are homogeneous configurations of the same processor type connected by an interconnect network, many MPSoCs are heterogeneous and contain several different types of processors. MPSoCs often make use of specialized accelerators that perform key, well-defined operations at high rates and low energy consumption. MPSoCs also provide a set of I/O devices tailored to an application space.

Many embedded vision systems make use of the MPSoCs developed for multimedia applications, either for plug-powered home entertainment systems or multimedia processors for cell phones and tablets. A multimedia MPSoC typically contains several elements:

- a RISC processor used for system control, operating system, and some miscellaneous multimedia computations;
- a VSP to execute multimedia kernels;
- accelerators for key operations such as DCT, motion acceleration, or display operations;
- on-chip memory;
- I/O devices for standard interfaces such as USB as well as video- and audio-centric I/O standards such as HDMI and SPDIF.

The TI TMS320DM814x Da Vinci processor [31] is an example of a modern multimedia MPSoC. It provides a media acceleration processor in parallel with its

VLIW and ARM cores. The HDVICP2 processor is designed primarily to support video, image, and audio compression and decompression. Some of its modules may be useful for computer vision operations: the iME3 accelerator performs motion estimation, iPE3 performs intraprediction estimation, CALC3 performs forward and inverse transforms, and the iLF3 performs deblocking filtering.

The Mobileye EyeQ [51] is an MPSoC designed to support automotive computer vision applications. Its major components include:

- two ARM CPUs, one for utility functions and the other for vision algorithms that are not well suited to hardware acceleration;
- an engine for image scaling, preprocessing, and image pattern classification;
- a tracker engine;
- a lane detection engine that identifies lane marks and road geometry;
- an engine for convolver, image pyramid comptuations, edge detection, and image filters;
- a 4×4 interconnection network.

1.4.2 Accelerators

Multimedia processors typically rely on accelerators for a significant fraction of their computation budget. The set of accelerators for multimedia has been well-explored. The design space for computer vision accelerators has not been mapped out to the same extent. One significant difference between the two is that multimedia accelerators usually require only fixed-point computation of limited precision; in contrast, many computer vision algorithms require floating-point arithmetic to provide sufficient accuracy.

Motion estimation is the most computationally expensive step in MPEG-style video compression and therefore has received a great deal of attention. Block matching motion estimation compares a current block (a) to a macroblock in a reference frame (b) in order to find the offset for the current macroblock relative to the reference frame. A common objective is to minimize the sum-of-absolute-difference (SAD) error between the two frames, where the SAD can be written as

$$\sum_{1 \leq i \leq N} \sum_{1 \leq j \leq N} \| a(i, j) - b(i + m_x, j + m_y) \| \tag{1.1}$$

A search over possible motion vectors m produces the motion vector that provides the best match between the reference and current macroblocks. The problem was originally formulated as a full search over the reference area but modern algorithms perform heuristic searches that cover only a subset of the reference area. Even with less-than-full search, block motion estimation requires a great deal of memory bandwidth. Yang et al. [68] developed an accelerator for motion estimation. Their architecture took advantage of the access patterns created by the correlation of the current block to successive areas of the reference block; it shifted one set of pixels from function unit to function unit while holding the other set in place. Schlessman et al. [47] extended a motion estimation architecture for shape recognition; this architecture used the correlation function performed for motion estimation to compare an image region to target shapes.

Disparity analysis of two images is a basic process in stereo vision. Barnard and Thompson [4] used an iterative algorithm to compute the likelihood of matches between features in images. Zitnick and Kanade [69] developed an algorithm for generating disparity maps including occlusion detection. Morat et al. [41] developed a method to evaluate the accuracy of stereo vision systems for cars. Their method is based on the accuracy with which the 3-D scene can be reconstructed from stereo data. Gudis et al. [23] developed an FPGA-based stereo vision system. Their system iteratively builds and processes six levels of Gaussian pyramids. At each level, the previous level's disparity estimate is used as an initial value for search. A shiftable window unit computes sum-of-absolute-differences for candidate disparities to produce a disparity map; the SAD unit is fed by a three-port memory with each word corresponding to eight pixels. A disparity optimizer refines the disparity results, followed by estimation of invalid disparity values produced by the previous stages. Their accelerator used a six-port external memory.

The discrete cosine transform (DCT) has also received a great deal of attention. 2-D DCT is often implemented as a pair of 1-D DCT operations and a transposition, although direct 2-D DCT is becoming more common. The forward 1-D 8-point DCT transform of $f(i)$ is given by

$$F(u) = \frac{1}{2} C(u) \sum_{1 \le i \le N} f(i) \cos\left[\frac{\pi u(2i+1)}{16}\right] \tag{1.2}$$

where $C(i) = \frac{1}{\sqrt{2}}$ for $i = 0$ and 0 otherwise. The 1-D DCT processor by Ruetz et al. [45] is one example of a hardware-oriented algorithm for DCT; it computes four-point inner products and some additional additions and subtractions. A variety of vision algorithms have been developed to exploit the DCT.

The Lucas-Kanade optical flow algorithm [38] is a good candidate for acceleration: it is widely used and combines regular memory access patterns with a significant numerical core. Given the partial derivatives (or differences) of a frame I as I_x, I_y, I_t; we talk about a pixel in the frame as $I(x, y)$. We compute the optical flow based on an $n \times n$ window in the frame. We want to compute the v_x, v_y motion vectors as:

$$\begin{aligned}\begin{bmatrix} v_x \\ v_y \end{bmatrix} = &\begin{bmatrix} \sum_{1\le i\le n}\sum_{1\le j\le n} I_x(i,j)^2 & \sum_{1\le i\le n}\sum_{1\le j\le n} I_x(i,j)I_y(i,j) \\ \sum_{1\le i\le n}\sum_{1\le j\le n} I_x(i,j)I_y(i,j) & \sum_{1\le i\le n}\sum_{1\le j\le n} I_y(i,j)^2 \end{bmatrix}^{-1} \\ &\times \begin{bmatrix} \sum_{1\le i\le n}\sum_{1\le j\le n} I_x(i,j)I_t(i,j) \\ \sum_{1\le i\le n}\sum_{1\le j\le n} I_y(i,j)I_t(i,j) \end{bmatrix}\end{aligned} \tag{1.3}$$

Schlessman et al. [46] designed an optical flow accelerator that made use of custom-designed limited-precision floating-point operators. The first step in the optical flow computation is a convolution of a 2×2 mask with the image; since the values of the mask are ± 1 this operation is straightforward. The next step is to compute $A^T A$ and $A^T G_t$ where $A = [G_x G_y]$. Careful organization of this computation limits the complexity of this step. The most computaionally expensive step is a matrix inversion required on $A^T A$. The matrix to be inverted is of size 2×2, which allows the form of the inversion to be simplified. However, the reciprocal of the matrix determinant must stilll be computed; the reciprocal is the most expensive operation required for the algorithm. The unit uses the Symmetric Bipartite Table Method [48], which performs a second-order Taylor approximation. A table lookup is at the core of this algorithm; the size of the table was reduced to $2^{10} \times 15 + 2^{10} \times 5$ bits.

The mixture-of-Gaussians approach algorithm is widely used for adaptive background elimination. An algorithm by Horprasert et al. [27] is one example. Several Gaussian models are kept, each with a different estimate of the state of each pixel. The algorithm first compares Gaussians of each model to find matches; it then updates Gaussian mean and variance and updates the weights. A pixel α is characterized as $\sqrt{\frac{X-\mu_x}{\sigma_x}}$. The Y, Cr, and Cb color components of each pixel are weighted relative to their average value a, combined and compared to a threshold to determine whether the pixel should be classified as foreground or background. The algorithm keeps several sets of pixel statistics (typically four sets) to account for varying image statistics over space and time; a weighting function is used to rank the usefulness of the Gaussians. Schlessman et al. [46] also described a hardware architecture for Horprasert et al.'s algorithm. It performed three major functions: comparison of match Gaussians to pixel intensity; updating of means and variances; and updating of Gaussian weights.

Image pyramids are widely used in vision algorithms. The Laplacian pyramid was introduced by Burt and Adelson [6] as a method for image compression. They use a Gaussian function such as

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}} \tag{1.4}$$

to smooth the image, then subtract the filtered image g_l from the original g_{l-1} giving a prediction error $L_l(i, j) = g_{l-1}(i, j) - g_l(i, j)$. They then apply the operation on g_l to produce g_{l+1}, repeating the operation several times to create the image pyramid.

Clemons et al. [7] developed the EFFEX accelerator that can support several feature extraction algorithms, including SIFT, FAST, and HoG. Consider SIFT [37] as an example. The algorithm is designed to provide results that are invariant on a variety of image transforms, including scaling, rotation, and translation. An image pyramid is built with a series of Gaussian smoothed images; smoothing is performed with a series of !-D Gaussian functions in x and y. Features are detected at ech level of the pyramid by computing image gradient and magnitude from the pixels A_{ij}:

$$M_{ij} = \sqrt{(A_{ij} - A_{i+1,j})^2 + (A_{ij} - A_{i,j+1})^2} \tag{1.5}$$

$$R_{ij} = atan2(A_{ij} - A_{i+1,j}, A_{i,j+1} - A_{ij}) \tag{1.6}$$

The resulting key is described using blurred samples in the region around the key; the samples are performed at two levels of the image pyramid. The EFFEX accelerator architecture includes several function units. A one-to-many comparison unit compares an operand to 16 other values, returning the number of values that are greater than/less than the operand. A convolution MAC performs an inner product on two floating-point vectors. A gradient unit uses the Sobel convolution kernel to compute the gradient of an image patch. Their memory is organized into 2-D patches; software that feeds the accelerator stores the image pixels in patch order; a single patch can be read by the accelerator by reading only one DRAM row. They compared their architecture to CPU and GPU implementations and showed that it provided improved frame rate per unit area.

1.4.3 Interconnect

Packet-based or flit-based networks-on-chips (NoCs) have been widely studied for single-chip multiprocessors. Many multicore processors use bus-based communication fabrics but other network architectures are also used in system-on-chip designs. Wolf surveys networks-on-chips [61].

Transaction-oriented NoCs are used in large systems-on-chip, notably cell phone processors. These networks augment the core, packet-based network with network adapters that deal with their host processor at a transaction level consisting of a block of data, a destination address, and a set of QoS requirements. The network adapters schedule packets on the network based on both QoS requirements and the current state of the network. Weber et al. [60] describe a QoS-oriented service model for NoCs. They divide time into epochs as a time granularity for arbitration. A global epoch is composed as a set of local epochs, each of which corresponds to a set of thread requests to a target or set of targets. Bandwidth allocations are set by the application. The network adapter uses a credit mechanism to keep track of a thread's use of bandwidth. Once a thread's credit counter goes negative, the thread is demoted. van der Wolf and Henriksson [63] identified the interface to bulk DRAM as a critical bottleneck for video processing systems. van der Wolf and Geuzebroek [62] describe QoS mechanisms for SoCs. They identified a set of traffic parameters for QoS-oriented traffic:

- bandwidth;
- the number of transactions in a burst;
- burstiness gap, or the time between bursts;
- transaction size;
- the ratio between reads and total transactions;
- the addressing relation between consecutive transactions;
- the maximum number of outstanding transactions;
- QoS class;
- average or maximum latency requested.

Xu et al. [67] studied network-on-chip designs using as a benchmark data from a smart camera gesture recognition system [65]. Their data showed frame-oriented periodicities as well as variations in traffic from frame to frame. In another study [66], they studied traffic from an H.264 decoder as well as the gesture recognition system. They showed that different parts of the vision pipeline generated different amounts of data and that an asymmetric network provided significant power savings over a symmetric network that provided equal bandwidth to all processing elements.

1.5 Distributed Camera Networks

Single cameras are often insufficient to cover a given scene, requiring the use of camera networks. A basic decision is whether to stream video to a remote node for processing or to perform at least some processing in-camera. Standard streaming systems can be used to deliver video streams to remote nodes. IP camera refers to a camera that sends a video stream over a network using the Internet Protocol. A variety of IP cameras are available with various combinations of wired and wireless network connections. Cloud-based analysis offers simplified algorithms at the expense of increased network traffic.

In this section, we consider the architecture of distributed camera networks. We first consider the software platform used to support distributed computing from the smart camera nodes. We then consider two types of algorithms that have been developed for distributed camera networks: calibration and tracking. Both these problems illustrate techniques used to generate compact descriptions that can be shared between cameras without transmitting full video.

1.5.1 Distributed System Architectures

Doblander et al. [11] designed a software architecture for distributed smart camera systems. Their architecture includes a DSP framework and a SmartCam framework. The SmartCam framework is designed to run on the network processor, on top of a standard operating system such as Linux. Communication is organized around a publish-subscribe model that abstracts messages from the underlying communication mechanisms. Communication is performed using mailboxes; large video data objects are not copied but rather identified by reference. The DSP framework abstracts the hardware and communication, supports dynamic loading/unloading of tasks; and management of on-chip and off-chip resources. The DSP algorithm component model is an extension of the Texas Instrumetns XDAIS model. The framework monitors several resources in order to satisfy QoS requirements: CPU utilization, DMA, memory limits, PCI bus load, memory usage, execution and communication times. The system provides fault tolerance using dynamic reconfiguration and QoS adaptation.

Lin et al. [36] describe the software architecture of a distributed gesture recognition system. They used model-based design techniques to create a service-oriented architecture. Services consisted of vision operations such as contour following and ellipse fitting. A middleware layer provided generic operations used to mediate between the application and particular services: data dispatch, service registration, dynamic binding, etc. It was also responsible for service scheduling and routing messages between services. Three types of messges are provided: service discovery; incoming service binding; and service access.

MPI [21] is widely used for parallel and grid computing. Communication between processes is organized around communicators. Communicators support both point-to-point communication between pairs of processes and collective communication between sets of processes. MPI is not widely used to build distributed embedded systems, but some efforts to develop MPI variants suitable for real-time embedded computing have been made. eMPICH [40] is designed for memory-constrained systems; they compared two approaches to building such a library and compared the object code size required for each. Kanevsky et al. [32] describe MPI/RT, whichprovides MPI-style communication with quality-of-service (QoS) requirements. It supports a time-driven communication model that specifices communication requirements over a time interval. An event-driven model allows QoS parameters to be used to control the start or stop of an application or data transfer. A priority-driven model specifies priorities for communication channels. Agbaria et al. [1] developed LMPI as a light-weight implementation of MPI. Their design distinguishes server nodes which run an MPI package and process nodes which provides basic communication operations. Processes on process nodes communicate with LMPI servers on server nodes, relying on the server nodes to perform MPI functionality. Ly et al. [39] developed an MPI implementation for FPGAs. They note that such systems must deal efficiently with three types of interaction: software-to-software; software-to-hardware accelerator; and accelerator-to-accelerator. They developed a DMA engine that uses burst transactions on memory to improve the performance of MPI transactions. They designed a non-blocking, non-interrupting request operation to allow software-hardware interactions to take maximum advantage of allowable parallelism. They also introduced an `MPI_Coalesce()` function to coalesce several MPI messages into a single physical operation whose performance can be optimized by the underlying hardware.

Hong et al. [26] proposed the target container programming model for the design of tracking systems. Each tracking target is associated with a target container, each with its own context. Some tracking algorithms require association between several targets, which is provided by allowing target containers to contain multiple trackers. They use an equlality checker to determine if two currently active target containers are, in fact, tracking the same target; in that case, the duplicative target containers are merged.

1.5.2 Calibration

Camera systems need to be calibrated in both space (determining what part of the scene is viewed by each camera) and time (synchronizing the video streams of the cameras). Hartley and Zisserman analyze the geometry of multi-camera systems in detail [24]. Devarajan et al. [10] developed a distributed algorithm metric camera calibration. They model the camera network using two graphs. The communication graph is based on network connectivity—it has an edge between nodes that directly communicate; this graph can be constructed using standard ad-hoc network techniques. The vision graph is based on signal characteristics—it has an edge between two nodes that have overlapping fields-of-view; this graph needs to be constructed during the calibration process. Each camera is described by one matrix that gives the rotation and optical center of the camera and another matrix that describes the camera's intrinsic parameters (focal length, etc.). An initial calibration estimate is constructed from a set of common scene points with outliers rejected using RANSAC. Belief propagation is used to improve the initial result.

Stone and Sekercioglu [52] developed an algorithm for estimating camera overlap; their algorithm requires relatively low communication bandwidth between the cameras. They use SURF to identifiy features in low-quality webcams. They then match features using a nearest-neighbor algorithm, then calculate the homography between the points using RANSAC. They compose the feature vectors into a matrix, then use singular value decomposition to find the principal components of the matrix. The number of components transmitted can be adjusted based on available bandwidth.

Temporal calibration ensures that correct sets of images are compared across cameras. Consumer-grade cameras can exhibit significant variations in frame rate; our laboratory measured several consumer video cameras and found variations of 20 % in frame rate between them. Even professional cameras can exhibit some variation in frame rates that can cause significant temporal registration errors when the cameras are run for long periods, as is required in many vision applications. Early multi-camera systems distributed a synchronization pulse to all the cameras; this technique requires careful engineering and doesn't work for cameras that are separated by a significant distance. Velipasalar and Wolf [56] developed an image-based algorithm for temporal calibration. They first calibrated the cameras spatially, then tracked a target that could be seen by the cameras being registered, generating a video sequence for each. A simple tracker selects an anchor point for the target in each frame based on the bounding box. Correlation is used to compare the positions of the anchor points to find the offset between the video sequences that minimized the tracking discrepancy.

1.5.3 Tracking

A key problem in tracking with distributed camera networks is to minimize the amount of data transmitted. This generally requires developing an appearance model as well as a position indicator; a combination of both position and appearance can be used to identify targets. Several distributed tracking algortihms have been developed that illustrate several compact representations for tracking data.

Oh et al. [43] developed an approach called Markov chain Monte Carlo multi-target tracking (MCMC-MTT). Each smart camera in the system generates observations y_i of a set of targets; the identity and appearance of the targets is not known in advance. The tracking algorithm partitions these observations into tracks $\tau = \{y_1, \ldots, y_t\}$ such that the assignment of observations to tracks is as consistent as possible. When a new observation is made available, the observation must be added to a track or a new track must be created; the information added by the observation may cause a previous observation to be moved to a different track. An update to the set of observations transforms the set of all tracks ω into the new set of tracks ω'. The optimization problem's goal is to maximize the posterior of ω'. Oh et al. formulated the search for the maximum-posterior track assignment as a Monte Carlo problem and proposed several types of moves that generate or destroy tracks, add observations to tracks, or move observations. Kim and Wolf [33] developed a distributed algorithm for MCMC-MTT. Their algorithm first estimates local paths using observations at each camera that partially overlap with the observations of nearby cameras; it then concatenates local paths. Concatenation does not rely on the appearance of observations but rathern is based on ownership and path characteristics.

Velipasalar and Wolf [55] developed a multi-camera tracking system for a network of uncalibrated cameras. They used field-of-view lines to determine the intersection of regions visible to multiple cameras. Each camera in the system tracks each target; they then compute the corresponding locations of targets in other cameras' coordinates to determine when several cameras can simultaneously see the same target. Their distributed tracking system [54] used MPI as its communication mechanism. The cameras communicated using non-blocking communication, allowing each camera to post its tracking results and then continue to analyze new frames. They used point-to-point communication between cameras with overlapping fields of view; no central message server was required. Each communication consisted of 256 bytes of data.

Kushwaha and Koutsoukos [34] developed a network algorithm for 3D tracking. They used the Bhattacharyya coefficient to model the similarity between a reference target model and an observation and a particle filter for tracking. The particle densities were not directly transmitted but were instead modeled using a Gaussian mixture model. Intermediate nodes in the network were used to further reduce network traffic; these agregation nodes first multiplied Gaussian models from severa cameras, then reduced the number of components in the result for further transmission.

Arth et al. [2] developed an algorithm for object reidentification in distributed smart camera networks. This algorithm is used for consistent identification of an object across multiple uncalibrated cameras. They identified keypoints using differences between normalized, Gaussian blurred images, then used principal component analysis to represent the set of features. To be able to compare two images, they organized the features into vocabulary trees, with the path thorugh the tree to the closest feature forming the signature.

1.6 Conclusions

Smart camera system designers have a wide range of computational fabrics on which to base embedded vision systems. Proper matching of the algorithm to the computing platform allows the designer to extract maximum accuracy from the system while meeting other goals such as power consumption. The space of accelerators for computer vision algorithms, however, is not as well understood as that for multimedia accelerators. Platforms for distributed computer vision are not as well advanced. Given the broad range of interest in Internet-of-things and networked cyber-physical devices, we can hold out some hope for the development of distributed computing platforms that can be leveraged for smart camera networks.

Acknowledgments Thanks to Tom Conte of Georgia Tech for discussions on computer architecture.

References

1. Agbaria A, Kang DI, Singh K (2006) LMPI: MPI for heterogeneous embedded distributed systems. In: 12th International conference on parallel and distributed systems 2006 (ICPADS 2006), vol 1, p 8. doi:10.1109/ICPADS.2006.56
2. Arth C, Leistner C, Bischof H (2007) Object reacquisition and tracking in large-scale smart camera networks. In: First ACM/IEEE international conference on distributed smart cameras 2007 (ICDSC ’07), pp 156–163. doi:10.1109/ICDSC.2007.4357519
3. Austin T, Blaauw D, Mahlke S, Mudge T, Chakrabarti C, Wolf W (2004) Mobile supercomputers. IEEE Comput 37(5):81–83
4. Barnard ST, Thompson WB (1980) Disparity analysis of images. IEEE Trans Pattern Anal Mach Intell PAMI 2(4):333–340
5. Boddie J, Daryanani G, Eldumiati I, Gadenz R, Thompson J, Walters S, Pedersen R (1980) A digital signal processor for telecommunications applications. In: Solid-state circuits conference. Digest of technical papers. 1980 IEEE international, vol XXIII, pp 44–45. doi:10.1109/ISSCC.1980.1156117
6. Burt P, Adelson E (1983) The laplacian pyramid as a compact image code. IEEE Trans Commun 31(4):532–540. doi:10.1109/TCOM.1983.1095851
7. Clemons J, Jones A, Perricone R, Savarese S, Austin T (2011) EFFEX: an embedded processor for computer vision based feature extraction. In: Design automation conference (DAC), 2011. 48th ACM/EDAC/IEEE, pp 1020–1025
8. Corporation A (2014) Stratix v device overview. Tech Rep SV51001
9. Corporation X (2013) 7 Series fpgas overview. Tech Rep DS180
10. Devarajan D, Cheng Z, Radke R (2008) Calibrating distributed camera networks. Proc IEEE 96(10):1625–1639. doi:10.1109/JPROC.2008.928759

11. Doblander A, Zoufal A, Rinner B (2009) A novel software framework for embedded multiprocessor smart cameras. ACM Trans Embed Comput Syst 8(3):24:1–24:30. doi:10.1145/1509288.1509296. http://doi.acm.org.prx.library.gatech.edu/10.1145/1509288.1509296
12. Dutta S, O'Connor K, Wolf W, Wolfe A (1998) A design study of a 0.25-µm video signal processor. IEEE Trans Circuits Syst Video Technol 8(4):501–519 doi:10.1109/76.709414.
13. Dutta S, Wolf W (1999) A circuit-driven design methodology for video signal-processing datapath elements. IEEE Trans VLSI Syst 7(2):229–240
14. Dutta S, Wolf W, Wolfe A (1998) A methodology to evaluate memory architecture design tradeoffs for video signal processors. IEEE Trans Circuits Syst Video Technol 8(1):36–53
15. Ercegovac MD, Lang T (2004) Digital arithmetic. Morgan-Kaufmann
16. Farabet C, Martini B, Corda B, Akselrod P, Culurciello E, LeCun Y (2011) Neuflow: a runtime reconfigurable dataflow processor for vision. In: IEEE computer society conference on computer vision and pattern recognition workshops (CVPRW 2011), pp 109–116. doi:10.1109/CVPRW.2011.5981829
17. Fritts J, Wolf W (1999) Understanding multimedia application characteristics for designing programmable media processors. In: Multimedia hardware architectures. SPIE
18. Fritts J, Wolf W (2000) Evaluation of static and dynamic scheduling for media processors. In: Proceedings, MICRO-33 MP-DSP2 workshop. ACM
19. Fung J, Mann S (2005) OpenVIDIA: parallel GPU computer vision. In: Multimedia '05: proceedings of the 13th annual ACM international conference on multimedia
20. Fung J, Mann S (2008) Using graphics devices in reverse: GPU-based image processing and computer vision. In: IEEE international conference on multimedia and Expo 2008, pp 9–12. doi:10.1109/ICME.2008.4607358
21. Gropp W, Lusk EL, Skjellum A (1999) Using MPI: portable parallel programming with the message-passing interface, 2nd edn. MIT Press, Cambridge
22. Gudis E, Lu P, Berends D, Kaighn K, van der Wal G, Buchanan G, Chai S, Piacentino M (2013) An embedded vision services framework for heterogeneous accelerators. In: IEEE conference on computer vision and pattern recognition workshops (CVPRW 2013), pp 598–603. doi:10.1109/CVPRW.2013.90
23. Gudis E, Van der Wal G, Kuthirummal S, Chai S (2012) Multi-resolution real-time dense stereo vision processing in FPGA. In: IEEE 20th annual international symposium on field-programmable custom computing machines (FCCM 2012), pp 29–32. doi:10.1109/FCCM.2012.15
24. Hartley RI, Zisserman A (2004) Multiple view geometry in computer vision, 2nd edn. Cambridge University Press, Cambridge
25. Hennessy JL, Patterson DA (2001) Computer architecture: a quantitative approach, 5th edn. Morgnn Kaufman, Los Altos
26. Hong K, Smaldone S, Shin J, Lillethun D, Iftode L, Ramachandran U (2011) Target container: a target-centric parallel programming abstraction for video-based surveillance. In: Fifth ACM/IEEE international conference on distributed smart cameras (ICDSC 2011), pp 1–8. doi:10.1109/ICDSC.2011.6042914
27. Horprasesert T, Harwood D, Davis LS (1999) A statistical approach for real-time robust background subtraction and shadow detection. In: IEEE international conference on computer vision FRAME-RATE workshop
28. Instruments T (2001) TMS320C55x DSP reference guide. Tech Rep SPRU371D
29. Instruments T (2010) TMS320C674x DSP CPU and instruction set reference guide. Tech Rep SPRUFE8B
30. Instruments T (2013) Floating-point digital signal processor. Tech Rep SPRS675
31. Instruments T (2013) Tms320dm814x davinci digital video processors technical reference manual. Tech Rep SPRUGZ8E. Revision 5
32. Kanevsky A, Skjellum A, Rounbehler A (1998) MPI/RT-an emerging standard for high-performance real-time systems. In: Proceedings of the thirty-first Hawaii international conference on system sciences, vol 3, pp 157–166. doi:10.1109/HICSS.1998.656130

33. Kim H, Wolf M (2010) Distributed tracking in a large-scale network of smart cameras. In: Proceedings of the fourth ACM/IEEE international conference on distributed smart cameras. ACM Press, p 816
34. Kushwaha M, Koutsoukos X (2010) 3D target tracking in distributed smart camera networks with in-network aggregation. In: Proceedings of the fourth ACM/IEEE international conference on distributed smart cameras (ICDSC '10). ACM, New York, pp 25–32. doi:10.1145/1865987.1865992. http://doi.acm.org.prx.library.gatech.edu/10.1145/1865987.1865992
35. Li E, Wang B, Yang L, ti Peng Y, Du Y, Zhang Y, Chiu YJ (2012) GPU and CPU cooperative accelaration for face detection on modern processors. In: IEEE international conference on multimedia and expo (ICME 2012), pp 769–775. doi:10.1109/ICME.2012.121
36. Lin CH, Wolf M, Koutsoukos X, Neema S, Sztipanovits J (2010) System and software architectures of distributed smart cameras. ACM Trans Embed Comput Syst 9(4):38:1–38:30. doi:10.1145/1721695.1721704. http://doi.acm.org.prx.library.gatech.edu/10.1145/1721695.1721704
37. Lowe D (1999) Object recognition from local scale-invariant features. In: The proceedings of the seventh IEEE international conference on computer vision, vol 2, pp 1150–1157. doi:10.1109/ICCV.1999.790410
38. Lucas B, Kanade T (1981) An iterative image registration technique with an application to stereo vision. In: International joint conference on artificial intelligence. AAAI
39. Ly D, Saldana M, Chow P (2009) The challenges of using an embedded MPI for hardware-based processing nodes. In: International conference on field-programmable technology 2009 (FPT 2009), pp 120–127. doi:10.1109/FPT.2009.5377688
40. McMahon T, Skjellum A (1996) eMPI/eMPICH: embedding MPI. In: Proceedings of MPI developer's conference, 1996, Second, pp 180–184. doi:10.1109/MPIDC.1996.534111
41. Morat J, Devernay F, Ibanez-Guzman J, Cornou S (2007) Evaluation method for automotive stereo-vision systems. In: Intelligent vehicles symposium, 2007 IEEE, pp 202–208. doi:10.1109/IVS.2007.4290115
42. Nagendra P (2011) Performance characterization of automotive computer vision systems using graphics processing units (GUPs). In: International conference on image information processing (ICIIP 2011), pp 1–4. doi:10.1109/ICIIP.2011.6108951
43. Oh S, Russell S, Sastry S (2004) Markov chain monte carlo data association for general multiple-target tracking problems. In: Proceedongs of the 43rd IEEE conference on decision and control
44. Peleg A, Weiser U (1996) Mmx technology extension to the intel architecture. Micro IEEE 16(4):42–50. doi:10.1109/40.526924
45. Ruetz P, Tong P, Bailey D, Luthi D, Ang PH (1992) A high-performance full-motion video compression chip set. IEEE Trans Circuits Syst Video Technol 2(2):111–122. doi:10.1109/76.143411
46. Schlessman J, Lodato M, Ozer B, Wolf W (2007) Heterogeneous MPSoC architectures for embedded computer vision. In: 2007 IEEE international conference on multimedia and expo, pp 1870–1873. IEEE
47. Schlessman J, Saha S, Wolf W, Bhattacharya S (2005) An extended motion-estimation architecture applied to shape recognition. In: IEEE international conference on multimedia and expo 2005 (ICME 2005), pp 1504–1507. doi:10.1109/ICME.2005.1521718
48. Schulte M, Stine J (1999) Approximating elementary functions with symmetric bipartite tables. IEEE Trans Comput 48(8):842–847. doi:10.1109/12.795125
49. Semiconductor F (2012) Mpc5676r microcontroller data sheet. Tech Rep MPC5676R. Rev 3
50. Soderquist P, Leeser M (1997) Division and square root: choosing the right implementation. Micro IEEE 17(4):56–66. doi:10.1109/40.612224
51. Stein G, Rushinek E, Hayun G, Shashua A (2005) A computer vision system on a chip: a case study from the automotive domain. In: IEEE computer society conference on computer vision and pattern recognition—workshops, 2005. CVPR workshops, pp 130–130. doi:10.1109/CVPR.2005.387
52. Stone W, Sekercioglu Y (2010) Analysis of distributed smart camera calibration accuracy using minimal data transmission. In: IEEE international conference on communication systems (ICCS), 2010, pp 655–661. doi:10.1109/ICCS.2010.5686597

53. Tallu D, John LK, Burger D (2003) Bottlenecks in multimedia pro-cessing with SIMD style extensions and multimedia enhancements. IEEE Trans Comput 52(8):1015–1031
54. Velipasalar S, Schlessman J, Chen CY, Wolf W, Singh J (2006) SCCS: a scalable clustered camera system for multiple object tracking communicating via message passing interface. In: IEEE international conference on multimedia and expo, 2006, pp 277–280. doi:10.1109/ICME.2006.262452
55. Velipasalar S, Wolf W (2005) Multiple object tracking and occlusion handling by information exchange between uncalibrated cameras. In: IEEE international conference on image processing 2005 (ICIP 2005), vol 2, pp II-418-21. doi:10.1109/ICIP.2005.1530081
56. Velipasalar S, Wolf WH (2008) Frame-level temporal calibration of video sequences from unsynchronized cameras. Mach Vis Appl J. doi:10.1007/s00138-008-0122-6
57. Viola P, Jones M (2001) Rapid object detection using a boosted cascade of simple features. In: Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition 2001 (CVPR 2001), vol 1, pp I-511–I-518. doi:10.1109/CVPR.2001.990517
58. Wang G, Xiong Y, Yun J, Cavallaro J (2013) Accelerating computer vision algorithms using opencl framework on the mobile GPU—a case study. In: IEEE international conference on acoustics, speech and signal processing (ICASSP 2013), pp 2629–2633. doi:10.1109/ICASSP.2013.6638132
59. Wang X, Leeser M (2010) Vfloat: a variable precision fixed- and floating-point library for reconfigurable hardware. ACM Trans Reconfigurable Technol Syst 3(3):16:1–16:34. doi:10.1145/1839480.1839486. http://doi.acm.org/10.1145/1839480.1839486
60. Weber WD, Chou J, Swarbrick I, Wingard D (2005) A quality-of-service mechanism for interconnection networks in system-on-chips. In: Proceedings of design, automation and test in Europe, 2005, vol 2, pp 1232–1237. doi:10.1109/DATE.2005.33
61. Wolf M (2014) High performance embedded computing, 2nd edn. Morgan Kaufmann, Los Altos
62. van der Wolf P, Geuzebroek J (2011) SoC infrastructures for predictable system integration. In: Design, automation test in Europe conference exhibition (DATE 2011), pp 1–6. doi:10.1109/DATE.2011.5763146
63. van der Wolf P, Henriksson T (2008) Video processing requirements on SoC infrastructures. In: Design, automation and test in Europe 2008 (DATE '08), pp 1124–1125. doi:10.1109/DATE.2008.4484827
64. Wolf W, Jerraya A, Martin G (2008) Multiprocessor system-on-chip (MPSoC) technology. IEEE Trans Comput Aided Des Integr Circuits Syst 27(10):1701–1713. doi:10.1109/TCAD.2008.923415
65. Wolf W, Ozer B, Lv T (2002) Smart cameras as embedded systems. IEEE Comput 35(9):48–53
66. Xu J, Wolf W, Henkel J, Chakradhar S (2006) A design methodology for application-specific networks-on-chip. ACM Trans Embed Comput Syst 5(2):263–280. doi:10.1145/1151074.1151076. http://doi.acm.org.prx.library.gatech.edu/10.1145/1151074.1151076
67. Xu J, Wolf W, Henkel J, Chakradhar S, Lv T (2004) A case study in networks-on-chip design for embedded video. In: Proceedings of design, automation and test in Europe conference and exhibition, 2004, vol 2, pp 770–775. doi:10.1109/DATE.2004.1268973
68. Yang KM, Sun MT, Wu L (1989) A family of vlsi designs for the motion compensation block-matching algorithm. IEEE Trans Circuits Syst 36(10):1317–1325. doi:10.1109/31.44348
69. Zitnick C, Kanade T (2000) A cooperative algorithm for stereo matching and occlusion detection. IEEE Trans Pattern Anal Mach Intell 22(7):675–684. doi:10.1109/34.865184

Chapter 2
A Survey of Systems-on-Chip Solutions for Smart Cameras

Ali Ahmadinia and David Watson

Abstract With the advances in electronic manufacturing technologies, integration of disparate technologies including sensors, analog components, mixed signal units and digital processing cores into a single chip has become reality, which is an increasing trend in different application domains, especially for distributed smart camera products. In this chapter, a survey of existing System-on-Chip solutions for distributed smart cameras able to capture and intelligently process video in real-time and communicate with other cameras and sensors remotely is presented.

2.1 Introduction

Smart cameras consist of three main units: Sensing, Processing, and Communications [18] as illustrated by Fig. 2.1. The sensing unit is responsible for image capture and may also perform pre-processing before the main processing task(s) are performed. The processing unit carries the main computation and is therefore the brains of the smart camera. The processing unit comprises Processing Elements (PEs)—which include all forms of processors such as Digit Signal Processors (DSPs), hardware functions/accelerators, and microcontrollers—on- and off-chip memories, and the communications architecture(s) used for inter- and intra-processor communications and data movement. The last unit is the communications unit which is responsible for transmitting the processed/abstracted data to output devices, where the end user can use it. These three units together make up the architecture of System on Chip (SoC)

A. Ahmadinia (✉) · D. Watson
School of Engineering and Built Environment, Glasgow Caledonian University, Cowcaddens Road, Glasgow, UK
e-mail: ali.ahmadinia@gcu.ac.uk

D. Watson
e-mail: david.watson2@gcu.ac.uk

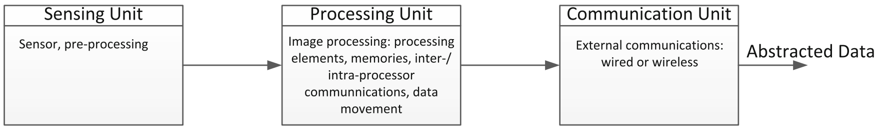

Sensing Unit
Sensor, pre-processing
Processing Unit
Image processing: processing elements, memories, inter-/ intra-processor communnications, data movement
Communication Unit
External communications: wired or wireless
Abstracted Data

Fig. 2.1 Architecture of smart camera SoC solutions [18]

smart cameras, and are therefore directly responsible for the abstracted representation of the input data following processing of input data [18].

Since smart camera SoC solutions provide a service to the end user, there are several Quality of Service (QoS) attributes associated with them. The QoS characteristics of smart camera SoCs encompass attributes such as frame-rate, transfer delay, image resolution, and video-compression rate [3]. However, as SoC-solutions for smart cameras are embedded devices, power consumption and resource consumption are also important attributes and should be minimised where possible. The frame-rate of a smart camera is dependent on the task it is performing. For example, a smart camera monitoring a car park may require real-time frame-rates of 30 frames per second (f/s)—in the interests of security; whereas a smart camera monitoring the flow of traffic on a busy stretch of road may not require real-time frame-rates: application discretion is required. Transfer delay can impact the frame-rate of the system and is dependent on the sensing unit used and data-movements of the system [14], which is directly impacted by the resolution of the input data from the imaging sensor and any compression methods used.

Each of theses attributes have a direct impact on the energy and resource consumptions of the smart camera SoC, and it is therefore paramount that the processing unit be optimised for the processing and movement of the input data. One last consideration for QoS in modern-day smart camera SoCs is security, where the collection of data that is sometimes private can introduce the need for the safe storage and transmission of data [20]. Smart camera SoC designers must also be aware of the need for encryption and secure network protocols, as the scale and complexity of smart cameras evolves. Table 2.1 summarises the QoS requirements of embedded smart camera SoC solutions.

Smart camera SoCs can be classified based on the decision classification network depicted in Fig. 2.2. Here we have decomposed the classifications of smart camera SoC solutions into the SoC used to perform the smart camera functions, such as processing, display, etc. and the smart camera solution as a whole and how it interacts with the end user. Note, we have abstracted the development/implementation platform into the SoC sub-categories, as this allows the classification of modern-day smart cameras more accurately. Smart cameras themselves can perform function(s) as a standalone agent, or as part of a larger network of smart cameras. The use of smart cameras in a distributed manner allows more data to be collected about a scene or environment and therefore allows more astute deductions to be made about image scenes [18]. The distribution of smart cameras can be wired or wireless,

Table 2.1 Table of QoS attributes for smart camera SoCs

Attribute	Description	Sample Operating Characteristics
Throughput	Time required to capture, process, and output and image	Real-time = 30 f/s
Power consumption	Power consumed by all units of the smart camera SoC: directly impacts running costs	Within the region of mW is nominal
Resource consumption	Computational and memory resources consumed by the SoC: directly impacts implementation costs	Ranges from bytes to kilobytes

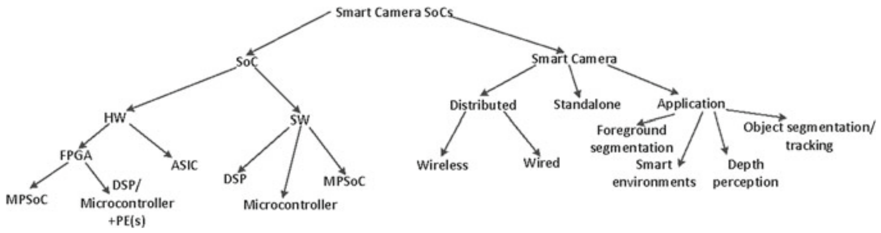


Fig. 2.2 Classification network for smart camera SoC solutions

however distributed smart cameras tend to be wired, as they have more computational power and are not as focussed on power/resource consumption [18].

Smart cameras are used in a variety of contexts, from surveillance and object tracking, to medical procedures and smart environments [18]. The use of smart cameras to create intelligent spaces—environments where the status of objects can be readily ascertained—has been investigated by Manbhai [22]. However the development environment is not embedded. With the increase in data available from smart cameras, the processing of input data has taken on more complex roles, including the rendering of stereo images for depth perception applications [19]. For a thorough and complete survey of smart camera SoC solutions related to embedded systems, we only include design and implementation work relevant to SoC design.

2.2 Sensing Unit

The sensing unit is responsible for image capture and can also be used for pre-processing images before the main processing is performed. CMOS imaging sensors are a popular choice for smart cameras as they allow for the access of pixels in a similar manner to that of random access memory [7]. CMOS sensors also allow for the fabrication of hardware functions close to the sensor [7], making them

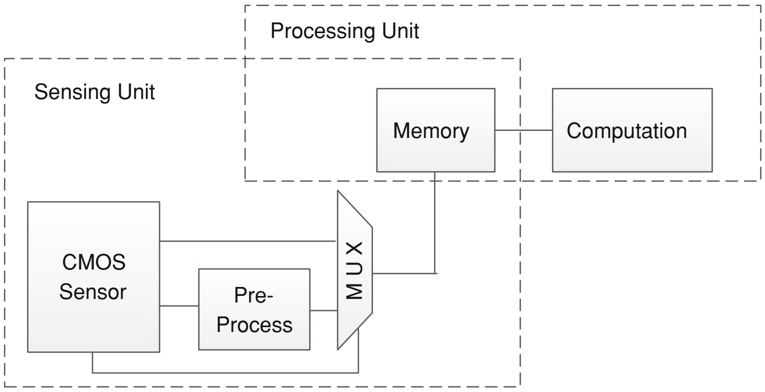


Fig. 2.3 Example architecture of a sensing unit, where the extraction of pixels can be performed in parallel

suitable for smart camera applications. Examples of this are that by Moorhead and Binnie [15], where the CMOS camera was capable of carrying out Canny edge detection on input images, and Heyrman et al. [9] whose CMOS camera was able to select Regions of Interest (ROI), which were then further processed by the processing unit. Designers can implement the optimal parallelisation of pixel extraction from CMOS imagers, allowing them to tailor the sensor the the smart camera's application. The 90 nm fabrication technology of 2005 required approximately 25 mm^2 of silicium for a parallel output of 64×64 pixel sub-window [9], but with modern 20 nm technology this could decrease. Lacassagne et al. [12] implement a programmable artificial retina—a network of pixels that brings processing to the data, as opposed to transferring data to PEs. Each pixel of the artificial network consists of an ADC and 48 bits of memory and can share data with its four surrounding pixels (neighbours). This allows data locality to be exploited with high-levels of parallelism implemented by dedicated hardware functions. Computation is performed using a boolean unit, which can perform bit-level computation.

A typical architecture of a smart camera sensor based on CMOS technology can be found in Fig. 2.3. Here we can see how the use of CMOS sensors allows designers to preprocess input images before they are passed on for further processing at the processing unit. The ability to extract pixels in parallel makes CMOS sensors an attractive option for smart cameras, but must also compliment the memory architecture of the SoC, and how data is moved into the processing unit. Figure 2.3 also shows how the sensing and processing units can overlap when preprocessing is used. In this example, the preprocessing block can access pixels from the sensor and process them, before writing them to memory for the processing unit to use.

2.3 Processing Unit

From Fig. 2.3, we can see the processing unit encompasses the computational agents of the SoC, the memory subsystem, and implicitly the communications system. We start with the arrangement of computational resources for smart camera SoCs, and then move onto the memory and communications subsystems.

2.3.1 Processing Unit: Processing Elements

The Processing Elements (PEs) of SoCs can be Digital signal Processors (DSPs), microcontrollers, dedicated hardware functions/accelerators, or general-purpose processors. Bramberger et al. [3] present a distributed embedded smart camera constructed from off-the-shelf components. TMS320C6416 DSPs by Texas Instruments are used as the processors of the SoC and are coupled together with memory via a PCI bus. The use of DSPs increases the flexibility of the SoC, as changes to the functionality are implemented in software. However, the use of DSPs for smart camera SoCs can limit the throughput of the system, as the processor's architecture is designed for general-purpose use: the memory subsystem can also present a bottleneck [5]. Microcontrollers are typically used to coordinate events of a SoC, but have been used to implement processing functions, object detection functions [10], where the 8-bit microcontroller is used to perform the feature extraction. However, the limited architecture of the microcontroller inhibited realtime QoS requirements and may be improved by upgrading to a 16- or 32-bit architectures. A Picoblaze soft-core microcontroller is used by Meng et al. [14] to synchronise and control all components of a Multiprocessor System on Chip (MPSoC). Each hardware core consists of a Picoblaze soft-core coupled with a coprocessor interface such, as that in Fig. 2.4a, to carry out dedicated functions in hardware. However, the flexibility of the smart camera is only in the ability to reprogram the Picoblaze, and not in the ability to reconfigure the co-processors.

Creating PEs from hardware functions as coprocessors is an attractive option for smart camera designers, as they are designed to carry out specific, dedicated functions within the SoC. PEs can be as simple as frame-grabbing [10], to applying image processing kernels and object detection functions [11]. Kruijtzer et al. [11] design smart imaging and motion estimation cores coupled to an ARM9 processor. The smart imaging core carries out low-level image processing algorithms, such as applying kernels, calculating histograms, etc., whereas the motion estimation core performs high-level motion estimation. This system uses the ARM9 to coordinate and control events, with the SoC implemented on a Field Programmable Gate Array (FPGA). However, this system requires large amounts of computational resources due to the generic nature of the PEs. Designers must take care to implement only time-critical or computationally intensive functions in hardware, in order to justify the resource utilisation.

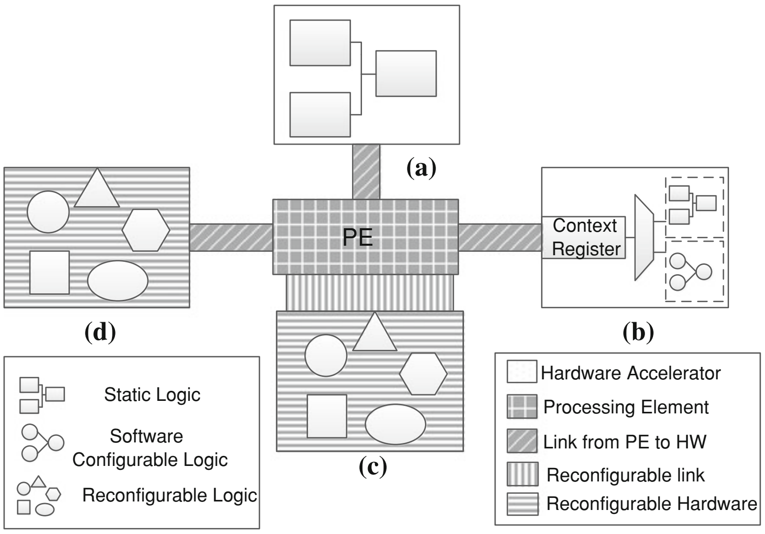


Fig. 2.4 Four ways in which hardware accelerators can be designed and interact with PEs of a SoC: Static arrangement (**a**), software programmable arrangement (**b**), reconfigurable arrangement with reconfigurable link (**c**), and reconfigurable arrangement with fixed link (**d**)

Chan et al. [4] apply dedicated programmable hardware accelerators with an ARM 926EJ-S for object detection/segmentation and face detection. The programmable morphology coprocessor is capable of performing dilation and erosion tasks based on the value in its context register (Fig. 2.4b), which is set by the ARM CPU. The programmable hardware increases the performance of the smart camera SoC, but may inefficiently utilise hardware resources depending on how they are used at runtime. This is example of reconfiguring a static hardware arrangement through software via the use of context registers. Computational resources would be utilised more efficiently if the physical makeup of the hardware accelerators was modified during runtime.

Chen et al. [5] implement a stream processor on an FPGA, where the data movements and PEs are optimised for high throughputs. The processing architecture is designed to be multipurpose, where PEs can be configured to carry out 1 of 6 popular image processing tasks, such as downsampling and applying 2D kernels. These PEs can be reconfigured together, to create larger and more powerful PEs which work together to perform more complex image processing tasks. Figure 2.4d illustrates this concept for one reconfigurable hardware accelerator. This reconfigurable architecture is an example of how smart camera SoCs can be implemented in reconfigurable hardware to achieve high throughputs and power efficiencies. However, the applicability of such an architecture is limited by the PEs used in the system, and how they can collaborate to create more complex processing operations. As smart cameras become more intelligent, the operations they are required to perform become more complex and may therefore increase the size and complexity of reconfigurable architectures.

Lacassagne et al. [12] evaluate the PowerPC (PPC), which is optimised for multimedia applications through its instruction set and vectorization. The authors found that the PPC processor is suited to low-level computer vision applications, but its throughput is limited by its instruction set and internal register architecture. Albani et al. [1] brought the processing of image data to generic, programmable embedded systems. A 32-bit RISC processor and vision/neural coprocessor for data processing—complimented by 512B instruction cache, 1MB RAM, and 512KB FLASH memory. This system improves design flexibility, as the processor can be reprogrammed to perform different tasks, as can the neural coprocessor within reason, but again the physical arrangement of the SoC resources is static.

Oetken et al. [17] also implement a reconfigurable SoC for a smart camera. The SoC is divided into static and dynamic regions, where an embedded CPU subsystem is placed in the static region, and the dynamic region can be used for custom hardware accelerators. To accommodate the variability of the hardware that may exist in the dynamic region, a reconfigurable bus (ReCoBus) is used, which allows connections between master and slave hardware accelerators, as well as the embedded CPU subsystem. Figure 2.4c illustrates this concept for one hardware accelerator. A recent example of dynamic reconfiguration for hardware accelerators in SoCs is [8], which makes use of one controlling PE that has dynamically reconfigurable hardware accelerators. However, such a large area for dynamic reconfiguration could introduce overheads for simple tasks that may have to be implemented in dynamic regions.

Table 2.2 summarises the PEs of smart camera SoCs, and their respective advantages and disadvantages. Based on this summary, designers of smart camera SoCs must make several key design decisions about the computational components of the smart camera SoC. Firstly, the required computation of the smart camera application: object detection/tracking, or basic background/foreground segmentation will dictate the types of PEs in the SoC. Secondly, how the smart camera will be deployed: will the smart camera perform a dedicated task? Will it be able to perform several tasks, or even be reprogrammable? Each of these considerations will have a direct impact on the memory and communications architecture of the SoC, which is discussed next. FPGAs can be used to create computationally efficient smart camera SoC-solutions, but can limit the design space of computer vision applications to specific functions implemented in hardware. Other examples of these techniques are [2, 4, 13, 16].

2.3.2 *Processing Unit: Memory and Communications*

The memory architecture of a smart camera SoC must compliment the processing requirements of the algorithm(s) executing on it [18]. As identified in the previous section, there are hardware and software solutions for designing smart camera SoCs: hardware functions and accelerators, and DSPs/microcontrollers respectively.

Table 2.2 Processing element summary

Processing Element(s)	Pros	Cons	Throughput	Power Consumption	Resource Consumption
TMS320C6416 DSP [3]	SW programmable SoC	Limited computational throughput	Low	Very high	Very high
	Application flexibility	Static memory arrangement			
Picoblaze micro-controller [14]	SW reprogrammable MPSoC; increased concurrency	May under utilise resources	Low	Low	Low
Dedicated hardware functions					
PPC [12]	SW reprogrammable	Static architecture	High	High	High
	Application flexibility	General purpose processing			
Reconfigurable hardware functions [5]	HW programmable MPSoC	Communications subsystem may limit scalability	High	Low	Low
	Increased resource utilisation	PEs may be limited by available resources			
	Application flexibility				
Microcontroller [10]	SW reprogrammable SoC	May under utilise resources	Medium	Medium	Low
Dedicated hardware functions		Limited ability of microcontroller			
ARM9 processor [11]	SW reprogrammable SoC	Static hardware functions	Unknown	Unknown	Unknown
Dedicated imaging and motion hardware functions		May under utilise resources			
ARM 926EJ-S [4]	SW programable MPSoC	May under utilise resources	High	Low	Low
Programmable hardware functions	Increased design flexibility and concurrency				
32-bit RISC processor [1]	SW reprogrammable hardware	May under utilise resources	Medium	Medium	Medium
Reprogrammable vision/neural co-processor	Increased design flexibility	Hardware must be redesigned for new vision tasks			

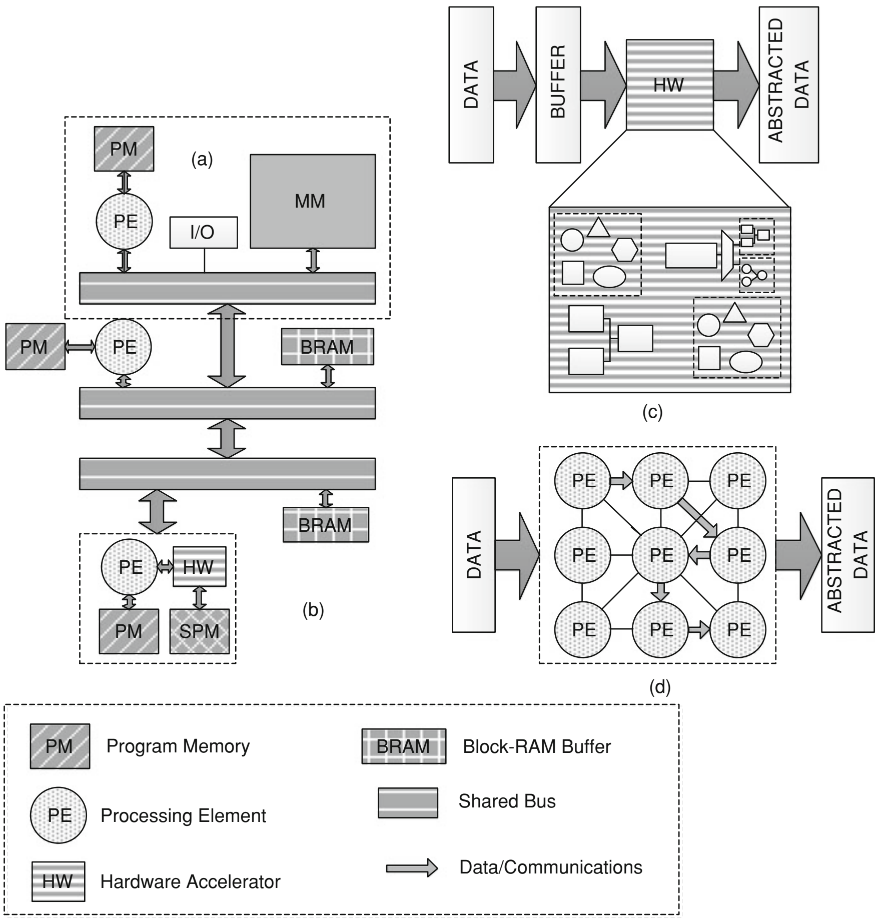


Fig. 2.5 Memory and communications architectures used for the works described in this chapter: shared bus (**a**), hierarchical (**b**), streaming (**c**), and associative mesh (**d**)

Software solutions increase the flexibility of the smart camera, but rely on caches to promote data reuse, which may not be effective [18]. On the other hand, hardware functions and accelerators can decrease the flexibility of the SoC, but are suited to memory customisations to compliment the data movements of the application. This section describes how the memory and communications architecture of smart camera SoCs can be tailored to compliment the PEs of a SoC, as well as the algorithms executing on them.

Meng et al. [14] use a hierarchical communications topology (Fig. 2.5b), where the top tier contains the system level components of the SoC, such as I/O and timers. The second tier is responsible for low-level, high bandwidth image capture and communication tasks. And the bottom tier performs high-level, low-bandwidth data processing tasks. Each tier is connected using a bridge, and localised communications within each tier is performed over a shared system bus. This topology is useful in defining the high-level blocks of the SoC, and allows designers to abstract themselves during software design: as the use of bridges creates a unified addressing space. The second tier contains a DMA engine that can transfer data to and from the localised external memories of tiers 2 and 3. This limits data accesses over the bridges and improves data locality and temporality. Such a hierarchy is useful when defining the computational arrangement of a SoC and allows designers to focus more on the design and selection of PEs.

Chan et al. [4] create a streaming architecture, similar to Fig. 2.5c, for their smart camera SoC. To efficiently utilise the system bus bandwidth, the hardware is carefully designed to avoid bit-width mismatches. The authors use sub-word level parallelism to address this, where input data is packed and aligned in memory such that the data of any eight aligned neighbouring pixels of an image are stored as a 64-bit word. This data arrangement allows the processing of eight 3×3 sub-windows at eight different PEs, creating a SIMD architecture of SW reconfigurable PEs. This is a low-level approach to extracting data-level parallelism for smart camera SoCs, and in contrast to the three-tier hierarchy of Meng et al. [14] is a fine-grained approach. This fine-grained approach allows a more efficient design of the PEs of the SoC, but can limit the flexibility of the smart camera application: since the PEs perform dedicated functions. However, the memory and communications architecture can be easily reused, and new computational functions could added in a plug-and-play fashion, provided the architecture is designed to accommodate the functions.

A similar stream-based processing architecture is used by Chen et al. [5]. Input data is moved via data streams over a master system bus. Line buffers are used to give one PE access to all pixels of 3×3 sub-window in one clock cycle where required, as opposed to the sub-word level parallelism of [4] that provided this parallelism to many PEs. However, the communications architecture of this work can be reconfigured to create more powerful PEs by changing/combining their interconnections through multiplexing. Reconfiguring the communications architecture increases the level of parallelism and could also be used to switch between different operating characteristics, such as power consumption and throughput. However, the reconfiguration of many PEs may require more complex interconnections and would therefore increase wire congestion of the multiplexers, which would have to be considered by the designer.

The bandwidth of the dynamic region implemented by [17] is limited by the number of interleaved signals the ReCoBus can implement (6 in this case); however hardware accelerators can directly access memory allowing high-speed transfers to and from the dynamic region. Communications between the static and dynamic regions are achieved through bridging and round robin arbitration. This technique does require some static switching hardware to ensure static and dynamic regions can communicate. However, it is a more flexible solution than [5], as the communications architecture supports variable hardware accelerators.

Lacassagne et al. [12] present an associative mesh (Fig. 2.5d) made-up from a grid of PEs that readily communicate with each of their eight neighbours. An application is characterised by an interconnection graph—an asynchronous path where data can circulate freely from processor to processor. Each PE contains an 8-bit graph register that emulates the presence or absence of an incoming value from one of its eight neighbours. The associative mesh is capable of reconfiguring its purpose and computation without the need to actively reconfigure the interconnections of PEs. This system relies on the profiling of an application to obtain the graph and also requires one PE for each pixel, which could lead to inefficient resource consumptions.

Xu et al. [25] investigate the communications architecture of MPSoCs, where the traditional bus-based model is replaced by a crossbar to give all processors access to input data and memories (Fig. 2.5a). Crossbars are a popular bus due to the ability to connect a large number of master to slaves. For an MPSoC it is important to consider congestion that may arise over shared buses, as PEs use them to access shared resources. Despite contributing to the processing of the same input data, this can be detrimental to performance. Arbitration is used as a means of policing shared accessed to bus and is often inherent in the bus controller. The authors found that arbiter controlled bus transactions performed better than processor controlled ones, and the crossbar provided the greatest average throughput, thus advocating the use of crossbar switches for MPSoC designs.

Based on these examples, there are several key points to consider when designing the memory and communications architecture of a smart camera SoC. Similar to the PEs' requirements, the memory and communications system must compliment the data movements and accesses of the intended application. If the smart camera SoC is intended to achieve a standalone purpose, a highly data-parallel architecture such as [4] can be used, whereas if the flexibility is key, the stream architecture of [5] may be more suited. A hierarchical configuration [14] can be used for programmable PEs, as can crossbar switches [25], and compliments software programming models, which will reduce development time. However, the nature of the application and number and types of PEs and memories required in the system will dictate the overall decision of bus choice. The associative mesh can be considered a combination of the sensing and processing unit of a smart camera SoC, and can be used where high-throughputs are required. Hardware accelerator flexibility is best achieved through dynamic reconfiguration [17], where the memories of the accelerators can be customised to suit the accelerators' purpose. However, care is required to design a communications system that compliments the static and dynamic regions of the SoC.

Designers are free to conceive new memory and communications architecture that satisfy the requirements of the smart camera SoC, such as the hypercube topology of [6] which makes use of standard IP components to implement up to a 64-core MPSoC. However, such an architecture must serve the needs of the SoC and justify the resource consumptions. Table 2.3 summarises the advantages and disadvantages of smart camera SoC communication and memory architectures.

2.4 Communications Unit

Smart cameras have quickly become distributed smart cameras, and introduced several design challenges. Distributed cameras refer to a system of physically distributed

Table 2.3 Summary of smart camera SoC communication and memory architectures

Work	Memory Architecture	Communications Architecture	Pros	Cons	Throughput	Power Consumption	Resource Consumption
[14]	Independent off-chip for tiers 2 and 3	Wishbone system bus per tier	Memory and communications system is scalable	Bridging can introduce additional latencies and reduce bus performance	Low	Low	Low
	128B private SPM, 256B instruction ROM, 64B LUT per PE	Wishbone bridges	Private instruction memory makes cores easier to program	Coherency issues may have to be resolved before runtime			
	32KB and 64KB private BRAMs for tier 3		Private memory simplifies memory coherency	Concurrency is limited to that which the designer implements			
[12]	Custom memory architecture for PEs	Point-to-point inter-PE communications	Globally coordinated SoC Communications resolved before runtime	Resources may be under utilised	High	High	High
	Streaming-based data flow from sensing to processing unit	Associative mesh	High throughput streaming system	PEs always require communications logic to determine the next stage of execution flow			
			Coherency methods not required Scalable	Communications links that are unused in a graph may still be present in the hardware			
[5]	Streaming data input	Reconfigurable interconnections	Design/application flexibility	May not be scalable	High	Low	Low
	Line buffers for increased parallelism	Coordination by main controller	Memory and communications architecture allows PEs to be combined	Increased resource consumption over static architectures			

(Continued.)

Table 2.3 (Continued.)

Work	Memory Architecture	Communications Architecture	Pros	Cons	Throughput	Power Consumption	Resource Consumption
[4]	Streaming memory architecture - data reuse/locality inherent	Communications resolved for the application	High throughput	Application dependent	Unknown	Unknown	Unknown
			Small memory usage	Design inflexibility			
[17]	Dedicated access to memory for HW accelerators	Reconfigurable interconnect	Design/application flexibility	Memory coherency issues may exist	High	Unknown	Unknown
	Flexible private memory architecture(s) for HW accelerators	Bus bridging	Communications for HW accelerators	Reconfiguration overhead			
		Static bus for embedded CPU region	Dedicated memory access for HW accelerators	HW accelerators limited to size of dynamic region available			
		Point-to-point HW accelerator communications					
[25]	Globally shared memory	Crossbar switch-variable size	Arbitration present in bus	Wire congestion can occur for unused links	Medium	Unknown	Medium
			Easy to implement and scale	PEs access memory via shared bus—limiting throughput			
			Suitable for SoCs with DSP/microcontroller based PEs	PE loads may have to be equalised to ensure arbitration does not impact individual PE performance			

camera that may or may not have overlapping fields of view [18], which inevitably increases the volume of input data to be processed [24]. The increase in the field of view of the smart cameras increases the information that can be extracted from an image scene [18, 21, 22]. However, there are several geometric constraints that must be considered when designing a distributed smart camera system [19], including central projection, epipolar geometry, and planar scenes. Bramberger et al. [3] present a distributed embedded smart camera constructed from off-the-shelf components. The TMS320C6416 DSP by Texas Instruments is used as the processors of the SoC and are coupled together with memory via a PCI bus. Network connections are made through Ethernet and Wireless communications.

Hardware accelerators can also be used to optimise the flow of data to and from camera nodes. Zarezadeh and Bobda [26] implement an object request broker (middleware) for distributed smart camera SoCs for FPGA implementation to improve network performance. The middleware is implemented in hardware and can directly access the memory of a smart camera. The performance criteria set out or distributed smart cameras (in terms of the network) are the time taken for packets to arrive at the receiver, the time to prepare and process packets, and the time taken for packets to dissipate through the network. Comparisons of the hardware object request broker to a software broker executing on a PPC 405 show that the hardware broker achieves lower latencies than the software broker—nearly 100x faster, and also achieves higher and scalable bandwidths. This study shows that the use of hardware to create a distributed smart camera network can achieve higher throughputs than networks controlled by software, and more importantly throughputs that are scalable.

The communications unit of a smart camera SoC is very much dependent on the environment in which it will be used. As argued by Rinner and Wolf [18], distributed smart cameras are different from Wireless Sensor Networks (WSNs), as WSNs involve the processing of small amounts of data and are primarily concerned with conserving power where possible. Recent works have created wireless smart cameras such as [23], however with limited functionality. Furthermore, the use of wireless transmission to propagate information from nodes of a distributed smart camera may decrease performance when compared to wired transmissions due to the bandwidth available. The communications unit is primarily concerned with sending abstracted (processed) data to the end user. However, the protocols that can be used for this (USB, Firewire, etc.) is not relevant to the overview and discussion of smart camera SoC sensing and processing units, where image output functionality is often an inherent part of the SoC.

2.5 Summary

Smart camera SoCs are a diverse and complex area of design. Designers must be aware of the smart camera's intended application and its resource requirements. CMOS sensors are the popular choice for the sensing unit due to their ability to be tailored at the hardware level to incorporate pre-processing modules, and their increasing image quality and capture rate. The computational requirements of the application must be fully satisfied by the SoC's processing unit. This can be achieved in several ways through hardware and software development. Software development suits programmable, flexible solutions with quick time-to-market constraints. Software-based solutions focus on DSPs and implicitly design a memory architecture in software by exploiting data locality through caches. Software solutions must also address the storage and arrangement of program memory for optimal runtime execution.

Designers must decide on the importance of design flexibility, operating characteristics such as resource and power consumption, and throughput. Software solutions will have a nominal power consumption as their architecture does not change, however hardware-based solutions have the ability to modify hardware that directly impacts the power and resource consumption of the SoC—often in a positive way. Hardware solutions can be dedicated hardware designs hand-crafted for the purpose of the smart camera applications. These systems tend to boast high degrees of parallelism and often do not require memories for data reuse, as the system is optimised to use a piece of data once. Hardware solutions can also be more flexible and interfaced with software solutions as accelerators or coprocessors to carry out computation on the software side's behalf. Flexibility can be increased through the use of dynamic reconfiguration, which allows the resources of hardware accelerators to be rearranged into new computational engines.

The memory and communications architecture of such hardware solutions must be carefully designed to decrease memory access bottlenecks. Programmable and reconfigurable hardware accelerators can access data themselves or be provided with data through a PE. In both cases, the link between the PE and accelerator must be optimised for the level of interaction between the hardware and software side. Furthermore, memory accesses can be performed directly by accelerators and greatly reduces memory latencies. However, memory coherency must be considered by the designer and implemented where necessary.

Lastly, creating scalable SoCs for smart cameras, where the number of PEs can be increased to improve throughput, is a maintenance issue for designers and should be implemented in systems where flexibility is required. Scalable memory and communications architectures must be used in these cases such as crossbar switches and hierarchical bus topologies. However, these systems must be carefully designed to prevent the instantiation of unused or inessential resources. Hierarchical bus topologies are very scalable through the use of bus-bridges and tier-local memory buffers, however throughput can be reduced by the levels of arbitration required at each tier. Hardware-only solutions are the least flexible in terms of communications and memory architecture, but can be designed such that hardware modules can be instantiated in a plug-and-play fashion around the memory and communication network. Smart cameras can also be distributed systems, where data is propagated and/or collected from node-to-node. In this case, hardware accelerators have been shown to decrease network latencies and improve the throughput of the system.

References

1. Albani L, Chiesa P, Covi D, Pedegani G, Sartori A, Vatteroni M (2002) VIsoc: a smart camera SoC. In: Proceedings of the 28th European on solid-state circuits conference, ESSCIRC 2002, pp 367–370.
2. Blanc, N., Oggier, T., Gruener, G., Weingarten, J., Codourey, A., Seitz, P.: Miniaturized smart cameras for 3d-imaging in real-time [mobile robot applications]. In: Sensors, 2004. Proceedings of IEEE, pp. 471–474 vol. 1 (2004). doi:10.1109/ICSENS.2004.1426202
3. Bramberger M, Doblander A, Maier A, Rinner B, Schwabach H (2006) Distributed embedded smart cameras for surveillance applications. Computer 39(2):68–75. doi:10.1109/MC.2006.55
4. Chan WK, Chang JY, Chen TW, Tseng YH, Chien SY (2009) Efficient content analysis engine for visual surveillance network. IEEE Trans Circuits Syst Video Technol 19(5):693–703. doi:10.1109/TCSVT.2009.2017408
5. Chen J, Shen CF, Chien SY (2007) Coarse-grained reconfigurable image stream processor for digital still cameras and camcorders. In: Proceedings of custom integrated circuits conference, CICC '07. IEEE, New Jersy, pp 81–84. doi:10.1109/CICC.2007.4405686
6. Damez L, Sieler L, Landrault A, Dérutin JP (2011) Embedding of a real time image stabilization algorithm on a parameterizable sopc architecture a chip multi-processor approach. J Real-Time Image Proc 6(1):47–58. doi:10.1007/s11554-010-0184-3
7. El Gamal A, Eltoukhy H (2005) Cmos image sensors. IEEE Circuits Devices Mag 21(3):6–20. doi:10.1109/MCD.2005.1438751
8. Fons F, Fons M, Cant E, Lpez M (2012) Deployment of run-time reconfigurable hardware coprocessors into compute-intensive embedded applications. J Sign Proc Syst 66(2):191–221. doi:10.1007/s11265-011-0607-9
9. Heyrman B, Paindavoine M, Schmit R, Letellier L, Collette T (2005) Smart camera design for intensive embedded computing. Real-Time Imaging 11(4):282–289. doi:10.1016/j.rti.2005.04.006, www.sciencedirect.com/science/article/pii/S1077201405000239
10. Kerhet A, Magno M, Leonardi F, Boni A, Benini L (2007) A low-power wireless video sensor node for distributed object detection. J Real-Time Image Proc 2(4):331–342. doi:10.1007/s11554-007-0048-7
11. Kruijtzer W, Reyes V, Gehrke W (2005) Design, synthesis and verification of a smart imaging core using systemc. Des Autom Embedded Syst 10(2–3):127–155. doi:10.1007/s10617-006-0069-7
12. Lacassagne L, Manzanera A, Denoulet J, Mrigot A (2009) High performance motion detection: some trends toward new embedded architectures for vision systems. J Real-Time Image Proc 4(2):127–146. doi:10.1007/s11554-008-0096-7
13. Lepisto, N., Thornberg, B., O'Nils, M.: High-performance fpga based camera architecture for range imaging. In: NORCHIP Conference, 2005. 23rd, pp. 165–168 (2005). doi:10.1109/NORCHP.2005.1597015
14. Meng H, Freeman M, Pears N, Bailey C (2008) Real-time human action recognition on an embedded, reconfigurable video processing architecture. J Real-Time Image Proc 3(3):163–176. doi:10.1007/s11554-008-0073-1
15. Moorhead T, Binnie T (1999), Smart cmos camera for machine vision applications. In: Seventh international conference on image processing and its applications (Conference, Publication No. 465), vol 2, pp 865–869. doi:10.1049/cp:19990448
16. Matsushita, N., Hihara, D., Ushiro, T., Yoshimura, S., Rekimoto, J., Yamamoto, Y.: Id cam: a smart camera for scene capturing and id recognition. In: Mixed and Augmented Reality, 2003. Proceedings. The Second IEEE and ACM International Symposium on, pp. 227–236 (2003).doi:10.1109/ISMAR.2003.1240706
17. Oetken A, Wildermann S, Teich J, Koch D (2010) A bus-based soc architecture for flexible module placement on reconfigurable fpgas. In: International conference on field programmable logic and applications (FPL), pp 234–239. doi:10.1109/FPL.2010.54
18. Rinner B, Wolf W (2008) An introduction to distributed smart cameras. Proceedings of the IEEE 96(10):1565–1575. doi:10.1109/JPROC.2008.928742

19. Sankaranarayanan A, Veeraraghavan A, Chellappa R (2004) Object detection, tracking and recognition for multiple smart cameras. Proc IEEE 96(10):1606–1624. doi:10.1109/JPROC.2008.928758
20. Senior A, Pankanti S, Hampapur A, Brown L, Tian YL, Ekin A, Connell J, Shu CF, Lu M (2005) Enabling video privacy through computer vision. IEEE Secur Priv 3(3):50–57. doi:10.1109/MSP.2005.65
21. Trivedi M, Gandhi T, Huang K (2005) Distributed interactive video arrays for event capture and enhanced situational awareness. IEEE Intell Syst 20(5):58–66. doi:10.1109/MIS.2005.86
22. Trivedi M, Huang K, Mikic I (2005) Dynamic context capture and distributed video arrays for intelligent spaces. IEEE Trans Syst Man Cybern Part A Syst Hum 35(1):145–163. doi:10.1109/TSMCA.2004.838480
23. Wang Y, Velipasalar S, Casares M (2010) Cooperative object tracking and composite event detection with wireless embedded smart cameras. IEEE Trans Image Proc 19(10):2614–2633. doi:10.1109/TIP.2010.2052278
24. Wolf W, Ozer B, Lv T (2002) Smart cameras as embedded systems. Computer 35(9):48–53. doi:10.1109/MC.2002.1033027
25. Xu J, Wolf W, Henkel J, Chakradhar S, Lv T (2004) A case study in networks-on-chip design for embedded video. Proceedings of Design, automation and test in Europe conference and exhibition 2:770–775. doi:10.1109/DATE.2004.1268973
26. Zarezadeh A, Bobda C (2010) Performance analysis of hardware/software middleware in network of smart camera systems. In: International conference on reconfigurable computing and FPGAs (ReConFig), pp 196–201. 2010, doi:10.1109/ReConFig.59

Chapter 3
Reconfigurable Architectures for Distributed Smart Cameras

Christophe Bobda, Michael Mefenza, Franck Yonga and Ali Akbar Zarezadeh

Abstract Embedded smart cameras must provide enough computational power to handle complex image understanding algorithms on huge amount of data in-situ. In a distributed set-up, smart cameras must provide efficient communication and flexibility in additional to performance. Programmability and physical constraints such as size, weight and power (SWAP) complicate design and architectural choices. In this chapter, we explore the use of FPGAs as computational engine in distributed smart cameras and present a smart camera system designed to be used as node in a camera sensor network. Beside the performance and flexibility size and power requirements are addressed through a modular and scalable design. The programability of the system is addressed by a seamless integration of the Intel OpenCV computer vision library to the platform.

3.1 Introduction

The importance of visual sensing and processing is continuously growing. Besides traditional areas like visual inspection in manufacturing, the processing of image sequences in order to gain knowledge is a reality in new application fields like automotive, security, and assisted living. Public access facilities such as airports,

C. Bobda (✉) · M. Mefenza · F. Yonga
CSCE Department, University of Arkansas, Fayetteville, AR, USA
e-mail: cbobda@uark.edu

M. Mefenza
e-mail: mmefenza@email.uark.edu

F. Yonga
e-mail: yfrancku@email.uark.edu

A. A. Zarezadeh
Dspace GmbH, Paderborn, Germany
e-mail: akzare@dspace.de

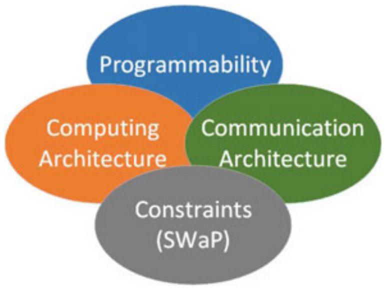


Fig. 3.1 System design space for embedded smart camera nodes in distributed setup

transport stations and banks use camera to increase security. In assisted living for older people, video processing can be helpful in detecting emergency situations [23]. Cameras are used in high-end cars to detect lane crossing, to measure and manage the distance to collision, obstacle and pedestrian avoidance, parking assistance, etc. The huge amount of stationary and mobile cameras already deployed can be leveraged to provide valuable information, which cannot be obtained from single isolated cameras. Current systems use a central computer to process data collected from dozens of cameras installed at various locations. This approach increases the bandwidth requirement as well as the computational requirement of the central processing platform. As consequence a real-time response to events becomes increasingly unlikely. Collaborative smart cameras can overcome this obstacle by processing information in-situ and exchanging relevant knowledge with neighbor. In order to reach this objective, smart cameras should provide computational and communication resource to (1) handle complex image processing on board, while ensuring real-time data and knowledge exchange among the cameras. The programmability of the whole system must be simplified for a broad acceptance in the image processing community, which consist essentially of software designers.

In a distributed embedded and mobile environment, system design must be done under size weight and power (SWAP) constraints. Performance is defined by the nature of computing and communication architectures with component specialization as the key ingredient in design. However, component specialization increase programmability burdens and make the use of the resulting platform difficult to computer vision designers. The relation between architectures and design objectives is illustrated in Fig. 3.1.

When designing a new architectures, all parameters must considered simultaneously for an optimal outcome.

3.2 Survey of Embedded Smart Camera Architectures

Since DVT together with the Georgia Institute of Technology pioneered smart cameras in 2000 [27], dozens of commercial and non commercial systems were developed for commercial and academic use, some of which are listed here.

In [6] a distributed and scalable smart camera for surveillance was presented. This multiprocessor hardware platform consists of a network of general purpose processors and digital signal processors. The wireless CITRIC camera introduced in [10] combines the OmniVision OV9655 with the Intel XScale PXA270 processor. The MeshEye [17] focusses on low-power and uses an Atmel AT91SAM7S micro controller.

More recently, a survey of smart cameras was presented by Rinner and Wolf [1]. Thereafter the computation is performed in software in almost all existing systems. High-end PCs are used to provide the required computational power. Most advanced embedded visual systems use sequential processors like the Intel Xscale or DSPs.

The problem with all those smart cameras on the market is that they are optimized in only one direction. Those using sequential processors are optimized for programmability and flexibility but not for performance. Those tailored for a custom computing are difficult to program or not programmable at all.

Some manufactures have recognized the benefit of custom implementation and use dedicated hardware (FPGA or ASIC) to accelerate a dedicate computation. FPGA-based platforms were investigated in [1, 3, 34]. In [3] a Xilinx Spartan-3E FPGA is used as co-processor for pre-processing of video input stream from two cameras. The result is forwarded to a 400 MHz PXA255 processor for knowledge inference. The systems in [3, 34] use a single FPGA for video processing. Tasks in [34] are limited to a single process like the gesture recognition of single hand.

Video processing is a complex task, which requires the coordination of distributed hardware and software components within a customized run-time architecture to provide real-time execution of various image understanding and fusion tasks. The computing infrastructure must be embedded within a camera to provide a versatile set of interfaces. To address these issues, we have developed a scalable and modular FPGA-based camera system that provide performance on a small footprint, low power and flexibility. Using FPGAs, complex tasks can be implemented as dedicated module on the FPGA logic, while control dominated parts of applications can be implemented in software either on an embedded processor. The partial reconfiguration capability of FPGAs allows for hardware restructuring to adapt system functionalities to run-time changes. Designing for FPGA is a challenging task, in particular for non-hardware programmers as is usually the case in the computer vision community. To make the computing platform seamlessly usable Intel's widely used OpenCV image processing library has been integrated on the FPGA SoC with the low-level kernel functions, available as hardware accelerators. Software designers can then gain performance typically associated with a hardware implementation, while using an interface at a very high level of abstraction that hides details of the implementation

3.3 Design of an FPGA-Based Smart Camera Platform

Our motivation in designing a new platform was to provide a universal video processing system, which could be used as prototype or end-product in various embedded imaging applications. As prototyping platform, resulting environment

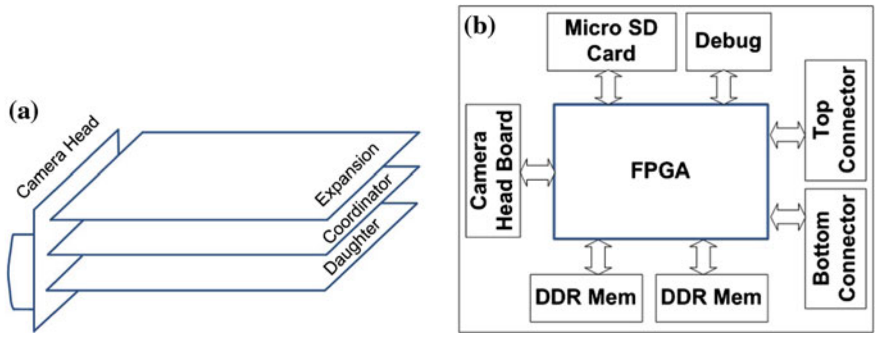


Fig. 3.2 **a** Modular organization of the FPGA-Based smart camera, and **b** organization of the computing and coordination module

would help researchers and developers in their investigations without having to rely on the cumbersome existing evaluation platforms and workstations. Video processing is mostly streaming based, with CCD and CMOS devices used as entry point for image acquisition. Processing is then performed in a given number of steps, some of which can be iterated several times, according to the algorithm. The requirements on the processing unit in the computation chain differ from problem to problem. While critical tasks in a real-time environments require more computational power and therefore a hardware implementation, control tasks are better served with a sequential processors. At the end of the chain, the computed data can be sent to other units in a network, displayed on a monitor or simply saved on the platform permanent storage. A universal platform for embedded environment must provide interconnection capabilities. Because number of interfaces that can be used for interconnection is very large, their inclusion in a single platform would have a negative impact on the size constraint. In order to overcome this limitation, we have designed a modular systems consisting of a set of components that can be composed in various configuration according to user's needs. The result is a 3D arrangement (Fig. 3.2a) aimed at reducing the overall size. The modules of the systems are explained below:

- *The Video Capture Module*: the video capture module used for image acquisition. It features a CCD/CMOS chip, a lens and a connection interface to the communication module. Video captured by the CCD/CMOS are provided to the FPGA computational module via the corresponding interface on the communication module. The capabilities of the smart camera can be modified by using a different video capture module with user's defined image sensors(CCD or CMOS), image resolutions, speed, etc...
- *The Communication Module*: The communication architecture is one of the most important components in a smart camera network. Because the operating environment is not known a priory, a rich set of interfaces must be provided to allows camera node integration in ubiquitous environments. A single platform with the FPGA computation module and all commonly needed interfaces would make the system

on a single module too large, thus negatively impacting the size requirements. We solved this issue by separating the computation from the communication. The communication module is solely used to connect the system the rest of the world, including other cameras in a network. This organization reduces the overall system size, since the grow is done the 3rd dimension. The current communication module provides two USB devices and one USB OTG, one Ethernet, one RS232, power supply and JTAG configuration connection. A small micro controller is used to control the data flow between the devices mounted on the communication module. Also, the configuration of the FPGA can be done at start-up by the micro controller. While the Ethernet interface provides only wired connection, the USB interfaces can be used for wired/wireless connection among distributed cameras.

- *The Coordinator Module*: As the name indicates, the coordinator module controls the whole system by providing configuration and operational parameters to the devices in the system. In default configuration, the coordinator module is used for computation besides system's control. As shown in Fig. 3.2b, the coordinator module features a FPGA Virtex-4 FX, DDR memories, connectors, a flash memory, additional image sensor connector and JTAG debug lines.
  With the embedded PowerPC and the surrounding logic, the FPGA provides the best prerequisites for Hardware/Software implementation needed for efficient video streaming. The connection to the communication module is done with a bottom connector, which uses high-speed LVDS signals. A second connector is available on the top side to attach additional processing modules for applications that require more performance. Additional computational moduled can be used to provide more FPGA logic in case the resource on the coordination FPGA are not enough.
  In order to increase the performance of computation inside the FPGA, parallel accessible DDR memories are used around the FPGA. This allows independent modules computing in parallel and using different set of data to access separated memory and reduce contention. Each memory provides 64Mbyte, which is enough to store several high-resolution images. The parallel connection of the memories on the FPGA further allows for the implementation of message passing like paradigms in the FPGA [8]. An additional connector for an image sensor with direct connection to FPGA, a flash memory, a debug serial RS232 and a JTAG programming/debug interface are also availble on the coordinator module.
- *The Processing Modules*: For applications with many complex and dedicated functions to be implemented in FPGA, the use of a single coordination and processing module might not be enough. In this case, the system can be extended with additionally processing modules. Each processing module is connected on the top side to the next processing module and on the bottom to the previous processing module, the first of which is connected to the coordination module. The processing modules are built in the same way like the coordinator module, without the flash memory and the second image sensor input connector.

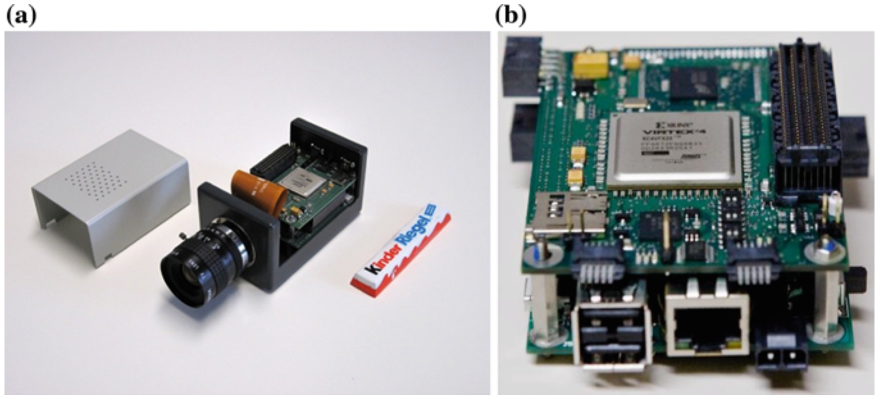


Fig. 3.3 **a** The PICSY camera in a case(a), PICSy internal: controller board featuring an FPGA, a top connector, flash drive and memory. **b** The communication module with two USB devices, one USB OTG, Ethernet interface, UART, GPIO and JTAG interfaces

3.4 Connecting Multiple Cameras

In many application domains, multiple-head cameras can provide additional information needed for an efficient processing. A multiple-head camera consist of several head board attached to a single processing unit. The picture captured by the head are combined in the processing unit to extract more exact information. A 3D image reconstruction would be helpful to better identify the contours of objects and increase the quality control in a production chain. Using the base configuration for this purpose will require to have several camera units working together in order to capture objects from different perspectives. This increase the cost of the resulting solution. We avoid this by using multiple-head camera to collect pictures from different area and fuse the data in the FPGA. Attaching multiple heads camera is done on our platform by using a dedicated expansion module connected to the coordinator module. The expansion module allows for the connection of up to three different camera heads using high-speed LVDS lines. The LVDS allows modules to communicate on a distance of up to 15 m without performance penalty.

The resulting smart camera was named PICSY camera, which stands for Potsdam Intelligent Camera System. Figure 3.3a shows the camera in the basic configuration in a case. The internals of the cameras are shown in Fig. 3.3b. The communication module at the bottom is connected to the coordination module at the top. The communication module offers many interfaces including two USB-devices and one USB-OTG, which can be used to provide wireless connection (bluetooth, Zigbee, WLAN). Wired connection can be done over the Ethernet port or the general purpose input/output lines.

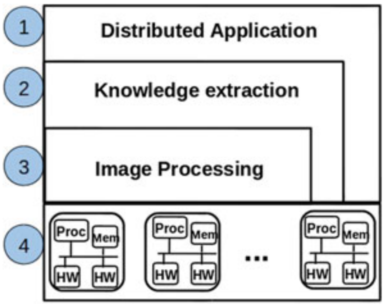


Fig. 3.4 Overall design flow of a smart camera network

3.5 Design Flow

3.5.1 High-Level Specification

In order to tackle the complexity of distributed visual applications, the design of applications for a set of distributed smart camera usually follow the layered approach of Fig. 3.4. The highest layer, the application layer, allows for the abstract specification of distributed visual applications regardless of the details of image processing. Knowledge provided by each camera node in symbolic form is used to define the behavior of the whole system. For instance for a cooperative traffic security application, the cooperative behavior could state that if a vehicle detects a pedestrian crossing the street, then all vehicles in a 300 m radius should be warned. Symbolic expressions like "*pedestrian crossing street*" and "*warn vehicle within* 300 m" are used in this case, without details on their implementation on each vehicle. The knowledge representation below the application level provides facts extracted from the cameras. This level describes the environment around each camera. Image understanding is used to analyze the scene around a single camera and provide a database of facts like "*person on the floor*", "*two way road*", and "*man running in direction x*") tagged with the corresponding camera. Image understanding algorithms are capable of tracking dozens of people in a single camera and infer that someone is running, based on the change in the frames and the speed of the camera. Input for knowledge extraction is provided by the lower level image processing, which segments pictures and extracts objects like rectangle and circle, or even more complex contour. The description at this level can be executed on any platform, in particular on a workstation, which can be used for verification and validation before mapping to the computer architecture inside the camera.

In this chapter, we will assume that upper layers have been implemented and focus on the implementation at a camera node-level.

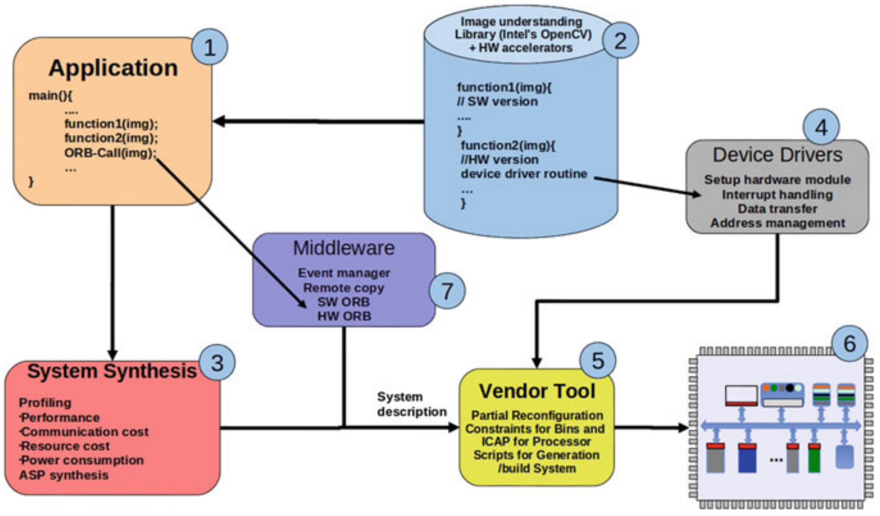


Fig. 3.5 The design flow for an embedded smart camera node

3.5.2 *Design Environment for Single Camera Nodes*

Once the specification of the overall application has been done and simulated, the implementation at a node level can be done in the environment illustrated in Fig. 3.5, which allows for the integration of hardware accelerators in OpenCV applications.

In step 1 (label 1, Fig. 3.5) a designer starts with an application written in C/C++ with image understanding functions, available in an image processing library, (label 2, Fig. 3.5) either as software or hardware implementation. In order to make the work easy for people in the image processing community, which essentially consists of software developers, we build our library around Intel's widely used image processing tool *OpenCV*. This allows developers willing to add new algorithms to write code for the new architecture in software as usual, and also to migrate existing applications on the target embedded platform. In order to perform the architecture generation, we need performance parameters from the functions used in the application. In step 2 (label 3, Fig. 3.5), we first profile the application to acquire performance, area usage on the FPGA, power consumption of all the functions used, and communication costs among them. Those parameters are then used to perform the synthesis and choose the best architecture for the application. The low-level image processing kernel will be implemented in hardware and available in the IP-library. Functions implemented in hardware require a device driver to be accessible by the processor. In [26] we developped a generic driver interface called the streaming data interface (SDI) that will be used in this framework. It consist of a set of status and command registers, and functions to access those registers for the communication between the processor and the function implemented in hardware. The parameterization (label 4, Fig. 3.5) defines the number and size of the registers and the operation mode of the access functions. Finally, the middleware components will be added to the abstract description of the architecture computed in the synthesis step (label 7, Fig. 3.5) to complete the description of the SOC architecture These is then provided as input to the FPGA vendor tool in order to generate the complete SOC.

The integration of hardware accelerators requires some drivers on the operating system (Linux in this case) side and a bus interface on the hardware side. Also, a scheduling must be available to serialize the access to shared resources (bus, memory) if many hardware accelerators are used in the system. To overcome this problem we designed a generic interface, the streaming data interface (SDI) in hardware and generic drivers functions in the operating system. Those are containers in which components can be placed and configured according to the computation to be done. This architecture offers a wide spectrum of possibilities to connect several hardware accelerator as low-level function in a computing stream attached to an embedded processor.

3.5.3 Streaming Data Interface

The SDI, which was originally introduced in [26], is a set of skeletons components to allow hardware accelerators to be uniformly designed and easily integrated in the OpenCV environment. It provides data and control signals to link hardware accelerators together and to access data source and data sink. A set of control and control and configuration registers is available to avoid a communication between the processors and the hardware accelerators. A scheduler is used to coordinate access to share resource such as buffers and memory by hardware and software functions in the system.

The major focus in developing the SDI was on the implementation of a concept which connects physical external memory to the several implemented application oriented processor units, to supply them with required data and to store the processed results back to memory without intervention of main processor. The SDI is very useful in image processing where computation takes place in a streaming way. A data source supplies pixels one by one and sends them to the first processing unit (PU). The PUs (which are performing the calculations) are cascaded in any number and order like a chain and finally the last PU is connected to a data sink.

An example of the SDI usage is shown on Fig. 3.6. Captured video frames from the CCD module are placed into the DDR memory after a conversion from Bayer to RGB. The processor then process the the video with part of the algorithm in hardware and parts in software. The result is sent to VGA output module for visualization of results.

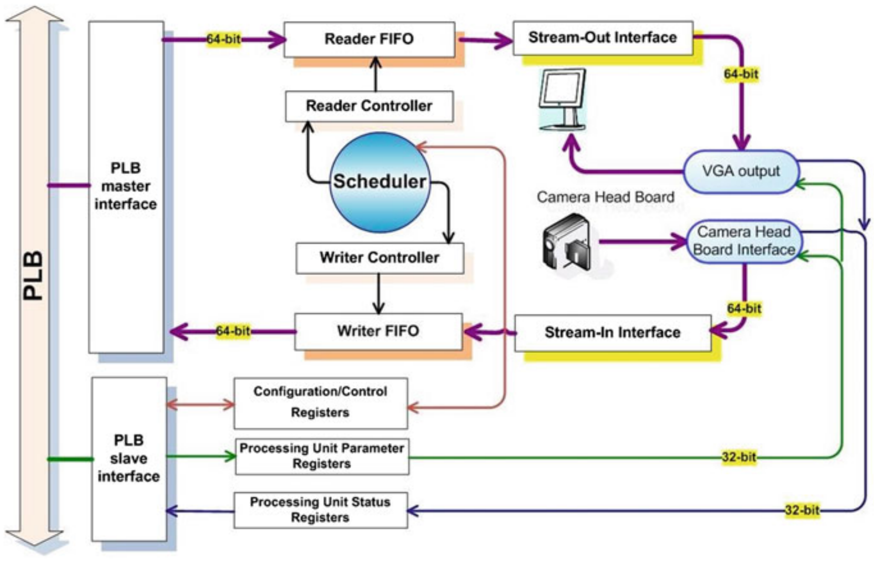


Fig. 3.6 Integration of SOC for video capture and display with the SDI

3.6 Evaluation of Communication Performance

The evaluation of the platform was done with the base system consisting of the interface module, the coordinator module and the camera head module. Using different applications implemented in OpenCV with hardware acceleration, we measured the resource usage, the power consumption and system performance. The results are explained in Tables 3.1 and 3.2. As case study, we implemented the image segmentation in three different configuration and compared the results.

3.6.1 Resource Usage

To evaluate the resource usage, two designs were compiled with OpenCV running under Linux on the integrated PowerPC and various IPs implemented in the surrounding FPGA logic. A video capture module was used to acquire images from the image sensor and place them in the memory after format conversion from Bayer to RGB. Thereafter the processor performed some computation in software and displayed the result on the VGA. The connection of the hardware for video capture and display was done using the SDI controller. As shown in Table 3.1, the two implementations differ only on the Ethernet IP used. In the first case, the more complex XPS LL TEMAC core is used, leading to an overall resource usage of 94 %, while the use of the lightweight Ethernet Lite leads in the second case to a resource usage of 74 %.

Table 3.1 Resource usage for different FPGA configurations on the base system

IP	# Slices design 1 (%)	# Slices Design 2 (%)	FIFO usage design 1 (%)	FIFO usage design 2 (%)
DDR SDRAM	47	29	22	16
cam i2c	2	2		
bayer2rgb	1	1		
SDI controller	15	15		
VGA output	1	1		
XPS intc	1	1		
LL TEMAC	6			
Ethernet lite		7	4	
RS232 UART	1	1		
Image sensorcap	1	1		
Total system	94	94	63	

Table 3.2 Power measurement for different FPGA configurations on the base system

Design	# Slices (%)	FIFO usage (%)	Power (Watt)
Empty	0	0	2.69
PowerPC + communication core	67	48	5.64
PowrPC + Ethernet lite	74	32	6.07
PowerPC + LL TEMAC + DMA	94	63	6.6
PowerPC + LL TEMAC + FIFO	87	42	5.67

Using the XPS LL TEMAC increases the size of the memory controller because of the available DMA-controller. According to the complexity of the image processing accelerators, FPGA might be enough. In this case, the processing module could just be replaced by a more powerful one featuring a most powerful FPGA with the same footprint.

3.6.2 Power Consumption

In the same way, we measured the power consumption by running OpenCV and Linux on the PowerPC and moving some functions into hardware. The implementation was done as in the previous case using the base system. According to the hardware load, the power consumption was measured and documented in Table 3.2.

3.6.3 Case Study

To demonstrate the performance superiority of our system, we implemented a segmentation algorithm. Segmentation is a basic step in many computer vision

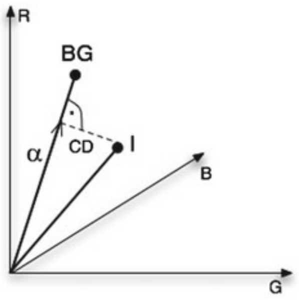


Fig. 3.7 Foreground computation using color distortion and brightness distortion between background and current image

application. Because of it's complexity and it's streaming and inherent parallel nature, it is a good candidate for hardware acceleration. We implemented the Horprasert et al. [19] variation because of its robustness and its capability to deal with shadow detection. A graphical significance of the algorithm is illustrated in Fig. 3.7. An initially trained background image BG is compared to the current image I. The decision for each pixel if it is a foreground or background pixel is primarily based on two properties: the color distortion CD and the brightness distortion α. The values are compared to some off-line trained thresholds.

First, we executed the application a standard workstation using OpenCV. The program needed approximately 23 ms to compute the foreground for a 640 × 480 sized image. This number does not even include the overhead need for grabbing images and displaying results.

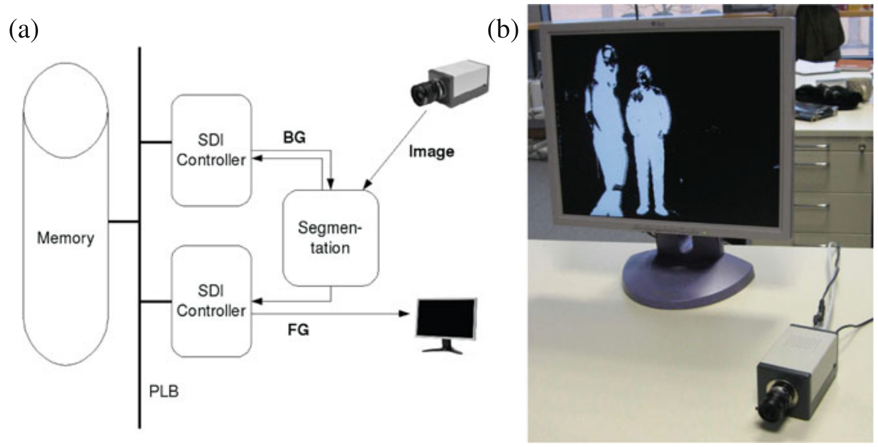
Second, we implemented the same implementation on the PICSy smart camera using the integrated OpenCV running on Linux, however without hardware acceleration. Compared to the Intel Core2Duo at 3.16 GHz of the workstation the embedded PowerPC-Processor in the Virtex4-FPGA has a 300 MHz speed is quite slow. As expected the pure software implementation on the smart camera performed lower that the pure software implementation on the workstation, with an execution time of 1,598 ms to compute a single frame.

Finally we implemented the segmentation algorithm in hardware with additional SDI controller to handle the memory access for the background image (Fig. 3.8a). The DSP slices of the Virtex4-FPGA increase the speed of operations such as multiplications. The result is obtained in one clock cycle only. To compute CD and α for two pixels 24 out of the 32 DSPs are needed. The hardware module uses 793 slices which is 9.28 % of the FPGA resource. With a data width of 64 bits two pixels can be processed per clock cycle. Due to the pipelined chain of hardware modules (grabbing $\rightarrow$ bayer to RGB convertion $\rightarrow$ segmenting) we get the foreground for two pixels in every clock cycle (50 MHz), thus a delay of 3.07 ms for each frame (640 × 480).

The time measurements for the software-only and hardware/software implementations are summarized in Table 3.3. The hardware implementation outperforms the other two by a factor of 7 (PC) and 325 (camera OpenCV) respectively. The software solution on the smart camera suffers from the slow memory access of

Table 3.3 Measurements of the segmentation on different systems

System	Time (ms/frame)	# Frames/s
PC, 2 × 3.2 GHz, OpenCV	≈23	44
Smart camera, OpenCV in SW	≈1598	2
Smart camera, HW/SW Co-Design	3.07	325



Two SDI controllers for segmentation task, one handling background, one handling foreground

Example output of the smart camera

Fig. 3.8 **a** Internal FPGA system-on-chip architecture, **b** video segmentation implemented as hardware/software system in the PICSy

the PowerPC to load and store the image pixels. Grabbing and converting however is fast because it is already made in hardware. The foreground segmentation is made independently for each pixel. Therefore the computation time for an image is linear with its size.

Figure 3.8b shows the system set-up with the PICSy attached to the VGA monitor, which displays the results of the computation. The connection to the monitor was done using a VGA adapter module connected to the GPIO-pins of the interface module.

3.7 Increasing Flexibility with Partial Reconfiguration

The partial reconfiguration capabilities of Xilinx FPGAs can be exploited to increase the flexibility of the platform, while keeping the performance high. The partial reconfiguration allows for keeping part of the FPGA unchanged, while replacing some hardware modules in other regions on the fly. Hardware can be temporally shared

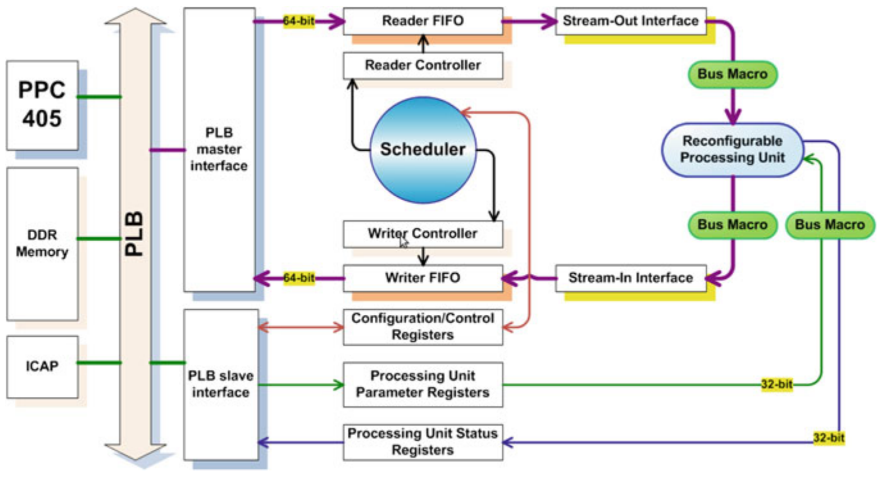


Fig. 3.9 Partial reconfigurable SDI controller

among tasks at run-time without performance loss. Controlling the partial reconfiguration partial from the embedded processor make this feature even more attractive and allows for the realization of self-organization in embedded systems as proposed in [29]. Self-organization would be helpful to allows surrounding cameras to overtake the tasks that were running on a node in case of failure.

Exploiting the partial reconfiguration for self-organization however requires the platform to be tailored for this purpose. The computation architecture within the smart camera can be reduced to a system on chip in FPGA with Linux running on the embedded PowerPC. The system can be therefore designed according to the techniques and recommendation for Partial Reconfiguration (PR) described in [39].

Handling the bitstreams used by the system to reconfigure the FPGA at run-time requires an efficient file system on the platform. Also, a communication medium and the protocol for exchanging huge amount of data must be provided to allow for exchange of bitstream among cameras at run-time. The focus in this section is the realization of the most critical part, which is the definition and the design of the partial reconfigurable regions.

The streaming data interface introduced in Sect. 3.5.3 is used to defining the partial reconfigurable regions and allow data exchange between the hardware accelerators in these regions and the rest of the system. The controller within the SDI handle the transfer of data processed by the hardware accelerators in the reconfigurable regions.

Figure 3.9 shows the integration of partial reconfigurable hardware accelerator in the FPGA-SoC using the SDI. Several reconfigurable processing units can be implemented in reserved regions on the FPGA delimited by bus macros to enforce the integrity of signals among reconfigurable regions and the rest of the system.

Table 3.4 Timing measurements for different size of reconfigurable region

	Small	Medium	Large
Number of CLB	96	192	288
Number of DSP48	8	16	24
Number of BRAM	4	8	12
Number of clock areas	1	2	3
Reconfig. time (ms)	43.637	45.089	49.739
Time std. deviation (ms)	0.879	0.890	0.552

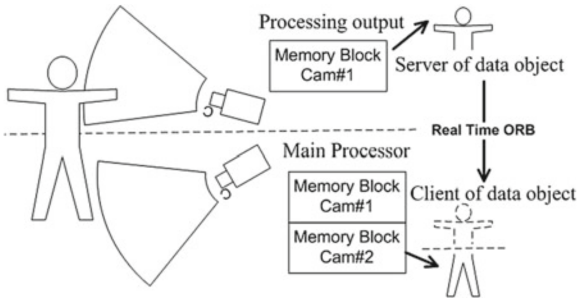


Fig. 3.10 Collaborative distributed sensing and processing application

To give an idea of the impact of the utilization of partial reconfiguration in the FPGA within the smart camera, the reconfiguration time measured from the operating system is provided for different design and used as indicator in Table 3.4.

Resource assignment in FPGA has a big impact on bitstream size. More than the number of CLB, their locations into the FPGA contribute to the size of the bitstream. The FPGA is divided into clock domains and the position of the PR Module in these areas influences bitstream sizes.

3.8 The Communication Architecture

Collaboration in a distributed camera network is important for event and knowledge exchange. The design of a collaborative video processing system is not only driven by the low-level communication infrastructure. High-level functions allowing for remote access to partial data is a prerequisite for performance, productivity and interoperability.

Figure 3.10 illustrates an emergency fall scenario, which can only be detected through a collaboration between two cameras. The person lying on the floor is only be partially visible by the two cameras. The partial information they see does not allow them to infer a fall. However, exchanging their partial information such as contours, histogram and location would will allow the situation to be detected.

Deploying a collaborative smart camera system like the one previously described requires a highly efficient communication infrastructure, which can be provided only by dedicated hardware implementation of complex components in hardware. Such dedicated architectures always place a programmability burden on designers, who must deal with low-level communication details in their application. A middelware approach can help overcome this burden, by providing high-level remote data access function to designers and making the mapping to the hardware infrastructure transparent to the user.

In this section, we describe a hardware middelware technology, the HardORB, and it's integration in smart camera node. The presented middelware is based on an embedded version of CORBA, which offers many benefits such as standardization by the Open Management Group (OMG), widely usage in hard real-time systems. The OMG has adopted CORBA/e for embedded systems as a standard which dramatically minimizes the footprint and overhead of typical middleware.

Implementing CORBA in FPGA can be done by running CORBA on Linux on an embedded microprocessor within FPGA. This approach has the advantage of simplicity but does not meet the performance requirements.

To meet the performance requirements and provide a low-latency and seamless communication among cameras, the entire ORB engine was implemented in hardware in FPGA, resulting in the Hardware Object Remote Broker (HardORB).

3.8.1 Hardware/Software Architecture for Real-Time Communication

An ORB is a mechanism for invoking operations on an object in a different remote process that may be running on the same or a different computer.

A CORBA Object (Fig. 3.11) is identified by an Interoperable Object Reference (IOR) containing an Object ID. This IOR contains all the information to obtain an object servant. A stub is a bridge between a client and the ORB, which acts as a proxy, converts method calls into messages and serializes the input parameters of an operation call into a General Inter ORB Protocol (GIOP) request message, and de-serializes reply messages to deliver operation results and exceptions to the client. A skeleton is a bridge between a servant and the ORB. It converts binary request packets into native data types and invokes the operation on the servant, and serializes results and exceptions in the reply to the client. An Interface Definition Language (IDL) compiler automatically generates stub and skeleton codes from the interfaces definition in IDL file to software programming languages. The on-the-wire protocol of CORBA is called GIOP. This protocol defines the different message types (request or reply) that can be exchanged between client and server applications. The major GIOP specialization is the Internet Inter-ORB Protocol (IIOP), which is use in TCP/IP networks.

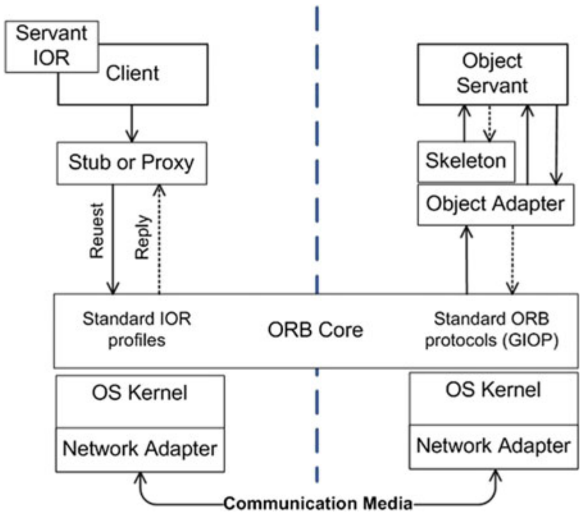


Fig. 3.11 CORBA middleware framework

The standardization of CORBA by OMG as object-oriented middleware has led to dozens of middleware systems developed for commercial and academic use. Reference [30] presents an embedded middleware for smart camera networks and sensor fusion has been with proprietary agent-based middleware. In [11] a software framework supports transparent intra- and interprocessor communication for network of DSCs has been proposed. Those solutions are pure software and do not meet the real-time communication requirements. There some existing native gate level implementations of ORBs, mainly exploited in the area of software defined radio (SDR). The Integrated Circuit ORB (ICO) engine [9, 20] implements a hardware CORBA ORB. PrismTech's ICO is such an ORB in SDR applications. The ICO engine, is responsible for implementing the transfer syntax used in CORBA messages. A complete semantic mapping of CORBA-IDL and GIOP has been investigated for hardware components for SDR application in [14].

The real-time performance of the Ethernet is determined by factors like protocol stack traversal times and delays related to complex operating system internal interactions in the end nodes of the network [28]. Stack traversal time and jitter are critical for real-time performance of the communication infrastructure. The successful usage of hardware ORBs in the SDR area inspired our solution for embedded smart camera networks.

3.8.1.1 Overall System Architecture

Figure 3.12 shows the system-on-chip architecture of embedded smart camera system incorporating hardware ORB component. Data are acquired from or supplied to

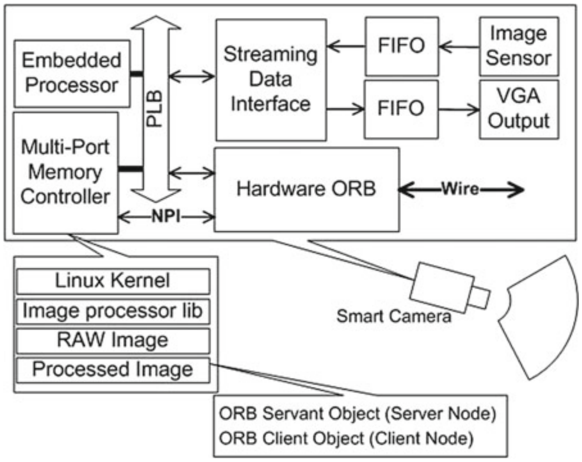


Fig. 3.12 Our developed SoC design for Hardware ORB approach

peripheral components trough input/output streaming channels connected to system memory. The HardORB is integrated into the System-on-Chip architecture described in Sect. 3.5.3, with the streaming data interface (SDI).

The embedded processor is used for the software part, while the HardORB is implemented on the surrounding FPGA logic.

3.8.1.2 The Hardware ORB Architecture

This lower layers of standard TCP/IP stack are implemented in hardware and supports independent read/write to DDR Memory with the Native Port Interface (NPI) of the Xilinx Multi-Port Memory Controller (MPMC). Processed information of recent video frame are held in a reserved location of DDR memory. On a remote request from a CORBA client in a different smart camera node, the HardORB fetches the corresponding data from memory and send it to the corresponding node without processor intervention. The HardORB is then in charge of serializing and deserializing of data through the GIOP/TCP/IP/Ethernet protocol stack. On the client side, the HardORB-client manages the low-level transactions, while the HaordORB-Server does the same on the server side. The internals of the HardORB-Server are illustrated in Fig. 3.13.

The embedded processor is connected to the trasmission (TX) and reception (RX) dual port buffers of the MAC layer (*Tx Buffer* and *Rx Buffer*) trough the processor local bus (PLB) slave interface and enables protocol stack on Linux stack. The main advantage of this architecture is to have the possibility of using other network services on the operating system. It is even possible to use another auxiliary ORB as software on the embedded processor and simultaneously handle individual real-time middleware services in Hardware ORB. As shown in Fig. 3.13, when a new

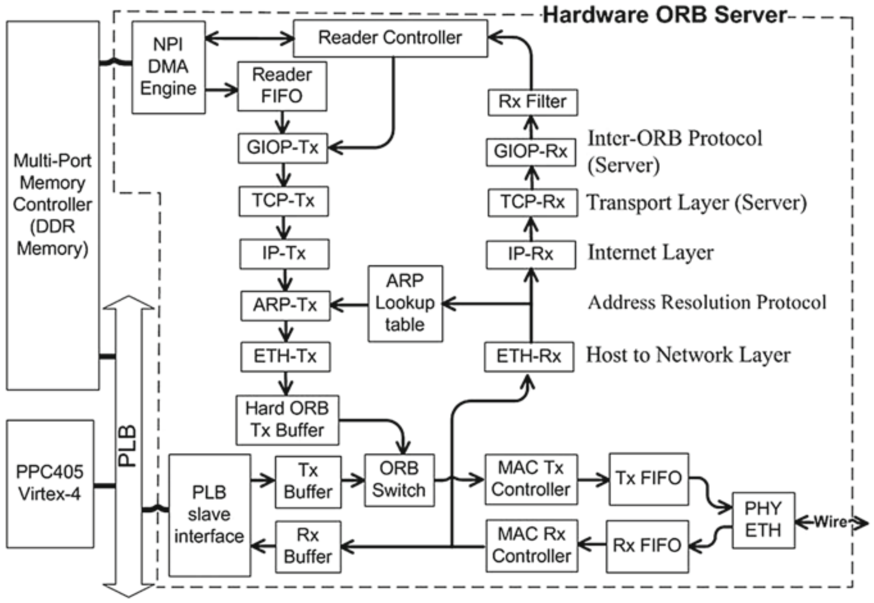


Fig. 3.13 The architecture of our developed Hardware ORB Server IP core

packet arrives from the *MAC Rx Controller*, one copy placed in the *Rx Buffer* of the embedded processor and simultaneously forwarded to the HardORB. The HardORB monitors incoming packets and search for packet of interest, with predefined TCP port number and CORBA Interoperable Object Reference. Upon detecting a packet of interest, the interface to the embedded processor is blocked to avoid racing between hardORB and embedded processor. Relevant data are then fetches from DDR memory by the *Reader Controller*, which constructs the TCP/IP message and stores it in the *HardORB Tx Buffer*. There are two separate transmission buffers, the *Tx Buffer* for the embedded processor and the *HardORB Tx Buffer* for the HardORB. The *ORB Switch* connects one of these two buffers to *MAC Tx Controller* for transmission.

Figure 3.14 shows the internal architecture of the HardORB-Cliend.

The configuration is similar to HardORB-Server. However the *Reader Controller* is replaced by a *Writer Controller*. Additionally, a transaction must be started by the embedded processor, which activates the transmission path by writing into a specific register of HardORB-Client. Data received from remote server enters the systems through the receive path of HardORB and written in the DDR memory by the *Writer Controller*.

The HardORB has a pipelined implementation, which is used to simultaneously processes data on various layers of the transmission chain, without blocking the arrival of new data.

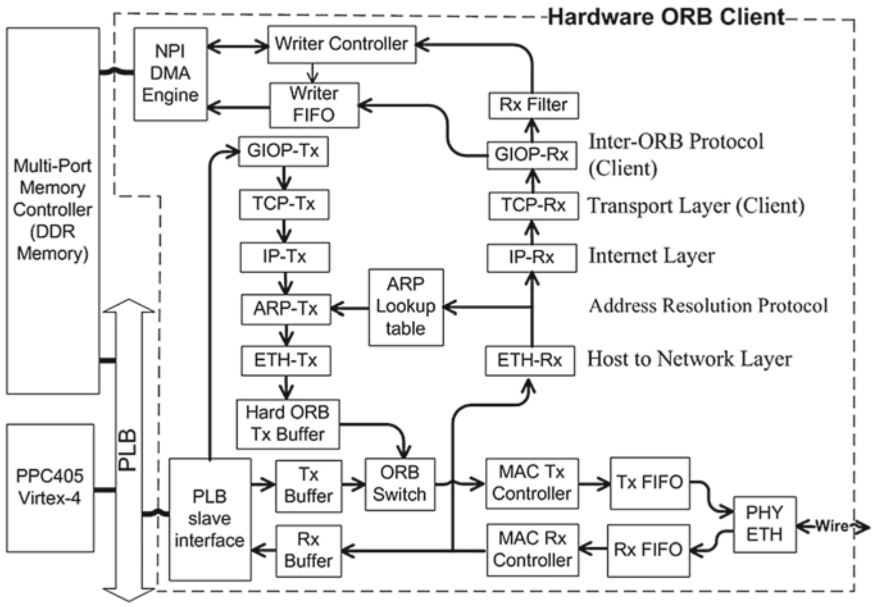


Fig. 3.14 The architecture of our developed Hardware ORB Client IP core

3.9 Evaluation of Communication Performance

3.9.1 Experimental Setup

Performance measurements show the efficiency of the HardORB compared to a pure Software ORB running on embedded processor. The measurements were performed with the communication configurations listed below:

(1) GPP(Client) to embedded processor on Smart Camera(Server).
(2) GPP(Client) to HardORB on Smart Camera(Server).
(3) HardORB on the first Smart Camera(client) to HardORB on the second smart camera(server).

The PICSy smart camera system is used as embedded hardware platform, featuring a capture module, a coordinator module, a communication module.

The ACE ORB (TAO) [35] is used in this experiment as software ORB (Fig. 3.15b), and compared to our HardORB module (Fig. 3.15a). It is based on the standard OMG CORBA reference model with opensource implementation built on the framework components and patterns provided by adaptive communication environment (ACE) [21]. Both modules were integrated into a system using OpenCV for computer vision on the PICSy.

In the first experiment CORBA TAO client and server are executed in software on the smart camera running Linux kernel 2.6.31 and TAO version 1.7.3 as ORB.

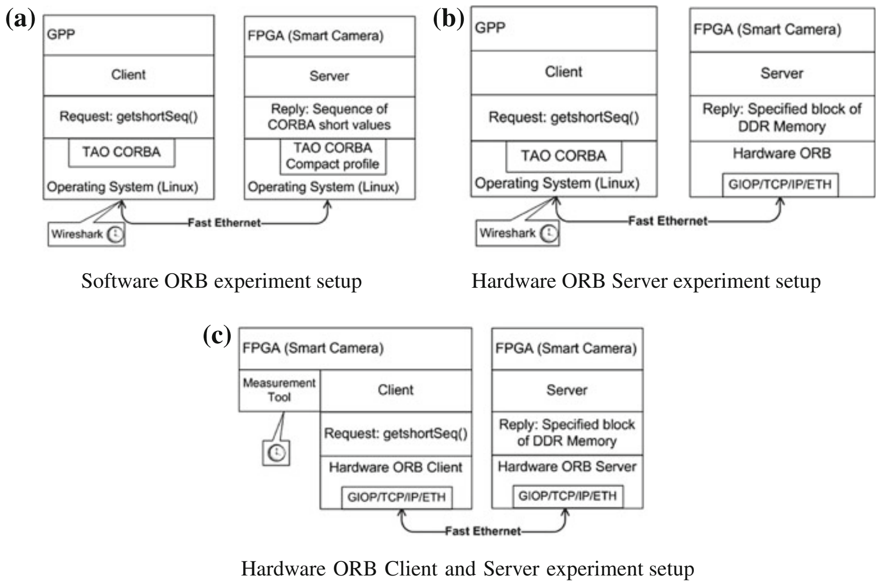


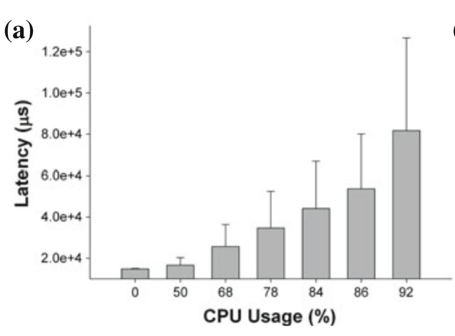
Fig. 3.15 Experimental setup: **a** the pure Software ORB on a personal computer and smart camera, **b** software ORB on a PC and HardORB in the smart camera, and **c** HardORB in two communicating smart cameras

The smart camera is connected to a workstation, which mimics a second smart camera. The processor is an Intel Core Duo CPU T7500 with 2.20GHz running Ubuntu Linux release 9.04 with kernel 2.6.28 and TAO version 1.7.4 as ORB. In both system, the communication architecture is pure software-based. In the second experiment, the workstation still mimic a smart camera, but the PICSy smart camera now implements the HardORB.

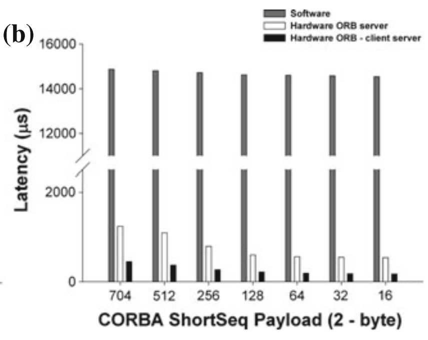
In a typical collaborative application, the TAO would be responsible for exchanging objects extracted from images by OpenCV between two nodes. OpenCV copies the processed information in a predefined blocks of memory. The TAO server views these blocks as CORBA Servant Objects (sequence of data defined in CORBA), serializes them and send them to the client.

Finally, the third benchmark (Fig. 3.15c) presents a realistic interconnection of two smart camera systems with fully utilization of HardORBs client and server inside.

In all test scenarios, the measurements have been achieved with a CORBA client invoking the operation getshortSeq() on a remote CORBA Servant to request a short sequence value as a result of the operation. In the first and second benchmarks, the latency was measured using Wireshark network packet analyzer [38]. The invocation procedure is repeated 1,000 times in order to get a statistically valid average and standard deviation from the obtained values. As the HardORB client and servant are used in the third test scenario. An auxiliary Measurement Tool is used to count in the



Effect of CPU load on software ORB latency.



Comparison of latencies between software and hardware ORBsp

Fig. 3.16 Performance measurements: **a** shows how the CPU loads affects the communication latency, while **b** shows the performance comparison between hardware and software ORB implementations

client node for a precise measurement of the number of clock cycles between start and end of transaction.

The network nodes were connected to each other back-to-back without any other interfaces, and, to avoid interference from other Ethernet traffic, the whole system was isolated from the internet during the tests.

In all benchmarks, the whole progress was repeated for different values of the ShortSequence parameter length, doubling the payload size starting from 32 bytes up to 1,408 bytes.

In the first case tests were performed with various processor loads. Several processes for computing the combination of contour and histogram of incoming video streams executed simultaneously to the ORB software exchanging information. The tests were preformed with the network interface speed of 100 Mbps.

3.9.2 Performance Measurement

The performance measurements with the three described scenarios are depicted in Fig. 3.16a, b.

Figure 3.16 shows that various embedded processor loads have a great impact on the non-deterministic behavior of the software ORB. In contrary, hardware ORB treats with incoming network traffic in a deterministic way.

The HardORB server (Fig. 3.16b) reduces the network latency nearly 10 times toward the software ORB. Figure 3.16b shows that the relative performance gained is nearly 30 times better than software approach. The software ORB with highly non-deterministic behavior is not appropriate solutions in distributed smart camera systems with real-time requirements.

3.10 Conclusion

Several authors have shown the efficiency of using FPGA for image and video processing, mostly on evaluation platforms. The use of FPGA for video processing in embedded environment such as smart cameras however requires many other factors to be considered. Besides the performance and flexibility available on the FPGAs, programmability remains one of the key factors for a broad use of those systems. Our goal in this work was not to necessary to develop image processing algorithms since decades of knowledge and developments in this area has flowed in available image and video processing frameworks on the market. Porting the open source framework OpenCV to the platform was therefore our best choice to increase the acceptance of the platform in the broad community of image processing research. We provided the required interfaces and skeletons needed for the integration of hardware accelerators in the system. Currently a dozen of hardware accelerators have been developed for processing at different levels and integrated into the development framework of OpenCV. We expect more to be done in the future in order to provide a solid alternative to the Intel optimization library for OpenCV. The structuration of the hardware to allow run-time partial reconfiguration.

3.11 Impact of Smart Camera Networks in Self-Organization Research

Increasing complexity due to rapid progress in information technology is making systems more and more difficult to integrate and control. Due to the large amount of possible configurations and alternative design decisions, the integration of components from different manufacturers in a working system cannot be done only at design-time any more. Furthermore the miniaturization of systems makes them more vulnerable to errors that may occur due to physical degradation, cosmic radiation, unpredictable interconnect delay on signals within chips.

Systems must be designed to cope with unexpected run-time environmental changes and interactions. They must be able to organize themselves to adapt to change and avoid non desirable or destructive behaviour. Self-organization (SO) is viewed as a means to cope with uncertainty and design of robust systems.

Self-organization is formally defined as a process in which pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying the interactions among the system's components are executed using only local information, without reference to the global pattern.

Self-organization in computing system is a very old matter of investigation. Several optimization algorithms like simulated annealing [24], genetic algorithms [15], particle swarm optimization [36] and ant algorithms [12] were developed in the past according to optimization principles available in the nature. The concept of swarm intelligence [5] was then invented to name such methods based on the behavior of natural systems, where a large number of single and interacting entities contribute locally to a global pattern. A detailed study on the emergence as well as the benefits and expectation in computing system was done in [7]. In [32], the causal architecture of a process is use to provide the best answer to its organization. The *computational mechanics*, represented by a mathematical object called the ε-machine is used as method for inferring causal architecture of a given process and for pattern discovery. The computational mechanics is then developed for increasingly sophisticated types of process: *memory less transducers*, *time series*, *transducers with memory*, and *cellular automata*.

Besides the previous development mostly limited to algorithm optimization, SO was employed in a more coarse grained manner to control systems. Multi-agent systems [13, 31] for instance was developed as a means for a coordinate and decentralize used of distributed computing resources. The approach of having independent and interacting agents can further be used to solve problems, which are difficult or impossible for an individual agent or monolithic system to solve. Examples of such problems include on-line trading, disaster response, and modelling social structures.

Recently the concept of autonomic computing [16, 37] were introduced at IBM-research to describe the ability of systems to be more self-managing. The overall objective of IBM was to designed software able to self-administrating them self in order to avoid error prone and time consuming configuration that arise when the administration is performed by humans. Meanwhile IBM claims to have few dozens of products behaving according to the autonomic principles.

Despite some experiments where the principles of self-organization where experienced in technical systems in particular in robotics [2, 5, 18, 22] and distributed manufacturing systems [4, 25, 33], the concepts previously described are mostly use to improve the functionality of software. Real-life examples that demonstrate the use of the self-organization properties in embedded systems are lacking. The purpose of this work is to overcome this deficit by designing a real-life embedded system according to the principles and gained sufficient experiences to guide us in the generic design of such kind of systems.

Smart Camera Network could provide a platform for a dual experimental/formal investigation of distributed cognitive systems. Based on a testbed, which is a distributed video-based surveillance system consisting of a set of rotating intelligent cameras, each of which is fixed at different locations or mounted on vehicles, all the requirements and performance parameters of such systems will be derived in order to perform a systematic design later. For the design automation of such distributed vision-based cognitive systems, a design framework can be derived based on the experiences acquired on the testbed. Such a testbed must rely on very efficient image understanding algorithms and architectures that was investigate in this work.

References

1. Aghajan H, Cavallaro A (2009) Multi-camera networks: principles and applications. Academic Press, Burlington, MA
2. Anderson J, Baltes J (2006) An agent-based approach to introductory robotics using robotic soccer. Int J Robot Autom 21(2):141–152
3. Blackham B (2006) The development of a hardware platform for real-time image processing. The University of Western Australia, Perth
4. Bohnenberger T, Fischer K, Gerber C (1999) Agents in manufacturing: online scheduling and production plant configuration. In: ASAMA '99: proceedings of the first international symposium on agent systems and applications third international symposium on mobile agents, p 66. IEEE Computer Society, Washington, DC, USA
5. Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm intelligence: from natural to artificial systems. Oxford University Press Inc, New York, USA. http://portal.acm.org/citation.cfmid=328320
6. Bramberger M, Doblander A, Maier A, Rinner B, Schwabach H (2006) Distributed embedded smart cameras for surveillance applications. IEEE Comput Soc 39:68–75
7. Brunner KA (2002) What's emergent in emergent computing? In: Trappl R (ed) Cybernetics and systems 2002: proceedings of the 16th European meeting on cybernetics and systems research, vol 1, pp 189–192
8. Bobda C, Majer M, Ahmadinia A, Haller T, Linarth A, Teich J (2005) The erlangen slot machine: a highly flexible FPGA-based reconfigurable platform. In: FCCM2005, pp 319–320
9. Casalino F, Middioni G, Paniscotti D (2008) Experience report on the use of corba as the sole middleware solution in sca-based sdr environments. In: SDR forum technical conference (2008)
10. Chen P, Ahammad P, Boyer C, Huang SI, Lin L, Lobaton E, Meingast M, Oh S, Wang S, Yan P, Yang AY, Yeo C, Chang LC, Tygar D, Sastry SS (2008) Citric: a low-bandwidth wireless camera network platform. In: Second ACM/IEEE international conference on distributed smart cameras, pp 1–10
11. Doblander A, Zoufal A, Rinner B (2009) A novel software framework for embedded multiprocessor smart cameras. ACM Trans Embed Comput Syst (TECS) 8(3):24
12. Dorigo M, Stützle T (2004) Ant colony optimization (Bradford books). The MIT Press. http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0262042193
13. Franklin S, Graesser A (1997) Is it an agent, or just a program?: a taxonomy for autonomous agents. In: ECAI '96: proceedings of the workshop on intelligent agents III., Agent theories, architectures, and languages, pp 21–35. Springer, London, UK
14. Gailliard G, Balp H, Sarlotte M, Verdier F (2008) Mapping semantics of corba idl and giop to open core protocol for portability and interoperability of sdr waveform components. In: DATE '08: proceedings of the conference on design, automation and test in, Europe, pp 330–335
15. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading, MA
16. Hariri S, Khargharia B, Chen H, Yang J, Zhang Y, Parashar M, Liu H (2006) The autonomic computing paradigm. Cluster Comput 9(1):5–17. http://dx.doi.org/10.1007/s10586-006-4893-0
17. Hengstler S, Prashanth D, Fong S, Aghajan H (2007) Mesheye: a hybrid-resolution smart camera mote for applications in distributed intelligent surveillance. In: IPSN'07, pp 360–369
18. Hoeing M, Dasgupta P, Petrov P, O'Hara S (2007) Auction-based multi-robot task allocation in comstar. In: AAMAS '07: proceedings of the 6th international joint conference on autonomous agents and multiagent systems, pp 1–8. ACM, New York. http://doi.acm.org/10.1145/1329125.1329462
19. Horprasert T, Harwood D, Davis LS (1999) A statistical approach for real-time robust background subtraction and shadow detection. In: ICCV Frame-Rate WS
20. Humcke F (2006) Making fpgas first class sca citizens. In: SDR forum technical conference

21. Huston SD, Johnson JC, Syyid U (2004) The ACE programmer's guide: practical design patterns for network and systems programming. Addison-Wesley, Reading, MA
22. Kaetsu H (1995) Cooperation between the human operator and the multi-agent robotic system: evaluation of agent monitoring methods for the human interface system. In: IROS '95: proceedings of the international conference on intelligent robots and systems, vol 1, p 206. IEEE Computer Society, Washington, DC, USA
23. Kim K, Medioni GG (2008) Distributed visual processing for a home visual sensor network. In: IEEE workshop on applications of computer vision, pp 1–6. http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/4539769/4543991/04544043.pdf?arnumber=4544043
24. Kirkpatrick S, Gelatt CD Jr, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680
25. Maione G, Naso D (2004) Modelling adaptive multi-agent manufacturing control with discrete event system formalism. Intern J Syst Sci 35(10):591–614. http://dx.doi.org/10.1080/00207720412331297947
26. Mühlbauer F, Rech LOM, Bobda C (2008) Hardware accelerated openCV on system on chip. In: Reconfigurable communication-centric systems-on-chip workshop 2008
27. Pham D, Alcock R (2003) Smart vision applications. In: Smart inspection systems, pp 157–191. Academic Press, London. doi:10.1016/B978-012554157-2/50006-7
28. Prytz G, Johannessen S (2005) Real-time performance measurements using udp on windows and linux. In: 10th IEEE conference on emerging technologies and factory automation 2
29. for blind review, O (2009)
30. Rinner B, Quaritsch M (2008) Embedded middleware for smart camera networks and sensor fusion. Elsevier, Amsterdam
31. Rosenschein JS (1986) Rational interaction: cooperation among intelligent agents. Ph.D. thesis, Stanford University, Stanford, CA, USA
32. Shalizi CR (2001) Causal architecture, complexity and self-organization in time series and cellular automata. Ph.D. thesis, The University of Wisconsin—Madison. Supervisor-Olsson, Martin
33. Shen W, Norrie DH (2001) Dynamic manufacturing scheduling using both functional and resource related agents. Integr Comput-Aided Eng 8(1):17–30
34. Shi Y, Tsui T (2007) An fpga-based smart camera for gesture recognition in hci applications. ACCV 1:718–727
35. TAO http://www.cs.wustl.edu/schmidt/tao-status.html
36. Venter G, Sobieszczanski-Sobieski J (2003) Particle swarm optimization. AIAA J 41(8):1583–1589
37. White SR, Hanson JE, Whalley I, Chess DM, Segal A, Kephart JO (2006) Autonomic computing: architectural approach and prototype. Integr Comput-Aided Eng 13(2):173–188
38. Wireshark http://www.wireshark.org/
39. Xilinx (2008) Early access partial reconfiguration user guide

Chapter 4
Design and Verification Environment for High-Performance Video-Based Embedded Systems

Michael Mefenza, Franck Yonga and Christophe Bobda

Abstract In this chapter, we propose a design and verification environment for computational demanding and secure embedded vision-based systems. Starting with an executable specification in OpenCV, we provide subsequent refinements and verification down to a system-on-chip prototype into an FPGA-based smart camera. At each level of abstraction, properties of image processing applications are used along with structure composition to provide a generic architecture that can be automatically verified and mapped to a lower abstraction level, the last of which being the FPGA. The result of this design flow is a framework that encapsulates the computer vision library OpenCV at the highest level, integrates Accelera's SystemC/TLM with the Universal Verification Methodology (UVM) and QEMU-OS for virtual prototyping, verification, and low-level mapping.

4.1 Introduction

Due to their advantages in size, cost, and programmability, embedded cameras have become a promising alternative to conventional sensors such as RADAR and LIDAR for gathering information about the surrounding environment. Robotics, driving assistance systems, autonomous driving cars, and unmanned aerial systems (UAS) are just few examples of application areas where embedded cameras have been successfully introduced. Many studies in the past have aimed at designing complex visual

M. Mefenza (✉) · F. Yonga · C. Bobda
Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR, USA
e-mail: mmefenza@uark.edu

F. Yonga
e-mail: yfrancku@uark.edu

C. Bobda
e-mail: cbobda@uark.edu

systems using embedded cameras for vision-aided flight control [10], tracking [13], terrain mapping [16], and navigation [11], with considerable achievements.

Despite their huge potential, designing video-based embedded systems remains challenging due to the many and sometimes conflicting design constraints that must be simultaneously addressed: high performance, high throughput, low power, or low density. This complexity is expected to increase in the near future as consumers and applications demand more functionality and performance. Nevertheless, with the rapid growing in the field of high-performance and reconfigurable computing, computational requirements of such video-based platforms could efficiently be satisfied through a combination of hardware and software on a single chip, also referred as system-on-chip (SoC). In such a decomposition, low-level repetitive computations are mapped into hardware, while complex reasoning parts are maintained in software.

The design process of a hardware/software system generally starts with a manual partition of the initial application, in which parallelizable and computational intensive blocks are extracted. Then follows a separate implementation of the sequential parts in software—using traditional languages such as C—and the parallelizable parts in hardware—using hardware description languages such as Verilog or VHDL. This segmentation is a tedious process that requires hardware and software skills as well as a deep understanding of image processing. Software design environments, such as OpenCV, are very popular in the software community for the design of video-based systems. While such frameworks increase the productivity by providing a rich set of library functions for image, video manipulation, and machine learning, they are limited to target only general purpose processors. As a consequence, there is a need to map applications captured in those frameworks onto dedicated hardware/software architectures while performing verification tasks. Manual translations are time consuming, error-prone, and require hardware design skills not available in the image processing community, the bulk of which is made up of software designers.

In this chapter, the focus is on providing a high-level framework for the rapid design and verification of embedded vision-based systems. The proposed work exploits high-level system modeling to define concepts necessary for capturing vision-based applications through the use of SystemC/TLM. SystemC/TLM has become the defacto standard for systems-on-chip design and modeling. It is widely used for development at higher abstraction levels and for system prototyping. After the high-level specification, model to model transformation rules are developed to allow subsequent refinements down to a hardware/software implementation. Finally, verification and analysis models are used to insure the correctness of the final design regarding the initial system specification.

4.2 Related Work

In this section we review the related work pertaining to the design and verification of video-based embedded systems. Published literature could be classified in two categories: hardware/software system design and verification environments.

4.2.1 Hardware/Software System Design

The review of embedded video processing systems provided in [19] shows that computation in current embedded systems is performed in software on a general purpose processor, sometimes optimized for multimedia computation. The system in [8] uses several digital signal processors on different PCI-boards, while the CITRIC [9] relies on the Intel XScale PXA270 processor. Considering the growing complexity of applications, the computational requirements of embedded cameras can be reached only through a combination of hardware and software, which are integrated today on a single system-on-chip. Usually, hardware and software are developed separately and the integration is done very late. Bugs become difficult to correct and hardware architectures are sometimes not optimal. Because of the high cost and reduce flexibility that will result in the design of a ASIC, this aspect is tackled with FPGAs in most embedded systems such as [5, 7]. However, the FPGAs were used as co-processors for a single task fixed at compile-time. For instance, in [7], a Xilinx Spartan-3E FPGA is used as a co-processor. In [23], a hardware/software implementation on a Xilinx FPGA platform is presented for a 3D facial pose tracking application; the most-computationally intensive part was implemented in hardware while the remaining were implemented in the soft-core processors. Various architectures employ programmable DSPs with additional resources, such as special graphics controllers and reconfigurable logic devices, as shown in [21]. These implementations are done manually using low-level languages, and there is no focus on verifying the resulting architectures. This approach is tedious and error- prone as we mentioned.

4.2.2 Verification Environments

Verification for embedded imaging systems is addressed in [3, 25], where a generic and automatic environment for the verification of image signal processing IPs is presented. In [25], the verification environment is based on Specman language, while [3] uses reusable eVC and C-Model. These works however are limited to hardware accelerators. In [18], a specific application system with the co-simulation of SystemC and RTL is presented. The co-simulation of SystemC TLM is used in a surveillance camera system using a one-bit motion detection algorithm for portable applications. The host controller interface (HCI) and the motion detection sensor (MDS) are implemented using SystemC TLM. The API program is implemented in C++, while other blocks of this system are implemented using RTL HDL. To verify co-simulation, HCI, MDS, and API program are operated at a PC workstation. Co-simulation is used to accelerate the simulation of the system. In [24] video data and synchronization signals are generated as testbench for the simulation of video processing IPs. However, simulation without emulation is limited since it does not catch every error in the RTL systems, especially timing errors. The scope in [22] is limited to algorithms by formally verifying complex loop transformations like loop folding, loop distribution, typically applied in the design trajectory of data dominated signal and data processing applications. Important part of the systems like interfacing, communication and data flow component, and memory are not addressed.

None of the existing work does address the high-level system design and system verification in tandem. The goal of the framework presented in this chapter is to provide software designers with a mean to capture their designs at a high abstraction level and subsequently refine their implementation down to an equivalent hardware implementation, with the final design being emulated into the FPGA. At each transformation step, the framework will ensure design safety, reliability, and correctness through a verification process based on the Universal Verification Methodology (UVM).

4.3 The Design and Verification Approach

For the automatic design and verification of embedded video systems, we propose the design approach sketched in Fig. 4.1. In the proposed flow, the initial specification undergoes several transformations. During this process, it must be guaranteed that a transformation does not produce different results than the original specification. Therefore, at each level of the design flow, the corresponding abstract representation of the target architecture is used with the required amount of details to verify and refine the representation to the lower level by means of structure composition and structure verification, explained in Sect. 4.3.1.2. As depicted in Fig. 4.1, the proposed design flow consist in four successive phases: input system specification, Hardware/Software decomposition, RTL implementation, and system emulation. Before explaining these steps, concepts and terminologies that are used throughout this chapter to present the proposed approach will be defined.

4.3.1 Definitions

4.3.1.1 Component Interconnect for Data Access

The Component Interconnect for Data Access (CIDA) is a portable interface module used for data exchange among software and hardware components in a system-on-chip. It is classified as a rich interface [6], which means that it could be used to specify the protocol aspects of component interaction. CIDA helps to model and to investigate component-based systems by strictly separating the behavioral description of components from their interaction.

Components are generally designed independently from their context of use and could be connected together via composition. In computer vision systems, communication between the different parallel components is very important. Consequently, streams of data have to be carefully designed in order to maximize the use of parallelization. Moreover, memory interfaces play a key role in data transfer because

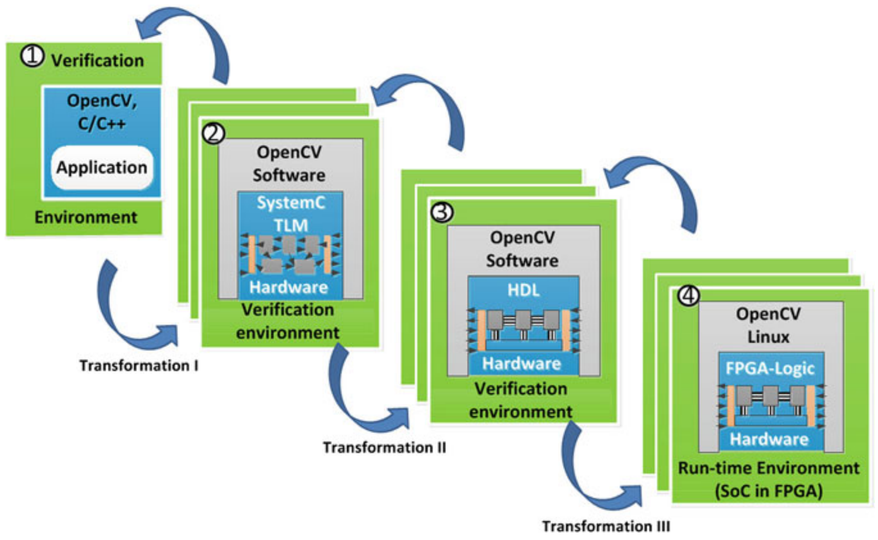


Fig. 4.1 Design flow for the automatic transformation and verification of a high-level video-based application into an RTL representation

of the large memory access requirements. CIDA is very useful in image processing where computation takes place in a streaming way. It has been designed to allow efficient data streaming among hardware accelerators and memories. CIDA consists of two disjoint sets:

- X^I set of inputs, composed of generic $stream\_in$ ports and memory-mapped inputs ports. The generic $stream\_in$ ports are:
  - $Data\_in < m >$: input payload of size m
  - $valid\_in$: valid payload at the input
  - $stop\_in$: request to continue to feed data at the output. This is an active low signal.
- X^O set of outputs, composed of generic $stream\_out$ ports and memory-mapped output ports. The generic $stream\_out$ ports are:
  - $Data\_out < n >$: output payload of size n
  - $valid\_out$: valid payload at the output
  - $stop\_out$: request to continue to feed data at the input. It is an active low signal.

Figure 4.2 shows the internal structure of CIDA. CIDA is a powerful interface to stream data between hardware modules. It consists of 2 sub-interface modules (stream-in and stream-out), a set of memory mapped signals, and a scheduler, easily implementable in SystemC/TLM. The AXI Interconnect of Xilinx [2] is used as the bus protocol to allow Direct Memory Acess (DMA) to the external memory. All Processing Units (PU) connected to the CIDA access the memory (for read and

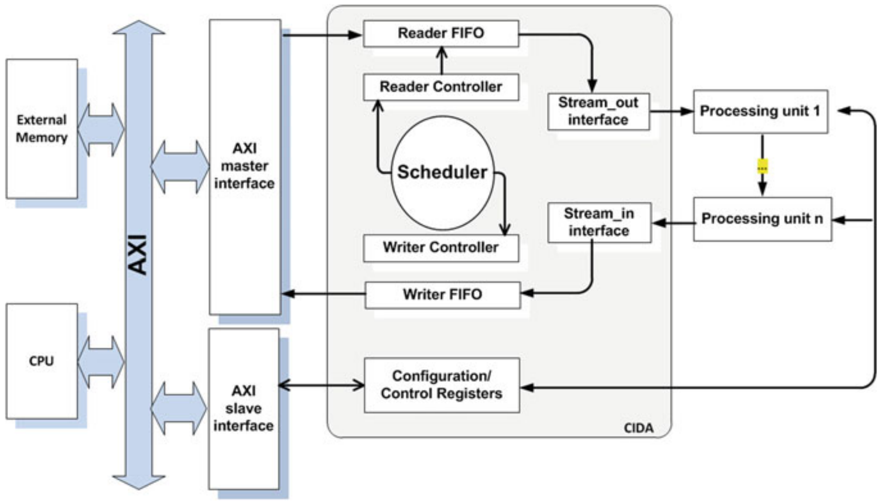


Fig. 4.2 Internal architecture of the CIDA

write operations) through an AXI master interface capable of burst transfers. CIDA leverages an internal scheduler to arbitrate the transactions between its streaming interfaces and the external memory. This scheduler locally handles concurrent read and write requests between the memory and the PUs attached to the CIDA interface. The scheduling policy currently used is based on Round-robin, but more complex scheduling mechanisms could be implemented. A data source, such as the CPU or the memory, can supply or receive pixels to/from a processing unit in a composition chain. The computation take place inside PUs and the results are sent either to the next PU in the chain or back to the data source. The state and the configuration of a PU is available to the CPU through software defined registers accessible via an AXI slave interface.

Figure 4.3 presents an example of processing chain where all functionalities of CIDA are used. The presented structure is made up of *n* PUs interconnected using CIDA and that can anytime access external memories for read and write operations. Here, an external scheduler arbitrates memory requests coming from multiples PUs through the CIDA components. By using CIDA for component interconnection, we can define transformation rules and perform an automatic synthesis from the high-level down to the hardware implementation, while insuring the correctness of resulting design.

4.3.1.2 Interface-Based Structure Composition

Structure composition consists of putting low-level configurable modules together in order to build high-level functions. For this purpose, CIDA is leveraged for the seamless integration of IPs together in the final architecture with transparent data

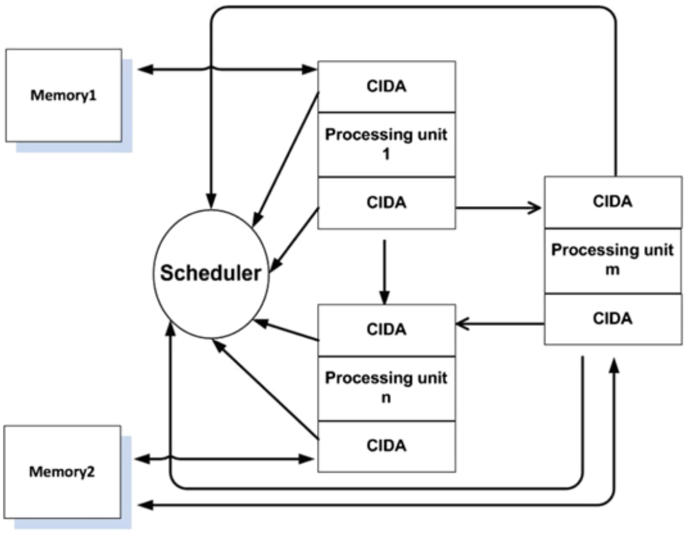


Fig. 4.3 Example of processing chain using CIDA

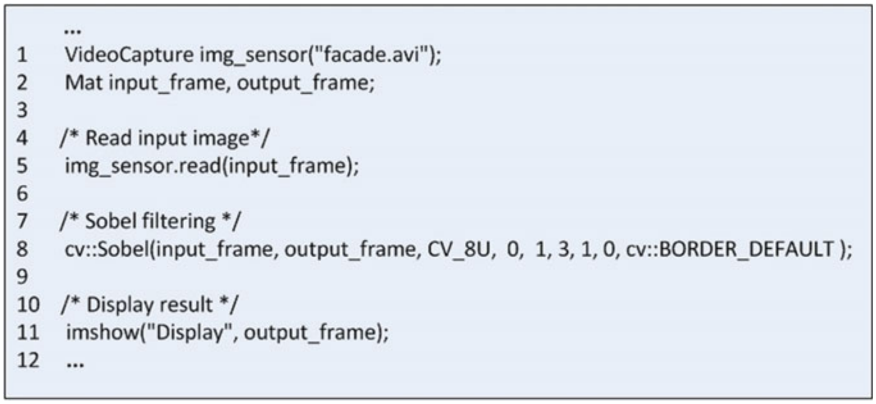
exchange in the system. The notion of composition can be explained by defining A/G (Assume/guarantee) interfaces. An A/G interface F = X^I, X^O, ϕ^I, ϕ^O consists of:

- two disjoint sets X^I and X^O of input and output variables;
- a satisfiable predicate ϕ^I over X^I called input assumption;
- a satisfiable predicate ϕ^O over X^O called output guarantee.

Two A/G interfaces F and G are said to be *syntactically composable*, iff the set of their output variables are disjoint: $X_F^O \bigcap X_G^O = \emptyset$. In our case, two CIDA interfaces will always be composable due to the fact that an input signal cannot have multiple drivers, which is a constraint for synthesizable designs.

4.3.1.3 Structure Verification

The main idea behind Structure Verification is to be able to verify that an abstract interface design meets a requirement specification. This is insured by verifying each individual element in the composition chain and each interface used. Using verification methodologies such as assertion-based or coverage-driven, we can prove that a specification satisfies a given property. Coverage metrics are measures of the exhaustiveness of a test suite. They measure the quality of verification effort, to which extends the functionality of a given design under test (DUT) has been verified during simulation. On the other hand, Assertion-Based Verification (ABV) has proven to enhance design quality and verification time tremendously [1].



```
     ...
1    VideoCapture img_sensor("facade.avi");
2    Mat input_frame, output_frame;
3
4   /* Read input image*/
5    img_sensor.read(input_frame);
6
7   /* Sobel filtering */
8    cv::Sobel(input_frame, output_frame, CV_8U,  0,  1, 3, 1, 0, cv::BORDER_DEFAULT );
9
10  /* Display result */
11   imshow("Display", output_frame);
12   ...
```

Fig. 4.4 Description of the Sobel filtering in software. Video reading, Sobel processing, and video displaying are implemented using the OpenCV library

4.3.2 Level 1: System Specification

The first step in the design flow is the specification of the system which corresponds to the description of the high-level input application, regardless of the target architectures. At this stage, only image processing skills are required to define the applications in executable form in C/C++ using the OpenCV environment. To feed input data to the application, synthetic videos or a webcam could be used, while any monitor screen maybe used to visualize the output results. An example of specification is shown in Fig. 4.4 which describes a Sobel filter image processing. The OpenCV library is leveraged to read data from a video file (line 25), then to process the incoming pixels framewise (line 28), and finally to display the result on the output monitor (line 31). The verification at this level is done by means of high-level simulation, just to ensure that the application is running correctly. Simulation results will be saved and used as reference values during the next refinement steps in order to verify that the corresponding transformations are correct.

4.3.3 Level 2: High-level Hardware/Software Decomposition

From the application specification in OpenCV, a hardware/software partition is produced. This decomposition aims to improve the application performance by moving computational intensive parts of the system to the hardware, while keeping the complex and reasoning parts in software. At this step, the behavior of the entire system is captured using a transaction-level model. Transaction Level Modeling (TLM) [20] is becoming increasingly popular in the industry as the ultimate tool to capture and verify systems consisting of several software processes and hardware components.

TLM is particularly appealing because of its compatibility with environments that use native C/C++.

Given that there is currently no tool for the automatic partition of the design, the user is in charge of selecting functions to be mapped onto the hardware. This could be done by using a profiling-based approach, where computational "hot spots" in the OpenCV application are identified after simulation. At this level of the design flow, we do not focus on implementation details of the communication among hardware and software blocks. OpenCV and SystemC are combined in such a way that all functionalities to provide images, to perform reasoning on extracted features, and to display results are handled using OpenCV, while SystemC is used to describe the abstract communication among all the different blocks in the system. To generate the equivalent SystemC-RTL implementation from the initial OpenCV application, two successive transformations are required.

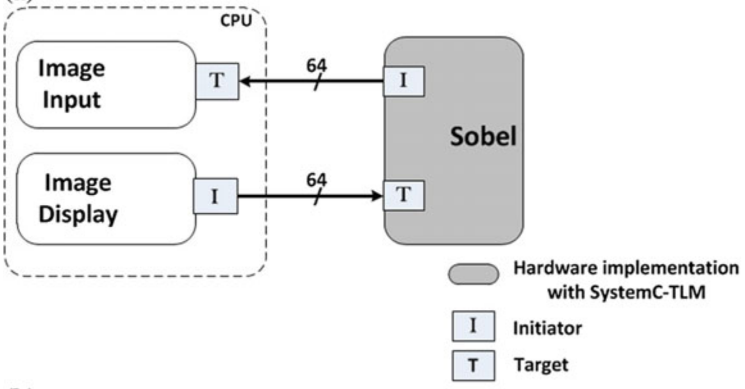
4.3.3.1 Transformation I: OpenCV → OpenCV/SystemC-TLM

The objective in the first transformation is to generate from the initial OpenCV application, an equivalent hardware/software decomposition, with the hardware being implemented using SystemC and the component communication being captured through Transaction-Level Modeling. This partitioning aims at increasing system performance and is done manually after a profiling of computationally intensive parts of the input specification.

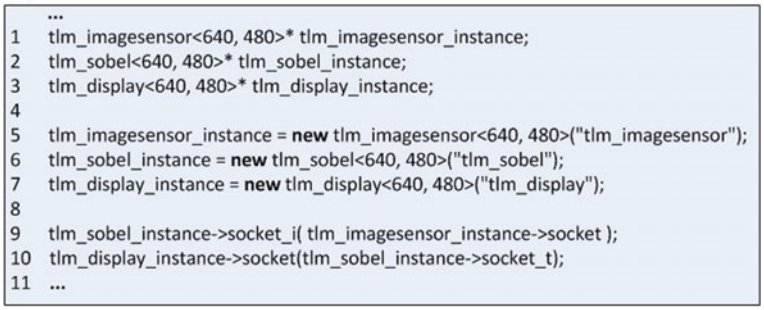
Transaction-Level Modeling (TLM) is a high-level approach to modeling systems where details of communication among modules are separated from the details of the actual implementation of the communication architecture. Communication mechanisms such as buses or FIFOs are modeled with TLM as channels and connected to other modules using SystemC interface classes. In this chapter, we will refer to such a class as a SystemC-TLM implementation. At the transaction level, the emphasis is more on the functionality of the data transfers—what data are being transferred and from what location—and less on their actual implementation; that is, on the actual protocol used for data transfer. A transaction is a data structure passed between initiators and targets using function calls. In TLM-2.0, an initiator is a module that initiates new transactions, and a target, a module that responds to transactions initiated by other modules. A transaction is characterized by its type (read or write), an address from where to read or write, a pointer to the data to be transferred, and the phase of the transaction which indicates the progress status (begin, request, end request, completed, ...). The same module can act both as an initiator and as a target. This is the case for a model of an arbiter, a router, or a bus, where transactions are usually bidirectional.

In the first transformation, the main challenge is to be able to successfully identify the hardware candidates during the profiling. Many computer vision operations, such as convolutions, filtering operations, or color conversions work on a pixel-by-pixel, windows, or matrix basis, and represent excellent candidates for parallel or pipelined processing. Given that the structure of such operations is well-known, we provided

(a)



(b)



```
   ...
1  tlm_imagesensor<640, 480>* tlm_imagesensor_instance;
2  tlm_sobel<640, 480>* tlm_sobel_instance;
3  tlm_display<640, 480>* tlm_display_instance;
4
5  tlm_imagesensor_instance = new tlm_imagesensor<640, 480>("tlm_imagesensor");
6  tlm_sobel_instance = new tlm_sobel<640, 480>("tlm_sobel");
7  tlm_display_instance = new tlm_display<640, 480>("tlm_display");
8
9  tlm_sobel_instance->socket_i( tlm_imagesensor_instance->socket );
10 tlm_display_instance->socket(tlm_sobel_instance->socket_t);
11 ...
```

Fig. 4.5 HW/SW decomposition of the Sobel processing using OpenCV and SystemC-TLM. **a** The Blockwise decomposition. **b** High-level implementation

a set of template implementations in SystemC-TLM, SytemC-RTL, and RTL where the user can directly select from in order to speed up the design process.

Upon completion of the profiling, the identified blocks (hardware and software) are first combined to build a composition chain, reflecting the streaming structure of an image processing. Then, all hardware candidates are replaced with their SystemC-TLM implementations. TLM sockets (initiator and target) are also integrated to the software blocks to allow transactions between hardware and software modules. Finally, the correctness of the transformation is verified using the UVM. Figure 4.5 sketches an example of such a transformation applied on the Sobel filter processing of Fig. 4.4. When evaluating the application of Fig. 4.4, the Sobel processing step is identified as a computational demanding task and will be accelerated in hardware, while the reading and displaying of images will remain in OpenCV and executed on the general purpose processor. The program is therefore segmented in three different blocks: video reading, Sobel processing, and video displaying. For a seamless data transaction between modules, three instances of TLM classes are created (line 5, 6, and 7 of Fig.4.5b). Data transaction in this new composition starts with the display module sending a read request to the target of the Sobel component. If the data buffer of the Sobel module is not empty, then a response is sent back with the requested data as packets of 64 bits. Otherwise, the request transaction will wait for a response from the Sobel without blocking the system (non-blocking transfer), while the initiator of the Sobel module will be issuing a read request to the image sensor module for data acquisition. In this transformation, the focus is not on the implementation details of the transactions between blocks, but rather on the functionality of the data transfers.

The verification at this level consists of either simulating the entire system using SystemC or verifying each of the hardware components using UVM. This later option will require to verify all assertions and coverages at the transaction level.

4.3.3.2 Transformation II: OpenCV/SystemC-TLM → OpenCV/SystemC-RTL

In this step, all SystemC-TLM modules of the previous transformation will be replaced with their SystemC-RTL counterparts. The use of TLM allows for faster simulation and rapid functional verification of the system since it works on transactions. However, it is also important to have a cycle-accurate or semi-accurate model of the hardware/software system in order to verify not only the timing constraints but also the functional coverage and to check more aggressively the protocol inside the parts of the system. We need therefore to rely on a cycle-accurate protocol that could interface with TLM. For this purpose, CIDA could be leveraged to construct the structure composition at the cycle-accurate and RTL level. Equivalence checking will be used to ensure that the system is still equivalent after the transformation. The verification of each structure composition will be guaranteed by verifying both the CIDA interface and each module inside the composition. For the verification of the resulting system, we proposed a generic environment using UVM (see Fig. 4.7) to verify each IP in the composition chain at the SystemC-RTL level, as well as the resulting system-on-chip.

When applying the second transformation to the implementation of Fig. 4.5, the TLM classes of the hardware components are replaced with CIDA interfaces. To allow communication among hardware and software components, a direct memory access (DMA) module is used. The DMA operates as a bridge and integrates both a TLM interface for transactions with the memory and a CIDA interface for communication with hardware components. For a cycle-accurate implementation, data transactions between hardware modules are synchronized. The software blocks communicate with the rest of the system by directly reading or writing data at a specific location inside the memory. Figure 4.6 illustrates such a transformation.

In the configuration of Fig. 4.6, the Image Display module initiates the processing by sending to the DMA (through the memory component) a request to place the processed pixels at a specific location inside the memory. The DMA will then send a read request to the Image Input for pixel acquisition. Once the raw data are read and copied in the memory, they will be transmitted to the Sobel module in packets of 64 bits. Such a data transaction reflects the realistic communication pattern existing between embedded components where the size of the bus is limited; requiring large

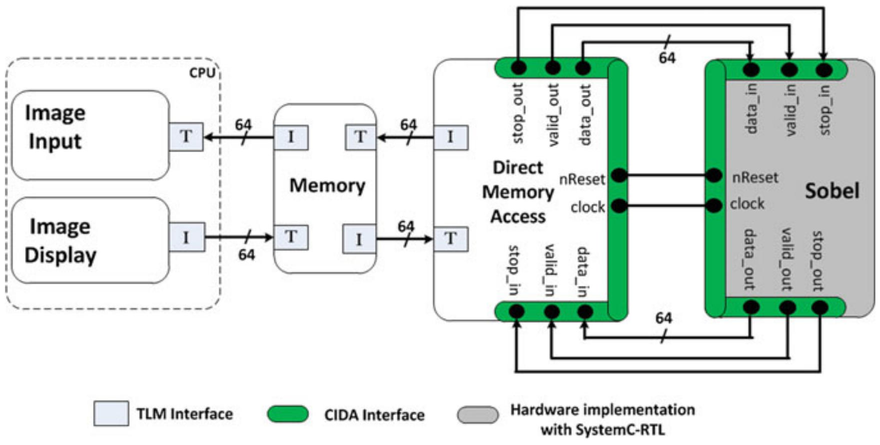


Fig. 4.6 Cycle-accurate modeling of the Sobel processing using SystemC-RTL, TLM classes, and CIDA interfaces

data to be exchanged as small packets. After the processing, the result will be stored back in the memory and the DMA module will notify the processor through an interrupt. Upon reception of the interrupt signal from the DMA, the processed data will be read from the memory and displayed on the monitor.

Figure 4.7 depicts an abstract representation of the generic system-on-chip that is used in this chapter to model a high-level application in the OpenCV/SystemC-RTL environment. The entire environment runs in software on a workstation and is used to emulate the low-level final architecture on our target platform. The input video system is modeled using any traditional webcam and the resulting output will be visualized on a conventional monitor. The refinement from high-level to the RTL implementation is done by means of structure composition. For this purpose, CIDA and TLM are used for the seamless integration of hardware components into the system and for the transparent data exchange among software and hardware components in the final architecture. We provide a TLM-Specification for the direct memory access (DMA), which uses TLM-Socket to allow for communication among hardware and software components on the chip. The bus is specified as a TLM-Module that accesses a transaction but does act neither as an initiator nor as a target. The ARM target processor is modeled using QEMU, a generic and open source machine emulator and virtualizer capable of running operating systems and programs with good performance. We integrated QEMU in the OpenCV + SystemC-TLM environment using TLMu [4], a TLM wrapper for QEMU that allows to communicate with the CPU core using TLM2.0 sockets. The resulting system (OpenCV + SystemC-TLM + QEMU) is then used to simulate the resul1ting system-on-chip with software and OS at the transaction-level, with inputs and output provided by the OpenCV.

The inter-block communication in the proposed environment is ensured by CIDA interfaces and TLM classes, such that by verifying the CIDA and TLM, we will automatically insure the correctness of the processing chain. The simulation of the

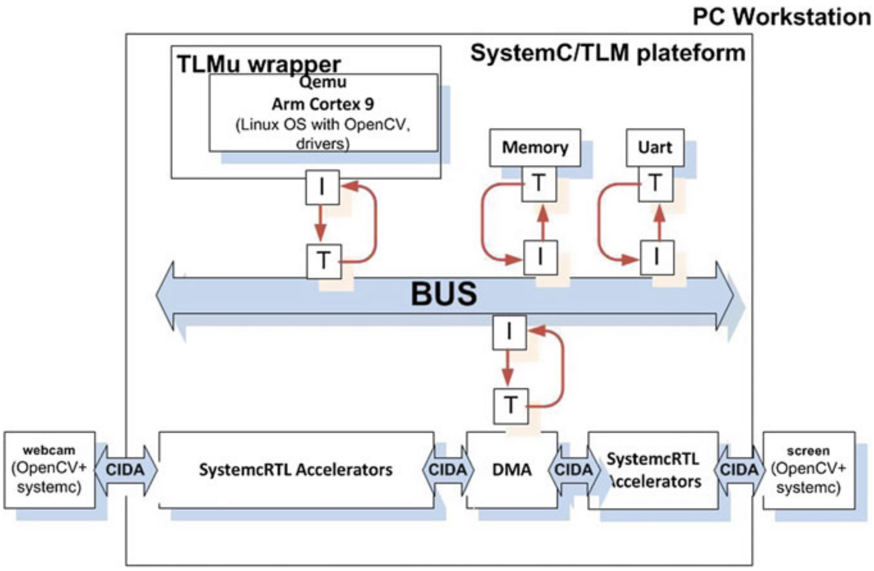


Fig. 4.7 Abstract representation of the target platform at the system-level

system will allow to verify the software and the hardware parts in the verification environment, presented in Sect. 4.3.6. The functional coverage and assertion-based verification of the SystemC-TLM environment is done by integrating UVM into our environment. The UVM environment contains verification intellectual property components. The coverage and assertions are integrated into the environment using SystemVerilog coverage and assertion properties. The processing flow consists of generating sequences into the UVM environment, sending them to the SystemC-TLM module, transferring any event or signal needed to the UVM environment for recording, checking, coverage and assertion analysis.

4.3.4 Level 3: Register Transfer Level Implementation

In this third step, the abstract TLM-description of step two is further transformed into a structure that can be synthesized by hardware compilers. The refinement includes the pin and cycle accurate implementation of the communication interface between software and hardware, a detailed description of the bus model, and a detailed implementation of buffers and memory. In embedded imaging systems, images are transmitted from a charge coupled device (CCD) or CMOS to the processing subsystem pixel after pixel. Furthermore, most image processing functions operate on pixel neighborhoods, which in case of streaming architectures are better served with some form of sliding windows and additional buffers to capture the neighborhood of the pixel currently being considered.

In our environment, we provide a structure consisting of a buffer to hold each line of pixels for the sliding window and a mask to capture the neighborhood of a pixel. The buffers are configurable in length and number to match the size of the picture and neighborhood being used. With such a clear structure for the image processing block, the function applied on the masks is also a template, whose implementation can simply be set automatically from the function specification in TLM and additional user-defined properties. Masks currently available in our environment are: convolutions, generic edge detection, segmentation, and thresholding. In order to translate the SystemC-TLM description into an RTL implementation that can be synthesized, we implemented the templates in VHDL and automatically map the description of level 2 into level 3, without requiring intervention of the user. Transaction parts that rely on object structures are mapped to software along with the part running in OpenCV, while the remaining RTL part of SystemC is mapped to our VHDL module implementation.

4.3.4.1 Transformation III: OpenCV/SystemC-RTL → OS/OpenCV/HW

This step consists of producing the final architecture and to rapidly prototype it in the FPGA to emulate the system in the field. Since there is no efficient tool to perform the translation from SystemC to RTL, this transformation will be done semi-automatically by mapping SystemC cores to pre-designed RTL IPs. The design of CIDA allows any construction in the SystemC platform to have a corresponding construction at the RTL level. Equivalence checking is then used to formally prove that the SystemC model is equivalent to the RTL IP. Since all modules of the environment are not synthesizable, some of them will be replaced by the corresponding resource in the target FPGA. For example the QEMU processor will be replaced by the Processing System in the target FPGA.

4.3.5 Level 4: Emulation

This is the last step in the design process where the final system is emulated. For this purpose, we have designed a versatile FPGA-based smart camera, the RazorCam, to allow for testing in a real-life environments. Our platform implements image processing directly inside the camera, instead of propagating the image to a workstation for processing. Its compact size and performance facilitates the integration in embedded environment like cars and UAS. The processing module consists of: one Xilinx Zynq FPGA, a flash drive, connectors for an infrared camera, a digital camera sensor, and an analog camera sensor. A TFT monitor can be connected to the platform so that the user would check the results of its application in real-time.

4.3.6 The UVM Environment for Verifying SystemC Models

The goal is to propose a generic environment to quickly verify our models after every transformation during the design process. We present in this chapter an automatic UVM environment (Fig. 4.8) with functional coverage and assertion-based verification for SystemC-TLM. The base environment is generated according to the inputs and outputs of the SystemC-TLM systems and contains verification intellectual property components. The result is a complete and working UVM testbench for SystemC-TLM (with all blocks built and connected) that can be compiled and simulated. The processing flow consists of generating sequences into the UVM environment, sending them to SystemC/TLM module using UVM connect from Mentor Graphic. Outputs of the hardware accelerator are sent back to the UVM environment for recording, checking, coverage and assertion analysis. Moreover, the environment is encapsulated with SystemC/TLM module such that it is possible to drive the module with either sequences from the environment or sequences from SystemC. This allows to verify the module either independently or as part of SystemC system-on-chip. The proposed environment reduces steps in the testbench creation while providing a high quality of the verification by using a combination of coverage and assertions. The following section details the capabilities of our environment.

4.3.6.1 Extracting Data

The UVM environment has to drive input ports of a design and observe output ports. A SystemC parser is used to retrieve the netlist (inputs and outputs) of the DUT. The netlist is used to generate one class Packet in SystemC, one class packet in UVM and a SystemVerilog interface. The class packet in UVM is defined as uvm sequence item and is used to generate sequences in the UVM environment. Sequence item represents data for the stimulus of the DUT. The stimulus can represent a command, a bus transaction, or a protocol implemented inside the DUT. The fields in a Sequence item may be randomized to generate different stimuli. UVM provides constructs to generate random and constrained packets. The class Packet in SystemC is used to drive input ports of the DUT and observe output ports of the DUT. The communication between a packet in SystemC and a packet in UVM is done using UVM Connect (UVMC). A similar procedure is applied in case of Transaction Level Modeling (TLM) IP. In TLM, there are 2 types of sockets: initiator and target. An initiator socket generates a transaction and sends it to a target socket. A transaction is characterized by the type of transaction (read or write), the address, a pointer to the data to be transferred and the phase of the transaction. These ccharacteristics are used to create packets for driving a TLM IP.

4.3.6.2 Sequencer

A sequence implements the procedure to create sequence items. It is a set of packets with specific values for each field of a packet.The sequencer is responsible for the

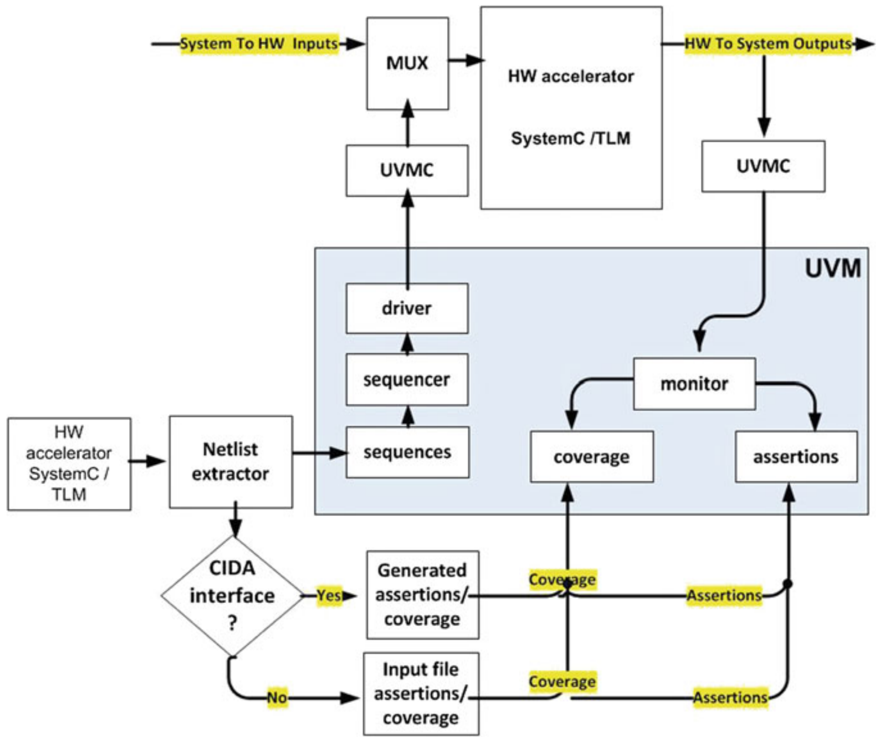


Fig. 4.8 The proposed UVM environment

coordination between sequences and the driver. It reads a packet or transaction from the *sequences* module and sends it to the driver. The driver implements the function *seq_item_port.get_next_item* to indicate to the sequencer when it needs a new packet or transaction. It also implements the function *seq_item_port.item_done* to signal when it has finished to process a packet or a transaction. This mechanism, similar to a blocking transport interface, allows the synchronization between the sequences module and the driver.

4.3.6.3 Driver

This module drives a packet or a transaction to the DUT ports. It receives sequence items and sends them to the DUT. The driver implements a function uvm_blocking _put_port to send a packet through UVM Connect (UVMC) to the SystemC side of the environment. We used a blocking port as a means for synchronization and this provides enough time to the DUT to process the packet.

4.3.6.4 UVM-SC Communication

UVM Connect (UVMC) from Mentor Graphic is an open-source UVM-based library that provides the communication between SystemC Components and SystemVerilog UVM Components using TLM connectivity between them. It also provides a means for accessing and controlling UVM simulation from SystemC. In the proposed environment, the communication from UVM to SystemC is done using *tlm_blocking_put_if* interface. This interface passes a packet from the driver to the SystemC/TLM hardware. *tlm_analysis_port* is used for the communication from SystemC to UVM. It sends a packet from SystemC/TLM hardware to the monitor. The use of TLM connectivity between UVM and SystemC provides a rapid simulation.

4.3.6.5 Monitor

This module receives transactions and signals and makes them available to other components. The communication between the monitor and the coverage module is done using a simple *uvm_analysis_port* function. The analysis port is used to perform non-blocking broadcasts (from one entity to several entities) of transactions from a component to its subscribers. An analysis port can be connected to more than one component. A subscriber component provides an implementation of *uvm_analysis_imp port*. An *uvm_analysis_imp* receives all transactions broadcasted by a *uvm_analysis_port* and implements the analysis interface such as coverage collection. The monitor instantiates a SV interface and maps it to the received transactions and signals. Assertions are checked inside the SV interface.

4.3.6.6 Functional Coverage

The coverage is integrated into the environment using SystemVerilog coverage properties. The SystemVerilog functional coverage constructs allow coverage of variables and expressions, as well as cross coverage between them. In SystemVerilog, the covergroup construct is used to specify a coverage model. Each covergroup can include a set of coverage points, cross coverage between coverage points and coverage options. A coverage point can be a variable or an expression. Each coverage point includes a set of bins associated with its sampled values or its value-transitions. The bins can be explicitly defined by the user or automatically created. For example, a *sc_logic* variable should have only two possible values: 0 or 1. Therefore, the coverage of a sc_logic variable can be done with a bin [0:1]; this can be generated automatically. In the proposed environment, the coverage model is implemented in a class *coverage*, subscriber component for the class *monitor*. We give to users the possibility to specify an external coverage input file to be used to generate the class *coverage*. SystemVerilog syntax must be used to define the coverage inside the file.

The coverage input file is included inside the class *coverage* during the generation of the UVM environment.

4.3.6.7 Assertions

Assertions are integrated into the environment using SystemVerilog assertions and are implemented to execute protocol or timing checking. They can be used to implement simple to-complex property/sequence checks for an interface or a protocol. Since in SystemVerilog, assertions are defined only within an interface, they will be added into the generated SystemVerilog interface. The purpose of an assertion is to specify and check a set of properties that is expected to hold true in a given component. SystemVerilog provides two types of assertions: immediate and concurrent:

- Immediate Assertions: are statements that include a conditional expression to be tested and a set of statements to be executed depending on the result of the expression evaluation.
- Concurrent Assertions: provide the means to specify sequential properties and to evaluate them at discrete points in time such as clock edges.

4.3.6.8 Testbench Generation

UVM will automatically generate a tesbench based on random sequences to drive the DUT. The base UVM environment can be configured to stop after a certain number of sequences or to stop after a percentage of coverage has been reached. This is useful depending on the verification test plan. The DUT encapsulated with the UVM environment can also be configured to be driven by a SystemC testbench. In that case, the UVM sequences are ignored and UVM is only used for coverage and assertion-based verification.

4.4 Experimentations

4.4.1 Emulation Platform

Our emulation platform is the RazorCam [14, 15], a smart camera system offering a flexible and extensible Hardware/software environment to prototype and to verify video applications. It is capable of streaming image data from 3 camera sensors, through a leading-edge Xilinx Zynq-7000 for processing and analysis. It offers several interfaces including UART and Ethernet connectivity. Linux is used as the embedded operating system on the ARM processor as it offers a solid, familiar platform for development with a feature-rich toolchain. The programmability and the seamless use of hardware accelerators in image processing application are insured

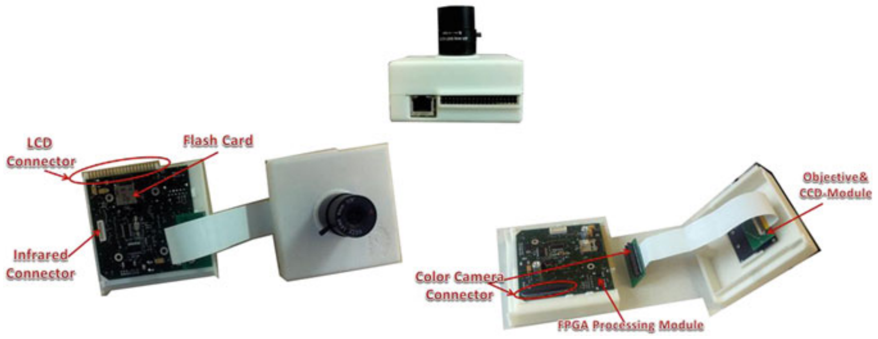


Fig. 4.9 The RazorCam embedded platform

through the design and implementation of a Component Interconnect for Data Access (CIDA). The CIDA allows interfacing amongst hardware accelerators, Hardware and software through its DMA capabilities. The drivers to control the CIDA from the operating system have been developed. The Intels OpenCV computer vision library has been ported to the system and is accessible in the Linux environment. The RazorCam also features a 320 × 240 Thin Film Transistor (TFT) LCD with serial interface for image display (Fig. 4.9).

4.4.2 Case Studies

The proposed methodology has been tested on two benchmarks. The first case study was a new algorithm for line segment detection using weighted mean shift procedures [17], which has application in lane departure and driving assistance. This was an extended version of the Sobel filter processing used as case example throughout this chapter. The processing chain consisted of image acquisition, RGB to Gray color conversion followed by a sobel edge detection, and a weighted mean shift segment detection. Figure 4.10 shows the results at the different levels of the design flow. The result of the emulation is displayed on the TFT display of the RazorCam. The final architecture in the FPGA performs slighly better than the initial OpenCV code when clocked at 50 Mhz.

The second case study (Fig. 4.11) was an implementation of a segmentation, which is at the center of many computer vision applications. The segmentation of Kim and Chalidabhongse [12] was selected. The method is very robust and includes shadow detection. An initially trained background image is compared to the current image using the three red, green, and blue (RGB) color parameters.

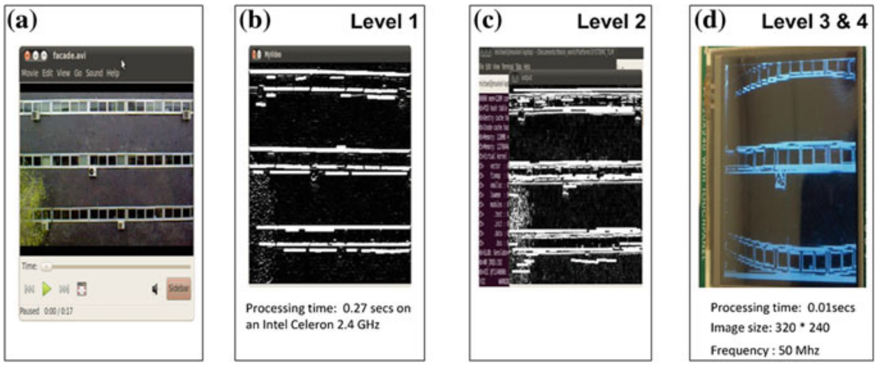

(a)
(b) Level 1
Processing time: 0.27 secs on an Intel Celeron 2.4 GHz
(c) Level 2
(d) Level 3 & 4
Processing time: 0.01secs
Image size: 320 * 240
Frequency : 50 Mhz

Fig. 4.10 Performance evaluation of the line segment detection. **a** The original frame. **b** Simulation of the pure software implementation. All the different blocks are implemented using OpenCV. **c** Simulation of the HW/SW decomposition with SystemC-TLM. **d** Emulation of the final design on the RazorCam

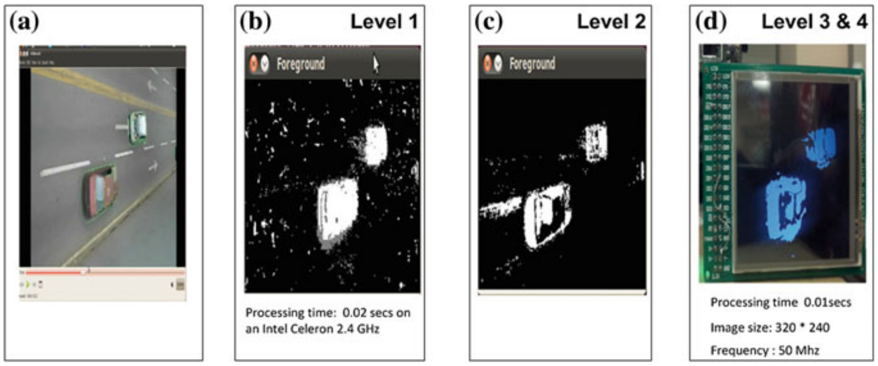


Fig. 4.11 Performance evaluation of the segmentation implementation. **a** Original image. **b** OpenCV. **c** SystemC-TLM. **d** RTL+Emulation on FPGA

4.4.3 Verification Evaluation

In order to assess the proposed verification framework, all experiments have been conducted on an Intel Celeron 2.4 GHz machine with 2 GB RAM running Linux. The generated environment was compiled and simulated using ModelSim from Mentor Graphic. Table 4.1 shows the number of lines in the DUT, the time to generate the environment for the DUT, the CPU simulation time for the DUT, the coverage reached, and the number of failed assertions for both UVM and SystemC testbenches. The UVM testbench has been configured to end after 3000 sequences. The SystemVerilog covergroups for the coverage were automatically derived from the DUT inputs and outputs and implemented using automatic bins in SystemVerilog. We derived the assertions to be checked by looking at the specification of the protocol implemented

Table 4.1 Coverage and assertions verification of the models used in the case studies

SC DUT	Size (lines)	Generation time (s)	Simulation time (s)	UVM testbench Coverage (%)	UVM testbench Failed assertion	SC testbench Coverage (%)	SC testbench Failed assertion
Line detection	420	1	34	79	None	58	None
Segmentation	800	1	47	93	None	64	None

in the DUT and how it was implemented.We can observe that the generation of the UVM environment is fast and practically does not scale with the size of the DUT. Additionally, we can see that an automatic UVM testbench performs a better coverage and assertion verification than a manual SystemC testbench.

4.5 Conclusion

In this chapter, a design flow to rapidly prototype a video application from its high level specification is proposed. The proposed design framework includes OpenCV, QEMU-OS, SystemC, and a target FPGA for emulation. This work also presents an automatic UVM environment to improve SystemC/TLM verification and take advantage of UVM capabilities for a more efficient and faster verification of SystemC/TLM systems. The prototyping environment for video applications allows for software functional verification, hardware functional verification, and rapid prototyping from a high-level specification. This environment used in hardware/software co-design can help to reduce time to market not only of video-based hardware/software systems, but system-on-chip in general.

References

1. Assertion-Based Verification. Springer, US. doi:10.1007/978-0-387-38152-7_13
2. Axi interconnect. http://www.xilinx.com/products/intellectual-property/axi_interconnect.htm
3. Generic and automatic specman based verification environment for image signal processing ips. http://design-reuse.com/articles/20907/specman-verification-image-signal-processing-ip
4. Tlmu. http://edgarigl.github.io/tlmu
5. Aghajan H, Cavallaro A (2009) Multi-camera networks: principles and applications. Academic Press, Waltham
6. Alfaro L, Henzinger T, Interface-based design 195:83–104. doi:10.1007/1-4020-3532-23
7. Appiah K, Hunter A, Kluge T, Aiken P, Dickinson P (2009) FPGA-based anomalous trajectory detection using SOFM. In: Proceedings of the 5th international workshop on reconfigurable computing: architectures, tools and applications, ARC'09, pp 243–254. Springer, Berlin. doi:10.1007/978-3-642-00641-8_24
8. Bramberger M, Doblander A, Maier A, Rinner B, Schwabach H (2006) Distributed embedded smart cameras for surveillance applications. IEEE Comput Soc 39:68–75

9. Chen P, Ahammad P, Boyer C, Huang SI, Lin L, Lobaton E, Meingast M, Oh S, Wang S, Yan P, Yang AY, Yeo C, Chang LC, Tygar D, Sastry SS (2008) Citric: a low-bandwidth wireless camera network platform. In: Second ACM/IEEE international conference on distributed smart cameras, pp 1–10
10. Guenard N, Hamel T, Mahony R (2008) A practical visual servo control for an unmanned aerial vehicle. IEEE Trans Rob 24(2):331–340. doi:10.1109/TRO.2008.916666
11. Kim J, Sukkarieh S (2004) Slam aided GPS/INS navigation in GPS denied and unknown environments. In: The 2004 international symposium on GNSS/GPS, Sydney, pp 6–8
12. Kim K, Chalidabhongse TH, Harwood D, Davis L (2005) Real-time foreground-background segmentation using codebook model. Real-Time Imaging 11(3):172–185. doi:10.1016/j.rti.2004.12.004
13. Lin F, Lum KY, Chen B, Lee T, Development of a vision-based ground target detection and tracking system for a small unmanned helicopter. Sci China Ser F. Inf Sci 52(11):2201–2215. doi:10.1007/s11432-009-0187-5
14. Mefenza M, Yonga F, Bobda C (2013) Razorcam: an embedded platform for image processing. In: ASEE midwest conference
15. Mefenza M, Yonga F, Bobda C (2013) ACM hotmobile 2013 poster: Razorcam: a prototyping environment for video communication. SIGMOBILE Mob Comput Commun Rev 17(3):13–14. doi:10.1145/2542095.2542103
16. Meingast M, Geyer C, Sastry S (2004) Vision based terrain recovery for landing unmanned aerial vehicles. In: 43rd IEEE conference on decision and control (CDC), 2004, vol. 2, pp 1670–1675. doi:10.1109/CDC.2004.1430284.
17. Nieto M, Cuevas C, Salgado L, Garcia N (2011) Line segment detection using weighted mean shift procedures on a 2d slice sampling strategy. Pattern Anal Appl 14(2):149–163. doi:10.1007/s10044-011-0211-4
18. Park J, Lee B, Lim K, Kim J, Kim S, Kwang-Hyun-Baek (2008) Co-simulation of SystemC TLM with RTL HDL for surveillance camera system verification. In: 15th IEEE international conference on electronics, circuits and systems, 2008 (ICECS 2008), pp 474–477. doi:10.1109/ICECS.2008.4674893
19. Rinner B, Winkler T, Schriebl W, Quaritsch M, Wolf W (2008) The evolution from single to pervasive smart cameras. In: Second ACM/IEEE international conference on distributed smart cameras, 2008 (ICDSC 2008), pp 1–10. doi:10.1109/ICDSC.2008.4635674
20. Rose A, Swan S, Pierce J, Fernandez J (2005) Transaction level modeling in SystemC. Open SystemC Initiative 1(1.297)
21. Saha S (2007) Design methodology for embedded computer vision systems. PhD thesis
22. Samsom H, Franssen F, Catthoor F, De Man H (1995) System level verification of video and image processing specifications. In: Proceedings of the 8th international symposium on system synthesis, 1995, pp 144–149. doi:10.1109/ISSS.1995.520626
23. Schlessman J, Chen CY, Wolf W, Ozer B, Fujino K, Itoh K (2006) Hardware/software co-design of an fpga-based embedded tracking system. In: Conference on computer vision and pattern recognition workshop, 2006 (CVPRW'06), pp 123–123. doi:10.1109/CVPRW.2006.92
24. Trost A, Zemva A (2012) Verification structures for design of video processing circuits. In: MIPRO, 2012 proceedings of the 35th international convention, pp 222–227
25. Wu S, Implementing an automated checking scheme for a video-processing device. www.cadence.com/rl/Resources/conference_papers/1.13Paper

Part II
Smart Cameras in Mobile Environments

Chapter 5
Distributed Mobile Computer Vision: Advances, Challenges and Applications

Niki Martinel, Andrea Prati and Christian Micheloni

Abstract The role of mobile devices has shifted from purely passively transmitting text messages and voice calls to proactively providing any kind of information that is also accessible to a PC. The recent advances in the field of micro technology have also made possible to include a camera sensor in any mobile device. This innovation is now attracting both the research community and the industries that aim to develop mobile applications that exploit recent computer vision algorithms. In this chapter we provide an analysis of the recent advances of mobile computer vision, then we discuss the current challenges that the community is currently dealing with. Next, an analysis of two recent case studies where mobile vision is used for augmented reality and surveillance applications is discussed. Finally, we introduce the next challenges in mobile vision where the mobile devices are part of a visual sensor network.

5.1 Introduction

Mobile devices are defined to be Web-enabled devices that are used not in a fixed location but they have been conceived and designed to be portable and usable in mobility. Typical mobile devices include Web-enabled mobile phones and Web-enabled pocket-sized Personal Digital Assistants (PDAs) [44]. The first mobile phone appeared on Detroit police cars in 1921. These devices were communicating together

N. Martinel (✉) · C. Micheloni
Univeristy of Udine, Via Delle Scienze, 206, Udine, Italy
e-mail: niki.martinel@uniud.it

C. Micheloni
e-mail: christian.micheloni@uniud.it

A. Prati
IUAV University of Venice, Santa Croce, 2957, Venice, Italy
e-mail: aprati@iuav.it

with a single high-power antenna installed on a skyscraper that allowed a transmission range of about a hundred kilometers. Twenty five years later the technology had reached the commercial service level.

Two main technological innovations allowed such rapid growing and they are still pushing the design of new, faster and more highly-featured devices. Over the years the advent of transistors boosted mobile device technology: the possibility of building tiny and sophisticated electronic components enabled phone companies to design a large number of models with many advanced features. The other important aspect—strictly connected with the diffusion of mobile devices—was the innovation of the communication infrastructure. Nowadays the community is talking about the fourth generation of mobile networks that should achieve a 1 GB/s transmission data speed. The new, more reliable and faster network infrastructures pushed more and more the mobile device companies to design and develop new, more sophisticated and innovative, mobile devices that take advantage of these resources.

Mobile devices were initially designed with a single physical keyboard, now they are equipped with touch-screens technologies borrowed by Personal Digital Assistans (PDAs) and other handheld devices: the keyboard has been replaced by a screen that allows the user to interact with a graphical interface using fingers or other pointing devices. These, together with faster microprocessors, allow an excellent user experience and make the tasks of multimedia processing and data logistic possible on the fly.

Historically speaking, the PDAs can be seen as the first generation of mobile devices. A PDA is a handheld computing device that combines multiple functions and features including telephone, fax, Internet and networking or other form of different connectivity capabilities. These devices are mainly used by users that need to compute operations while moving, simply called "Mobile Computing". Before that such features were provided only by laptops or desktop computers. The PDA is the first step to a future trend that would be later defined as "Technological Convergence" [17]. The next generation of mobile computing, mainly represented by smartphones, will foster the convergence of communication, computing and consumer electronics, three traditionally distinct industries with quite low interoperability. While mobile phones were previously equipped with a simple address book and agenda, now a smartphone has several features such as a camera, a voice control system and so on. In other words, a smartphone can be seen as a mobile phone with computer capabilities that allows it to interact with computerized system, send email and access to the web.

We are now in an era where the communication and computing environment is moving to interact with the physical environment or even become part of it. Mark Weiser, chief scientist at Xerox PARC and considered the father of Ubiquitous Computing, claimed that in the 21st century the technology revolution will move into "the every, the small and the invisible". That is what is really happening now: the information processing moves to the background so as humans concentrate on the tasks, not on the tools. The technology is viewed as a tool to serve the needs of people, not something to depend on. Weiser suggested that the most deep technologies are those that disappear ("Disappearing Technology"). They weave themselves into the fabric of everyday life until they are indistinguishable from it. The concepts of access anywhere, anytime, from any device are intertwined with the progress of the network infrastructures and with the aspects of convergence.

5.2 Chapter Contributions

Mobile devices are an inseparable part of our society now. The role of such devices has shifted from purely passively transmitting text messages and voice calls to proactively providing any kind of information that is also accessible to a PC. In particular, due to the advances in smart and micro technology, camera sensors are now a standard component in all mobile devices. This innovation is now attracting both the research community and the industries that aim to design and develop applications that exploit the powerful features of mobile devices and combine them together with the more advanced computer vision and image processing techniques.

In this chapter, we contribute to the research in distributed smart cameras introducing the most relevant advancements in mobile computer vision. Then we discuss the main challenges faced by the mobile vision community such as: (1) the limited energy, (2) the limited storage, (3) the limited computational capabilities, (4) the wireless network communication, (5) the scalability of applications. Next, we discuss two main research studies where computer vision techniques and distributed frameworks are used for mobile applications of augmented reality and security purposes. Finally we introduce the concept of smartphone networks, where the mobile device is part of a visual sensor network.

5.3 Mobile Computer Vision

A few years ago, it was very difficult to imagine that in the near future digital cameras would become a standard component of mobile devices. In these days, such devices have achieved a good level of maturity and they are now equipped not only with camera sensors, but with various other sensors such as accelerometers, gyroscopes, and GPS receivers. The exponential evolution of image and video processing devices with ever increasing computational capability equipped with high-resolution cameras and hardware-accelerated graphics has opened to a broad and new emerging research area that exploits the mobile device camera for applications of computer vision technologies. The broadband wireless network connection also enables mobility applications to use the acquired video data to initiate queries and exchanges of information with other mobile devices or higher computational power infrastructures. Though it is still in its infancy, it has attracted much attention from both industry and academia. This is not surprising, since we are indeed in an era of transition from a focus on PC-based computing to a greater emphasis on smart devices and cloud-based computing.

Among the multitude of mobile applications that have been designed to exploit the images coming from device cameras for computer vision applications, we can generally group them into three main clusters: location-based services, mixed/augmented reality, and car safety applications. These topics have become very popular in recent years, largely in the context of consumer applications where user-centered visual computing is essential.

Regarding the location-based services, industry has put a lot of effort to achieve real-time visual image recognition for these applications. Deployments of such systems include Google Goggles [13], Nokia Point and Find [34], Kooaba [20] and Snaptell [3]. A specialized case is represented by Leafsnap [21] which is an electronic field guide that uses visual recognition software to help identify tree species from photographs of their leaves. In essence, all these applications start from a snapshot image taken by a user on a mobile device. Then, the photo is matched against a pre-annotated image database to extract useful information that is provided to the user. In the recent years, the problems of image retrieval [12], landmark [24] and location [39] recognition using appearance-based features have also been deeply investigated by the community. In order to achieve their objectives, these methods match appearance features against a large database of location-tagged images [42]. In [48] authors propose a system for determining a user's location from a mobile device via image matching. The authors first build a "bootstrap database" of images of landmarks and train a CBIR algorithm on it. Since the images in the bootstrap database are tagged with keywords, when a query image is matched against the bootstrap database, the associated keywords can be used to find more textually related images through a web search. Finally, the algorithm is applied to the images returned from the web search to produce only those images that are visually relevant.

Mixed/augmented reality mobile video gaming is another area which has caught the attention of many. With the increasing quality and spatial resolution of mobile device cameras we have now high level of augmented reality where real-time interaction is possible. Another motivation of the spreading of AR solutions is the diffusion of available Software Development Kit (SDK) for efficiently and timely construct your own mobile AR application. Two examples above all: Qualcomm Vuforia [35] which is generic SDK for optimized augmented reality and object recognition and has been used by more than 3,500 mobile apps world wide; and Sentisight [33] which is a SDK for object recognition, computer vision and augmented reality used in a number of mobile vision applications.

Although, it is a matter of fact that, for some applications, we may need superior devices as they typically have much more computational capacity and additional sensors, enabling computationally expansive mobile applications on the fly. A recent demonstration of an outdoor mobile augmented reality application running on a cell phone is Nokia's MARA project [14]. The system does not perform any image analysis, instead it uses an external GPS for localization and an inertial sensor to provide orientation. PhoneGuide [9] is one of the first object recognition systems performing the computation on a mobile phone, instead of sending the images to a remote server. The system employs a neural network trained to recognize normalized color features and is used as a museum guide. Similarly, in [28] authors propose a novel framework to support AR for painting on mobile devices. In [18, 22, 46] recent techniques for tracking and occlusion handling in an AR framework were discussed. In [40] authors introduce a mobile system based on a hand-held device, GPS sensor, and a camera for roadside sign detection and inventory. Their algorithm was efficient enough to ensure good quality results in mobile settings. In the context of augmented reality, in [11] authors use a modified version of the SIFT algorithm for object detection and recognition in a relatively small database of mobile phone imagery of urban environments. The system uses a client-server architecture, where a mobile phone client captures an image of an urban environment and sends it to the server for analysis. The SURF algorithm has been used successfully in a variety of applications, including an interactive museum guide [5]. Local descriptors have also been used for tracking. In [43] authors track SURF features using video coder motion vectors for mobile augmented reality applications. The challenges of real-time recognition and camera pose estimation system for planar shapes were addressed in [16]. The proposed system performs shape recognition by analyzing contour structures and generating projection-invariant signatures. Similarly, in [41] SIFT features are used for recognition, tracking, and virtual object placement. Camera tracking is done by extracting SIFT features from a video frame, matching them against features in a database, and using the correspondences to compute the camera pose. In [10], the monoSLAM system estimates the hand-held camera's motion from the live image stream to achieve high AR performance. In [15], an AR rendering pipeline that supports global illumination techniques was proposed.

Another interesting and diffused field of application of mobile vision is the applications for car safety. In this case, the smartphone is positioned in front of the car and used for both analyzing the scene outside (ahead situation) or inside the car. Two existing products deserve special mention. CarSafe [49] uses rear and rear-facing front cameras for in-vehicle applications. The rear camera is used for monitoring distances from other vehicles and for tracking lane changes, whereas, rear-facing front camera tracks the driver's head position and direction as well as eyes and blinking rate as indicators of microsleep, drowsiness, and distraction. iOnRoad [19] uses Qualcomm's FastCV mobile-optimized computer vision library for frontal collision warning and lane departure warning. It also monitors headway and can be used for identifying and locating other cars in the field-of-view.
All of these applications pose a unique set of challenges.

5.4 Challenges

The proliferation of mobile and hand-held devices, along with advances in multimodal and multimedia technologies, are producing a new wave of applications that enable users to quickly and more naturally perform many tasks. These include: finding music, videos, and business listings; surfing the Web; sending a short text message; interacting with social media Web sites; just to mention a few. Mobility is central to this growing number of applications. It is a matter of fact that, as the number of smartphones and emerging devices continue to grow, user demand for new multimodal and multimedia interfaces that allow them to interact with the device while in mobility. Speech recognition and text-to-speech synthesis, are recent examples of this. Over the next few years, we expect to see multiple variants of speech and image processing technologies on smartphones to be features as standard as a keyboard. This is partially driven by regulatory requirements prohibiting texting while driving, and the need to provide a natural and a more compelling user interface with hands-free, eyes-free operation.

Another driver to the mobile revolution is cloud computing. It is significantly reducing the cost for deploying and maintaining large-scale mobile media services and enabling location-aware technologies for video, music, speech, and language to be more readily available as Web services. Indeed, developers can easily access such technologies through Web services application programming interfaces (APIs) that take advantage of standards such as HTML-5 and W3C EMMA [45].

The convergence of media and mobility is not only creating new opportunities, but also opening to a new set of challenges. In addition to the challenge posed by the limited battery life of mobile devices, we can find three prominent challenges in mobile vision when compared with traditional computer vision applications.

First of all, although the computational capacity of mobile devices is constantly increasing, it is still not sufficient to handle large-scale visual computing tasks. From this perspective, migrating much of the computation to powerful devices or to the cloud is essential. But, what part of the processing should be performed on the mobile client, and what part is better carried out by the server? On one hand, processing images is now possible on mobile devices in seconds or less. On the other hand, transmitting an image could take tens of milliseconds over a high speed wireless link. There are several possible architectures we can think about

- The mobile client transmits a query image to the server. The algorithms run entirely on the server, including an analysis of the query image. The final result is then sent back to the device as in a standard client-server architecture.
- The mobile client processes the query image and transmits (abstract) data. The algorithms run on the server using the data as query and returns the result to the device.
- The mobile client downloads data from the server and all the processing is performed on the device. This solution has the advantage to limit the required bandwidth (data can be downloaded only when the application starts or whenever they change), but can be slow due to the limited computational resources of the device.

It is worth emphasizing that this type of applications usually call for a fast response to address the user's requirement of a fluid interaction with the device. Therefore, a way to ensure real-time responses to user interaction will be a major issue to be tackled. Moreover, the bandwidth requirement is an important issue since connectivity can be not always available while on move or be in general expensive for the user.

Secondly, most mobile vision applications are driven by large amounts of annotated visual data. For example, to enable vision-based location recognition, a large collection of street-view images is a prerequisite. How to acquire and process such data is a challenge. One way to acquire the data is to harness the massive user-generated visual databases on the Internet, such as online image/video sharing systems. But then, a lot of information must be either locally stored by the mobile device or exchanged through the network.

Thirdly, not all mobile devices and smartphones that run the same application are equal. Some of them have very high computational resources and memory capabilities, while others only have reduced memories and limited computing cycles. What is more, they may have a reduced network connection speed so as only fewer round trips between the device and the server can be performed per second. This introduces the problem of scaling application requirements to each specific device.
Generally, scaling refers to growing computing resources to support an application. Vertical scaling is increasing single device capabilities, with powerful and faster CPUs, more memory, or faster disk I/O. While, horizontal scaling involves adding additional computational devices to support the overall applications computing requirements, and load balancing across those resources. So, scalability is generally viewed from a hardware designer perspective. We want to change this point of view and refer to the scalability as to adapt an application to support mobile devices that have different features. This aspect will be more clear in the second case study introduced below.

5.5 Case Studies

In this section two recent case studies that exploit advanced computer vision techniques are described. The former introduces an efficient marker-less approach to support augmented reality for paintings. While, the latter introduces a cooperative method to address the challenges of the person re-identification problem using both mobile devices and remote processing units.

5.5.1 Augmented Reality for Musems

Recognizing paintings and computing the transformation to align the acquired image of a painting and its image in a database to support Augmented Reality (AR) applications is nontrivial tasks. Common static computer vision issues (e.g.illumination, view, scale changes, etc.) are more severe in context of moving cameras as different effects (e.g.blur, noise, motion, etc.) arise. In addition, reflection of spotlights, image saturation and image exposure add up for the recognition of the paintings present in exhibitions.

While specifically designed markers have been the dominant choice by state-of-the-art AR methods, here a marker-less approach is introduced. Using the principle of Hough line detection, a method to detect and extract the relevant painting region (RPR) from a given input image is first applied, then local features are extracted from

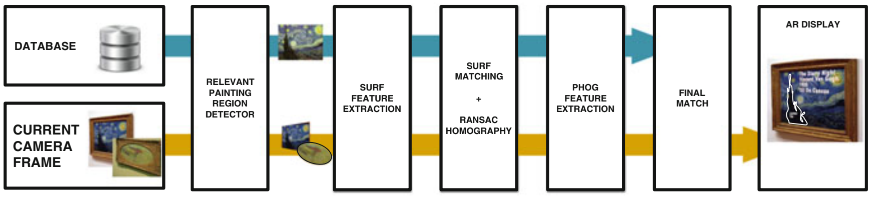


Fig. 5.1 Architecture of the proposed system. The current camera frame is processed by the three main modules of the system. The first module extracts the RPR. The second module matches every database image with the current camera frame. Given the best match, the third module uses the RANSAC homography transformation to overlay the additional content to the current camera frame

the RPR and matched with a candidate target in the database. RANSAC is used to detect feature outliers. As the current image may not be aligned with the candidate target image, extracting global feature from it considerably reduces robustness, so the homography transformation output by RANSAC is used to sidestep this issue and align the current RPR to the candidate target RPR. This allows to extract global feature from the aligned RPR only, thus noticeably improving performance. Then, a weighted similarity measure is used to compute the final match between image signatures composed of local and global features. Once a match is found, the same RANSAC homography is exploited in an AR framework to properly overlay the additional content to the current frame.

5.5.1.1 System Overview

The architecture of the proposed approach is shown in Fig. 5.1. Three main modules are used to achieve the proposed goal: (1) the RPR detector module, (2) the matching module and (3) the AR display module. Given the current camera frame, the RPR detector module extracts the RPR so as the painting frame and the background are not considered anymore. Once the RPR is detected, the matching module extracts the local features from such region and matches them with each candidate target local features. RANSAC is used to reject matching outliers. Then, the homography matrix output by RANSAC is used to align the current RPR with the candidate target RPR from where the global features are extracted. The extracted local and global features are finally matched with the candidate signature using an affine combination of similarity measures. Given the best candidate target match, the same homography transformation output by RANSAC is used to properly overlay the additional content to the current camera frame.

5.5.1.2 Relevant Painting Region Detector

Assuming that the frames of paintings have elliptical or rectangular shape, the RPR detector goal is to remove the background and the painting frame to keep only

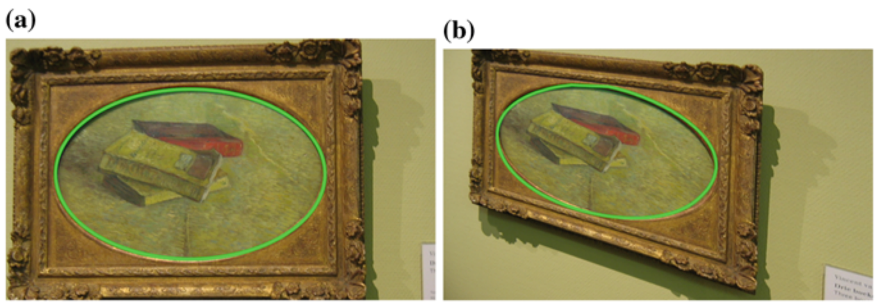


Fig. 5.2 RPR detection examples. In **a** an ideal example where the painting plane is almost aligned with the given image plane is shown. In **b** a harder example with severe perspective distortion is depicted

the portion of the image that contains the painting, i.e. the RPR. To achieve such objective saving computational cycles, the Randomized Hough Transform (RHT) method [47] is used. Such technique is based on the standard Hough Transform (HT) but it avoids the computationally expensive voting procedure. As shown in [47], the RHT can achieve a computational complexity lower than an upper bound of order $O((n_t N^n)/n_{min}^n)$, that is considerably smaller than the order $O(N N_a^{n-1})$ of the standard HT. N and N_a are the total number of pixels in the image and the size of the accumulation array respectively. n is the number of curve parameters, n_{min} is the length of the shortest curve in the image, and n_t is a small number.

To detect the RPR, we first apply the Canny operator to the grayscale representation of the current camera frame $Q \in \mathbb{R}^{M \times N}$. Then, as shown in Fig. 5.2, the RHT is used to fit ellipses and rectangles. As a painting may contain more than a single rectangular or ellipse shaped object, the RPR boundary is selected as the detected rectangle or ellipse which area is at least r times the input image size. The detected RPR is denoted as $R_Q \in \mathbb{R}^{M' \times N'}$ where $M'N' \geq rMN$ and $r \in [0, 1]$.

5.5.1.3 Matching

Let R_Q and R_I be the RPR of the current camera frame Q and the RPR of a database candidate target image I, respectively. To match R_Q and R_I two local and global features have been considered. In particular to match two RPRs with lower computational costs, the Speeded-Up Robust Features (SURF) [4] and the Pyramid of Histogram of Oriented Gradients (PHOG) [7] features have been used. Both a local and a global feature have been used as the relevant region detector module may fail, i.e. only a small area of the painting or the painting with the frame can be extracted. In such cases, if only global descriptor is extracted non-interesting information is considered.

Local features: As illumination invariance is intrinsic to SURF [4], such features are extracted from the grayscale representation of R_Q by exploiting the standard integral image. The SURF feature detector is based on an approximation of the Hessian matrix, while the feature descriptor $\psi_F^{R_Q} \in \mathbb{R}^{64}$ describes the distribution of Haar-wavelet responses within the neighborhood of the detected interest points $\psi_K^{R_Q} = [x, y]$. The computed SURF feature vector is denoted as $\psi^{R_Q} = \langle \psi_F^{R_Q}, \psi_K^{R_Q} \rangle$.

Given two SURF feature descriptors $\psi_F^{R_Q}(q)$ and $\psi_F^{R_I}(i)$, q, i is defined to be a match if the similarity

$$S_F(\psi_F^{R_Q}(q), \psi_F^{R_I}(i)) = \frac{1}{1 + \|\psi_F^{R_Q}(q) - \psi_F^{R_I}(i)\|_2} \tag{5.1}$$

is higher than a fixed threshold Th_s. If that is satisfied, the two matching SURF feature vectors are analyzed to detect outliers using a RANSAC-based approach similar to the one proposed in [8]. In particular, given 4 feature correspondences, the homography $H_{Q,I}$ is computed using the Direct Linear Transformation method. The process is repeated with t trials, and the solution that has the maximum number of inliers is selected. A SURF feature keypoint $\psi_K^{R_Q}(q)$ is considered to be an inlier if the corresponding keypoint projection $\widehat{\psi}_K^{R_Q}(q)$ is consistent with $H_{Q,I}$ within a tolerance of σ pixels.

Global features: Given $H_{Q,I}$, that is used to to align the two RPR regions, namely R_Q and R_I. The resulting transformed RPR is denoted as $\hat{R}_Q$. By applying such transformation to global feature can be extracted in a more reliable fashion as the edges used to compute the global features are aligned and have similar orientations to the edges of the candidate target.

PHOG features are extracted from $\hat{R}_Q$ to capture information about the shape and the whole appearance of the painting. Before extracting PHOG features, the transformed RPR region $\hat{R}_Q$ is projected into the HSV color space to achieve illumination invariance. Then, edges and orientation gradients are used to compute the PHOG feature matrix $\rho^{\hat{R}_Q} \in \mathbb{R}^{m \times 3}$ by concatenating the PHOG histograms extracted from the three image channels at the different levels of the spatial pyramid. m is the total number of histogram bins for each image channel.

Candidate target matching: Once local and global features have been extracted, the two given images Q and I are matched using an affine combination of feature similarities.

SURF features similarity is computed as

$$\Phi_\psi(\psi^{R_Q}, \psi^{R_I}) = \frac{\sum_{q,i \in match} S_F(\psi_F^{R_Q}(q), \psi_F^{R_I}(i))}{\varepsilon + match} \tag{5.2}$$

where $match$ is the total number of matched SURF features and ε is a small constant used to prevent division by zero.

PHOG features are matched using the same similarity function suggested in [27]. Let $\rho^{\hat{R}_Q}$ and ρ^{R_I} be the PHOG feature matrices of $\hat{R}_Q$ and R_I respectively. The PHOG similarity is computed as

$$\Phi_\rho(\rho^{\hat{R}_Q}, \rho^{R_I}) = 1 - \sum_c \lambda_c \chi^2(\rho_c^{\hat{R}_Q}, \rho_c^{R_I}) \tag{5.3}$$

where $\rho_c^{\hat{R}_Q}$ and $\rho_c^{R_I}$ are the PHOG features computed for the c-th color channel. λ_c is the normalization weight.

As the final objective is to find the database image that gives the best match with the query image, a minimization task needs to be performed. Towards this end, let $\mathbb{I}$ be the set of all database images, then, the final objective is to find $\arg\max_{I\in\mathbb{I}} \Phi(Q, I)$ where

$$\begin{aligned}\Phi(Q, I) = {} & \alpha\, \Phi_\psi(\psi^{R_Q}, \psi^{R_I}) \\ & + \beta\, \Phi_\rho(\rho^{\hat{R}_Q}, \rho^{R_I})\end{aligned} \tag{5.4}$$

α and $\beta = 1 - \alpha$ are the affine coefficients.

5.5.1.4 AR Display

The last module of the proposed system is in charge to overlay the additional content to the current camera frame Q. As both paintings and the additional content are planar surfaces, the transformation that needs to compute can be described as an homography. In this work a feature-based homography computation method is used. Towards this goal, the same feature-based homography transformation $H_{Q,I}$ computed in Sect. 5.5.1.3 is exploited. As shown in Fig. 5.3, using the inverse homography transformation matrix $H_{Q,I}^{-1}$ it is possible to overlay the additional content (given in the original region I coordinate system) to the current camera frame Q.

5.5.1.5 Experimental Results

To show the performance of the proposed method, experiments have been carried out on a dataset built using 607 publicly available pictures of 70 Vang Gogh paintings. The dataset has pictures taken from different viewing angles and with severe illumination changes. Some pictures come with light reflections and occlusions as well.

Implementation details: The values of the algorithm parameters given in the following have been selected using 4-fold cross validation:

- RPR boundary: r has been set to 0.55;
- SURF features: features have been extracted using 5 octaves and 4 scale levels;

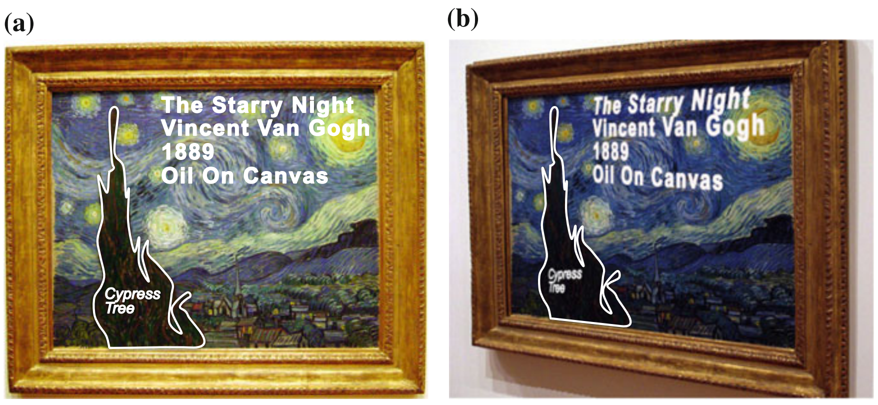


Fig. 5.3 Example of AR. **a** Reference frame with the additional information to display. **b** Current frame with the additional information transformed using the feature-based homography transformation

Table 5.1 Average nAUC values for different values of alpha (from 0 to 1 with steps of 0.01)

α	0	0.25	0.28	0.5	0.75	1
RPR detection	0.7927	0.8110	**0.8320**	0.7485	0.6917	0.6281
No RPR detection	0.6368	0.6572	**0.6635**	0.6470	0.6341	0.6290

Best results are in boldface font

- PHOG features: edges have beeen extracted using the Canny operator, while orientation gradients have been computed using a 3×3 Sobel mask. The extracted HOG features extracted from the 4 levels of the spatial pyramid have been quantized in 9 bins;
- Matching: The normalization weight vector λ has been set to [0.5, 0.3, 0.2]. The tolerance σ has been set to 4 pixels and $t = 500$ trials are performed to compute H. The matching threshold Th_s has been set to 0.85.

To show the performance of the method results have been reported in terms of ROC curves and normalized Area Under Curve (nAUC) values.

The algorithm has been tested on a standard PC with P4 CPU 2.0GHz, 1GB RAM, Windows XP and on a Tablet with ARMv7 processor 1GB RAM, Android 4.2.2. In the first test with non-optimized MATLAB code the average recognition and registration time for a single frame was 0.591s, while in the latter with optimized code the same activities took 0.632s.

Experiments: In the following we show the performance of the proposed method as a function of both the scale of the images and the rotation of them with respect to the target image orientations.

In Table 5.1, nAUC values computed as a function of the similarity normalization weight α are reported. Each value is computed averaging all the results computed for images scaled to 1/2, 3/4 and 1/1 of the original image size and for different image

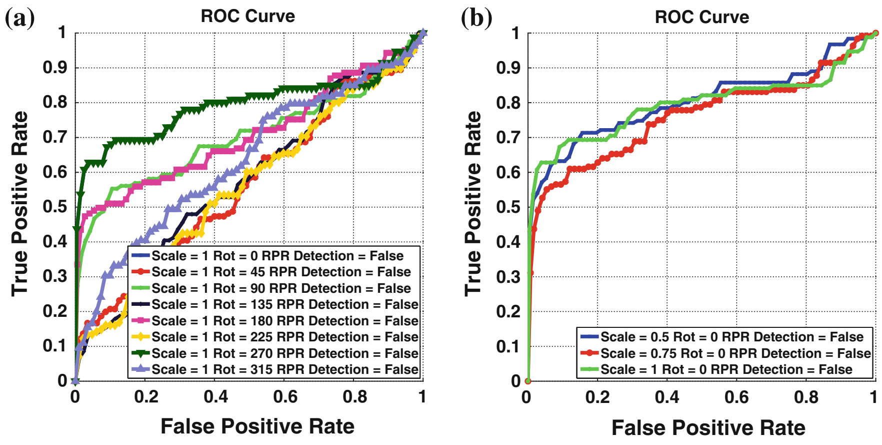


Fig. 5.4 Recognition performance computed without using the relevant region detector module. In **a** test images are rotated by multiples of 45°. In **b** test images are not rotated but their scaling factor has been changed

rotations from 45° to 315°, with intermediate rotations of 45°. The best results are achieved for $\alpha = 0.28$. In light of this, such value has been used in the following experiments.

In Fig. 5.4 the performance of the proposed method without using the RPR detector are shown. In Fig. 5.4a results are shown as a function of the rotation of the test images. The original scale has been used. The method achieves reasonable results for rotations multiple of 90°. The performance decreases in the other cases. This is probably due to the changes occurring in the oriented gradients used to compute the PHOG features. On average, a true positive rate of 49 % is achieved for a false positive rate of 20 %. In Fig. 5.4b results for different image scales are shown. Thanks to SURF invariance properties and the pyramidal approach used to compute PHOG, the performance are not much affected by the scaling issues. In such scenario, an average true positive rate of 67 % is achieved for a false positive rate of 20 %.

In Fig. 5.5 the performance of the proposed method using the RPR detector are depicted. In Fig. 5.5a results have been computed for different rotations to the test images as in Fig. 5.4a. On average, a 71 % true positive rate is reached for a false positive rate of 20 %. Though, the worst results are reached for rotations of 135° and 225°, where a true positive rate of about 59 % is reached for the same false positive rate of 20 %. If compared to the results shown in Fig. 5.4a, a significant improvement has been achieved. Most importantly, the performance has increased of about 33 % for a false positive rate of 0 %. In Fig. 5.5b the results are computed varying the scale of test images. If compared to Fig. 5.4b, an average improvement of 37 % is achieved for a false positive rate of 0 %.

In Table 5.2 results of the proposed method are shown as nAUC values. Images scaled to 1/2, 3/4 and 1/1 of the original image size and rotations from 45° to

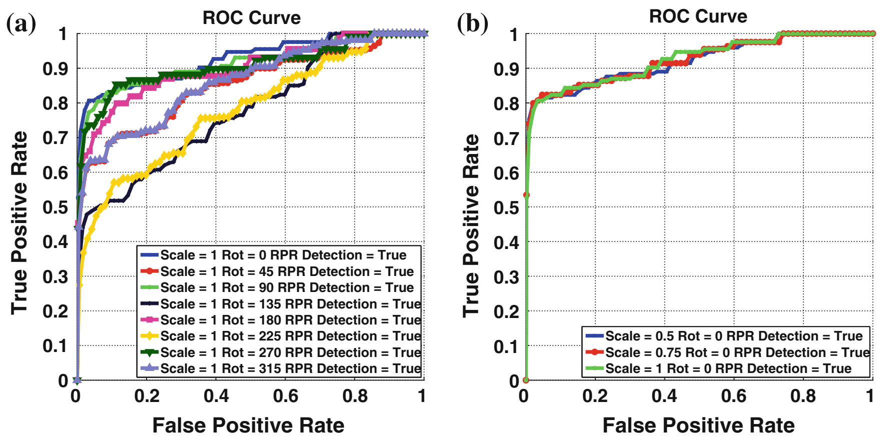


Fig. 5.5 Recognition performance computed using the relevant region detector module. In **a** test images are rotated by multiples of 45°. In **b** test images are scaled down using multiple reduction factors

Table 5.2 nAUC values computed for test images scaled to 1/2, 3/4 and 1/1 of the original size and rotated from 0° to 315°(steps of 45°)

	Rotation	0°	45°	90°	135°	180°	225°	270°	315°
RPR detection	Scale = 0.5	0.9223	0.8025	0.8918	0.7178	0.9020	0.6708	0.8949	0.7518
	Scale = 0.75	0.9243	0.8049	0.9032	0.7158	0.9128	0.7010	0.8986	0.8209
	Scale = 1	0.9258	0.8233	0.9138	0.7188	0.9180	0.7044	0.9053	0.8226
No RPR detection	Scale = 0.5	0.8130	0.5502	0.7015	0.5492	0.6810	0.5476	0.7963	0.6133
	Scale = 0.75	0.8104	0.5529	0.7023	0.5510	0.6966	0.5498	0.8082	0.6214
	Scale = 1	0.8244	0.5655	0.7026	0.5738	0.7082	0.5585	0.8144	0.6329

315° (with steps of 45°) have been considered. The first three rows show the results of the proposed method using the proposed RPR detector, while in the last three rows results have been computed without using the RPR detector. For both such cases the best results are achieved when the original image size is kept and no rotation is applied. However, using the relevant region detector, performance increases of more than 17 % on average. In particular, for rotation of 90°and 180°, an average increment of 20 % is achieved.

5.5.1.6 Conclusion and Discussion

In this case study, a marker-less method for painting recognition and registration to support mobile AR applications has been introduced. A RPR detector is used to extract only the relevant painting region, that is next considered to extract local features that are matched with a candidate target using RANSAC. The homography transformation output from RANSAC is also applied to align the detected RPR to the RPR of the candidate target. This allows to robustly extract global features that are finally used, together with local features, to compute the match between the current frame and the candidate target. Once a valid match is detected, the same homography transformation output by RANSAC is used to overlay the additional content to the current frame. The method has been evaluated using a dataset of publicly available images showing significant achievements.

5.5.2 Cooperative Person Re-Identification

As discussed before, mobile devices with exponentially increasing capabilities have been recently introduced into the market, thus boosting the development of computer vision algorithms for mobile devices. Though standard computer vision algorithms can be ported to mobile devices, the computational costs required and the limited resources have reduced their applicability. Many problems have been investigated by the computer vision community especially for security purposes [31]. Despite this, there is limited work that exploits the cooperation between mobile devices and a camera network for surveillance purposes.

To monitor a wide area traditional surveillance CCTV systems are still the most common choice. Despite the benefits of surveillance systems and the advantages of recent and performing surveillance applications [29], static camera infrastructures still show several disadvantages. Among of them we find the restricted field-of-views, the low resolution of most static cameras, and the limited communication bandwidth, just to mention a few. In this section a client-server system is introduced to tackle these challenges with particular focus to the task of person re-identification, that is, the task of assigning the same label to the same person viewed by different cameras at different time instants. The proposed system brings several advantages as it allows to monitor a large portion of the environment using moving cameras (i.e. mobile cameras). It also introduces a better use of resources and reduces the computational cost of the re-identification. In particular, in this case study the objective is to find the configuration of the mobile client device that should be used to achieve a real-time processing while keeping high re-identification performance. To achieve such objective and save network resources the system limits the exchanged information -between the device and the server- to data vectors of small dimensions.

5.5.2.1 System Architecture

An overview of the proposed system architecture is shown in Fig. 5.6. The system flow goes as follow. Given the image of a suspicious person acquired by a camera in the network, the server computes its signature and sends it to all the mobile devices (i.e. clients). Once the signature is received, each client starts capturing the scene. Then, the first acquired frame of a person is sent to the server together with mobile

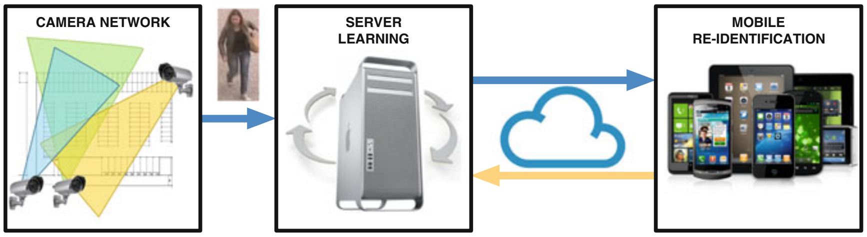


Fig. 5.6 Client-server communication overview. The image of a suspicious person acquired by a camera in the network is sent to the server that routes it to each of the connected mobile devices. Then, the first image acquired by each mobile device is sent back to the server, that, exploiting a learnt model, instructs the mobile device about which features have to be extracted to perform the re-identification

device information regarding its computational and networking capabilities. The server analyzes the received data and instructs the client with which features should be extracted to achieve the best re-identification. This process is mainly guided by a model learned during a separate off-line training phase (details in the following). Then, each client starts to process the frames to detect the persons blob and to extract the features as instructed by the server. The computed features are finally matched with the previously received signature to perform the re-identification. However, the re-identification performance may not stable over a certain period due to the changes in illuminations, pose, etc.. To sidestep such issue a server re-configuration is performed. This is done by forcing the client to send another frame and restarting the whole procedure.

More in detail, as shown in Fig. 5.7, the proposed system has been designed to exploit three main phases: (1) an offline training phase, in which the server collects images and device characteristics (resolution, CPU performance, etc.) to train a classifier; (2) an online learning phase used to instruct the querying device about which feature algorithm should be used; (3) a re-identification phase where the selected feature algorithm is used to extract and match the person acquired by the querying device with the suspect image sent by the server.

5.5.2.2 Image Complexity and Feature Extraction

The training phase and the re-identification phase share a common step, that is, the computation of the image complexity and the extraction of image feature.

Images are classified with respect to their complexity by extracting the edges and corners using Canny and Harris corner detector methods respectively. Let n_e be the number of edges and n_c be the number of detected corners for a given image. The image complexity denoted as $C \in [0, 1)$ is computed as

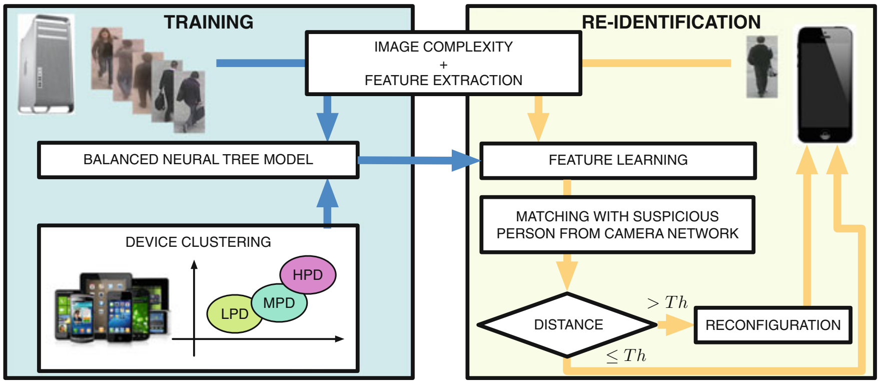

TRAINING
IMAGE COMPLEXITY + FEATURE EXTRACTION
BALANCED NEURAL TREE MODEL
DEVICE CLUSTERING
HPD
MPD
LPD
RE-IDENTIFICATION
FEATURE LEARNING
MATCHING WITH SUSPICIOUS PERSON FROM CAMERA NETWORK
DISTANCE
$> Th$
$\leq Th$
RECONFIGURATION

Fig. 5.7 Main system training and re-identification steps. A set of training images and the device complexity clustering model are used to train a balanced neural tree. During the re-identification phase, frames acquired by the mobile devices are sent to the server together with device details. The server analyses the input data to select the most discriminative features for re-identification according to model learnt in the training phase. Then the matching is performed with such features. If the threshold on feature distances is reached, the system gets re-configured and the querying devices gets instructed again about what features should be used in the next steps. This allows to adapt the re-identification performance to the context

$$C = 1 - \left(\alpha \frac{1}{n_e + 1} + \beta \frac{1}{n_c + 1}\right) \tag{5.5}$$

where α and $\beta = 1 - \alpha$ are the normalization weights.

The community has developed a large number of approaches to detect and describe discriminative patterns of a given image. Scale Invariant Feature Transform (SIFT) [26] and Speed-up Robust Feature (SURF) [4] are two well-known feature algorithms that achieve great performance under a variety of image transformations. More recently other feature detector algorithms as BRISK [23], FAST [37], STAR [1] and ORB [38] have been proposed and designed to be runtime efficient. A more detailed description of these can be found in the computational performance evaluation review given in [32]. In our approach we used the aforementioned feature detectors implemented in the OpenCV libraries. As the FAST and ORB do not incorporate a descriptor, these features are completed with the Fast Retina Keypoint (FREAK) descriptor [2].

5.5.2.3 Training Phase

As described in Sect. 5.5.2.2, several feature algorithms can be used but, due to their different implementations and configurations, they differ in performance and in the number of features extracted. Differences arise depending on the kind of image and on the device capabilities as well. An example of this is shown in Fig. 5.8. The key

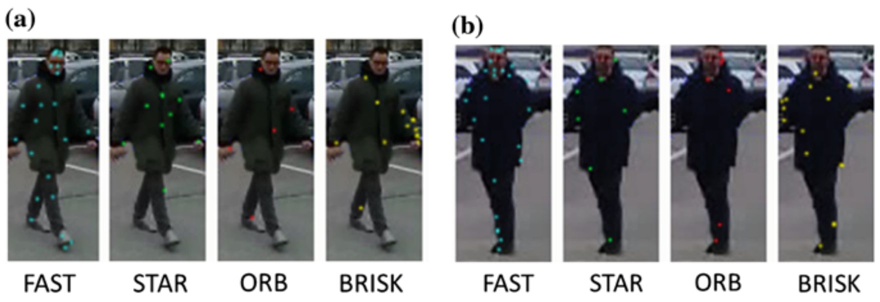
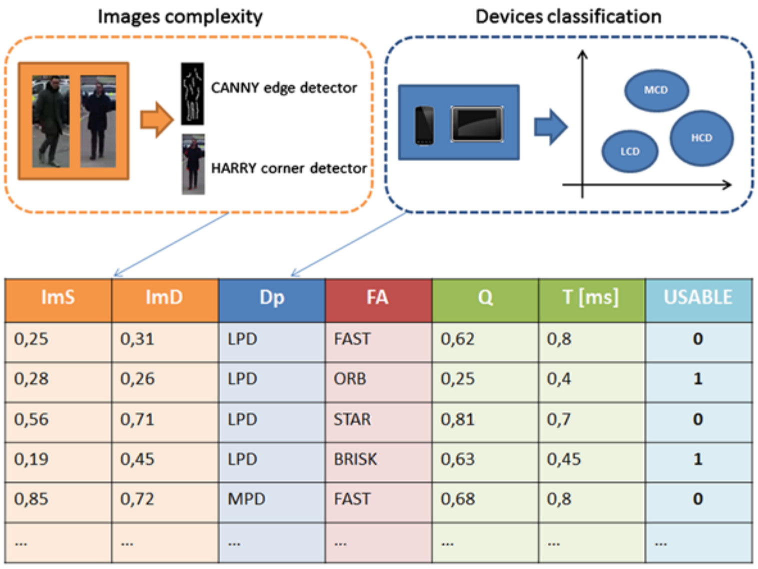


Fig. 5.8 Different state-of-the-art features are extracted from two different persons using the same mobile device with same configuration



ImS	ImD	Dp	FA	Q	T [ms]	USABLE
0,25	0,31	LPD	FAST	0,62	0,8	**0**
0,28	0,26	LPD	ORB	0,25	0,4	**1**
0,56	0,71	LPD	STAR	0,81	0,7	**0**
0,19	0,45	LPD	BRISK	0,63	0,45	**1**
0,85	0,72	MPD	FAST	0,68	0,8	**0**
...	...	...	...	...	...	...

Fig. 5.9 Training set example. The first two columns represent the complexity of images captured from static cameras (*ImS*) and from mobile device cameras (*ImD*). *Dp* represents the performance index of each device. *FA* represents the feature algorithm used to compute the match. Finally *Q* and *T* are two performance parameters and refer to the quality index and to the computational times respectively

idea is to determin -for a given image complexity/structure and for a given device- the most discriminating features for re-identification.

To achieve such an objective we first partitioned the space consisting of all mobile devices by clustering it into three different groups on the basis of their capabilities. The amount of RAM, the frequency and the CPU model (e.g.dual core, quad core etc.) have been considered to manually cluster devices in: (1) low-performance devices (LPD), (2) medium-performance devices (MPD), and (3) high-performance devices

(HPD). This has been accomplished by first manually labeling a set of devices, then a multiclass SVM has been trained using the one-vs-all method. SVM parameters have been computed using 4-fold cross-validation.

Then, given a set of training images acquired by mobile devices, the image complexity and the device capabilities are used to compute the performance of each considered feature algorithm (see Sect. 5.5.2.2). In particular, we measure the feature extraction time, and a quality parameter (a distance value better discussed in Sect. 5.5.2.4) to decide whether a feature algorithm can be used or not. If the extraction times is too high or the quality is too low, the feature algorithm is considered as not usable on the device that is currently under inspection. It is usable vice versa. The so computed images complexity, the device type and feature type, form the input pattern in the training set. The label *usable (1) / not usable (0)*, represents the label that the classifier should return for such an input pattern. Once the training set is built, it is used to train a balanced neural tree [30] that is next exploited by the re-identification phase.

5.5.2.4 Re-identification Phase

During the re-identification phase, the image acquired by the mobile device camera is used to create the pattern that is input to the trained balanced neural tree such that the best feature algorithm for re-identification is chosen. Once the feature algorithm is selected, it is used to extract image features and match the current image with the signature of the suspicious person given by the camera network.

Feature Learning: The main goal of the learning is to automatically select the best feature algorithm for the specific type of mobile device making the request. This operation is required in two different situations: (1) when the client application starts and sends for the first time a frame to the server and (2) when the re-identification performance is not stable (i.e. accuracy is decreasing) over a given period of time. In a typical scenario, the client device sends to the server the first frame and information related to its performance (e.g.CPU model, RAM usage etc.). Once the server receives such information, the complexity of the image is computed and the client device is classified. A pattern containing the complexity of the images, the device classification and the type of feature algorithm to use is created and input to the neural tree classifier. The results of this operation is a packet containing the type of feature and the parameters that should be used by the client.

Matching metchanism: Once the client receives the configuration packet the selected feature is used to perform re-identification. This process is executed locally on-board of the device. Since the exploitable feature represents a subset of the suspect's signature, only the corresponding subset is used to match the device image with the suspect. To measure the distance between the features descriptors computed by the mobile device $feat^C$ and the feature descriptors of the suspicious person signature $feat^S$ we compute

$$\Phi(feat^C, feat^S) = 1 - \frac{1}{\sum_{c,s \in match} d(feat^C(c), feat^S(s))/(match + \varepsilon)} \quad (5.6)$$

where $d(\cdot,\cdot)$ is the L^2-norm distance between feature descriptors. ε is a small value that avoid dividing by zero, and $match$ is the set of matches such that the L^2-norm distance between feature descriptors is lower than a given threshold Th_{l2}. c and s represent the subset of features matched from $feat^C$ and $feat^S$ respectively. If the distance $\Phi(feat^C, feat^S)$ is higher than a given threshold Th, a system reconfiguration is required.

5.5.2.5 Experimental Results

In this section we report the performance of the proposed method. First, an analysis of the feature algorithm is given, then results of the re-identification are presented.

Experimental scenario: Tests have been performed using an Asus TF101 tablet (device A) and a Samsung Galaxy III smartphone (device B). Both the devices run Android O.S. version 4.2.1. For each of them two different videos (V1 and V2) that have 60 and 70 persons respectively, have been used. The video were acquired with a resolution of 358×255 pixels at 30 fps. To calculate which is the best detector algorithm for the device in use, the computational time required by each feature algorithm has been computed considering only the time required to detect and extract the keypoints from the given frame. Other image processing operations on the captured frame were not considered. The reference frames from the camera network have been acquired by an AXIS 213 PTZ and an AXIS 221 camera. The default configuration defined by the OpenCV libraries has been used for STAR, FAST, ORB and BRISK algorithm parameters.

Training: The first evaluation index taken into consideration to build the training pattern is the features extraction time. As shown in Fig. 5.10 for both video V1 and video V2, ORB (red lines) is the fastest algorithm in terms of computational times. The average frame is processed in 0.62 ms. FAST (green chequered) is the one achieving lowest performance with a processing time of about 0.81 ms.

The second evaluation index taken into consideration is the number of matching features. Table 5.3 presents the average number of features matched by devices A and B with the reference images acquired by the two cameras. Best average values are highlighted in boldface font. FAST algorithm gives the best results with an average of 71 matches. STAR and BRISK match respectively an average of 29 and 21 matches. ORB achieves the lowest performance with an average of 17 matches.

All such results are then used to form the train set for the balanced neural tree.

Re-identification results: During the re-identification phase the threshold Th_{l2} was set to 0.25 for both V1 and V2. Similarly Th was set to 0.29. These values were chosen by using 4-fold cross-validation.

In Fig. 5.11 the matching results of the first 30 frames of one person in video V1 captured using the mobile device A are shown. ORB features have been extracted

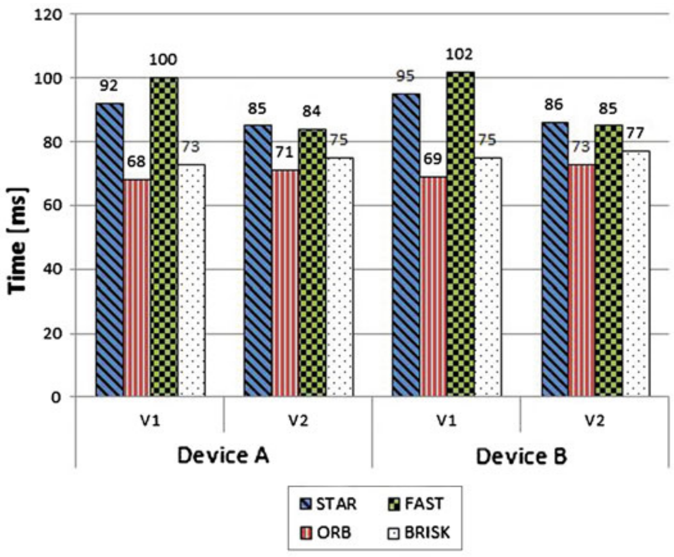


Fig. 5.10 Evaluation results for STAR, ORB, FAST, BRISK feature algorithms on the used devices

Table 5.3 Average number of matching features for both device A and B. The highest average number of matching features is highlighted in boldface font

	Feature type	Video 1	Video 2	Average
Device A	STAR	31.6	27.1	29.3
	ORB	19.1	15.6	17.4
	FAST	70.1	72.1	**71.1**
	BRISK	20.8	22.3	21.6
Device B	STAR	30.1	26.7	24.2
	ORB	18.8	14.6	16.4
	FAST	71.3	73.9	**72.1**
	BRISK	20.9	23.0	22.2

from the first 15 frames as instructed by the feature learning mechanism. Then, at frame 15, the distance value increases to 0.6 due the change of brightness. Such a value is greater than Th, thus a new configuration from the server was required. The device was then instructed to extract the FAST features. That results in a new a distance value of 0.17 between the reference frame and the current image.

Re-identification performance on video V2, where the same device A was used, are shown in Fig. 5.12. In this case two reconfigurations occurred. At frame 13, FAST feature detector has been replaced by ORB as the distance of 0.34 was higher than the threshold. Next, at frame 25 a change of the brightness forced another device reconfiguration. The STAR feature algorithm was then used.

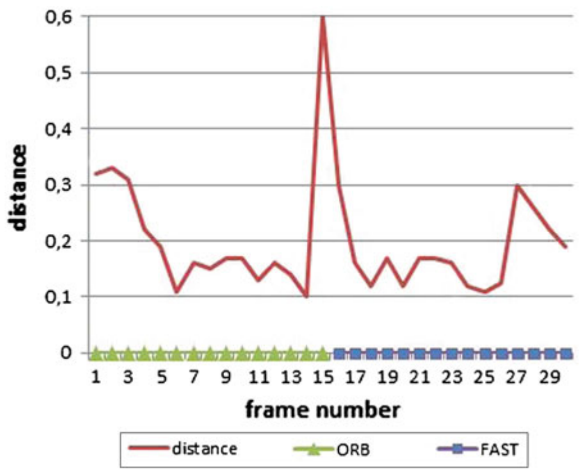


Fig. 5.11 Re-identification performance on video V1 using device A. The first 15 frames were acquired using ORB feature detector. After a change in image brightness, the device got reconfigured and instructed to use the FAST algorithm

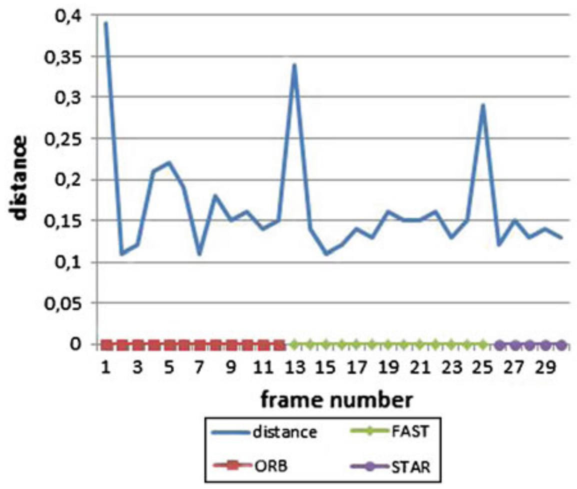


Fig. 5.12 Re-identification performance on video V2 using device A. Three changes of the feature detector was required. The average distance calculated was around the value of 0.15. Two peaks occurred respectively at frame 13 and 26

In Fig. 5.13a results of re-identification performance for device A are shown in terms of ROC curves. The feature algorithms in Sect.5.5.2.2 have been compared to the proposed adaptive feature algorithm. Results shows that the adaptive algorithm outperform all other methods with a true positive rate of about 80 % for a false

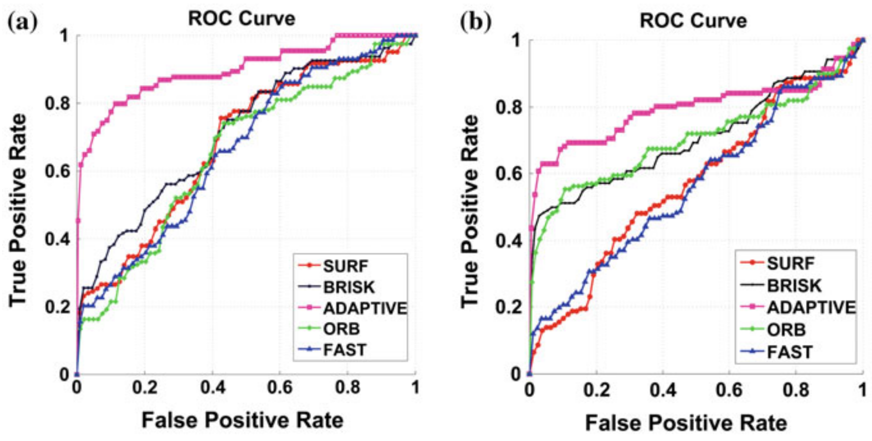


Fig. 5.13 Re-identification results in terms of ROC curves. The ROC curves have been computed using all the frames in V1. In **a** results are computed for device A. In **b** results are computed for device B

positive rate of about 20 %. For the same false positive rate, the other algorithms achieve a true positive rate of about 37 %.

In Fig. 5.13b, results of re-identification performance using device B are shown. As for device A, the proposed adaptive algorithm outperform all other algorithms. In particular, considering a false positive rate of about 40 %, the adaptive algorithm achieves a true positive rate of 80 %, while other methods have performance of 45 % on average.

5.5.2.6 Conclusion and Discussion

In this case study a client-server system for re-identification using smart devices has been introduced. The system allows to save network and computational resources by exploiting a feature learning mechanism. A training phase is performed to cluster devices on the basis of their capabilities and to train a classifier to select the best feature algorithm for a given device and image complexity. Such classifier is used in the re-identification phase to select which feature should be used to maximize the re-identification performance. Experimental results that the client-server method outperforms all other standard methods.

5.6 Smartphone Networks: What Next?

Camera networks have received increasing attention in recent years, in part due to their many uses in applications ranging from surveillance and security, smart spaces, urban monitoring, traffic management, etc. Another recent development is the diffusion of consumer devices with built-in cameras, e.g., mobile phones, tablets, and Google Glass. Currently, however, camera networks and these consumer devices occupy different use spaces. An obvious question then is whether it is possible to integrate these consumer devices into camera networks, treating these devices as sensor nodes within the larger camera network infrastructure. In fact, it should be technically feasible to integrate such consumer devices into more traditional camera networks and video surveillance infrastructures comprising primarily of stationary and pan/tilt/zoom (PTZ) cameras. These new camera networks will usher a whole new set of applications, from crowd-sourced and people-centric surveillance to participatory event recording.

However, none of the existing mobile vision applications and frameworks considers the possibility of multiple consumer devices working together towards common sensing (or imaging) tasks. Furthermore, these techniques do not explore the possibility of integrating a camera-enabled consumer device into existing (surveillance) camera networks. The work in [6] is noteworthy in that it hints at the possibility of setting up a smart camera network comprising mobile phones. This work also lists the key requirements for setting up such a camera network. Chiefly among these requirements is the cooperative sensing by the mobile phones comprising the network [6], however, fails to provide an in depth analysis of the technical challenges associated with (1) setting up smart camera networks using mobile phones and (2) integrating camera-enabled consumer devices into existing (surveillance) camera networks. The greatest shortcoming of the proposal outlined in [6], perhaps, is the fact that it assumes that mobile devices are stationary. It then essentially treats mobile cameras as traditional cameras, ignoring the most notable feature of mobile phones, i.e. mobility. The fact that mobile devices are not stationary makes setting up camera networks of these phones that much more interesting and challenging at the same time.

The advances in mobile computing, sensing hardware and communication technologies combined with the success of mobile vision have made it possible to stretch the concept of a "‘smart camera"’ to the extreme: *every smart device equipped with a camera (and may be other sensors) can serve as a node in a smart camera network*. Setting up ad hoc networks comprising camera-equipped consumer devices or integrating such devices into existing camera networks poses unique challenges, in addition to those already mentioned in Sect. 5.4, with regard to stand-alone mobile vision applications:

- These devices exhibit *rapid, jittery, fast and unconstrained motions*. Most existing computer vision algorithms cannot deal with imagery collected under such extreme motions.
- Depending on the application, it may be required that the (camera) nodes in a camera network establish a *common coordinate system*. In case of mobile devices, no existing calibration algorithms can be readily applied to estimate the extrinsic parameters of a mobile device, in part because the extreme and unknown motions these devices go through. The method presented in [25] employs four reference points (correspondences) whose locations are encoded by means of some location

sensor (such as, an ultra-sound Cricket receiver). Four low-resolution pictures (taken from different viewpoints) of a reference point are sufficient to estimate the location and the orientation of a visual sensor node. There are, however, several issues with this approach, including the requirement of having this large number of reference points to be present in the scene and the quality or accuracy of the calibration.

- For the reasons of the above point, the nodes within the camera network also *need a common clock* for the purposes of collaborative sensing. A common clock allows information captured at multiple nodes to be fused together to construct a more complete picture of the event in question. Even in the absence of fine-grained time synchronization, it is often desirable to (time) order lower-level events to exploit causal relationship and infer higher-order event detection. An interesting paper on this topic is [36] where the issues related to the time synchronization in ad-hoc wireless sensor networks are studied. An important result reported in [36] is that classical clock synchronization algorithms are unsuitable for wireless sensor networks for two reasons: (1) limited communication range of each node and (2) high mobility of these nodes. [36] addresses these problems as follows: it does away with synchronizing clocks at each node. Rather it generates timestamps using (unsynchronized) local clocks. These timestamps are shared between nodes. The timestamps are transformed to the local time of the receiving node. While this method works well for ad hoc wireless sensor networks, it might not be straightforward to deploy it on camera networks comprising mobile devices. As stated earlier, mobile devices can exhibit a very high degree of mobility, which can make it difficult to ensure that the assumption of connection between two nodes for the time necessary for the complete exchange of sync messages holds under all conditions.
- Mobile devices are unique in how these are used. A typical (camera) node in a camera network is subservient to the network. A mobile device, on the other hand, is there to serve the need of its user. The user might choose to use this device to act as a part of a larger camera network. Similarly, a user might remove this device from the camera network without any notice. A mobile device is not always on. Even if it is turned on, it may not be pointing the right direction, the user may not want to use it to record the events that are of interest to the camera network, the user may be doing some else that is totally unrelated to the sensing or processing requested by the camera network, etc.. This *dynamic availability of the phone camera* is a crucial challenge that must be addressed before we can integrate mobile devices into camera networks. One possibility is to devise ways to inform the user that the mobile device is needed by the camera network to carry out sensing and processing.
- Another challenge of using mobile devices in camera networks has to do with their motion profile. As mentioned elsewhere in this chapter, mobile devices exhibit unpredictable and extreme motions, so it is often not easy to estimate the network topology. Centralized processing, perhaps, represents a good starting point for estimating network topology when mobile devices are integrated into camera networks. Another option might be setup spatially-oriented clusters, where different

mobile devices are clustered together based upon their locations as estimated by GPS- or WIFI-based localization.

5.7 Conclusions

Mobile devices are an inseparable part of our society now. In particular, due to the advances in smart and micro technology, camera sensors are now a standard component in all mobile devices. This innovation has attracted both the research community and the industries that aim to build advanced applications that exploit the powerful features of mobile devices and combine them with the more advanced computer vision and image processing techniques. In this chapter, we contributed to the research in distributed smart cameras introducing the most relevant advancements in mobile computer vision, then we discussed the main challenges that the community for such real-time distributed smart systems is facing at this time. We also introduced two main research studies where computer vision techniques and distributed frameworks are used for mobile applications of augmented reality and security purposes. Finally, we discuss the next challenges in mobile vision where the mobile devices become part of a visual sensor network.

References

1. Agrawal M, Konolige K, Blas MR (2008) CenSurE: center surround extremas for realtime feature detection and matching. In: Forsyth D, Torr P, Zisserman A (eds) European conference on computer vision. Lecture notes in computer science, vol 5305. Springer, Berlin, pp 102–115. doi:10.1007/978-3-540-88693-8
2. Alahi A, Ortiz R, Vandergheynst P (2012) FREAK: fast retina keypoint. In: International conference on computer vision and pattern recognition, pp 510–517. doi:10.1109/CVPR.2012.6247715, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6247715
3. Amazon: SnapTell (2007) http://www.A9.com
4. Bay H, Ess A, Tuytelaars T, Van Gool L (2008) Speeded-up robust features (SURF). Comput Vis Image Underst 110(3):346–359. doi:10.1016/j.cviu.2007.09.014, http://linkinghub.elsevier.com/retrieve/pii/S1077314207001555
5. Bay H, Fasel B, Gool LV (2006) Interactive museum guide: fast and robust recognition of museum objects. In: International workshop on mobile vision
6. Bolliger P, Köhler M, Römer K (2007) Facet: towards a smart camera network of mobile phones. In: 1st international conference on Autonomic computing and communication systems, p 17
7. Bosch A, Zisserman A, Munoz X (2007) Image classification using random forests and ferns. In: International conference on computer vision, pp 1–8. doi:10.1109/ICCV.2007.4409066
8. Brown M, Lowe DG (2006) Automatic panoramic image stitching using invariant features. Int J Comput Vis 74(1):59–73. doi:10.1007/s11263-006-0002-3
9. Bruns E, Bimber O (2008) Adaptive training of video sets for image recognition on mobile phones. Pers Ubiquitous Comput 13(2):165–178. doi:10.1007/s00779-008-0194-3
10. Davison AJ, Reid ID, Molton ND, Stasse O (2007) MonoSLAM: real-time single camera SLAM. IEEE Trans Pattern Anal Mach Intell 29(6):1052–67. doi:10.1109/TPAMI.2007.1049, http://www.ncbi.nlm.nih.gov/pubmed/17431302

11. Fritz G, Seifert C, Paletta L (2006) A mobile vision system for urban detection with informative local descriptors. In: International conference on computer vision systems, pp 30–30. doi:10.1109/ICVS.2006.5, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1578718
12. Girod B, Chandrasekhar V, Chen D, Cheung NM, Grzeszczuk R, Reznik Y, Takacs G, Tsai S, Vedantham R (2011) Mobile visual search. IEEE Sig Process Magazine 28(4):61–76. doi:10.1109/MSP.2011.940881, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5888642
13. Google: Google Goggles (2009). http://www.google.com/mobile/goggles
14. Greene K (2006) Hyperlinking reality via phones. MIT Technology Review, USA
15. Gruber L, Richter-Trummer T, Schmalstieg D (2012) Real-time photometric registration from arbitrary geometry. In: 2012 IEEE international symposium on mixed and augmented reality (ISMAR), pp 119–128. doi:10.1109/ISMAR.2012.6402548
16. Hagbi N, Bergig O, El-Sana J, Billinghurst M (2011) Shape recognition and pose estimation for mobile Augmented Reality. IEEE Trans Vis Comput Graph 17(10):1369–79. doi:10.1109/TVCG.2010.241, http://www.ncbi.nlm.nih.gov/pubmed/21041876
17. Hall SP, Anderson E (2009) Operating systems for mobile computing. J Comput Sci Coll 25(2):64–71
18. Hoff WA, Nguyen K, Lyon T (1996) Computervision-based registration techniques for augmented reality. In: Casasent DP (ed) Intelligent robots and computer vision XV, 2904, pp 538–548, doi:10.1117/12.256311, http://proceedings.spiedigitallibrary.org/proceeding.aspx?
19. IOnRoad: iOnRoad. http://www.ionroad.com/
20. Kooaba: Kooaba (2007). http://www.kooaba.com
21. LeafSnap: LeafSnap. http://www.leafsnap.com/
22. Lepetit V (2008) On computer vision for augmented reality. In: 2008 international symposium on ubiquitous virtual reality, pp 13–16 (2008). doi:10.1109/ISUVR.2008.10, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4568635
23. Leutenegger S, Chli M, Siegwart RY (2011) BRISK: binary robust invariant scalable keypoints. In: 2011 International conference on computer vision, pp 2548–2555. doi:10.1109/ICCV.2011.6126542, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6126542
24. Li Z, Yap KH (2012) Content and context boosting for mobile landmark recognition. In: IEEE Sig Process Lett 19(8):459–462. doi:10.1109/LSP.2012.2203120, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6213072
25. Liu X, Kulkarni P, Shenoy P, Ganesan D (2006) Snapshot: a self-calibration protocol for camera sensor networks. In: IEEE 3rd international conference on broadband communications, networks and systems, 2006 (BROADNETS 2006), pp 1–10
26. Lowe DG (2004) Distinctive image features from scale-invariant keypoints. Int J Comput Vis 60(2):91–110. doi:10.1023/B:VISI.0000029664.99615.94
27. Martinel N, Micheloni C (2012) Re-identify people in wide area camera network. In: 2012 IEEE computer society conference on computer vision and pattern recognition workshops, pp 31–36. IEEE, Providence, RI. doi:10.1109/CVPRW.2012.6239203, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6583277
28. Martinel N, Micheloni C, Foresti GL (2013) Robust painting recognition and registration for mobile augmented reality. IEEE Sig Process Lett 20(11):1022–1025. doi:10.1109/LSP.2013.2279014, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6604402
29. Martinel N, Micheloni C, Piciarelli C, Foresti GL (2013) Camera selection for adaptiveHuman-computer interface. In: IEEE transactions on systems, man, and cybernetics: systems pp 1–1. doi:10.1109/TSMC.2013.2279661, http://www.sciencedirect.com/science/article/pii/S0893608011002693
30. Micheloni C, Rani A, Kumar S, Foresti GL (2012) A balanced neural tree for pattern classification. Neural Netw 27:81–90. doi:10.1016/j.neunet.2011.10.007
31. Micheloni C, Remagnino P, Eng HL, Geng J (2010) Intelligent monitoring of complex environments. IEEE Intell Syst 25(3):12–14. doi:10.1109/MIS.2010.85

32. Miksik O, Mikolajczyk K (2012) Evaluation of local detectors and descriptors for fast feature matching. In: International conference on pattern recognition. Tsukuba, Japan, pp 2681–2684. ISBN:978-1-4673-2216-4
33. Neurotechnology: Neurotechnology Sentisight. http://www.neurotechnology.com/sentisight.html
34. Nokia: Nokia Point and Find (2006) https://betalabs.nokia.com/trials/nokia-point-and-find
35. Qualcomm: Qualcomm Vuforia. http://www.qualcomm.com/solutions/augmented-reality
36. Römer K (2001) Time synchronization in ad hoc networks. In: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking and computing, pp 173–182. ACM (2001)
37. Rosten E, Porter R, Drummond T (2010) Faster and better: a machine learning approach to corner detection. IEEE Trans Pattern Anal Mach Intell 32(1):105–19, doi:10.1109/TPAMI.2008.275, http://www.ncbi.nlm.nih.gov/pubmed/19926902
38. Rublee E, Rabaud V, Konolige K, Bradski G (2011) ORB: an efficient alternative to SIFT or SURF. In: IEEE International conference on computer vision. Barcellona, Spain, pp 2564–2571. doi:10.1109/ICCV.2011.6126544, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6126544
39. Schroth G, Huitl R, Chen D, Abu-Alqumsan M, Al-Nuaimi A, Steinbach E (2011) Mobile visual location recognition. IEEE Sig Process Mag 28(4):77–89. doi:10.1109/MSP.2011.940882, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5888650
40. Seifert C, Paletta L, Jeitler A, Hödl E, Andreu JP, Luley P, Almer A (2004) Visual object detection for mobile road sign inventory. In: International symposium on mobile, human–computer interaction, pp 491–495. doi:10.1007/978-3-540-28637-0_63
41. Skrypnyk I, Lowe D (2004) Scene modelling, recognition and tracking with invariant image features. In: IEEE International symposium on mixed and augmented reality, pp 110–119. doi:10.1109/ISMAR.2004.53, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1383048
42. Takacs G, Chandrasekhar V, Gelfand N, Xiong Y, Chen WC, Bismpigiannis T, Grzeszczuk R, Pulli K, Girod B (2008) Outdoors augmented reality on mobile phone using loxel-based visual feature organization. In: Proceeding of the 1st ACM international conference on Multimedia information retrieval—MIR'08, p 427. doi:10.1145/1460096.1460165, http://portal.acm.org/citation.cfm?doid=1460096.1460165
43. Takacs G, Chandrasekhar V, Girod B, Grzeszczuk R (2007) Feature tracking for mobile augmented reality using video coder motion vectors. In: IEEE international symposium on mixed and augmented reality, pp 1–4. doi:10.1109/ISMAR.2007.4538838, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4538838
44. W3C: Mobile Web Initiative Device Description Working Group Charter (2005) http://www.w3.org/2005/01/DDWGCharter/
45. W3C: W3C Extensible multiModal annotation markup language (2009) http://www.w3.org/TR/emma/
46. Wagner D, Reitmayr G, Mulloni A, Drummond T, Schmalstieg D (2010) Real-time detection and tracking for augmented reality on mobile phones. IEEE Trans Visual Comput Graph 16(3):355–68 (2010). doi:10.1109/TVCG.2009.99, http://www.ncbi.nlm.nih.gov/pubmed/20224132
47. Xu L, Oja E (1993) Randomized hough transform (RHT): basic mechanisms, algorithms, and computational complexities. CVGIP Image Underst 57(2):131–154. doi:10.1006/ciun.1993.1009, http://linkinghub.elsevier.com/retrieve/pii/S1049966083710090
48. Yeh T, Tollmar K, Darrell T (2004) Searching the web with mobile images for location recognition. In: IEEE International conference on computer vision and pattern recognition, vol 2, pp. 76–81. doi:10.1109/CVPR.2004.1315147
49. You B, Lane ND, Chen F, Wang R, Chen Z, Bao TJ, Cheng Y, Lin M, Torresani L, Campbell AT (2013) CarSafe App: alerting drowsy and distracted drivers using dual cameras on smartphones. http://now.dartmouth.edu/2012/09/dartmouth-smartphone-app-targets-driver-safety/

Chapter 6
Autonomous Tracking of Vehicle Taillights and Alert Signal Detection by Embedded Smart Cameras

Akhan Almagambetov and Senem Velipasalar

Abstract An important aspect of collision avoidance and driver assistance systems, as well as autonomous vehicles, is the tracking of vehicle taillights and the detection of alert signals (turns and brakes). In this chapter, we present the design and implementation of a robust and computationally lightweight algorithm for a real-time vision system, capable of detecting and tracking vehicle taillights, recognizing common alert signals using a vehicle-mounted embedded smart camera, and counting the cars passing on both sides of the vehicle. The system is low-power and processes scenes entirely on the microprocessor of an embedded smart camera. In contrast to most existing work that addresses either daytime *or* nighttime detection, the presented system provides the ability to track vehicle taillights *and* detect alert signals regardless of lighting conditions. The mobile vision system has been tested in actual traffic scenes and the obtained results demonstrate the performance and lightweight nature of the algorithm.

6.1 Introduction

A large number of mobile tracking algorithms in existence today require a powerful centralized processing device (i.e. a full-featured computer) to perform processing on video data streamed from dedicated video cameras. This, however is not a power-efficient approach and requires significant equipment investment. In an embedded smart camera, the necessary computer vision operations are performed in their entirety on the microprocessor of an embedded smart camera board and none

A. Almagambetov (✉)
Norwich University, 158 Harmon Drive, Northfield, VT 05663, USA
e-mail: akhan@norwich.edu

S. Velipasalar
Syracuse University, 111 College Place, Syracuse, NY 13210, USA
e-mail: svelipas@syr.edu

of the captured frames are stored or transmitted. The camera can communicate event flags and text-based information (such as the position of detected vehicles) to other components of the automatic driver assistance system

Significant differences between existing work and the method presented in this chapter include:

- computationally lightweight tracking during the day and at night;
- reliable classification of vehicle alert signals (brake lights, turn signals);
- detection and counting of passing vehicles in neighboring lanes;
- sophisticated correction and recovery mechanisms, in conjunction with a Linear Kalman Filter and codebook for robust tracking;
- a sufficiently generalized algorithm for tracking vehicles with varying light configurations (single-red lights commonly found in American cars, as well as red-yellow segmented lights characteristic of European import vehicles);
- an algorithm that runs on the microprocessor of an embedded smart camera.

6.1.1 Background

As reported by the National Safety Council in 2009, about a third of all automobile accidents that occur in the U.S. constitute rear-end collisions, with 30 % of them resulting in severe injuries [20]. Due to this fact, various detection systems have become popular for use with advanced driver-assistance systems (ADAS) and potential autonomous vehicle applications (i.e. lead vehicle following, collision avoidance).

Out of the detection systems being researched and marketed at the present time [27], the ability of computer vision-based systems to provide visual data for other advanced applications make them appealing for use with taillight detection and tracking. With the development of embedded smart cameras capable of performing onboard processing and wireless communication, vision-based mobile tracking systems with decision capabilities have become a viable application. Other system types include radar-based [23] and laser-based [29] vehicle detection.

Currently, there exist strict UN [1] and US DOT [21] regulations governing vehicle signal light colors, proportions and configurations. Despite a seemingly wide range of variations among different car manufacturers, these regulations allow us to make reasonable assumptions regarding vehicle light colors and symmetry when developing the algorithm.

Most of the research on vision-based algorithms can be classified into the following two categories (features may be shared to increase reliability [22]):

1. **Local feature-based**: individual frame information is used, extracted via morphology [18] and color/intensity thresholds [26, 28]. Different color spaces may be employed (RGB [10], HSV [22], YCbCr [19], or Lab [4]); and
2. **Temporal information-based**: tracking one [12] or many [15] vehicles across many frames, with interference from other objects (typically encountered during

daytime [24]—a more challenging scenario). Computationally more intensive methods, such as mean-shift tracking and particle filters [6], can be used to provide more reliable results.

Currently, there exist daytime algorithms that focus on detecting vehicles via object detection methods like Haar and nighttime algorithms that focus on detection of lights through different threshold techniques. One significant drawback of these algorithms is that they *cannot operate during both daytime and nighttime*: the visual information available from a monocular camera at night is not sufficient for any state-of-the-art object detection or template matching algorithms; similarly, static-exposure nighttime detection approaches fail in the daytime.

The proposed algorithm is a lightweight and robust solution for tracking vehicle taillights and detecting the most common alert signals, such as turns and brakes; it is capable of processing live camera data on an embedded platform, with no user interference, despite varying and difficult lighting conditions. The main advantage of this algorithm over existing work [3, 7, 9, 22, 28] is the ability to track taillights and detect vehicle signals at night, as well as during the day—a challenging and computationally expensive task (as shown in [2, 5]).

6.2 Wireless Embedded Smart Camera Platform

CITRIC camera mote [8] is used for the implementation of the algorithms in this chapter. This device consists of an embedded smart camera and a detachable wireless transmitter (mote) for communicating event flags and other textual information wirelessly. One advantage of this system is that the image sensor is positioned close to the processor and all of the frames are processed internally, after which they are discarded. This eliminates any potential privacy concerns associated with tracking and reduces power utilization and bandwidth waste associated with sending uncompressed video feeds to a central processing device. The smart camera is powered by Embedded Linux.

6.2.1 CITRIC Camera Board

The CITRIC embedded smart camera board features a color image sensor, storage and operating memory, as well as a fixed-point microprocessor. It is capable of processing up to 15 frames per second (fps) with VGA (640 × 480) resolution and operating uninterrupted on 4 AA batteries for approximately 8 h (time varies due to the complexity of the algorithms being executed).

At the heart of the CITRIC camera board is a frequency-scalable general-purpose Intel XScale PXA270 fixed-point microprocessor (manufactured by Marvell) with supported frequencies of 208, 312, 416, 520 MHz. Included on the chip are 256 kB

SRAM for latency and power consumption minimization, a USB-Host and camera interfaces, and a wireless MMX co-processor that is capable of executing eight addition or four multiply-accumulate operations in parallel during a single cycle, which significantly speeds up graphics and image manipulation operations (multimedia encoding and decoding in particular). PXA270 supports Intel's SPEED STEP™ technology, which greatly reduces power utilization during sleep and idle cycles.

The CMOS color image sensor (OmniVision OV9655) is detachable from the camera board and is interfaced to the microprocessor using a native camera interface. It is capable of capturing at SXGA (1,280 × 1,024) or lower resolutions and it is sensitive for capturing in low-light conditions.

Within the same board and connected to the microprocessor are 64 MB (largest size supported by the PXA270) of 1.8 V mobile SDRAM that is used for storing temporary information during the run cycle, as well as 16 MB Intel NOR FLASH used for storing the execution code. As no images are stored during code execution and the operating memory gets cleaned after each processed frame, this amount of memory is sufficient for most computer vision algorithms.

During the experiments, the CITRIC camera board was powered through the USB interface (one of the available power options). No video or individual frames were transferred to the computer and all of the processing was done on-board.

6.3 System Design

The pseudocode for the proposed method is given in Algorithm 1. The individual functions are described in greater detail in subsequent sections.

6.3.1 Colorspace

The Y'UV colorspace was used for the embedded smart camera implementation, since the hardware is capable of providing a Y'UV video stream directly, which avoids a computationally expensive Y'UV-to-HSI conversion on the microprocessor. In the Y'UV colorspace, luminance (Y-channel) and color data (U- and V-channels) are separated, increasing the reliability of color thresholds. With separate color and brightness channels, it is possible to pick a narrow range of colors using the UV components, while still describing a wide range of color variations through the use of the Y component.

As a side note, due to the way the CMOS sensor is accessed during a frame grab operation, using RGB (GBR 4:2:2) causes fringing artifacts with high-speed objects in images stored by the smart camera.

Sections of the UV color plane representing taillight color during daytime and nighttime/dawn/dusk scenarios are shown in Fig. 6.1. The embedded smart camera implements Y'UV as Y'UV 4:2:2, where for every two chrominance (U or V) pixels, there are four luminance (Y) pixels. Pixel data is stored in a single vector.

Algorithm 1: MAIN ALGORITHM($Frame$)

```
while $NoInterrupt$
  do
    $Luminance_{avg} \leftarrow$ SCANFRAME($Frame$)
    if $Luminance_{avg} > Threshold_{day}$
      then $ColorThresholds \leftarrow Thresholds_{day}$
      else $ColorThresholds \leftarrow Thresholds_{night}$
    $Regions \leftarrow$ DETECTCANDREGIONS($Frame$)
    $Blobs \leftarrow$ MORPHOLOGY($Regions$)
    for $i = Blob_1 : Blob_N$
      do
        if $Blob_i.Area > Area_{acceptable}$
          then DISCARDBLOB($Blob_i$)
        for $j = Blob_1 : Blob_N$
          do
            if $i! = j$
              then
                comment: Symmetry tests
                comment: $Y$-coordinate = $Y$
                if $(Y_{Blob_i} \approx Y_{Blob_j})$
                 and $(Area_{Blob_i} \approx Area_{Blob_j})$
                  then $Pair_M \leftarrow$
                STOREPAIR($Blob_i, Blob_j$)
    for $k = Pair_1 : Pair_M$
      do
        comment: $Left$ = L, $Right$ = R
        3DHISTGENERATE($Pair_k.L, Pair_k.R$)
        if 3DHISTTEST($Pair_k$) $< Threshold_{Bhat}$
          then DISCARDPAIR($Pair_k$)
          else
            CODEBOOKSTORE($Pair_k$)
            KALMANFILTERING($Pair_k$)
    CORRECTIONMECHANISMS($Pair_k$)
    SIGNALDETECTIONLUMINANCE($Pair_k$)
    SIGNALDETECTIONAREA($Pair_k$)
```

6.3.2 *Intensity-Based Threshold Selection*

When the algorithm is first initialized, light candidates are automatically detected and filtered (without user intervention). Detection relies on soft color and brightness thresholds (U, V, and Y, respectively) in order to outline areas of potential light candidates. Since thresholds are used in this step, it is necessary to differentiate between lighting conditions (daytime *or* nighttime). Using one set of soft thresholds in all lighting conditions can result in unwanted behavior, such as over-saturation of the captured image.

Although a different set of thresholds is used for each of the two lighting conditions, the true novelty of the algorithm comes from the fact that the same processing steps can be used during both daytime and nighttime—something that similar algorithms do not do.

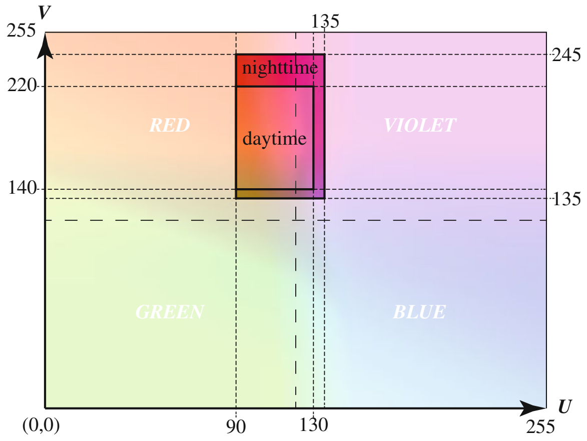


Fig. 6.1 Color (UV) regions selected for taillight representation

At the start of the algorithm, the entire frame is scanned and the average luminance level is determined. Based on this level, either "nighttime" or "daytime" thresholds are used for a set interval, during which it is assumed that no drastic lighting changes will occur. The luminance level is periodically re-evaluated in order to verify that the lighting conditions have not changed. Since the brightness levels of nighttime, dusk, and dawn lighting conditions are very similar, they are grouped into one category of thresholds; daytime conditions are placed into a separate category.

The thresholds were empirically obtained by analyzing taillight information from a database of over 400 images taken at various times throughout the day (to account for changing lighting conditions). The resulting analysis provided broad daytime and nighttime soft thresholds in the Y'UV colorspace.

6.3.3 Candidate Light Pair Identification

After the average luminance for the first frame is calculated and appropriate soft thresholds are selected, the algorithm detects potential taillight pairs to be tracked. This step is automatic, requiring no user input. At this stage, soft color thresholds are used to avoid eliminating too much information from the image—the resulting false positive regions do not constitute a problem, since they are eliminated during later stages of the algorithm. The image obtained in this step is run through a number of tests before final light candidates are selected. Figure 6.2 provides an illustration and explanation of this step.

Newer LED taillights and their associated 'beat frequencies' [due to duty-cycle decrease through pulse-width modulation (PWM)] do not affect the detection of potential taillight pairs.

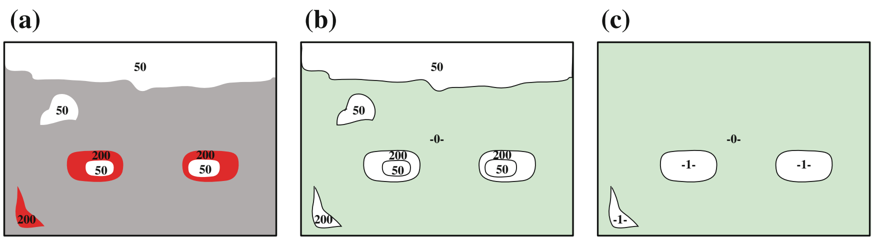


Fig. 6.2 A graphical illustration of the initialization process. **a** Thresholding. **b** Detected regions. **c** Grouping and cleanup

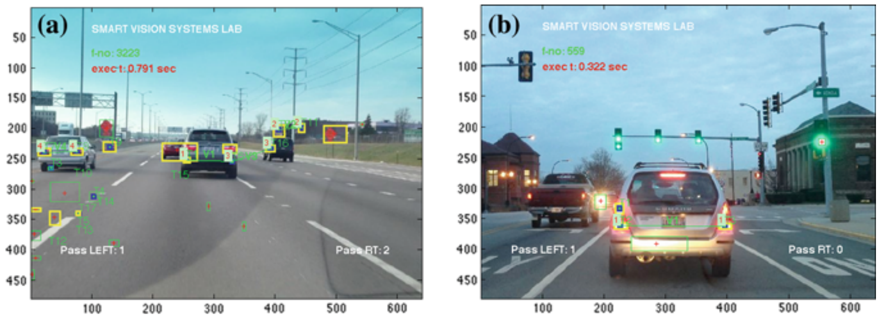


Fig. 6.3 Dawn and Dusk scenarios. Note the high number of false-positives (due to relaxed color thresholds), which are later successfully eliminated by the algorithm. **a** Dawn scenario. **b** Dusk scenario

6.3.3.1 Candidate Region Detection

Potential light candidates are detected using a set of soft color thresholds for red and white/yellow colors, most commonly found in vehicle taillights, shown in Fig. 6.2a. Using soft color thresholds often results in a high number of false-positives, which are eliminated using symmetry and histogram tests, as well as through the use of tracking and correction mechanisms (Fig. 6.3a). Both white/yellow- and red-containing areas are assigned non-zero values, while other areas are assigned a zero-value (Fig. 6.2b).

In order to eliminate areas that may not be the taillights of a vehicle, areas that are white/yellow and are not adjacent to any red regions are eliminated, while other areas are preserved and converted into a binary image (Fig. 6.2c). Morphological operations (*closing* followed by *opening*) are applied to the remaining areas to generate "blobs". A bounding box is drawn around each "blob" and centroid coordinates are computed. Steps (a–c), as applied to a video capture, are shown in Fig. 6.4.

For computational efficiency purposes, the area of each potential light candidate is verified in order to eliminate any regions that fall outside of the acceptable light dimensions (specified as a percentage of the captured frame area in order to make the algorithm scalable for higher resolutions). This step also eliminates large areas, which may pass through the soft color thresholds and do not represent actual vehicle

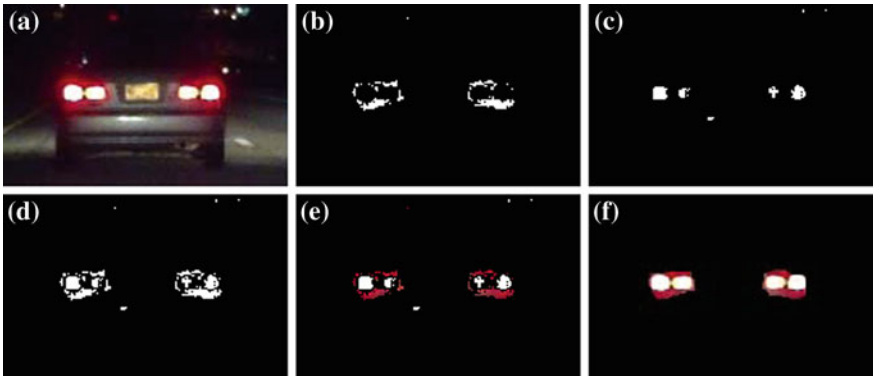


Fig. 6.4 Illustration of the various candidate light processing stages. **a** Captured frame. **b** Red components. **c** White components. **d** Red/white mixed. **e** Segmented lights. **f** Morphology

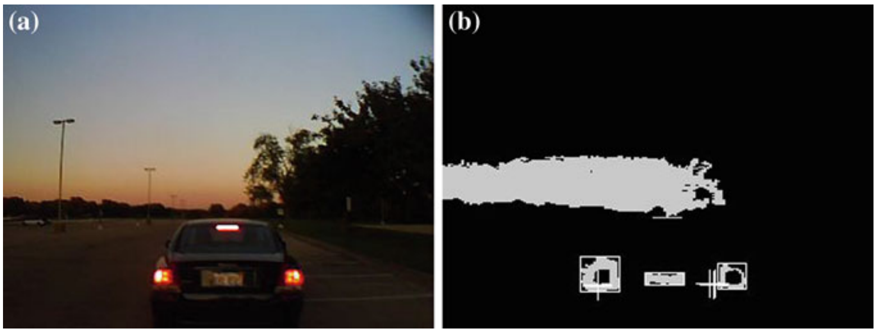


Fig. 6.5 Examples of interference by the sky (dusk). The top non-taillight region can be eliminated by the area verification or symmetry tests and results from the use of soft thresholds. **a** Captured image. **b** Processed image

taillights (i.e. red cars, sunset-lit skies, etc). An example of this type of situation is given in Fig. 6.5. When the areas that fall outside of normal light sizes are discarded, the resulting image processing becomes faster, as explained in the *Symmetry Verification* section. It should be noted that this step is without loss of generality and for efficiency purposes only, since such areas can be eliminated by the proposed algorithm using two additional tests, discussed below (namely symmetry and 3D histogram tests—the final stage of frame cleanup prior to the detected "blobs" getting stored into the codebook and initialized as new Kalman Filter trackers).

6.3.3.2 Symmetry Verification

The distance between "master" and "slave" blob centroids is calculated along the Y-axis (Fig. 6.6); a pair is considered to satisfy the symmetry test if the Y-direction

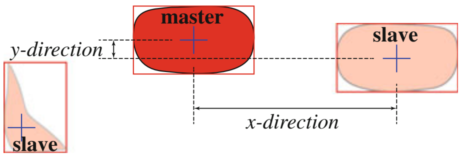


Fig. 6.6 Y-direction symmetry test

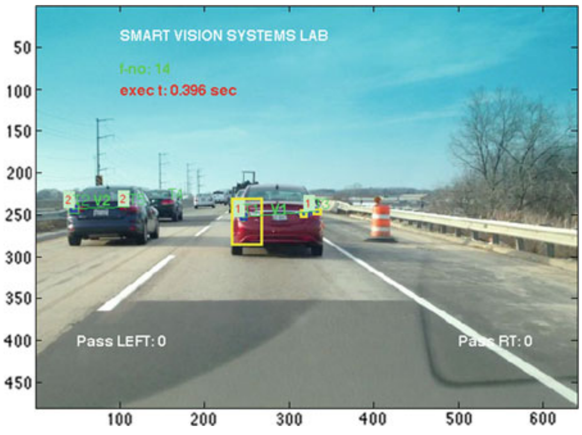


Fig. 6.7 Elimination of a *red* vehicle chassis using symmetry. The *red* chassis is asymmetrical to other objects in the frame

distance is less than the height of the "master" blob, in addition to the area of the slave blob being ±25 % of the master blob. Symmetrical pairs are kept in memory, while others are discarded.

The symmetry test is performed between all pairs of the light candidate "blobs". Since both leading and following vehicles are usually on similar terrain, it is safe to assume that potential vehicle taillights are symmetrical in the Y-direction. This aids with the elimination of red car bodies (the chassis of a red vehicle is asymmetrical to other objects in the frame), as shown in Fig. 6.7.

The total number of pairs that are tested can be expressed as $N(N-1)/2$, in effect a complete graph K_N, where N represents the number of blobs present at the start of the symmetry test. Symmetry tests are run until all $N(N-1)/2$ blob combinations are tested. Symmetrical pairs are individually stored as "left" and "right" lights; the labels are determined by the location of lights relative to each other on the X axis.

6.3.3.3 3D Histogram Test

In special circumstances (especially in high-traffic areas), clusters of brake lights or other artifacts in adjacent lanes may pass symmetry tests and be considered

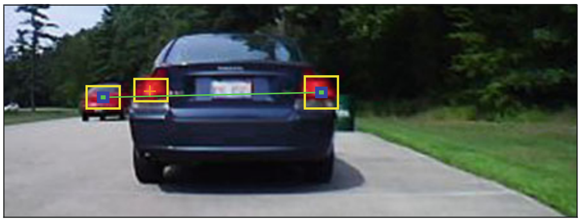


Fig. 6.8 Incorrect detection of "symmetrical" light pairs without the use of 3-dimensional histograms (*red* passing car and right taillight)

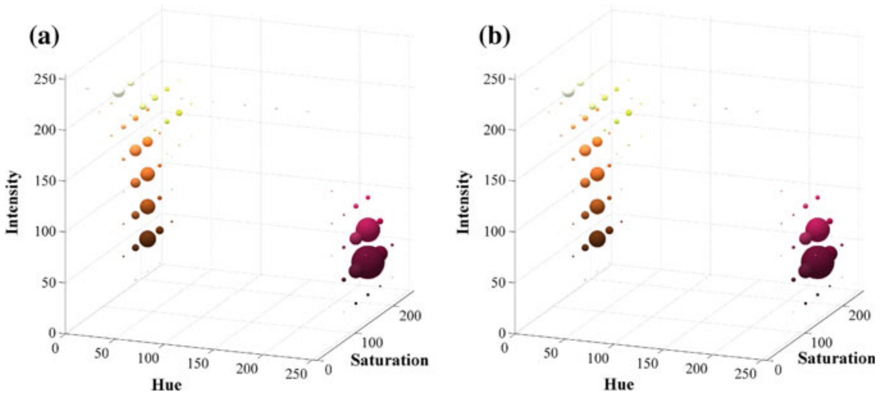


Fig. 6.9 A sample set of 3D histograms for a taillight pair. **a** Left taillight. **b** Right taillight

"symmetrical pairs". One example is shown in Fig. 6.8, where a red passing vehicle is detected and passes the symmetry test with the right taillight.

To mitigate these possible errors, color information is used to construct a 3D histogram for both left and right lights. Each monochrome color channel is binned into 8 bins, with $m = 512$ bins ($8 \times 8 \times 8$) representing each light. A sample 3D histogram for a taillight with an active brake signal is given in Fig. 6.9. Resulting histograms from "left" and "right" lights are compared using the Bhattacharyya coefficient (Eq. 6.1) [11]. If the coefficient value is higher than an empirically determined threshold, a match is declared. The empirical threshold was carefully determined by analyzing several dozen video sequences and the same value is used for all lighting conditions.

The Bhattacharyya coefficient is:

$$\hat{\rho}(y) \equiv \rho\left[\hat{\mathbf{p}}\left(\mathbf{y}\right), \hat{\mathbf{q}}\right] = \sum_{u=1}^{m} \sqrt{\hat{p}_u\left(\mathbf{y}\right), \hat{q}_u}, \tag{6.1}$$

where $\hat{\mathbf{q}} = \left\{\hat{q}_u\right\}_{u=1\ldots m}$ and $\hat{\mathbf{p}}\left(\mathbf{y}\right) = \left\{\hat{p}_u\left(\mathbf{y}\right)\right\}_{u=1\ldots m}$ represent probabilities calculated by normalizing the m-bin 3D histogram for each light.

Table 6.1 Data stored for each tracker

Tracker	
$x_{min}, x_{max}, x_{centroid}$	4/4/4 bytes
$y_{min}, y_{max}, y_{centroid}$	4/4/4 bytes
3-dimensional histogram*	2,048 bytes
Average intensity level	4 bytes
`TrackerID`	4 bytes

*previous frame only

The mismatched pairs in Fig. 6.8 produce a low Bhattacharyya coefficient and would not constitute a valid taillight "pair".

6.3.4 Tracking and Codebook Update

6.3.4.1 Codebook

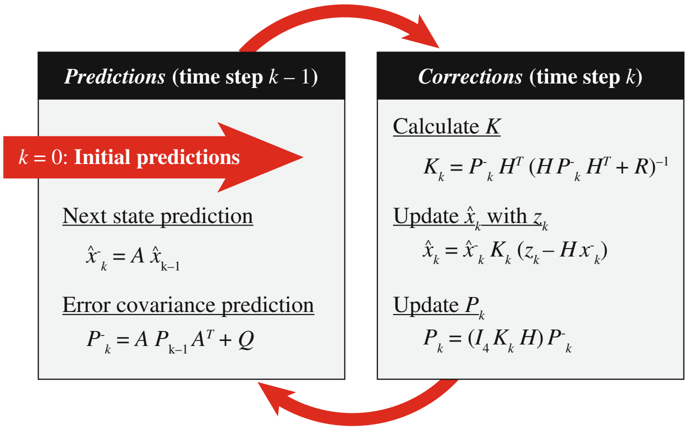
To make a provision for effective correction mechanisms (discussed in later sections), as well as to accurately detect vehicle alert signals, a codebook is maintained throughout the execution of the algorithm for each tracker (data is kept separate for left and right lights). The data contained in each tracker are shown in Table 6.1.

6.3.5 Kalman Filter Tracking

Any application using system models is inaccurate to some degree, as models can seldom account for measurement noise and dynamically adjust to varying measurements or other outside changes [13]. The use of a linear Kalman filter (LKF) has the advantage of generating next state predictions by carefully weighting the current system state with the prediction, instead of relying on a static model. The weights placed on the current state and the prediction can be adjusted by adjusting the filter 'gain' ($K = 0$ uses only the predictions, $K = 0.5$ equates to simple averaging, and $K = 1$ uses only the current state to make the predictions), as outlined in Eq. (6.2) [14, 16, 25]. Another advantage of using LKF is that the entire system state history does not need to be known in order to generate a prediction (only the current state and the current prediction are needed to predict system behavior in the next state).

$$x_k = Ax_{k-1} + w_{k-1}, \tag{6.2}$$

where x_k is the estimated next state, x_{k-1} is the current system state, A is a 4×4 movement matrix that expresses how the system state changes from $k - 1$ to k, and



Predictions (time step $k-1$)
$k = 0$: Initial predictions
Next state prediction
$\hat{x}^-_k = A\,\hat{x}_{k-1}$
Error covariance prediction
$P^-_k = A\,P_{k-1}A^T + Q$
Corrections (time step k)
Calculate K
$K_k = P^-_k H^T (H P^-_k H^T + R)^{-1}$
Update $\hat{x}_k$ with z_k
$\hat{x}_k = \hat{x}^-_k K_k (z_k - H x^-_k)$
Update P_k
$P_k = (I_4 K_k H) P^-_k$

Fig. 6.10 Linear Kalman Filter (*LKF*) operation. Initial values consist of $\hat{x}_{k-1}$ and P_{k-1}

w_{k-1} is the process noise that is assumed to be Gaussian for the purposes of this application. Note that there is no control input u, therefore the term Bu_{k-1} is set to zero in Eq. (6.2). All of the necessary matrices are given in the appendix.

LKF can reliably track and predict the future position of an object at time $t+1$ (x_k), preventing the loss of objects due to inconsistent detection from frame to frame (i.e. momentary occlusion of the field-of-view). The LKF tracks the centroid $[x, y]$ of an object, hence the states of the Kalman filter are the x and y positions, (pos_x, pos_y), as well as the velocity, (v_x, v_y).

In the measurement calculation outlined in Eq. (6.3), the measurement noise v_k is also assumed to be Gaussian.

$$z_k = Hx_k + v_k \tag{6.3}$$

The operation of the Kalman filter is given in Fig. 6.10. After the filter is initialized with arbitrary values $\hat{x}_{k-1}$ [Eq. (6.4)] and P_{k-1}, the LKF alternates between time update (prediction) and the measurement update (correction). Over time, the calculated Kalman filter gain K and the estimation error covariance P_k will become constant, therefore arbitrary values are sufficient for initializing the Kalman filter.

$$\hat{x}_{k-1} = \left[\frac{width_{image}}{2}, \frac{height_{image}}{2}\right] = [x, y] \tag{6.4}$$

As can be seen from Fig. 6.10, at filter initialization, arbitrary values of $\hat{x}_{k-1}$ and P_{k-1} are used as inputs to the system. From this point, the Kalman filter alternates between the prediction and correction states. The predictions for the system state $\hat{x}^-_k$ and estimation error covariance P^-_k are generated at time $k-1$ (before the actual measurements are known, hence a superscripted minus sign is used). At time k, the Kalman filter gain K is calculated, after which the system state $\hat{x}_k$ and the estimation

Table 6.2 Data stored for each Vehicle Object

Vehicle Object	
`TrackerID` $_{leftLight}$	4 bytes
`TrackerID` $_{rightLight}$	4 bytes
Consecutive tracking failures	4 bytes
Removal flag*	4 bytes

*initialized to zero

error covariance P_k are updated with the measurement values. The weighting of the prediction to the measurement is chosen based on the calculated value of gain K, which dynamically adjusts itself for optimal performance.

Every object that needs to be tracked is tracked using the Kalman filter. Due to the nature of the tracking method, the Kalman filter performs well when tracked objects are partially or entirely occluded (which is the case with taillight tracking). LKF is able to track centroids of lights identified as a "working pair" (lights that pass both symmetry and 3D histogram tests). Once the error between the predicted centroid coordinates and the centroids of detected light candidates becomes too large, the correction mechanisms are engaged. If the tracker cannot lock back on the target, it is discarded.

6.3.6 Vehicle Object Structure

Trackers for light pairs that were matched using both symmetry and 3D histogram tests are stored within a Vehicle Object (VO) structure, containing data in Table 6.2.

The VO structure simplifies the maintenance of trackers in memory: once the number of consecutive tracking failures for the VO exceeds the frame-rate equivalent of 2 s, the VO is destroyed and "orphaned" trackers are cleared.

All possible VOs are tracked (even if trackers are shared), which yields a higher overall reliability. Erroneously detected VOs are destroyed, leaving only VOs corresponding to actual vehicles in Fig. 6.11. **V1** is a VO created for vehicle on right, consisting of trackers `T1` and `T4`, while **V2** is a VO for vehicle on left, consisting of `T2` and `T3`. **V3**, consisting of `T1` and `T2`, is created (Fig. 6.11b) and subsequently destroyed (Fig. 6.11c).

6.3.6.1 Passing Vehicle Counter

The approximate number of vehicles passing on the left and right can be determined by counting valid VO destruction events and referencing the last-good-known location of the VO. A VO is considered valid only if it can be reliably tracked for a frame-rate equivalent of at least 2 consecutive seconds. If both lights within the

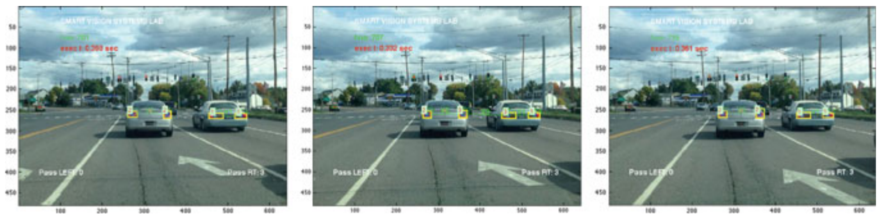


Fig. 6.11 Daytime (cloudy) scenario, demonstrating the creation and destruction of VOs

VO were on the left side of the frame when the VO was destroyed, the "Pass Left" counter is incremented. If both lights within the VO were on the right, a "Pass Right" event is registered. If the lights within the VO happen to be located on different sides of the frame when the VO was destroyed, no pass is counted, as it is assumed that the vehicle accelerated and was "Lost" (these events are counted, however the output is not displayed). These counters provide valuable information, since vehicles can act as mobile probes and provide traffic information from locations that are out of reach of static sensors.

6.3.7 Correction Mechanisms

Three correction mechanisms are used, which take effect in extraordinary situations, in order to prevent the corruption of the codebook or the divergence of the Kalman filter. The pseudocode for these correction mechanisms is given in Algorithm 2.

6.3.7.1 Distance Tracking

This mechanism prevents corruption of the codebook and the divergence of the Kalman filter by eliminating erroneous data (Fig. 6.12).

When multiple potential light candidates are detected (and identified as "pairs"), the predicted distance between trackers *dp* is compared with the distance between detected lights *d* to determine which of the light candidates are actual lights.

Algorithm 2: CORRECTIONMECHANISMS(dp_{t-1}, L, R, C_n)

```
comment: Left = $L$, Right = $R$, Candidate = $C$
procedure DISTANCETRACKING($dp_{t-1}, L, R, C_n$)

  $d_n \leftarrow \text{dist}(L, C_n)$
  for $i \leftarrow 1$ to $n$
    do $dist(i) \leftarrow \text{abs}(dp_{t-1} - d_{t,i})$
  $index \leftarrow \min_{index}(dist)$
  $R \leftarrow C_{index}$

procedure KALMANFILTERERRORCORRECTION($L, R$)

  if KALMANERROR($L_t, L_{t,predicted}$) $>> Threshold_{Kalman}$
    then { if 3DHISTTEST($L_t, L_{t-1}$) $> Threshold_{Bhat}$
             then $L_t \leftarrow L_{t-1}$

procedure TESTFAILURECORRECTION($L, R$)

  if 3DHISTTEST($L, R$) $= fail$ or SYMTEST($L, R$) $= fail$
    then { if 3DHISTTEST($L_t, L_{t-1}$) $> Threshold_{Bhat}$
           and 3DHISTTEST($R_t, R_{t-1}$) $> Threshold_{Bhat}$
             then { $L_t \leftarrow L_{t-1}$
                    $R_t \leftarrow R_{t-1}$
```

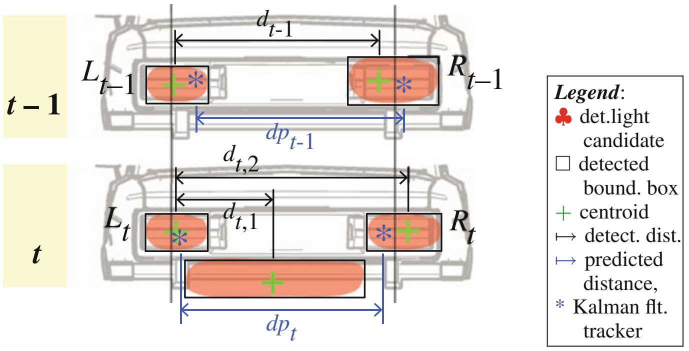


Fig. 6.12 Distance tracking correction mechanism

For example, at time t, there are three light candidates. By comparing the predicted distance from the codebook, dp_{t-1}, with the distance between light candidates $d_{t,1}$ and $d_{t,2}$, it is possible to eliminate the mis-detected candidate and prevent codebook and Kalman filter tracker corruption.

6.3.7.2 Kalman Filter Error Correction

This scenario occurs when both symmetry and 3D histogram tests are passed by the light candidates, but the Kalman filter error between the candidate location and the prediction is too large.

For example, if the left light L_t at time t passes both symmetry and 3D histogram tests, but has a large error between the detected light centroid location and predicted location, the 3D histogram for light L_{t-1} from the codebook is used and the 3D histogram test is run between the histograms of L_t and L_{t-1} (from codebook). If the 3D histogram test is satisfied, then the codebook is updated with the information from L_t and the Kalman filter tracker integrates the position information from L_t. If the candidate light fails the 3D histogram test, it is discarded. A similar procedure is performed if right light R_t causes a large Kalman filter error.

This particular correction is performed in order to make sure that the Kalman filter is provided with a new measurement at each frame. When the linear Kalman filter is applied to images from a static camera, it can compensate for the lack of measurements (prediction over measurement). In the case of a mobile camera, however, a drastically incorrect prediction at $t + 1$ would greatly increase the probability of the Kalman filter diverging.

6.3.7.3 Test Failure Correction

This mechanism is engaged when either the symmetry or the 3D histogram test is failed by one of the candidate lights at time t.

Two 3D histogram tests are run: one between detected candidate light L_t and codebook information stored for L_{t-1}, and the other between candidate light R_t and codebook information for R_{t-1}. If both tests are passed, then the codebook is updated with new light data and the position information is integrated into the Kalman filter tracker. If these “lookback” tests are failed by at least one light, this data is discarded.

6.3.8 Alert Signal Detection

There are two ways of detecting vehicle alert signals: intensity tracking (applicable to all vehicle makes and models) and area tracking (mostly foreign cars with segmented lights).

6.3.8.1 Intensity Tracking

The average intensity of each light is monitored over time. Bounding box coordinates for lights that passed all required tests (or were corrected) are used to extract intensity data for each light.

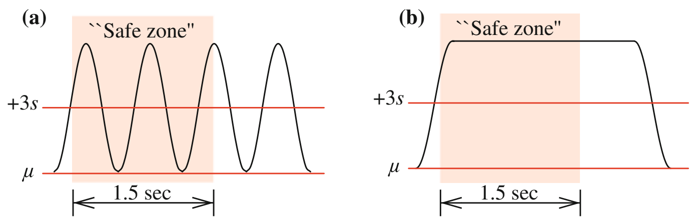


Fig. 6.13 Alert signal detection algorithms. **a** Turn detection. **b** Brake detection

A standard deviation of the luminance is computed for each light and its value is updated at every frame [17], as given in Eq. (6.5), which can be converted to a running standard deviation, given in Eq. (6.6), where p is the power sum average, μ is the running average (mean), and N is the total number of elements.

$$s = \sqrt{\frac{N \sum_{k=1}^{N} x_k^2 - \left(\sum_{k=1}^{N} x_k\right)^2}{N(N-1)}} \tag{6.5}$$

$$s = \sqrt{\frac{N\left(p - \mu^2\right)}{N-1}} \tag{6.6}$$

If the intensity level exceeds ± 3 s around μ, the mean and standard deviation values are locked against updating and the light is declared to be “on”. If overall lighting conditions change, the lock is subsequently released.

The detection algorithm features a “safe zone” (marked in light-red in Fig. 6.13, which illustrates the detection approaches). This detection zone is equal to a frame-rate equivalent of 1.5 s, during which no decision regarding braking is made. If the braking action (both lights ‘on’) is still detected after that time, then the system records the *brake* signal. Similarly, if one of the lights goes through two complete cycles within the “safe zone”, a *turn* is recorded (either a *right* or *left*, based on the light that cycled).

Figure 6.14a–c show the intensity levels for both lights over a number of frames, their patterns corresponding to a left turn, right turn, and braking, respectively. The distinctive pattern for each action can be clearly seen in all of the figures.

6.3.8.2 Area Tracking

Another algorithm runs alongside intensity tracking to prevent mis-detection of alert signals. Vehicle light shapes and sizes vary from manufacturer to manufacturer and while most domestic cars have one set of lights for signaling turns and braking, most foreign automobiles have segmented lights. Figure 6.15a demonstrates that turn lights are located *under* the main set of lights that are tracked, potentially preventing the turn light from being detected using the intensity tracking method.

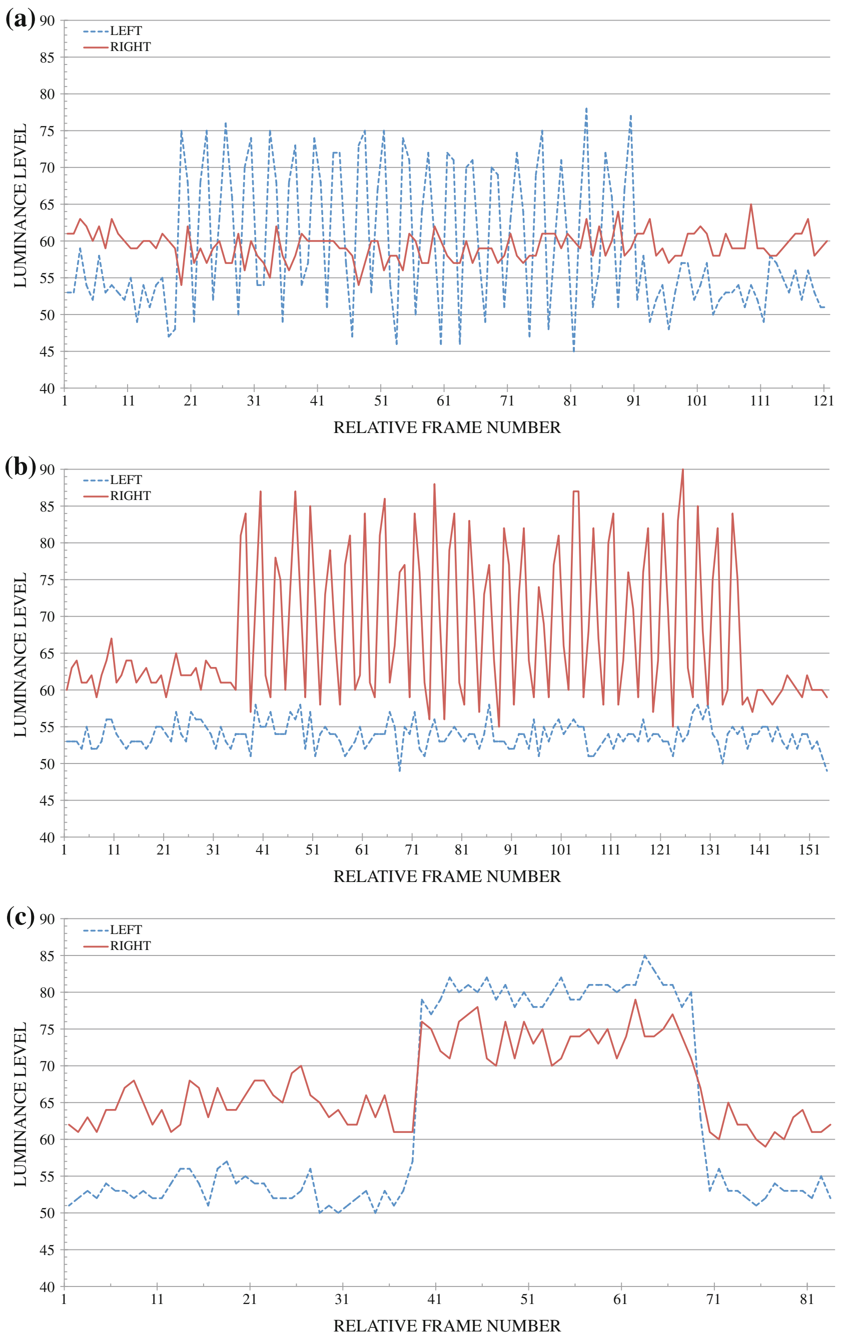


Fig. 6.14 Luminance level for both lights. **a** Left turn. **b** Right turn. **c** Braking

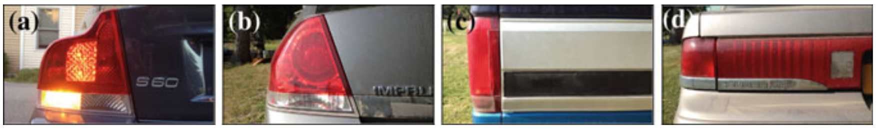


Fig. 6.15 Examples of domestic non-segmented and foreign segmented taillights. **a** Volvo turn. **b** Chevrolet Impala. **c** Ford F-150. **d** Mercury Cougar

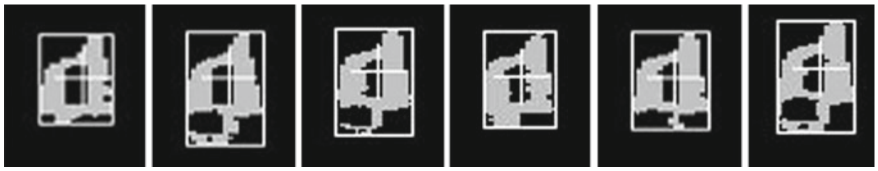


Fig. 6.16 Area change through one compete cycle of a turn signal

The light area changes in unison with the turn signal being on or off, one compete cycle of which is shown in Fig. 6.16. This fact is used for tracking the area of the light over time to determine its state. The same set of rules from intensity tracking (Fig. 6.13) is applied to area tracking, resulting in successful detection of turn signals regardless of light configuration. Intensity and area tracking methods are combined via a logical `OR` statement for robust signal detection. Area tracking plays an important role during nighttime detection, since the size of the lights increases when brakes are applied or turn signals are engaged.

6.4 Main Results

6.4.1 Results on Recorded Video

The initial algorithm was tested on recorded video sequences in MATLAB. Algorithm steps and approaches were then adjusted before porting it to the embedded camera platform. Tested scenarios are shown in Table 6.3. As videos were recorded on the highway, the occurrence of alert signals is not very frequent.

6.4.1.1 Colorspace

Unlike the embedded camera implementation, the recorded videos were processed in the HSI (hue, saturation, intensity) colorspace.The HSI colorspace is adapted for computer vision algorithms and it separates the intensity and saturation components from hue (color) information. The Y'UV thresholds that were empirically derived from evaluating a database of 400 vehicle taillight pairs were converted to HSI for use in this part of the experiments.

Table 6.3 Recorded video results for a variety of lighting conditions

Scene type	Totals		t_{avg}	Left pass				Right pass				Left turn				Right turn				Brake			
	frames	t (s)	(ms)	gt	det	rate	f+	gt	det	rate	f+	gt	det	rate	f+	gt	det	rate	f+	gt	det	rate	f+
Sunny	12,590	839	391	0	1	—	1	10	7	0.70	0	1	1	1.00	0	1	1	1.00	0	7	7	1.00	0
Cloudy	15,390	1,026	354	23	17	0.74	0	18	14	0.78	0	7	4	0.57	0	6	4	0.67	0	18	16	0.89	0
Dusk	4,931	328	355	0	3	—	3	0	0	—	0	1	1	1.00	0	0	0	—	0	3	3	1.00	0
Night	4,497	300	453	0	0	—	0	1	1	1.00	0	1	1	1.00	0	4	3	0.75	0	0	0	—	0
Dawn	4,790	319	791	8	9	1.00	1	3	6	1.00	3	0	0	—	0	4	4	1.00	0	0	0	—	0

t_{avg} = average processing time; gt = 'ground truth', the number of events observed
det = number of detected events; rate = detection rate; f+ = false positives

6.4.1.2 Discussion

Sunny: total of 839 s (13.98 min), with all vehicles passing on the right. As seen in Table 6.3, the signal detection rate (turns and brakes) for this scenario is excellent—being 100 % for all signals. Out of 10 actual passing vehicles, 7 were detected correctly. The left passing category had a single false-positive detection, which may have been due to orange construction cones appearing in view.

Cloudy: total of 1,026 s (17.10 min), with some passing vehicle mis-detections on both sides. Figure 6.11 shows creation and destruction of Vehicle Objects.

Dusk: total of 328 s (5.47 min). As seen in Table 6.3, the signal detection rate for this scenario is also 100 % for left turn and brake signals. There were no right turn signals observed. No passing vehicles were observed due to the target vehicle being on a one-lane highway. Although there were no passing vehicles, the left counter was incremented by three, possibly due to vehicles turning left at intersections.

Night: least challenging scenario in terms of tracking and detection, since it is a nighttime video. Signal detection rate is high. Two vehicle objects are created and tracked for 300 s (5 min), with 1 vehicle passing on right.

Dawn: total of 319 s (5.32 min). All the right turn signals were detected successfully. There were no instances of left turns and brakes. Passing vehicles present on both left and right, with 1 false-positive for the left passing zone and 3 false-positives for right-side passing. A large number of regions were detected as possible light candidates, due to use of soft color thresholds at initialization (and pinkish hue on concrete from rising sun). Most erroneous light candidates are eliminated. Average execution time per frame is significantly higher due to active correction mechanisms and tests for eliminating erroneous light candidates.

6.4.2 Embedded Smart Camera Implementation Results

In order to provide a better illustration of the algorithm, the results are broken down into two sections:

- Section 6.4.2.1 provides the results for vehicle taillight **tracking only**, meaning that there is no recognition of vehicle alert signals; and
- Section 6.4.2.2 provides the results for a **full implementation** of the algorithm, complete with the detection of vehicle alert signals.

6.4.2.1 Tracking Only

Experiments to test the algorithm on the embedded smart camera were performed in five lighting conditions, with results obtained from the experiments summarized in Table 6.4 and described in this section. No color captures were written to FLASH, as writing color JPEG images to memory takes approximately 53 ms/cycle.

Table 6.4 Tracking-only algorithm results on an embedded smart camera

Scene	t_{total}	# frames	Detection failure		LKF tracker	$t_{frame}(ms)$		
description	(s)		One light (%)	Both lights (%)	update rate (%)	avg	min	max
Dawn	257	1,500	30.733	0.0667	69.200	171.33	130	189
Sunny	199	1,097	20.146	—0—	79.763	181.40	131	191
Cloudy	224	1,320	22.121	4.0150	73.788	169.70	131	191
Dusk	252	1,445	18.685	1.8690	73.080	174.39	129	189
Night	305	1,598	3.2541	—0—	97.935	190.86	131	191

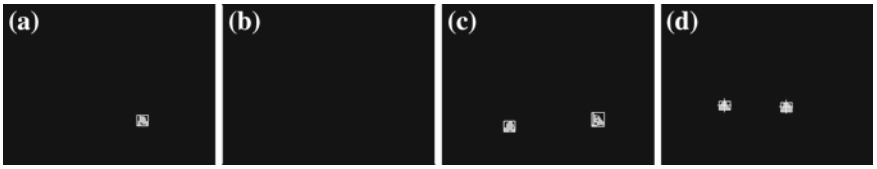


Fig. 6.17 Different detection scenarios. **a** Missing 1 light, **b** missing both, **c** tracker not upd. **d** Tracker updated

Note: White boxes ("□") while white crosses ("+") represent Kalman filter updates with new data from lights that were detected and successfully matched to existing Kalman filter trackers.

The collected data consists of:

- *total time of the trial* (in seconds);
- *number of processed frames*;
- percentage of frames, where a detection failure occurred with either one (Fig. 6.17a) or both lights being undetected, and where the last-known-good location of the lights was successfully tracked;
- percentage of frames used for updating Kalman filter with new data (Fig. 6.17c, d);
- average, min., and max. processing time per frame.

Case 1 (Dawn): This case was tested at dawn, while tracking a white Honda Accord vehicle for 4:17 min. The darker lighting conditions take the least amount of time to process, due to more-or-less even lighting and less interference from glare and similar-colored objects (which are sometimes recognized as possible taillights and have to be filtered out). Despite a lower processing time, however, dawn, dusk, and daytime cloudy scenarios have a greater level of failures, especially failures due to the loss of a single light. Complete video (high-speed): `youtu.be/smzgzAueDH4`.

Case 2: (Sunny): A green Volkswagen Passat vehicle was tracked for 3:19 min. Despite this lighting condition having less failures, a lot of similar-colored objects had to be filtered using symmetry and Bhattacharyya tests. This led to a significantly higher average processing time per frame than any other day, dawn, or dusk scenario. Nonetheless, the algorithm was capable of analyzing live video using the embedded camera hardware. The lines protruding out of the right taillight at certain parts of the video are caused by the mis-detection of concrete as part of the taillight, which does not affect the effectiveness of the tracking algorithm. Complete video: `youtu.be/8KMbFTNkYmQ`.

Case 3 (Cloudy): In this scenario, a light-blue Hyundai Accent was tracked for 3:44 min, with an average time per frame of 169.70 ms—the lowest among daytime scenarios, due to uniform lighting, no glare from the pavement/concrete and other passing vehicles or objects. Complete video: `youtu.be/XsqIVVLSGNM`.

Case 4 (Dusk): For the dusk scenario, a white Chevrolet Impala was tracked for 4:12 min. The high percentage of Kalman filter tracking failures can be attributed to the vehicle being on an uneven road, where the height separation between the centroids due to the tilt of the vehicle was greater than the full height of the master blob (as explained in Sect. 6.3.3.2. Complete video: `youtu.be/xBr9zZFRaCc`.

Case 5 (Night): The nighttime scenario caused the least failures, although the average time per frame was significantly higher (190.86 ms) due to headlight and street lamp detection and subsequent elimination. In this scenario, a gray Chevrolet Impala was tracked for 5:05 min. The splatter pattern (a "corona") around the taillights is caused by oversaturation, as captured by the camera. Since nighttime detection is the easiest scenario, the Kalman filter trackers are updated with new data more frequently. Complete video: `youtu.be/JlEFRY99lp4`.

6.4.2.2 Full Implementation

The detection algorithm was fully implemented on an embedded smart camera described in this chapter and is able to track vehicle taillights and detect turn signals (L and R) and brakes (B). Table 6.5 demonstrates execution times and rates of detection for this implementation. The 'gt' (ground truth) column indicates the number of times the turn signal or brake has been engaged during the entire video sequence (dark-gray column on left), followed by the number of detected events ('det'), as well as the total number of frames the action lasted (light-gray column on right).

Reliability of taillight detection was dependent on the distance of the target vehicle from the camera. Lower reliability and tracking failures were observed when the vehicle was further than 300 ft from the tailing vehicle. Economically speaking, the cost of an error or an undetected light (resulting in an accident) rises inversely proportional to the distance between the two vehicles. In this case, the performance reliability of the system at a distance of 0–300 ft was high.

The scenarios were tested live, using an actual embedded smart camera mounted on a car. A high detection rate was achieved in varying lighting conditions, with an average processing time per frame of approximately 186 ms (or 5.38 fps). This is on a microprocessor with no floating point support and with unoptimized code. Sample captures are provided in Fig. 6.18, where sensor data is shown alongside processed frames. Actual captures during the execution of the algorithm were not collected, since it takes upwards of 53 ms per frame to write the capture to FLASH and may degrade the tracking performance.

Table 6.5 Embedded smart camera detection and tracking results

Scene type	Totals		t_{frame} (ms)			Left turn					Right turn					Brake				
	# f	t (sec)	min	max	avg	gt	det	# f	rate	f+	gt	det	# f	rate	f+	gt	det	# f	rate	f+
Sunny	1,489	267.8	128	188	179.9	**1**	**1**	64	**1**	**0**	**1**	**1**	155	**1**	**0**	**4**	**4**	100	**1**	**0.**
Cloudy	2,099	386.6	127	187	184.2	**3**	**3**	247	**1**	**0**	**2**	**2**	249	**1**	**0**	**9**	**9**	329	**1**	**0.**
Dusk	1,508	246.6	127	190	163.5	**1**	**1**	64	**1**	—	**1**	**1**	34	**1**	—	**1**	**1**	82	**1**	—
Night	2,190	352.9	133	196	161.1	**1**	**1**	49	**1**	—	**3**	**3**	194	**1**	—	**11**	**11**	302	**1**	**.364**
Dawn	1,764	323.4	131	192	183.3	**2**	**2**	190	**1**	—	**2**	**2**	132	**1**	—	**8**	**8**	394	**1**	**.250**

t_{frame} = frame processing time; gt = 'ground truth', the number of events observed;
det = number of detected events; rate = detection rate; f+ = false positives; # f = number of frames

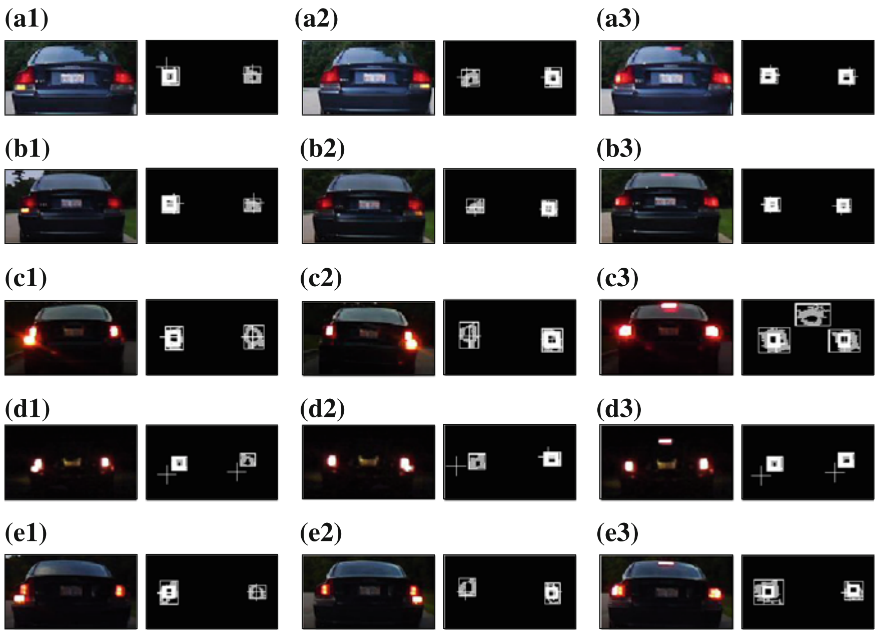


Fig. 6.18 Sample captures from embedded camera execution (capture on *left* full-color; *right* after embedded camera processing). **a1** Left turn, Day (sunny). **a2** Right turn, Day (sunny). **a3** Brake, Day (sunny). **b1** Left turn, Day (cloudy). **b2** Right turn, Day (cloudy). **b3** Brake, Day (cloudy). **c1** Left turn, Dusk. **c2** Right turn, Dusk. **c3** Brake, Dusk. **d1** Left turn, Night. **d2** Right turn, Night. **d3** Brake, Night. **e1** Left turn, Dawn. **e2** Right turn, Dawn. **e3** Brake, Dawn *Note* □: potential light candidates; +: LKF trackers; *thick white sq.* signal detection

6.5 Comparison with Other Algorithms

The taillight tracking algorithm implemented in this chapter was compared to O'Malley et al. [22] and a Haar-based object detection algorithm implemented in OpenCV, which was modified for vehicle detection. O'Malley et al. [22] implements nighttime detection only and is a heavily-cited approach to vehicle tracking. The Haar-based vehicle detection approach, on the other hand, can only work during the daytime. A summary of quantitative comparisons is shown in Fig. 6.19.

To the best of our knowledge, *the existing algorithms focus on either daytime or nighttime detection and tracking, but not both simultaneously.* In addition, no current algorithms are able to detect and classify vehicle alert signals in all lighting conditions. For these reasons, the three algorithms are evaluated only on the reliability of detection and tracking.

O'Malley et al. [22] implements set-exposure *nighttime* vehicle lamp detection and tracking. No daytime detection is possible. The authors use a set exposure to avoid interference from other light sources, which may degrade the detection performance. Figure 6.20 demonstrates the performance of the O'Malley algorithm when

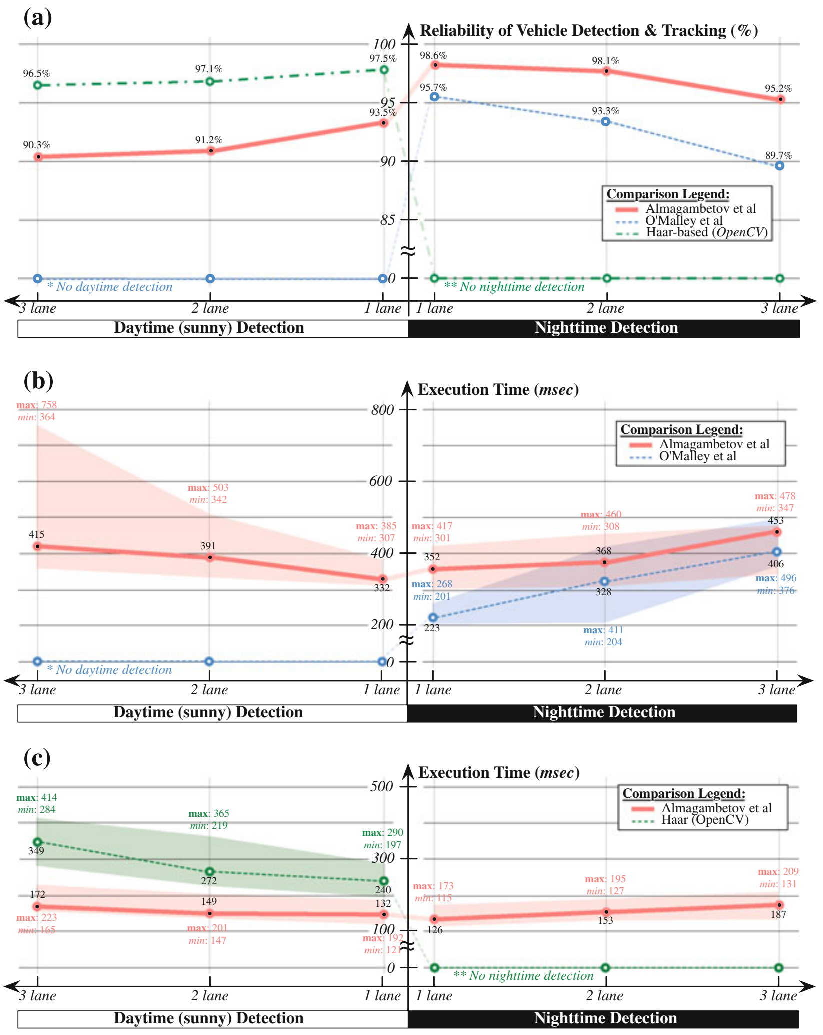


Fig. 6.19 Quantitative algorithm comparison. *Note* All algorithms within the same chart were run on the same platform in order to provide the most accurate comparison. **a** Reliability of detection and tracking using the proposed algorithm (*red*), O'Malley et al. [22] (*blue*), and Haar-based vehicle detection (*darkgreen*). *Since the other algorithms are designed for either daytime or* nighttime, 0 % detection rates are listed for [22] and Haar object detection-based algorithms for daytime and nighttime detection, respectively. Although Haar-based vehicle detection seems to have a higher detection rate, the false-positive rate is upward of 20 % (best-case scenario). **b** Execution time comparison between the proposed algorithm (*red*) and the method in [22] (*blue*). The *shaded areas* represent the possible execution time values (between the minimum and maximum times). Both of the algorithms were implemented in the same programming language and were run on the same platform in order to provide the most accurate comparison. **c** Execution time comparison between the proposed algorithm (*red*) and Haar detection (implemented using OpenCV libraries) (*green*). The *shaded areas* represent the possible execution time values (between the minimum and maximum times). Both of the algorithms were implemented in the same programming language and were run on the same platform in order to provide the most accurate comparison

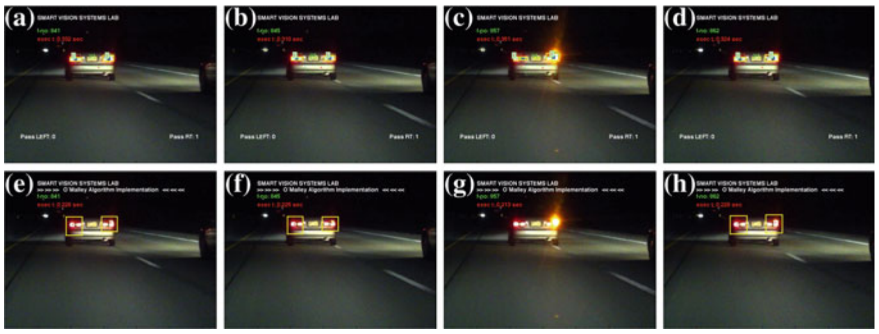


Fig. 6.20 An illustration of tracking through a turn signal using the proposed algorithm (*top row*) and the method in [22] (*bottom row*) using auto-exposure settings. The algorithm in [22] has problems when there are drastic light shape, color, and intensity changes. In addition, daytime–nighttime detection requires auto-exposure to adjust for changes in lighting conditions, whereas the approach in [22] implements the algorithm using a set exposure. **a** Frame 841. **b** Frame 845. **c** Frame 857. **d** Frame 862. **e** O'Malley (841). **f** O'Malley (845). **g** O'Malley (857). **H** O'Malley (862)

auto-exposure settings are used (critical for daytime/nighttime detection). No other features, such as codebook, sophisticated Kalman filter correction mechanisms, or signal detection are implemented in this algorithm.

The daytime detection performance is compared to a Haar-based detection algorithm (that cannot provide nighttime detection), implemented in OpenCV. The false-positives generated through the use of this detection algorithm are not listed in Fig. 6.19a, although they can be upward of 20 %. The Haar classifier was trained with a database of 300 positive and 300 negative training images.

6.6 Summary

Tracking moving objects from a mobile platform can be a challenging task, as both the background and the foreground are constantly changing. In this chapter, we presented a standalone system for tracking vehicle taillights and detecting turn and alert signals from a mobile platform that uses an embedded smart camera to process live video streams. The novelty of the approach described in this chapter stems from the ability to track vehicle taillights both during the day and at night, unlike some state-of-the-art research that either tracks vehicles using Haar wavelets only during the day **or** using thresholds only at night. In addition to being able to track vehicle taillights despite drastic changes in environment lighting conditions, the presented system uses a single approach for tracking vehicles regardless of illumination, without having to execute two or more separate algorithms.

We tested the algorithm extensively to ascertain its robustness. Since we used a modular approach, all of the lightweight algorithms that were designed for this implementation can be used for other computer vision applications for Intelligent Transportation Systems (ITS).

As future work, the algorithm developed in the current chapter may be integrated with lightweight lane detection. This will aid in the localization of the vehicle in the scene and taillights can be rejected or verified based on the road perspective (from the vanishing point obtained from lane detection).

Acknowledgments This work has been funded in part by NSF CAREER grant CNS-1206291.

Appendix

Below are the matrices used for the Kalman filter.

- A: **Movement Matrix**, which represents how the state of the system changes by drawing a relationship between the current state of the system at time step k to the state of the system at the previous time step $k - 1$, Eq. (6.7). A^T represents the transposed movement matrix A.

$$A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \Delta t & 0 & 1 & 0 \\ 0 & \Delta t & 0 & 1 \end{bmatrix} \tag{6.7}$$

- H: **Measurement Matrix**, representing the dependency of the measurement on the system state, Eq. (6.8). H^T represents the transposed movement matrix H.

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad H^T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{6.8}$$

- R: **Measurement Noise Covariance**, (constant) Eq. (6.9).

$$R = \begin{bmatrix} 0.2845 \\ 0.0045 \\ 0.0045 \\ 0.2845 \end{bmatrix} \tag{6.9}$$

- Q: **Process Noise Covariance**, (constant) Eq. (6.10).

$$Q = \begin{bmatrix} 0.01 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 \\ 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0.01 \end{bmatrix} \tag{6.10}$$

- P_{k-1} (initial estimate): **Estimation Error Covariance**, Eq. (6.11).

$$P_{k-1}(\text{initial estimate}) = \begin{bmatrix} 50 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 \\ 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 50 \end{bmatrix} \tag{6.11}$$

- I_4: 4 × 4 **Identity Matrix**, Eq. (6.12).

$$I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.12}$$

References

1. Regulation no. 7 of the Economic Commission for Europe of the United Nations (UN/ECE)–uniform provisions concerning the approval of front and rear position lamps, stop-lamps and end-outline marker lamps for motor vehicles (except motor cycles) and their trailers. Official Journal of the European Union L, 148/1-33 (2010)
2. Almagambetov A, Casares M, Velipasalar S (2012) Autonomous tracking of vehicle rear lights and detection of brakes and turn signals. In: Proceedings of IEEE symposium on computational intelligence for security and defence applications (CISDA), pp 1–7
3. Alt N, Claus C, Stechele W (2008) Hardware/software architecture of an algorithm for vision-based real-time vehicle detection in dark environments. In: Proceedings of design, automation and test in Europe (DATE), pp 176–181. doi:10.1109/DATE.2008.4484682
4. Cabani I, Toulminet G, Bensrhair A (2005) Color-based detection of vehicle lights. In: Proceedings of IEEE IV Symposium, pp 278–283. doi:10.1109/IVS.2005.1505115
5. Casares M, Almagambetov A, Velipasalar S (2012) A robust algorithm for the detection of vehicle turn signals and brake lights. In: Proceedings of IEEE ninth international conference on advanced video and signal-based surveillance (AVSS), pp 386–391. doi:10.1109/AVSS.2012.2
6. Chan YM, Huang SS, Fu LC, Hsiao PY (2007) Vehicle detection under various lighting conditions by incorporating particle filter. In: Proceedings of IEEE intelligent transportation systems conference (ITSC), pp 534–539. doi:10.1109/ITSC.2007.4357745
7. Chen DY, Lin YH (2010) Frequency-tuned nighttime brake-light detection. In: Proceedings of sixth int intelligent information hiding and multimedia signal processing (IIH-MSP) conference, pp 619–622. doi:10.1109/IIHMSP.2010.157
8. Chen P, Ahammad P, Boyer C, Huang SI, Lin L, Lobaton E, Meingast M, Oh S, Wang S, Yan P, Yang AY, Yeo C, Chang LC, Tygar JD, Sastry SS (2008) CITRIC: a low-bandwidth wireless camera network platform. In: Proceedings of second ACM/IEEE international conference on distributed smart cameras (ICDSC), pp 1–10. doi:10.1109/ICDSC.2008.4635675
9. Chen YL, Wu BF, Fan CJ (2009) Real-time vision-based multiple vehicle detection and tracking for nighttime traffic surveillance. In: Proceedings of IEEE international conference on systems, man and cybernetics (SMC), pp 3352–3358. doi:DOIurl10.1109/ICSMC.2009.5346191
10. Chern MY, Hou PC (2003) The lane recognition and vehicle detection at night for a camera-assisted car on highway. In: Proceedings of IEEE international conference on robotics and automation (ICRA), vol 2, pp 2110–2115. doi:10.1109/ROBOT.2003.1241905

11. Comaniciu D, Ramesh V, Meer P (2000) Real-time tracking of non-rigid objects using mean shift. In: IEEE conference on computer vision and pattern recognition, vol 2, pp 142–149
12. Cuevas E, Zaldivar D, Rojas R (2005) Kalman filter for vision tracking. Technical report B 05–12. Freie Universitat Berlin, Inst. Informatik, Berlin, Germany
13. Gelb A (1974) Applied optimal estimation. MIT Press, Cambridge, MA
14. Kalman RE (1960) A new approach to linear filtering and prediction problems. Trans ASME, J Basic Eng 81(1):33–45
15. Koller D, Weber J, Malik J (2006) Robust multiple car tracking with occlusion reasoning. Springer, Germany
16. Maybeck PS (1979) Stochastic models, estimation, and control, vol 1. Academic Press Inc, Orlando, FL
17. McCusker J (2008) Running standard deviations. Blog
18. Ming Q, Jo KH (2011) Vehicle detection using tail light segmentation. In: Proceedings of 6th Int IFOST, forum, 2, pp 729–732. doi:10.1109/IFOST.2011.6021126
19. Nagumo S, Hasegawa H, Okamoto N (2003) Extraction of forward vehicles by front-mounted camera using brightness information. In: Proceedings of Canadian conference on electrical and computer engineering (IEEE CCECE), vol 2, pp 1243–1246. doi:10.1109/CCECE.2003.1226124
20. National Safety Council: Injury facts, 2011 edn. NSC (2011)
21. Nat'l Highway Traffic Safety Admin. and DOT: Standard no. 108; lamps, reflective devices, and associated equipment. CFR 49 6(571):356–538 (2011)
22. O'Malley R, Jones E, Glavin M (2010) Rear-lamp vehicle detection and tracking in low-exposure color video for night conditions. IEEE Trans Intel Transp Syst (ITS) 11(2):453–462. doi:10.1109/TITS.2010.2045375
23. Park SJ, Kim TY, Kang SM, Koo KH (2003) A novel signal processing technique for vehicle detection radar. In: Proceedings of IEEE international microwave symposium digest (MTT-S), vol 1, pp 607–610. doi:10.1109/MWSYM.2003.1211012
24. She K, Bebis G, Gu H, Miller R (2004) Vehicle tracking using on-line fusion of color and shape features. In: Proceedings of IEEE conference on 7th international intelligent transportation systems (ITS), pp 731–736. doi:10.1109/ITSC.2004.1398993
25. Sorenson HW (1970) Least-squares estimation: from Gauss to Kalman. IEEE Spect 7:63–68
26. Sukthankar R (1993) RACCOON: a real-time autonomous car chaser operating optimally at night. In: Proceedings of intelligent vehicles symposium (IV), pp 37–42. doi:10.1109/IVS.1993.697294
27. Sun Z, Bebis G, Miller R (2006) On-road vehicle detection: a review. Pattern Anal Mach Intel, IEEE Trans 28(5):694–711. doi:10.1109/TPAMI.2006.104
28. Thammakaroon P, Tangamchit P (2009) Predictive brake warning at night using taillight characteristic. In: Proceedings of IEEE international symposium industrial electronics (ISIE), pp 217–221. doi:10.1109/ISIE.2009.5218254
29. Wang CC, Thorpe C, Suppe A (2003) LADAR-based detection and tracking of moving objects from a ground vehicle at high speeds. In: Proceedings of IEEE intelligent vehicles symposium (IV), pp 416–421. doi:10.1109/IVS.2003.1212947

Chapter 7
Automatic Fall Detection and Activity Classification by a Wearable Camera

Koray Ozcan, Anvith Mahabalagiri and Senem Velipasalar

Abstract Automated monitoring of everyday physical activities of elderly has come a long way in the past two decades. These activities might range from critical events such as falls requiring rapid and robust detection to classifying daily activities such as walking, sitting and lying down for long term prognosis. Researchers have constantly strived to come up with innovative methods based on different sensor systems in order to build a robust automated system. These sensor systems can be broadly classified into wearable and ambient sensors. Various vision and non-vision based sensors have been employed in the process. Most popular wearable sensors employ non-vision based sensors such as accelerometers and gyroscopes and have the advantage of not being confined to restricted environments. But resource limitations leave them vulnerable to false positives and render the task of classifying activities very challenging. On the other hand, popular ambient vision based sensors like wall mounted cameras which have resource capabilities for better activity classification are confined to a specific monitoring environment and by nature raise privacy concerns. Recently, integrated wearable sensor systems with accelerometers and camera on a single device have been introduced wherein the camera is used to provide contextual information in order to validate the accelerometer readings. In this chapter, a new idea of using a smart camera as a waist worn fall detection and activity classification system is presented. Therefore, a methodology to classify sitting and lying down activities with such a system is introduced in order to further substantiate the concept of event detection and activity classification with wearable smart cameras.

K. Ozcan (✉) · A. Mahabalagiri · S. Velipasalar
4-206 Center for Science and Technology, Syracuse University, Syracuse, NY, USA
e-mail: kozcan@syr.edu

A. Mahabalagiri
e-mail: akattema@syr.edu

S. Velipasalar
e-mail: svelipas@syr.edu

7.1 Introduction

7.1.1 Motivation

Elderly Healthcare has become a significant area of interest for the researchers in the recent years. Advancement in the fields of technology, healthcare and medicine have contributed towards an improved quality of life. U.S Census Bureau [6] predicts that number of people aged 65 and over relative to those between 15 and 64 is going to increase to 37 % by the year 2050. Consequentially, the requirement for a reliable and robust automated system for activity monitoring of elderly people will become imperative. Activities of elderly can in general be categorized into two categories: (1) Critical events such as falling down requiring immediate medical response; (2) Non-critical events such as walking, sitting, lying down etc. which help in the long term posture and motion analysis for chronic diseases such as arthritis and neurodegenerative diseases. Thus for an autonomous monitoring system to be robust, not only should it display accurate and real time capabilities but also be smart about expending its resources based on the criticality of an event.

7.1.2 Wearable and Ambient Sensors

The type of sensor being used plays a vital role for performed activity. Current state of the art methods can fundamentally be catergorized into wearable body sensors and ambient sensors. Wearable sensors have the advantage of not being restricted to confined environments. At the same time, such systems have limited computational and power resources at their disposal. The most popular method is the use of accelerometer based systems.These systems are simple and cost effective. However they are prone to a lot of false positives. Further, in order to be able to classify subtle activities, it would require the use of multiple acceleromters placed at strategic locations which would introduce inconvenience as a wearable system. Ambient sensors on the other hand provide convenience. However the monitoring is restricted to a particular region only. While vision based sensors have dominated ambient sensor systems, other sensors such as acoustic and vibrational sensors have yet to prove their robustness.

7.1.3 Cameras in Activity Monitoring

Cameras have been very popular choice as ambient vision based sensors. Most of the times, concerns are raised regarding issues of privacy since such a monitoring system can make people feel uncomfortable that they are being watched. Recently, hybrid sensor systems with accelerometers and cameras have been thought of being

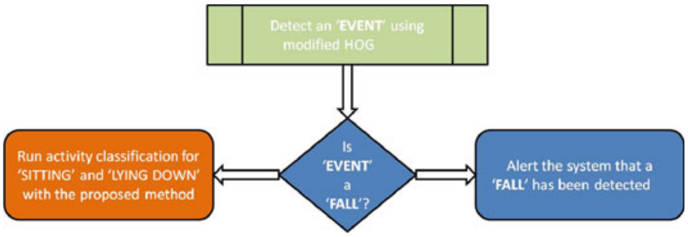

Detect an 'EVENT' using modified HOG
Run activity classification for 'SITTING' and 'LYING DOWN' with the proposed method
Is 'EVENT' a 'FALL'?
Alert the system that a 'FALL' has been detected

Fig. 7.1 Hierarchical flow chart

effective wearable systems. While accelerometers provide the activity information, cameras provide contextual information in order to validate the detected activities. This can help reduce false positives. However, cameras have much more to bring to the table than just providing contextual information. The means of harnessing such a potential is objective of this chapter.

Smart cameras, over the years, have gotten better in terms of increased processing capabilities and low power consumption. Such cameras can process sufficient features in a scene at real time, which might prove to be enough to learn about the activities in the images captured by the camera.Some of the static embedded camera based systems such as [2, 5] have displayed these capabilities effectively. In this chapter, a new methodology is introduced with such smart cameras as wearable sensors. Thus, such a system would not only be able to monitor the subject beyond restricted environments but also be able to provide contextual information timely. The privacy concerns with regard to a camera being used as a sensor can be considered as being alleviated to an extent in the sense that the person being monitored does not feel as being watched. In the next few sections, an application of fall detection and activity classification using wearable sensors will be introduced together with results to validate the methodology.

7.2 Proposed Method

The developed system consists of two different layers. First layer is for the detection of an event that has been started by the user. Once the event has been detected, the second layer is used for the classification of this event. Figure 7.1 shows the complete heirarchical flow.

Fall detection part of the algorithm employs Histogram of Oriented Gradients (HOG) as image feature descriptive components. HOGs are originally used in human detection problem by Dalal and Triggs [4] as descriptive components of the training dataset, which is essentially used for training a model for human silhouette. With the implementation proposed by [4], the image frame is divided into blocks and then each block is divided into n cells. For each pixel, the gradient magnitude and orientation for the grayscale value is calculated for generating histograms of strength and orientation for each cell in the image. Apart from HOG histogram generated for human detection method, for every cell, two separate m-bin histograms are built for edge strength and orientation. The concatenation of n histograms forms the HOG descriptive histogram, with the size of m × n components. As it will be described in detail in the next parts, these descriptors are the key components that are used to detect the occurrence of both an event and a falling down. When an event is detected, it is first tested to find out whether it is a fall or not. If the event detected is not a falling down, then an optical-flow based method is used to classify this particular event as either sitting or lying down. The complete algorithm description is presented below in Algorithm 1 along with detailed explanations of event detection and classification in Sects. 7.2.1 and 7.2.2.

Algorithm 1 Fall Detection and Edge Based Optical Flow for Activity Classification

```
for $All\ Frames$ do
    if $t == 1$ (First frame captured) then
        Initialize histogram and cell vectors.
    else
        if $Average\ Intensity \leq 30$ then
            Camera view is occluded.
        else
            if $max(B_\triangle) \geq \rho$ then
                $eventDetected = true$
                if $d^t_{t-1} \geq \tau$ then
                    $fallDetected = true$
                end if
            end if
        end if
        if $eventDetected == true$ and $event$ != fall then
            Step1 : (Perform recordings over $n$ consecutive frames)
            for $t = 1:n$ do
                a: Get the two consecutive images $I_t$, $I_{t-1}$
                b: Find canny edge images $bw_t$,$bw_{t-1}$
                c: Subject Edge regions to Optical Flow to get $uv$
                d: Get the mean horizontal and vertical flow vectors
                $\rightarrow h[t] = mean(real(uv))$
                $\rightarrow v[t] = mean(imag(uv))$
            end for

            Step2 : (Perform analysis on vectors $h$ and $v$)
            a: Calculate running means $h_{mean}$,$v_{mean}$ of $h$ and $v$
            b: Classify as sitting or lying based on running means
        end if
    end if
end for
```

7.2.1 Event Detection

First layer of the algorithm is detecting the occurrence of an event. Different from HOG implementation for human detection, separate histograms of edge orientation (EO) and edge strength (ES) are constructed. During the course of a falling down, edge orientations tend to change significantly, which can be observed from the changes in the gradient orientation histograms. Since falling down is a sudden and rapid type of movement, the edges in the captured images get blurry as it can be observed in Fig. 7.2a, b. During the experiments, it has been occurred that using original HOG may create false positives while walking. Different from the original HOG, we are not employing all of the cells in each block. Instead, the cells that do not contain significant edge information in are removed from the computation autonomously since they do not contribute to the overall edge information. It has been observed that for the detection of abrupt changes, reduced number of blocks is sufficient. With the simulations, using only one block gave the desired result of detecting a falling down sequence. Also, using one block for the image frame helped us to reduce the processing load of the embedded camera. Therefore, our implementation uses one block that is divided into 16 cells since using larger number of blocks would not necessarily improve the efficiency of the proposed algorithm. In order to improve the robustness of the algorithm for detecting falling downs, we are not using all of the cells that are fragmented in one block. Instead, the cells that do not contribute much to the overall edge information are removed autonomously. In other words, we are employing particular cells, which have significantly large edge information within the image frame, adaptively. In order to build the histograms, horizontal(dx) and vertical(dy) gradients are computed first for every pixel within a cell. Next, these values are used to compute the gradient strength ($\sqrt{dx^2 + dy^2}$) and orientation ($\tan^{-1}(dy/dx)$) for each pixel location. With the original HOG algorithm, for optimal detection purposes the orientation values are placed in a 9-bin histogram, which covers the angular range 0°–180°, along with the voting mechanism based on the gradient strength. Using the same HOG, causes false alarms even with walking sequences. For instance, an example shown in Fig. 7.4, illustrates a scenario where "lying down" and "sitting down" were classified as a fall with the HOG implementation used for people detection. Another example is provided in Fig. 7.5, where the fall occurs between frames 46 through 60, as walking triggers a false "fall" alarm a little after frame 30. For our EO histogram, we use 9-bin histograms as in [4], whereas we use 18 bin for the histogram for making the strength component more descriptive. Using a more descriptive component for edge strength values give the capability to detect especially falling down events, which includes significant changes with edge strength distribution. The histograms that are constructed from every cell in the image are concatenated to form a multi-dimensional histogram vector. Hence, we have two descriptive and concatenated histograms for edge strength (ES) and edge orientation (EO), which allows us the capability for measuring a falling down event.

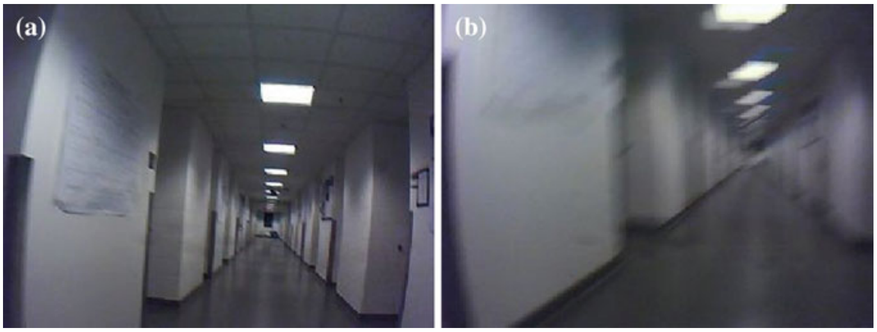


Fig. 7.2 Example frames captured by the camera during a fall

7.2.1.1 Dissimilarity Distance

After the feature histograms ES and EO are normalized, the dissimilarity distance between two histograms (r and s) is calculated using the following formula:

$$D = 1 - \frac{\sum_{i=0}^{N-1}(r_i - \bar{r})(s_i - \bar{s})}{\sqrt{\left[\sum_{i=0}^{N-1}(r_i - \bar{r})^2 \sum_{i=0}^{N-1}(s_i - \bar{s})^2\right]}}, \tag{7.1}$$

$$\bar{r} = \frac{1}{N}\sum_{i=0}^{N-1}(r_i) \quad , \quad \bar{s} = \frac{1}{N}\sum_{i=0}^{N-1}(s_i)$$

Dissimilarity distance values for ES(D_{ES}) and EO(D_{EO}) are cross-correlated in order to attenuate the noise in the signal while emphasizing the peaks for the detection. To increase the robustness, cross-correlated signal is autocorrelated ($d = (D_{ES}D_{EO})^2$). The outcome of this operation is provided in Figs. 7.4c and 7.5.

When the two distances are cross-correlated(or multiplied), proceeded with the autocorrelation of the resulting feature(or taking the square of it) provides us the ability to attenuate the gradual motion caused by walking, lying, or sitting. As we can observe from Fig. 7.5, since the distance values caused by the gradual motion is attenuated we observe a clear peak corresponding to actual falling down events.

In order to detect if there is an event occurring, we store the distance values $d^t_{t-\triangle}$ in an array of $B_\triangle$ for the last $\triangle$ frames. Therefore, the $B_\triangle$ saves $\triangle$-many $d^t_{t-\triangle}$ values, which is computed between the current frame t and the frame $t - \triangle$, such that $\triangle$ is an integer value. If the maximum distance value in the buffer array $B_\triangle$ is larger than a threshold ρ, which has been chosen empirically, it implies that the occurrence of an *event* taking place. The $\triangle$ value is chosen to cover one-second window in order to eliminate potential false positives that could be initiated by a rapid change between consecutive frames caused by sudden illumination changes,

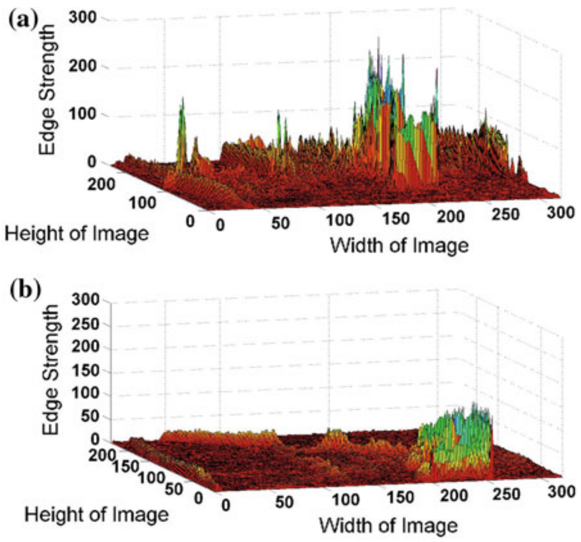


Fig. 7.3 Edge strength values corresponding to frames in **a** Fig. 7.2a, and **b** Fig. 7.2b, respectively

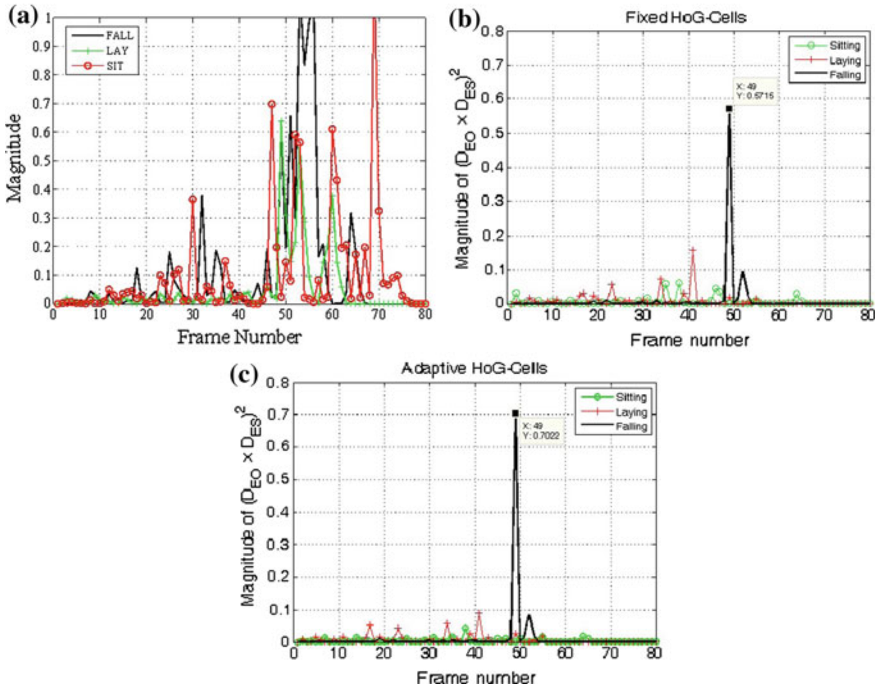


Fig. 7.4 **a** False 'fall' alarms are generated during lying down and sitting events when using (**a**) the original HOG; **b** proposed approach with fixed number of cells; **c** proposed approach with adaptive number of cells

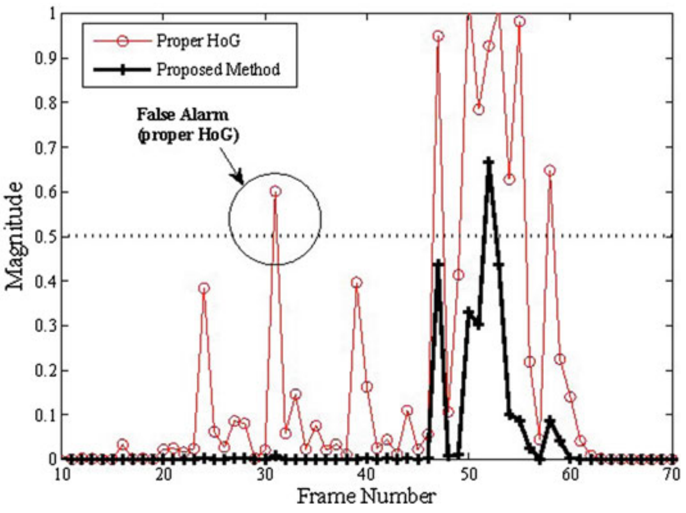


Fig. 7.5 False ‘fall’ alarm when using the original HOG

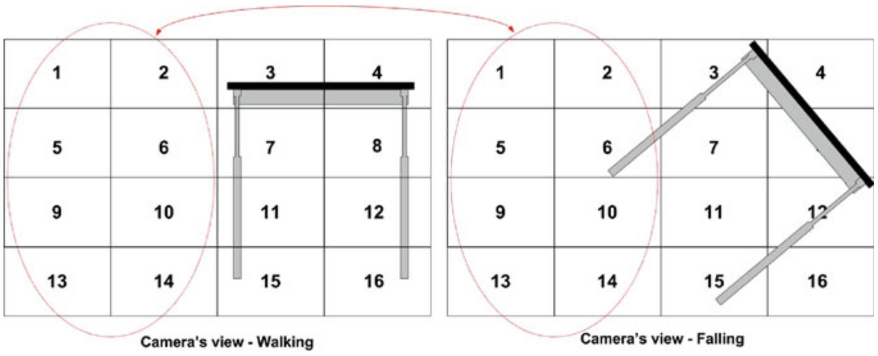


Fig. 7.6 Cells before and after a fall event

elongated time difference between captured frames, or camera occlusion etc. As a result, our algorithm detects an event first in order declare a *falling down* event next.

7.2.1.2 Adaptive Number of Cells

Different from HOG for human detection problem [4], we propose a method that adaptively selects the number of cells to be used as the feature descriptor according to their edge content. The purpose of applying such mechanism to our algorithm is that cells that contain either no edges or edges with low strength do not contribute to the purpose of the algorithm, and increase the similarity score between concatenated histograms. As it is illustrated with Fig. 7.6, a synthetic scenario is provided for a falling down event. As can be observed, cells numbered as 1, 2, 5, 6, 9, 10, 13, and 14 gives minimal or no useful feature to detect fall or differentiate from other gradual activities. Using the histograms corresponding to these cells with the feature descriptor would lower the dissimilarity between corresponding frames.

The importance of using adaptive cells can be observed in Fig. 7.4b, c. The peak of the amplitude value for the dissimilarity distance during a fall is higher for the same experiment as shown in Fig. 7.4c. A higher dissimilarity value for the same experiment allows us to have more robust system to detect falling downs with our algorithm. As an anticipated result, the system is also becomes less prone to false negatives, i.e., missing fall events. More results presented with the overall experiments regarding to the comparison between adaptive and fixed number of cells are presented in Sect. 7.3.

For selecting which cells to remove, the maximum amplitude value among the bins of each cell is found first. Next, the mean and standard deviation of the vector of maximums is calculated from the n cells in an image frame. Then, the algorithm ignores the cells whose maximum value are α standard deviation away from the overall mean of maximum bin values in each cell. Also, it is not only the cells that are adaptively selected but also the threshold is adaptively selected according to cell content within current frame. In order to avoid having false positive caused by removing too many cells in the image frame, the algorithm is allowed to remove at most half of the number of cells, which is 8 cells for our experiments (Fig. 7.3).

7.2.2 *Event Classification*

For every frame captured, the algorithm presented in Sect. 7.2 first checks whether there is an event or not. When there is an event that has been started, then the algorithm checks if the fall condition is satisfied. In the cases when the event is not a fall, it performs optical flow calculations to classify the event as either sitting or lying down. Since fall detection part of the algorithm is more critical compared to the activity classification part, it has higher priority within the algorithm structure. In this section, we will further describe the details of fall detection and activity classification parts of the algorithm.

7.2.2.1 Fall Detection Using Modified HOG

For detecting a falling down event, the dissimilarity distance d_{t-1}^{t} is computed between the current and previous frames. If the computed value is greater than the threshold τ, a $fall\ event$ is detected and a fall alarm is declared. For different distance values, a typical falling down is plotted in Fig. 7.7. By observing the maximum of auto-correlated dissimilarity distance values for dEO and dES over $\triangle$ frames(solid red plot) gives us the measure to detect events when they are in progress. After an event has been detected, the $fall$ is differentiated by using the dissimilarity

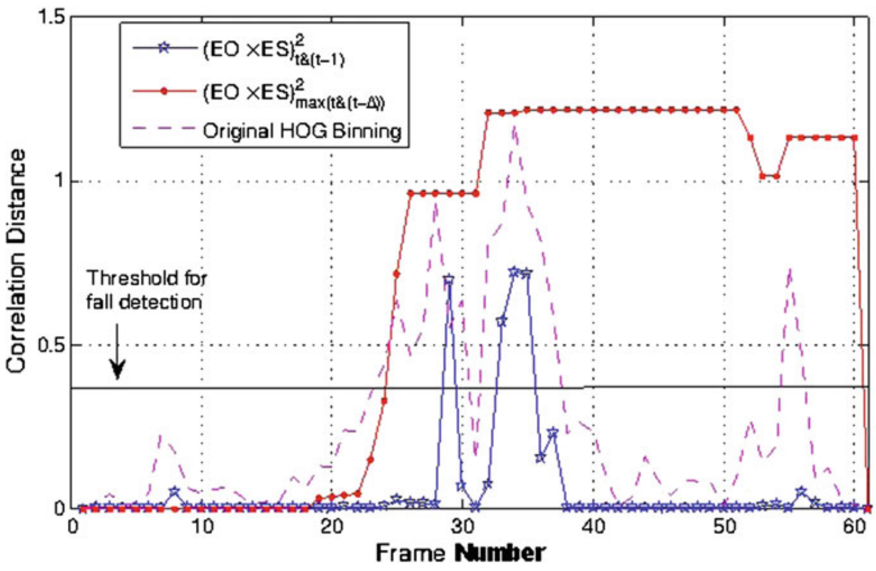


Fig. 7.7 Plots of different distances for a typical falling down

distance between the current and the previous frames(indicated with solid blue plot). Whenever the dissimilarity distance is greater than the marked threshold τ in the image, it is declared as a fall.

As mentioned above, we built two separate histograms that is different from the descriptive component employed with the original HOG algorithm [4]. The advantage of using such modified algorithm can be observed by looking at the dissimilarity values plotted in Fig. 7.7. In the plot, the distance values attained with original HOG implementation(dashed curve) and the one with the separate EO and ES histograms(solid blue curve) are demonstrated. As it can be verified, the original HOG implementation generates false positive when there is not a falling down between frames 50 and 60.

7.2.2.2 Event Classification Using Optical Flow

As described by Horn and Schunk [3], optical flow vectors are the distribution of significant velocities of motion of brightness patterns in an image resulting from the relative motion of objects or a camera. According to [1], four general types specify the relative motion between the object and a camera:

Type1:	Relative object motion at a distance
Type2:	Relative object motion towards a camera
Type3:	Relative object rotation at a distance
Type4:	Relative object rotation about its axis

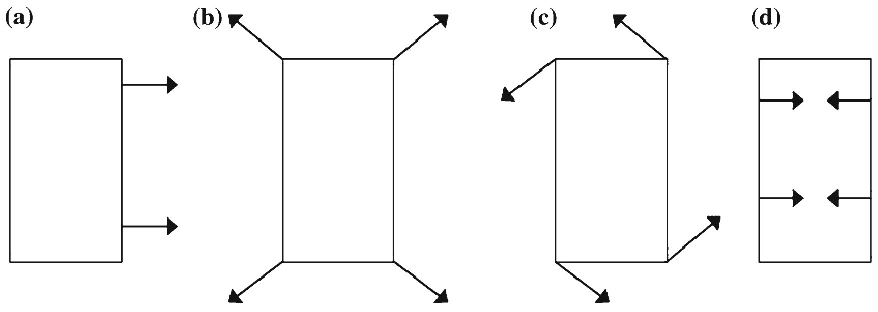


Fig. 7.8 **a** Relative object motion at a distance, **b** relative object motion towards the camera, **c** relative object rotation at a distance, **d** relative object rotation about its axis

All other types of relative motion can be described by combination of the types given above. Figure 7.8 provides illustrative descriptions of the four different motions. Since the motion descriptions are relative, same definitions applies to the situation where the objects are still and the camera attached to a human waist is moving around the scene. This realization helped us to develop a methodology for event classification based on the meaning of optical flow vectors. Vertical, horizontal, and rotational movements of the camera would provide significant respective velocity vector components. By separating the overall vector information into horizontal and vertical velocities and investigating the characteristic behaviours of significant activities such as sitting and lying down, we were able to develop an algorithm to detect activities when the subjects are performing experiments with the camera attached to their waist.

In our proposed algorithm, we provide a fast and accurate method to compute the optical flow vectors, which can also be feasible to implement on an embedded smart camera platform. The method is provided in the event classification part of Algorithm 1. First, the edges in a pair of consecutive image frames are extracted. Then, based on the features from extracted edges, the optical flow calculation is restricted to the edge regions instead of entire image. As a result, the optical flow vectors are analyzed further to perform event classification.

For edge detection, we employed Canny edge detection algorithm, which is known to be fast and accurate. The optical flow calculations depend on the density of edges. Hence, the threshold selected for edge detection is important for further processing of the algorithm. When the threshold is too high, it may result in losing significant edges causing the loss in the flow vector values. On the other hand, when the threshold is too low, we might have excess amount of edges, which may increase the computation load while affecting the accuracy. Therefore, a nominal threshold selection is important to have balanced trade-off.

In order to simplify embedded system implementation, a fine balance between speed and accuracy is required. Using a multi-level hierarchial optical flow method such as Lucas-Kanade (LK) can be computationally quite expensive. Generally, LK method processes 3–4 levels of pyramid in order to establish detailed optical flow

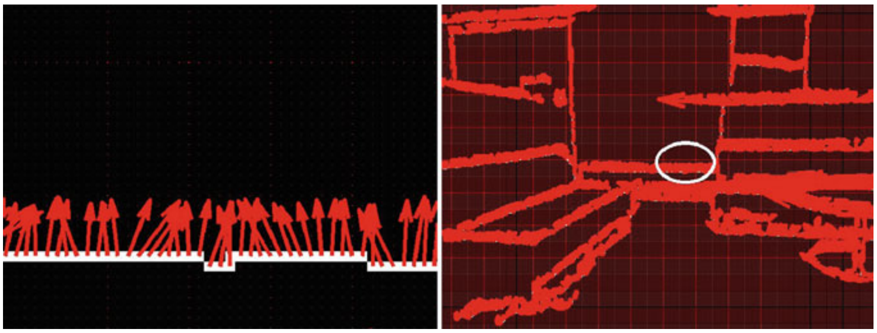


Fig. 7.9 Optical flow around the edge regions and a segment of the image with magnified flow vectors

vectors. However, for our application optical flow calculations on a single level of pyramid provided sufficient information to differentiate relative motion of the camera. By looking at the execution times and its resulting accuracy, a nominal threshold value of 0.5 provided a balanced trade-off between speed and accuracy.

An edge-based optical flow method proposed by [9] can be employed for our method but it is not feasible since it relies on the fact that complete contours being available in our image. Hence, we constructed a method depending on global edge flow directions rather than matching corresponding edge points. After the edges have been detected, we mark the regions around these edges and apply a single pyramid level of LK optical flow method. Optical flow vectors calculated along the edges are displayed in Fig. 7.9. A portion of the image is zoomed in to display the flow vectors along the edges. Figure 7.10 gives a more detailed insight into the process.Once the edge regions are identified, pairs of connected edges are considered in order to evaluate the optical flow between them. Next, we separate the flow vectors into horizontal and vertical components in order to further analyze the event into classification. This event classification method is fast and sufficient to analyze the global displacement of the camera and consequently the movement of subject's body.

For classification of activities of sitting and lying down, it is important to analyze the optical flow vectors in an event window. Then, the final classification result can be considered as a binary classification problem. Generally, we need to analyze the event over a period of three to four seconds in order to differentiate between sitting and lying down. In other words, since the videos are captured at 15 fps, our event window should consist of 45–60 frames. As it is provided in Step 1 of the event classification of Algorithm 1, after an event has been detected, optical flow computation is performed over a window size n. For a predefined window size n, we compute the vertical and horizontal optical flow values around the significant edges. With the Step 2 of Algorithm 1 we calculate the running mean of horizontal and vertical flow vectors.

Figure 7.11, shows running mean values of horizontal and vertical vectors for a typical sitting down event captured with a wearable camera. Also, Fig. 7.12 provides

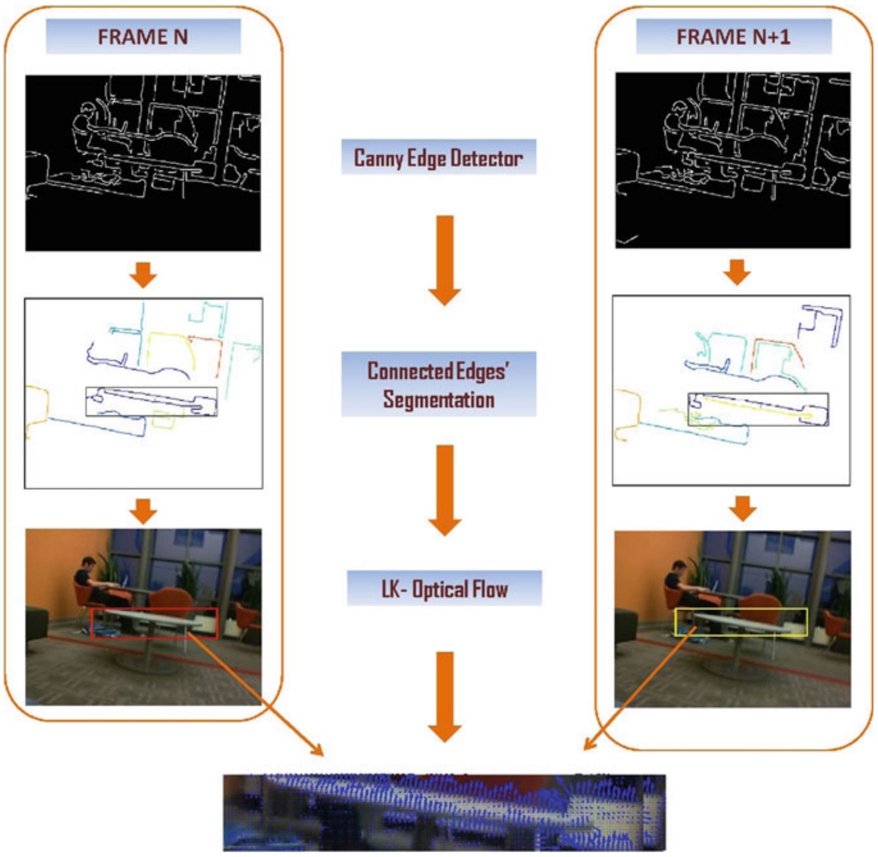


Fig. 7.10 Steps involved in finding the optical flow around the edge regions

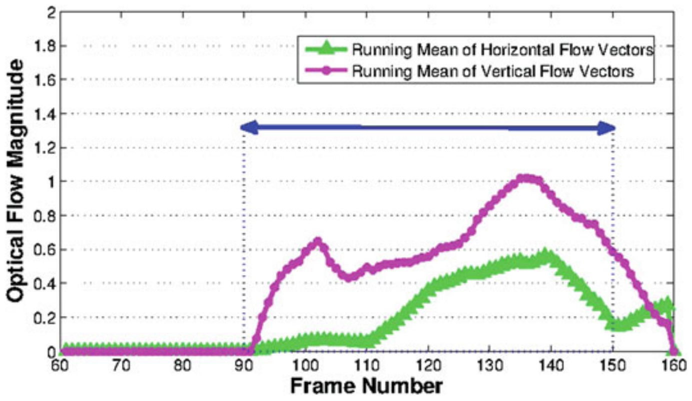


Fig. 7.11 Horizontal and vertical magnitude of optical flow vectors for a sitting down event

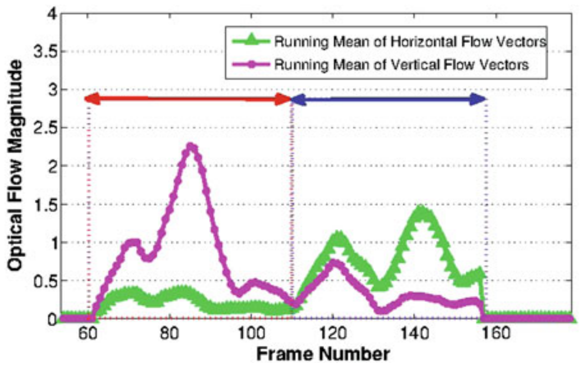


Fig. 7.12 Horizontal and vertical magnitude of optical flow vectors for a lying down event

the same kind of information for a typical lying down. As it can be observed from Fig. 7.11, a 60-frame window, which is indicated with a blue arrow, is chosen during a sitting event and it implies that a dominant vertical flow is observed. Since there are no other preceding or following event in the current state, dominant vertical movement of the camera implies that the person is transforming into a sitting state. In a similar manner, as it can be seen in Fig. 7.12, an elderly person typically sits down first and then continues to lie down continuously. Sitting duration is marked with a red arrow, whereas the lying duration is marked with a blue arrow. The first 50 frames indicates sitting down followed by a significantly greater horizontal motion with the next frames. Therefore, for a typical lying down event the camera can be assumed to have dominant vertical motion followed by a change in the orientation to horizontal motion indicating a lying down event. This differentiating information combined with the contextual information helped us to validate lying over sitting.

7.3 Experimental Results

In order to prove the robustness and efficiency of the proposed algorithm, different types of experiments have been performed such as:

- Experiments wherein videos are captured with a camera attached at a waist of different subjects, and then they are later processed on a PC.
- Embedded smart camera experiments wherein images are captured and then processed on-board in real-time on the microprocessor of the CITRIC camera board.

7.3.1 Sensitivity and Specificity Comparison

For evaluation of the performance of the falling experiments, sensitivity and specificity measures described by Noury et al. [7] is preferably used. Therefore, sensitivity is the percentage of the falls that are detected with the proposed algorithm while specificity is defined as the percentage of non-fall (sitting and lying down) that does not trigger the false alarm signal. *Sensitivity* and *Specificity* are defines as:

$$Sensitivity = \frac{TP}{TP + FN}$$
$$Specificity = \frac{TN}{TN + FP}, \tag{7.2}$$

where True Positive (TP) is the system detecting a fall when happened, False Positive (FP) is detecting a fall when it did not happen, True Negative(TN) is the system not detecting a fall when fall does not occur, and False Negative (FN) is the system not detecting a fall when it occurs.

7.3.2 Experiments with People

All of the experiments are performed with a camera attached to a belt around waist as it can be observed in Sect. 7.3.3. The camera view is positioned to be at the center of the waist facing the front view of the body. The camera is attached to the waist because it is stable region of the body representing the actual speed and movement trajectory of the body. The head, legs and arms of the body tend to be exposed to faster movement with people's daily activities. The nature of the experiments is meant for the subjects to imitate or act as an elderly person while performing. Therefore, it is rather challenging to recreate a free fall. The consciousness and fear of subjects can sometimes prevent the subjects' imitation an actual fall. Even with precautions such soft cushions in place, it occurred during the experiments that some subjects are too preventive to perform an 'actual fall'. Moreover, since the experiments are repetitive, the performance or attention of the subjects could degrade over time.

The initial set of experiments are recorded with eight different subjects. The video sequences are recorded with Microsoft® LifeCam™ camera at the frame rate of 30 fps. In order to be on the same line with embedded smart camera application, the captured image size is 320×240 and we are processing only even-numbered frames in order to have reduced computation load. Again for the purpose of not increasing the computation load unnecessarily, we used only one block and 16 cells in our implementation. The other parameters of the algorithm is selected to be $\triangle = 17$, $\rho = 0.2$, and $\tau = 0.37$. In all the experiments, same values have been set. In order to cover last one second of the movement, $\triangle$ value is selected to be 17. ρ and τ are selected experimentally for correlation distance thresholds.

Table 7.1 Falls from standing up position: sensitivity and specificity using different methods

	Proposed method		Fixed-cell modified histogram		Original HOG [4]	
	Sensitivity (%)	Specificity (%)	Sensitivity (%)	Specificity (%)	Sensitivity (%)	Specificity (%)
Subject1	100	95.24	100	95.24	100	61.84
Subject2	100	90.48	90.91	100	100	66.66
Subject3	100	76.19	100	76.19	100	38.1
Subject4	90.91	80.95	72.73	90.48	100	50
Subject5	90	100	70	100	100	40
Subject6	81.82	85	63.64	90	100	5
Subject7	80	100	70	100	100	72.73
Subject8	60	85	50	95	100	50
Overall	**87.84**	**89.11**	**77.16**	**93.36**	**100**	**48.04**

Table 7.2 Falling from sitting position: sensitivity and specificity

	Falling from sitting	
	Sensitivity (%)	Specificity (%)
Subject 1	100	95.24
Subject 2	100	90.48
Subject 3	80	76.19
Subject 4	70	80.95
Subject 5	44.44	100
Subject 6	50	85
Subject 7	27.27	100
Overall	**67.39**	**89.69**

7.3.2.1 Fall Experiments

For falling down experiments, 8 different subjects performed 10 falling down from standing up position and 10 falling down from sitting position. Hence, 80 falls are imitated from standing up and sitting down position each. Table 7.1 summarizes the sensitivity and specificity values for falls starting with standing up position with:

- The proposed method with adaptive number of cells,
- When employing fixed number of cells,
- When using original HOG [4].

As it can extracted from table, the proposed method with adaptive number of cells provides the best overall sensitivity-specificity combinations among three different approaches.

Sensitivity and specificity results for falls from sitting down position are presented in Table 7.2. In general, falling from sitting down position is more complicated to detect compared to falling from standing up position. Since the amount of distance traveled to hit to the floor is less for sitting position, it is more challenging to detect.

Table 7.3 Classification rates on 195 sitting and lying down trials

	Sensitivity of sitting down (%)	Specificity (%)	Sensitivity of lying down (%)	Specificity (%)
Subject 1	100	90	90	100
Subject 2	70	100	100	80
Subject 3	90	100	100	90
Subject 4	100	80	80	100
Subject 5	100	100	80	100
Subject 6	90	80	100	100
Subject 7	100	90	90	100
Subject 8	60	80	80	60
Overall	**91.26**	**89.13**	**89.13**	**93.20**

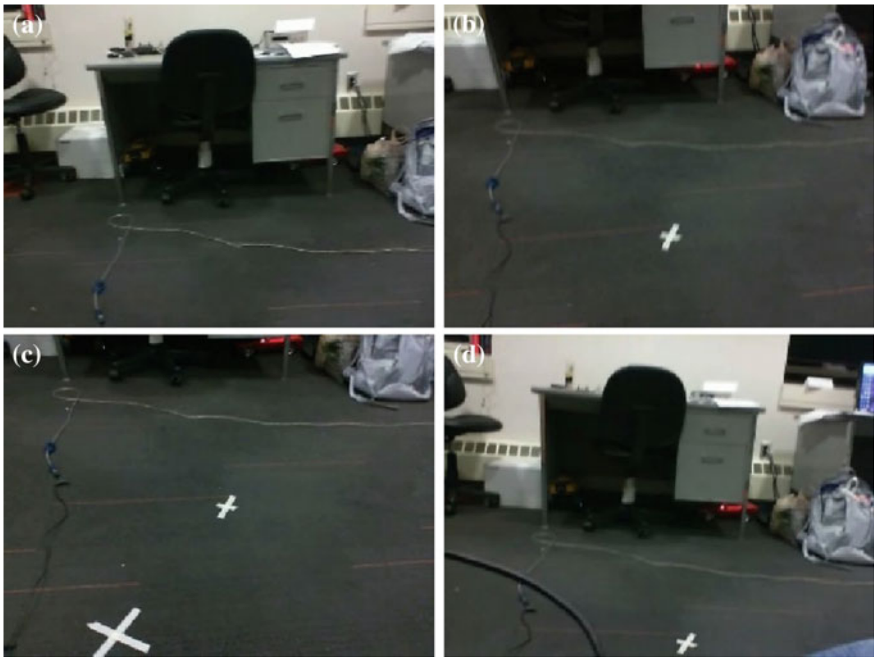


Fig. 7.13 Example frames captured during a sitting down event in the order of occurrence **a**, **b**, **c**, **d**

Therefore, for some subjects falling from a sitting position does not create dissimilarity distances as high as standing up position. Then, the sensitivity rate occurred to be less than the one for falling from standing up position. Specificity rates in Table 7.2 is computed according to other significant events such as sitting and lying down.

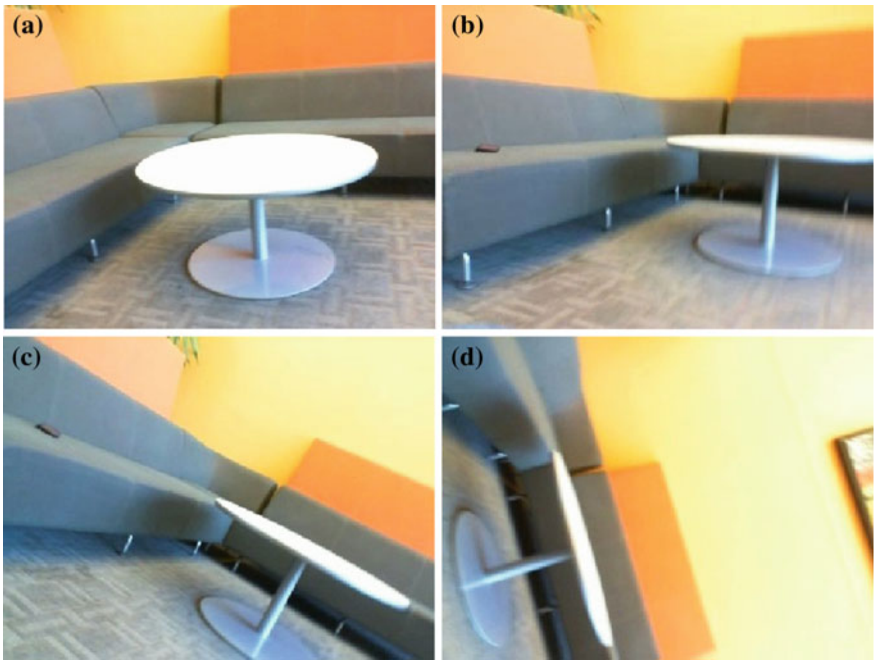


Fig. 7.14 Example frames captured during a lying down event in the order of occurrence **a**, **b**, **c**, **d**

7.3.2.2 Sitting and Lying Down Classification Experiments

The experiments for classification are also performed with 8 different subjects with approximately 10 trials of sitting and lying down each. Table 7.3 presents results of significant event classification. The overall sensitivity rates are achieved to be 91.26 and 89.13 % for sitting and lying down, respectively. Furthermore, the overall specificity rates are achieved to be 89.13 % for sitting and 93.2 % for lying down.

Example of captured frames for a typical sitting down event is presented in Fig. 7.13. Although being not consecutive, the given frames summarize the movement by showing several key frames in the event duration. As it can be seen, sitting down consists of a gradual vertical movement of the camera with respect to the scene facing forward.

In Fig. 7.14, some of the key frames of a typical lying down event is presented. As it can be verified, the lying down event starts with a sitting down first and then it continues with a transition of lying down that causes a horizontal motion of the scene captured by the camera. It should also be added that the algorithm is also capable of detecting lying down movement for the people who are already sitting.

Example frames captured during a falling down are shown in Figs. 7.15 and 7.16. As we can observe from the figures, falling causes a rapid change in the view causing

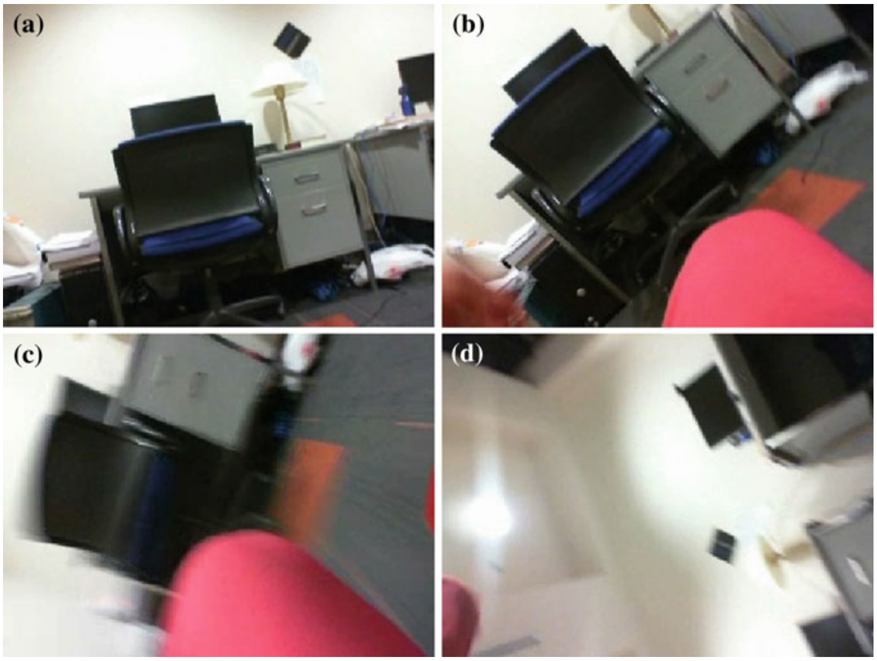


Fig. 7.15 Example frames captured during a fall from sitting position sitting down position in the order of occurrence **a**, **b**, **c**, **d**

blurriness. With the help of the sudden change in edge orientations and strengths, we were able to detect falls with high accuracy.

7.3.3 Embedded Camera Experiments with CITRIC Platform

Fall detection part of the algorithm is implemented and tested on CITRIC embedded smart camera platform [8], which consists of 624-MHz fixed-point microprocessor, 64 MB SDRAM, 16 MB NOR FLASH. The platform is also capable of wireless transmission of data with Crossbow TelosB mote. The captured images are processed on-board and then dropped. Thus, they are neither transferred nor stored due to privacy concerns. *Only* when a fall has been detected as a result of the processing on CITRIC, the corresponding fall alarm and captured frames during the fall may be sent wireless to emergency response personnel to locate the subject with precision due to provided scene details in the environment.

Figure 7.17a, b shows the CITRIC platform and its attachment to the belt, respectively. Everyone who performed the experiment wore the camera at a waist level. Since the camera is facing forward at the waist, it provided beneficial information

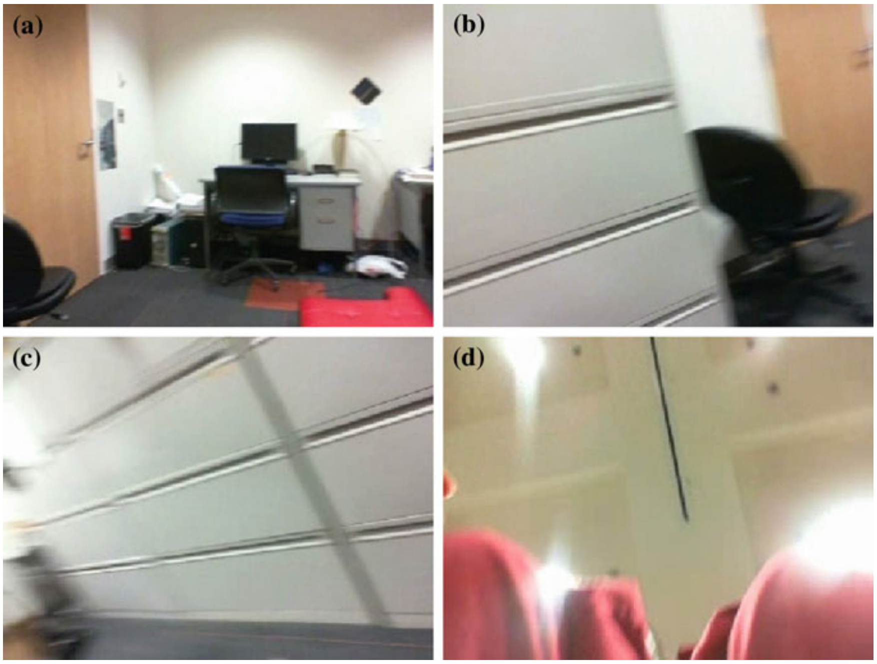


Fig. 7.16 Example frames captured during a fall from standing up position in the order of occurrence **a**, **b**, **c**, **d**

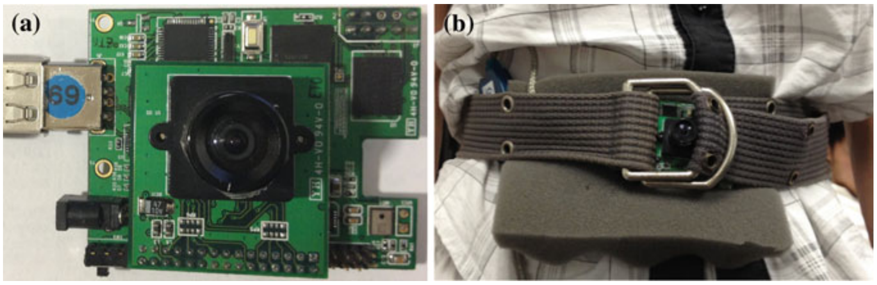


Fig. 7.17 **a** CITRIC platform, (**b**) which is connected around the waist

regarding the environment. Also, the location of the camera did not interfere with movement of experimenters.

The results of the experiments with CITRIC platform are presented in Table 7.4. Experiments have been performed with 3 different subjects including 50 falling down, 15 sitting down along with standing up, and 15 lying down events. The sensitivity rate of fall trials was in the range 84–86 %. False positive rates, which are wrongly alerted 'fall' alarms, are also presented for sitting down, standing up, and lying down in their corresponding columns. For falling down experiments, false positives represent the

Table 7.4 Sensitivity and false positive with sitting and lying down on CITRIC platform

	Falling down		Sitting down & Standing up/Lying down	
	Sensitivity	False positive	False positive	False positive
Subject 1	43/50	1/50	2/30	2/15
Subject 2	45/50	5/50	1/30	1/15
Subject 3	42/50	0/50	0/30	2/15

situations where a 'fall' alarm has been declared before actually falling down. The rate of false positives for lying down is higher compared to sitting down or standing up. Intuitively, sitting consists of straightforward motion of going up and down. Hence, it is less prone to creating false positives compared to lying. The implementation on the CITRIC platform gave promising results showing the robustness of the proposed algorithm.

The program developed on CITRIC platform consists of event detection, fall detection and occlusion handling. Although we tried to develop computationally light algorithm for optical flow computations, it is not simple enough to perform at real time along with fall detection. Our future goal is to have the complete algorithm to run on a smart camera.

Caused by the design of the CITRIC camera, exposure adjustment is done only once before the program starts. Since it is not adjusted periodically, the implementation performs well when the lightning intensity is similar in the environment. However, when the experimenter changes room or opens a door causing sudden changes of lightning intensity, it may create a false alarm.

7.4 Conclusions

In this chapter, we presented a novel algorithm to detect fall events and other significant activities like sitting and lying down by a wearable camera. The application is particularly applicable to elderly health or activity monitoring systems. Fall detection part of the algorithm employs histograms of edge orientations and strengths while activity classification part is using optical-flow based approach.

The camera, being worn by the subject, can monitor the environment wherever the subject may travel including outdoors. Moreover, since the captured images are not including the scene of the subject, wearable camera approach alleviates the privacy concerns of the users. Also, on-board processing power of the CITRIC platform allows the algorithm implementation such that captured images are neither stored nor transmitted except for the crucial frames that illustrates the course of falling down. Only when a fall has been detected, an alarm signal may be sent to a emergency response personnel with the images during and after a fall. The images of the surrounding environment may help the emergency personnel to localize the subject with accuracy. Considering the fact that GPS (Global Positioning Systems) does not provide accurate location information especially indoor environments, surrounding images may help better localization in a shorter time.

Fall detection part of the algorithm is implemented on an actual smart camera, which is CITRIC camera platform. Eight different subjects performed over 320 trials to prove the effectiveness of the algorithm in detecting falls and classifying other significant activities of sitting and lying down. Furthermore, 50 falls and 30 non-fall trials were performed with 2 different subjects. Also, the method is tested in outdoor environments and it is verified with 15 trials with the camera attached to a broomstick to imitate free falls.

For the fall experiments that start with standing up position, an average detection rate of 87.84 % is achieved with prerecorded videos. With the embedded smart camera application, the correct fall detection rate is 86.66 %. Also, the sensitivity rates of sitting and lying down are 91.26 % and 89.13 %, respectively.

The idea of employing optical flow algorithms can be further extended to classify other types of human activities.

Acknowledgments The authors would like to thank Khadeer Ahmed, Yi Li, Mustafa Ozmen, Ahmet Dundar Sezer, Chuang Ye, and Yu Zheng for their contributions with performing the experiments.

References

1. Shafie AA, Hafiz F, Ali MH (2009) Motion detection techniques using optical flow. World Acad Sci Eng Technol 56:559–561
2. Belbachir AN, Litzenberger M, Schraml S, Hofstatter M, Bauer D, Schon P, Humenberger M, Sulzbachner C, Lunden T, Merne M (2012) CARE: a dynamic stereo vision sensor system for fall detection. In: 2012 IEEE international symposium on circuits and systems (ISCAS), vol 731, no 734, pp 20–23
3. Horn BKP, Schunck BG (1981) Determining optical flow. Artifi Intell 17:185–203
4. Dalal N, Triggs B (2005) Histograms of oriented gradients for human detection. In: Proceedings of IEEE computer social conference computer vision pattern recognit, vol 1, pp 886–893
5. Fleck S, Loy R, Vollrath C, Walter F, Strasser W (2007) SMARTCLASSYSURV—a smart camera network for distributed tracking and activity recognition and its application to assisted living. In: First ACM/IEEE international conference on distributed smart cameras, ICDSC ’07, vol 211, no 218, pp 25–28
6. Vincent GK, Velkoff VA (2010) The next four decades, the older population in the united states: 2010 to 2050, current population reports, P25–1138. US Census Bureau, Washington
7. Noury N, Herve T, Rialle V, Virone G, Mercier E, Morey G, Moro A, Porcheron T (2000) Monitoring behavior in home using a smart fall sensor and position sensors. In: Proceedings 1st annual international conference microtechnology medical biology, pp 607–610
8. Chen P et al (2008) Citric: a low-bandwidth wireless camera network platform. In: Proceedings of ACM/IEEE international conference on distributed smart cameras, pp 1–10
9. Liao SK, Liu BY (2010) An edge-based approach to improve optical flow algorithm. In: 2010 3rd internatonal conference on advanced computer theory and engineering (ICACTE), vol 6, pp V6-45–V6-51

Part III
Applications

Chapter 8
Tracking by Detection Algorithms Using Multiple Cameras

Zixuan Wang and Hamid Aghajan

Abstract Detecting and tracking people using cameras is a basic task in many applications such as video surveillance and smart environment. In this chapter, we review approaches that detect and track targets using a single camera. After that, we explore the approaches that fuse multiple sources of information to enable tracking in a camera network. At last, we show an application that estimates the occupancy in a smart room.

8.1 Tracking by Detection Algorithms Using a Single Camera

8.1.1 Cascaded Classifier

Viola and Jonce [10] propose a visual object detection framework that is capable of processing images extremely rapidly while achieving high detection rates. It shows the state of the art performance in the object detection and is widely used in face detection. They use Haar features and use integral image to speed up the computation. Other features such as Histogram of Oriented Gradients (HOG) [1] and Local Binary Patterns (LBP) [6] can also be integrated into this framework. The learning algorithm bashed on AdaBoost, which selects a small number of critical visual features and learns a strong classifier by combining a set of weak classifiers. The learning algorithm iterates T times. In each time, a weak classifier is learned from all training set using a single feature and the weights are updated according to the error.

Z. Wang (✉) · H. Aghajan
Department of Electrical Engineering, Stanford University, Stanford, USA
e-mail: zxwang@stanford.edu

H. Aghajan
e-mail: aghajan@stanford.edu

The final hypothesis is a weighted linear combination of the T hypotheses where the weights are inversely proportional to the training error.

To improve the efficiency and to reduce the false positive rate, a series of classifier are applied to the image region. The initial classifier eliminates a large number of negative examples with very little processing. Subsequent layers eliminate additional negative but require additional computation. After several stages of processing the number candidates have been reduces radically. Further processing can take any form such as additional stages of the cascade or an alternative detection system.

8.1.2 Particle Filter

Particle filter has been a successful numerical approximation technique for Bayesian sequential estimation with non-linear, non-Gaussian models. The posterior probability can be formulated as follows:

$$p(x_t|y_{0:t}) \propto p(y_t|x_t) \int p(x_t|x_{t-1})p(x_{t-1}|y_{0:t-1})dx_{t-1} \tag{8.1}$$

where x_t and y_t denote the hidden state and the observation at time t respectively. $y_{0:t}$ denotes $(y_0, \ldots, y_t)$. In the case of linear Gaussian state space models, it can be solved with close form using Kalman filter. But in practice, due to the non-linearity of the hidden state, Kalman filter and its approximation are usually not suitable.

$$p(x_t|y_{0:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|y_{0:t-1})dx_{t-1} \tag{8.2}$$

$$p(x_t|y_{0:t}) = \frac{p(y_t|x_t)p(x_t|y_{0:t-1})}{\int p(y_t|x_t)p(x_t|y_{0:t-1})dx_t} \tag{8.3}$$

where the process is initialized by the prior distribution $p(x_0|y_0) = p(x_0)$, $p(x_t|x_{t-1})$ is the transition model of the target and $p(y_t|x_t)$ is the likelihood model. Particle filter uses a set of weighted samples $\{x_t^{(i)}, w_t^{(i)}\}_{i=1}^N$ to approximate the posterior distribution in the filtering. The sample set is propagated by sampling from a designed proposal distribution $q(x_t|x_{t-1}, y_{0:t})$, which is called importance sampling. The importance weights of the particles are updated in each iteration as follows

$$w_t^{(i)} \propto \frac{p(y_t|x_t^{(i)})p(x_t^{(i)}|x_{t-1}^{(i)})}{q(x_t^{(i)}|x_{t-1}^{(i)}, y_{0:t})} w_{t-1}^{(i)}, \quad \sum_i w_t^{(i)} = 1 \tag{8.4}$$

Generally, we can approximate $q(x_t^{(i)}|x_{t-1}^{(i)}, y_{0:t}) = p(x_t^{(i)}|x_{t-1}^{(i)})$ and therefore

$$w_t^{(i)} \propto p(y_t|x_t^{(i)})w_{t-1}^{(i)} \tag{8.5}$$

8.1.2.1 Color Model

The color model proposed by Pérez et al. [9] is widely used in particle filters. The observation of the target is represented by an N-bin color histogram extracted from the region $R(x_t)$ centered at the location x_t. It is denoted as $Q(x_t) = \{q(n; x_t)\}_{n=1,\ldots,N}$, where

$$q(n; x_t) = C \sum_{k \in R(X_t)} \delta[b(k) - n] \tag{8.6}$$

where δ is the Kronecker delta function, C is a normalization constant, k is any pixel within the region $R(x_t)$. By normalizing the color histogram, $Q(x_t)$ becomes a discrete probabilistic distribution. The similarity between the current observation $Q(x_t)$ and the reference model Q^*, which is constructed at the initialization step either manually or from the automatic detector, is evaluated based on the Bhattacharyya coefficient

$$d(x_t, x_0) = \sqrt{1 - \rho[Q(x_t), Q^*]} \tag{8.7}$$

$$\sqrt{[Q(x_t), Q^*]} = \sum_{n=1}^{N} \sqrt{q(n; x_t) q^*(n; x_0)} \tag{8.8}$$

In order to encode the spatial information of the observation, a multi-part color model is employed, which splits the targets vertically into two parts. The color histogram of the two parts are constructed separately and concatenated in parallel as new histogram. The likelihood is then evaluated as

$$p(y_t | x_t) \propto e^{-\lambda d^2(x_t, x_0)} \tag{8.9}$$

8.1.3 Boosted Particle Filter

Okuma et al. [7] propose the approach which combines the AdaBoost detector and the particle filter. The expression for the proposal distribution is given by the following mixture.

$$q_B^*(x_t | x_{0:t-1}, y_{0:t}) = \alpha q_{ada}(x_t | x_{t-1}, y_t) + (1 - \alpha) p(x_t | x_{t-1}) \tag{8.10}$$

where the parameter α can be set dynamically. When $\alpha = 0$, the algorithm reduces to the regular particle filter. By increasing α, more importance is placed on the AdaBoost detections.

8.1.4 Multiple Instance Learning

Traditional discriminative learning algorithm for training a binary classifier that estimate $p(y|x)$ require a training data set of the form $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ where x_i is a feature vector and $y_i \in \{0, 1\}$ is a binary label. In the Multiple Instance Learning framework [11] the training data has the form $\{(X_1, y_1), \ldots, (X_n, y_n)\}$ where a bag $X_i = \{x_{i1}, \ldots, x_{im}\}$ and y_i is a bag label.

The bag labels are defined as

$$y_i = \max_j (y_{ij}) \tag{8.11}$$

where y_{ij} are the instance labels, which are assumed to exist, but are not known during training. A bag is considered positive if it contains at least one positive instance. MILBoost uses the gradient boosting framework to train a boosting classifier that maximizes the log likelihood of bags

$$\log \mathscr{L} = \sum_i \log p(y_i|X_i). \tag{8.12}$$

8.2 Tracking by Detection Algorithms Using Multiple Cameras

8.2.1 Collaborative Particle Filters

Particle filters are conventional in multi-camera tracking. Tracking information from multiple cameras are fused on the ground plane using homographies.

The homography is defined as a 3×3 matrix, where

$$\begin{pmatrix} x \\ y \\ w \end{pmatrix} = H \cdot \begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} \tag{8.13}$$

where $(x, y, w)^{\mathrm{T}}$ is the homogeneous coordinate on the image plane and $(x', y', w')^{\mathrm{T}}$ is the homogeneous coordinate on the ground plane. Each camera is calibrated so the homography of each camera is known. To obtain the location of each people, the foot position of the tracked people have to be detected. However, this is a difficult error-prone task. Du et al. [2] solve this problem by exploiting the principal axes of the targets, the intersections of which give better ground position.

Clustering methods such as Expectation Maximization (EM) [5] can also be used to fuse the information from multiple cameras.

8.3 Application: Occupancy Detection

In this work, we attempt to detection the occupancy in a smart environment covered by camera networks and understand occupancy patterns. Understanding the occupancy patterns benefits energy saving and environment securities etc. Computers can interact with other appliance in the smart room and the user. Concrete examples include that the air conditioning is controlled by the current occupancy estimation: when the number of people in a room is high, the temperature is set to low and when the room is empty the air conditioner can be turned off to save energy. Another example is that the computer can prompt the administrator when the abnormal occupancy pattern is detected: if the room should be empty at night but one people is detected during this time period, the computer should notify the security about this case.

Current approaches to occupancy detection take place mostly in commercial buildings through the use of passive infrared (PIR) motion detectors. However, motion detectors have inherent limitations when occupants remain relatively still. Recent depth sensors including Kinect make motion detection more accurate but they have the limitation that the object should be within a certain distance. Therefore, to achieve these goals, we propose to use visual sensors, i.e. video cameras, to monitor the user. We use the prevalent camera networks to detect the location of the users and thus estimate the occupancy and further understand the occupancy patterns over time.

8.3.1 Overview

Our proposed system consists of three main components: sensor layer, behavior layer and service layer. In the sensor layer, low level image processing and computer vision algorithms are applied to detect human and to estimate the occupancy. The occupancy confidence are propagated in both temporal and spatial domains to provide the interface for the behavior layer. In the behavior layer, the occupancy patterns are modeled by considering the temporal and group information. Finally, in the service layer, alerts or recommendations are prompts to the user based on the current occupancy estimation and learned behavior patterns via the decision model. The system overview is illustrated in Fig. 8.1.

8.3.2 Approach

In this section, we illustrate the algorithms in the sensor layer, which contain four major components: cascade classifier based on HOG descriptors, motion filter, geometric filter and confidence propagation module. The outputs of each cascade classifier are a set of bounding boxes, which contain detections of people from a single camera. The following motion filter and geometric filter are applied to remove false positive detections. The confidence propagation method is used to combine

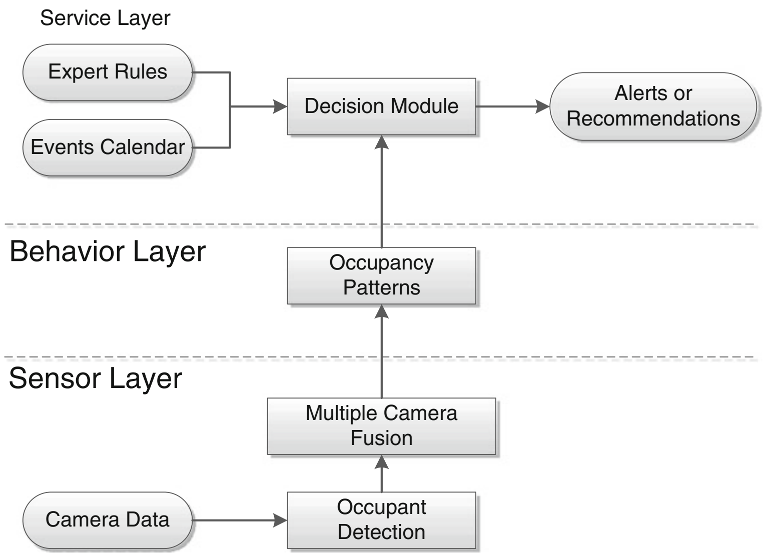


Fig. 8.1 The overview of the system

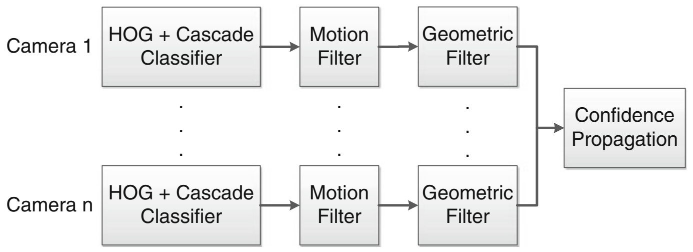


Fig. 8.2 The pipeline of the algorithm. Each camera has a separate cascade classifier, motion filter and geometric filter. The outputs are merged at the confidence propagation module to generate the occupancy detection

the detection results from multiple cameras. The inputs to the confidence propagation module are bounding boxes from multiple cameras and the final output is the occupancy detection. The pipeline is shown in Fig. 8.2.

8.3.3 Cascade Classifier

To make our people detector invariant to the viewpoint change, we train four different cascade classifiers: front, back, left side and right side. We try Haar [10] and HOG descriptors [1] and find the HOG descriptor can give us the best accuracy and is also fast to compute.

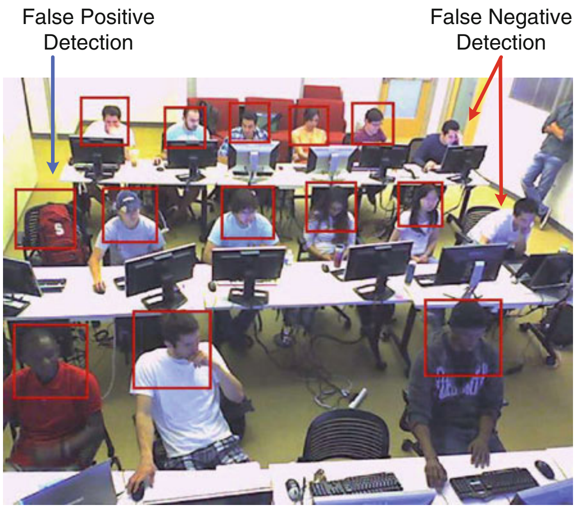


Fig. 8.3 The false positive and false negative detection examples. The false negative detections are often caused by movements of the people and the false positive detections are from the cluttered background such as chairs or bags

A general problem with the cascade classifier is the reliability of the resulting detection; i.e., not all people are detected in each frame (false negative detections) and some detections are not caused by a person (false positive detections). These problems are demonstrated in Fig. 8.3. To address them, we rely on motion filters (see Sect. 8.3.4) and geometric filters (see Sect. 8.3.5) to remove false positive detections and rely on confidence propagation module (see Sect. 8.3.6) to recover the false negative detections.

8.3.4 Motion Filter

The false positive detections are often caused by the background whose appearances are similar to the human such as Ω shape objects. One way to remove detections on those regions is to consider the motion on these objects. The image region of the true positive detection will contain small motions over time, i.e. 1 min. Whereas, the false positive detections result from the background stay static for a long time. For this reason, we compute the dense optical flow at each pixel using the method proposed by Farneback [3] and define the motion image to remove false positive detections. The motion magnitude is defined as:

$$m_t(x, y) = \sqrt{V_x^2 + V_y^2} \tag{8.14}$$

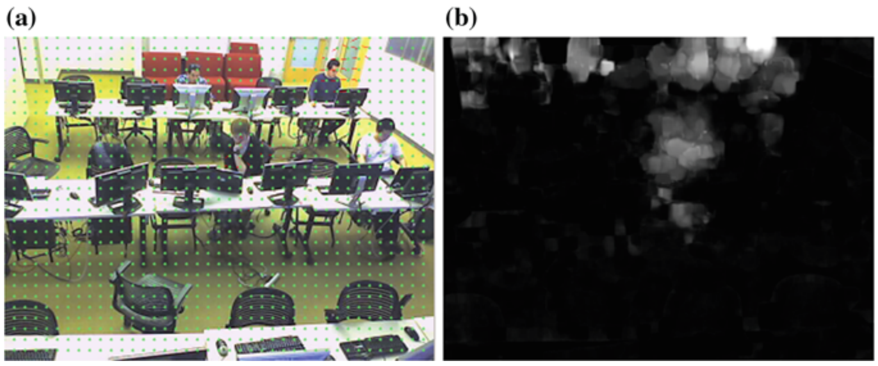


Fig. 8.4 An example of the motion image. **a** Shows the dense optical flow of the frame. **b** Shows the motion image when $k = 20$. The lighter pixels represent that areas are with stronger motion

where V_x and V_y are the x and y components of the optical flow at time t. $m_t(x, y)$ denotes the motion magnitude at time t. We define the motion image as the maximum of k consecutive motion magnitudes as follows:

$$M_t(x, y) = \max_{i=0}^{k} \{m_{t-i}(x, y)\} \tag{8.15}$$

One example of the motion image is shown in Fig. 8.4. We set a threshold to the detection bounding box: if the average of pixel values in the motion image within the bounding box is above a threshold, then we keep this detection. Otherwise, we consider the detection to be false positive and filter it out.

8.3.5 Geometric Filter

We assume that the height of people when sitting are approximately the same and the horizontal edge of the image plane is aligned with the ground. If these assumptions are met, the geometric constraints can be used to remove false positive detections.

We define the height of the upper body to be h and the height of the camera to be H. The angle between the principal axis and the horizontal plane is α. The distance between the optical center and the ground is $H/\sin\alpha$. Assume the focal length is f. The distance from the pixel to the image center is $f\tan(\alpha - \theta)$. The angle spanned by each upper body can be derived from the sine law:

$$\frac{h}{\sin\beta} = \frac{H}{\sin\alpha\cos(\alpha - \beta)} \tag{8.16}$$

$$f\beta = \arctan\frac{h\sin\alpha\cos\alpha}{H + h\sin^2\alpha} \tag{8.17}$$

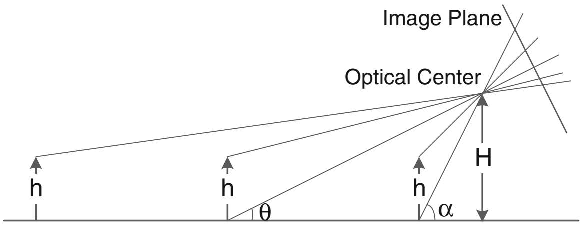


Fig. 8.5 The geometry of the camera assuming the height of the detected objects are the same and the *horizontal edge* of the image plane is aligned with the ground

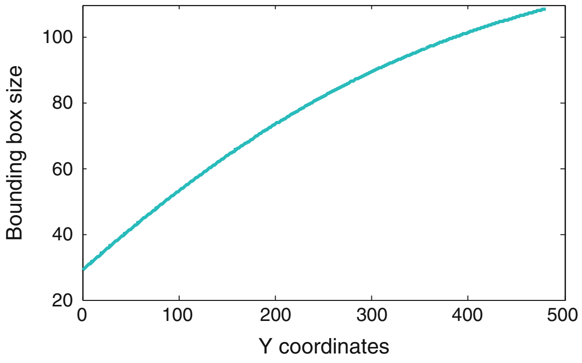


Fig. 8.6 An example of the people size versus the bottom position of the bounding box. $\theta = 22.5$, $H = 3$, $h = 0.85$, field of view=32. x axis represents the position of the bounding box within the frame. y axis represents the size of the bounding box satisfying the geometric constraints

The size of the upper body on the image is given by

$$s = f[\tan(\alpha - \theta) - \tan(\alpha - \theta - \beta)] \tag{8.18}$$

Figure 8.6 shows an example of the geometric constraints on the bounding box sizes. We only allow the size of the bounding box to be βs, where $0.8 < \beta < 1.2$ and s is computed from Eq. 8.18.

8.3.6 Confidence Propagation

We solve the false negative detection problem by using the algorithm we call *confidence propagation*. We associate a confidence score with each detection and create a *confidence image* for each camera. The confidence are propagated in both temporal and spatial domains to mitigate false negative detections.

8.3.6.1 Temporal Domain

It is quite frequent that one people was detected in the previous frame but due to the movement of the body, the same people is missing in the current frame. We need to propagate the confidence of the detection from past frames via the temporal domain. We define the temporal confidence propagation as follows:

$$\mathrm{conf}(i,t) = \begin{cases} 1, & \text{if detection at } t \\ e^{\lambda(t'-t)}, & \text{last detection is at } t' \end{cases}$$

where $\mathrm{conf}(i,t)$ is the confidence of the ith detection at time t. $\lambda > 0$ is the coefficient to control the confidence propagation in the temporal domain.

8.3.6.2 Spatial Domain

We first consider the spatial propagation within each view. The positions of the bounding boxes may jitter due to the noise. Therefore, we associate a Gaussian function with each detection, which is defined as $\mathrm{conf}(i,t)\mathcal{N}(p,\Sigma)$, where p is the center of the bounding box and Σ covariance matrix of Gaussian. In this way, we construct a *confidence image*, where each pixel represents the confidence that this pixel belongs to the foreground.

People detections from a single view are not reliable. Multi-cameras with overlapping regions can help us to gain confidence of the people detection. According to the epipolar geometry [4], for each pair of cameras, which have overlap in their views, the fundamental matrix between them can be computed and one point on one image corresponds to an epipolar line on the other image. In this article, we add a new assumption that, if we know the z value of the point in the world coordinate, which is the height of the object. We can find the corresponding point rather than the epipolar line on the other image. The image projection is defined as follows:

$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = A(R \cdot P + T) \tag{8.19}$$

where R is the rotation matrix and T is the transition vector, which are the extrinsic parameters of the camera. A contains the intrinsic parameters of the camera. R, T and A can be obtained from the camera calibration. P is the 3D coordinate of the object and $(u, v, 1)^{\mathrm{T}}$ is the homogeneous coordinate of the object on the image plane. Assume we have two calibrated cameras with parameters: $\{A^{(1)}, R^{(1)}, T^{(1)}\}$ and $\{A^{(2)}, R^{(2)}, T^{(2)}\}$. The object has the height h. Its homogeneous coordinate on image 1 is x_1. We will compute its corresponding homogeneous coordinate on image 2, denoted by x_2. Let B be the inverse of $A^{(1)}$ and Q be the inverse of $R^{(1)}$. The derivation is the following:

$$s = \frac{h + Q_{3,1}T_1^{(1)} + Q_{3,2}T_2^{(1)} + Q_{3,3}T_3^{(1)}}{Q_{3,1}B_1x_1 + Q_{3,2}B_2x_1 + Q_{3,3}B_3x_1} \tag{8.20}$$

$$P = Q(sBx_1 - t_1) \tag{8.21}$$

$$x_2 = A^{(2)}(R^{(2)} \cdot P + T^{(2)}) \tag{8.22}$$

where $Q_{i,j}$ represents the element in ith row and jth column in Q. B_i is the ith row of B. $T_i^{(1)}$ is the ith element in $T^{(i)}$.

We use this relationship to construct a graph called the *confidence graph*, which is used to propagate the confidence of the people detection. The graph is constructed with the following guidelines:

- Each vertex in the confidence graph is a pixel in the confidence image with non-zero values.
- Two vertices from different views are connected if they are conforming to the geometric constraints using Equation from 8.20 to 8.22.
- The weight associated with each edge is determined by the similarity of appearances.

First, we use each pixel with non-zero value in the confidence image as one vertex. The edge between two vertices means the coordinates of two pixels are conforming to the epipolar geometric constraints. We also define weights to edges, which reflects the association between two vertices. The weight of the edge is computed from the similarity of appearances. Here we use the color information to define the weight of the edge. We adopt the color model in [9] in our application. The color model is formulated the same way as in Sect. 8.1.2.1.

We use PageRank [8] to propagate the rewards on the social graph. Let M be the column stochastic matrix. The initial value of the PageRank vector is obtained from the confidence propagation in the temporal domain. The converged PageRank vector r is computed from the following equation.

$$r = \beta M r + \left[\frac{1-\beta}{N}\right]_N \tag{8.23}$$

where $\left[\frac{1-\beta}{N}\right]_N$ is an N vector with all entries $(1-\beta)/N$ and β is set to 0.85. We have one PageRank vector associated with each rule, which is computed from several iterations by using Eq. 8.23.

After PageRank converges, we set a threshold in the confidence image and run the agglomerative clustering algorithm on the pixels that are above the threshold. We initialize each pixel which is above the threshold as one cluster and continue merging two clusters if their distance is less than a threshold. The distance between two clusters is defined as the distance between their mass centers. If the distance between any pair of clusters is above the threshold, the merging is complete and each cluster represents one detection.

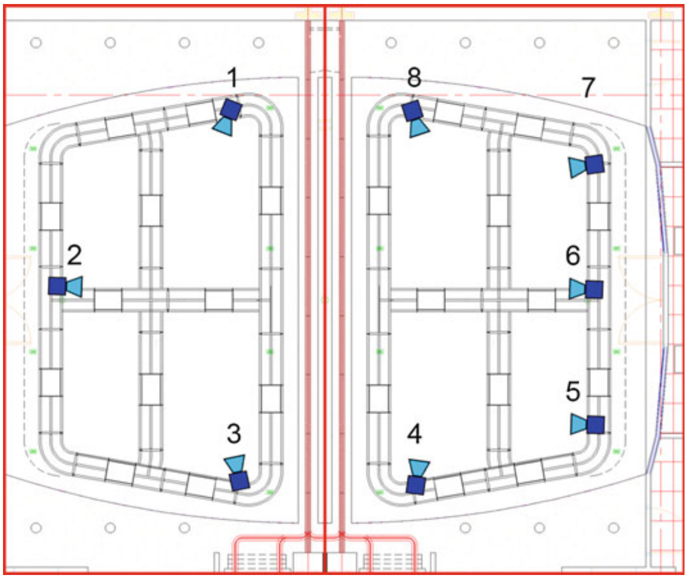


Fig. 8.7 The layout of the iRoom. 8 camera are deployed on the ceiling of the room. Camera *1* and *8* have the frontal view. Camera *3* and *4* have the rear view. Camera *2* has the left side view. Camera *5*, *6* and *7* have the right side view

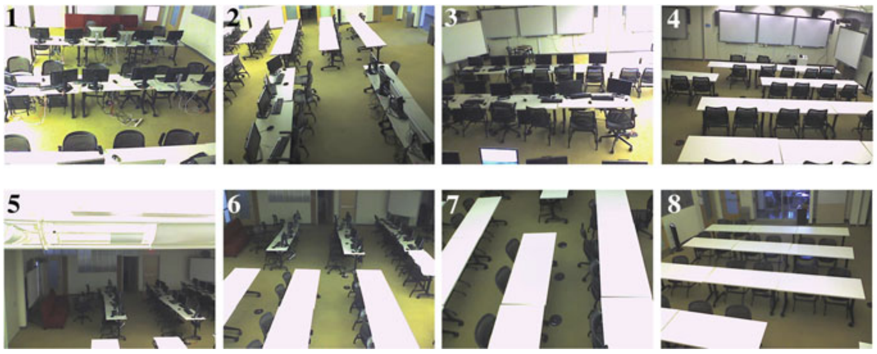


Fig. 8.8 The views of 8 cameras

8.3.7 Evaluations

We evaluate our algorithm using the dataset collected from a multi-purpose classroom with 8 Internet cameras. The layout of the room and the deployment of cameras is shown in Fig. 8.7. The views of 8 cameras are shown in Fig. 8.8.

We manually annotate positive samples on each frame. The negative samples are generated from images taken from the empty room under different illuminations. The number of positive samples of each view is 1,000 and the number of negative

Table 8.1 The first column shows the accuracy using HOG descriptor only. The second shows the accuracy when the motion filter and the geometric filter are added. The last column shows the accuracy when the confidence propagation module is added. Each result contains the mean and the standard deviation

HOG	HOG + filters	All
0.437 ± 0.128	0.614 ± 0.084	0.693 ± 0.101

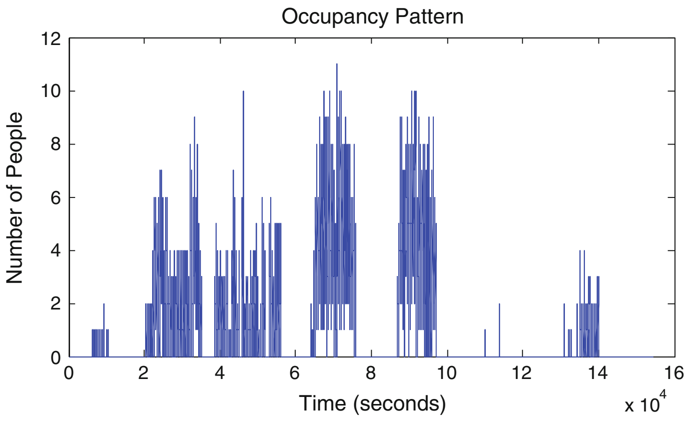


Fig. 8.9 The occupancy pattern of the iRoom over 1 week

samples is 1,000. We set the number of stages to be 20 and the size of the upper body patch to be 64×64.

To compare with the baseline occupancy detection algorithm, we use a video clip of 10 minutes and evaluate the performance of our algorithm. The class contains 14 students and we sample 10 timestamps to check the accuracy of each method. The accuracy is defined as $1 - |d - D|/D$, where D is the ground truth and d is the number of people estimated from the algorithm. The results are shown in Table 8.1.

To get the occupancy pattern of the iRoom, we run our occupancy detection algorithm for one week and the pattern is shown in Fig. 8.9. The starting time is from Sunday and the ending time is Saturday. We can see there are seven clusters, which represents each day during one week. We can see that on Sunday, Friday and Saturday, the room is almost empty and the other weekdays the room is more occupied.

8.4 Summary

We have presented approaches that detect and track targets using a single camera and we have also explored the approaches that fuse multiple sources of information that enables tracking in a camera network. At last, we show an occupancy detection algorithm in camera networks. The algorithm combines the appearance, motion and geometric information and exhibits higher accuracy. Our confidence propagation algorithm can be integrated with other low level visual processing algorithms including localization and tracking.

References

1. Dalal N, Triggs B (2005) Histograms of oriented gradients for human detection. In: CVPR, vol 1. IEEE, New Jersy, pp 886–893
2. Du W, Piater J (2007) Multi-camera people tracking by collaborative particle filters and principal axis-based integration. In: Computer vision-ACCV 2007. Springer, Heidelberg, pp 365–374
3. Farnebäck G (2003) Two-frame motion estimation based on polynomial expansion. In: Image analysis. Springer, Heidelberg, pp 363–370
4. Kellogg T, Truesdale R, Kellogg L, Osterman MDW, Brady H, Martin H, Webb P (1981) A computer algorithm for reconstructing a scene from two projections. Nature 293:133
5. Ni Z. Sunderrajan S, Rahimi A, Manjunath B (2010) Distributed particle filter tracking with online multiple instance learning in a camera sensor network. In: 17th IEEE international conference on image processing (ICIP), 2010. IEEE, New Jersy, pp 37–40
6. Ojala T, Pietikainen M, Harwood D (1994) Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In: Proceedings of the 12th IAPR international conference on pattern recognition, 1994. Conference A: computer vision and image processing, vol 1. IEEE, New Jersy, pp 582–585
7. Okuma K, Taleghani A, De Freitas N, Little JJ, Lowe DG (2004) A boosted particle filter: multitarget detection and tracking. In: ECCV. Springer, Heidelberg, pp 28–39
8. Page L, Brin S, Motwani R, Winograd T (1999) The pagerank citation ranking: bringing order to the web
9. Pérez P, Hue C, Vermaak J, Gangnet M (2002) Color-based probabilistic tracking. In: ECCV. Springer, Heidelberg, pp 661–675
10. Viola P, Jones MJ (2004) Robust real-time face detection. IJCV 57(2):137–154
11. Zhang C, Platt JC, Viola PA (2005) Multiple instance boosting for object detection. In. Advances in neural information processing systems, pp 1417–1424

Chapter 9
Consistent Human Tracking Over Self-organized and Scalable Multiple-camera Networks

Kuan-Hui Lee, Chun-Te Chu, Younggun Lee, Zhijun Fang and Jenq-Neng Hwang

Abstract In this chapter, a self-organized and scalable multiple-camera tracking system that tracks human across the cameras with nonoverlapping views is introduced. Given the GPS locations of uncalibrated cameras, the system automatically detects the existence of camera link relationships within the camera network based on the routing information provided by Google Maps. The connected zones in any pair of directly-connected cameras are identified based on the feature matching between the camera's view and Google Street View. To overcome the adverse issues of nonoverlapping field of views among cameras, we propose an unsupervised learning scheme to build the camera link model, including transition time distribution, brightness transfer function, region mapping matrix, region matching weights, and feature fusion weights. Our unsupervised learning scheme tolerates well the presence of outliers in the training data and the learned camera link model can be continuously updated even after the tracking is started. The systematic integration of multiple features enables us to perform an effective re-identification across cameras. The pairwise learning and tracking manner also enhances the scalability of the system. Thanks to the unsupervised pairwise learning and tracking in our system, the camera network is

K.-H. Lee (✉) · C.-T. Chu · Y. Lee · J.-N. Hwang
Department of Electrical Engineering, University of Washington, Box 352500,
Seattle, WA 98195, USA
e-mail: ykhlee@uw.edu

C.-T Chu
e-mail: ctchu@uw.edu

Y. Lee
e-mail: lygstj@uw.edu

J.-N. Hwang
e-mail: hwang@uw.edu

Z. Fang
School of Information Technology, Jiangxi University of Finance and Economics,
Nanchang 330032, Jiangxi, China
e-mail: zjfang@gmail.com

self-organized, and our proposed system is able to be scale up efficiently when more cameras are added into the network. Thanks to the unsupervised pairwise learning and tracking in our system, the camera network is self-organized, and our proposed system is able to scale up efficiently when more cameras are added into the network.

9.1 Introduction

Tracking humans across multiple cameras has recently attracted lots of interests in the visual surveillance community. Due to the limited field of view (FOV) of a camera, a target's information is no longer available once the target leaves the view of the camera. Hence, a surveillance system is required to have multiple networked cameras covering a range of areas. Tracking multiple people across the uncalibrated cameras with disjoint views is a rather challenging problem. Some researchers aim to structure it as a re-identification problem and have tried to come up with distinctive features of the targets, such as SIFT, SURF, covariance matrix, etc [1–7]. The re-identification is done based on the assumption that these kinds of features are invariant under different cameras' views. However, the ideal features for describing humans have not been discovered yet since the human appearance varies dramatically due to different perspectives, poses and illuminations when perceived from different cameras. On the contrary, instead of relying on the complex feature representations, we focus on solving the tracking problem based on systematically building the pairwise links among cameras [8]. If there exists a path allowing people traveling between two cameras without passing through any other camera, we call the two cameras are directly connected. An entry/exit zone is defined as an area that people tend to enter in or leave from within a camera's view. There may be several entry/exit zones in a camera's view, and a path between a pair of directly-connected cameras actually only connects one zone each in these two cameras (see Fig. 9.6 as an example). Given an identified pair of directly connected camera zones, the temporal/spatial/appearance relationships between each pair of *entry/exit zones* in two cameras corresponding to the path can be characterized by a *camera link model*. The model enables us to utilize particular features, which may not be invariant under different cameras, for re-identification purposes. For instance, due to different lighting conditions and camera color responses, the same object may appear with different colors under different views. The brightness transfer function (BTF) [9, 10], which stands for the mapping of color models between two cameras, can be applied to compensate for the color difference between two cameras before we compute the distance between the color histograms of two observations under two directly-connected cameras. In our system, several components are included in the formulation of the camera link model, namely transition time distribution, brightness transfer function, region mapping matrix, region matching weights, and feature fusion weights.

A camera link model is obtained based on the given training data which consists of two sets of observations detected from a pair of entry/exit zones between two directly-connected cameras. Each observation includes the time stamp and appearance information of the human. If people manually identify the correct correspondences between the observations in these two sets, the camera link model can be straightforwardly estimated in a supervised manner. For example, the transition time values between pairs of correct correspondences can be computed based on the difference between the entry time stamps and exit time stamps, from which the transition time distribution can be estimated [11, 12]. Supervised learning of camera link models is less feasible since large amount of human efforts are required, not to mention the additional efforts required for continuous updating of the learned camera link models, and how to perform the learning and updating automatically, i.e., unsupervised learning, while obtaining none or limited amount of accuracy degradation becomes a challenging research topic [14–18]. In our proposed system, inspired by the concept of feature points matching between two images, the camera link model is determined based on a fully unsupervised scheme [19] in the training stage. The camera link model is estimated given the training data without manually labeling the correspondences in advance. Moreover, in practice, since the connections between cameras may be arbitrary, the outliers usually exist in the training data for constructing the camera link model between a pair of connected cameras; that is, one departing from a camera does not enter the other connected camera, or one enters into a camera does not come from the other connected camera, and these people are called outliers. Our proposed method also effectively takes care of the presence of the outliers. With the ability of unsupervised learning and the subsequent continuous update of the camera link model between a pair of connected zones, it is now possible to scale up the tracking system to a large number of non-overlapping cameras as long as the connected zones can be systematically and effectively identified whenever a new camera is added into the system [20].

Given a camera network consisting of multiple cameras, two pieces of information are required before the camera link model estimation can be performed: (i) The system needs to identify which pairs of cameras have link models between them, i.e., which pairs are directly connected. Wrong links or redundant links can deteriorate the tracking performance easily, due to the increased searching range and resulting in reduced recall rate and increased false positives, not to mention the exponentially increased computational complexity. (ii) To our observation, the link actually only connects two entry/exit zones in a pair of directly-connected cameras; that is, if a person is traveling between two cameras, he/she will likely leave from one particular zone and enters into the other. Hence, the training data used in camera link model estimation (and the subsequent re-identification tracking) should only include the observations happening in these two specific zones in order to avoid too many outliers. Therefore, to identify which specific zones are linked together is another critical issue and a systematic method is thus proposed to perform the camera link identification by incorporating the information from Google Maps and Google Street View.

As shown in Fig. 9.1, our system is divided into two stages, the training stage and the testing stage. In the training stage, camera link models are estimated between all the directly-connected camera pairs (more specifically, the pairs of the entry/exit zones) based on our proposed unsupervised learning scheme. In the testing stage, the trained models are utilized to facilitate the consistent labeling during the tracking

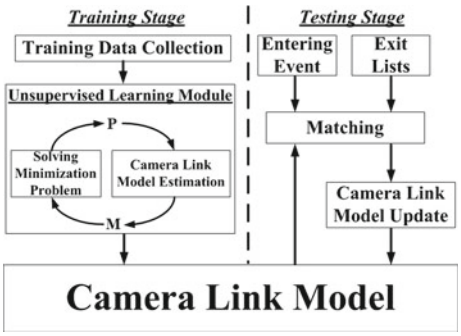


Fig. 9.1 System overview

across multiple cameras, and the models are also continuously updated based on the tracking results. The labels for the tracked targets within a single camera are generated based on the single camera tracking system proposed in [13, 21].

Our major contributions are to propose an unsupervised method that effectively estimates the camera link models and to apply the models in the real world scenario for tracking humans across the cameras with non-overlapping views. More specifically, (1) we formulate the camera link model estimation as an optimization problem and apply a deterministic annealing method to obtain the optimal solution. A reliable model is built based on this unsupervised scheme even under the presence of outliers in the training data. (2) We include the region mapping matrix and region matching weights, enabling the effective regional matching of colors and textures, in the estimation process. (3) We systematically determine the feature fusion weights for testing by integrating multiple features used in the training stage. (4) We build the complete system to track humans across the cameras deployed in the real world based on camera link models, which can be continuously updated in the testing stage in order to refine the model and to adapt to environmental changes. (5) We present a scalable and self-organized multiple-camera tracking system which automatically and unsupervisedly identifies the links between cameras, estimates the camera link model, and tracks objects across the cameras. The only prior information is the user specified GPS locations of the cameras.

9.2 Camera Link Identification

The camera link model between each pair of directly-connected cameras has been shown to be effective in tracking human across multiple cameras [8, 16, 22]. Before the camera link model is estimated, the prior knowledge needed is to identify which pairs of cameras within the camera network should possess a camera link model. In this section we introduce how our system detects the existence of links given the GPS locations of the cameras. The camera link identification includes the following two modules: link existence detection and connected zones identification.

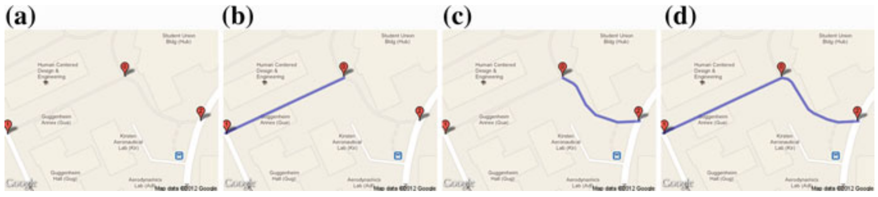


Fig. 9.2 An example of the routing associated with camera 0, 1, and 2. **a** The locations of three cameras. The shortest route between **b** camera 0 and 1, **c** camera 0 and 2, **d** camera 1 and 2

9.2.1 Link Existence Detection

Given the locations of the cameras, which are easily obtained when setting up the cameras in the environment, we are able to access the routing information provided by Google Maps. The routing information contains the possible routes between any pair of two locations. If there exists one route that connects two cameras without passing by another camera, we recognize them as directly-connected, and there should be a link between them. If all the routes between two cameras pass by at least one other camera, we recognize the link should not exist between these two cameras. Figure 9.2 shows an example of the routing associated with three cameras denoted by C_0, C_1, and C_2, whose locations are shown in Fig. 9.2a. To our observation, in practice people tend to follow the similar paths due to the presence of available pathway, obstruct, and shortest route, so it is reasonable to utilize the estimated paths from Google Maps as the routing information. Figure 9.2b–d show the shortest routes between each pair of cameras. Since the route between C_1 and C_2 passes C_0, the system only detects the links between C_0 and C_1 as well as C_0 and C_2.

9.2.2 Connected Zones Identification

There may be several entry/exit zones within a camera's view. The link between two directly-connected cameras actually only connects one zone each in these two cameras. So far, we can only know the existence of the link from the link existence detection without knowing the specific zones that are connected together. If we can know the connected zones, we only need to collect the training data, i.e., the exit and entry observations, from the associated zones so as to reduce large number of outliers in the training data during the estimation of camera link model and also obtain better accuracy when tracking the objects.

First of all, all the zones within each camera are detected in an unsupervised manner by using the Gaussian Mixture Model (GMM) based on the collected entry/exit measurements [22]. Then, we match the camera's view with the panoramic images automatically retrieved from Google Street View to estimate the principal orientation of the camera. The scheme of the street view matching is described as follows:

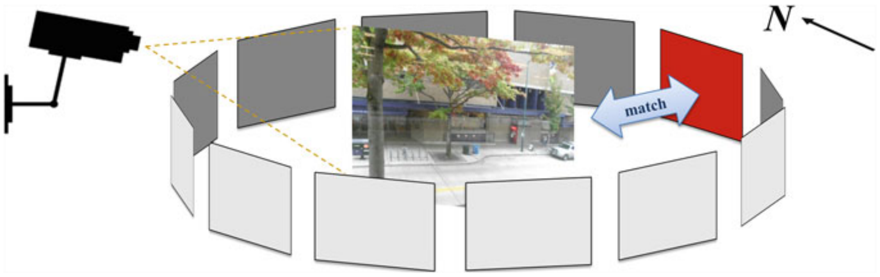


Fig. 9.3 The estimation of principal orientation of the camera

(i) given a GPS location, the system can access the images from Google Street View with different viewing angles θ, pitches φ, and foveations f. (ii) Perform feature point matching between the images and camera's view. (iii) identify the image with the maximum number of the matched points, and the corresponding viewing angle θ offers the principal orientation of the camera. Figure 9.3 shows an illustration of the scheme, where the camera's view matches well with one of the panoramic images. Moreover, the direction of the route is provided by Google Maps according to the GPS locations. Therefore, given the principal orientations, the direction of the route, and the detected entry/exit zones, we can determine the two zones that are connected together.

Figure 9.4 shows an example of the idea. To obtain the information from Google Maps, we implement a user interface by using Google Maps APIs 3.0.[1] In the street view matching, we adopt SIFT feature [23] for our point matching algorithm. We divide the viewing angle θ into 24 segments, i.e., $\theta = 15^\circ \times k, k = 0, 1, 2, \ldots, 23$. For each angle, we retrieve a set of 9 images which are the combination of 3 different pitches φ and 3 different foveations f ($\varphi = -20, -10, 0$; $f = 80, 100, 120$). The image resolution used is 640×480. Figure 9.4a is the camera's view, and the entry/exit zones are marked as red ellipses. Four panoramic images with different θ from Google Street View are shown in Fig. 9.4b. Figure 9.4c shows the result of the SIFT feature similarity matching, where the red dot block is the ground truth of the principle orientation obtained manually. One can see that the number of the matched points are relatively high from 105° to 120°, which is close to the ground truth. Since $\theta = 0^\circ$ stands for the orientation toward north, the principal orientation of the camera is estimated as toward East, and the left entry/exit zone is at North side of the view while the right one is at South side of the view. We tried 13 cameras, and all of their principal orientations can be determined well through the matching against Google Street View.

[1] https://developers.google.com/maps/documentation/streetview/.

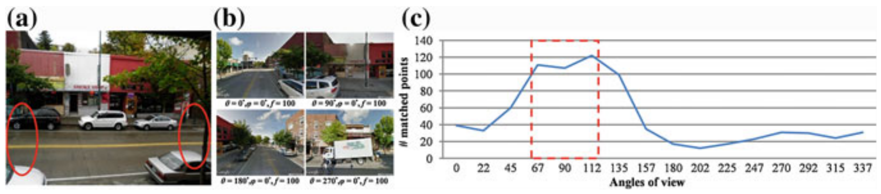


Fig. 9.4 **a** Camera's view. *Red ellipses* are the entry/exit zones. **b** Four panoramic images from Google Street View. **c** Number of the matched SIFT features. *Red dot* block is the ground truth of the camera orientation

9.3 Overview of Camera Link Model

The overall system shown in Fig. 9.1 is divided into the training stage and the testing stage:

9.3.1 Training Stage

Given the camera topology, the directly-connected camera pairs and the corresponding entry/exit zones can be manually identified easily. Training takes place pairwisely between all the corresponding entry/exit zones pairs of directly-connected cameras. For each pair, the training data, i.e., two observation sets from a pair of corresponding entry/exit zones, is automatically collected by the single-camera tracking module [13]. Each entry/exit observation contains temporal, color and texture features of a person who is entering/leaving the field of view (FOV) of a particular camera. As mentioned earlier, to construct a camera link model $\mathbb{M}$ between two zones, the correspondence in the training data needs to be automatically identified first. Denote $\mathbf{X}$ and $\mathbf{Y}$ as the exit and entry observations, within a specific time window, from a pair of corresponding entry/exit zones in a pair of directly-connected zones (in this chapter, we used connected zones and connected cameras interchangeably), respectively:

$$\mathbf{X} = \left[\mathbf{x}_1 \ldots \mathbf{x}_{N_1}\right], \mathbf{Y} = \left[\mathbf{y}_1 \ldots \mathbf{y}_{N_2}\right], \tag{9.1}$$

where $\mathbf{x}_i$ and $\mathbf{y}_j$ are exit and entry observations, and N_1 and N_2 are the numbers of the observations. Inspired by the concept of feature points matching between two images [19, 20], we formulate the identification of the correspondence as finding an $(N_1 + 1) \times (\mathrm{N}_2 + 1)$ binary matrix $\mathbf{P}$, where each row and column stand for an exit and entry observation, respectively. The entry P_{ij} in $\mathbf{P}$ is 1 if $\mathbf{x}_i$ corresponds to $\mathbf{y}_j$; otherwise, it is 0. The $(N_1 + 1)$th row and the $(N_2 + 1)$th column represent the outliers. Between a pair of directly connected zones, exit observations are people who have left one camera's view, and entry observations are people who have entered into the other camera's view within the same period of time window. The matched pairs are a pair of exit and entry observations who are the same person. Outliers are the remaining exit/entry observations who only appear in exactly one camera between this pair of cameras, either an exit observation from one camera never enters the other camera later within the specified time window period or an entry observation to one camera did not exit from the other camera earlier within the specified time window period. Note that $P_{N_1+1,\ N_2+1}$ has no physical meaning, so all the following discussion will exclude it automatically.

Analogous to the affine transformation in 2D image points matching [19, 20], the camera link model serves as the transformation between the observations from two cameras. The correspondence matrix **P** is estimated based on the distances between the feature vectors from the observation sets, and the camera link model (transformation) is required to compensate for the deviation between the cameras during the distances calculation. Moreover, the camera link model $\mathbb{M}$ can be estimated given a correspondence matrix **P**. Hence, it is reasonable to employ an EM-like approach to iteratively estimate the matrix **P** and the camera link model $\mathbb{M}$ (see Fig. 9.1). In each iteration, **P** is obtained by solving a minimization problem given the most recently estimated $\mathbb{M}$. After that, the model $\mathbb{M}$ is estimated based on the newly updated **P**. In Sects. 9.4 and 9.5, we will introduce the formulation of the minimization problem and the estimation of the camera link model including transition time distribution, brightness transfer function, region mapping matrix, region matching weights, and feature fusion weights. One will see that our estimation procedure is quite general. People can always add more features and estimate the corresponding transformation in the camera link model. The unsupervised learning scheme will take care of the estimation of the unknown parameters adaptively and systematically.

9.3.2 Testing Stage

In the testing stage, each camera $C_i, i = 1 \sim N_C$ maintains an exit list $\mathbf{L}_{i,k}$ for each entry/exit zone k, within a T_{max}-second interval. It consists of the observations $O_{i,k}$ of the people who have left the FOV from zone k within this T_{max}-second interval, i.e.,

$$\mathbf{L}_{i,k} = \left\{ O_{i,k}^{1}, O_{i,k}^{2} \ldots O_{i,k}^{|\mathbf{L}_{i,k}|} \right\}. \tag{9.2}$$

Whenever a person enters a camera's view, the system finds the best match among the people in the exit lists corresponding to the linked zones of the directly-connected cameras. Based on the camera link model, the matching distance between two observations O_1 and O_2 can be computed as the weighted sum of distances:

$$match\_dist = \sum_{i=1}^{N_{feature}} \alpha_i \times feature\_dist_i(O_1, O_2), \tag{9.3}$$

where α_i is the weight for the distance $feature\_dist_i(\cdot)$ corresponding to the feature i. In our system, four different features ($N_{feature} = 4$), namely, temporal, holistic color, region color and region texture features, are utilized to compute distances. Given an entrance observation, if the lowest distance among all the competing exit observations from all directly-connected cameras within the time interval is smaller than a certain threshold, the label handoff is performed; otherwise, we will treat it as a new person within the camera network. The re-identification results can be further used to update the camera link models.

9.4 Minimization Problem Formulation

Given the most recently estimated camera link model $\mathbb{M}$, the optimum correspondence matrix $\widehat{\mathbf{P}}$ can be obtained by solving a constrained minimization integer programming problem:

$$\widehat{\mathbf{P}} = \underset{\mathbf{P}}{argmin}\, J(\mathbf{P}) \tag{9.4}$$

$$s.t.\;\; P_{ij} \in \{0,1\} \qquad \forall\, i \le N_1 + 1,\; j \le N_2 + 1, \tag{9.5}$$

$$\sum_{i=1}^{N_1+1} P_{ij} = 1 \;\;\; \forall\, j \le N_2, \sum_{j=1}^{N_2+1} P_{ij} = 1 \;\;\; \forall\, i \le N_1, \tag{9.6}$$

where $J(\cdot)$ is the objective function to be minimized. The constraints (9.5) and (9.6) enforce one-to-one correspondence (except for the outlier row and column). By incorporating the soft-assign [19] instead of hard decision all the time, the problem is relaxed by substituting constraint (9.5) with (9.7)

$$P_{ij} \ge 0 \qquad \forall\, i \le N_1 + 1,\; j \le N_2 + 1. \tag{9.7}$$

In this way, the variables P_{ij} are continuous real numbers indicating how likely the ith exit and the jth entry observations are a matched pair. Moreover, the relaxation reduces the chance of getting trapped in poor local minima during the optimization search. By incorporating the deterministic annealing method, the solution eventually converges at a binary permutation matrix [19]. The objective function $J(\cdot)$ comprises several cost functions, and each of them stands for a distance function between the exit and entry observations associated with one specific feature, e.g., time, color, texture. In the following, we will introduce separately the cost functions considered in the objective function and explain how a camera link model associates with different features.

9.4.1 Temporal Feature

According to our observation, people tend to follow similar paths in most cases due to the presence of available pathways, obstructs, or shortest routes. Thus, the transition time t forms a certain distribution $f_{tran}(t)$. Given previously estimated $\mathbf{P}$, we can get the transition time values $\mathbf{T}_{tran} = \left[t_1^{exit} \dots t_{N_1}^{exit}\ t_1^{entry} \dots t_{N_2}^{entry} \right]$, where

$$t_i^{exit} = \sum_{j=1}^{N_2} P_{ij} \left(y_j^t - x_i^t \right) \qquad \forall\, i \le N_1, \tag{9.8}$$

$$t_j^{entry} = \sum_{i=1}^{N_1} P_{ij} \left(y_j^t - x_i^t \right) \qquad \forall\, j \le N_2, \tag{9.9}$$

and y_j^t and x_i^t represent the time stamps of the observations $\mathbf{y}_j$ and $\mathbf{x}_i$, respectively. The transition time is always positive if two cameras have no overlapping area because the entry time of a person should be greater than the exit time of the correct correspondence from the other camera. Also, the outliers should lead to zero transition time. For example, if the kth exit observation is an outlier, P_{kj} should be zero for all $j \le N_2$ resulting in $t_k^{exit} = 0$ according to (9.8). Hence, the optimal solution of $\mathbf{P}$ satisfies the constraints $t_i^{exit} \ge 0$, $t_j^{entry} \ge 0$ and $P_{ij} = 0$ if $y_j^t - x_i^t \le 0$, which are required to be included in the problem formulation. A set of valid time values $\mathbf{T}_{valid}$ that excludes the outliers, i.e., takes only the nonzero entries in $\mathbf{T}_{tran}$, is further established, and the transition time distribution $f_{tran}(\cdot)$ is built based on the kernel density estimation:

$$\mathbf{T}_{valid} = \left\{ \hat{t} \mid \hat{t} \neq 0, \hat{t} \in \mathbf{T}_{tran} \right\} = \left[t_1 \dots t_{N_{valid}} \right], \tag{9.10}$$

$$f_{tran}(t) = \frac{1}{N_{valid}} \sum_{i=1}^{N_{valid}} \frac{1}{\sigma_{tran}\sqrt{2\pi}} exp\left(-\frac{(t-t_i)^2}{2\sigma_{tran}^2} \right), \tag{9.11}$$

where σ_{tran}^2 is the predefined variance of the Gaussian kernel. For each possible correspondence, we compute the likelihood value $f_{tran}\left(y_j^t - x_i^t \right)$ given the model and consider the maximum likelihood estimation, i.e., $\left(1 - f_{tran}\left(y_j^t - x_i^t \right) \right)$ is used as the individual cost. Thus, the total cost can be written as:

$$cost_{time} = \sum_{i=1}^{N_1}\sum_{j=1}^{N_2} P_{ij}\left(1 - f_{tran}\left(y_j^t - x_i^t\right)\right) \tag{9.12}$$
$$= \sum_{i=1}^{N_1}\sum_{j=1}^{N_2} P_{ij} feature\_dist_1(\mathbf{x}_i, \mathbf{y}_j).$$

9.4.2 Holistic Color Feature

The same object may appear differently under two cameras with non-overlapping views due to illumination changes and different camera color responses. The color deviation can be modeled as a brightness transfer function (BTF) [9]. The BTF is applied to compensate for the color difference between two cameras before we compute the distance between the holistic color histograms of two observations. Thus, the total cost function for the holistic color feature is:

$$cost_{holistic\_color} = \sum_{i=1}^{N_1}\sum_{j=1}^{N_2} P_{ij} D\left(f_{BTF}\left(\mathbf{y}_j^h\right), \mathbf{x}_i^h\right) \tag{9.13}$$
$$= \sum_{i=1}^{N_1}\sum_{j=1}^{N_2} P_{ij} feature\_dist_2(\mathbf{x}_i, \mathbf{y}_j),$$

where $\mathbf{y}_j^h \in \mathbb{R}^d$ and $\mathbf{x}_i^h \in \mathbb{R}^d$ are the holistic color histograms of the observations, $\mathbf{y}_j$ and $\mathbf{x}_i$; $D(\cdot)$ is the distance function between two histograms; $f_{BTF}(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is the BTF, with d being the total bin number of the color histogram.

9.4.3 Region Color and Texture Feature

Since the viewpoints vary in two cameras, some parts of the human body may only be seen in either one of the cameras' views. Hence, we divide the human into multiple regions for more detailed comparison. However, the corresponding regions do not always cover the same area of the human due to different viewpoints (see Fig. 9.5). We observe that the entering (or exiting) directions of different people at an entry/exit zone of a fixed camera are similar, so we use a mapping matrix to link the regions between two bodies. The histogram extracted from one region of human leaving from the first camera can be modeled as the linear combination of the histograms extracted from multiple regions of human entering into the second camera. The snapshots for a person exiting one camera and entering another are shown in Fig. 9.5b and c, respectively.

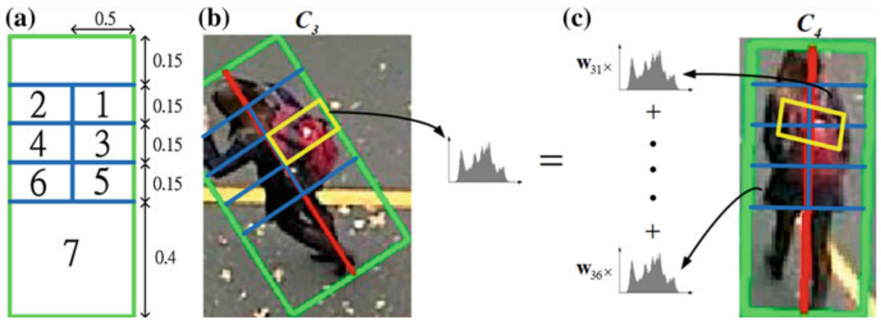


Fig. 9.5 **a** *Green box* is the bounding box of the target. The target is divided into 7 regions (exclude the head) based on the shown ratios. Each region has predefined label number. **b** An exit observation in camera C_3. **c** An entry observation of the same person in camera C_4. The *red line* is the principal axis. Region 3 in **b** and region 3 in **c** do not cover the same areas. The *yellow rectangles* cover the same areas on the target. The histogram extracted from region 3 in **b** can be modeled as the linear combination of the histograms extracted from six regions in **c**, and the coefficients are $w_3 = [w_{31} \dots w_{36}]^T$

First, the principal axis of a person is identified by applying principal component analysis to the human silhouette which is obtained from the single camera tracking module [13]. After that, the whole body is automatically divided into head, torso, and leg regions based on the predefined ratios (Fig. 9.5a). We discard the head region in the region matching since this part usually has lower discriminability due to its relatively small area and similar hair/face color information. The torso is further divided into six regions, and the mapping matrix will be trained for linking two six-regions from a pair of matched people. Because the leg region usually changes little under different perspectives, we compute the distance between the two whole leg regions without further dividing it. The region color histograms extracted from the region k of the observations x_i and y_j are denoted as $x_i^{rh_k} \in \mathbb{R}^d$ and $y_j^{rh_k} \in \mathbb{R}^d$, respectively, where $k = 1 \sim 7$. As shown in Fig. 9.5, the regions 1–6 are from the torso, and region 7 is from the leg. We denote the mapping matrix $\mathbf{W}_{map} \in \mathbb{R}^{6\times 6}$ as:

$$\mathbf{W}_{map} = [\mathbf{w}_1 \dots \mathbf{w}_6]\,, \tag{9.14}$$

where $\mathbf{w}_k \in \mathbb{R}^6$ is the weighting for linear combination.

Moreover, since some regions may not be visible under both cameras' views, they should be assigned with smaller weights in the region feature distance computation. The cost function is the weighted sum of the distances from all 7 regions

$$cost_{\mathbf{r}egion\_\mathbf{c}olor} = \sum_{i=1}^{N_1}\sum_{j=1}^{N_2} P_{ij}\left[\sum_{k=1}^{6} q_k \times D\left(\mathbf{y}_j^{map_k}, \mathbf{x}_i^{rh_k}\right) + q_7 \times D\left(f_{BTF}\left(\mathbf{y}_j^{rh_7}\right), \mathbf{x}_i^{rh_7}\right)\right]$$

$$= \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} P_{ij} feature\_dist_3(\mathbf{x}_i, \mathbf{y}_j), \tag{9.15}$$

where $\mathbf{y}_j^{map_k} \in \mathbb{R}^d$ is the linear combination of the torso region color histograms $\left[\mathbf{y}_j^{rh_1} \dots \mathbf{y}_j^{rh_6}\right] \in \mathbb{R}^{d\times 6}$ with weights $\mathbf{w}_k$ after applying the BTF,

$$\mathbf{y}_j^{map_k} = \left[f_{BTF}\left(\mathbf{y}_j^{rh_1}\right) \dots f_{BTF}\left(\mathbf{y}_j^{rh_6}\right)\right] \mathbf{w}_k, \tag{9.16}$$

and $\mathbf{q} = [q_1 \dots q_7]^T$ denote the weights for all 7 region distances. Note that all the seven regions are included in the distance computation, but only the torso regions are considered for the region mapping by using the mapping matrix $\mathbf{W}_{map}$.

The texture feature is considered in the similar manner. The local binary pattern (LBP) [24] is utilized as the texture feature and is expressed as r-dimensional LBP histograms $\mathbf{x}_i^{rLBPh_k} \in \mathbb{R}^r$ and $\mathbf{y}_j^{rLBPh_k} \in \mathbb{R}^r$, where $k = 1 \sim 7$. Hence, the cost function is:

$$\begin{aligned} &\mathbf{c}ost_{\mathbf{r}egion\_\mathbf{t}exture} = \\ &\sum_{i=1}^{N_1} \sum_{j=1}^{N_2} P_{ij} \left[\sum_{k=1}^{6} q_k \times D\left(\mathbf{y}_j^{mapLBP_k}, \mathbf{x}_i^{rLBPh_k}\right) + q_7 \times D\left(\mathbf{y}_j^{rLBPh_7}, \mathbf{x}_i^{rLBPh_7}\right) \right] \\ &= \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} P_{ij} feature\_dist_4(\mathbf{x}_i, \mathbf{y}_j), \end{aligned} \tag{9.17}$$

where $y_j^{mapLBP_k} \in \mathbb{R}^r$ is the linear combination of torso region LBP histograms $\left[y_j^{rLBPh_1} \dots y_j^{rLBPh_6}\right] \in \mathbb{R}^{r\times 6}$ with weights w_k.

$$\mathbf{y}_j^{mapLBP_k} = \left[\mathbf{y}_j^{rLBPh_1} \dots \mathbf{y}_j^{rLBPh_6}\right] \mathbf{w}_k \tag{9.18}$$

Since the LBP is robust to the brightness change [21], the BTF is not applied here.

9.4.4 Maxima Entropy Principle

In deterministic annealing [18, 25], a widely used iterative scheme to solve optimization problems, the procedure starts with emphasizing high "uncertainty", measured as the entropy, of the entries in **P**, i.e., to maximize the entropy. The importance of the maximum entropy principle is gradually decreased by increasing a parameter β through the iterations. Thus, the cost function is written as the negative of the entropy:

$$cost_{entropy} = \frac{1}{\beta}\sum_{i=1}^{N_1+1}\sum_{j=1}^{N_2+1} P_{ij} log P_{ij}. \tag{9.19}$$

In the early stage of the training process, the factor β starts with a low value that raises the importance of this cost function with respect to the overall objective function $J(\mathbf{P})$, enabling the value P_{ij} to move freely in the space for searching the optimum. β is then gradually increased to lower the importance and eventually leads to convergence. The function (9.19) can also be seen as a barrier function [26] for the constraint defined in (9.7).

9.4.5 Outlier Cost

Since the presence of outliers is considered, there is an additional term which controls how large the distance we can tolerate before treating a particular observation as an outlier. The penalty term is:

$$cost_{outlier} = -\theta\sum_{i=1}^{N_1}\sum_{j=1}^{N_2} P_{ij}, \tag{9.20}$$

where θ is a control factor. If θ is set large, for instance, the estimation process is allowed to tolerate larger distance before treating one as an outlier.

9.4.6 Estimation of $\mathbf{P}$

By incorporating all the cost functions above, our final problem formulation becomes a convex optimization problem:

$$\widehat{\mathbf{P}} = \underset{P}{argmin}\ J(\mathbf{P}) \tag{9.21}$$

$$s.t.\ \ P_{ij} \geq 0 \quad \forall\, i \leq N_1 + 1,\ j \leq N_2 + 1, \tag{9.22}$$

$$\sum_{i=1}^{N_1+1} P_{ij} = 1 \ \ \forall\, j \leq N_2, \sum_{j=1}^{N_2+1} P_{ij} = 1 \ \ \forall\, i \leq N_1, \tag{9.23}$$

$$t_i^{exit} = \sum_{j=1}^{N_2} P_{ij}\left(y_j^t - x_i^t\right) \geq 0 \qquad \forall\, i \leq N_1, \tag{9.24}$$

$$t_j^{entry} = \sum_{i=1}^{N_1} P_{ij} \left(y_j^t - x_i^t \right) \geq 0 \qquad \forall\, j \leq N_2, \tag{9.25}$$

$$P_{ij} = 0 \quad if \quad y_j^t - x_i^t \leq 0 \qquad \forall\, i \leq N_1,\ j \leq N_2. \tag{9.26}$$

The objective function $J\,(\mathbf{P})$ is the combination of the above cost functions:

$$\begin{aligned} J\,(\mathbf{P}) = cost_{time} + cost_{holistic\_color} + cost_{region\_color} \\ + cost_{region\_texture} + cost_{entropy} + cost_{outlier}. \end{aligned} \tag{9.27}$$

Given the current camera link model $\mathbb{M} = \left\{ f_{tran}, f_{BTF}, \mathbf{W}_{map}, \mathbf{q} \right\}$, the objective function is formulated, and P is updated by solving (9.21)–(9.27). Instead of incorporating the barrier function for constraints (9.24) and (9.25) as is done in our previous work [19], we use a projection-based convex optimization method [26] to deal with the constraints (9.24)–(9.26) since it leads to faster convergence in practice. First of all, given the current estimated camera link model, the objective function $J\,(P)$ is convex in P, so the optimum can be obtained by equating the derivative to zero,

$$\frac{\partial J\,(P)}{\partial P_{ij}} = 0 \rightarrow P_{ij} = \begin{cases} e^{(\beta(\theta - dist_{ij}) - 1)} & if \quad i \leq N_1 \; and \; j \leq N_2 \\ e^{-1} & if \quad i = N_1 + 1 \; or \quad j = N_2 + 1 \end{cases}, \tag{9.28}$$

where $dist_{ij} = \sum_{k=1}^{N_{feature}} feature\_dist_k(\mathbf{x}_i, \mathbf{y}_j)$ is the sum of distances. After that, we project the solution onto the feasible space in which all the constraints are satisfied. Specifically, (i) Assign the zero values to particular elements to satisfy the constraints (9.24)–(9.26), like

$$P_{ij} = \begin{cases} 0, & y_j^t - x_i^t \leq 0, \quad \forall\, i \leq N_1 \; and \; j \leq N_2 \\ P_{ij}, \; otherwise & (include \; outlier \; row \; and \; column) \end{cases}. \tag{9.29}$$

(ii) Perform alternately row-column normalization based on Sinkhorn's theorem [18, 27] for constraints (9.23). (iii) The cost function (9.19) (maximum entropy) can be seen as a barrier function for the constraint (9.22) that makes the solution exponential as shown in (9.28), which is always nonnegative, so that the constraint (9.22) always holds. In this way, $\mathbf{P}$ can be updated based on the most recently estimated camera link model $\mathbb{M}$ and the current value of β. Finally, we adopt an innovative EM-like algorithm to sequentially solve the matrix $\mathbf{P}$ and the camera link model in each iteration. Deterministic annealing is also employed in our estimation process to obtain the optimal solution [8].

The information obtained from Google Maps not only provides the routes between two locations but also gives an estimation of the traveling time between them. It enables us to pose a good initial state of the matrix P before the estimation process starts. Denote the values of the traveling time estimated by Google Maps as s_i,

$i = 1 \sim K$. Note that K may be greater than one if there exist multiple alternative routes between two cameras. We can have

$$P_{ij} = \begin{cases} 0, \quad y_j^t - x_i^t \le 0, \quad \forall i = 1 \sim N_1\,, \ j = 1 \sim N_2 \\ \frac{1}{K} \sum\limits_{m=1}^{K} exp\left(-\frac{\left(y_j^t - x_i^t - s_m\right)^2}{2\sigma^2} \right), \qquad otherwise \end{cases} \tag{9.30}$$

Since the traveling time is always positive if two cameras have nonoverlapping area, if the entry time stamp is smaller than the exit time stamp $\left(y_j^t - x_i^t \le 0\right)$, it is not possible for them to be a matched pair, hence P_{ij} is set as 0. Otherwise, we assume it takes people roughly the estimated amount of time s_i to move from one camera to the other, so P_{ij} is set as the likelihood based on a parzen window built by s_m. By incorporating this information as prior knowledge to the estimation process, it enables the system to reach the convergence with fewer iterations than are required in [8].

9.5 Experimental Results

9.5.1 Camera Link Model Estimation

We set up four cameras C_1–C_4 around the department building shown in Fig. 9.6, and the FOVs of the cameras are spatially disjointed (non-overlapping). Cameras do not need any calibration in advance. The only prior knowledge is the GPS locations of the cameras which is accessible in real world practice.

Figure 9.7 shows the estimation result of the transition time distribution between camera C_2 and camera C_3. There are two 12-minute videos collected from two cameras as training data, including 104 exit observations from camera C_2 and 40 entry observations from camera C_3, where 33 pairs of people are matched pairs, i.e., there are total 78 outliers (54 %). The results of other approaches [15–18] and the ground truth are also shown in the figure. The ground truth is obtained by manually labeling the correspondence and estimate the transition time based on (9.11), namely supervised learning as in [10] and [11]. Table 9.1 shows the quantitative error report of the above simulations. The error is calculated as the distance between the estimated distribution and the ground truth. One can see that our estimation has the smallest error to the ground truth.

Figure 9.8 gives an example of the estimation results of the brightness transfer functions. Four curves correspond to four different links, and only the BTFs of the blue color channel of each are shown here for demonstration purpose. We can see that the color deviation does exist between different cameras. We do not compare the results with the ground truth, since it is not easy for human to determine the ground truth of region mapping matrix $\mathbf{W}_{map}$ and region matching weights $\mathbf{q}$, which

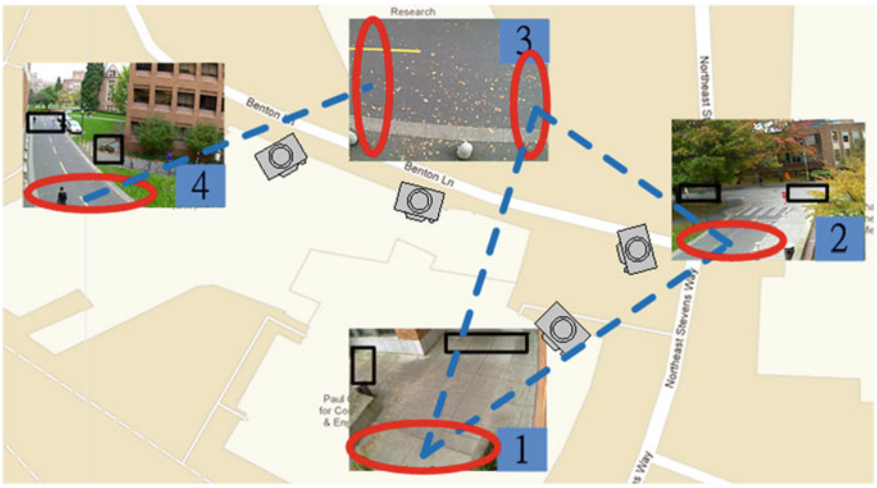


Fig. 9.6 Camera topology. Four links are denoted as *blue broken lines*, and the corresponding entry/exit zones are denoted as *red ellipses*. *Black rectangles* are the other entry/exit zones which do not have any link between them

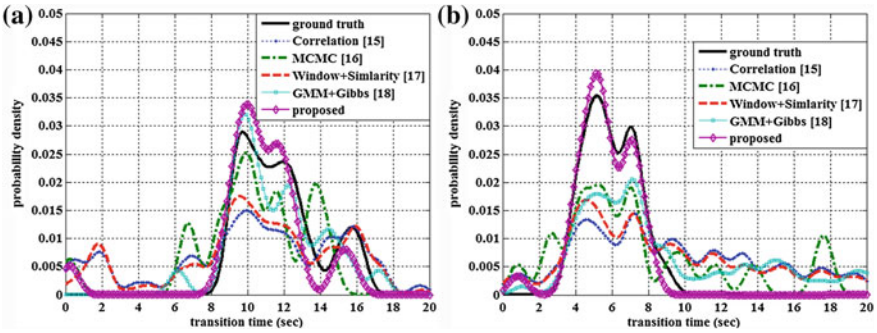


Fig. 9.7 Comparison of the distributions of the transition time **a** between camera C_1 and camera C_2, and **b** between camera C_2 and camera C_3

are necessary for obtaining the ground truth BTF. Hence, the ground truth of BTF is not shown here. However, we can justify the effectiveness of BTF and region mapping matrix $\mathbf{W}_{map}$ by examining the region histogram matching. For example, the histogram (blue curve in Fig. 9.9) of region 3 in Fig. 9.5b is compared with the ones with and without applying camera link model. The red curve in Fig. 9.9 is the one after applying BTF and region mapping matrix to the histograms in Fig. 9.5c. The green curve is the histogram of region 3 in Fig. 9.5c. By applying the correct camera link model, BTF and region mapping matrix $\mathbf{W}_{map}$, the one with the camera link model applied gets the better matching which gives preferable similarity measurement between the same object under two cameras.

Table 9.1 Error of transition time distribution

Error comparison	Camera C_2 and C_3	Camera C_1 and C_2	Camera C_1 and C_3	Camera C_3 and C_4	i-LIDS dataset
Correlation [15]	0.1074	0.0787	0.12	0.1084	0.1170
MCMC [16]	0.0824	0.0773	0.0903	0.052	0.1208
Window + similarity [17]	0.0952	0.0688	0.1076	0.1057	0.1229
GMM+Gibbs [18]	0.0738	0.0368	0.0734	0.029	0.0861
Proposed	0.0158	0.0339	0.0474	0.0173	0.0586

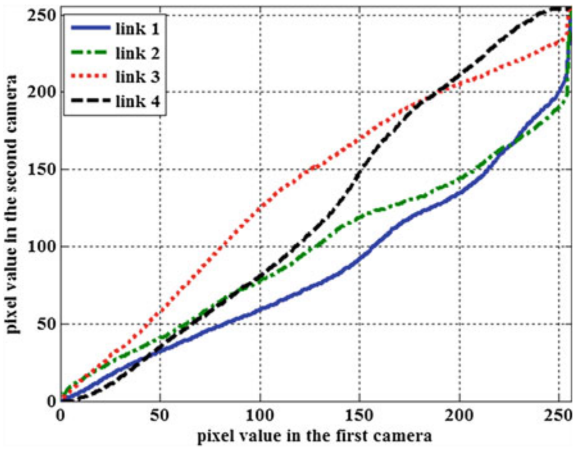


Fig. 9.8 Brightness transfer function. *Note* only one channel is shown here for demonstration purpose

We further implement an automatic multiple-camera tracking system that tracks humans across cameras based on the learned camera link models. The method in [13] is utilized to accomplish the tracking within a single camera. One can see in Fig. 9.4, the uncertainty of the exit and entry events increases the difficulty of the tracking. For example, a person exiting from camera C_1 can enter into camera C_2 or camera C_3. In our 20-min testing video, there are 336 people appearing in the deployed camera network. The accuracy reported in Table 9.2 shows the results of using different feature combinations. The re-identification accuracy is defined as the fraction of the people being correctly labeled. By incorporating the estimated traveling time from Google Maps Eq. (9.30), the number of the required iteration in the camera link model estimation process drops about 11 % compared to [8].

Since our system is based on the pairwise learning and tracking scheme, the system can be scaled up easily. Here we present a simple scenario to illustrate the scalability of the system. Assume there are N_C cameras C_i, $i = 1 \sim N_C$, already in the network, and we would like to add one camera C_{N_C+1} in the network. Providing the new camera's location, the system automatically identifies the links and the

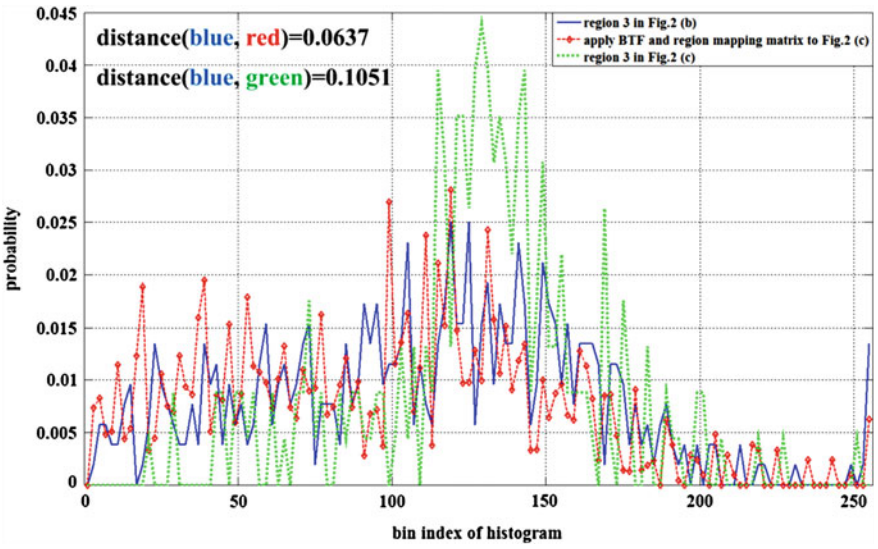


Fig. 9.9 Histogram comparison. *Blue* the histogram from region 3 in Fig. 9.2b. *Red* the histogram after applying BTF and region mapping matrix to Fig. 9.2c. *Green* the histogram from region 3 in Fig. 9.2c. *Note* only one channel is shown here for demonstration purpose

Table 9.2 Re-identification accuracy under different feature sets combination

Feature set	1	2	3	4
Accuracy (%)	62.8	61.6	58.9	47.6
Feature set	1,2	2, 3	1, 3, 4	
Accuracy (%)	69.0	66.7	73.8	
Feature set	1, 2, 3, 4 with uniform fusion weight		1, 2, 3, 4 (proposed)	
Accuracy (%)	72.9		79.5	

feature 1 temporal, *2* holistic color, *3* region color, *4* region texture
Features are combined with normalized adaptive weights except the one with explicitly noted

connected zones between C_{N_C+1} and the other cameras. After that, the camera link model estimation is performed pairwisely for those newly created links. By applying the models, tracking across multiple cameras is carried out within this new camera network. Following the similar manner, the camera network can be scaled up without human intervention.

9.6 Conclusion

We propose a tracking system that tracks humans across multiple cameras with disjointed FOVs based on the application of camera link models. The camera link model, including transition time distribution, brightness transfer function, region mapping matrix, region matching weights, and feature fusion weights, is correctly estimated by an unsupervised scheme even under the presence of outliers in the training data. Our estimation procedure can be generalized easily by adding more features and the corresponding transformation in the camera link model. The unsupervised learning scheme will take care of the estimation of the unknown parameters adaptively and systematically. The proposed method is applied in a self-deployed camera network in the real world. By providing the GPS locations of uncalibrated cameras and incorporating with Google Maps and Google Street View, our system automatically identifies the camera links within the camera network, estimates the camera link models for pairwise zones, and performs multiple-camera tracking. The pairwise learning and tracking scheme enables the system to be self-organized and be scaled up efficiently.

References

1. Farenzena M, Bazzani L, Perina A, Murino V, Cristani M (2010) Person re-identification by symmetry-driven accumulation of local features. In: Proceedings of IEEE conference on computer vision and pattern recognition, pp 2360–2367
2. Gheissari N, Sebastian TB, Tu PH, Rittscher J, Hartley R (2006) Person reidentification using spatiotemporal appearance. In: Proceedings of IEEE conference on computer vision and pattern recognition, pp 1528–1535
3. Gray D, Tao H (2008) Viewpoint invariant pedestrian recognition with an ensamble of localize features. In: Proceedings of ECCV, pp 262–275
4. Bauml M, Stiefelhagen R (2011) Evaluation of local features for person re-identification in image sequences. In: IEEE international conference on advanced video and signal based surveillance
5. Zhang Y, Li S (2011) Gabor-LBP based region covariance desciptor for person reiidentification. In: IEEE international conference on image and graphics, pp 368–371
6. Jungling K, Arens M (2011) View-invariant person re-identification with an implicit shape model. In: IEEE international conference on advanced video and signal based surveillance, pp 197–202
7. Bak S, Corvee E, Bremond F, Thonnat M (2010) Person re-identification using Haar-based and DCD-based signature. In: IEEE international conference on advanced video and signal based surveillance
8. Chu C, Hwang J, Yu J, Lee K (2012) Tracking across nonoverlapping cameras based on the unsupervised learning of camera link models. In: ACM/IEEE international conference on distributed smart cameras
9. D'Orazio T, Mazzeo P, Spagnolo P (2009) Color brightness transfer function evaluation for non overlapping multi camera tracking. In: ACM/IEEE international conference on distributed smart cameras, pp 1–6
10. Prosser B, Gong S, Xiang T (2008) Multi-camera matching using bi-directional cumulative brightness transfer functions. In: British machine vision conference
11. Javed O, Rasheed Z, Shafique K, Shah M (2003) Tracking across multiple cameras with disjoint views. In: Proceedings of IEEE conference on computer vision, pp 952–957
12. Javed O, Shafique K, Rasheed Z, Shah M (2008) Modeling inter-camera space-time and appearance relationships for tracking across non-overlapping views. Comput Vis Image Underst 109(2):146–162

13. Chu C, Hwang J, Wang S, Chen Y (2011) Human tracking by adaptive Kalman filtering and multiple kernels tracking with projected gradients. In: ACM/IEEE international conference on distributed smart cameras
14. Makris D, Ellis T, Black J (2004) Bridging the gaps between cameras. In: Proceedings of the IEEE conference on computer vision and pattern recognition, vol 2, pp 205–210
15. Tieu K, Dalley G, Grimson W (2004) Inference of non-overlapping camera network topology by measuring statistical dependence. In: Proceedings of the IEEE conference on computer vision, vol 2, pp 1842–1849
16. Gilbert A, Bowden R (2006) Tracking objects across cameras by incrementally learning inter-camera colour calibration and patterns of activity. In: Proceedings of ECCV, pp 125–136
17. Huang C-C, Chiu W-C, Wang S-J, Chang J-H (2010) Probabilistic modeling of dynamic traffic flow across non-overlapping camera views. In: IEEE international conference on pattern recognition, pp 3332–3335
18. Wang X,Tieu K, Grimson W (2005) Correspondnece-free activity analysis and scene modeling in multiple camera views. IEEE Trans Pattern Anal Mach Intell 32:56–71
19. Chu C, Hwang J, Chen Y, Wang S (2012) Camera link model estimation in a distributed camera network based on the deterministic annealing and the barrier method. In: Proceedings of IEEE conference on ASSP, pp 997–1000
20. Chu C, Lee K, Hwang J (2013) Self-organized and scalable camera networks for systematic human tracking across nonoverlapping cameras. In: Proceedings of IEEE conference on ASSP
21. Chu C, Hwang J, Pai H-I, Lan K-M (2013) Tracking human under occlusion based on adaptive multiple kernels with projected gradients. IEEE Trans Multimedia 15(7):1602–1615 Nov
22. Chen K, Lai C, Lee P, Chen C, Hung Y (2011) Adaptive learning for target tracking and true linking discovering across multiple non-overlapping cameras. IEEE Trans Multimedia 13:625–638
23. Lowe DG (2004) Distinctive image features from scale-invariant keypoints. Intl J Comput Vis 60(2):91–110
24. Ojala T, Pietikainen M, Maenpaa T (2002) Multiresolution gray-scale and rotation invariant texture classificaiton with local binary patterns. IEEE Trans Pattern Anal Mach Intell 24(7):971–987
25. Rose K (1998) Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. Proc IEEE 86(11):2210–2239
26. Boyd S, Vandenberghe L (2004) Ch8.1 in convex optimization. Cambridge University Press, Cambridge
27. Sinkhorn R (1964) A relationship between arbitrary positive matrices and doubly stochastic matrices. Ann Math Statist 35(2):876–879
28. Chiu H, Rangarajan A (2003) A new point matching algorithm for non-rigid registration. Comput Vis Image Underst 89:114–141
29. Maciel J, Costeira J (2003) A global solution to sparse correspondence problems. IEEE Trans Pattern Anal Mach Intell 25(2):187–199

Chapter 10
Soft-Biometrics and Reference Set Integrated Model for Tracking Across Cameras

Xiaojing Chen, Le An and Bir Bhanu

Abstract Multi-target tracking in non-overlapping cameras is challenging due to the vast appearance change of the targets across camera views caused by variations in illumination conditions, poses, and camera imaging characteristics. Therefore, direct track association based on color information only is difficult and prone to error. In most previous methods the appearance similarity is computed either using color histograms directly or based on pre-trained Brightness Transfer Function (BTF) that maps color between cameras. In this chapter, besides color histograms, other soft-biometric features that are invariant to illumination and view changes are also integrated into the feature representation of a target. A novel reference set based appearance model is proposed to improve multi-target tracking in a network of non-overlapping video cameras. Unlike previous work, a reference set is constructed for a pair of cameras, containing targets appearing in both camera views. For track association, instead of comparing the appearance of two targets in different camera views directly, they are compared to the reference set. The reference set acts as a basis to represent a target by measuring the similarity between the target and each of the individuals in the reference set. The effectiveness of the proposed method over the baseline models on challenging real-world multi-camera video data is validated by the experiments.

X. Chen (✉) · L. An · B. Bhanu
University of California, Riverside, CA 92521 , USA
e-mail: xchen010@ucr.edu

L. An
e-mail: lan004@ucr.edu

B. Bhanu
e-mail: bhanu@cris.ucr.edu

10.1 Introduction

Recently, a major effort has been underway in the vision community to develop effective and automated video surveillance and monitoring systems [12, 20, 22, 23]. The requirement for surveillance cameras at public areas (e.g., airport, parking lots, and shopping malls) is increasingly growing. In most cases, it is not feasible to use a single camera to observe the complete area of interest, and using multiple cameras with overlapping field-of-views (FOVs) has high costs in both economical and computational aspects. Therefore, camera networks with non-overlapping field of views are widely adopted in the real world.

Multi-target tracking is an extensively explored topic in the surveillance and monitoring domain, as it is the foundation for many higher level applications, such as anomaly detection, activity detection and recognition [25], and human behavior understanding [4]. The goal of multi-target tracking is to estimate the trajectories of all moving targets and keep their identities consistent from frame to frame. In single camera tracking, successive observations of the same target often have large proximity in appearance, space and time [17]. However, it is not the case for tracking people across cameras with non-overlapping FOVs. The appearance of the same target may have large difference even in two adjacent cameras due to a sudden change in illumination conditions (e.g., from outdoor to indoor). Other aspects, such as variations in pose (e.g., frontal view to rear view) and camera imaging conditions further complicate the tracking task in multiple cameras. In Fig. 10.1 some sample frames are shown in which the appearance of the same target in different camera views differs significantly.

A possible way to tackle the appearance difference in multiple cameras is to learn Brightness Transfer Function (BTF) [5, 6, 9, 10, 14, 16] that is a mapping of color models between a pair of cameras. However, BTF is not suitable for a camera network that has a large within camera illumination change. For example, camera i and camera j both have dark and bright regions in their camera views. A BTF that is able to map colors in dark region of camera i (low brightness) to colors in bright region of camera j (high brightness) will not work well for mapping colors in bright region of camera i (high brightness) to dark region of camera j (low brightness).

In this chapter, to enhance the appearance model for tracking, in addition to color histogram, we use soft biometrics which are invariant to view and illumination changes to build a discriminative and robust appearance representation. Soft biometrics are characteristics that can be used to describe a person [8], for instance, height, weight, gender, hair color and clothes color. Although each one of them is not discriminative enough to uniquely identify an individual, when bundled as a whole they can provide a coarse representation of a target. Because soft biometrics can be directly acquired from surveillance videos without any subject's cooperation, they are suitable for constructing appearance models of tracked targets. Soft biometrics have been widely used for retrieval and recognition tasks on image datasets [8, 11, 13], target identification in surveillance video data [19], and person re-identification across cameras [2]. However, to the best of the authors' knowledge, soft biometrics

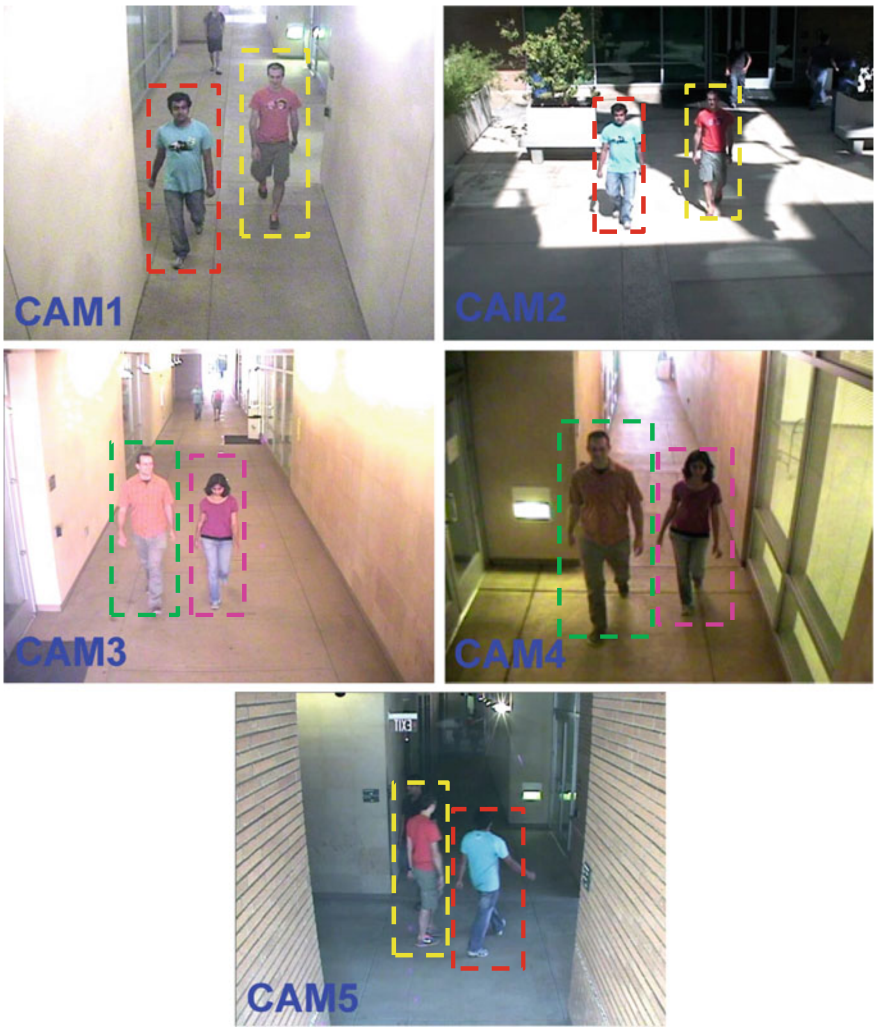


Fig. 10.1 Sample frames from each camera view. *Bounding boxes* with the same color indicate the same target. Notice that illumination may change drastically within camera and across cameras. As a result, the appearances of the same target may have significant variations

have not been used to improve tracking performance in a network of non-overlapping video cameras.

In addition, to further mitigate the ambiguities caused by illumination and pose changes, we propose a novel reference set based appearance model to estimate the similarity of multiple targets in different cameras. Based on the tracking results from a single camera, the goal is to associate tracks in different cameras that contain the same person. Our method is inspired by the recent advances in face verification/recognition [21, 24] and person re-identification [3] in which an external reference set or library is used to facilitate the matching process of the same objects in different imaging conditions. The reference set contains the appearance of individuals in different camera views under different imaging conditions. For tracking, instead of comparing two targets directly, targets from different cameras are compared to the individuals in the reference set. The individuals in the reference set act like basis functions and for a given target, its similarity to each of the individuals in the reference set is used as its new representation rather than the original low level color or texture features based representation.

10.2 Related Work

To cope with the illumination change in different camera views, BTF has been studied extensively [5, 6, 9, 10, 14, 16]. An incremental unsupervised learning method is proposed in [10] to model color variations and posterior probability distributions of spatial-temporal links between cameras in parallel. The model becomes more accurate over time with accumulated evidence. Prosser et al. [16] proposed a cumulative BTF to map color between different cameras and significant improvement over BTF-based methods is reported. Javed et al. [14] learn the inter-camera relationships using multivariate probability density of space-time variables. It is shown that BTFs from one camera to another camera lie in a low dimensional subspace and this subspace is learned for appearance matching. Chun et al. [5] built BTFs from the overlapping area during tracking to compensate for the color difference between camera views. In addition, the perspective difference is compensated for with tangent transfer functions (TTFs) by computing the homography between two cameras. Different methods are compared to evaluate the color BTFs between non-overlapping cameras and experimental results show BTFs limitations in people association when a new person enters in one camera's FOV [9]. To track people across non-overlapping cameras, Chun et al. [6] estimated a camera link model including BTF, transition time distribution, region mapping matrix/weight, and feature fusion weight in an unsupervised manner.

Compared to low-level features such as color histogram which may drift significantly over time, soft biometrics, such as gender, height, are rather stable with respect to changes in appearance, time, and motion. Soft biometrics contain high level semantic information, which has been used for recognition or retrieval tasks [2, 13, 19]. Jain et al. [13] used facial marks such as freckles and scars for improving face recognition. Reid and Nixon [19] use soft biometrics to retrieve specified subjects in surveillance video data. Most recently, soft biometrics have been used to improve the person re-identification accuracy across non-overlapping surveillance cameras [2].

Recently, the reference-based idea has been used in the field of computer vision, for example, face verification [21], face recognition [24], and person re-identification [3]. The reference-based framework is data-driven and different entities to be matched or compared are first described using the elements in the reference set and

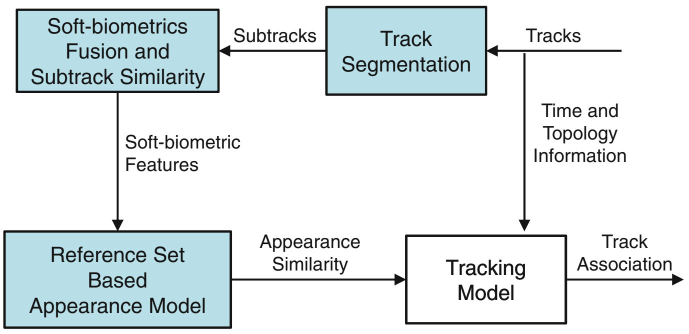

Soft-biometrics Fusion and Subtrack Similarity
Subtracks
Track Segmentation
Tracks
Time and Topology Information
Soft-biometric Features
Reference Set Based Appearance Model
Appearance Similarity
Tracking Model
Track Association

Fig. 10.2 An overview of the multi-camera tracking system

reference-based descriptors are generated. Therefore, direct comparison of objects with different modalities (e.g., faces at different poses) is avoided. Schroff et al. [21] achieved pose, illumination, and expression invariant face verification using a library of faces in various appearances to describe a given face based on the insight that it is most meaningful to compare faces with the same imaging conditions. Yin et al. [24] proposed an "Associate-Predict" model which is built on a generic identity data set that contains multiple images with large intra-person variation. Given a face, it is first associated to like identities in the data set and then its appearance under settings of another input face is predicted. In this way the intra-personal variation is handled. Recently, to improve person re-identification in different camera views, An et al. [3] used a reference set to generate reference-based descriptors for probe and gallery subjects, bypassing the need to direct compare the features from subjects with significant appearance change.

10.3 Technical Approach

An overview of the tracking system is presented in Fig. 10.2. Tracks obtained by single camera tracking are used as input for the multi-camera tracking system. Each track is further divided into several subtracks that are visually very similar. A fusion method is designed to combine soft-biometric features extracted from multiple detections in a subtrack and similarity between subtracks is computed based on the fused soft-biometrics. The appearance similarity between two tracks that are from different cameras is computed by the proposed reference set based appearance model. Finally, track associate is carried out based on appearance, time, and topology information.

10.3.1 Formulation of the Multi-camera Tracking Problem

Suppose we have m cameras $C_1, C_2, \ldots, C_m$ with non-overlapping FOVs, and we assume tracking multiple targets in the same camera has already been done. Given the tracking results in each single camera, we can generate a set $T = \{T_1, \ldots, T_N\}$

that contains all the within-camera tracks. A track T_i is a consecutive sequence of detections that contain the same target, its time interval is denoted as $[t^i_{begin}, t^i_{end}]$, and its corresponding camera is denoted as C^i. The problem of tracking across cameras is essentially to find out tracks that contain the same target, given certain spatial-temporal constraints. Let association a_{ij} define the hypothesis that track T_i and T_j contain the same target, with T_i occurring before T_j and $C^i \neq C^j$. A valid association matrix A is defined as follows:

$$A = \{a_{ij}\}, \quad a_{ij} = \begin{cases} 1 & \text{if } T_i \text{ is associated to } T_j \\ 0 & \text{otherwise} \end{cases} \tag{10.1}$$
$$\text{such that} \sum_i a_{ij} = 1 \text{ and } \sum_j a_{ij} = 1$$

The constraints for matrix A indicate that each track cannot be associated to more than one other track.

The cost S_{ij} for linking track T_i and T_j is based on time, appearance, and camera topology constraints, as defined in Eq. (10.2).

$$S_{ij} = Time(T_i, T_j) + Topo(T_i, T_j) + Appr(T_i, T_j) \tag{10.2}$$

where $Time(\cdot)$, $Topo(\cdot)$, and $Appr(\cdot)$ are the time, topology, and appearance models, respectively. The time model is defined as:

$$Time(T_i, T_j) = \begin{cases} 0 & \text{if } 0 < Gap_{ij} < GAP \\ \infty & \text{otherwise} \end{cases} \tag{10.3}$$

where Gap_{ij} is the time difference between T_i and T_j, and only when Gap_{ij} is within the pre-defined maximum allowed gap GAP the two tracks can be linked. The topology model is similar to the time model, which gives the restriction that T_i can be associated with T_j only when there is a path allowing people to walk between camera C^i and C^j without entering the view of any other cameras.

Let Σ be the set of all possible association matrices, the task of multi-target tracking across disjoint camera views is formulated as the following optimization problem:

$$A^* = \arg\min_{A \in \Sigma} \sum_{ij} a_{ij} S_{ij} \tag{10.4}$$

This assignment problem can be solved by Hungarian algorithm [15] in polynomial time. In order to reduce the computational cost, a pre-defined time sliding window is used, and the association is carried out independently in each time sliding window. Normally, there is a 50 % overlap for the neighboring two time sliding windows. Instead of using the cost matrix $S(N \times N)$ directly, we use the augmented matrix $S'(2N \times 2N)$ in [17] as the input for the Hungarian algorithm. The augmented matrix S' can be constructed by four matrixes of size $N \times N$, as shown in Eq. (10.5).

Table 10.1 Soft-biometrics extracted from detection response

Name	Value	Type
HairColor	Light, dark	Symbolic
SkinColor	Caucasian, non_caucasian	Symbolic
Height	Centimeters	Scalar
Weight	Kilograms	Scalar
BodyColor	1-D probability distribution	Vector
TorsoColor	1-D probability distribution	Vector
LegsColor	1-D probability distribution	Vector

$$S' = \left[\begin{array}{c|c} S & C \\ \hline B & B \end{array}\right] \tag{10.5}$$

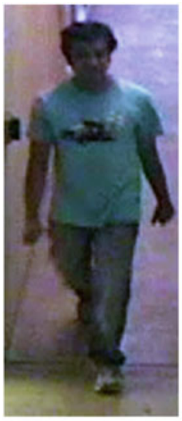
where S is the original cost matrix, C is a diagonal matrix (infinity off diagonal) with values c on the diagonal, and B is a matrix of infinity. The value c in matrix C indicates the threshold for association, a pair of tracks can only be associated when their cost is lower than the threshold. In the following sections, we explain in detail the proposed soft-biometrics integrated appearance representation and the reference set based appearance model.

10.3.2 Soft-Biometrics Fusion and Subtrack Similarity

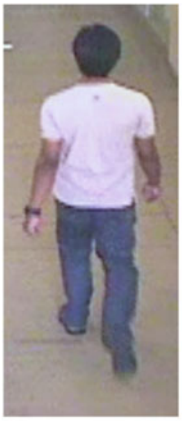
For each of the detection responses in a track, a set of soft-biometric features are extracted, as shown in Table 10.1, where the potential values for each feature are also listed. These soft-biometric features are extracted using state-of-the-art techniques [8, 18], and can be categorized into three types: symbolic, scalar-valued, and vector-valued. A confidence level which scales from 0 to 1 is associated with each feature to indicate the prediction confidence. In Figs. 10.3 and 10.4, examples of soft-biometrics extracted from multiple detection responses are shown.

In order to handle within camera illumination variation, each track is further segmented into small subtracks according to a pre-defined subtrack length (e.g., 5 frames) so that detections in each subtrack are visually very similar. After track segmentation, each subtrack is an appearance instance for a target under certain illumination condition. To generate concise representation for a given subtrack, we design a fusion method that can combine common soft-biometric features extracted from several detections into a single one. In the remainder of this chapter, fn represents the feature name, $fval$ is the feature value, and fc is the confidence level.

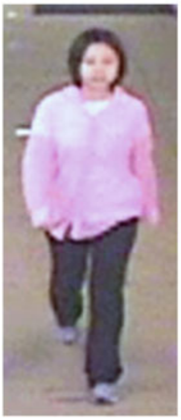
For binary symbolic features, the sum of confidence levels of all potential values is equal to 1. Thus, given the confidence level of one potential value, the confidence level for the other potential value can be inferred. When fusing symbolic features, the averaged confidence level for each potential value is computed and the one with the highest score is selected as the fused confidence level, and the corresponding value is the fused feature value. For scalar-valued and vector-valued features, the fused value



```
<Height>170</Height>
<HeightConfidence>0.73</HeightConfidence>
<Weight>85</Weight>
<WeightConfidence>0.6</WeightConfidence>
<HairColor>DARK_HAIR</HairColor>
<HairColorConfidence>0.72</HairColorConfidence>
<SkinColor>NON-CAUCASIAN</SkinColor>
<SkinColorConfidence>1</SkinColorConfidence>
<BodyColor>[0.0506 0.0013 0 0 0 0 0 0 0.0179 0.016 0.0004 0 0 0 0 0 0
0.001 0.001 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0132 0.0011 0 0 0 0 0 0 0.0511 0.1031
0.0036 0 0 0 0 0 0.0002 0.0042 0.0995 0.0468 0 0 0 0 0 0 0.0006 0.0093 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0.0006 0.0025 0.0006 0 0 0 0 0 0.002 0.0693 0.0808 0.0047
0.0003 0 0 0 0 0.0006 0.01 0.0138 0.0046 0.0001 0 0 0 0 0 0 0.0003
0.0006 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0.0025 0.0253 0.0025 0 0 0 0 0.0005 0.0023 0.031
0.0354 0.018 0.0004 0 0 0 0 0.0003 0.0003 0.0054 0.0045 0.0006 0 0 0 0 0
0 0 0.0002 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0.0002 0.0011 0.0101 0.0309 0.0045 0 0 0 0.0003
0.0015 0.0409 0.0348 0.0442 0.0018 0 0 0 0 0.0012 0.0003 0.0002 0.0015
0.0009 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0.0001 0 0 0 0 0 0 0.0009 0.0055 0.0076 0.0157
0.0005 0 0 0 0.0074 0.0258 0.0019 0.0013 0.0123 0 0 0 0.0001 0.0052
0.0001 0 0.0001 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0001 0 0 0 0
0.0003 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0]</BodyColor>
<BodyColorConfidence>1</BodyColorConfidence>
```

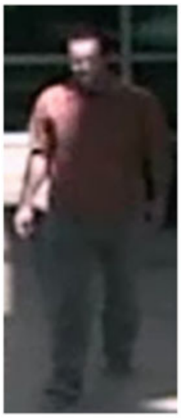


```
<Height>166</Height>
<HeightConfidence>0.67</HeightConfidence>
<Weight>75</Weight>
<WeightConfidence>0.60</WeightConfidence>
<HairColor>DARK_HAIR</HairColor>
<HairColorConfidence>0.83</HairColorConfidence>
<SkinColor>NON-CAUCASIAN</SkinColor>
<SkinColorConfidence>1</SkinColorConfidence>
<BodyColor>[0.0036 0.0004 0.0001 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0.0003 0.0012 0.0002 0 0 0 0 0 0 0.0018 0.0021 0.0004 0 0 0 0
0 0 0 0.0001 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0001 0.0015 0.0035 0.0001
0 0 0 0 0 0.0006 0.005 0.0031 0.0011 0 0 0 0 0 0 0.0003 0.0048 0.0017 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0001 0.0055 0.0026 0.0001 0 0 0 0 0 0.0014
0.0103 0.0286 0.0349 0.0003 0 0 0 0 0 0.0038 0.0346 0.0337 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.0002 0 0 0 0 0 0 0 0 0.0051 0.0093 0.0007 0 0 0 0 0 0.0051 0.2025
0.0583 0.0054 0 0 0 0 0 0.0028 0.0101 0.0007 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0001
0.0002 0 0 0 0 0 0 0.0001 0.083 0.0572 0.0007 0 0 0 0 0 0.0179 0.0813
0.0205 0.0004 0 0 0 0 0 0 0.0047 0.0021 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0009 0 0 0 0
0 0 0 0.0083 0.0208 0.0003 0 0 0 0 0 0.0076 0.0083 0.0163 0 0 0 0 0 0 0
0.007 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0.0032 0.0007 0 0 0 0 0 0 0.003 0.0053 0 0 0 0
0 0 0 0.1625]</BodyColor>
<BodyColorConfidence>1</BodyColorConfidence>
```



```
<Height>155</Height>
<HeightConfidence>0.70</HeightConfidence>
<Weight>65</Weight>
<WeightConfidence>0.63</WeightConfidence>
<HairColor>DARK_HAIR</HairColor>
<HairColorConfidence>0.72</HairColorConfidence>
<SkinColor>NON-CAUCASIAN</SkinColor>
<SkinColorConfidence>1</SkinColorConfidence>
<BodyColor>[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.0009 0.0014 0 0 0 0 0 0 0.0002 0.0117 0.0142 0 0 0 0 0 0 0 0.0011 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0.0005 0.0009 0.0001 0 0 0 0 0 0 0.0142 0.0324 0.0002 0 0
0 0 0 0.0006 0.0678 0.0136 0 0 0 0 0 0 0.0003 0.0032 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.0007 0.0006 0.0002 0 0 0 0 0 0 0.0049 0.0042 0.0002 0 0 0 0 0 0.0012
0.0146 0.0053 0 0 0 0 0 0 0.0001 0.0003 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0004 0.0008
0.0005 0 0 0 0 0 0 0.0176 0.0094 0.0001 0 0 0 0 0 0.0491 0.179 0.0047 0
0 0 0 0 0 0.001 0.0011 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0005 0.0059 0.0006 0 0 0 0 0
0.0071 0.1573 0.0176 0.0007 0 0 0 0 0.0002 0.0596 0.0336 0.0009 0 0 0 0
0 0 0.0002 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0.0018 0.0001 0 0 0 0 0 0.0008 0.0075 0.0098 0.0002 0 0
0 0 0 0.0071 0.0127 0.0048 0 0 0 0 0 0 0.0017 0.0027 0 0 0 0 0 0 0
0.0001 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0.0028 0.0006 0 0 0 0 0 0.0008 0.0098 0.0634 0 0 0 0 0 0.0002
0.0043 0.1009 0 0 0 0 0 0 0 0.024]</BodyColor>
<BodyColorConfidence>1</BodyColorConfidence>
```

Fig. 10.3 Examples of soft-biometric features extracted from detection responses that have high consistency with the ground-truth. For each detection response, an xml file is generated to store all the feature values



```
<Height>190</Height>
<HeightConfidence>0.87</HeightConfidence>
<Weight>80</Weight>
<WeightConfidence>0.74</WeightConfidence>
<HairColor>DARK_HAIR</HairColor>
<HairColorConfidence>0.83</HairColorConfidence>
<SkinColor>NON-CAUCASIAN</SkinColor>
<SkinColorConfidence>1</SkinColorConfidence>
<BodyColor>[0.1654 0.0003 0 0 0 0 0 0 0.0085 0.0183 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0.0228 0.0008 0 0 0 0 0 0 0.0229 0.4133 0.004 0 0 0 0 0
0 0.0082 0.0255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0007 0 0 0 0 0 0 0 0.0006 0.0187
0.0017 0 0 0 0 0 0 0.0024 0.1941 0.0002 0 0 0 0 0 0 0 0.0003 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0.0014 0.0006 0 0 0 0 0 0 0.0001 0.0071 0.0003 0 0 0 0 0 0 0.0008
0.0052 0.0001 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0021 0.0007 0 0 0 0 0
0 0.0007 0.0064 0.0007 0 0 0 0 0 0 0.0002 0.0027 0.0001 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0.0034 0.0009 0 0 0 0 0 0 0.0008 0.0085 0.0009 0 0 0 0
0 0 0.0002 0.0024 0.0002 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0021 0.0007
0 0 0 0 0 0 0.0003 0.0071 0.0013 0 0 0 0 0 0 0.0002 0.0021 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0.0005 0 0 0 0 0 0 0 0.0025 0.0013 0 0 0 0 0 0 0 0.0085 0.0018 0 0
0 0 0 0 0.0009 0.0153]</BodyColor>
<BodyColorConfidence>1</BodyColorConfidence>
```

```
<Height>180</Height>
<HeightConfidence>0.79</HeightConfidence>
<Weight>50</Weight>
<WeightConfidence>0.68</WeightConfidence>
<HairColor>DARK_HAIR</HairColor>
<HairColorConfidence>0.92</HairColorConfidence>
<SkinColor>NON-CAUCASIAN</SkinColor>
<SkinColorConfidence>1</SkinColorConfidence>
<BodyColor>[0.1654 0.0003 0 0 0 0 0 0 0.0085 0.0183 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0.0228 0.0008 0 0 0 0 0 0 0.0229 0.4133 0.004 0 0 0 0 0
0 0.0082 0.0255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0007 0 0 0 0 0 0 0 0.0006 0.0187
0.0017 0 0 0 0 0 0 0.0024 0.1941 0.0002 0 0 0 0 0 0 0 0.0003 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0.0014 0.0006 0 0 0 0 0 0 0.0001 0.0071 0.0003 0 0 0 0 0 0 0.0008
0.0052 0.0001 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0021 0.0007 0 0 0 0 0
0 0.0007 0.0064 0.0007 0 0 0 0 0 0 0.0002 0.0027 0.0001 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0.0034 0.0009 0 0 0 0 0 0 0.0008 0.0085 0.0009 0 0 0 0
0 0 0.0002 0.0024 0.0002 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0021 0.0007
0 0 0 0 0 0 0.0003 0.0071 0.0013 0 0 0 0 0 0 0.0002 0.0021 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0.0005 0 0 0 0 0 0 0 0.0025 0.0013 0 0 0 0 0 0 0 0.0085 0.0018 0 0
0 0 0 0 0.0009 0.0153]</BodyColor>
<BodyColorConfidence>1</BodyColorConfidence>
```

```
<HeightClassification/>
<Height>152</Height>
<HeightConfidence>0.71</HeightConfidence>
<Weight>86</Weight>
<WeightConfidence>0.68</WeightConfidence>
<HairColor>DARK_HAIR</HairColor>
<HairColorConfidence>0.44</HairColorConfidence>
<SkinColor>NON-CAUCASIAN</SkinColor>
<SkinColorConfidence>1</SkinColorConfidence>
<BodyColor>[0.0011 0.0002 0 0 0 0 0 0 0.0012 0.0158 0.0007 0 0 0 0 0
0.0006 0.0001 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0002 0.0557 0.0389 0
0 0 0 0 0.001 0.018 0.1009 0.0035 0 0 0 0 0 0.0002 0.0002 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0.0031 0.0127 0 0 0 0 0 0 0.0006 0.1177 0.0627 0.0003 0 0 0 0.0001
0.0007 0.007 0.126 0.0047 0 0 0 0 0 0.0001 0.0001 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0.0029 0.0069 0 0 0 0 0 0 0.0002 0.0795 0.0649 0.0002 0 0 0 0 0.0001
0.0007 0.0078 0.0035 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.0136 0.0155 0 0
0 0 0 0 0.0001 0.0379 0.0236 0.0006 0 0 0 0 0.0002 0.0006 0.0262 0.0095
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.008 0.0037 0.0001 0 0 0 0 0 0.0008
0.0065 0.0248 0.0016 0 0 0 0 0.0006 0.0001 0.0094 0.0266 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0.0004 0.0002 0 0 0 0 0 0.0001 0.0002 0.0009 0.0163 0
0 0 0 0.0003 0.0014 0.0005 0.0154 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0.0018 0.0022 0.0077]</BodyColor>
<BodyColorConfidence>1</BodyColorConfidence>
```

Fig. 10.4 Examples of soft-biometric features extracted from detection responses that have low consistency with the ground-truth. For each detection response, an xml file is generated to store all the feature values

is the weighted sum of all $fval$s, where the weights are the corresponding fc, and normalization is carried out to make the result lie in the range of [0, 1], as shown in Eq. (10.6). The fused confidence level is the average of all fcs, as shown in Eq. (10.7).

$$fval\_{fused} = \frac{1}{\sum fc_i} \sum_{i=1}^{n} fval_i \times fc_i \tag{10.6}$$

$$fc\_{fused} = \frac{1}{n} \sum_{i=1}^{n} fc_i \tag{10.7}$$

where n is the number of features used for fusion.

After soft-biometrics fusion, each subtrack is represented by a single set of soft-biometric features. The similarity of two subtrack is computed based on the similarity between common features for each feature type (symbolic, scalar, vector). For the symbolic features (HairColor, SkinColor), if the symbolic value of the two features is the same then the similarity is the average of the two confidence levels. If the symbolic values are dissimilar, then the similarity is the maximum confidence level, as defined in Eq. (10.8).

$$sim_1(fval_1, fval_2) = \max(fc_1 \times (1 - fc_2), (1 - fc_1) \times fc_2) \tag{10.8}$$

For the scalar-valued features (Height and Weight), we assume that the feature values are from a normal distribution with parameters μ and σ^2. As the height accuracy is ± 12.7 cm and the weight accuracy is ± 9 kg (learned by analyzing soft-biometrics extracted from previous data), we define the standard deviation so that for the height $P([fval - 12.7, fval + 12.7]) = 80\,\%$ and for the weight $P([fval - 9, fval + 9]) = 80\,\%$. For the accumulated probability to be equal to 80 % the range should be $(\mu - 1.28\sigma, \mu + 1.28\sigma)$. Thus, the standard deviations are equal to $1.28\sigma = 12.7$ for height and $1.28\sigma = 9$ for weight, i.e. $\sigma_{height} = 9.92$ and $\sigma_{weight} = 7.03$. The similarity score sim_2 is defined as:

$$sim_2(fval_1, fval_2) = 1 - \sqrt{1 - e^{\frac{-(fval_1 - fval_2)^2}{8\sigma^2}}} \tag{10.9}$$

For the vector-valued features (BodyColor, TorsoColor, LegsColor), the Bhattacharyya Coefficient [7] is used to measure the similarity sim_3, which approximates the amount of overlap between two probability distributions, as given in Eq. (10.10):

$$sim_3(fval_1, fval_2) = \sum_{i=1}^{n} \sqrt{fval_{1i} \times fval_{2i}} \tag{10.10}$$

10.3.3 Reference Set Based Appearance Model

The basic idea of reference set based appearance model is illustrated in Fig. 10.5. In Fig. 10.5, when comparing track T_1 in C_i with tracks T_2 and T_3 in C_j, by using their color histograms directly, T_3 are more likely to be matched with T_1. Even though they contain totally different targets, the significant illumination change in C_i makes T_1 looks much more darker than its actual appearance. To handle such problem, a reference set $RefSet_{ij}$ is constructed for a pair of cameras C_i and C_j. It contains a set of reference targets $R = \{R_1, R_2, \ldots, R_m\}$ that appear in both C_i and C_j. The tracks for all the reference targets appear in C_i form $RefSet_{ij}^i$, and the tracks for all the reference targets appear in C_j form $RefSet_{ij}^j$, as shown in Fig. 10.5. Given two tracks T_p and T_q with T_p captured in the view of camera C_i and T_q captured in the view of camera C_j, the appearance similarity between these two tracks are not computed by comparing T_p and T_q directly. Instead, T_p is compared with all the tracks in $RefSet_{ij}^i$ and T_q is compared with all the tracks in $RefSet_{ij}^j$, and their similarity with the reference set are used to calculate the similarity of T_p and T_q. In other words, track T_p and T_q are compared with other tracks that undergo the same illumination conditions as T_p and T_q, and if they are the tracks of the same target, they should have high similarities with the same set of reference targets. Otherwise, they are more likely to be tracks that contain different targets.

A track is segmented into multiple subtracks, and each subtrack is regarded as an appearance instance for the target contained in current track. By this means, we generate multiple representations for each target that covers all the appearance changes of that target in a certain camera.

When comparing the similarity of two tracks T_a and T_b in the same camera, every subtrack in T_a is compared with every subtrack in T_b. Let t_a^k denotes the k-th subtrack in track T_a, $simi(t_x, t_y)$ be the similarity of two subtracks (described in the following part), and N_a and N_b be the number of subtracks in T_a and T_b respectively. The similarity score for T_a and T_b is defined as follows:

$$Simi(T_a, T_b) = \frac{1}{N_a} \sum_{i=1}^{N_a} max(\{simi(t_a^i, t_b^j), \quad j \in [1, N_b]\}) \tag{10.11}$$

Namely, each t_a^i is compared with all subtracks in T_b, and the maximum score is used as the similarity between t_a^i and T_b. Similarity between T_a and T_b is the average of all these maximum scores.

In the reference set, each reference target may have several tracks in the same camera (e.g., walking towards and away from the camera). The similarity between a track T_l and a reference target R_n is the maximum of the similarities of T_l and all the tracks for R_n. This lays the strength of our reference set based appearance model—the tracks from different cameras that contain the same target under various pose and illumination conditions have a chance to get high similarity scores with similar reference targets. In other words, each reference target is an indirect feature

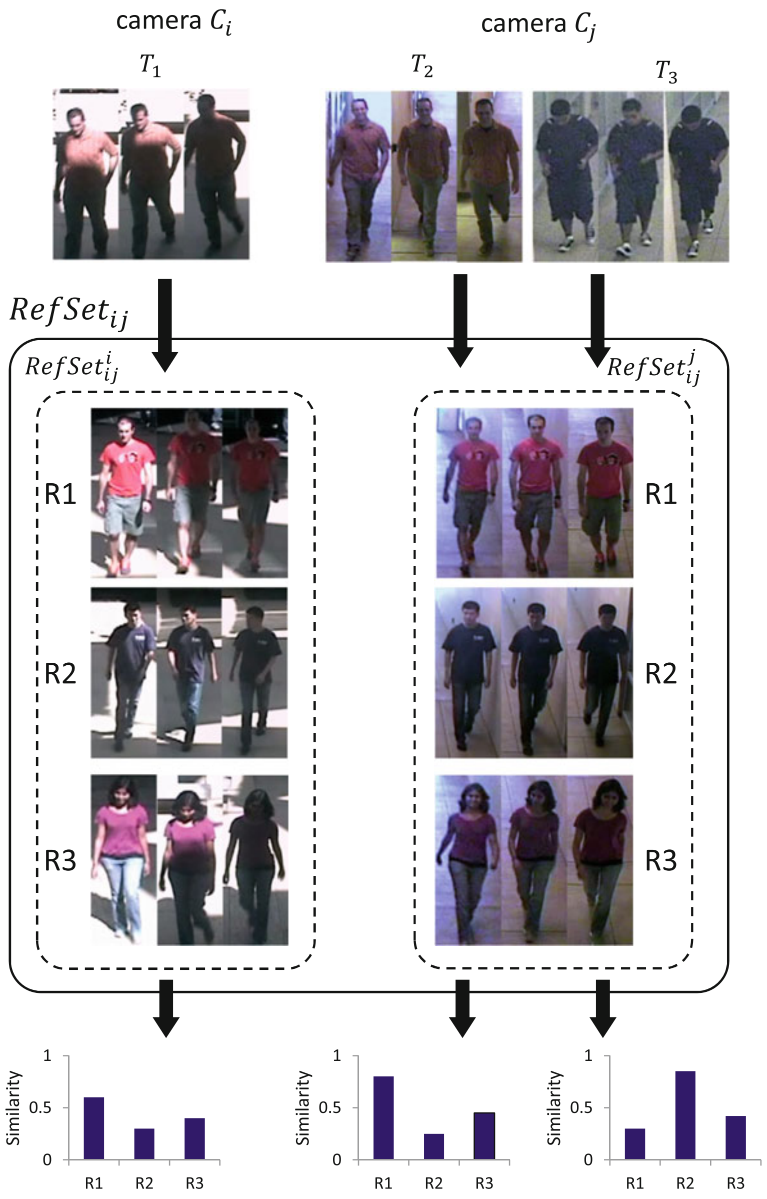

camera C_i
camera C_j
T_1
T_2
T_3
$RefSet_{ij}$
$RefSet_{ij}^i$
$RefSet_{ij}^j$
R1
R2
R3
Similarity
1
0.5
0

Fig. 10.5

◀ **Fig. 10.5** Illustration of the reference set based appearance model. For a pair of cameras C_i and C_j, a reference set $RefSet_{ij}$ (the *middle part*) is constructed that contains three reference targets ($R1$, $R2$, and $R3$) appearing in both cameras. Tracks for all the reference targets appear in C_i form $RefSet_{ij}^i$, and tracks for all the reference targets appear in C_j form $RefSet_{ij}^j$. To find matched tracks between track in C_i (T_1) and tracks in C_j (T_2 and T_3), each input track is described by all the reference targets. The description is a vector of ordered similarities (the *bottom part*), and each description is generated by comparing the input track with the corresponding reference set, i.e., T_1 with $RefSet_{ij}^i$, T_2 with $RefSet_{ij}^j$, and T_3 with $RefSet_{ij}^j$. After representing T_1, T_2 and T_3 by the reference set, it is clear that T_1 and T_2 have high similarities with the same set of reference targets. Note that, both the input tracks and the reference targets have multiple appearance instances (only three instances are presented for demonstration) that cover all the appearance changes

that describes some characteristics of the target's appearance, and having the tracks in two different cameras compared to the same set of reference targets enables us to compare the similarity of these two tracks. In addition, the reference set based appearance model does not require any extra training process. Besides variation in illumination conditions, difference in poses are also taken care of by the various appearance instances in each reference target.

After comparing tracks T_p and T_q with each reference target in its corresponding reference set, we get two vectors of ordered similarities, as shown in Fig. 10.5. Let $Ref_{ij}^i(T_p)$ and $Ref_{ij}^j(T_q)$ be the representations of T_p and T_q by the reference set Ref_{ij}, the similarity of T_p and T_q is computed by the Kendall tau Correlation Coefficient [1], and is further normalized to the range of [0, 1]. In order to get the appearance model, we use the negative logarithm function to calculate the cost, as defined in Eq. (10.12):

$$Appr(T_p, T_q) = -log(\tau'(Ref_{ij}^i(T_p), Ref_{ij}^j(T_q))) \tag{10.12}$$

where $\tau'(\cdot)$ is the normalized Kendall tau Correlation Coefficient.

10.4 Experimental Results

In order to evaluate the proposed model, five cameras (four indoor and one outdoor) are used to establish the desired non-overlapping setting, the topology is presented in Fig. 10.6 and sample frames from each camera is shown in Fig. 10.1. All the videos are taken during the same time period and each video is about 20 min in duration. The resolution is 704 × 480, the frame rate is 20 fps. The number of participants involved in each video ranges from 7 to 10.

This setting is very challenging for multi-target tracking in non-overlapping camera views due to following reasons:

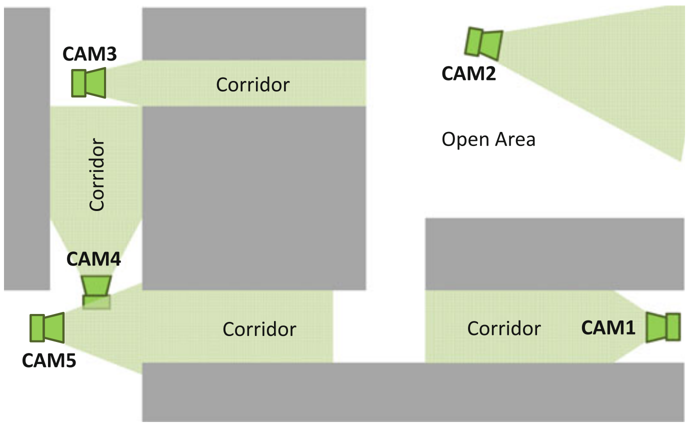


Fig. 10.6 Topology for cameras used in the experiments

1. The outdoor camera view contains intense illumination changes, and there exists lighting variations for indoor camera views as well. This makes it unreliable to use a single transformation to map colors in a pair of cameras, such as BTFs.
2. The number of camera involved is greater than most of the previous work that normally use 2–3 cameras [14, 16].

In order to construct the reference set another set of data is used. It is collected under the same setting but with participants either not included in the testing data or they are included in the testing data but with very different clothes. The number of participants involved in each reference set ranges from 9 to 11.

10.4.1 Soft-Biometrics Verification

As we use soft-biometrics to represent each target, the quality of soft-biometrics and the soft-biometrics similarity measurement are crucial for tracking. In the verification, for each video (captured in a single camera) we compute the similarity between any pair of tracks based on the proposed method, and these similarities are categorized into intra-class (tracks from the same target) and inter-class (tracks from different targets). The histograms for each category are plotted. Four sample plots are shown in Fig. 10.7. The plots suggest that most intra-class similarities are larger than most inter-class similarities and with a single threshold these two classes can be coarsely separated. Therefore, the soft-biometrics extracted from the same target have high degree of consistency.

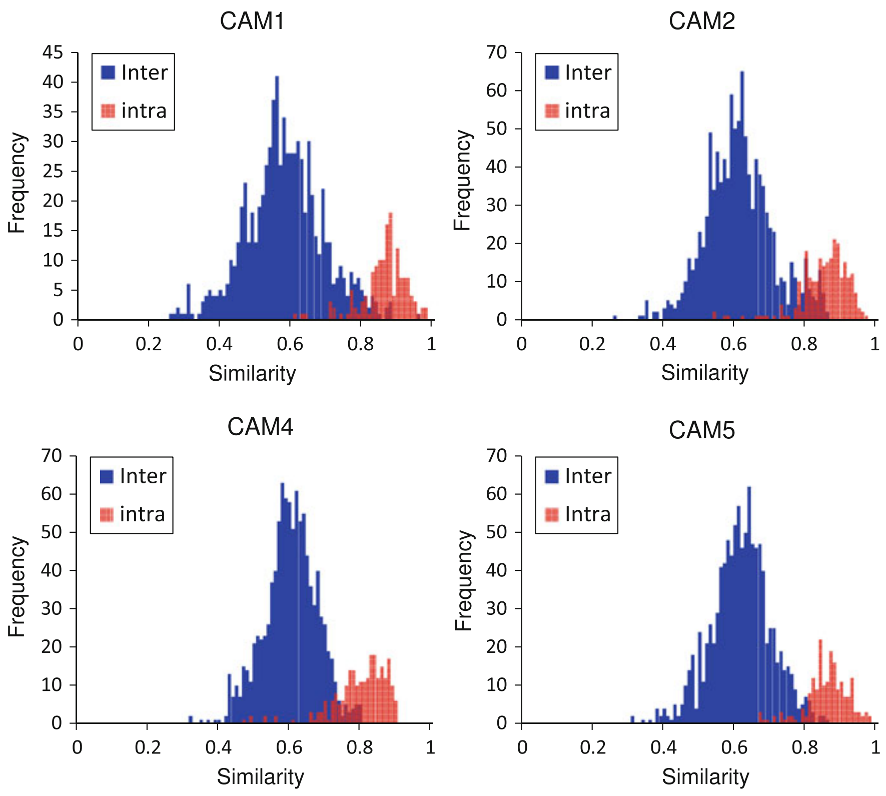


Fig. 10.7 Histograms of inter-class and intra-class similarities from two testing videos. Best viewed in color. "Intra" stands for tracks that contain the same target and "Inter" stands for tracks that contain different targets

10.4.2 Tracking Results

In this evaluation, our main focus is to associate tracks that contain the same target in different camera views given certain spatial-temporal constraints. We applied our reference set based appearance model with soft-biometric features (RefSet2) on the testing data. Three baseline models are presented for comparison: (1) Use the Bhattacharyya distance of holistic color histograms directly to measure the appearance similarity (Color). (2) Generate the appearance model based on the BTF model in [14] (BTF). (3) Our proposed reference set based appearance model with only holistic color histograms as features (RefSet1).

In all our experiments, the length of subtrack is set to 10 frames. For each model, various thresholds (ranges from 0.2 to 0.6) are tested for the augmented cost matrix, and the best result is chosen. We hand labeled the ground-truth which consists of 220 track associations (there are 368 single camera tracks in total). Two metrics are used for evaluation, as defined in Eq. (10.13). The comparison is presented in Fig. 10.8.

$$ErrorRate = \frac{Error}{N_{result}}, \quad MatchRate = \frac{Match}{N_{GT}} \tag{10.13}$$

where $Error$ and $Match$ are the number of incorrectly and correctly associated track pairs in the result, N_{result} and N_{GT} are the number of track associations in the result and the ground-truth respectively.

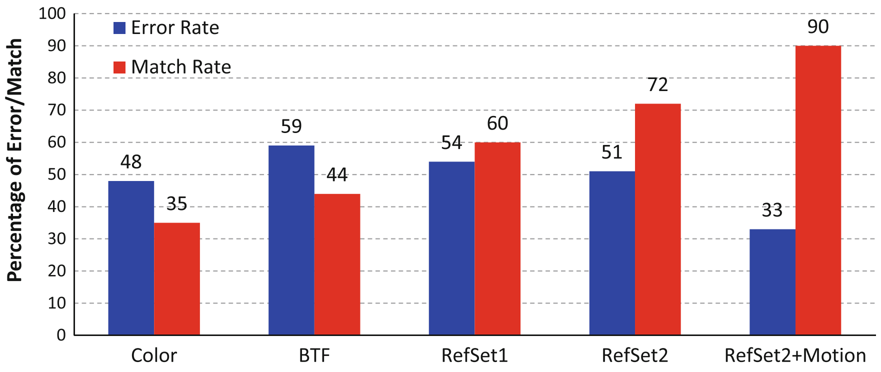


Fig. 10.8 Comparison of the proposed methods and other baseline models

It can be observed that when using the reference set based appearance model with the soft-biometric features, we achieve the highest match rate and the lowest error rate compared with all the baseline models. Compared with BTF, the RefSet2 model increases the match rate by almost 30 % and reduces the error rate by about 10 %. Even with color histograms only, the reference set based appearance model (RefSet1) provides better performance than BTF in terms of both the error rate and the match rate. The comparison between RefSet1 and RefSet2 demonstrates that the other soft-biometric features are complementary to color histograms and reduce ambiguities, as they capture the appearnce information that is overlooked by color histograms. It is worth noting that although the error rate is high even for RefSet1 and RefSet2 (more than 50 %), these results are obtained by using the appearance information only.

As another kind of clue, motion information plays an important role in multi-target tracking and can greatly reduce the number of false positive. For example, in a time sliding window, a track in CAM4 can be associated with tracks in both CAM3 and CAM5 based on the camera topology. Given the knowledge that the target is walking away from the camera, we can easily eliminate tracks in CAM5 from possible associations. When a motion model that measures the walking direction of the target is integrated into the tracking system (RefSet2+Motion), the error rate is greatly reduced to about 30 %. Also, with motion information our proposed method can correctly associate 90 % track pairs in the ground-truth, which further demonstrates the effectiveness of our method. Visual results obtained by using BTF and RefSet2 on some challenging cases are presented in Figs. 10.9 and 10.10. The comparison between visual results of BTF and RefSet2 further validates the robustness of our method.

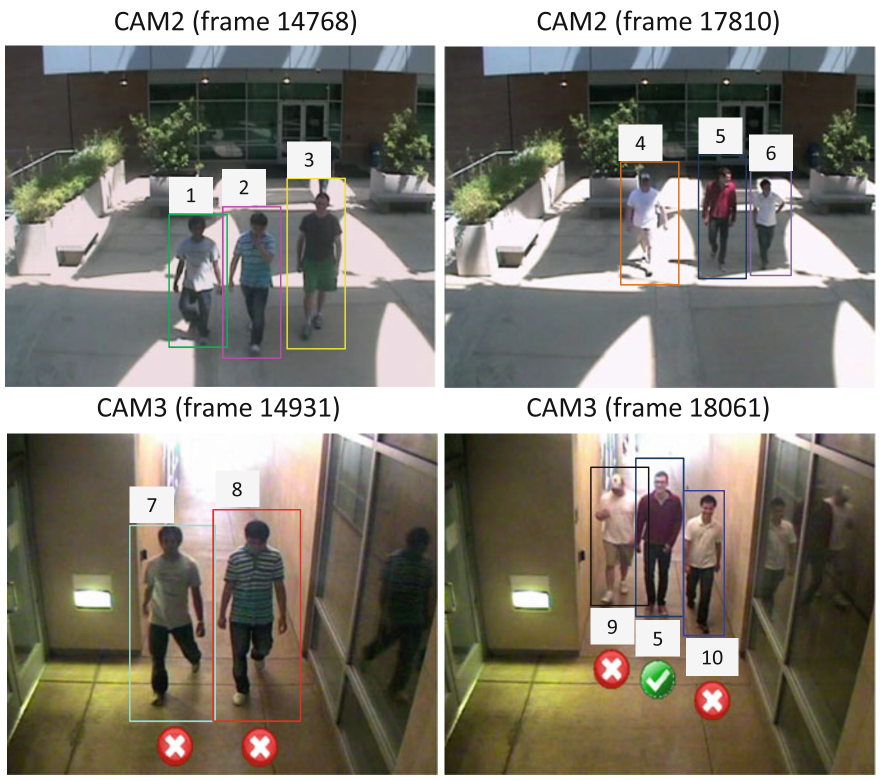


Fig. 10.9 Example tracking results obtained by using BTF in [14]. Best viewed in color. The *first row* shows that targets first appear in the FOV of CAM2, and the *second row* shows the tracking results after the targets enter the FOV of CAM3. The method in [14] fails to associate tracks that contain the same target under challenging conditions

10.4.3 Implementation

We implemented our multi-camera tracking system in Matlab without code optimization or parallelization and tested it on a PC with 3.0 GHz CPU and 8 GB memory. The computational time is greatly affected by the number of targets in a video and the length of the video. In our experiments, the average computational time for track association is 372 s. Note that computational time for track generation and soft-biometric features extraction are not included.

10.5 Conclusions

In this work, we integrate illumination and pose invariant soft-biometric features into the appearance representation of a tracked target, and design a fusion method to combine the same type of soft-biometric features extracted from multiple detections into

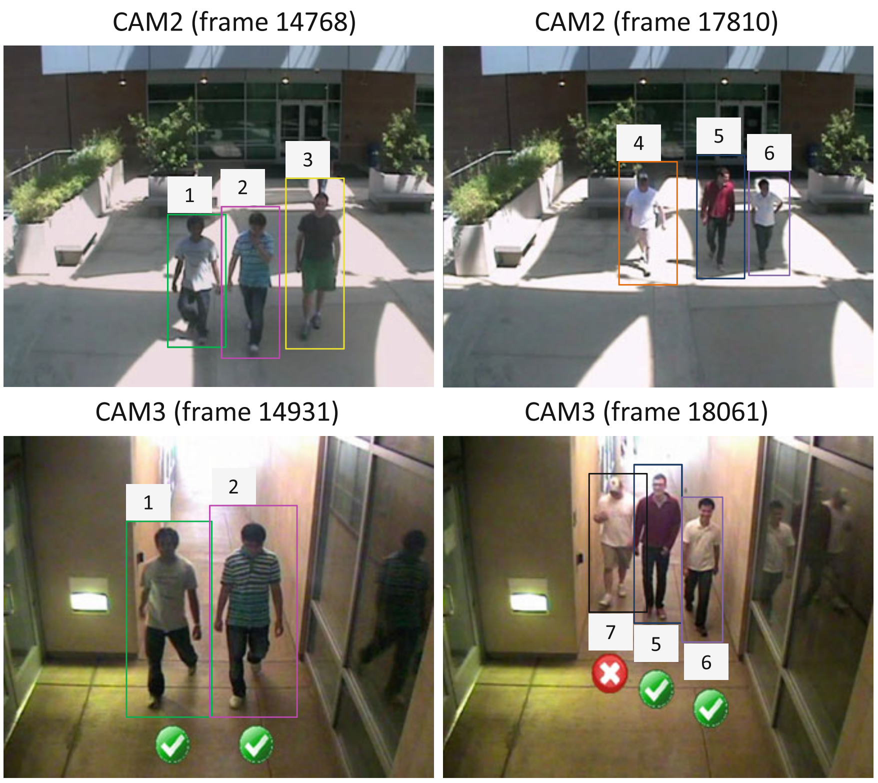


Fig. 10.10 Example tracking results obtained by using our proposed method RefSet2. The *first row* shows that targets first appear in the FOV of CAM2, and the *second row* shows the tracking results after the targets enter the FOV of CAM3. Best viewed in color. With soft-biometric features and the reference set, our method is able to match most of the targets where there exist drastic within camera and across camera illumination variations

a single one. A novel reference set based appearance model for multi-target tracking in a camera network with non-overlapping FOVs is proposed which addresses the problems caused by both within camera and across camera illumination variation. The proposed appearance model is easy to implement with no parameters and requires no additional training process, yet provides promising results. The experimental results demonstrate the superiority of the combination of reference set based appearance model and soft-biometric features over other baseline models on a challenging real-world video dataset.

Acknowledgments This work was supported in part by NSF grant 0905671 and ONR grant N00014-12-1-1026. The authors would like to acknowledge the soft-biometrics data provided by Progeny that have been used in this work. The contents and information do not reflect the position or policy of the U.S. Government.

References

1. Abdi H (2007) Kendall rank correlation. SAGE Publications, Inc., California, pp 509–511
2. An L, Chen X, Kafai M, Yang S, Bhanu B (2013) Improving person re-identification by soft biometrics based reranking. In: ACM/IEEE international conference on distributed smart cameras (ICDSC), pp 1–6
3. An L, Kafai M, Yang S, Bhanu B (2013) Reference-based person re-identification. In: IEEE international conference on advanced video and signal-based surveillance (AVSS)
4. Candamo J, Shreve M, Goldgof D, Sapper D, Kasturi R (2010) Understanding transit scenes: a survey on human behavior-recognition algorithms. IEEE Trans Intell Transp Syst 11(1): 206–224
5. Chu CT, Hwang JN, Lan KM, Wang SZ (2011) Tracking across multiple cameras with overlapping views based on brightness and tangent transfer functions. In: Fifth ACM/IEEE international conference on distributed smart cameras (ICDSC), pp 1–6
6. Chu CT, Hwang JN, Yu JY, Lee KZ (2012) Tracking across nonoverlapping cameras based on the unsupervised learning of camera link models. In: Sixth International conference on distributed smart cameras (ICDSC), pp 1–6
7. Cover TM, Thomas JA (1991) Elements of information theory. Wiley-Interscience, New York
8. Dantcheva A, Velardo C, D'angelo A, Dugelay JL (2011) Bag of soft biometrics for person identification. Multimedia Tools Appl 51(2):739–777
9. D'Orazio T, Mazzeo P, Spagnolo P (2009) Color brightness transfer function evaluation for non overlapping multi camera tracking. In: Third ACM/IEEE international conference on distributed smart cameras (ICDSC), pp 1–6
10. Gilbert A, Bowden R (2006) Tracking objects across cameras by incrementally learning inter-camera colour calibration and patterns of activity. In: European conference on computer vision 3952:125–136
11. Guo G, Mu G, Ricanek K (2010) Cross-age face recognition on a very large database: the performance versus age intervals and improvement using soft biometric traits. In: 20th International conference on pattern recognition (ICPR), pp 3392–3395
12. Hu W, Hu M, Zhou X, Tan T, Lou J, Maybank S (2006) Principal axis-based correspondence between multiple cameras for people tracking. IEEE Trans Pattern Anal Mach Intell 28(4): 663–671
13. Jain AK, Park U (2009) Facial marks: soft biometric for face recognition. In: ICIP
14. Javed O, Shafique K, Rasheed Z, Shah M (2008) Modeling inter-camera space-time and appearance relationships for tracking across non-overlapping views. Comput Vision Image Underst 109:146–162
15. Munkres J (1957) Algorithms for the assignment and transportation problems. J Soc Ind Appl Math 5(1)
16. Prosser B, Gong S, Xiang T (2008) Multi-camera matching using bi-directional cumulative brightness transfer functions. In: Proceedings of the British machine vision conference
17. Qin Z, Shelton CR (2012) Improving multi-target tracking via social grouping. In: IEEE conference on computer vision and pattern recognition
18. Reid D, Samangooei S, Chen C, Nixon M, Ross A (2013) Soft biometrics for surveillance: an overview. In: Machine learning: theory and applications. Elsevier, pp 327–352
19. Reid DA, Nixon M (2011) Using comparative human descriptions for soft biometrics. In: International joint conference on biometrics (IJCB), pp 1–6
20. Saleemi I, Shafique K, Shah M (2009) Probabilistic modeling of scene dynamics for applications in visual surveillance. IEEE Trans Pattern Anal Mach Intell 31(8):1472–1485
21. Schroff F, Treibitz T, Kriegman D, Belongie S (2011) Pose, illumination and expression invariant pairwise face-similarity measure via doppelgänger list comparison. In: IEEE international conference on computer vision (ICCV), pp 2494–2501
22. Siebel NT, Maybank SJ (2004) The advisor visual surveillance system. In: ECCV 2004 workshop applications of computer vision (ACV)

23. Wang X (2013) Intelligent multi-camera video surveillance: a review. Pattern recognition letters 34(1):3–19
24. Yin Q, Tang X, Sun J (2011) An associate-predict model for face recognition. In: 2011 IEEE conference on computer vision and pattern recognition (CVPR), pp 497–504
25. Zhu Y, Nayak N, Roy-Chowdhury A (2013) Context-aware activity recognition and anomaly detection in video. IEEE J Sel Top Sign Proces 7(1):91–101

Chapter 11
A Parallel Approach for Statistical Texture Parameter Calculation

Narcisse Talla Tankam, Albert Dipanda, Christophe Bobda, Janvier Fotsing and Emmanuel Tonyé

Abstract This chapter focusses on the development of a new image processing technique for the processing of large and complex images, especially SAR images. We propose here a new and effective approach that outperforms the existing methods for the calculation of high order textural parameters. With a single processor, this approach is about 256^{n-1} times faster than the co-occurrence matrix approach considered as classical, where n is the order of the textural parameter for a 256-gray scales image. In a parallel environment made of N processor, this performance can almost be multiply by the factor N. Our approach is based on a new modeling of textural parameters of a generic order $n > 1$ equivalent to the classical formulation, but which is no longer based on the co-occurrence matrix of order $n > 1$. By avoiding the calculation of the co-occurrence matrix of order $n > 1$, the resulted model enables a gain of about 256^n bytes of the required memory space.

N. Talla Tankam (✉)
Automatic and Computer Engineering Laboratory (LAIA), Fotso Victor University Institute of Technology Bandjoun, University of Dschang, Dschang, Cameroon
e-mail: narcisse.talla@gmail.com

N. Talla Tankam · J. Fotsing · E. Tonyé
Electronique and Signal Processing Laboratory (LETS), National High School of Engineering, University of Yaounde 1, Yaounde, Cameroon

J. Fotsing
e-mail: jfotsing@gmail.com

E. Tonyé
e-mail: tonyee@hotmail.com

A. Dipanda
Electronique, Computer and Image Engineering Laboratory (Le2i), University of Burgundy, Burgundy, France
e-mail: albert.dipanda@u-bourgogne.fr

C. Bobda
University of Arkansas, Arkansas, USA
e-mail: cbobda@uark.edu

11.1 Introduction

Automated classification is very useful for skills recognition. Unsupervised image classification methods try to subdivide a set of observations into statistical classes. A class being defined as a set of observations with density greater than the one of the surrounding environment [1].

Texture analysis is a robust approach for SAR image processing. It consists of a set of mathematical techniques enabling the quantification of various grey scales present in an image in terms of intensity or roughness and their distribution. The usefulness of textural analysis for SAR images classification is not to be demonstrated nowadays [2]. In fact, in [3], the authors used the grey level dependent matrix also called co-occurrence matrix to develop an approach of image segmentation based on frontier preservation in image texture. In [4], authors combined SAR and optical satellite images to develop a robust strategy of cultures identification. Huber [5] developed an approach of image classification based on the merger of information derived from measures observed on several SAR images acquired on the same site at different dates. In [6], authors used textural analysis to recognize relief shapes in SAR images. They transformed SAR image into elementary topographical symbols to extract ridge and valley lines. The major problem faced by all these works remains the computing time optimisation.

Many research works have been conducted in this field of research. These works can be classified into two main methods, notably structural and statistical methods [7]. Structural methods describe textures by defining primitives contained in the image and relationships between them. Statistical methods study the relationship between each pixel and its surrounding pixels. These methods combine statistical parameters, based on co-occurrence matrices defined at various orders. The calculation of these matrices is very high time consuming, especially for high order parameters. This is why, most of the time, researchers limit their works only on order 2 statistical parameters despite the fact that, without being strictly better than the lower order, high order parameters provide complementary information for texture analysis [8]. In this field of research, the preoccupation of researchers has always been the optimisation of parameters time calculation.

For this purpose, [9] replaced the co-occurrence matrix by the sum and the difference of histograms defining the principal axes of second order probabilities of stationary processes. Marceau et al. [10] proposed a textural and spectral approaches for image classification based on the reduction of grey scales, moving from 256 to 32. Obviously, one limit of this method was the loss of information. Kourgly and Belhadj-Aissa [11] developed a new algorithm for the calculation of textural parameters based on histograms. This method required the allocation of an array instead of the co-occurrence matrix and proposed a new formulation of textural parameters, based on one variable instead of two initially. Later run, based on this work, Akono et al. [2] proposed a new approach for the calculation of order 3 textural parameters. They defined specific image masks that helped to reformulate order 3 textural parameter as a function of a single variable instead of three variables initially.

In this chapter, we propose a parallel approach for the calculation of textural parameters that requires a constant time, independently to the order of the parameter. This approach consists in producing a new formulation of textural parameters that avoids the calculation of high order co-occurrence matrix [12] and enables a real-time image processing. In the following, we successively present: the classical approach based on co-occurrence matrices; the new approach; the time complexities calculation; the experimentation and the conclusion.

11.2 Methodology

In this section, through a series of developments, we start from the classical expression of textural parameters and obtain the new formulation.

11.2.1 Classical Expression of Textural Parameters

An order-2 textural parameter $Para_2$ applied on a digital image with maximum grey scale $MaxGs$ is a real function of a couple of integers (i, j) defined as follows:

$$Para_2 = \sum_{i=0}^{\mathrm{Max}Gs} \sum_{j=0}^{\mathrm{Max}Gs} \varphi(i, j, C_{ij}) \tag{11.1}$$

C_{ij} being the entry (i, j) of the co-occurrence matrix.

For example, the Dissimilarity parameter is expressed as follow:

$$Diss_2 = \frac{1}{N} \sum_{i=0}^{\mathrm{Max}Gs} \sum_{j=0}^{\mathrm{Max}Gs} |i - j|\, C_{ij} \tag{11.2}$$

where N is the total number of possible couples in the image. This number depends on the selected direction and the inter-pixel distance.

Let denote by P_{ij}, the probability of occurrence of the couple of grey scales (i, j) in the image. Then the following equation is obtained:

$$P_{ij} = \frac{C_{ij}}{N} \tag{11.3}$$

The generalisation of Eq. 11.1 to a generic order $n > 1$ gives the following equation:

$$Para_n = \sum_{i_0=0}^{\mathrm{Max}Gs} \sum_{i_1=0}^{\mathrm{Max}Gs} \dots \sum_{i_{n-1}=0}^{\mathrm{Max}Gs} \varphi(i_0, i_1, \dots, i_{n-1}, P_{i_0 i_1 \dots i_{n-1}}) \tag{11.4}$$

$P_{i_0 i_1 \ldots i_{n-1}}$ expresses the probability of occurrence of the n-uplet $(i_0, i_1, \ldots, i_{n-1})$ of grey scales in the image, following the $(n-1)$-uplet $(\theta_0, \theta_1, \ldots, \theta_{n-2})$ of directions and the $(n-1)$-uplet $(d_0, d_1, \ldots, d_{n-2})$ of inter-pixels distances. The $(2n-2)$-uplet $(d_0, d_1, \ldots, d_{n-2}, \theta_0, \theta_1, \ldots, \theta_{n-2})$ constitutes what we call the connexion rule, denoted by R_n. In this connexion rule, d_k and $\theta_k (0 \leq k \leq n-2)$ express respectively the inter-pixels distance and direction between pixels i_k and i_{k+1}.

We restrict our methodology to structural parameters expressed as follows:

$$Para_n = \sum_{i_0=0}^{\mathrm{Max}Gs} \sum_{i_1=0}^{\mathrm{Max}Gs} \cdots \sum_{i_{n-1}=0}^{\mathrm{Max}Gs} \left(\psi(i_0, i_1, \ldots, i_{n-1}) \times P_{i_0 i_1 \ldots i_{n-1}} \right) \tag{11.5}$$

Equation 11.5 is a subset of Eq. 11.4 that doesn't consider textural parameters expressed as quadratic, logarithmic or exponential function of the co-occurrence matrix. The Table 11.1 presents some of the concerned structural parameters, where:

- N is the number of elements of the connexion domain D;
- n is the order of the texture parameter;
- μ_x is the marginal mean with respect to the first column x;
- σ_{i_k} is the standard deviation with respect to the kth column;

11.2.2 From the Classical to the New Formulation

The term $\psi(i_0 i_1 \ldots i_{n-1}) \times P_{i_0 i_1 \ldots i_{n-1}}$ can be written in the following form:

$$\psi(i_0 i_1 \ldots i_{n-1}) \times P_{i_0 i_1 \ldots i_{n-1}} = \underbrace{\psi(i_0 i_1 \ldots i_{n-1}) + \psi(i_0 i_1 \ldots i_{n-1}) + \cdots + \psi(i_0 i_1 \ldots i_{n-1})}_{P_{i_0 i_1 \ldots i_{n-1}} \text{ times}} \tag{11.6}$$

Let denote by $\Omega_{\mathrm{Max}Gs}$ the set of all possible n-uplets $(i_0, i_1, \ldots, i_{n-1})$ of grey scales in the image. The following equation can be written.

$$\sum_{i_0=0}^{\mathrm{Max}Gs} \sum_{i_1=0}^{\mathrm{Max}Gs} \cdots \sum_{i_{n-1}=0}^{\mathrm{Max}Gs} \varphi(i_0, i_1, \ldots, i_{n-1}) = \sum_{(i_0, i_1, \ldots, i_{n-1}) \in \Omega_{\mathrm{Max}Gs}} \varphi(i_0, i_1, \ldots, i_{n-1}) \tag{11.7}$$

Since $\Omega_{\mathrm{Max}Gs}$ contains $(\mathrm{Max}Gs + 1)^n$ terms, it can be written in extension as follows:

$$\Omega_{\mathrm{Max}Gs} = \{(i_0^1, i_1^1, \ldots, i_{n-1}^1), (i_0^2, i_1^2, \ldots, i_{n-1}^2), \ldots, (i_0^m, i_1^m, \ldots, i_{n-1}^m)\} \tag{11.8}$$

with $m = (\mathrm{Max}Gs + 1)^n$.

Combining Eqs. 11.7 and 11.8, we obtain the following:

Table 11.1 Experimental texture parameters

Parameter	Classical expression	Corresponding function ψ
Mean	$\frac{1}{N}\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\dots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\left(i_0\times P_{i_0 i_1\dots i_{n-1}}\right)$	i_0
Dissymmetry	$\frac{1}{N}\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\dots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\left(\sum_{k=0}^{n-1}\sum_{l=k+1}^{n}\lvert i_k-i_l\rvert\times P_{i_0 i_1\dots i_{n-1}}\right)$	$\sum_{k=0}^{n-1}\sum_{l=k+1}^{n}\lvert i_k-i_l\rvert$
Inverse difference	$\frac{1}{N}\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\dots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\left(\frac{P_{i_0 i_1\dots i_{n-1}}}{1+\sum_{k=0}^{n-1}\sum_{l=k+1}^{n}\lvert i_k-i_l\rvert}\right)$	$\frac{1}{1+\sum_{k=0}^{n-1}\sum_{l=k+1}^{n}\lvert i_k-i_l\rvert}$
Contrast	$\frac{1}{N}\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\dots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\left(\sum_{k=0}^{n-2}\sum_{l=k+1}^{n-1}(i_k-i_l)^2 P_{i_0 i_1\dots i_{n-1}}\right)$	$(i_k-i_l)^2$
Correlation	$\frac{1}{N}\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\dots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\left[\frac{\Pi_{k=0}^{n-1}(i_k-\mu_{i_k})}{\Pi_{k=0}^{n-1}\sigma_{i_k}}P_{i_0 i_1\dots i_{n-1}}\right]$	$\frac{\Pi_{k=0}^{n-1}(i_k-\mu_{i_k})}{\Pi_{k=0}^{n-1}\sigma_{i_k}}$
Covariance	$\frac{1}{N}\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\dots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\left[\Pi_{k=0}^{n-1}(i_k-\mu_{i_k})P_{i_0 i_1\dots i_{n-1}}\right]$	$\Pi_{k=0}^{n-1}(i_k-\mu_{i_k})$
Standard deviation	$\frac{1}{N}\sqrt{\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\dots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\left[(i_0-\mu_x)^2 P_{i_0 i_1\dots i_{n-1}}\right]}$	$(i_0-\mu_x)^2$
Cluster shade	$\frac{1}{N}\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\dots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\left(\sum_{k=0}^{n-1}i_k-n_{\mu_x}\right)^3 P_{i_0 i_1\dots i_{n-1}}$	$\left(\sum_{k=0}^{n-1}i_k-n_{\mu_x}\right)^3$
Cluster prominence	$\frac{1}{N}\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\dots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\left(\sum_{k=0}^{n-1}i_k-n_{\mu_{i_1}}\right)^4 P_{i_0 i_1\dots i_{n-1}}$	$\left(\sum_{k=0}^{n-1}i_k-n_{\mu_{i_1}}\right)^4$
Great numbers influence	$\frac{1}{N}\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\dots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\left[\left(\sum_{k=0}^{n-1}(i_k)^2\right)P_{i_0 i_1\dots i_{n-1}}\right]$	$\left(\sum_{k=0}^{n-1}(i_k)^2\right)$
Small numbers influence	$\frac{1}{N}\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\dots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\left(\frac{P_{i_0 i_1\dots i_{n-1}}}{\sum_{k=0}^{n-1}(i_k)^2}\right)$	$\frac{1}{\sum_{k=0}^{n-1}(i_k)^2}$
Reverse differential moment	$\frac{1}{N}\sqrt{\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\dots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\left(\frac{P_{i_0 i_1\dots i_{n-1}}}{1+\sum_{k=0}^{n-2}\sum_{l=k+1}^{n-1}(i_k-i_l)^2}\right)}$	$\frac{1}{1+\sum_{k=0}^{n-2}\sum_{l=k+1}^{n-1}(i_k-i_l)^2}$
Variance	$\frac{1}{N}\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\dots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\left[(i_0-\mu_x)^2 P_{i_0 i_1\dots i_{n-1}}\right]$	$(i_0-\mu_x)^2$

$$\sum_{i_0=0}^{\mathrm{Max}Gs}\sum_{i_1=0}^{\mathrm{Max}Gs}\cdots\sum_{i_{n-1}=0}^{\mathrm{Max}Gs}\varphi(i_0,i_1,\ldots,i_{n-1}) = \varphi(i_0^1,i_1^1,\ldots,i_{n-1}^1)+\varphi(i_0^2,i_1^2,\ldots,i_{n-1}^2)+\cdots+\varphi(i_0^m,i_1^m,\ldots,i_{n-1}^m) \tag{11.9}$$

Equation 11.5 becomes:

$$\begin{aligned}Para_n &= \sum_{i_0=0}^{Ng}\sum_{i_1=0}^{Ng}\cdots\sum_{i_{n-1}=0}^{Ng}\psi(i_0,i_1,\ldots,i_{n-1})P_{i_0,i_1,\ldots,i_{n-1}}\\ &= \psi(i_0^1,i_1^1,\ldots,i_{n-1}^1)P_{i_0,i_1,\ldots,i_{n-1}}+\cdots+\psi(i_0^2,i_1^2,\ldots,i_{n-1}^2)P_{i_0,i_1,\ldots,i_{n-1}}\end{aligned} \tag{11.10}$$

Combining Eqs. 11.6 and 11.10, we obtain the following equation:

$$Para_n = \begin{cases}\left(\underbrace{\psi(i_0^1,i_1^1,\ldots,i_{n-1}^1)+\psi(i_0^1,i_1^1,\ldots,i_{n-1}^1)+\cdots+\psi(i_0^1,i_1^1,\ldots,i_{n-1}^1)}_{P_{i_0^1 i_1^1\ldots i_{n-1}^1}\text{ times}}\right)\\ +\left(\underbrace{\psi(i_0^2,i_1^2,\ldots,i_{n-1}^2)+\psi(i_0^2,i_1^2,\ldots,i_{n-1}^2)+\cdots+\psi(i_0^2,i_1^2,\ldots,i_{n-1}^2)}_{P_{i_0^2 i_1^2\ldots i_{n-1}^2}\text{ times}}\right)\\ m=(\mathrm{Max}Gs+1)^n\\ +\cdots\\ +\left(\underbrace{\psi(i_0^m,i_1^m,\ldots,i_{n-1}^m)+\psi(i_0^m,i_1^m,\ldots,i_{n-1}^m)+\cdots+\psi(i_0^m,i_1^m,\ldots,i_{n-1}^m)}_{P_{i_0^m i_1^m\ldots i_{n-1}^m}\text{ times}}\right)\end{cases} \tag{11.11}$$

This equation is equivalent to the following:

$$Para_n = \sum_{(i_0,i_1,\ldots,i_{n-1})\in\bigsqcup_{Ng}}\left\{\psi(i_0,i_1,\ldots,i_{n-1})\times\left[\delta\left\langle(i_0,i_1,\ldots,i_{n-1}),R_n,W\right\rangle\right]_0^1\right\} \tag{11.12}$$

where $\left[\delta\left\langle(i_0,i_1,\ldots,i_{n-1}),R_n,W\right\rangle\right]_0^1$ is a binary function that takes the value 1 if the n-uplet $(i_0,i_1,\ldots,i_{n-1})$ following the connexion rule R_n is entirely included in the image window W. This operation aims at eliminating the entries of the co-occurrence matrix with value 0, i.e. the n-uplets of $\Omega_{\mathrm{Max}Gs}$ that don't occur in the image window W, considering the connexion rule.

Equation 11.12 can be written as follows:

$$Para_n = \sum_{D=\left\{(i_0,i_1,\ldots,i_{n-1})_{R_n,W}\right\}\subset\Omega_{\mathrm{Max}Gs}} \psi(i_0, i_1, \ldots, i_{n-1}) \tag{11.13}$$

where $D = \left\{(i_0, i_1, \ldots, i_{n-1})_{R_n,W}\right\}$ called *connexion domain* is the subset of $\Omega_{\mathrm{Max}Gs}$ containing all the n-uplets $(i_0, i_1, \ldots, i_{n-1})$ of grey scales that verify the connexion rule R_n in the image window W.

Let define a function φ_{R_n} by:

$$\begin{aligned} \varphi_{R_n} : W &\longmapsto \{0, 1, 2, \ldots, \mathrm{Max}Gs\}^n \\ (p, q) &\longmapsto (i_0, i_1, \ldots, i_{n-1}) \end{aligned} \tag{11.14}$$

With $(i_0, i_1, \ldots, i_{n-1})$ following the connexion rule R_n and $W[p, q] = i_0$. Combining Eqs. 11.13 and 11.14, one obtain the following:

$$Para_n = \sum_{p=0}^{NC-1} \sum_{q=0}^{NL-1} \left\{ \psi\left(\varphi_{R_n}(p, q)\right) \times \left[\delta_{R_n}(p, q, W)\right]_0^1 \right\} \tag{11.15}$$

where NC and NL are respectively the number of columns and lines of the image window W; $\left[\delta_{R_n}(p, q, W)\right]_0^1$ being a binary function that returns the value 1 if from the pixel at location (p, q) in the image window, one can construct an n-uplet $(W[p, q], i_1, \ldots, i_{n-1})$ following the connexion rule R_n. On the connexion domain, the value of this function is 1 and out of the connexion domain, its value is 0. Equation 11.15 is then equivalent to the following:

$$Para_n = \sum_{(p,q)\in(D\subset W)} \psi\left(\varphi_{R_n}(p, q)\right) \tag{11.16}$$

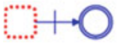
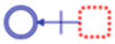
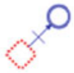
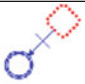
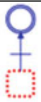
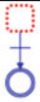
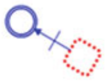
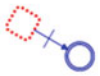
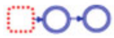
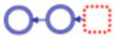
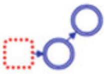
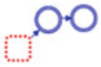
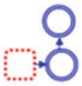
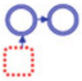
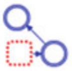
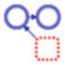
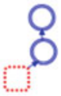
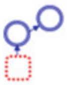
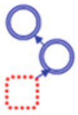
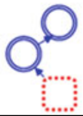
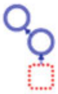
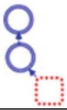
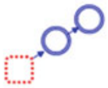
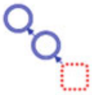
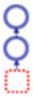
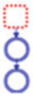
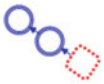
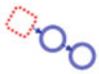
Equation 11.16 is the proposed formulation of the parameter. It gives exactly the same value as the classical expression in Eq. 11.5, but the calculation of the co-occurrence matrix is avoided. The last thing to do now is the construction of the connexion domain D.

11.2.3 Construction of the Connexion Domain

Let materialise the connexion rule by a *connexion arm* with the following principles:

1. the doted square materialises the first pixel of the n-uplet, i.e. the pixel at location (p, q) in Eq. 11.16;

Table 11.2 Some connexion arms associated to their connexion rules

Order	**Connexion rules with their connexion arms**			
2	**R2(2,0°)**	**R2(2,180°)**	**R2(2,45°)**	**R2(2,225°)**
				
	R2(2,90°)	**R2(2,270°)**	**R2(2,135°)**	**R2(2,315°)**
				
3	**R3(1,1,0°,0°)**	**R3(1,1,180°,180°)**	**R3(1,1,0°,45°)**	**R3(1,1,45°,0°)**
				
	R3(1,1,0°,90°)	**R3(1,1,90°,0°)**	**R3(1,1,0°,135°)**	**R3(1,1,135°,0°)**
				
	R3(1,1,45°,90°)	**R3(1,1,90°,45°)**	**R3(1,1,45°,135°)**	**R3(1,1,135°,45°)**
				
	R3(1,1,90°,135°)	**R3(1,1,135°,90°)**	**R3(1,1,45°,45°)**	**R3(1,1,225°,225°)**
				
	R3(1,1,90°,90°)	**R3(1,1,270°,270°)**	**R3(1,1,135°,135°)**	**R3(1,1,315°,315°)**
				

2. each cross bar materialises the jump of one pixel (case inter-pixel distance greater than one);
3. each circle materialises a pixel to consider and
4. the connexion arm, to produce a valid n-uplet, must be entirely included in the image window. The Table 11.2 presents some connexion arms.

The various locations that can occupy the doted square such that the connexion arm is totally included in the image window constitute the connexion domain. Therefore, once the connexion rule known, the connexion domain can be delimited. This domain is always a rectangle. Then Eq. 11.16 can be written in the following form:

Table 11.3 Definition of δ_i and δ'_i

$\theta_\mathbf{i}$	0°	45°	90°	135°	180°	225°	270°	315°
δ_i	1	1	0	−1	−1	−1	0	1
δ'_i	0	1	1	1	0	−1	−1	−1

$$Para_n = \sum_{i=D_{x0}}^{D_{x1}} \sum_{j=D_{y0}}^{D_{y1}} \psi\Big(\varphi_{R_n}(i, j)\Big) \tag{11.17}$$

with $0 \le D_{x0} \le D_{x1} < NC$ and $0 \le D_{y0} \le D_{y1} < NL$. NL and NC being respectively the number of lines and columns of the image window.

11.2.4 Construction of $\varphi_{R_n}(p, q)$

Considering D as the connexion domain, $\varphi_{R_n}(p, q)$ can be defined by the following equation:

$$\begin{aligned} \varphi_{R_n} : D \subset W &\longmapsto \{0, 1, 2, \ldots, \mathrm{Max}Gs\}^n \\ (p, q) &\longmapsto \Big(\varphi^0_{R_n}(p, q), \varphi^1_{R_n}(p, q), \ldots, \varphi^{n-1}_{R_n}(p, q)\Big) \end{aligned} \tag{11.18}$$

with

$$\begin{cases} \varphi^0_{R_n}(p, q) = (p, q) \\ \varphi^{i+1}_{R_n}(p, q) = \varphi^i_{R_n}(p, q) + (\delta_i \times d_i, \delta'_i \times d_i),\ 0 \le i < n - 1 \end{cases} \tag{11.19}$$

δ_i and δ'_i being defined at the Table 11.3, with $R_n = \Big((d_i)_{0\le i<n}, (\theta_i)_{0\le i<n}\Big)$.

11.2.5 Parallelisation

Let suppose that we have a parallel machine with N (N less than the size of the connexion domain) processors. Since the connexion domain D is a rectangle, we can divide it into N sub-rectangles D_i, $1 \le i < N$ verifying the following equation:

$$D = \cup_i D_i \quad and \quad D_k \cap D_j = \Phi \quad \forall k \neq j$$

Equation 11.16 becomes the following:

$$Para_n = \sum_{D_i} \left[\sum_{(p,q)\in D_i} \psi\left(\varphi_{R_n}(p,q)\right) \right] \tag{11.20}$$

For example, considering Eq. 11.17, we have:

$$Para_n = \sum_{i=D_{x0}}^{D_{x1}} \sum_{j=D_{y0}}^{D_{y1}} \psi\left(\varphi_{R_n}(i,j)\right) = \sum_{i=D_{x0}}^{D_{x1}} \left(\sum_{j=D_{y0}}^{D_{y1}} \psi\left(\varphi_{R_n}(i,j)\right)\right) = \sum_{i=D_{x0}}^{D_{x1}} \left(P_i(i)\right) \tag{11.21}$$

So, with a CPU made of $D_{x1} - D_{x0} + 1$ processors $\left\{P_i\right\}_{0\le i\le D_{x1}-D_{x0}}$, each processor can compute one line of the connexion domain and the final result is obtained by adding the various sub-results given by each processor. Since the connexion domain is a rectangle, it is possible to decompose it into N sub-domains Di verifying the Eq. 11.16.

11.2.6 Characterisation of the Parallelisation

11.2.6.1 Acceleration

The acceleration of the parallelisation measures the performance growth of the parallel approach with respect to the number of processors.

Let denote by $A(N)$ the acceleration of the proposed parallel approach; T_{sec} the required time for a sequential functioning and T_{par} the required time for the parallel functioning. Considering Eq. 11.16, since the intermediary results are not reused and the final result obtained just by adding the subresults obtained by each processor, the following equation is verified.

$$A(N) = \frac{T_{sec}}{T_{par}} \cong N \tag{11.22}$$

11.2.6.2 Effectiveness

The effectiveness of a parallelisation expresses the degree of parallel ressources usage. It is given by the ratio of the acceleration by the number of processors. According to Eq. 11.22, we have the following equation:

$$E = \frac{A(N)}{N} \cong 1 \tag{11.23}$$

11.2.6.3 Scalability

The scalability measures the capacity of the approach to see its performances increase with the number of processors. The scalability of this parallelisation is given by the following equation:

$$S(N) = \frac{T_{par}(K * N)}{T_{par}(N)} \cong K \tag{11.24}$$

Equations 11.22, 11.23 and 11.24 demonstrate that this parallelisation is an ideal case.

11.2.7 Algorithmic Complexity

This section deals with time and space complexities. For this purpose, we consider an image window W of size $NL * NC$, NL and NC being respectively the number of columns and lines of the image window. Let consider MaxGs as the maximum grey scale in the image and let compute successively the time and space complexities required by both the co-occurrence matrix approach and the new approach, considering the case of a single processor.

11.2.7.1 Time Complexity

Let denote by $T_C(Para_n)$ and $T_N(Para_n)$ respectively the time complexity of the co-occurrence matrix approach and the time complexity of the proposed (new) approach for the calculation of the parameter $Para_n$ related to the image window W.

Classical method

- The classical method $Para_n$ defined in Eq. 11.5 generates exactly $(\text{Max}Gs + 1)^n$ terms $\psi(i_0, i_1, \ldots, i_{n-1}) * P_{i_0 i_1 \ldots i_{n-1}}$.
- Each term requires the computation of $\psi(i_0, i_1, \ldots, i_{n-1})$ which takes $O(n)$ as time complexity.
- The computation of the order-n co-occurrence matrix requires $\Omega\left[(\text{Max}Gs + 1)^n\right]$ as time complexity.

The time complexity of the classical approach is then given by the following equation:

$$T_C(Para_n) = \Omega\left[(\text{Max}Gs + 1)^n\right] + O(n) * (\text{Max}Gs + 1)^n \tag{11.25}$$

We remember that $\Omega\Big(T(n)\Big)$ is the set of functions that grow at least as fast as $T(n)$ and $O\Big(T(n)\Big)$ is the set of functions that grow at most as fast as $T(n)$.

Proposed method

According to Eq. 11.17, the proposed approach requires the calculation of less than $NL * NC$ terms $\psi\Bigg(\varphi_{R_n}(i, j)\Bigg)$, and each term requires approximately $O(n)$ instructions. So, the time complexity of the proposed approach is given by the following equation:

$$T_N(Para_n) = O(n) * \Big(O(NL * NC)\Big) \tag{11.26}$$

Comparison of the time complexities

The ratio between the two time complexities expressed in Eqs. 11.25 and 11.26 is given by the following equation:

$$\frac{T_C(Para_n)}{T_N(Para_n)} = \Omega\Big[(\text{Max}Gs + 1)^n\Big] \tag{11.27}$$

We realise that this ratio exponentially increases with the order of the parameter. i.e. for example, at order 5, for an image with 256 grey scales, the new approach is more than 2^{32} times faster than the classical approach by co-occurrence matrix.

11.2.7.2 Spatial Complexity

Let denote by $S_C(Para_n)$ and $S_N(Para_n)$ respectively the spatial complexity of the classical co-occurrence matrix approach and the spatial complexity of the proposed (new) approach for the calculation of the parameter $Para_n$ related to the image window W.

Classical approach

The classical approach mainly requires the storage of the high order co-occurrence matrix and the global image. Considering the integer MaxGs as the Maximum grey scale in the image, the order-n co-occurrence matrix requires at least $(\text{Max}Gs + 1)^n$ bytes. Considering that the image is an 8-bits image, the storage of the image requires $NC * NL$ bytes. NL and NC being respectively the number of lines and columns of the image. The total required space for the computation of the order-n parameter is then given by the following equation:

$$S_C(Para_n) = O\left((\text{Max}Gs + 1)^n\right) + O(NL * NC) \tag{11.28}$$

Proposed approach

With the new approach, the calculation of the higher order co-occurrence matrix is avoided. The space required is then the space needed for image storage. So, the spatial complexity of the proposed approach is given by the following equation.

$$S_N(Para_n) = O(NL * NC) \tag{11.29}$$

Comparison of the two approaches

From Eqs. 11.28 and 11.29, one can deduce the following equation:

$$S_C - S_N = (\text{Max}Gs + 1)^n \tag{11.30}$$

Most of the time, as the order n increases, the quantity $NL * NC$ becomes too small, as compared to the quantity $(\text{Max}Gs + 1)^n$. In this case, one can write the following comparative equation:

$$\frac{S_C}{S_N} = O\left((\text{Max}Gs + 1)^{n-1}\right) \tag{11.31}$$

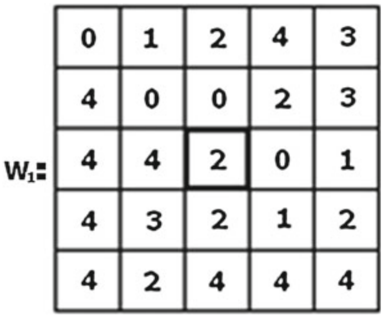
In conclusion, we deduce that the new approach for textural parameters computation saves both time and space, compared to the classical method by co-occurrence matrix.

11.3 Experimentation

In the first sub-section, we use successively the classical and the new approaches to calculate the parameter Dissimilarity at orders 2 and 3. In a second sub-section, we experiment the approach on a synthetic texture of Brodatz.

11.3.1 Computation of the Parameter Dissimilarity

Let consider the following image window W_1 (Fig. 11.1) and the connection rule $R_2 = (2, 45°)$. The goal is to verify that both the two approaches give the same result.



$$W_1 = \begin{array}{|c|c|c|c|c|}\hline 0 & 1 & 2 & 4 & 3 \\ \hline 4 & 0 & 0 & 2 & 3 \\ \hline 4 & 4 & \boxed{2} & 0 & 1 \\ \hline 4 & 3 & 2 & 1 & 2 \\ \hline 4 & 2 & 4 & 4 & 4 \\ \hline \end{array}$$

Fig. 11.1 Experimental image window

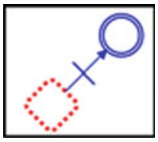
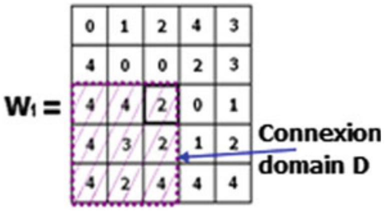


Fig. 11.2 Connexion arm corresponding to the connexion rule $R_2 = (2, 45°)$



$$W_1 = \begin{array}{|c|c|c|c|c|}\hline 0 & 1 & 2 & 4 & 3 \\ \hline 4 & 0 & 0 & 2 & 3 \\ \hline 4 & 4 & \boxed{2} & 0 & 1 \\ \hline 4 & 3 & 2 & 1 & 2 \\ \hline 4 & 2 & 4 & 4 & 4 \\ \hline \end{array}$$

Fig. 11.3 Identification of the connexion domain in the image window

11.3.1.1 Order 2 Parameter

The mathematical expression of the parameter Dissimilarity at order 2 is given by the following equation:

$$Diss_2 = \sum_{i=0}^{\mathrm{Max}Gs} \sum_{j=0}^{\mathrm{Max}Gs} |i - j| \, P_{ij} \tag{11.32}$$

New approach

The maximum grey scale in W_1 is 4. The connexion arm corresponding to the connexion rule $R_2 = (2, 45°)$ and the resulting connexion domain are given in the Figs. 11.2 and 11.3.

Once the connexion domain identified, we can enumerate all the couples (i, j) involved in Eq. 11.17 thanks to the Fig. 11.4. The above figure helps to define in extension the connexion domain as follows:

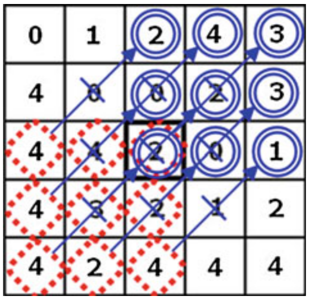


Fig. 11.4 Identification of all the couples (i, j) involved in Eq. 11.17 on the experimental image

Table 11.4 construction of the function $\varphi_{(2,45^\circ)}$

$\varphi_{(2,45^\circ)}$	2	3	4
0	(4,2)	(4,0)	(4,2)
1	(4,4)	(3,2)	(2,0)
2	(2,3)	(2,3)	(4,1)

$$D = \Big\{(0,2), (1,2), (2,2), (0,3), (1,3), (2,3), (0,4), (1,4), (2,4)\Big\} \qquad (11.33)$$

From the connexion domain D, we construct the following function $\varphi_{(2,45^\circ)}$ that enumerates all the couples (i, j) involved in the computation of the parameter (Table 11.4).

The function $\varphi_{(2,45^\circ)}$ defines all the couples of pixels (grey scales) that follow the connexion rule in the image window W_1. Since the connexion domain is made of 9 pixels, the function $\varphi_{(2,45^\circ)}$ will generate 9 couples of grey scales in the image window following the connexion rule $R_2 = (2, 45^\circ)$. These couples are the following: $(4,2), (4,4), (2,3), (4,0), (3,2), (2,3), (4,2), (2,0)$ *and* $(4,1)$.

From Eq. 11.17, one can write the following equation:

$$Diss_2 = \sum_{(p,q)\in D} \psi\Big(\varphi_{(2,45^\circ)}(p,q)\Big) = \sum_{p=2}^{4}\sum_{q=0}^{2} \psi\Big(\varphi_{(2,45^\circ)}(p,q)\Big), \text{ i.e.}$$

$$\begin{aligned}
Diss_2 = {} & \psi\Big(\varphi_{(2,45^\circ)}(2,0)\Big) + \psi\Big(\varphi_{(2,45^\circ)}(2,1)\Big) + \psi\Big(\varphi_{(2,45^\circ)}(2,2)\Big) \\
& + \psi\Big(\varphi_{(2,45^\circ)}(3,0)\Big) + \psi\Big(\varphi_{(2,45^\circ)}(3,1)\Big) + \psi\Big(\varphi_{(2,45^\circ)}(3,2)\Big) \\
& + \psi\Big(\varphi_{(2,45^\circ)}(4,0)\Big) + \psi\Big(\varphi_{(2,45^\circ)}(4,1)\Big) + \psi\Big(\varphi_{(2,45^\circ)}(4,2)\Big) \\
= {} & \psi(4,2) + \psi(4,4) + \psi(2,3) + \psi(4,0) + \psi(3,2) + \psi(2,3) \\
& + \psi(4,2) + \psi(2,0) + \psi(4,1) \\
= {} & |4-2| + |4-4| + |2-3| + |4-0| + |3-2| + |2-3| +
\end{aligned}$$

$$
\begin{aligned}
&|4-2|+|2-0|+|4-1| \\
&= 2+0+1+4+1+1+2+2+3 \\
&= 16
\end{aligned}
$$

Classical approach

Let now calculate the same parameter, using the classical approach. According to Eq. 11.32, it is necessary to calculate the co-occurrence matrix. The related order-2 co-occurrence matrix is the following:

$$
P = \begin{pmatrix} 0\ 0\ 0\ 0\ 0 \\ 0\ 0\ 0\ 0\ 0 \\ 1\ 0\ 0\ 2\ 0 \\ 0\ 0\ 1\ 0\ 0 \\ 1\ 1\ 2\ 0\ 1 \end{pmatrix} \tag{11.34}
$$

Equation 11.32 is specified as follows:

$$
\begin{aligned}
Diss_2 &= (|0-0|P_{00}) + (|0-1|P_{01}) + (|0-2|P_{02}) + (|0-3|P_{03}) + (|0-4|P_{04}) \\
&\quad + (|1-0|P_{10}) + (|1-1|P_{11}) + (|1-2|P_{12}) + (|1-3|P_{13}) + (|1-4|P_{14}) \\
&\quad + (|2-0|P_{20}) + (|2-1|P_{21}) + (|2-2|P_{22}) + (|2-3|P_{23}) + (|2-4|P_{24}) \\
&\quad + (|3-0|P_{30}) + (|3-1|P_{31}) + (|3-2|P_{32}) + (|3-3|P_{33}) + (|3-4|P_{34}) \\
&\quad + (|4-0|P_{40}) + (|4-1|P_{41}) + (|4-2|P_{42}) + (|4-3|P_{43}) + (|4-4|P_{44}) \\
&= (0+0+0+0+0) + (0+0+0+0+0) + (2+0+0+2+0) \\
&\quad + (0+0+1+0+0) + (4+3+4+0+0) \\
&= 16
\end{aligned}
$$

We can notice that both the two approaches give the same result.

11.3.1.2 Order 3 Parameter

An order-3 parameter that can be calculated with our approach is expressed in the following form:

$$
Para_3 = \sum_{i=0}^{\mathrm{Max}Gs} \sum_{j=0}^{\mathrm{Max}Gs} \sum_{k=0}^{\mathrm{Max}Gs} \psi(i, j, k) * P_{ijk} \tag{11.35}
$$

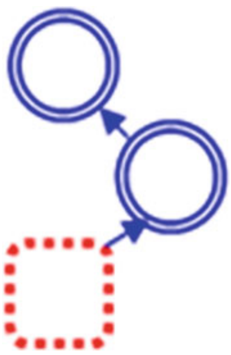


Fig. 11.5 Connexion arm related to the connexion rule R_3

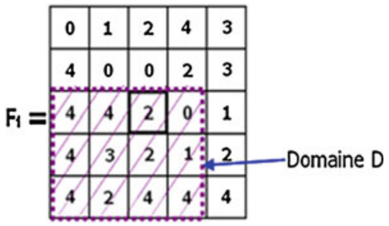


Fig. 11.6 Connexion domain in the experimental image

The order-3 Dissimilarity parameter is expressed as follows:

$$Diss_3 = \sum_{i=0}^{4} \sum_{j=0}^{4} \sum_{k=0}^{4} (|i-j| + |i-k| + |j-k|) * P_{ijk} \tag{11.36}$$

Proposed approach

The maximum grey scale in W_1 is 4. The connexion arm corresponding to the connexion rule $R_3 = (1, 1, 45°, 135°)$ and the resulting connexion domain are given at Figs. 11.5 and 11.6.

The doted square of the connexion arm in the following figure defines the connexion domain D. In extension, the set of pixels constituting the connexion domain is the following (Fig. 11.7):

$$D = \{(2,0), (2,1), (2,2), (2,3), (3,0), (3,1), (3,2), (3,3), (4,0), (4,1), (4,2), (4,3)\} \tag{11.37}$$

Each element of the connexion domain D generates through the application $\varphi_{(1,1,45°,135°)}$ a triplet (i, j, k) of grey scales of pixels following the connexion rule R_3 in the image window. All the triplets of grey scales generated by the function

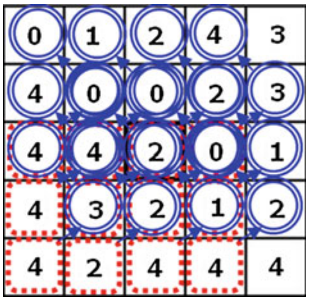


Fig. 11.7 Identification of the connexion domain in the image window

$\varphi_{(1,1,45°,135°)}$ from the connexion domain are the following: (4,0,0), (4,0,1), (2,2,2), (0,3,4), (4,4,4), (3,2,0), (2,0,0), (1,1,2), (4,3,4), (2,2,4), (4,1,2) and (4,2,0).

From Eq. 11.17, one can write the following equation:

$$Diss_3 = \sum_{(p,q)\in D} \psi\left(\varphi_{(1,1,45°,135°)}(p,q)\right) = \sum_{p=2}^{4}\sum_{q=0}^{2} \psi\left(\varphi_{(1,1,45°,135°)}(p,q)\right) \tag{11.38}$$

so,

$$\begin{aligned}
Diss_3 &= \psi\left(\varphi_R(2,0)\right) + \psi\left(\varphi_R(2,1)\right) + \psi\left(\varphi_R(2,2)\right) + \psi\left(\varphi_R(2,3)\right)\\
&\quad + \psi\left(\varphi_R(3,0)\right) + \psi\left(\varphi_R(3,1)\right) + \psi\left(\varphi_R(3,2)\right) + \psi\left(\varphi_R(3,3)\right)\\
&\quad + \psi\left(\varphi_R(4,0)\right) + \psi\left(\varphi_R(4,1)\right) + \psi\left(\varphi_R(4,2)\right) + \psi\left(\varphi_R(4,3)\right)\\
&= \psi(4,0,0) + \psi(4,0,1) + \psi(2,2,2) + \psi(0,3,4) + \psi(4,4,4) + \psi(3,2,0)\\
&\quad + \psi(2,0,0) + \psi(1,1,2) + \psi(4,3,4) + \psi(2,2,4) + \psi(4,1,2) + \psi(4,2,0)\\
&= 8+8+0+8+0+6+4+2+2+4+6+8\\
&= 56
\end{aligned}$$

Classical approach

Let now calculate the same parameter, using the classical approach. According to Eq. 11.36, it is necessary to calculate the co-occurrence matrix. The related order-3 co-occurrence matrix is the following: This matrix expresses, for each entry

$$p = \left\{\begin{array}{cccccccccccccccccccccccccc}
ij/k & 00 & 01 & 02 & 03 & 04 & 10 & 11 & 12 & 13 & 14 & 20 & 21 & 22 & 23 & 24 & 30 & 31 & 32 & 33 & 34 & 40 & 41 & 42 & 43 & 44\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0\\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\
2 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\
3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}\right]$$

(i, j, k), the number of times this triplets of grey scales occur in the image window with respect to the connexion rule R_3.

From Eq. 11.36, it appears that, each time the entry of the co-occurrence matrix P is null, the term is also null. Then we can just consider the triplets (i, j, k) of grey scales for which the entry in the matrix is not null. Equation 11.36 becomes:

$$\begin{aligned}
Diss_3 &= \sum_{i=0}^{4}\sum_{j=0}^{4}\sum_{k=0}^{4} \psi(i, j, k) P_{ijk} \\
&= \psi(0,3,4)P_{0,3,4} + \psi(1,1,2)P_{1,1,2} + \psi(2,0,0)P_{2,0,0} + \psi(2,2,2)P_{2,2,2} \\
&\quad + \psi(2,2,4)P_{2,2,4} + \psi(3,3,0)P_{3,2,0} + \psi(4,0,0)P_{4,0,0} + \psi(4,0,1)P_{4,0,1} \\
&\quad + \psi(4,1,2)P_{4,1,2} + \psi(4,2,0)P_{4,2,0} + \psi(4,3,4)P_{4,3,4} + \psi(4,4,4)P_{4,4,4} \\
&= \psi(0,3,4) + \psi(1,1,2) + \psi(2,0,0) + \psi(2,2,2) \\
&\quad + \psi(2,2,4) + \psi(3,3,0) + \psi(4,0,0) + \psi(4,0,1) \\
&\quad + \psi(4,1,2) + \psi(4,2,0) + \psi(4,3,4) + \psi(4,4,4) \\
&= (|0-3| + |0-4| + |3-4|) + (|1-1| + |1-2| + |1-2|) + (|2-0| \\
&\quad + |2-0| + |0-0|) \\
&\quad + (|2-2| + |2-2| + |2-2|) + (|2-2| + |2-4| + |2-4|) + (|3-3| \\
&\quad + |3-0| + |3-0|) \\
&\quad + (|4-0| + |4-0| + |0-0|) + (|4-0| + |4-1| + |0-1|) + (|4-1| \\
&\quad + |4-2| + |1-2|) \\
&\quad + (|4-2| + |4-0| + |2-0|) + (|4-3| + |4-4| + |3-4|) + (|4-4| \\
&\quad + |4-4| + |4-4|) \\
&= 8 + 2 + 4 + 0 + 4 + 6 + 8 + 8 + 6 + 8 + 2 + 0 \\
&= 56
\end{aligned} \tag{11.39}$$

We can notice that both the two approaches give exactly the same result.

11.3.2 Experimentation on a Synthetic Image

For experimentation, let construct the following experimental image of size 500 lines and 500 columns from the Brodatz texture image, notably textures $D4$, $D91$, $D86$ and $D32$.

For this experimental image, let calculate the image textures generated by the texture parameters Mean, Dissimilarity and Reverse Difference. Wang [8] demonstrated that, above order 5 the image texture is blurred. For this reason, we will limit on order 5 in our experimentation. The expression of these texture parameters using successively the classical and the proposed formulation for a generic order n is presented in the Table 11.5.

Where D is the connexion domain, (p, q) are the coordinates of a pixel in the connexion domain, x_0 is the grey scale of the pixel at location (p, q) and i_k is the k^{th} grey scale of the n-uplet starting at location (p, q) and following the connexion rule (Fig. 11.8).

For this purpose, we consider the connexion rule $R_2 = (2, 45°)$ for the order-2 parameter, $R_3 = (1, 1, 45°, 135°)$ for the order-3 parameter, $R_4 = (1, 1, 1, 0°, 45°, 135°)$ for the order-4 parameter and $R_5 = (1, 1, 1, 1, 0°, 45°, 0°, 135°)$ for the order 5 parameter.

Obviously, the results are exactly the same. But the time complexity differs. We implemented the algorithm on a computer having the following characteristics: Hard disk: 500 GB; RAM: 4 GB; CPU AMD Dual Core processor $E - 300$, 3.6 GHz. We introduced a parameter *counter* in each approach to count the number of instructions executed by the program and we came out with the following time complexities.

11.3.2.1 Operational Complexity

Parameter Mean

The following table presents the time complexity of the parameter Mean, provided by the two approaches (Fig. 11.9).

Parameter Dissimilarity

The following table presents the time complexity of the parameter Dissimilarity, provided by the two approaches (Fig. 11.10).

Parameter Reverse Difference

The following table presents the time complexity of the parameter Reverse Difference, provided by the two approaches (Fig. 11.11).

Comparative study of the two computational complexities

From the three comparative graphs, one can make the following observations:

- The number of instructions required by the new approach is almost constant with respect to the order of the parameter;
- The number of instructions required by the classical approach increases exponentially with the order of the parameter;

Table 11.5 Experimental texture parameters

Parameter	Classical expression	New expression
Mean	$\frac{1}{N}\sum_{i_0=0}^{\text{Max}Gs}\sum_{i_1=0}^{\text{Max}Gs}\ldots\sum_{i_{n-1}=0}^{\text{Max}Gs}\left(i_0 \times P_{i_0 i_1 \ldots i_{n-1}}\right)$	$\frac{1}{N}\sum_{(p,q)\in D} x_0$
Dissymmetry	$\frac{1}{N}\sum_{i_0=0}^{\text{Max}Gs}\sum_{i_1=0}^{\text{Max}Gs}\ldots\sum_{i_{n-1}=0}^{\text{Max}Gs}\left(\sum_{k=0}^{n-1}\sum_{l=k+1}^{n} \mid i_k - i_l \mid \times P_{i_0 i_1 \ldots i_{n-1}}\right)$	$\frac{1}{N}\sum_{(p,q)\in D}\left(\sum_{u=0}^{n-1}\sum_{v=u+1}^{n} \mid x_u - x_v \mid\right)$
Reverse difference	$\frac{1}{N}\sum_{i_0=0}^{\text{Max}Gs}\sum_{i_1=0}^{\text{Max}Gs}\ldots\sum_{i_{n-1}=0}^{\text{Max}Gs}\left(\frac{P_{i_0 i_1 \ldots i_{n-1}}}{1+\sum_{k=0}^{n-1}\sum_{l=k+1}^{n} \mid i_k - i_l \mid}\right)$	$\frac{1}{N}\sum_{(p,q)\in D}\left(\frac{1}{1+\sum_{u=0}^{n-2}\sum_{v=u+1}^{n-1} \mid x_u - x_v \mid}\right)$

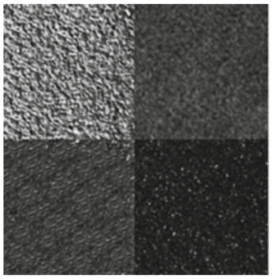


Fig. 11.8 Experimental synthetic image

(a)

Order	Number of instructions		Ratio
	$Classic(C)$	$New(N)$	$R = C/N$
2	$1,31E+5$	100000	$1,31$
3	$3,35E+7$	75000	447
4	$8,59E+9$	50000	$1,72E+5$
5	$2,19E+12$	25000	$8,76E+7$

(b)

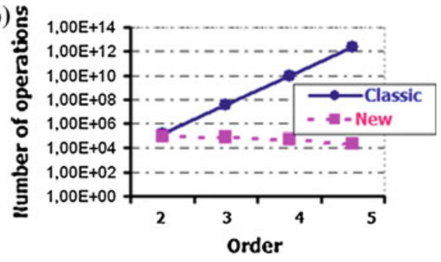


Fig. 11.9 Comparative study of the time complexities of the parameter mean. **a** Complexity. **b** Comparative graph

(a)

Order	Number of instructions		Ratio
	$Classic(C)$	$New(N)$	$R = C/N$
2	$4,91E+8$	150000	3273
3	$3,77E+11$	337500	1117037
4	$1,93E+14$	450000	$4,29E+08$
5	$7,14E+16$	325000	$2,19E+11$

(b)

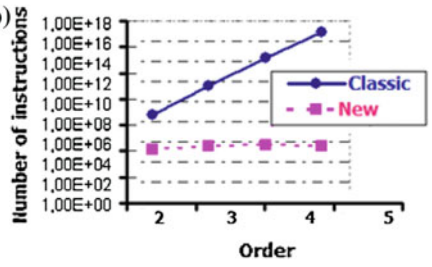


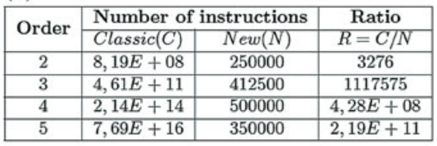
Fig. 11.10 Comparative study of the time complexities of the parameter dissimilarity. **a** Complexity. **b** Comparative graph

- The maximum number of instructions required by the new approach is less than the minimum number of instructions (offered by order 2) required by the classical approach.

11.3.2.2 Time Complexity

As we included a variable *Counter* to count the number of instructions executed by each approach in the previous section, we also included a variable *Clock* in each algorithm to measure the time (seconds) spent by the program to provide the image

(a)



Order	Number of instructions		Ratio
	Classic(C)	*New*(N)	$R = C/N$
2	$8,19E + 08$	250000	3276
3	$4,61E + 11$	412500	1117575
4	$2,14E + 14$	500000	$4,28E + 08$
5	$7,69E + 16$	350000	$2,19E + 11$

(b)

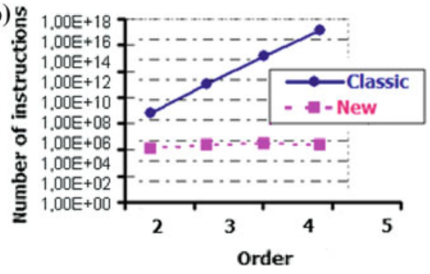
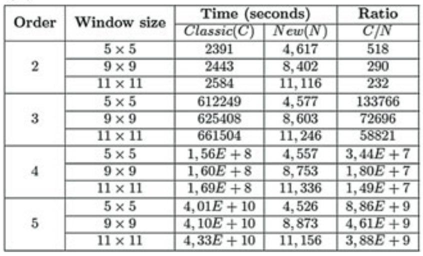


Fig. 11.11 Comparative study of the time complexities of the parameter reverse difference. **a** Complexity. **b** Comparative graph

(a)



Order	Window size	Time (seconds)		Ratio
		Classic(C)	*New*(N)	C/N
2	5×5	2391	$4,617$	518
	9×9	2443	$8,402$	290
	11×11	2584	$11,116$	232
3	5×5	612249	$4,577$	133766
	9×9	625408	$8,603$	72696
	11×11	661504	$11,246$	58821
4	5×5	$1,56E + 8$	$4,557$	$3,44E + 7$
	9×9	$1,60E + 8$	$8,753$	$1,80E + 7$
	11×11	$1,69E + 8$	$11,336$	$1,49E + 7$
5	5×5	$4,01E + 10$	$4,526$	$8,86E + 9$
	9×9	$4,10E + 10$	$8,873$	$4,61E + 9$
	11×11	$4,33E + 10$	$11,156$	$3,88E + 9$

(b)

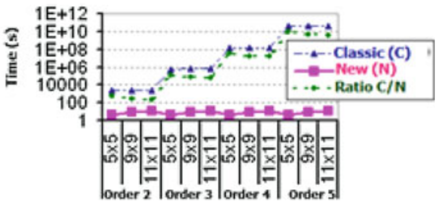


Fig. 11.12 Comparative study of the time complexity of the parameter mean. **a** Complexity. **b** Comparative graph

texture of each parameter. For each order, we varied the size of the image window, considering notably the sizes $5 * 5$, $9 * 9$ and $11 * 11$. We finally obtained the following results:

The parameter Mean

The following table presents the time complexity in seconds of the parameter Mean, provided by the two approaches (Fig. 11.12).

The parameter Dissimilarity

The following table presents the time complexity in seconds of the parameter Dissimilarity, provided by the two approaches for various sizes of image window (Fig. 11.13).

(a)

Order	Window size	Time (seconds)		Ratio
		$Classic(C)$	$New(N)$	C/N
2	5×5	2414, 91	4, 766	506, 69
	9×9	2467, 43	8, 933	276, 21
	11×11	2609, 84	16, 453	158, 62
3	5×5	618371, 49	4, 987	123996, 68
	9×9	631662, 08	9, 854	64102, 09
	11×11	668119, 04	14, 55	45918, 83
4	5×5	$1,57E+8$	4, 909	$3,20E+7$
	9×9	$1,61E+8$	10, 685	$1,51E+7$
	11×11	$1,70E+8$	13, 089	$1,30E+7$
5	5×5	$4,05E+10$	4, 977	$8,13E+9$
	9×9	$4,14E+10$	11, 717	$3,53E+9$
	11×11	$4,37E+10$	11, 757	$3,72E+9$

(b)

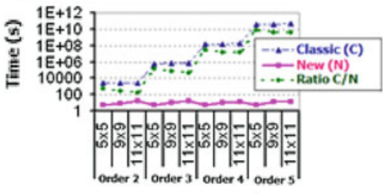


Fig. 11.13 Comparative study of the time complexity of the parameter dissimilarity. **a** Complexity. **b** Comparative graph

(a)

Order	Window size	Time (seconds)		Ratio
		$Classic(C)$	$New(N)$	C/N
2	5×5	2414, 91	4, 766	506, 69
	9×9	2467, 43	8, 933	276, 21
	11×11	2609, 84	16, 453	158, 62
3	5×5	$6,18E+5$	4, 987	$1,23E+5$
	9×9	$6,31E+5$	9, 854	$6,41E+4$
	11×11	$6,68E+5$	14, 55	45918, 83
4	5×5	$1,57E+8$	4, 909	$3,20E+7$
	9×9	$1,61E+8$	10, 685	$1,51E+7$
	11×11	$1,70E+8$	13, 089	$1,30E+7$
5	5×5	$4,05E+10$	4, 977	$8,13E+9$
	9×9	$4,14E+10$	11, 717	$3,53E+9$
	11×11	$4,37E+10$	11, 757	$3,71E+9$

(b)

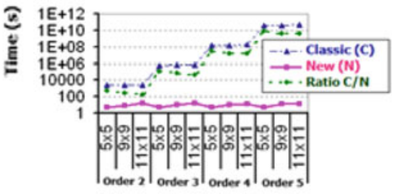


Fig. 11.14 Comparative study of the time complexity of the parameter reverse difference. **a** Complexity. **b** Comparative graph

The parameter Reverse Difference

The following table presents the time complexity in seconds of the parameter Reverse Difference, provided by the two approaches for various sizes of image window (Fig. 11.14).

11.3.3 Result Analysis

From the above result, one notice that the time complexity doesn't automatically increase with the size of the image window.

Using the classical approach, the time complexity exponentially increases with the order of the parameter, in contrary to the new approach for which the time complexity is almost constant, independently to the order of the parameter. It is established that a suitable combination of various orders improves the quality of image classification [13]. This approach offers the opportunity to combine all the four orders in less time than the order-2 classical approach.

11.4 Conclusion

The goal of this chapter was to propose a new approach of texture parameters computation. We started from the mathematical expression of the texture parameter at a generic order-n. This expression is based on the grey level dependent matrix, also called co-occurrence matrix. This co-occurrence matrix is time consuming. Following a series of transformations, we found an equivalent expression of the texture parameter that doesnt depend on the co-occurrence matrix. The proposed approach is about $(\text{Max}Gs + 1)^n$ times faster and about $(\text{Max}Gs + 1)^{n-1}$ times less space consuming than the co-occurrence matrix approach. MaxGs being the maximum grey scale in the image window W. Moreover, in contrary to the classical method, the new approach is parallel. So, with a parallel computer, the new approach allows a real time image analysis.

References

1. Sbai EH (1999) La classification automatique par les statistiques d'ordre. Traitement du Signal 16(6):437–449
2. Akono A, Tonyé E, Ndi Nyoungui A (2003) Nouvelle méthodologie d'évaluation des paramètres de texture d'ordre trois. Int J Remote Sens 24(9):1957–1967
3. Jobanputra R, Clausi DA (2006) Preserving boundaries for image texture segmentation using grey level co-occurring probabilities. Pattern Recogn 39:234–245
4. Blaes X, Vanhalle L, Defourny P (2005) Efficiency of crop identification based on optical and SAR image time series. Remote Sens Environ 96:352–365
5. Huber R (2001) Scene classification of SAR images acquired from antiparallel tracks using evidential and rule-based fusion. Image Vis Comput 19:1001–1010
6. Chorowicz J, Rouis T, Rudant J-P, Manoussis S (1998) Computer aided recognition of relief patterns on radar images using a syntax analysis. Remote Sens Environ 64:221–233
7. Haralick RM (1979) Statistical and structural approaches to texture. In: Proc IEEE 67(5): 786–804
8. Wang L (1994) Vector choice in the texture spectrum approach. Int J Remote Sens 15(18): 3823–3829
9. Unser M (1995) Texture classification and segmentation using wavelet frames. IEEE Trans Image Process 4(11):1549–1560
10. Marceau D, Howarth PJ, Dubois JM, Gratton DJ (1990) Evaluation of the grey-level co-occurrence method for land-cover classification using SPOT imagery. IEEE Trans Geosci Remote Sens 28:513–519
11. Kourgly A. and Belhadj-Aissa A., Nouvel algorithme de calcul des paramètres de texture appliqué la classification d'images satellitaires. Actes des 8 èmes Journées Scientifiques du Réseau Télédétection de l'AUF.
12. Tonyé E, Akono A, Rudant J-P, Dzepa C, Talla Tankam N (2005) Utilisation des signatures de texture d'ordre élevé pour une meilleure discrimination des classes d'occupation du sol sur une image radar a synthèse d'ouverture, vol 179. Revue franaise de photogrammétrie, pp 3–17. ISSN 1768–9791
13. Talla Tankam N, Tonyé E, Dipanda A, Akono A (2006) Classification d'images satellitaires radars RSO par valeurs propres de texture. Application la mangrove littorale Camerounaise, CARI 2006, Cotonou, Benin

Chapter 12
Multi-modal Sensing for Distracted Driving Mitigation Using Cameras and Crowdsourcing

Amol Deshpande, Mahbubur Rahaman, Nilanjan Banerjee, Christophe Bobda and Ryan Robucci

12.1 Introduction

Driving-related accidents and human casualties are on the rise in the US and around the globe [1]. The US government spends more than 10 billion dollars a year to address the aftermath of accidents caused due to distracted driving and driving in dangerous conditions. The situation is exacerbated by the pervasive use of smartphones while driving, integration of app stores into car dashboards, and the increase in the use of cars as a primary mode of transport [1–4]. While certain states in the US have banned texting and the use of handhelds while driving, this point solution is neither comprehensive nor adequate [5]. There are several causes of road accidents that are independent of texting. For instance, poor driving skills, distractions caused by car dashboards, advertisements displayed on roadsides, steep road geometry, inclement weather and road conditions, traffic congestion, and road construction can lead to fatal road accidents. Unfortunately, the state of the art solutions such as intrusive smart phone applications [6–10], initiatives from insurance companies [11] and automobile dealers [12] address a small niche in the problem space. For example, the popular DriveSafely application, which reads out text messages to prevent

A. Deshpande (✉) · M. Rahman · N. Banerjee · R. Robucci
University of Maryland Baltimore County, Baltimore, MD, USA
e-mail: amold1@umbc.edu

M. Rahman
e-mail: mahbub1@umbc.edu

N. Banerjee
e-mail: nilanb1@umbc.edu

R. Robucci
e-mail: robucci@umbc.edu

C. Bobda
University of Arkansas, Fayetteville, AR, USA
e-mail: cbobda@uark.edu

distraction, can in fact increase cognitive distraction [13, 14]. To fill this gap, in this book chapter we introduce an *embedded and sensor systems solution to understanding dangerous driving and techniques to proactively reduce susceptibility to accidents*. Our solution combines behavioral understanding of driving with mobile, sensor, and crowd-sourced systems. Using sophisticated camera sensor analysis, rule mining, and causal analysis, we propose techniques to alert drivers of potentially dangerous driving conditions, and increase awareness of safe driving practices.

The design and implementation of a comprehensive system that proactively determines whether a driver is susceptible to accidents must address the following non-trivial research challenges. First, the problem requires *in-depth understanding of the factors that cause human distraction, and non-intrusive techniques that can detect these factors*. While research exists on human multi-tasking and distraction in the human-centered computing (HCC) and psychology communities [15–21], techniques to understand and detect these factors in a real world setting are scarce. Secondly, *numerous complementary data streams must be fused together to meaningfully infer the susceptibility of users to accidents when driving*. For example, variance in ambient sound (a common cause of distraction) can be captured using smartphone microphones; advertisements on roadsides and road geometry can be apprehended using on-board cameras; and weather condition and traffic congestion data can be gathered using location based web services [22, 23]. Moreover, there are certain dimensions of data that can best be captured using human intervention. For instance, up-to-date information on road conditions (whether it is icy) or roadside constructions can be captured using a human crowd. However, a key challenge is to design techniques that can fuse disparate data collected from machine intelligence (from sensors) and human intelligence (crowd-sourcing) sources. Third, it is important that *the analysis and recommendations to drivers are made pro-actively and in real time*. It is futile to alert a driver of a potential accident five minutes after the accident has occurred. Hence, near real time sensor data analysis, fusion, and inferencing is a fundamental design pillar. In this chapter we describe the following research contributions to address the above challenges.

- **A First Principles Approach to Understanding Driving Behavior**: We present a data-driven approach to understand factors governing distracted driving and significant indicators of dangerous driving. We use non-intrusive sensors on smartphones, cameras, an eye tracking system, and crowd-sourced input from drivers to collect quantitative data on driving behavior and distracted driving. We augment the real-world data collection with in-vivo smartphone surveys and structured interviews. The novel contribution lies in fusing subjective and "in the wild" experimentation to determine statistically significant causal relationships between independent variables (such as weather conditions and road geometry) and accident susceptibility.
- **Proactive Inference of Dangerous Driving Conditions**: The practical applicability of a system that alerts a user of a possible dangerous driving zone is predicated on real time, low latency inference of data collected from multiple and often disparate data sources. Additionally, storing data from cameras and sound

signatures has implicit privacy concerns. To mitigate this challenge, we use a system design that pushes computation closer to the edge devices. For example, an FPGA-based high resolution camera can process images on the edge device. While mitigating privacy concerns, the solution minimizes the amount of data transferred and processed at backend servers. We present a machine learning technique that gathers summary information from edge sensors and fuses it with crowd-sourced data and web services to detect susceptibility to accidents.

The rest of the chapter is organized as follows. Section 12.2 presents driving behavior analytics that combines data from cameras, sensors, and user input. Section 12.3 presents methods for in vitro detecting dangerous and distracted driving. Section 12.4 concludes the chapter.

12.2 Driving Behavior Analytics

Several factors cause driving distraction and can indicate dangerous driving conditions. For example, city drivers are susceptible to accidents due to high traffic congestion, unfriendly road geometries, and distraction caused by ambient noise and roadside advertisements. Moreover, inclement weather conditions, roadside constructions, and incoming phone calls and text messages can exacerbate driving risks. While attempts to study distracted driving and effects of multi-tasking using simulation environments and from a philosophical perspective [16, 18, 19, 24, 25] exist, data analysis on real world driving has primarily focussed on route planning [26–28, 28, 29] and transportation [30–32]. The first step, therefore, is the use of multiple sensor and data collection modalities to understand fundamental indicators of distracted and reckless driving, and factors that make drivers susceptible to accidents. The described system builds on research on data mining [33–35], and wheelchair driving [36–38] and applies it to an unique application domain.

Drawing meaningful inferences from real life data collected in a high risk environment like driving presents several research challenges. First, it is key that reliable groundtruth is collected on whether a car driver is distracted or is driving in a dangerous zone. Unlike controlled driving simulation environments [39] where an expert can continuously scrutinize a subject as he drives in a virtual environment, in the real world, automated reliable groundtruth collection is challenging. To this end, we propose to use eye tracking to determine whether the driver is focussing on the road, driver orientation to infer whether he has his hands on the steering wheel, and input from a human crowd on dangerous driving conditions, as groundtruth. Secondly, there are several factors that can exacerbate distraction and dangerous driving conditions, including ambient sound signatures, inclement weather, incoming phone calls, roadside advertisements, and traffic congestion. Collecting high dimensional data that captures all these attributes using a minimal set of sensors is a fundamental design pillar. We propose to use a collection of custom camera sensors, smartphone sensors, and location services to address the problem. Thirdly, there are several data

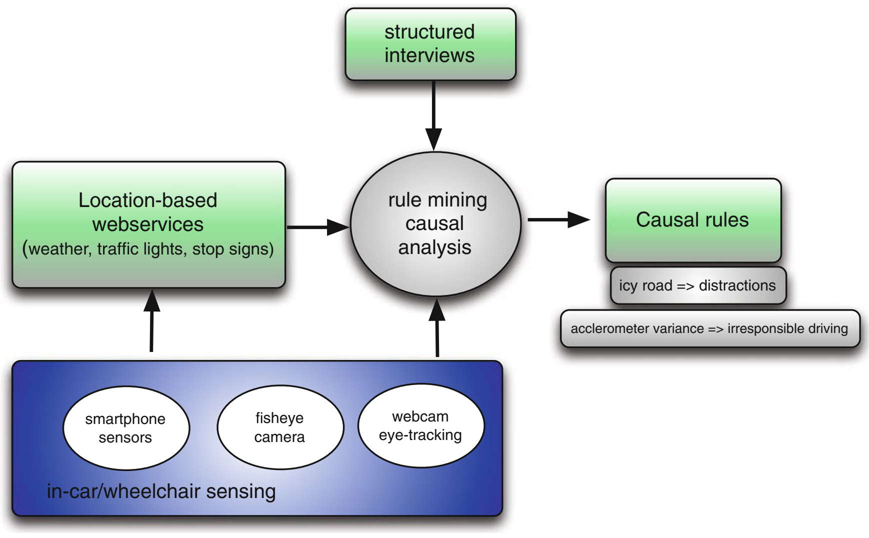

structured interviews
Location-based webservices (weather, traffic lights, stop signs)
rule mining causal analysis
Causal rules
icy road => distractions
acclerometer variance => irresponsible driving
smartphone sensors
fisheye camera
webcam eye-tracking
in-car/wheelchair sensing

Fig. 12.1 The figure shows the overall architecture for the sensing and analysis infrastructure

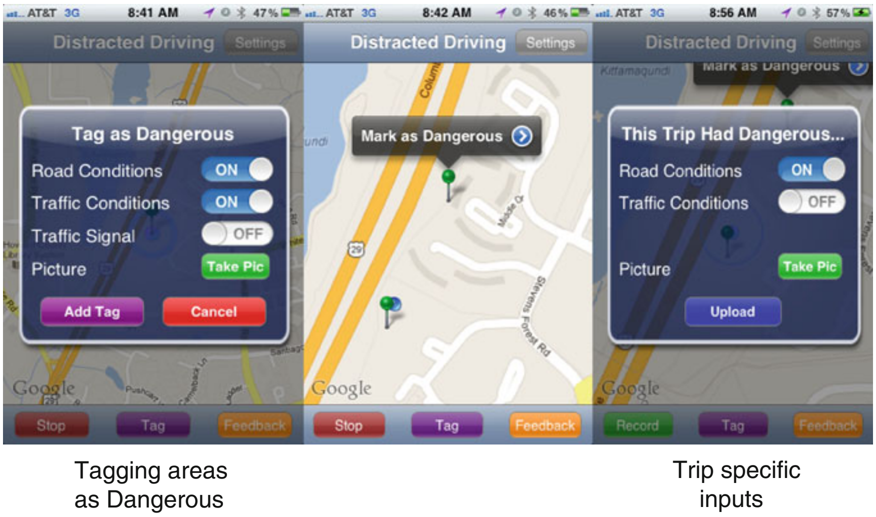
sources that are non-trivial to collect using deployed sensors. These include road construction and road accidents that block road segments. We propose to use human crowds through a smartphone application (e.g., Fig. 12.1) to capture these inputs.

The overall architecture of our data collection infrastructure is illustrated in Fig. 12.1. Our data collection methodology comprises of a two fold approach: (1) data collected from cars using on-board cameras and other sensors; and (3) in-vivo smartphone surveys, crowd-sourced data and structured interviews.

In-vivo smartphone surveys use push notification to gather contextual information from drivers. Figure 12.2 illustrates an example from an iPhone application that we have deployed on 10 subjects in Arkansas and Maryland to collect accelerometer, location, and image data while driving. At the end of the trip (detected automatically by the system through speed changes), the user is prompted to provide feedback on driving and road conditions during the trip. To complement the above two data collection methods, *offline structured interviews* would include questionnaires on perceived driving risks as well as environmental and technological factors that cause human distraction. Our multi-modal data collection will help derive underlying causes that influence dangerous driving. Below we describe the data collection sensors and a rule mining technique to infer causes of distracted and dangerous driving.

12.2.1 Sensor Modalities

In this section, we introduce the novel elements of our sensor hardware that would be deployed in cars. One of our primary design goals is non-intrusiveness—if cars or wheelchairs are heavily instrumented with sensors, users would eschew our


Tag as Dangerous

Road Conditions ON

Traffic Conditions ON

Traffic Signal OFF

Picture

This Trip Had Dangerous...

Road Conditions ON

Traffic Conditions OFF

Picture

Fig. 12.2 Screenshots from the iPhone application deployed to 10 car drivers. The application collects accelerometer, ambient sound, and location data. It also collects crowdsourced data on dangerous traffic and road conditions

system. Therefore, we plan to leverage sensors resident on smartphones, namely accelerometers (for driving patterns), microphones (for ambient sound signature), and GPS units (for location and speed). We will use location based services to extract rich information on traffic, weather, and road parameters. The smartphone software would collect information on incoming and outgoing calls, and text messages when the user is driving. In addition to smartphone sensors, we propose to use two custom designed sensor systems: (a) a FPGA-based multi-sensor camera with a custom designed 360° fisheye view lens to capture fisheye images of surroundings; and (b) a webcam for tracking the driver's eye movement. The camera allows analysis of video and image data on the FPGA board, minimizing the amount of data transferred to the backend using a cellular modem (attached to the module). Images or videos are captured using an image sensor module on an interface board that houses a set of CCD and CMOS sensors. The camera can capture videos at different resolutions. Our feature extraction and analysis algorithm is described in §12.3. The eye tracking software runs on a FitPC [40] module—eye orientation and whether the driver is focussed on the road is used as groundtruth to infer whether the driver is distracted. Groundtruth on dangerous driving conditions is collected using a human crowd (using the smartphone application illustrated in Fig. 12.1). Figure 12.3 illustrates variance in sound collected using a smartphone microphone.

12.2.2 Rule Inferencing

Inferring rules that associate causes to dangerous and irresponsible driving, and driving in a dangerous zone, is key to designing systems that can alert end-users

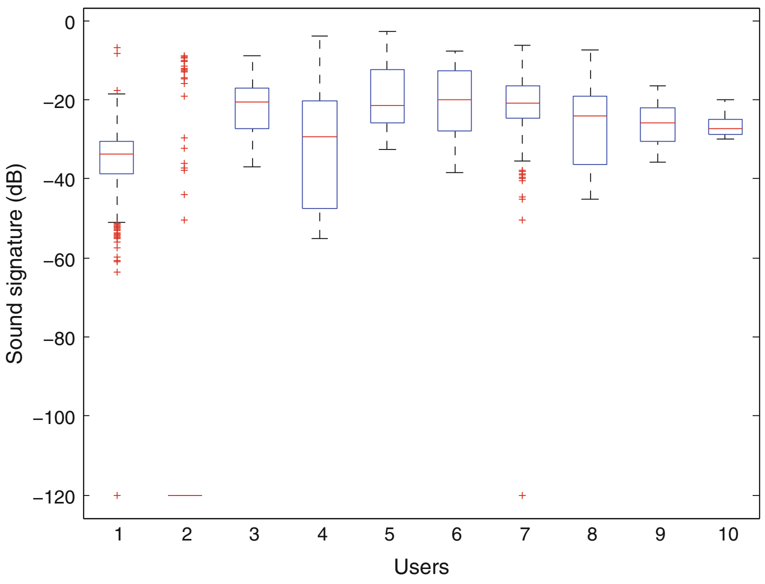


Fig. 12.3 Variance in sound observed from data collected using a smartphone application. Sound values below −70 dB is silence and above 0 dB is loud

or modify driving behavior. Given the amount of data that will be collected from our measurement analysis, automated mining-based analytic techniques must be designed to infer these rules. However, deriving underlying rules from unlabeled sensor data in a driving context faces unique challenges. To understand why, consider the following rule. Rule A associates high traffic congestion and icy roads to dangerous driving conditions. The question whether this rule is statistically relevant requires quantifying its significance. The most common measure of significance is co-occurrence. Although icy roads might not be a frequent occurrence, it is strongly associated with accidents. Hence, frequency of co-occurrence is an incorrect metric to derive rule significance in a driving scenario. The second associated research challenge is to account for hidden parameters that can not be captured using sensor data—for example, cognitive distraction. To address these challenges, we propose an automated rule mining technique (based on network trace mining [33]) augmented with structured interviews. Our rule mining algorithm uses time annotated data from the sensors. This includes features detected by the FPGA-based camera, eye orientation from the eye tracking system, accelerometer data, data on sound ambience from smartphone microphones, data from the human crowd on dangerous driving conditions and road constructions, phone calls and text messaging, driving speed, road metrics (stop signs and traffic lights), and weather conditions (icy roads, wind storms) collected from web services. The rules are defined by two variable sets X and Y. Y indicates whether a driver is distracted or is driving in a dangerous condition. For example, assume that $Y_1 \in \mathbf{Y}$ corresponds to distraction. Y_1 is `true` if the eye tracker detects that the driver is not focussing on the road or does not have his hands on the steering wheel. The variables in **X** correspond to sensor values that cause driving distraction or increase susceptibility to accidents. For example, $X_i \in \mathbf{X}$ could be variance in acceleration, pedestrians detected around a car, traffic congestion, and weather conditions. The rule mining engine derives rules of the form $X_1 \wedge X_2 \ldots \wedge X_n \Rightarrow Y_1$. For instance, X_1 could be high traffic congestion, X_2 could indicate icy roads, X_n could correspond to steep road curvatures, and Y_1 is true, indicating high susceptibility to accidents. To derive these rules, the analyzed data is divided into time windows. Each variable in **X** and **Y** have binary states—the states are derived using a pre-processing technique described in §12.3. For example, traffic congestion can be divided into two states—high traffic and low traffic. The number of traffic lights in a road segment can be `high` or `low` based on a threshold. A rule is generated if variables in **X** and **Y** co-occur in a time window. Similar to deriving significance of mined rules in network traces [33], we use the forward element of JMeasure ($JMeasure(X \Rightarrow Y) = I(Y; X = 1) = P(X \wedge Y) \cdot \log(\frac{P(Y|X)}{P(Y)})$) to quantify rule significance. JMeasure calculates the reduction of entropy of Y when X occurs. Because the metric is normalized by the occurrence of Y, it accounts for associations that occur infrequently. Rules that have low JMeasure are discarded as insignificant.

The above rule mining technique accounts for correlation and time dependence but it does not correct for hidden parameters that might influence driving. Additionally, our data will be collected from power wheelchairs, cars, and subjective interviews. Hence, it is important to design techniques that combine inferences drawn from each source. To address the above challenge, we propose a Quasi-experimental design (QED) framework. We illustrate our QED through an example rule $X_1 \Rightarrow Y_1$, where X_1 corresponds to a large number of pedestrians detected by our camera, and Y_1 is driver distraction. Hence, the rule implies that a large number of pedestrians around a vehicle causes driver distraction. Now, the rule may have different JMeasure coefficients for in car experiments (experimental group) and wheelchair experiments (control group). If the JMeasure trends match for the experimental and control group, we conclude that pedestrians causes distraction. If they conflict, we propose to use structured interviews on all subjects to break ties and validate the effect of hidden parameters like cognitive distraction. The structured interview questions would be formulated to resolve the conflict in mined inferences. For example a plausible question in the above scenario might be "Do crowded spaces distract drivers?".

12.3 Automated Detection of Dangerous Driving

The causal analysis technique discussed in the previous section outputs a set of rules of the form $X \Rightarrow Y$, where X are sensor values and Y corresponds to a state of distracted or dangerous driving. For example, X could be high variance in accelerometer readings during a time window or a large number of pedestrians

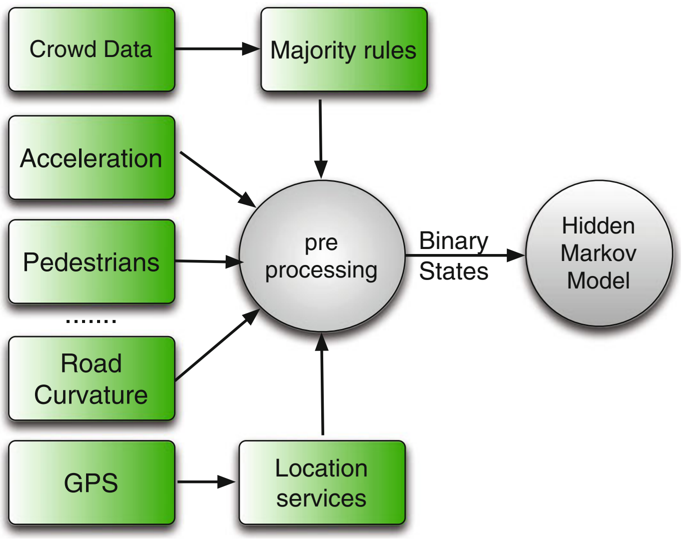


Fig. 12.4 Pre-processing and sensor fusion architecture at the backend

around a car. Our next goal is to use these input parameters (X) to quantify a driver's susceptibility to accidents, and alert the driver in real time Fig. 12.4.

It is critical, that these recommendations or alerts are computed in real time since alerting a driver of a dangerous driving zone five minutes after an accident is futile. This is equally important for vehicular drivers. To facilitate low latency and accurate recommendations, we propose two novel research contributions. First, we present techniques to push sensor computation close to the edge device (i.e., devices inside a car). Such techniques can mitigate transferring large amounts of data to a backend server—a potential latency bottleneck. Secondly, we propose a technique to fuse data from sensors and human crowds to accurately infer whether the driver is driving in a dangerous zone or is distracted. Our work builds on previous work on vehicular crowdsourcing [41–44], our work on image analysis [45–49], and related research on sensor fusion [50–52].

12.3.1 Image and Sensor Analysis

Data analysis to determine variance in acceleration (using smartphone accelerometers) [53, 54], ambient sound signature using microphones [55], or eye tracking using webcameras [56, 57] are well studied areas. Hence, we use standard techniques for eye tracking [58–60], and on-phone computation of acceleration and microphone data [55]. Our contributions however, lie in pushing computation close to the sensor, and the use of a minimal set of sensors to collect a variety of data related to driving.

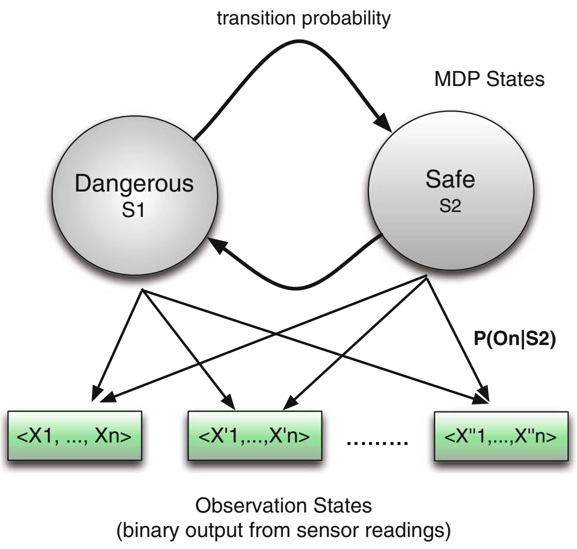


Fig. 12.5 The Hidden Markov Model for predicting whether users are susceptible to accidents

For instance, we use a novel FPGA-enabled camera with a fisheye lens to capture 360° images on vehicular or wheelchair surroundings. The 360° fisheye lens ensures that a single image can capture front, rear, and side views of a car. The camera images and videos are used to extract features corresponding to humans (in and around the car), road geometries, and roadside advertisements.

Detecting humans, road geometries, or advertisements from a single video source in a moving vehicle is non-trivial. It requires accounting for variable lighting and environmental conditions, and vehicular speeds. Moreover, since the goal is to implement a low latency and accurate feature detection algorithm locally at the sensor (FPGA-fabric), it is key that the algorithm exhibits high data parallelism. To address this challenge, we use a Gaussian mixture model (GMM) based segmentation scheme for determining image features. The training phase calculates feature-specific parameters for the Gaussian. For example, consider car occupants as an extracted feature. We propose to express this feature by analyzing the color histogram of the clothes worn by the car's occupants. The frames are segmented using our designed contour detection algorithm [61]. An extracted r-bin Hue histogram from each segmented video frame is trained under different lighting and environmental conditions. Building a color histogram allows massive parallelism where hardware co-processors can calculate the Hue values for disjoint blocks of pixels. An Expectation-Maximization (EM) algorithm is then used to calculate the Gaussian distribution parameters that fit the histogram for this feature. The resulting Gaussian can be represented as a r-dimensional mean vector μ_p, and a $r \times r$ covariance matrix Σ_p. In the online system, we use the same segmentation approach based on contour detection [61] to compute the edges in the image, and the corresponding closed spaces. The segmentation approach also exhibits high data parallelism. The ratio of the length to height of the segmented feature, and the distance of its Gaussian from the trained Gaussian differentiates a human from roads and advertisements. The road geometry (high curvature or low curvature) is determined by applying the concept of moments on the image segmented by the GMM algorithm. The normalized second order moments of the segmented image describe the orientation of the object. The second order moment can be calculated in parallel using hardware co-processors executing on blocks of pixels. The orientation of the road can be used to determine its curvature. Similarly, the orientation of the driver can be used to interpret distracted or drunk driving. The segmentation and orientation computation are implemented as hardware co-processors built into the FPGA fabric. Migration of the proposed algorithm into the FPGA fabric minimizes the amount of data that must be transferred over the cellular model, reduces the processing latency, and mitigates privacy issues with storing image data at the backend.

12.3.2 Multi-sensor Fusion

The susceptibility of a driver to an accident is predicated on two dependent factors—whether the driver is distracted (e.g., drunk driving, road side advertisements, pedestrians) *and* whether he is driving in an environment that is dangerous or increases distraction (e.g., poor weather conditions, crowded roads, high ambient sound). In our system, the occurrence of any one of these two factors triggers an alert to the driver. To infer whether a driver is distracted, we propose to use eye tracking and image analysis from the fisheye camera. The eye-tracker determines whether the driving is focussing on the road and the camera images can determine the orientation of the driver. However, inferring whether the user is driving in a dangerous zone is more involved. It requires fusing data from multiple and disparate data sources such as smartphone sensors, camera, a human crowd, and location services (weather, traffic lights, stop signs). While a subset of the data is gathered using machines (sensors), the remaining data sources use human intelligence (crowd sourcing). To combine orthogonal data modalities and ensure tractability, we propose a data pre-processing scheme and a Hidden Markov Model based supervised learning approach. The goal is to accurately determine a driver's susceptibility to accidents while minimizing human and sensor bias. The pre-processor resides at a backend server and performs a two-fold task. First, it processes data from the human crowd collected using our iPhone application. The data includes road and traffic conditions. The pre-processor uses a *majority rule* on the crowd sourced input to determine whether a location is dangerous to drive or the road conditions are unfriendly. Secondly, the pre-processor takes the input from the sensors and uses thresholding to convert the sensor values into binary states. For instance, traffic congestion is considered either `high` or `low` and the weather condition is either `dangerous` or `safe`. Similarly, the number of pedestrian observed by the camera is `high` or `low`. This use of binary states help maintain a tractable number of observable states for the Hidden Markov Model, described below.

Our core contribution to sensor fusion is a simple yet robust HMM [62] technique that uses per-user sensor data and inputs from the human crowd to determine whether the user in a vehicle or wheelchair is susceptible to accidents. The MDP and observation states of the Hidden Markov Model is illustrated in Fig. 12.5. For example, $< X_1, X_2, \ldots, X_n >$ can represent a state observation O, where $X_1 = \texttt{high}$ traffic congestion (source: human crowd), $X_2 = \texttt{dangerous}$ weather conditions (source: location services), and $X_3 = \texttt{large}$ number of pedestrians. Our HMM has two MDP states, `dangerous` and `safe`. The transition and initial steady state probabilities for the states are determined using training data collected by deploying our iPhone application (illustrated in Fig. 12.1)—user tags trips as dangerous or safe for driving. Our system will calculate the stationary probabilities from data over multiple trips. The observation probabilities $P(O|S_1)$, for a state $O =< X_1, \ldots, X_n >$ is calculated as the sum of the JMeasure coefficients for the rules of the form $X_1 \wedge X_1 \wedge \ldots \wedge X_n \Rightarrow Y$, where Y is driver distraction or dangerous driving state, and X_i is the pre-processed binary output for sensor i. We solve the HMM decoding problem where the input is a set of observation vectors during a time window and the HMM determines the sequence of states that maximize the probability of the observations. If the final state is `dangerous`, we use techniques in §12.3.5 to alert drivers to modify driving behavior.

12.3.3 *Optimizations*

In-vivo data fusion using the HMM approach can incur latencies due to unpredictable network conditions and processing delays at the backend. It is important, however, that alerts are disseminated before a predicted accident. To address this issue, we plan to use prefetching and proactive techniques. Humans are creatures of habit [63] and they follow similar routes to similar destinations most of the time. Therefore, we will augment our system with mobility prediction [63, 64] to prefetch and cache data on traffic, weather, and road conditions, to pre-calculate the susceptibility to accidents in a geographic region. We can then combine the prefetched data with data from the in-car camera, eye tracker, and smartphone sensors to trigger alerts proactively.

12.3.4 *Minimizing Sensor Bias*

Data from image sensors, accelerometers, microphones, and web cameras can be noisy. The use of human crowds and smartphone surveys (shows in Fig. 12.1) provide an orthogonal data modality that can help mask sensor inaccuracies in the HMM model. Additionally, we plan to supplement our sensor analysis with structured interviews that use neutral and open ended questions on *whether* the subjects thought they were distracted or were driving in a dangerous zone during a trip. The subjective interviews will help validate the inferences made by our sensor fusion framework. They will also provide groundtruth to train the probabilities associated with the HMM framework.

12.3.5 Driving Behavior Modification

Persuasive technology has been used in several application domains such as sustainability and healthcare [65–67]. It has been shown that intelligent visualization and recommendations can, in many cases, persuade individuals to adopt environmentally friendly approaches [68–71]. In the safe driving domain, however, it is an open research challenge to design minimally intrusive techniques that can incentivize drivers (especially teenagers) to adopt safe driving practices. In this project, we propose to explore the following technique to inculcate behavioral modifications in drivers.

First, our sensor fusion framework, on detecting high susceptibility to accidents, will use Minimal Attention User Interfaces (MAUI) [72, 73] to push notifications and recommendations to a user's handheld device. This could include audio alerts, earcons [74], haptic solutions that leverage tactile feedback such as phone vibrations [75, 76], and visual cues such as danger icons on the smartphone screen. The phones can be paired with screens or car dashboards and hence the cues will appear on large screens in the car. We have also designed techniques that can re-route incoming phone calls when the system determines that the driver is susceptible to accidents. In our preliminary implementation, the incoming call is blocked, and the phone number is uploaded to a web service that uses `Skype` and an automated voicemail to make a callback to the callee, informing him that the user is driving in a dangerous zone. The closed loop system ensures that proper feedback is sent to the callee — the callee could be a business partner or a worried parent. The notification-based system has a dual advantage. Since it is solely an alert, it is up to the driver to adopt it, and hence it is minimally intrusive. Additionally, using the sensor data collected, we can infer whether the driver adopted the recommendation. Therefore, we can evaluate the efficacy of our data analytics. We can augment the above alerts with information on *why* the system thinks that the driver is susceptible to accidents. For example, in addition to a danger icon, the system will display icons on road construction, traffic congestion, pedestrians, or stop signs. The user can then verify that the alert is not a false positive, and can react accordingly.

12.4 Conclusion

The book chapter presents a system that combines on-board camera, web camera, sensors on smartphones, and crowdsourcing techniques to detect distracted driving, driving in dangerous zones, and dangerous driving. The system alerts the driver when it detects the above problems. The system uses a combination of image analysis and multi-sensor fusion to accurately infer that the driver is distracted or driving dangerously.

References

1. Brace C, Young K, Regan M (2007) Analysis of the literature:the use of mobile phones while driving. In Vagverket
2. Rakauska M, Gugerty L, Ward N (2011) Effects of naturalistic cell phone conversations on driving performance. J Saf Res
3. David Strayer, Drews Frank (2006) Drivers on Cell Phone Are as Bad as Drunks. In U News Center University of Utah, Salt Lake
4. Article (2000) Cellular phones and driving: weighing the risks and benefits. In: Risk in perspective. Harvard Center for Risk Analysis
5. Carol Cruzan Morton Why cell phone ban's won't work. http://news.sciencemag.org/sciencenow/2012/08/why-cell-phone-bans-dont-work.html?ref=hp
6. Startup. TextBlocker. http://txtblocker.com/
7. Startup. izup. http://www.getizup.com/
8. Startup. Zoomsafer. http://zoomsafer.com/
9. Startup. Driversafely. http://www.drivesafe.ly/
10. Lindqvist J, Hong J (2011) Undistracted driving: a mobile phone that doesnt distract. In: HotMobile
11. Insurance. Progressive snapshot. http://www.progressive.com/auto/snapshot.aspx
12. News Article. Ford driving safety. http://corporate.ford.com/innovation/innovation-detail/pr-cars-talking-to-traffic-lights-and-34198
13. Salvucci DD, Bogunovich P (2010) Multitasking and monotasking: the effects of mental workload on deferred task interruptions. In: SIGCHI conference on human factors in computing systems
14. Bogunovich P, Salvucci DD (2011) The effects of time constraints on user behavior for deferrable interruptions. In: SIGCHI conference on human factors in computing systems
15. Gould SJJ, Brumby DP, Cox AL, Gonzlez VM, Salvucci DD, Taatgen NA (2012) Multitasking and interruptions: a SIG on bridging the gap between research on the micro and macro worlds. In: SIGCHI conference on human factors in, computing systems
16. Lee J, Lee JD, Salvucci DD (2012) Evaluating the distraction potential of connected vehicles. In: International conference on automotive user interfaces and interactive vehicular applications (AutoUI)
17. Kushleyeva Y, Salvucci DD, Lee FJ (2005) Deciding when to switch tasks in time-critical multitasking, In: Cognitive systems research
18. Salvucci DD, Mandalia HM, Kuge N, Yamamura T (2007) Lane-change detection using a computational driver model. In: Human factors
19. Salvucci DD (2006) Modeling driver behavior in a cognitive architecture. In: Human factors
20. Pradhan AK, Divekar G, Masserang K, Romoser M, Zafian T, Blomberg RD, Thomas FD, Reagan I, Knodler, Pollatsek AM, Fisher DL The effects of focused attention training (FOCAL) on the duration of novice drivers' glances inside the vehicle. In: Ergonomics
21. Rothenberg H, Knodler MA , Fisher D (2010) An analysis of motorcycle crash causation, In: Accident analysis and prevention
22. Garay-Vega L, Pradhan AK, Weinberg G, Schmidt-Nielsen B, Harsham B, Shen Y, Divekar G, Romoser M, Knodler M, Fisher DL (2010) Evaluation of different voice and touch interfaces to in-vehicle music retrieval systems. In: Accident analysis & prevention

23. Pradhan AK, Pollatsek A, Knodler M, Fisher DL (2009) Can younger drivers be trained to scan for information that will reduce their risk in roadway traffic scenarios that are hard to identify as hazardous? In: Ergonomics
24. Gunzelmann G, Moore LR, Salvucci DD, Gluck KA (2011) Sleep loss and driver performance: quantitative predictions with zero free parameters. In: Cognitive systems research
25. Chan E, Pradhan AK, Knodler MA, A. Pollatsek A, Fisher DL (2010) Evaluation on a driving simulator of the effect on drivers' eye behaviors from distractions inside and outside the vehicle. In: Transportation research
26. Kim S, Dey AK (2009) Cognitive mapping aid for senior driver's navigation: a windshield-based 2.5D display. In: CHI
27. Steinfeld A, Rao SL, Tran A, Zimmerman J, Tomasic A (2012) Co-producing value through public transit information services. In: International conference on human side of service, engineering
28. Steinfeld A, Zimmerman J, Tomasic A, Yoo A, Aziz R (2012) Mobile transit rider Information via Universal design and crowdsourcing. In: Transportation research record—J Transport Res Board
29. Vzquez M, Steinfeld A (2010) An assisted photography method for street ccenes. In: IEEE workshop on applications of computer vision (WACV)
30. Beyene N, Cooper R, Steinfeld A (2010) Transportation independence and community participation among older Americans in Pittsburgh, PA. In: Proceedings of the rehabilitation engineering and assistive technology society of North America conference
31. Beyene N, Lane A, Seelman K, Songer T, Cooper R, Steinfeld A (2011) Navisection: a novel method joining naturalistic driving data collection with expert witness event-logging for enhanced assessment of driver safety. In: 3rd international conference on road safety and simulation
32. Zimmerman J, Tomasic A, Garrod C, Yoo D, Hiruncharoenvate C, Aziz R, Thirunevgadam N, Huang Y, Steinfeld A (2011) Field trial of Tiramisu: crowd-sourcing bus arrival times to spur co-design. In: Conference on human factors in computing systems (CHI)
33. Kandula S, Chandra R, Katabi D (2008) What's going on? Learning communication rules in edge networks, In: ACM Sigcomm
34. Smyth P, Goodman RM (1991) Knowledge discovery in databases. In: MIT Press, Cambridge
35. Vempala S, Kannan R, Vetta A (2000) On clusterings good. Bad and spectral, In: In FOCS
36. Dicianno B, Cooper R, Coltallaro J (2010) Joystick control for powered mobility: current state of technology and future directions. In: Physical medicine and rehabilitation clinics of North America
37. Salatin B, Rice I, Teodorsk IE, Ding D, Cooper R (2010) A survey of outdoor electric powered wheelchair driving. In Proceedings of the rehabilitation engineering and assistive technology society of North America Conference
38. Salatin B, Wang H, Grindle GG, Ding D, Cooper RA (2009) Electric powered wheelchair driving strategies over difficult outdoor terrain: a focus group study. In: Rehabilitation engineering and assistive technology society of North America Conference
39. Simulators. Expensive driving simulators. http://www.bornrich.com/entry/12-high-tech-racing-simulators-for-ultimate-speed-thrills/
40. http://www.fit-pc.com/web/. fitPC2
41. Musthag M, Raij A, Ganesan D, Kumar S, Shiffman S (2011) Exploring micro-incentive strategies for participant compensation in high-burden studies. In: Ubicomp, Nov 2011
42. Yan T, Kumar V, Ganesan D (2010) Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In: Proceedings of the 8th annual international conference on mobile systems, applications and services
43. Biagioni J, Gerlich T, Merrifield T, Eriksson J (2011) Easytracker: automatic transit tracking, mapping, and arrival time prediction using smartphones. In: Sensys
44. Arvind T, James B, Tomas G, Eriksson J(2010) Cooperative transit tracking using gps-enabled smartphones, In: Sensys

45. Zarezadeh AA, Bobda C (2011) Probabilistic framework for person tracking on embedded distributed smart cameras. In: ICDSC, pp 1–6
46. Djakou Chati H, Muhlbauer F, Braun T, Bobda C, Berns K (2007) Hardware/software co-design of a key point detector on fpga. In: FCCM '07 proceedings of the 15th annual IEEE symposium on field-programmable custom computing machines, IEEE computer society. Washington, DC, USA, pp 355–356
47. Chati HD, Mühlbauer F, Braun T, Bobda C, Berns K (2007) Sopc architecture for a key point detector. In: FPL, pp 710–713
48. Muhlbauer F, Bobda C (2006) Design and implementation of an object tracker on a reconfigurable system on chip. In: RSP '06: proceedings of the 17th IEEE international workshop on rapid system prototyping. Washington, DC, USA IEEE computer society, p 230
49. Christophe B, Zarezadeha Ali A, Felix M, Robert H, Kevin C (2010) Reconfigurable architecture for distributed smart cameras. In: ERSA. CSREA Press
50. Brooks RR, Iyengar SS (1998) Multi-sensor fusion: fundamentals and applications with software. Prentice-Hall Inc, Upper Saddle River, NJ
51. Sun S, Meng X, Ji L, Wu J, Wong WC (2010) Adaptive sensor data fusion in motion capture. In: Information fusion (FUSION), 2010 13th conference on, pp 1–8, July 2010
52. Munz M, Ma andhlisch M, Dietmayer K (2010) Generic centralized multi sensor data fusion based on probabilistic sensor and environment models for driver assistance systems. Intell Transport Syst Mag, IEEE 2(1):6–17
53. Agrawal S, Constandache I, Gaonkar S, Roy Choudhury R, Caves K, DeRuyter F (2011) Using mobile phones to write in air. In: 9th international conference on mobile systems, applications, and services, mobiSys '11, New York, NY, USA, 2011. ACM, pp 15–28
54. Lung Chu H, Raman V, Shen J, Choudhury RR, Kansal A, Bahl V (2011) Poster: you driving? talk to you later. In: 9th international conference on mobile systems, applications, and services, MobiSys '11, New York, NY, USA, ACM, pp 397–398
55. Azizyan M, Constandache I, Choudhury RR (2009) Surroundsense: mobile phone localization via ambience fingerprinting. In 15th annual international conference on mobile computing and networking, MobiCom '09, New York, NY, USA, ACM, pp 261–272
56. Hansen DW, MacKay DJC, Hansen JP, Nielsen M (2004) Eye tracking off the shelf. In Proceedings of the 2004 symposium on eye tracking research & applications, ETRA '04, New York, NY, USA, ACM, pp 58–58
57. Chau M, Betke M (2012) Real time eye tracking and blink detection with usb cameras. In: Computer science technical report BU
58. Duchowski AT (2007) Eye tracking methodology, theory and practice. In: Springer, Berlin
59. Holzman Philip S, Proctor Leonard R, Hughes Dominic W (1973) Eye-tracking patterns in schizophrenia. Science 181(4095):179–181
60. Granka LA, Joachims T, Gay G (2004) Eye-tracking analysis of user behavior in www search. In Proceedings of the 27th annual international ACM SIGIR conference on research and development in information retrieval, SIGIR '04, New York, NY, USA, ACM, pp 478–479
61. Hartmann R, Al Machot F, Mahr P, Bobda C (2010) Camera-based system for tracking and position estimation of humans. In DASIP, pp 62–67
62. Sean R (1996) Eddy Hidden markov models. Current Opin Struct Biol 6(3):361–365
63. Nicholson AJ, Noble BD (2008) Breadcrumbs: forecasting mobile connectivity. In Proceedings of the 14th ACM international conference on mobile computing and networking, MobiCom '08, New York, NY, USA, ACM, pp 46–57
64. Constandache I, Choudhury RR, Rhee I (2010) Towards mobile phone localization without war-driving. In INFOCOM, 2010 proceedings IEEE, March 2010, pp 1–9
65. Intille SS (2004) A new research challenge: persuasive technology to motivate healthy aging. Inf Technol Biomed, IEEE Trans 8(3):235–237
66. Consolvo S, Klasnja P, McDonald DW, Avrahami D, Froehlich J, LeGrand L, Libby R, Mosher K, Landay JA (2008) Flowers or a robot army?: encouraging awareness & activity with personal, mobile displays. In Proceedings of the 10th international conference on ubiquitous computing, UbiComp '08, New York, NY, USA, ACM, pp 54–63

67. Klasnja P, Consolvo S, Pratt W (2011) How to evaluate technologies for health behavior change in hci research. In: Proceedings of the 2011 annual conference on human factors in computing systems, CHI ’11, New York, NY, USA, ACM, pp 3063–3072
68. Froehlich J, Findlater L, Landay J (2010) The design of eco-feedback technology. In: Proceedings of the 28th international conference on human factors in computing systems, CHI ’10, New York, NY, USA, ACM, pp1999–2008
69. Gajos Krzysztof Z, Hurst Amy, Findlater Leah (2012) Personalized dynamic accessibility. Interactions 19(2):69–73
70. Froehlich JE, Larson E, Campbell T, Haggerty C, Fogarty J, Patel SN (2009) Hydrosense: infrastructure-mediated single-point sensing of whole-home water activity. In: Proceedings of the 11th international conference on Ubiquitous computing, Ubicomp ’09. ACM, New York, USA, pp 235–244
71. Cohn G, Gupta S, Froehlich J, Larson E, Patel SN (2010) Gassense: appliance-level, single-point sensing of gas activity in the home. In: Proceedings of the 8th international conference on pervasive computing, Pervasive’10. Springer, Berlin, pp 265–282
72. Pascoe Jason, Ryan Nick, Morse David (2000) Using while moving: Hci issues in fieldwork environments. ACM Trans Comput Hum Interact 7(3):417–437
73. Holland Simon, Morse David R, Gedenryd Henrik (2002) Audiogps: spatial audio navigation with a minimal attention interface. Pers Ubiquit Comput 6:253–259. doi:10.1007/s007790200025
74. Blattner MM, Sumikawa DA, Greenberg RM (1989) Earcons and icons: their structure and common design principles. Hum Comput Inter 4(1):11–44
75. Jeon M, Nazneen N, Akanser O, Ayala-Acevedo A, Walker B (2012) "Listen2droom": helping blind individuals understand room layouts. In: Proceedings of the 2012 ACM annual conference extended abstracts on human factors in computing systems extended abstracts, CHI EA ’12. ACM, New York, USA, pp 1577–1582
76. Nees Michael A, Walker Bruce N (2011) Auditory displays for in-vehicle technologies. Reviews of Human Factors and Ergonomics 7(1):58–99

Index