A Flexible Software Architecture for a Network of Heterogeneous Smart Cameras

Dominik Pieczyński, Marek Kraft(✉), and Michał Fularz

Institute of Control and Information Engineering, Poznań University of Technology, Piotrowo 3a, 60-965 Poznań, Poland
marek.kraft@put.poznan.pl

Abstract. This paper presents a flexible software solution, facilitating easy deployment and control of individual sensor nodes in a camera network. The presented solution is lightweight, extensible and provides an easy to use base for the implementation of a range of collaborative tasks performed by multi-camera setups.

1 Introduction

In recent years, video surveillance has become a widespread technology. With the constant growth of demand for visual supervision, increasing camera and computational platform capability and their price decrease one can predict, that the growth in this sector will be sustained for year to come. As the scale of video surveillance systems increases and the number of cameras in a typical surveillance system grows larger, the analysis of data by human operators becomes increasingly hard, or even impossible because of the sheer scope and volume of visual data. Moreover, scientific studies reveal, that tiredness and loss of attention over prolonged periods of time is a serious issue with human operators of video surveillance systems [8]. The most common way to remedy those issues is the employment of automated, image processing based surveillance.

Automated surveillance is, however, not without its own issues. Extraction of useful data from surveillance camera images is not a trivial task. Most image and video processing algorithms are considered computationally intensive, which is an even bigger concern in the case of large scale systems. Moreover, to make use of the full potential of a number of cameras, one should not treat them as isolated sensors and try to employ a holistic approach to sensor data analysis. A straightforward approach to data aggregation is to use a centralized server for all the processing. This, however, usually calls for constant transmission of video streams, resulting in a significant strain on the communication infrastructure. In extreme cases, the communication channels may not be able to support continuous transmission from all the installed cameras. One way to remedy this is the use of constantly improving video compression methods. This approach,

The original version of this chapter was revised: The spelling of the first author's name was corrected. The erratum to this chapter is available at DOI: 10.1007/978-3-319-47274-4_34

R.S. Choraś (ed.), *Image Processing and Communications Challenges 8*,
Advances in Intelligent Systems and Computing 525, DOI 10.1007/978-3-319-47274-4_11

however, is not without its own issues. Video compression requires additional computational power and contributes to the power consumption of the cameras in the network [11]. Moreover, increasing the compression rate may have a negative impact on the results of the image processing algorithms applied to the transmitted images [7]. The limitations of the central processing paradigm can be overcome by using distributed processing.

In the case of distributed processing, a significant portion of computations is performed by the camera network nodes themselves, following the 'smart camera' concept. Since modern embedded processors are currently powerful enough to handle typical workloads associated with real-time image and video processing, the main limitation of such an approach becomes less and less significant [3, 5]. Nodes of such camera networks are to a large extent autonomous, and can communicate with the central server and each other [1,4]. With the majority of image processing operations performed in-place, the amount of transmitted data can be significantly reduced. Taking full advantage of network-wide information raises the need for specialized algorithms for node and network management and control. Moreover, the flexibility and scalability of software that manages such vast networks is a very desirable feature.

In this paper, we describe an architecture of a software system capable of governing a network of heterogeneous smart cameras. The network performs the task of adaptive activity monitoring [10], but can be easily adapted to host a range of other image and video processing applications. Smart cameras of different types and architectures can dynamically join and leave the network. Moreover, they are being made aware of their mutual relations and receive software updates automatically. Ethernet or WiFi wireless network can be used for communication, making the integration with existing infrastructure virtually effortless.

2 Related Work

While the methods for data processing and the communication technology utilised in camera networks show constant progress, less attention is paid to system-level software (middleware). The focus is mainly on the design of individual sensor nodes facilitating node communication using a variety of protocols [6,13] or information fusion [2] and collaborative processing in a variety of scenarios [15]. This, however, is bound to change, as proper middleware makes the design, implementation and deployment of vast camera network significantly less troublesome. In [14] the authors propose a camera network software architecture which moves the majority of processing to a cloud-based solution. According to the authors, the smart cameras are not well suited for high-performance image processing, hence the data is moved, aggregated and processed in virtual local hubs. The closest counterpart to the solution described in this was presented in [9]. It is also an architecture and system independent solution, but based on a more general publisher/subscriber architecture and its potential use cases extend beyond camera networks.

3 Description of the Implemented System

The main contribution of this paper is a software framework that facilitates the deployment and testing of new solutions for visual sensor networks. It consist of two separate parts - one is a firmware for a single smart camera (sensor node) while the other one is a network coordinator module.

The described middleware offers a few distinctive advantages:

- automatic discovery of new nodes that join the network,
- centralized way of distributing the messages,
- general architecture for a software dedicated for a smart camera,
- ability to automatically update all the nodes when the new firmware version is deployed on the server,
- support for different types of hardware, as the software is hardware-agnostic,
- ability to remotely set up the parameters of a camera,
- an optional UI on the server, enabling system monitoring.

The block schematic of the proposed system is shown in the Fig. 1.

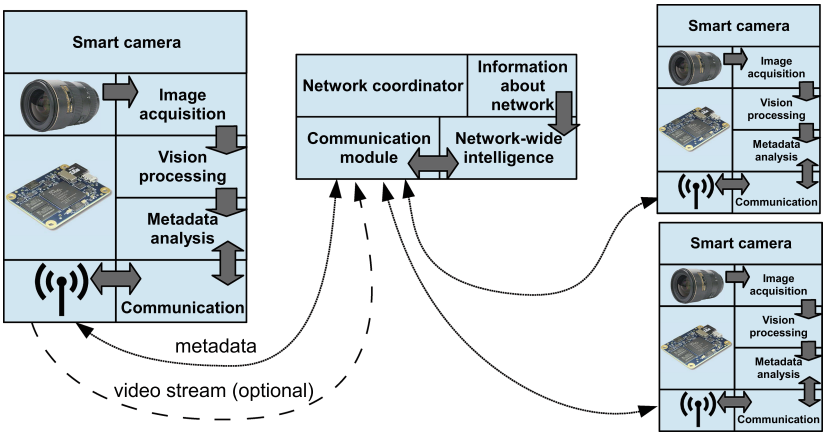


Fig. 1. Network schematic

The smart camera software is split into four cooperating and interchangeable modules, responsible for following tasks: image acquisition, vision processing, metadata analysis and communication. Partitioning tasks in such way makes it really easy to replace one of the modules. The image acquisition module captures the data from the image sensor and configures low level parameters of the camera. In our example code two different implementations are provided - one for dedicated Raspberry Pi camera using CSI (Camera Serial Interface) and the other one using general UVC (USB video class). The vision processing module is supposed to extract the high level information from the acquired images.

It is tightly integrated with the established OpenCV video processing library, enabling quick and efficient implementation of desired functionality. Implementation of average median algorithm for background subtraction and movement detection (Sect. 4) is provided with the code as an example. After the image processing step and extraction of high level information about the scene, the metadata analysis module is charged with the task of interpreting this information and taking whatever action is appropriate like eg. sending an alarm message to others nodes of the system, performing additional image analysis etc. The final module is responsible for communication with the rest of the system. It is able to update the camera's other modules when the new version of the system software is available. It is also responsible for the procedure of joining the network and negotiating the node's parameters.

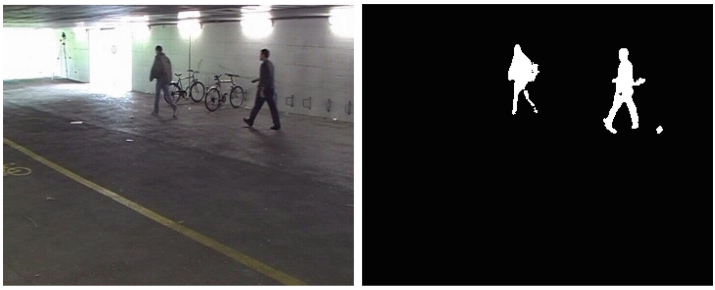
The software for the network coordinator is also modular. The communication module is similar to the one found in the smart camera – it is responsible for managing data transfers with other nodes of the network. The network-wide intelligence module is gathering the information from all the cameras and, using stored information about the network topology, is capable of performing actions like sending information to all or just a few nodes of the network or controlling what algorithms are executed in each node. In the example application, the coordinator gathers data about the activity (defined as the amount of movement in the scene observed by the camera) from all of the nodes and resends that data augmented with the information about the activity of the neighbours [10].

The described framework is written using high level Python programming language in a way to make it as hardware agnostic as possible. As an example three different processing platforms were used to test the solution. The requirements are modest and limited to the operating system capable of running Docker application container engine. In addition to that the developed software relies on open sources solutions like Linux operating system and an established image processing library (OpenCV). All this makes the software accessible and easy to use by other researchers, so that they can deploy and test new solutions for visual sensor networks.

4 Image Processing

The activity on the scene observed by any given camera in the network is computed based on a background subtraction algorithm. Approximate median algorithm presented in [12] was selected for its simplicity and relatively good quality of results. The background model is updated by performing the following steps:

- if the intensity value $I_{x,y}$ of currently investigated pixel of the current frame I is greater than the value of the corresponding background model pixel $B_{x,y}$, the value of $B_{x,y}$ is incremented by one,
- if the intensity value $I_{x,y}$ of currently investigated pixel of the current frame I is smaller than the value of the corresponding background model pixel $B_{x,y}$, the value of $B_{x,y}$ is decremented by one,



(a) input image (b) resulting foreground image

Fig. 2. Example results from the background subtraction pipeline. Moving (active) objects are successfully detected and highlighted

– if the intensity value $I_{x,y}$ of currently investigated pixel of the current frame I has the same value as the corresponding background model pixel $B_{x,y}$, the value of $B_{x,y}$ remains unchanged.

Computing the difference image D as an absolute value of the difference between the current frame I and the background model B returns the current foreground. The difference image is subjected to additional post-processing using a 3×3 Gaussian, fixed-threshold binarization and binary morphological closing to improve the quality of results and foreground consistency. The percentage of foreground pixels reflects the activity observed by the local sensor node. Example output produced by the background subtraction pipeline is shown in Fig. 2.

5 Communication Scheme

TCP, abstracted by the Transports and Protocols API from Python's asyncio module, is used as the transport protocol. Binary packets, encoded using MessagePack module are sent in a bidirectional fashion between clients and a server.

The server, when started, opens a port, which the cameras will later connect to. Additionally, the server begins the process of discovering Docker daemons available in the network using mDNS/DNS-SD protocols. When a new device is discovered, the server connects to the client's Docker daemon, downloads or updates the service image if necessary and starts a new application container. The server's address is passed to the container during creation as an environment variable, allowing the client software to begin negotiating settings with the server. When the server detects a new connection from the client software, it sends a configuration packet to that device. After accepting new settings the client sends the packet containing its identifier, based on the device's hostname, back to the central node. The client device becomes acknowledged by the server and can start transmitting data packets.

The central node processes all the packets from the connected clients and calculates camera's neighbours activity level. The result is then transmitted to the appropriate devices.

There is also a possibility that the change of camera settings is needed. In such case, the server software prepares a new configuration packet and fills it with the requested settings' values. This packet is then transmitted to the client, which changes the settings accordingly.

The connection termination is normally initialized by the server, which performs a standard TCP Connection Termination.

6 Results and Discussion

The prepared solution was tested on the Raspberry Pi model B evaluation board (with Broadcom BCM2835 processor - ARMv6 single core, 0.7 GHz), Odroid XU4 evaluation board (with Samsung Exynos 5422 processor - ARMv7 octa core, 2 GHz) and notebook computer (with AMD E2-1800 processors - x86-64 compatible dual core, 1.7 GHz).

The same application was run on each of the targets and the CPU usage and data transfer was recorded. The application analyses the observed scene and detects the movement. If the amount of movement detected in a node and

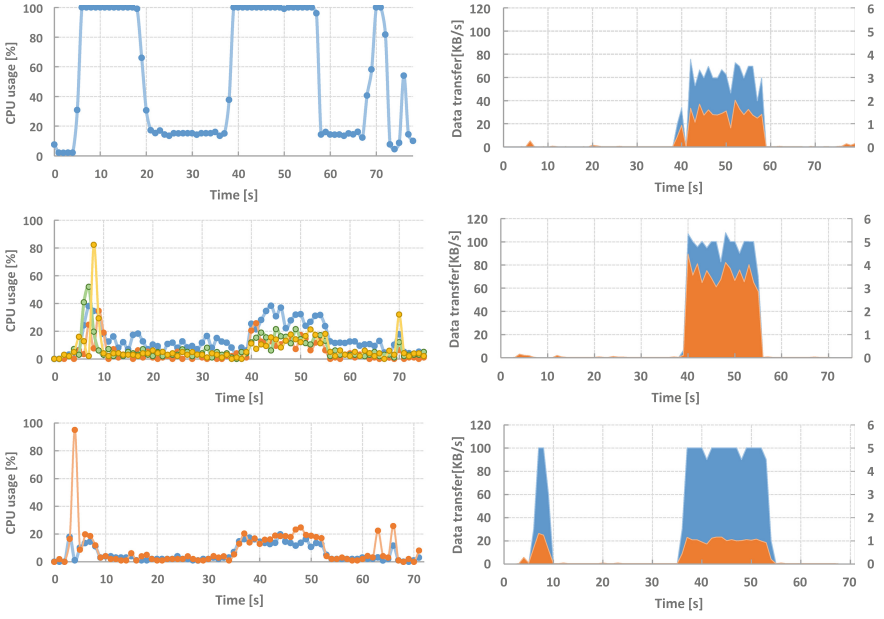


Fig. 3. The CPU usage (on the left) and the amount of data sent and received (on the right). First row shows results for Raspberry Pi, second one for the Odroid, while the last one shows results for PC. On the CPU usage charts different colors denote the usage of different CPU cores (1 for RPi, 4 for Odroid and 2 for PC). Blue color on the right charts represents the amount of data sent with scale on left axis; the orange color represents the amount of data received with scale on the right axis (Color figure online)

received from neighbouring nodes exceeds the threshold, the smart camera starts sending the images to the network coordinator for recording. The gathered data is shown if Fig. 3.

The figures shows that the usage of processor and amount of data transferred is high only during the periods of activity, when system processes 15 frames per second and sends them to server. During the normal operation, the processor usage is modest and the amount of data transferred is negligible, which shows that the proposed middleware software does not impose a significant load on the processor or the communication interface. It is clearly visible, that the slowest target – the Raspberry Pi – is not capable of achieving the 15 fps, which influences the amount of data sent.

7 Conclusions

The paper presents a scalable, flexible software, serving as a base for operation of a network of distributed smart camera sensors. The described software facilitates the deployment, configuration and testing of remote sensor nodes, which greatly simplifies the development of applications involving multi-camera systems. Moreover, the solution's computational complexity is very low, enabling its use on low-end hardware without a significant overhead. The software is released with a permissive open source license and can be downloaded from GitHub[1,2].

Acknowledgement. This research was financed by the Polish National Science Centre grant funded according to the decision DEC-2011/03/N/ST6/03022, which is gratefully acknowledged.

References

1. Aghajan, H., Cavallaro, A.: Multi-camera Networks: Principles and Applications. Academic press, London (2009)
2. Bajo, J., Paz, J.F.D., Villarrubia, G., Corchado, J.M.: Self-organizing architecture for information fusion in distributed sensor networks. Int. J. Distrib. Sen. Netw. **2015**, 2 (2015)
3. Belbachir, A.N.: Smart Cameras. Springer, Heidelberg (2010)
4. Bhanu, B., Ravishankar, C., Roy-Chowdhury, A., Aghajan, H., Terzopoulos, D.: Distributed Video Sensor Networks. Springer, London (2011)
5. Bobda, C., Velipasalar, S.: Distributed Embedded Smart Cameras. Springer, New York (2014)
6. Chen, P., Hong, K., Naikal, N., Sastry, S.S., Tygar, D., Yan, P., Yang, A.Y., Chang, L.C., Lin, L., Wang, S., Lobatón, E., Oh, S., Ahammad, P.: A low-bandwidth camera sensor platform with applications in smart camera networks. ACM Trans. Sens. Netw. **9**(2), 21:1–21:23 (2013)

[1] https://github.com/PUTvision/VSNServer.
[2] https://github.com/PUTvision/VSNClient.

7. Cozzolino, A., Flammini, F., Galli, V., Lamberti, M., Poggi, G., Pragliola, C.: Evaluating the effects of MJPEG compression on motion tracking in metro railway surveillance. In: Blanc-Talon, J., Philips, W., Popescu, D., Scheunders, P., Zemčík, P. (eds.) ACIVS 2012. LNCS, vol. 7517, pp. 142–154. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33140-4_13
8. Dadashi, N., Stedmon, A., Pridmore, T.: Semi-automated CCTV surveillance: the effects of system confidence, system accuracy and task complexity on operator vigilance, reliance and workload. Appl. Ergon. **44**(5), 730–738 (2013)
9. Dieber, B., Simonjan, J., Esterle, L., Rinner, B., Nebehay, G., Pflugfelder, R., Fernandez, G.J.: Ella: middleware for multi-camera surveillance in heterogeneous visual sensor networks. In: 2013 Seventh International Conference on Distributed Smart Cameras (ICDSC), pp. 1–6. IEEE (2013)
10. Kraft, M., Fularz, M., Schmidt, A.: Collaborative, context based activity control method for camera networks. In: Battiato, S., Blanc-Talon, J., Gallo, G., Philips, W., Popescu, D., Scheunders, P. (eds.) ACIVS 2015. LNCS, vol. 9386, pp. 118–129. Springer, Heidelberg (2015). doi:10.1007/978-3-319-25903-1_11
11. Ma, T., Hempel, M., Peng, D., Sharif, H.: A survey of energy-efficient compression and communication techniques for multimedia in resource constrained systems. Commun. Surv. Tutor. IEEE **15**(3), 963–972 (2013)
12. McFarlane, N., Schofield, C.: Segmentation and tracking of piglets in images. Mach. Vis. Appl. **8**, 187–193 (1995)
13. Miller, L., Abas, K., Obraczka, K.: Scmesh: solar-powered wireless smart camera mesh network. In: 2015 24th International Conference on Computer Communication and Networks (ICCCN), pp. 1–8, August 2015
14. Saini, M.K., Atrey, P.K., Saddik, A.E.: From smart camera to smarthub: embracing cloud for video surveillance. Int. J. Distrib. Sens. Netw. **2014**, 1–10 (2014)
15. Tessens, L., Morbee, M., Aghajan, H., Philips, W.: Camera selection for tracking in distributed smart camera networks. ACM Trans. Sen. Netw. **10**(2), 23:1–23:33 (2014)