A case study on decentralized object tracking in smart camera networks

Herwig Guggi   Georg Nebehay   Bernhard Rinner

Abstract—**This paper analyses a multi-camera system that is used for object tracking. The system uses smart cameras to perform tracking in a distributed and decentralized manner. The tracking for an object is always performed by one single camera. As soon as the camera loses the object, a handover is automatically initiated by this camera. After handover, the other camera is responsible for this object and the first camera can work on a different task.**

I. Introduction

This paper presents the measurement results of an implemented multi-camera object tracking system. The main properties of this system are: No central coordination is needed for tracking. No neighbourhood-information is required during system startup. Autonomous tracking handover is performed within the camera network.

This work is the first approach towards a resource efficient middleware system. The main idea is to reduce the required resources to a minimum but still solve the task.

These properties will in future enable the fast setup of an observation system. Furthermore due to the flexible design a future system will be able to overcome failures of single cameras and make it easy to add new cameras to the system during runtime.

There are two important properties to have a scalable system. The first is that there must not be a single part of the system that can be only executed on a single device. The two main parts of our system are the image processing (tracking) and the handover of the tracking responsibilities. In the current implementation, all these tasks are executed by individual smart cameras. The second property of a scalable system is the ability to add more devices. By adding more cameras to the system, the performance of the system is increased as the observation area is increased.

Section II focuses on the related work of smart camera networks and middleware systems for smart cameras. Section III describes the implemented software architecture. Section IV describes the experimental setup, visualizes details about the implementation of the handover algorithm and lists time measurements from this system. The final Section V concludes the paper and outlines plans for future development.

II. Related Work

The subject of distributed smart cameras is to integrate a number of smart cameras in a common network. The main reasons for doing so are to (1) resolve occlusion, and (2) extend sensor coverage. Compared to a network of ordinary dull cameras, smart cameras offer the benefit that raw data do not have to be transmitted via the network. These raw data are processed on the sensor platform and the results - which have a reduced size compared to the raw data - are transmitted. As each camera has reasonable computing and communication capabilities, such a network of smart cameras can also be treated as a distributed system for image processing. Distributed smart cameras can be organized either in a centralized manner, where some sort of pre-processing is done on the individual cameras but a central node controls each camera and merges information provided by individual cameras, or in a completely decentralized fashion, where cameras have to organize themselves and collaborate on a certain task.

Implementing and deploying distributed smart cameras with decentralized coordination poses several new research challenges. Rinner et al. [1] and Lin et al. [2] have already discussed that a substantial middleware would greatly enhance application development. Such a middleware has to integrate the camera's image processing capabilities and provide a transparent inter-camera communication mechanism.

The data transfer mechanisms must be designed for timely data delivery to support real-time and low power computation. One open question in middleware architectures is the breakdown between generic and application-specific services. Generic services can be shared among different applications, saving development time and cost, but application-specific versions of services can be tuned to the characteristics of the application.

Quaritsch et al. [3] propose to use agents as top-level abstraction. A distributed application comprises several mobile agents, whereas agents represent image processing tasks within the system. Introducing mobility to agents allows to move the image processing tasks between cameras as needed.

Patricio et al. [4] also use the agent-oriented paradigm. But in their approach, an agent manages a single camera. The agents have an internal state representing beliefs, desires and intentions. Collaboration of cameras hence maps to collaboration of agents, i.e, an agent can inform its neighbor about an object expected to appear or ask other agents whether they currently track the same object.

Fleck et al. [5] demonstrate a multi-camera tracking implementation. However camera coordination and object handoff between cameras is done on a central host. Each camera uses a particle-filter based tracking algorithm to track the individual objects within a single camera's field of view. The camera nodes report the tracking results along with the object description to the central server node.

Designing, implementing and deploying applications for distributed execution is much more complex than for single device systems. On general-purpose platforms, distributed applications are often based on a middleware system which provides services for networking and data transfer [6]. While there are systems that support a developer of a distributed application to access sensor data like ICE middleware [7], there is no reasonable system-level software available that is able to support cooperation and collaboration within a distributed sensor network.

Adaptive middleware systems can be used to implement applications that adapt to current system requirements. Rahnama et. al. [8] describe such an adaptive system and compare it to a non-adaptive implementation. Another example is shown by Hurtado et. al [9]. Their middleware system is based on the Open Service Gateway Interface (OSGi). This interface provides a universal publish-subscribe based protocol which is used to dynamically load and unload components during runtime. The described systems [8] and [9] do not modify parameters of single components to adapt to new conditions. They instead exchange components and modify the behaviour of the system.

III. System Architecture

This section is split into three subsections. In Section III-A, the principle tracking approach is explained. Section III-B describes the system architecture. Section III-C concentrates on the single camera tracking algorithm and Section III-D explains the software integration through the middleware.

A. Multi-Camera Tracking Approach

To achieve an independent camera assignment, a novel market-based approach where cameras bid for objects being in their field of view [10] is applied. Vickrey auctions are used to decide which camera has the best view of the object out of all neighbouring cameras. This approach does not need a central component at all and allows each camera to valuate its view on the object on its own. To initiate an auction, a communication channel between the cameras has been created to ensure confident tracking within the entire network.

Analysing the communication patterns between the cameras enables the middleware to generate a local model of neighbouring cameras. This model can for example be used to reduce communication or to inform neighbouring nodes about incoming objects to be tracked. This approach has been tested in a dedicated simulation environment and will be employed in a future version of this system.

For further details and results of the handover algorithm including comparisons to other handover strategies, see [10].

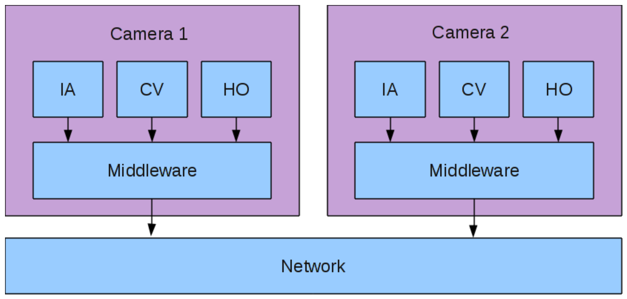

Fig. 1: Sketch of how the main components are integrated. IA (image acquisition), CV (computer vision), HO (handover)

B. Prototype Architecture

The implemented system autonomously detects and tracks an object within a smart camera network. Figure 1 depicts the architecture of the system which consists of multiple smart cameras operating within a network. All processing is performed within the network of these smart cameras, and each camera operates completely autonomously. Furthermore, all devices are connected via Ethernet to each other as well as to a dedicated client-side application which is able to show the video streams from all cameras.

The tracking of a designated object within a field of view of a single camera is accomplished by the Computer Vision (CV) module. The CV uses a model of the object to be tracked. This model is learned by using object features. As soon as the object has been identified within the field of view of the camera, the CV starts tracking the object.

C. Single Camera Tracking

The single camera tracking algorithm is an implementation for the computer vision component. This component represents the tracking and detection algorithm. The main functionalities are:

- Calculation of a model representation from a given image and a bounding box marking the object of interest
- Searching for an object which is described by a model
- Tracking of an object which is described by a model

Conventional frame-to-frame tracking methods suffer from the problem of error accumulation, since the tracking result for the current frame is based directly on the tracking result of the previous frames. Additionally, for frame-to-frame tracking mechanisms typically there is an implicit assumption that the time between every two frames presented to the tracking algorithm is small and constant. In this section we describe a Tracking-by-Detection [11] mechanism that overcomes these two problems. In Tracking-by-Detection methods object detection is performed in every frame and the consecutive output of the object detector is interpreted as a trajectory.

We use the following features for object detection: We calculate one histogram for each color channel. Each histogram is composed of 10 equally spaced bins. By concatenating these histograms we obtain a feature vector of $m = 30$ dimensions. Features are calculated on rectangular regions of interest. We refer to the region of interest that was selected by the user and that initially contains the object of interest as the model.

In order to compare a region of interest to the model, we perform histogram intersection [12] between the features of the region of interest R and the features of the model M:

$$h(M, R) = \sum_{j=1}^{n} \min(M_j, R_j)$$

The result of the histogram intersection is then normalized by dividing by the sum of all pixel values in the model

$$h_{norm}(M, R) = \frac{h(M, R)}{\sum_{j=1}^{m} M_j},$$

thus yielding values between 0 and 1. We define a value of $h_{norm}(M, R) > 0.5$ to be a positive match of the model. This mechanism can be extended to track multiple objects by storing one model $M_1, \ldots, M_n$ for each object and by performing a nearest neighbor search in order to find the best match.

In order to detect regions of interest in an input frame, we first calculate the absolute difference from the frame to a background image. Next, we apply a binary thresholding of the result ($\theta = 32$). We then perform a labeling of connected components in the thresholded image, using a linear-time labeling algorithm from [13]. The bounding box around each component is a region of interest.

D. Middleware System

To combine multiple cameras and allow tracking of an object through a network of cameras, communication between the cameras is needed. This communication is handled by the middleware (MW). To facilitate flawless tracking among multiple cameras, each object is dynamically assigned to a camera during runtime.

Furthermore, a simple user interface has been created to allow a user to view the images from different cameras, to select an object to be tracked as well as to visualise the autonomous tracking within the various video streams.

Figure 1 visualises the main components of two cameras in a network of cameras. The middleware provides the connections between the single components on one hardware device and the data link between different devices. The image acquisition provides image data. The source can either be the image sensor or pre-recorded images locally stored on the camera. The computer vision component receives images from the image acquisition component and calculates and forwards the bounding box and the tracking confidence to the handover component.

The handover component manages the tracking process. Incoming requests from either the user via the user interface or from other cameras are processed by this component. If an object is about to leave the camera's field of view, this component initialises a search request (auction) on the other cameras. If a search request from another camera is received, this component initialises the image acquisition and the computer vision. After initialisation, the model to search

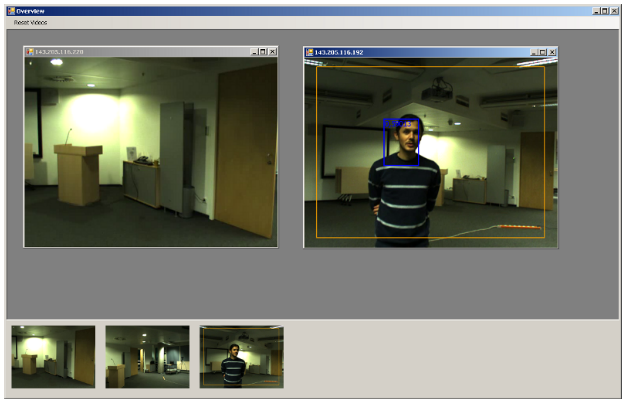

Fig. 2: Screenshot of the graphical user interface. The blue boundary marks the tracked object.

for is set and the search is started. The handover then waits until either the computer vision reports that the object was detected in the image source or a message is received from the requesting camera indicating that the search process for this specific model can be stopped. If the model was detected in the local image source the confidence, which is provided by the computer vision component, is returned to the requesting camera. The requesting camera collects all returned confidences and the camera with the highest confidence is selected to continue tracking. All other cameras are informed that they can stop to search for the object.

The current system is able to track a single object within a static camera network. The tracking process works in the network in a semi-automatic way. The tracking process is described as semi-automatic because the user has to select the object. After initialising it, the computer vision tasks run automatically. The handover of tracking responsibilities is automatically done as well. However, in the current implementation, no neighbourhood relations are taken into account which means that the offer to continue tracking is forwarded to all cameras in the network. Adding or removing of cameras during runtime is not supported within this setup.

To autonomously track a dedicated person or object in our system, a user has to select an object from a video stream at the client-side application. Afterwards, this object is being tracked automatically within the network of smart cameras. The end user receives a video stream on the client-side user interfaces, showing the field of view of the camera currently tracking the object. Figure 2 shows a screenshot of the user interface.

IV. Experiments

The experimental setup consists of three smart cameras with Linux operating system and a Windows workstation which hosts the user interface (cp. Figure 3). We use custom-built smart cameras from SLR Engineering which are equipped with an Intel Atom processor running at 1.6 GHz and an 100 MBit Ethernet interface. The camera has a CCD image sensor with a native resolution of 1360×1024 pixels. The resolution used

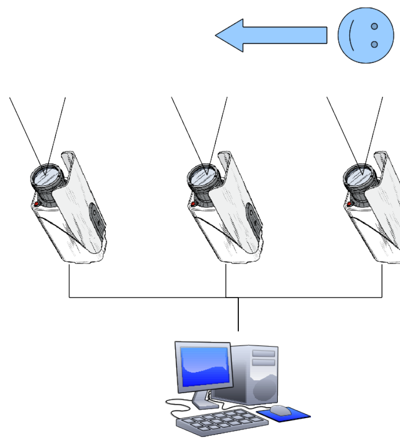

Fig. 3: Block diagram of the test setup. One moving object is observed by three smart cameras with partial overlapping field of view. The tracking results are visualised on the graphical user interface of the workstation.

for processing is 640×480 pixels.

The test scenario has a single object which is in the view of one camera and moves to the view of a different camera. The cameras have a partial overlapping field of view and no neighbourhood relations are provided to the cameras. The region of interest of each camera covers the whole image. If an object is lost (i.e., cannot be detected by the computer vision algorithm) or leaves the region of interest, an auction is initialized and the other cameras are requested to search for the object. The important steps of this scenario are listed below in their proper timely order.

Figure 4 shows the relevant part of the sequence diagram of the test setup. Initially, all cameras are active and transmit the current image to the user interface. In order to start tracking, an object must be marked by the user in the GUI (graphical user interface). The messages that lead to this starting situation are not visualised in the diagram. In the following we describe the key steps for tracking the object among three cameras.

At the beginning (1), cam1 receives an image with an object to track. The camera generates an object model from this image (2) and starts local tracking (3) this object. From this moment on, cam1 is responsible for tracking this object. The tracking results are returned to the user interface (4). At some point (5) the object leaves the field of view of cam1. This event is detected by the tracking algorithm of this camera which then initialises an auction for the object. Thus, the other cameras start with object detection given the object's model. In the current implementation, this command is executed as broadcast which means that all cameras receive this initialisation of the auction for the lost object.

All cameras are searching for the object (7). As soon as one camera detects the object in its field of view (8), a message is sent to the responsible camera (9) which includes information about the object that was found and the confidence about the detection. This message can be seen as a bid for the opened auction.

The responsible device (cam1) registers this bid and waits for a specified time for bids from other cameras (10). After the timeout (11) all bids are evaluated and the best bid is selected. The responsibility to track is then forwarded to the device with the highest bid (12). All other cameras are informed that the auction is over and they can stop searching for the object (13). The camera which is now responsible for tracking of the object (cam2) starts forwarding the tracking data to the user interface (14).

The camera that initiates the auction can use the bids as indicator for possible neighbouring cameras.

Table I describes the meaning of the messages that are visualized in Figure 4.

A. *Results*

This section summarises the delays and performance that is achieved with the current implementation of the system. The given data corresponds to the steps described in the previous section. Most of the timings of the system are defined by software. Other timing values depend on the input data. These have been measured.

Measured timing values:

- (2.) - 2.1)) Model generation : $\sim$ 130 ms
  This time is measured from the reception of the message 2.) till the model was generated 2.1).
- 3.) Start tracking : $\sim$ 15 ms
  This is the time that it took to set the model to the computer vision and start tracking.

Values defined by software:

- 4.) Track updates transmitted to the user interface: all 200 ms
  This is a continuous process that is executed every 200 ms.
- 5.) Model is about to get lost: 15×200 ms
  If the model cannot be tracked for 15 times, it is assumed that the object is lost.
- 8.) Model detected: maximum 100 ms after the object is seen by the camera
  The status of the detector is queried all 100 ms.
- 10.) Start of Timer
  The auction is started as soon as the first camera returns a "model detected" message.
- 11.) Timeout
  The auction is finished 1000 ms after the auction was started (10.)).

The tracking algorithm is the most critical part of the system as it is responsible for detecting and tracking within a single camera. At the moment a standard algorithm is used to test the handover mechanism. In real-world settings object detection and tracking is imperfect. To compensate misdetection at individual frames we currently initialize a handover only if the object could not be detected within 15 consecutive frames.

The current implementation of the tracking algorithm which is a standard algorithm without optimisations takes an average

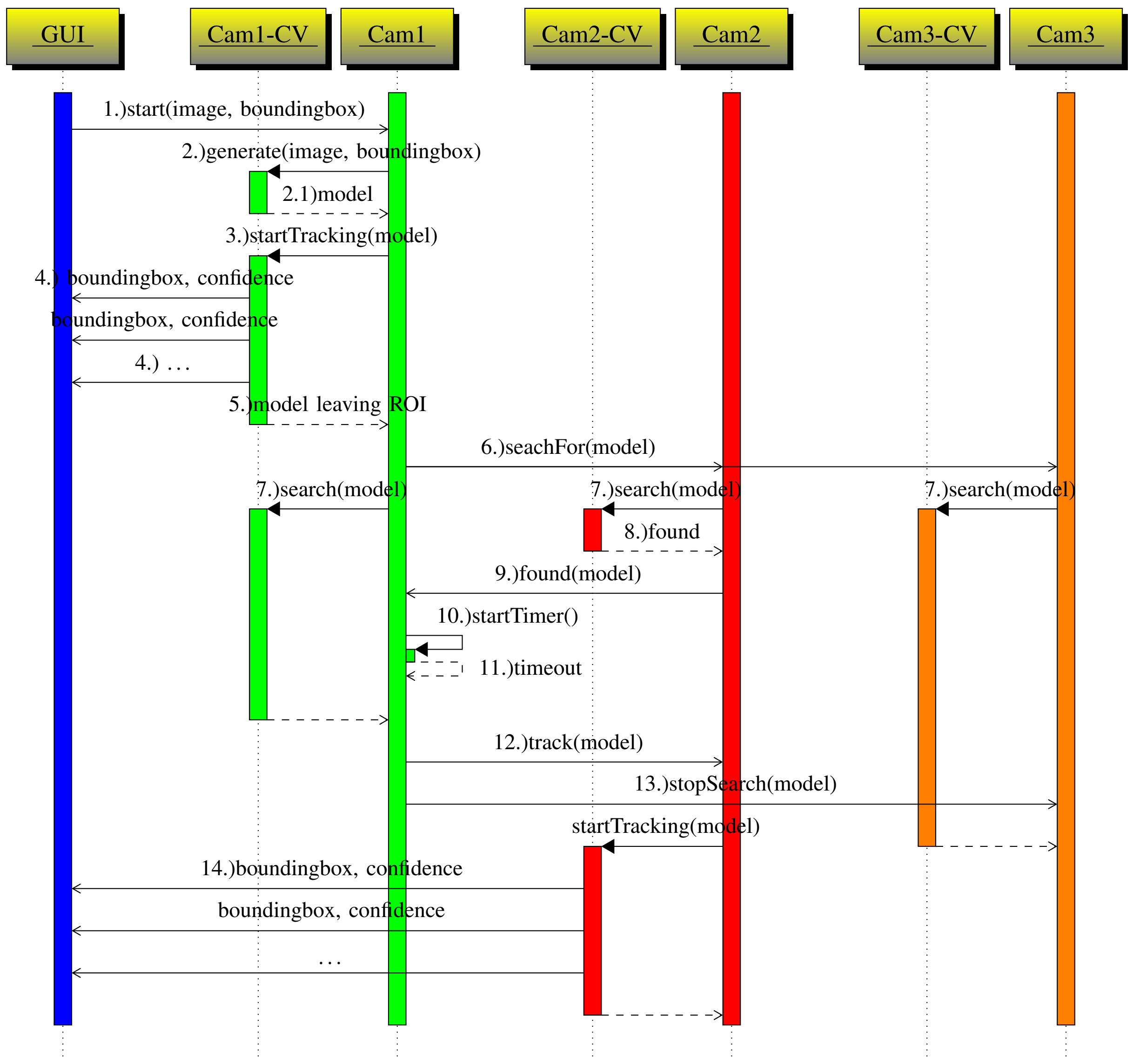

Fig. 4: Sequence diagram of test-setup

ID	Message	Description
1	start(image, boundingbox)	The Image where the object was marked and the bounding box (which marks the object) is transmitted to the camera which had this object in view.
2	generate(image, boundingbox)	A model is generated from the image and the provided bounding box.
3	startTracking(model)	If found, the model is tracked by the camera.
4	boundingbox, confidence	The output of the tracking algorithm (bounding box of object and confidence) is returned to the user interface. This interface visualises the detected object by drawing a blue rectangle around the object.
5	model leaving ROI	The object is about to leave the region of interest of the camera.
6	searchFor(model)	The handover process is initiated. This is done with a single UDP message which contains the model and a command to search for this model.
7	search(model)	All cameras search for the model
8	found	The model was found by one of the cameras
9	found(model)	The confidence is returned to the camera that initialised the handover.
10	startTimer()	Start of timer to finish the auction. After the first answer is received a timer is started that finishes the auction. During this time, all confidence messages belonging to the searched model are processed.
11	timeout	After the auction was finished, the camera that provided the message with the best confidence is selected to be the next camera to be responsible for tracking.
12	track(model)	The camera with the best confidence is informed that it should continue tracking the object.
13	stopSearch(model)	All other cameras are informed that the auction is over and they can stop searching
14	boundingbox, confidence	The other camera is now tracking and returning the tracking updates to the user interface.

TABLE I: Description of Messages in the sequence diagram

of 45 ms (with a standard deviation of 5 ms) to process a single image. This results in a framerate of approximately 20 fps.

V. Conclusion

Even though the tracking algorithm is not able to detect the object in each frame, the main focus of the system, the multi-camera tracking could still be shown. The available tracking results have been visualized in the user interface in form of a bounding box around the object and a number representing the confidence. If the object left the view of a camera, the tracking responsibility was automatically forwarded to the camera which had the best view of the object. After the handover was completed, the remaining cameras did no longer execute the detection algorithm and therefore saved resources.

The required resources are reduced to a minimum as each object is only tracked by one single camera. The used handover algorithm ensures that the camera that continues the tracking is the one that has the best view of the object.

Our prototype system shows potential for future work in a number of directions. First, an improved tracking algorithm would increase the overall tracking reliability. Second, the current implementation works with broadcast messages for the initialisation of the handover. The learned neighbourhood-relations can be used to prevent the broadcasting of messages and only transmit the detection request to the probable neighbours. Third, if a camera gets removed in the current system, it can occur that this camera was the one responsible for tracking. By adding redundancy to the system, this problem can be prevented. Other possible extensions are the tracking of multiple objects or to increase the tracking reliability by using multiple views of a single object for tracking. In this case single errors can be prevented as the information about the objects position is available redundant.

Acknowledgment

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement no 257906.

References

[1] B. Rinner, M. Jovanovic, and M. Quaritsch, “Embedded Middleware on Distributed Smart Cameras,” in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, 2007. ICASSP 2007.*, vol. 4, april 2007, pp. IV–1381 –IV–1384.

[2] C. H. Lin, W. Wolf, A. Dixon, X. Koutsoukos, and J. Sztipanovits, “Design and Implementation of Ubiquitous Smart Cameras,” in *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous and Thrustworthy Computing*, Taichung, Taiwan, Jun. 2006, pp. 32–39.

[3] M. Quaritsch, B. Rinner, and B. Strobl, “Improved Agent-Oriented Middleware for Distributed Smart Cameras,” in *Proceedings of the First ACM/IEEE International Conference on Distributed Smart Cameras ICDSC ’07*, Vienna, Austria, 2007, pp. 297–304.

[4] M. A. Patricio, O. Carbó, J.and Pérez, J. García, and J. M. Molina, “Multi-Agent Framework in Visual Sensor Networks,” *EURASIP Journal on Applied Signal Processing*, vol. 2007, no. 1, pp. 226–226, January 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1289187

[5] S. Fleck, F. Busch, and W. Straßer, “Adaptive Probabilistic Tracking Embedded in Smart Cameras for Distributed Surveillance in a 3D Model,” *EURASIP Journal on Embedded Systems*, vol. 2007, p. 17, 2007.

[6] D. C. Schmidt, “Middleware for real-time and embedded systems,” *Commun. ACM*, vol. 45, pp. 43–48, June 2002.

[7] ICE Middleware. [Online]. Available: http://www.zeroc.com/

[8] H. Rahnama, P. Kramaric, A. Sadeghian, and A. Shepard, “Self-adaptive middleware for the design of context-aware software applications in public transit systems,” in *Proceedings of the 13th International Conference on Ubiquitous Computing (UbiComp’11)*. New York, NY, USA: ACM, 2011, pp. 491–492.

[9] S. Hurtado, S. Sen, and R. Casallas, “Reusing legacy software in a self-adaptive middleware framework,” in *Proceedings of the International Workshop on Adaptive and Reflective Middleware*, ser. ARM ’11. New York, NY, USA: ACM, 2011, pp. 29–35.

[10] L. Esterle, P. R. Lewis, M. Bogdanski, B. Rinner, and X. Yao, “A Socio-Economic Approach to Online Vision Graph Generation and Handover in Distributed Smart Camera Networks,” in *Proceedings of the Fifth ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC 2011).* IEEE Press, 2011, pp. 1–6.

[11] V. Lepetit and P. Fua, “Monocular model-based 3D tracking of rigid objects,” *Found. Trends. Comput. Graph. Vis.*, vol. 1, no. 1, pp. 1–89, 2005. [Online]. Available: http://dx.doi.org/10.1561/0600000001

[12] M. J. Swain and D. H. Ballard, “Color indexing,” *International Journal of Computer Vision*, vol. 7, no. 1, pp. 11–32, Nov. 1991. [Online]. Available: http://dx.doi.org/10.1007/BF00130487

[13] F. Chang, “A linear-time component-labeling algorithm using contour tracing technique,” *Computer Vision and Image Understanding*, vol. 93, no. 2, pp. 206–220, Feb. 2004. [Online]. Available: http://dx.doi.org/10.1016/j.cviu.2003.09.002