Distributed Smart Cameras and Distributed Computer Vision

Marilyn Wolf and Jason Schlessman

Abstract Distributed smart cameras are multiple-camera systems that perform computer vision tasks using distributed algorithms. Distributed algorithms scale better to large networks of cameras than do centralized algorithms. However, new approaches are required to many computer vision tasks in order to create efficient distributed algorithms. This chapter motivates the need for distributed computer vision, surveys background material in traditional computer vision, and describes several distributed computer vision algorithms for calibration, tracking, and gesture recognition.

1 Introduction

Distributed smart cameras have emerged as an important category of distributed sensor and signal processing systems. Distributed sensors in other media have been important for quite some time, but recent advances have made the deployment of large camera systems feasible. The unique properties of imaging add new classes of problems that are not apparent in unidimensional and low-rate sensors. **Physically distributed cameras** have been used in computer vision for quite some time to handle two problems: **occlusion** and **pixels-on-target**. Cameras at different locations expose and occlude different parts of the scene. Their imagery can be combined to create a more complete model of the scene. Pixels-on-target refers to the resolution with which a part of the scene is captured, which in most applications is primarily limited by sensor resolution and not by optics. Wide-angle lenses cover a large area but at low resolution for any part of the scene. Imagery from multiple cameras can be combined to provide both extended coverage

M. Wolf (✉)
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA
e-mail: wolf@ece.gatech.edu

J. Schlessman
Department of Electrical Engineering, Princeton University, Princeton, NJ, USA

S. S. Bhattacharyya et al. (eds.), *Handbook of Signal Processing Systems*,
https://doi.org/10.1007/978-3-319-91734-4_10

and adequate resolution. Distributed smart cameras combine physically distributed cameras and distributed algorithms. Early approaches to distributed-camera-based computer vision used server-based, centralized algorithms. While such algorithms are often easier to conceive and implement, they do not scale well. Properly-designed distributed algorithms scale to handle much larger camera networks. VLSI technology has aided both the image-gathering and computational abilities of distributed smart camera systems. Moore's Law has progressed to the point where very powerful multiprocessors can be put on a single chip at very low cost [32]. The same technology has also provided cheap and powerful image sensors, particularly in the case of CMOS image sensors [33]. Distributed smart cameras have been used for a variety of applications, including tracking, gesture recognition, and target identification. Networks of several hundred cameras have been tested. Over time, we should expect to see much larger networks both tested and deployed. Surveillance is one application that comes to mind. While surveillance and security are a large application—analysts estimate that 25 million security cameras are installed in the United States—that industry moves at a relatively slow pace. Health care, traffic analysis, and entertainment are other important applications of distributed smart cameras. After starting in the mid-1990s, research on distributed smart cameras has progressed rapidly.

We start with a review of some techniques from computer vision that were not specifically developed for distributed systems but have been used as components in distributed systems. Section 3 reviews early research in distributed smart cameras. Section 4 considers the types of challenges created by distributed smart cameras. We next consider calibration of cameras in Sect. 5, followed by algorithms for tracking in Sect. 6 and gesture recognition in Sect. 7. Section 8 discusses computing platforms suitable for real-time distributed computer vision.

2 Approaches to Computer Vision

Several algorithms that used in traditional computer vision problems such as tracking also play important roles as components in distributed computer vision systems. In this section we briefly review some of those algorithms. Tracking refers to the target or object of interest as **foreground** and non-interesting objects as **background** (even though this usage is at variance with the terminology of theater). Many tracking algorithms assume that the background is relatively static and use a separate step, known as **background elimination** or **background subtraction**, to eliminate backgrounds. The simplest background subtraction algorithm simply compares each pixel in the current frame to the corresponding pixel in a reference frame that does not include any targets. If the current-frame pixel is equal the reference-frame pixel, then it is marked as background. This algorithm is computationally cheap but not very accurate. Even very small amounts of movement in the background can cause erroneous classification; the classic example of extraneous motion is the blowing of tree leaves in the wind. The mixture-of-Gaussians approach [14] provides much

more results. Mixture-of-Gaussian systems also use multiple models so more than one candidate model can be kept alive at a time. The algorithm proceeds in three steps: compare Gaussians of each model to find matches; update Gaussian mean and variance; update weights. Given a pixel value as a vector of luminance (Y) and chrominance (C_b, C_r) information, $X \in (Y, Cb, Cr)$ and $\alpha_x = \sqrt{\frac{|X-\mu_x|}{\sigma_x}}$, we can compare a current-frame and reference-frame pixels using a threshold T:

$$\left(\frac{\alpha_Y}{a_Y}\right)^2 + \left(\frac{\alpha_{Cb}}{a_{Cb}}\right)^2 + \left(\frac{\alpha_{Cr}}{a_{Cr}}\right)^2 < T \tag{1}$$

Matching weights are updated as follows, where n is the number of frames over which this Gaussian distribution has been active:

$$N = \begin{cases} n, & n < N_{max} \\ N_{max}, & n \geq N_{max} \end{cases} \tag{2}$$

$$\mu_{Xn} = \mu_{Xn-1} + \frac{(X_n - \mu_{Xn-1})}{N} \tag{3}$$

$$\sigma_{Xn} = \sigma_{Xn-1} + \frac{(X_n - \mu_{Xn-1})(X_n - \mu_{Xn}) - \sigma_{Xn-1}}{N} \tag{4}$$

A weight w_m evaluates the system's confidence in that model and pixel position:

$$w_m = \begin{cases} w_{m-1} + \frac{(\rho - w_{m-1})}{m}, & m < M_{max} \\ w_{m-1} + \frac{(\rho - w_{m-1})}{M_{max}}, & m \geq M_{max} \end{cases} \tag{5}$$

An appearance model is used to compare the appearance of targets in different frames; appearance models are useful both in a single-camera system to ensure tracking continuity or between cameras to compare views. Appearance models may make use of shape and/or color. Bounding box is a simple shape model, but more complex curve-based models may also be used. A color or luminance histogram is often used to represent detail within the image. One common method for comparing two histograms is the Bhattacharyya distance. More than one appearance model may be necessary since many interesting targets change in both shape and color as a function of the observer's position. The two major approaches to single-camera tracking are Kalman filtering [6] and particle filters. Particle filters [8] use Monte Carlo methods to estimate a probability distribution. Weighted particles represent samples of the hidden states as weighted by Bayesian estimates of probability masses. Coates [10] describes a distributed algorithms for computing particle filters. In their architecture, each sensor node ran one particle filter, with random number generators on each node that were synchronized. They described two approaches to the problem of distributing observations over the network, one parametric and one based on adaptive encoding. Sheng et al. [27] developed two distributed particle filter algorithms: one that exchanges information between cliques (nearby sensors

whose signals tend to be correlated) and another in which cliques compute partial estimates based on local information and forward their estimates to a fusion center.

3 Early Work in Distributed Smart Cameras

The DARPA-sponsored Video Surveillance and Monitoring (VSAM) program was one of the first efforts to develop distributed computer vision systems. A system developed at Carnegie Mellon University [11] developed a cooperative tracking system in which tracking targets were handed off from camera to camera. Each sensor processing unit (SPU) classified targets into categories such as *human* or *vehicle*. At the MIT Media Lab, Mallet and Bove [20] developed a distributed camera network that could hand-off tracking targets in real time. Their camera network consisted of small cameras mounted on tracks in the ceiling of a room. The cameras would move to improve their view of subjects based on information from other cameras as well as their own analysis. Lin et al. [19] developed a distributed system for gesture recognition that fuses data after some image processing using a peer-to-peer protocol. That system will be described in more detail in Sect. 7. The distributed tracker of Bramberger et al. [7] passed off a tracking task from camera to camera as the target moved through the scene. Each camera ran its own tracker. Handoffs were controlled by a peer-to-peer protocol.

4 Challenges

A distributed smart camera is a data fusion system—samples from cameras are captured, features are extracted and combined, and results are classified. There is more than one way to perform these steps, providing a rich design space. We can identify several axes on which the design space of distributed smart cameras can be analyzed:

- How abstract is the data being fused: pixel, small-scale feature, shape, etc.? What methods are used to fuse data?
- What groups/cliques of sensors combine their data? For example, groups may be formed by network connectivity, location, or signal characteristics. How does the group structure evolve as the scene changes?
- How sparse in time and space is the data?

We can also identify some axes based on the underlying technology:

- Number of cameras.
- Heterogeneity of cameras.
- Fixed and/or moving cameras.
- Fields-of-view (overlapping, non-overlapping).

- Server-assisted (directories, etc.) vs. peer-to-peer.
- Data fusion level: pixels (server-based) vs. mid-level vs. high-level

The simplest form of data fusion is to directly combine pixels. Image stitching algorithms [21] can be used to merge several images into a larger image. Such algorithms are more often used for photography—that is, directly-viewed images—than for computer vision. Thanks to the large amount of data involved, some amount of preprocessing is often done before merging data. Features can be extracted using a variety of methods. Scale-invariant feature transform (SIFT) is a widely used feature extraction algorithm. A set of feature vectors is extracted from the image as min/max of the difference-of-Gaussians function to smoothed and resampled images. The features are then clustered to identify corresponding features in the images to be compared. These features can be combined in a variety of ways using distributed algorithms, either tree-based or graph-based. Group structure is an issue that cuts across image processing and distributed algorithms. The natural structure of the problem suggests certain groupings of data that are reflected in the formulas that describe the image processing being performed. Distributed algorithms and networks also encourage grouping. In many cases, the groupings suggested by those two disciplines are complementary but in some cases their recommendations conflict.

Group structure in networks is often based on network distance as illustrated in Fig. 1. Network distance is determined in part in a wireless network by the transmission radius of a node, which determines what other nodes can be reached directly. It is also determined by the path length between nodes in a network. In contrast, group structure for image processing algorithms is determined by field-of-view as illustrated in Fig. 2. When cameras have overlapping fields of view, a target is in general visible to several cameras, These cameras are not always physically adjacent—in fact, to battle occlusion, cameras at widely different angles provide the best coverage. The cameras also may not be close in the network sense. However, these cameras do need to talk to each other. In the case of non-overlapping fields-of-view, as shown in Fig. 3, communication between nodes is determined by the possible or likely paths of targets, which may or may not correspond to network topology. Features from multiple sensors need to be combined at some point. Where

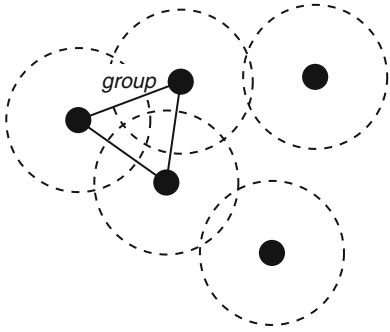

Fig. 1 Groups in wireless networks

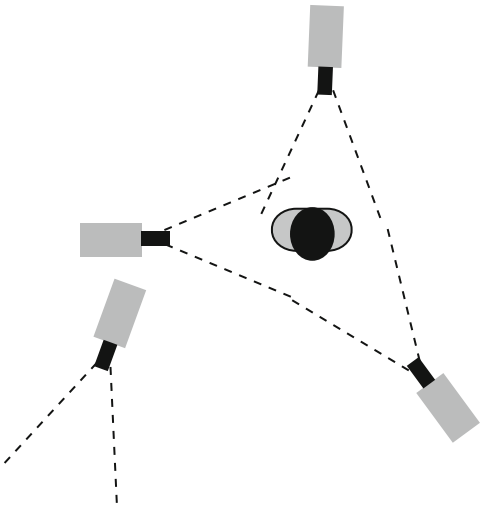

Fig. 2 Groups in overlapping field-of-view image processing

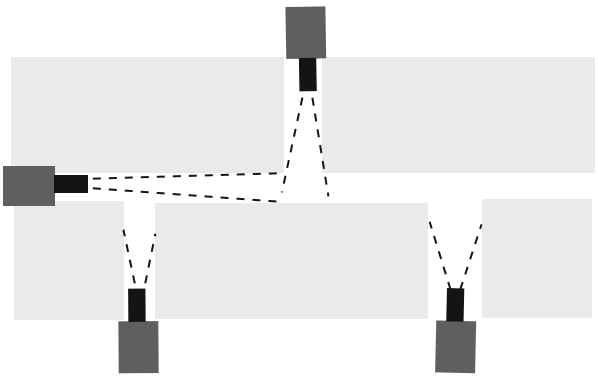

Fig. 3 Groups in non-overlapping cameras

they are combined and how they are combined are both factors. We will use two example systems to illustrate different approaches to feature analysis, fusion, and classification. A separate chapter in this book describes sensor networks in more detail.

5 Camera Calibration

Several aspects of the camera network need to be calibrated:

- **Intrinsic** calibration determines camera parameters such as focal length and color response.

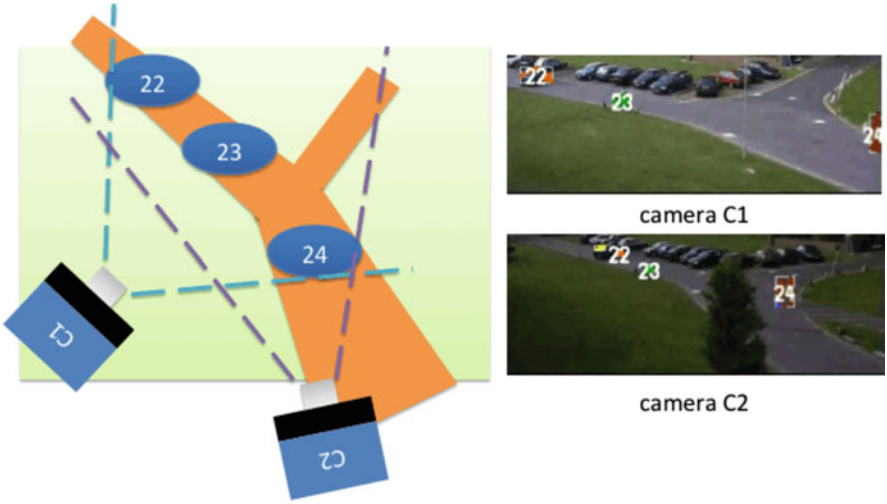

Fig. 4 Overlapping fields of view

- **Extrinsic** spatial calibration determines the position of the camera relative to the scene that it views and, in the case of camera networks, to the other cameras in the system.
- **Temporal** calibration determines the time at which frames are captured and, in the case of camera networks, how the cameras are synchronized relative to each other.

Some calibration problems for camera networks are similar to the calibration problems for a single camera. Camera networks also introduce new calibration problems. Without proper calibration, we cannot properly compare data or analysis results between cameras.

Figure 4 shows a simple example of extrinsic calibration. The two cameras have different views of the road and the tracking subjects.

The book by Hartley and Zisserman [13] provides a thorough discussion of camera calibration. A thorough review of the subject is beyond the scope of this article; we concentrate here on distributed algorithms to solve calibration problems. However, we can identify a few basic techniques for calibration. Intrinsic calibration is necessary to determine the relationship between image features and objects in world coordinates. When we have multiple cameras, we need to determine the relationships between the cameras as well as the internal parameters of each camera. The **fundamental matrix** describes the relationship between two views of a point in space. The **three-view geometry** problem determines the relationships between three cameras, which is based on the correspondence between two lines and a point. The **trifocal tensor** describes the required relationship, along with a set of internal constraints. The **four-view geometry** problem is yet more complex and is often

solved using the simpler affine camera model. This problem can be generalized to the n-camera case.

Radke et al. [25] developed a distributed algorithm for the metric calibration of camera networks. External calibration of a camera determines the position of the camera in world coordinates using a rotation matrix and translation vector. Intrinsic parameters include focal length, location of the principal point, and skew. Calibrating cameras through image processing is usually more feasible than by directly measuring camera position. Devarajan et al. model the camera network using a communication graph and a vision graph. 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 a 3×4 matrix P_i that gives the rotation matrix and optical center of the camera. The intrinsic parameter matrix for camera i is known as K_i. A set of points $X = \{X_1, \ldots, X_n\}$ are used as reference points for calibration; these points can be selected using a variety of methods, such as SIFT. Two cameras have overlapping fields-of-view if they can both see a certain number of the Xs.

The reconstruction is ambiguous and we need to estimate a matrix H that turns the reconstruction into a metric form (preserving angles, etc.). A bundle adjustment step uses nonlinear minimization to improve the calibration. Synchronization, also known as temporal calibration, is necessary to provide an initial time base for video analysis. Cameras may also need to be resynchronized during operation. Even professional cameras may not provide enough temporal stability for long-running experiments and low-cost cameras often display wide variations in frame rate. Lamport and Melliar-Smith [18] developed an algorithm for synchronizing clocks in distributed systems; such approaches are independent of the distributed camera application. Velipasalar and Wolf [30] developed a video-based algorithm for synchronization. The algorithm works by matching target positions between cameras, so it assumes that the cameras have previously been spatially calibrated. A set of frames is selected for comparison by each camera. For each frame, an anchor point is selected. The targets are assumed to be on a plane, so the anchor point is determined by dropping a line from the top of the bounding box to the ground plane. Given these reference points, a search algorithm compares the positions of the targets to minimize the distance $D_{i,j}^{1,2}$ between frame sequences 1 and 2 at offsets i and j.

Pollefeys et al. [23] developed an algorithm for combined spatial and temporal calibration of camera networks. They use a moving target, such as a person, to provide data for calibration in the form of boundary points on the target's silhouette. They use RANSAC to match points from frames from two videos. The RANSAC search evaluates both the epipolar geometry and the temporal offset between the frames. They first generate a consistent set of fundamental matrix for three cameras.

They then add cameras one at a time until either all cameras have been calibrated or the errors associated with a camera's calibration are too large.

Porikli and Divakaran [24] calibrate color models of cameras by recording images of identical objects from each camera and then computing a correlation matrix of the histograms generated for the object from each camera.

6 Tracking

Tracking models the movement of targets over an extended period. The tracking problem can take different forms when several cameras are used, depending on whether the fields-of-views of the cameras overlap.

6.1 Tracking with Overlapping Fields-of-View

Velipasalar et al. [29] developed a peer-to-peer tracking system that fuses data at a higher level. At system calibration time, the cameras determine all other cameras that share overlapping fields-of-view. This information is used to build groups for sharing information about targets. Each camera runs its own tracker. As illustrated in Fig. 5, a camera maintains for each target in its field-of-view the position and appearance model of the target. The cameras then trade information about targets. The cameras that have a given target in their fields-of-view form a group. Cameras communicate every n frames, where n is chosen at design time. A more adaptive approach would be to adapt the communication rate based upon

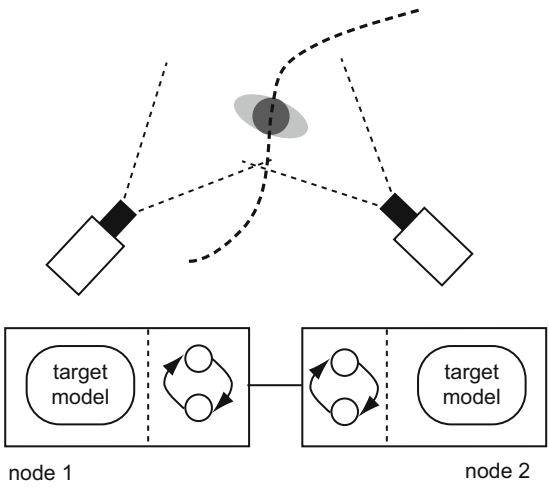

Fig. 5 Combining tracks during distributed tracking

the rate at which targets move. The group is built on top of a logical ring structure, which is commonly used to communicate in multiprocessors. The cameras in the group exchange information on the position of the target. Exchange allows cameras to estimate position more accurately. It also allows cameras whose view of the target is occluded to know where the target is during the occlusion interval. The protocol also negotiates an identity for each target. When a target enters the system, the cameras must also agree upon a common label for the target. They compare appearance models and positions to determine that they, in fact share a target. They then select a common label for that target.

Fleuret et al. [12] use dynamic programming to track multiple people in a scene. They discretize the planar floor into discrete regions and compute the most likely positions of people from frame to frame. They solve overlapping windows of 100 frames. Models for motion and color constrain the search space. They identify tracks one subject at a time to minimize the size of the search space.

6.2 Tracking in Sparse Camera Networks

In many environments, we cannot set up cameras whose fields-of-view cover the entire area of interest. In sparse camera networks, we must find alternative methods to correlate observations between cameras. The most common approach is to use Bayesian metrics to group observations into sets that correspond to paths. Search algorithms find the most likely assignments of observations to paths. A graph models the set of observation positions as nodes and the possible paths between those positions as edges.

Oh et al. [22] developed a Markov chain Monte Carlo (MCMC) algorithm for multi-target tracking (MCMC-MTT). We are given a set of target positions x_i and a set of observations y_i. We partition the available observations into a set of tracks $\tau = \{y_1, \ldots, y_t\}$. The set of all tracks for a given scene is ω. Given a new set of observations y_{t+1}, we want to assign them to tracks and possibly reassign existing observations to different tracks, generating an updated set of tracks ω' such that we maximize the posterior of ω'. Oh et al. showed that the posterior of ω' is:

$$P(\omega \mid Y) = \frac{1}{Z} \prod_{1 \le t \le T} p_z^{z_t} (1 - p_z)^{c_t} p_d^{d_t} (1 - p_d)^{u_t} \lambda_b^{a_t} \lambda_f^{f_t} \times \prod_{\tau \in \omega / \{\tau_0\}} \prod_{1 \le i \le |\tau| - 1} N(\tau(t_{i+1} \mid \bar{x}_{t_{i+1}}(\tau), b_{t_{i+1}}(\tau)) \tag{6}$$

In this formula, Z is a normalizing constant and $N()$ is the Gaussian density function with mean μ and covariance matrix Σ. In the approach of Oh et al., MCMC multi-target tracking makes use of several types of moves to generate new candidate tracks:

- Birth/death: A birth move creates a new track while death destroys an existing one.
- Split/merge: A split breaks one track into two pieces while a merge combines two tracks into one.
- Extension/reduction: Extension lengthens an existing track by adding new observations at the end of the track while reduction shortens a track.
- Track update: An update adds an observation to a track.
- Track switch: A track switch exchanges observations between two tracks.

A move is accepted or rejected with probability

$$A(\omega, \omega') = \min\left(1, \frac{\pi(\omega')q(\omega', \omega)}{\pi(\omega)q(\omega, \omega')}\right) \tag{7}$$

where $\pi(\omega)$ is the stationary distribution of the states and $q(\omega, \omega')$ is the proposal distribution.

Kim et al. [16] used a modified graph model to encapsulate information about priors that guides the search process. Entry and exit nodes are often connected by cycles of edges that model entry and exit. Such cycles can induce long paths to be generated that consist of the target shuttling between two nodes. A supernode models a set of nodes and associated edges. The supernode represents the prior knowledge that such cycles are unlikely. Kim and Wolf [17] developed a distributed algorithm to perform the search over candidate paths. A camera is able to communicate only with nearby cameras. Each camera estimates local paths for the targets using its own observations and observations obtained from nearby cameras. A maximum weighted bipartite matching algorithm is used to match observations with their immediate predecessors. The local paths are then concatenated to create global paths for the targets.

Song and Roy-Chowdhury [28] used a modified form of optimal path solving with random variables for weights. They compare the similarity between observations to compute a similarity score for a pair of observations. They then solve a maximum matching problem in a weighted bipartite graph. They handle variations among the observations by relaxing independence of the correspondences between features using a path smoothness function.

Javed et al. [15] model variations in appearance of targets from camera to camera due to a combination of camera parameters, illumination, and pose using brightness transfer functions (BTFs). They generated inter-camera BTFs for pairs of cameras during a training phase using object histograms. They then use inter-camera BTFs to modify the similarity metric for observations during computation of the track.

Tracking from pan-tilt-zoom (PTZ) cameras is more challenging. de Agapito et al. [1] developed an algorithm for calibrating using a set of images from a camera that rotates. They assume that pixels are square, that the images are not skewed, and that there is a known principal point. Given those constraints, they can formulate the homographies between the reference image and the other images from the camera.

Standard least-squares algorithms can be used to solve the system of equations. Del Bimbo et al. [4] perform real-time tracking taking into account the homography between the reference view and successive frames of the PTZ camera. They use a particle filter to estimate the camera parameters. They use a SIFT algorithm to extract points in the images.

7 Gesture Recognition

A gesture is a sequence of poses [34]. Each pose can be extracted from a frame. The sequence of poses can then be determined from the poses using a variety of techniques such as hidden Markov models.

Van den Bergh et al. [2] developed an algorithm for pose estimation. They extract a three-dimensional hull over the subject using voxel carving. They achieve real-time performance by using a fixed lookup table for each voxel that gives for each pixel the set of voxels that project onto that pixel. They use an example-based classifier to bin the 3D hulls into poses.

Lin et al. [19] developed a peer-to-peer algorithm for gesture recognition. The distributed system is based on a single-camera gesture recognition system [34]. That algorithm extracts shapes, builds a graph model of the relationships between the shapes, and then matches that graph against a library of graphs representing poses.

The basic challenge in extending the single-camera system to multiple cameras is that a subject may stand such that no single camera has a complete view. In general, data must be combined from multiple cameras before a graph can be built. A brute-force approach would be to share frames, but this entails a substantial bandwidth penalty. Lin's system combines the features created by pixel-level region identification. The regions from different cameras are easily fused but they can be transmitted at lower cost. As shown in Fig. 6, when the target appears in the view

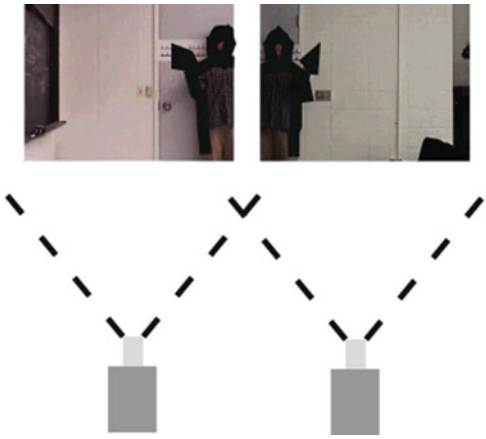

Fig. 6 Combining features for gesture recognition

of multiple cameras, each camera extracts the foreground object and extracts pixel boundaries for the foreground regions. A protocol determines which camera will fuse these boundaries and complete the recognition process. A group is formed by the cameras that can view the target. One member of the group, the lead node, has a token that denotes that it is responsible for fusion. The token moves to a different node when the target's position, as determined by the centerline of its bounding box, moves out of the field-of-view of the lead node. The token is passed to a camera whose field-of-view contains the target centerline.

8 Platform Architectures

The computing platform includes processing elements, interconnect, memory, and networking. The details of the platform can vary widely, but the fundamental assumptions made by most distributed computer vision algorithms include the nodes and network in a distributed system. Processing nodes are assumed to have reasonably powerful processors that can perform significant amounts of work in real time. RAM is generally not unlimited but enough is available for the required algorithms. Most systems assume that power is available to nodes for continuous operations, although it is possible to build a battery-operated system that will operate for a short interval. Network bandwidth and quality-of-service are an important consideration. Wireless networks generally do not provide enough bandwidth to allow a large number of cameras to stream video for processing in the cloud. Wired networks provide more bandwidth and better QoS but streaming video from multiple nodes will still require significant network resources. Distributed smart cameras that process video streams locally are motivated in part by networking limitations.

Heterogeneous platforms are commonly used for embedded computer vision. These platforms provide different types of computational units that can be applied to different parts of the system. In addition to CPUs, heterogeneous platforms may make use of graphics processing units (GPUs), digital signal processors (DSPs), and field-programmable gate arrays (FPGAs). The OpenCV library (http://opencv.org) has become a popular development environment for computer vision applications. OpenCV has been ported to a wide variety of platforms, including both CPUs and GPUs. Many platforms that target embedded computer vision applications include GPUs. These platforms fall into one of two categories: embedded GPUs such as the NVIDIA Tegra used in their Jetson platform is designed for mobile graphics and computer vision; smartphone platforms such as the Qualcomm Snapdragon family include both computational and wireless networking resources.

A GPU provides large number of floating-point units that can operate on local memory, features that can be exploited by computer vision algorithms. Modern GPUs use a multithreaded programming model to provide highly parallel execution. Yu and van Engelen [35] developed a GPU-based algorithm for importance sampling with parallelization of irregular wavefronts. They exploited properties of the probability distribution to determine an address calculation for accessing

importance function tables. Chen et al. [9] developed a hardware architecture for k-means clustering with hierarchical data sampling; their design included pipelined processors for logarithm and Bayesian Information Criterion computations. Bodensteiner and Arens [5] developed an algorithm for registration of 2D video to 3d LiDAR imagery; they used a GPU to perform backprojection of the video 2D features to the LiDAR models. Berjon et al. [3] developed a Bayesian classifier for moving object detection. They used region-of-interest masks to reduce computation time; they mapped the irregular set of RoI areas onto the GPU using a 1D list. Wang et al. [31] developed an MCMC-based learning algorithm for GPUs.

DSPs are instruction set processors optimized for streaming and array processing workloads found in digital signal processing. Very long instruction word (VLIW) architectures, which statically dispatch several instructions per cycle, are widely used in DSPs. Vector processors are also commonly used for signal processing workloads to take advantage of the vector and matrix operations commonly found in DSP and computer vision workloads.

Field-programmable gate arrays (FPGAs) can be used to build custom accelerators and heterogeneous architectures without resorting to a custom chip design. Typically an FPGA will be utilized either as a design and test platform for a final custom build or as a target reconfigurable fabric for these cases, respectively. Much like DSPs, these provide optimized processing ability, in this case with specialized processing units within the FPGA fabric. For example, high-speed integer multipliers and adders are found in many FPGA cells currently on the market. Furthermore, many FPGAs incorporate a simple embedded processor for algorithm flow control. The complexity in using this type of platform is in design overhead. An FPGA requires significantly more design consideration than the previously mentioned platforms. This is due to the inherent need for hardware and software task analysis. Also, the determined hardware tasks typically cannot be directly mapped to FPGA logic, and instead require algorithmic redesign and hardware timing scheme design and analysis.

All of these platforms raise the need for memory bandwidth analysis during architectural design and mapping. This is due to the platforms having a smaller footprint, and resultant lower amount of available storage on the board, not to mention on the chip itself. As vision algorithm complexity increases through use of larger pixel representation schemes, greater temporal frame storage, and also accuracy requirements which abound in real-time critical vision systems, the amount of data which needs to be both transferred on and off chip as well as stored on-chip intermediately increases dramatically. Schlessman [26] addresses each of these issues with the MEAD methodology for embedded architectural design, ultimately providing vision designers with a clearer concept of what platform architecture is best suited for the algorithm under consideration.

9 Summary

Distributed algorithms offer a new approach to computer vision. Nodes in a distributed system communicate explicitly through mechanisms that have non-zero cost in delay and energy. This constraint encourages us to consider algorithms that refine the representation of a scene in stages with predictable ways of combining data within a stage. Distributed algorithms are, in general, more difficult to develop and prove correct, but they also provide advantages such as robustness. Distributed signal processing may require new statistical representations of signals that can be efficiently shared over the network while preserving the useful properties of the underling signal. We also need a better understanding of the accuracy of the approximations underlying distributed signal processing algorithms—how do termination conditions affect the accuracy of the estimate, for example.

Acknowledgements This work was supported in part by the National Science Foundation under grant 0720536.

References

1. de Agapito, L., Hartley, R., Hayman, E.: Linear self-calibration of a rotating and zooming camera. In: Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on., vol. 1, pp. 2 vol. (xxiii+637+663) (1999). https://doi.org/10.1109/CVPR.1999.786911
2. den Bergh, M.V., Koller-Meier, E., Kehl, R., Gool, L.V.: Real-time 3d body pose estimation. In: H. Aghajan, A. Cavallaro (eds.) Multi-Camera Networks: Principles and Applications, chap. 14. Academic Press (2009)
3. Berjon, D., Cuevas, C., Moran, F., Garca, N.: Region-based moving object detection using spatially conditioned nonparametric models in a GPU. In: 2014 IEEE International Conference on Consumer Electronics (ICCE), pp. 359–360 (2014). https://doi.org/10.1109/ICCE.2014.6776041
4. Bimbo, A.D., Dini, F., Pernici, F., Grifoni, A.: Pan-tilt-zoom camera networks. In: H. Aghajan, A. Cavallaro (eds.) Multi-Camera Networks: Principles and Applications, chap. 8. Academic Press (2009)
5. Bodensteiner, C., Arens, M.: Real-time 2d video/3d lidar registration. In: Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), pp. 2206–2209 (2012)
6. Boykov, V., Huttenlocher, D.: Adaptive Bayesian recognition in tracking rigid objects. In: Proceedings, IEEE Conference on Computer Vision and Pattern Recognition, pp. 697–704. IEEE (2000)
7. Bramberger, M., Quaritsch, M., Winkler, T., Rinner, B., Schwabach, H.: Integrating multi-camera tracking into a dynamic task allocation system for smart cameras. In: Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance (AVSS 2005), pp. 474–479. IEEE (2005)
8. Candy, J.V.: Boostrap particle filtering. IEEE Signal Processing Magazine **73**, 73–85 (2007)
9. Chen, T.W., Chien, S.Y.: Flexible hardware architecture of hierarchical k-means clustering for large cluster number. IEEE Transactions on Very Large Scale Integration (VLSI) Systems **19**(8), 1336–1345 (2011). https://doi.org/10.1109/TVLSI.2010.2049669

10. Coates, M.: Distributed particle filters for sensor networks. In: Information Processing in Sensor Networks, 2004. IPSN 2004. Third International Symposium on, pp. 99–107. IEEE (2004)
11. Collins, R.T., Lipton, A.J., Fujiyoshi, H., Kanade, T.: Algorithms for cooperative multisensory surveillance. Proceedings of the IEEE **89**(10), 1456–1477 (2001)
12. Fleuret, F., Berclaz, J., Lengagne, R., Fua, P.: Multicamera people tracking with a probabilistic occupancy map. Pattern Analysis and Machine Intelligence, IEEE Transactions on **30**(2), 267–282 (2008). https://doi.org/10.1109/TPAMI.2007.1174
13. Hartley, R.I., Zisserman, A.: Multiple View Geometry in Computer Vision, second edn. Cambridge University Press, ISBN: 0521540518 (2004)
14. Horprasesert, T., Harwood, D., Davis, L.S.: A statistical approach for real-time robust background subtraction and shadow detection. In: IEEE International Conference on Computer Vision FRAME-RATE Workshop (1999)
15. Javed, O., Shafique, K., Shah, M.: Appearance modeling for tracking in multiple non-overlapping cameras. In: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 2, pp. 26–33 (2005). https://doi.org/10.1109/CVPR.2005.71
16. Kim, H., Romberg, J., Wolf, W.: Multi-camera tracking on a graph using Markov chain monte carlo. In: Proceedings, 2009 ACM/IEEE International Conference on Distributed Smart Cameras. ACM (2009)
17. Kim, H., Wolf, M.: Distributed tracking in a large-scale network of smart cameras. In: Proceedings of the Fourth ACM/IEEE International Conference on Distributed Smart Cameras, p. 8–16. ACM Press (2010)
18. Lamport, L., Melliar-Smith, M.: Synchronizing clocks in the presence of faults. Journal of the ACM **32**(1), 52–78 (1985)
19. Lin, C.H., Lv, T., Wolf, W., Ozer, I.B.: A peer-to-peer architecture for distributed real-time gesture recognition. In: Proceedings, International Conference on Multimedia and Exhibition, pp. 27–30. IEEE (2004)
20. Mallett, J., Jr., V.M.B.: Eye society. In: Proceedings IEEE ICME 2003. IEEE (2003)
21. McMillan, L., Bishop, G.: Plenoptic modeling: an image-based rendering system. In: Proceedings, ACM SIGGRAPH, pp. 39–46. ACM (1995)
22. Oh, S., Russell, S., Sastry, S.: Markov chain monte carlo data association for general multiple-target tracking problems. In: Proc. 43rd IEEE Conf. Decision and Control (2004)
23. Pollefeys, M., Sinha, S.N., Guan, L., Franco, J.S.: Multi-view calibration, synchronization, and dynamic scene reconstruction. In: H. Aghajan, A. Cavallaro (eds.) Multi-Camera Networks: Principles and Applications, chap. 2. Academic Press (2009)
24. Porikli, F., Divakaran, A.: Multi-camera calibration, object tracking and query generation. Tech. Rep. TR-2003-100 (2003)
25. Radke, R., Devarajan, D., Cheng, Z.: Calibrating distributed camera networks. Proceedings of the IEEE **96**(10), 1625–1639 (2008)
26. Schlessman, J.: Methodology and architectures for embedded computer vision (2012). PhD Thesis, Department of Electrical Engineering, Princeton University, in preparation
27. Sheng, X., Hu, Y.H., Ramanathan, P.: Distributed particle filter with gmm approximation for multiple targets localization and tracking in wireless sensor network. In: Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on, pp. 181–188. IEEE (2005)
28. Song, B., Roy-Chowdhury, A.: Robust tracking in a camera network: A multi-objective optimization framework. Selected Topics in Signal Processing, IEEE Journal of **2**(4), 582–596 (2008). https://doi.org/10.1109/JSTSP.2008.925992
29. Veliapasalar, S., Schlessman, J., Chen, C.Y., Wolf, W.H., Singh, J.P.: A scalable clustered camera system for multiple object tracking. EURASIP Journal on Image and Video Processing **2008** (2008). Article ID 542808

30. Velipasalar, S., Wolf, W.H.: Frame-level temporal calibration of video sequences from unsynchronized cameras. Machine Vision and Applications Journal (DOI 10.1007/s00138-008-0122-6) (2008)
31. Wang, Y., Qian, W., Zhang, S., Liang, X., Yuan, B.: A learning algorithm for bayesian networks and its efficient implementation on gpus. IEEE Transactions on Parallel and Distributed Systems **27**(1), 17–30 (2016). https://doi.org/10.1109/TPDS.2014.2387285
32. Wolf, W.: High Performance Embedded Computing. Morgan Kaufman (2006)
33. Wolf, W.: Modern VLSI Design: IP-Based Design, fourth edn. PTR Prentice Hall (2009)
34. Wolf, W., Ozer, B., Lv, T.: Smart cameras as embedded systems. IEEE Computer **35**(9), 48–53 (2002)
35. Yu, H., van Engelen, R.: Importance sampling on Bayesian networks with deterministic causalities