Towards a Visual Sensor Network Built from Smartphones

Kristóf Csorba, Dávid Szalóki and Norbert Koszó
Department of Automation and Applied Informatics
Budapest University of Technology and Economics
Budapest, Hungary
{kristof.csorba, david.szaloki, norbert.koszo}@aut.bme.hu

***Abstract*—With the developments of the last years, smartphones became suitable to be used as smart cameras in a visual sensor network. They have the necessary cameras with an acceptable quality of lenses and CCD, the required processing power, and WiFi or Bluetooth interfaces for communication. This paper describes the architecture of the SMEyeL (Smart Mobile Eyes for Localization) system. It is designed to utilize both smartphones and PC-s as processing nodes of a visual sensor network. It aims to achieve high spatial and temporal resolution, energy consumption optimization via computation offloading and distributed processing. The SMEyeL project is open source: sourcecode and measurement data sets are available on the web. We present the detailed description of the architecture and exprimental results related to timing accuracy and localization precision.**

***Keywords*-smartphone; visual sensor network; motion tracking; computer vision; distributed processing**

I. Introduction

Smartphones are evolving rapidly in the last years. They allow access to highly integrated, sensor-rich embedded devices in the consumer price category. Especially if one is aiming at the application of smartphones that are some months behind the current trends, one gets the possibility to access a large amount of smart cameras with low prices and still good quality. For this reason, the Smart Mobile Eyes for Localization (SMEyeL) project aims to create a visual sensor network from smartphones. The system core is designed to be suitable for a wide range of computer vision based applications focusing on localization and tracking.

The implementation is based on Android smartphones, and Windows and Linux based desktop PC-s. The performance critical parts are written in C++ using the common OpenCV library [1][2]. As the primary application, we aim to create a motion tracking system capable to track a swarm of quadrocopters with high spatial and temporal resolution. Both of these are achieved by using serveral smartphones with highly overlapping fields of view and precise imaging timing. Fig. 1 shows a typical scenario where a quadrocopter is tracked using stationary and free moving smartphones held in a persons hand. Tracking results are presented in a tablet PC using augmented reality technology.

The system is currently in development phase and this paper presents the key aspects of the architecture. Beside the primary objective of high accuracy, the design emphasizes

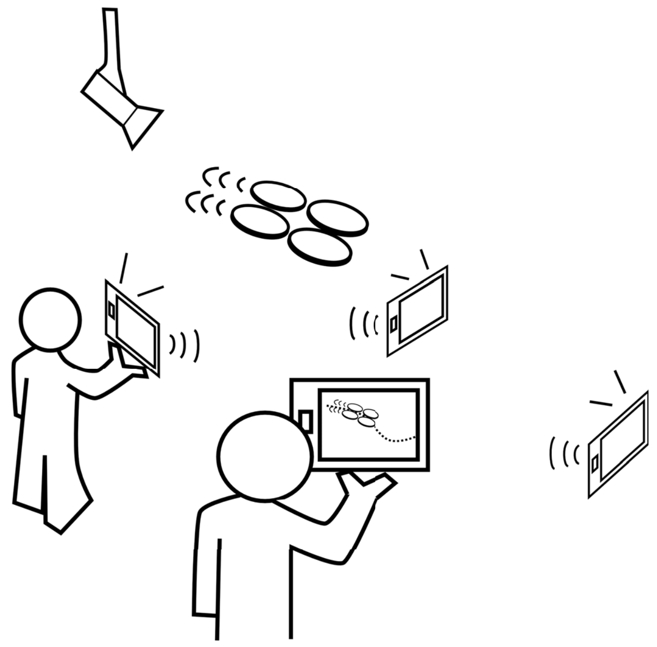
Fig. 1. Quadrocopter tracking and augmented reality application.

robust distributed processing, flexible load balancing and task assignment, and energy efficiency. This way, the tracking system will be reliable even if some smartphones experience technical problems, communication delays appear, or some phones simply loose the tracked object due to lighting conditions. As for example in a field application the smartphones cannot be charged continuously, energy efficiency is important. The time of maximal continuous operation can be increased by rotating the most energy-consuming tasks between the smartphones which are capable to perform it, or by suspending the operation of smartphones which are not necessary at the moment.

The SMEyeL project is open-source and can be downloaded[3] together with related papers, documentation and experimental datasets allowing full reproducability of the presented results.

The remaining part of the paper is organized as follows. First, section II summarizes related work. Section III describes the architecture of the SMEyeL system. First, it describes the general requirements the system needs to satisfy. Then, basic operations in the system are described, such as message handling and taking a picture. Finally, some advanced scenarios are described. After the architecture, section IV describes measurement results, mainly related to timing precision and delays caused by the overhead of the architecture. Finally, section V concludes the paper.

II. Related work

There are several main areas related to the work proposed in this paper: sensor networks, computer vision systems, CCTV surveillance systems, mobile peer-to-peer systems, and applications of motion tracking systems.

A. Sensor networks

Most sensor networks are built from relative cheap and simple nodes. The main reason is usually the low energy consumption, as sensor networks are often deployed into areas where the node batteries cannot be easily recharged. These systems are designed to perform some kind of environmental monitoring or tracking as long as possible. For example [4] presents a sensor network used to monitor coral reefs. The most common communication solution is a MESH network using radio links. In these networks, the various nodes do not have sufficient energy to communicate to every other node, only to their nearest neighbours, often only by broadcasting with limited range. Distant communication is performed via routing the messages from node to node. For instance [5] presents a relative simple protocol allowing heterogeneous devices to communicate in an IP-like environment, but designed for sensor network environment and simple devices. The topology of the links is often unknown and has to be automatically mapped prior to, or simultaneously with the data communication. Visual sensor networks are a special case where the real-time processing and the transfer of significant amount of information is very important. [6] discusses the information flow in such systems.

B. Computer vision systems with wired cameras

Computer vision systems are nowadays on a very advanced level. Real-time image processing is usually performed on powerful processors, often on dedicated video processors. The image sources are either cheap cameras, or expensive, fast and high-resolution cameras. The image sources are usually directly connected to the processing computer to minimize delay and to provide the necessary bandwidth for the high-resolution image transfer.

A typical multi-view application is the motion tracking [7][8] of sport events. [9] presents a system used for visual object tracking in soccer games. Other typical application is the tracking of moving objects from flying UAVs [10].

Motion tracking systems often face the problem of motion blur in the images. [11] presents a method for handling this in augmented reality applications. The method proposed in that paper can not only handle the motion blur of the tracked image patches, but it can also apply the estimated blur on the virtual objects appearing in the augmented reality application.

Special applications may need hard real-time or resource limited image processing. [12] describes a solution for this by adaptive setting of the resolution the images are processed at.

C. CCTV surveillance

CCTV surveillance systems are a rich research field of computer vision. Most common applications are related to human or traffic surveillance[13] either in private areas, or open areas like pedestrians on the streets[14]. These applications aim for movement traking of humans, for recognizing their actions, or for detecting dangerous situations like sick people or possible criminal actions. [15] presents a solution to understand dynamical properties of crowded scenes using sparse optical flows.

CCTV systems use external cameras deployed in large areas. The cameras have relative low overlapping in their fields-of-view (FoV) and the cameras themselves do not perform any image processing: all video streams are routed into the processing center.

Two important differences with SMEyeL are that in SMEyeL

- camera FoVs have high overlapping ratio to provide a very redundant view of the scene, and
- the intelligent camera nodes perform as much image processing as possible to minimize communication load.

D. Mobile P2P networks

Peer-to-peer networks are very robust and scaleable solutions for connecting a massive amount of nodes, which is one of the goals of our proposed architecture, too. Due to their distributed nature, these systems have a high tolerance towards disconnecting nodes, overload of nodes, or towards the appearance of new nodes. The system may be completely decentralized, where many nodes can substitute each other and they synchronize the roles between themselves without external interaction. Less decentralized solutions require a minimal central coordination for task distribution or connection establishment, but the main part of the operation is performed without this central control. An intermediate solution is to allow central coordination nodes to improve performance, but the system is capable to operate at its full functionality even with the absence of the central control. An example for this is the BitTorrent system, where the tracker servers significantly improve the content localization capabilities, but the system can operate using only the distributed hash table (DHT). Popular examples for BitTorrent clients for mobile phones are MobTorrent[16] and SymTorrent[17].

[18] presents a peer-to-peer network suitable for multi-camera sensor networks which makes it especially closely related to SMEyeL. It allows IP-based communication and utilization of a fully decentralized architecture. It supports a wide variaty of sensors like IP cameras, hardware sensors or

file sources. The communication may be built over peer-to-peer systems like Peer Channel in the Windows Communication Foundation (WCF) or Gnutella.

The architecture of SMEyeL requires a central server to keep track of the nodes, but this is only needed for establishing new connections. Data transfer and processing is done in a decentralized way to improve scalability and robustness.

E. Applications of related technologies

Applications of these systems have a very wide range, including motion tracking[19] like movie animation or sport motion analysis, augmented reality, mapping applications, or robotic navigation applications[20]. For example the solution in [21] uses flying robots to first localize simple sensor motes and then use them for localization and navigation.

There are also many systems which do not employ online processing: they only collect the high quality images during the measurement, and run the image processing later, offline. An example for this is Google Streetview[22].

The SMEyeL system also uses similar image processing techniques as motion tracking and augmented reality applications. The main difference is the image source, which is a set of smartphones connected via WiFi. Smartphone based tracking is proposed in [23] where cars are tracked using handheld smartphones.

III. SMEyeL Architecture

This section describes the architecture of SMEyeL. First, the basic components and requirements are introduced (subsection III-A), followed by the processing nodes and their communication (subsection III-B). After that, the basic functions are described, such as message handling (subsection III-C) and the handling if image sources (subsection III-D). Finally, time synchronization considerations (subsection III-E) and geometric data fusion (subsection III-F) are presented.

A. Components and requirements

A SMEyeL system may contain several of the following hardware components:

- Smartphone: the architecture is designed to employ smartphones as intelligent cameras. The phones may take only pictures, stream video, or perform image processing as well. Additionally, further data processing tasks may be assigned to the smartphones as well, especially if there are no other devices but smartphones. Currently, the target smartphone platform is Android.
- Desktop PC: one or more desktop computers may be added to the system to provide coordination and fast data processing capabilities. Desktop computers have significantly higher processing power, and are not limited by energy consumption as the smartphones (if powered from battery). The SMEyeL implementation is portable to Windows and Linux based systems.
- Additional embedded systems: using embedded Linux, additional embedded hardware may be added to the system. This may include for example a swarm of quadrocopters carrying onboard cameras.

There are several requirements the SMEyeL architecture must satisfy. Many possible applications can be built over the core system, but the following requirements are independent of the specific application, and are almost always necessary:

- Speed: being a hard-realtime computer vision system, the data transfer between the processing elements has to have high bandwidth and low delay. If the image processing is performed at the place of capture, bandwidth may not be so critical. In these cases, the small data messages containing the calculated subresults have to be transferred with low delay.
- Robustness: the system has to be robust against disconnected nodes and increased communication delays to some nodes. For example the motion tracking of quadrocopters may not be suspended due to the loss of one single camera. Similarly, if the communication with a smartphone is disturbed, the system has to be able to detect it and work without that node. Even if it means the reassignment of some tasks in the network.
- Dynamically reconfigurability: related to the robustness mentioned before, the system has to be capable to reassign tasks between the nodes. Energy efficiency may need this feature as well, as balancing the energy consumption of the smartphones is essential for maximizing the timespan of continuous operation.
- Cross-platform portability: the system core has to be portable to at least Windows and Linux based systems. This is important because several different hardware platforms may need to cooperate in a specific application.

Beyond these, several additional constraints may be required by specific applications built over the core SMEyeL system.

B. Processors and Communication

In the proposed SMEyeL architecture, the processors have a key importance. These are the processing nodes of the distributed system which perform any kind of operation. They communicate with each other using subscription based multicast communication, so that every processor is subscribed for the messages carrying its input data, and sends its output data in the form of similar messages. Subscription is handled by filters defining various conditions whether the given processor is interested in a given message or not. The communication medium inside a hardware device is usually via shared memory and direct function calls. Inter-device communication is performed via TCP/IP sockets, and we keep the possibility of UDP multicast open for later extensions.

This architecture allows fast intra-device data flow, but it also supports the transparent distribution of the processing nodes on multiple devices, like several smartphones, desktop computers, and even cloud services.

C. Messages and subscription

The messages use the common JSON format, as it is portable, easy to process, and easy to extend in terms of data content. The protocoll allows the extension of the JSON part with a successive binary part, mainly used for JPEG image transfer. This is reasonable as transferring an image inside the JSON part would require additional, time and space consuming encoding like base64 which would not be suitable in the current application. Fig. 2 presents the format of a SMEyeL message containing a JPEG file as a binary attachment. If there is a binary part after the JSON part, the size of the binary part is given in the JSON header.

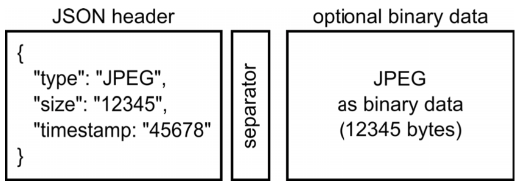

Fig. 2. Example message structure. Most data are transferred in JSON format with a type identifier in the beginning. Auxiliary binary data (like a JPEG file in this case) may be attached after the JSON part, separated by a zero character. Size of the binary attachment is given in the JSON part.

As mentioned before, intra-device communication uses direct function calls. In this case, no JSON serialization is performed, and images (in JPEG or in OpenCV's Mat format) are transferred simply by sharing their memory location. This is essential due to performance considerations of the real-time nature of the system.

The Communication Router (CR) is the essential component of the communication. It consists of a central server and proxies in all the hardware devices. The data communication if performed among the proxies: every proxy stores every subscription and forwards the incoming messages to every recipient's proxy automatically. The central server is only needed to keep a list of all the proxies and to broadcast the changes in the subscriptions to all the proxies. This way, the server does not cause a bottleneck in the communication. The initialization of the Communication Router (CR) is visualized in Fig. 3. Every processor in the system needs to get a unique ProcessorID which is assigned by its local proxy. For this reason, the proxies get a unqiue ProcessorID interval from the central server during the initialization.

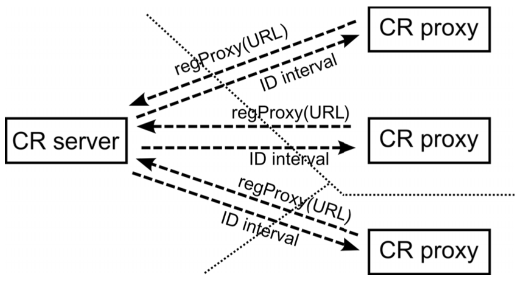

Fig. 3. Communication router internal initialization. All proxies notify the server, so that every further subscription request can be sent to all of them. Beside this, they receive an interval of ProcessorID values they will be allowed to distribute among their processors. The boundaries between different hardware devices are indicated by the dotted lines. The dashed arrows stand for TCP/IP based inter-device communication.

A key element in message handling is the subscription process. If a processor wants to receive all messages corresponding to a given filter, it sends a subscription to the CR including a subscription filter. This filter may select messages of a given type (like JPEG images or 3D locations), messages from a given sender (like Camera 2), or messages sent to a given target (like Camera 1). It should be noted that a target processor may be addressed in a message, but the delivery is still controlled by the subscriptions, so that for example a logging processor can still subscribe to every message. The steps of a subscription are outlined in Fig. 4.

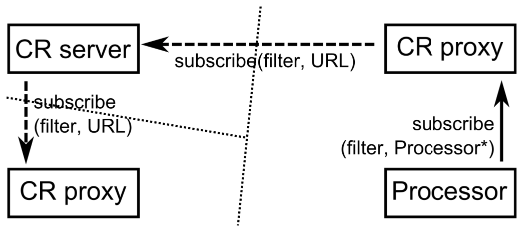

Fig. 4. Subscription handling. The identity and the subscription filter of the subscribing processor is stored in its local CR Proxy. Other proxies receive the subscription (filter and URL of target proxy) through the CR Server. Solid arrows stand for intra-device function calls, Processor* stands for a C++ pointer to a processor, and URL is the URL the proxy can be accessed with. Subscriptions inside the devices are managed by their local proxies, so that the other proxies do not need to address remote processors, only their proxies.

After the processors have subscribed themselves for the required messages, the transfer of a specific message is shown in Fig. 5. Inside the local proxy, the message is transferred via direct function call using shared memory for image information, so there is no need to copy it. Between proxies, the message is serialized and sent via TCP socket. The CR Server does not participate in the message transfer. Every CR Proxy sends the messages to other proxies according to their subscriptions.

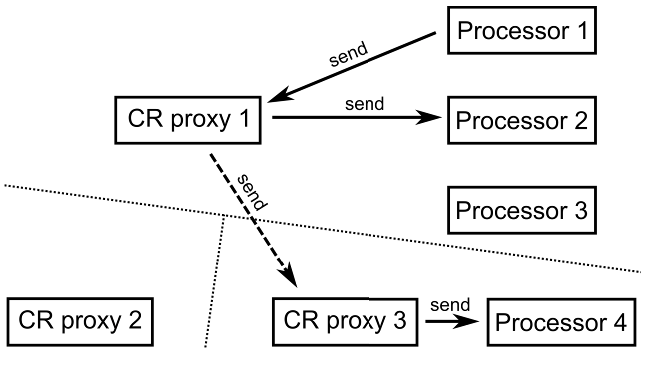

Fig. 5. Communication router message transfer. The transfer medium is direct function call and shared memory (solid arrows), or TCP/IP socket with JSON and binary serialization (dashed arrows). In this example, processor 3 and processors connected to proxy 2 are not subscribed for the message.

Even the simplest scenarios require several types of messages. The list of available message types are the following:

- *Trigger Photo*: Command to take a photo and send it. The answer is an *Image* message.
- *Image*: Message containing an image in binary format. This may be a JPEG image taken by a smartphone, or an image in OpenCV Mat format.
- *Trigger Chessboard*: Command to take a picture and find a chessboard in it. If successful, the answer is a *Chessboard Data* message.
- *Chessboard Data*: Detection result of a chessboard in the image. It contains the pixel coordinates of the chessboard corners.
- *Trigger Detection*: Command to start a detection. For example to take a photo and find the markers in it.
- *Processor Enumeration Request*: Request for all processors to respond with their ID and their type. Used for instance to enumerate all image sources.
- *Processor Enumeration Response*: Response containing the ID and type of the sender processor.
- *No More Data*: Processor indicates that no more data will be available. For example it reads from AVI file and has reached its end.
- *Aquisition Done*: This message is used to indicate that the processing of received geometry information can be started as there will be no further information in the current time frame. (Detection information received after this will not be incorporated into the results for the current frame.)
- *Point2D, Point2DList, Ray2D, Ray, Point3D, Point3DList, Marker2D, Marker3D*: Messages containing 2D or 3D geometry information like directions of markers or arbitray points, 3D locations of objects etc.

D. Image source handling

In the SMEyeL system we have several image source processors like still image cameras, video camera, and AVI file readers. The data provided by these processors can be raw images or information resulting from image processing like marker locations. These data can be collected in two essentially different ways: via streaming and via polling. If the sources send their results automatically, the processor performing the data fusion only needs to subscribe for these types of messages and it will become everything needed. A more complex situation arises if – for instance – we want to poll the smartphones to take high-resolution still images in a rotating schedule allowing better time-resolution (see Fig. 8 later). In this case, an image source scheduler (a distinct processor node) is required.

In order to send the image sources a request for pictures, first we need to get a list of all image sources. This is retrieved using a broadcast request (*Processor Enumeration Request*) which every processor replies to (*Processor Enumeration Response*). (Technically, this broadcast request is a message type all processors have to be subscribed for.) The image source scheduler can then filter out the image sources form the replies, as the replies contain the type of the answering processor.

As soon as the list of image sources is available, the scheduler can trigger these processors after each other to take pictures. As a global system time is not necessarily available for the smartphones, the scheduler keeps track of the time offsets of the various processors. The time synchronization is described in subsection III-E in details. The procedure of taking a still image is presented in Fig. 6.

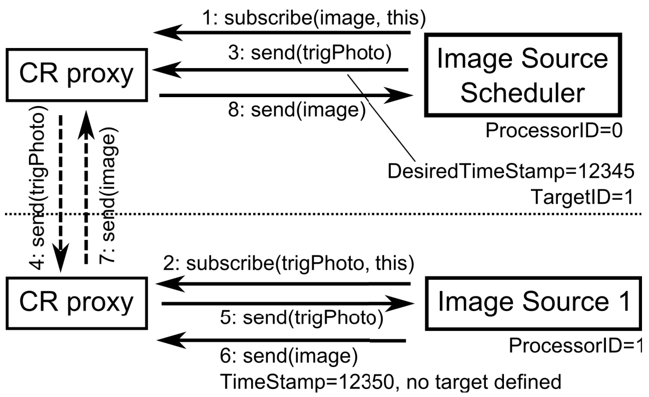

Fig. 6. Procedure for making a still image. The image source may be asked to take a still image in a given time, defined w.r.t its own clock. If it is not defined, the image is taken as soon as possible. The image is sent as a response, without any image processing. The subscription steps 1–2 are performed during initialization. There is no need for a TargetID in the response, but the image is accompanied by a timestamp. The image is transferred in steps 6 and 8 via shared memory, and in step 7 in a binary serialized form.

The processors responsible for 2D-3D transformations are the Geometry Processors discussed in subsection III-F. Geometry processors are capable to calculate the 3D locations of markers using the 2D observations and the calibrated external parameters (location and orientation) of the cameras. This processing should be performed after every image source has sent its results. As the scheduler may trigger the image sources in many ways, it also has the responsibility to inform the remaining processors that the data gathered in the current timeframe should be processed, and no more information should be waited for. This is indicated by the *Aquisition Done* message mentioned earlier. An overview of the synchronized image taking process is presented in Fig. 8.

E. Time synchronization

Precise tracking requires precise timestamps for the observations. The SMEyeL system does not assume that the clock of the smartphones can be synchronized via NTP (Network Time Protocol), GPS or similar technology. The main reasons are WiFi accesses and indoor applications. Instead of this, a special time sync beacon is used. This is essentially a hardware showing a high-resolution clock. The image sources make a photo of this beacon in the calibration phase and send the image together with a timestamp w.r.t. their own internal clocks. The exact time of the picture can be read from the image itself (where the high-resolution clock is visible), and comparing it with the timestamp, the time offset of the image source can be calculated.

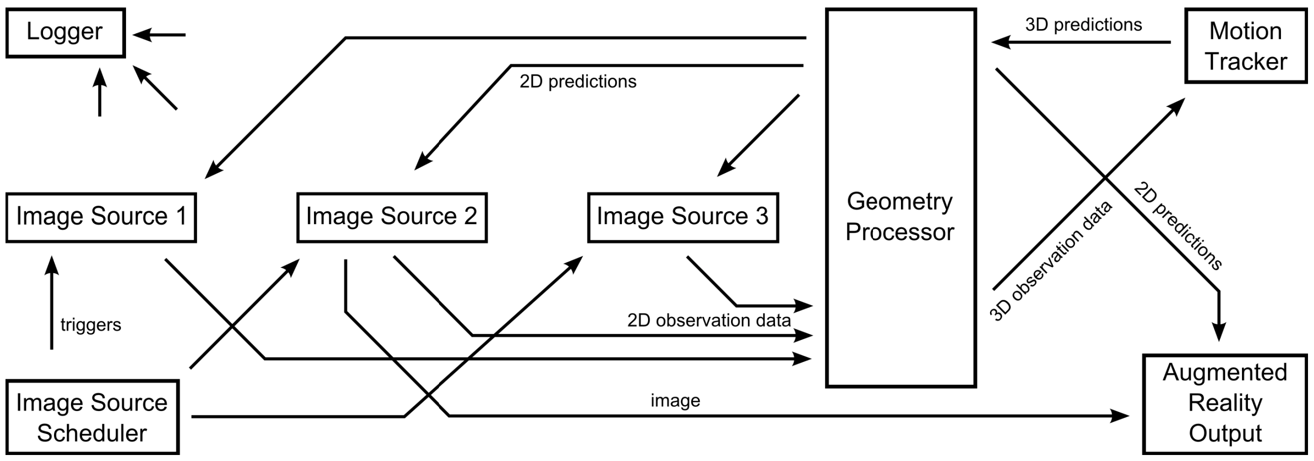

Fig. 7. Overview of example scenario. The Image Source Scheduler sends the periodic triggers to the Image Sources. These perform the image processing, so they send 2D observation data to the Geometry Processor which transforms them into 3D world coordinates. The 3D observation data are processed by the Motion Tracker which provides 3D prediction data. These are transformed into 2D for every image source by the Geometry Processor. The Augmented Reality Output shows the user the images of one of the sources, with the prediction data overlayed. The Logger logs every event, so it is subscribed to every message type.

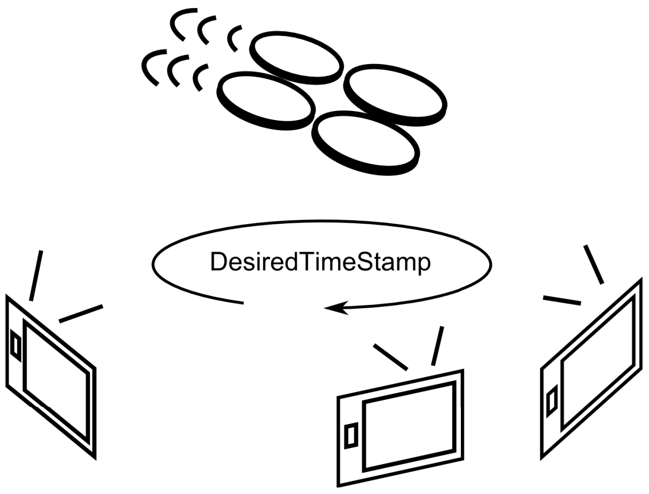

Fig. 8. Overview of scenario with smartphones as synchronized photo sources. The *Image Source Scheduler* asks every image source to take a picture in a given time. The resulting images are sent back as response for later image processing. The clock of the phones is not necessarily synchronized, so the scheduler needs to keep track of the clock offsets of the various image sources.

F. Geometry Processor

Beside the already mentioned image source and image source scheduler processors, a Geometry Processor is an essential component of a SMEyeL application. As mentioned before, this is the processor responsible to derive 3D information from the 2D observations. It receives the 2D observations from the image processing (performed by the smartphones or by other processors), and merges these into 3D observations. This requires the tracking of camera external parameters, that is, the coordinate system bases of the various cameras. As all processors but the geometry processor work either in 2D image coordinates or 3D world coordinates, the coordinate system base tracking is performed entirely inside the geometry processor.

If all the cameras in the scenario are stationary, the Geometry Processor may perform the camera calibration only in the beginning. If there are mobile cameras (moving entirely free or still satisfying some constraints), the Geometry Processor is responsible to track the changes of these cameras external parameters and utilize their observation results accordingly. This case is presented in Fig. 9.

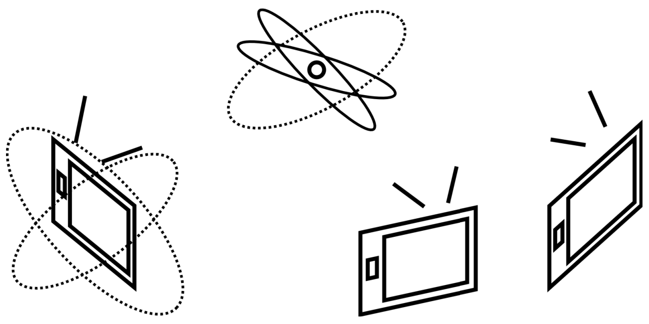

Fig. 9. Overview of scenario with free moving camera. If not all cameras are stationary, the moving cameras need continuous re-calibration as their external parameters (position and orientation) is changing. This is mainly done by using the stationary cameras to localize some easy-to-detect points in the scenario, and then use these to calculate the moving cameras external parameters.

The overview of a SMEyeL system scenario is presented in Fig. 7.

IV. Measurement results

Two major measurement setups were used to evaluate the capabilities of the SMEyeL system. The first was designed to investigate the localization precision of a marker based, multi-camera system. The second aims at the measurement of timing precision with remote controlled Android based smartphones.

In order to measure the localization precision of the system, 3 PS3Eye USB cameras were connected to a PC and were used for tracking a marker moved by an industrial standard Mitsubishi MELFA robotic arm. Ps3Eye cameras are considered suitable for image processing due to low distortions. In the current measurement setup, this makes them suitable to measure the upper limits of the localization system without the limitations of smartphone cameras having very different qualities. The robot has a positioning error of 0.05 mm. The marker was moved into 27 points of a grid, and the images of the three cameras were used to triangulate the location of these points. The tracking results are shown in Fig. 10.

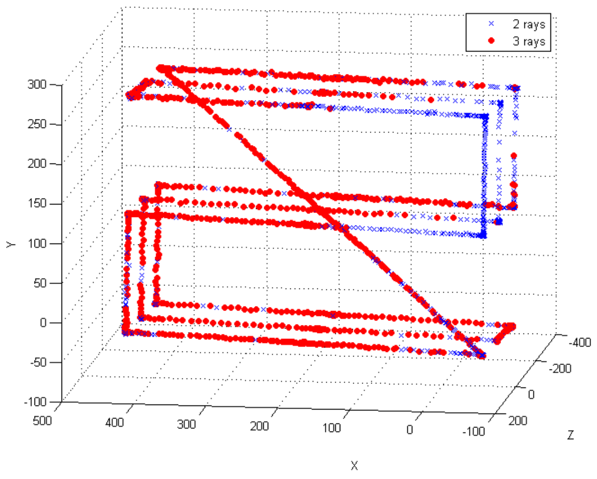

Fig. 10. Localization measurement results with fixed cameras. In this measurement, a robotic arm moves a marker into 27 points in a grid, and 3 cameras are calculating its relative direction in every frame. After the calculation of the 3D locations, the system achieved a localization variation of 1-2 mm.

The localization error was measured in means of standard deviation of the distances between the marker locations, as these were known exactly. The standard deviation of the distances was around 1 mm which is in the magnitude of the pixel quantization error of a 56° field-of-view camera from an observation distance of 1.5-2 meters. We concluded that the multi-view image processing system is suitable for the needs of the SMEyeL system.

In the second major measurement, an Android based smartphone was remotely controlled and asked to take pictures in given times and send them to a PC via WiFi. The most important question was the precision the timestamp of the image can be defined with.

Fig. 11 shows the work phases of taking an image. The elapsed times of the various phases are presented in Table I.

- First, the command for taking an image is sent to the smartphone. This contains the desired timestamp the image should be taken at. The required time is mainly a function of the WiFi speed.
- After receiving the command, the JSON format has to be parsed and evaluated. This does not take significant time compared to the other tasks.
- The waiting phase waits until the current time of the phone arrives to the desired timestamp. If we know the time required to initialize the photo taking, we can set the wait phase to finish before the desired timestamp. This way, we will be able to precisely schedule the exposition of the image. In the current measurement, the most important question is the standard deviation of the time between the desired timestamp and the start of the image taking.
- Taking a photo required significant time. The measurement was performed in constant lighting conditions to avoid the influence of the exposure time.
- The OnShutter event of the Android platform is called immediately after the exposition to allow accurate timing. This is when the timestamp of the image is saved.
- After the OnShutter event, the image is processed and compressed into JPEG format. This has also significant time requirement, but it does not influence the exact timing for the desired timestamp anymore.
- After the JPEG image is created, the response is assembled to be sent through the network.
- Sending the response consists of sending the JSON header with approximately constant size and the JPEG image with variable size. (During the measurement the image sizes were between 650-750 KB.) In the table of the results we provide the elapsed time of sending of the header and the JPEG file separately.

The time measurement is essential in this measurement. We used the hardware level performance counters of the smartphone which can be accessed in a platform independent way through the OpenCV library. The tick frequency we used for the timestamps was $10^9 Hz$ as reported by OpenCV. The timestamps were stored with a precision of $1 \mu s$.

The most important timing measurement results with respect to applications asking smartphones to take a picture in a given point of time, are the following:

- We can schedule the start of picture taking with a standard deviation of 0.02 ms.
- The turnaround time from asking for a picture until receiving it is around 1800 ms from which the capture takes approx. 527 ms, and the JPEG compression is around 690 ms. It should be noted that these values may change due to the exposition time and image content.

V. Conclusion

This paper presented the architecture of the Smart Mobile Eyes for Localization (SMEyeL) system. This distributed visual sensor network architecture uses smartphones as smart cameras and provides a highly extensible communication and data processing framework. The central component of the architecture is the communication router which realizes a subscription based message transfer system. All the image aquisition, measurement, and data processing functionalities are implemented in processor nodes connected by this communication router. The paper presented some specific examples

TABLE I
EXECUTION TIME OF REMOTE IMAGE TAKING ON ANDROID PLATFORM

Operation	Average time (ms)	Standard deviation	Comment
Receiving command	47.75	9.75	Network communication
Preparing	7.62	5.22	Parsing the command
Taking picture	527.31	3.55	Android retrieves image from camera
Post-processing picture	690.89	10.19	JPEG compression
Post-processing	57.32	11.96	Assembly of response
Sending JSON header	3.89	4.51	Network communication
Sending JPEG image	657.42	196.38	Network communication
All except waiting and communication	1173.16	359.43	Limits FPS of local image processing
All except waiting	1821.45	589.22	Limits FPS
Delay of taking picture	0.15	0.02	Precision of picture taking w.r.t. desired timestamp

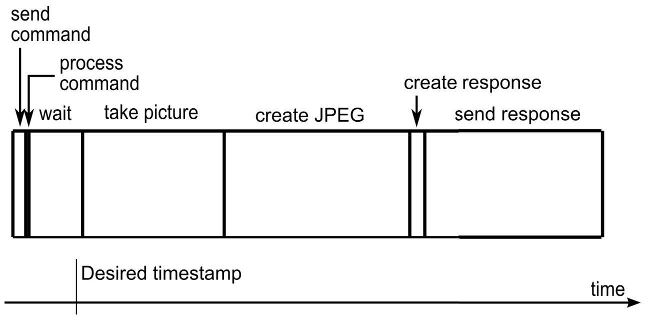

Fig. 11. Timing accuracy of image taking

for the realization of common scenarios and measurement results on the performance of the basic functions of the framework.

ACKNOWLEDGMENT

This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013) organized by VIKING Zrt. Balatonfüred.

The authors would like to thank Milán Tenk for his help with the implementation of the measurement environment.

REFERENCES

[1] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[2] G. Bradski and A. Kaehler, *Learning OpenCV*. O'Reilly Media Inc., 2008. [Online]. Available: http://oreilly.com/catalog/9780596516130

[3] Smart Mobile Eyes for Localization (SMEyeL). [Online]. Available: https://www.aut.bme.hu/Pages/ResearchEn/SMEyeL/Overview

[4] M. Bromage, K. Obraczka, and D. Potts, "SEA-LABS: A wireless sensor network for sustained monitoring of coral reefs," in *NETWORKING*, ser. Lecture Notes in Computer Science, I. F. Akyildiz, R. Sivakumar, E. Ekici, J. C. de Oliveira, and J. McNair, Eds., vol. 4479. Springer, 2007, pp. 1132–1135.

[5] I. Solis, K. Obraczka, and J. Marcos, "FLIP: a flexible protocol for efficient communication between heterogeneous devices," in *ISCC*. IEEE Computer Society, 2001, pp. 100–106. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/ISCC.2001.935361

[6] K. Obraczka and R. Manduchi, "Managing the information flow in visual sensor networks," Apr. 04 2003. [Online]. Available: http://citeseer.ist.psu.edu/665709.html; http://www.cse.ucsc.edu/ manduchi/Papers/178-obraczka.pdf

[7] O. Zoidi, N. Nikolaidis, and I. Pitas, "Exploiting disparity information in visual object tracking," in *3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON), 2012*, 2012, pp. 1–4.

[8] Q. Wang, A. Lobzhanidze, H. Jang, W. Zeng, and Y. Shang, "Video based real-world remote target tracking on smartphones," in *Multimedia and Expo (ICME), 2012 IEEE International Conference on*, 2012, pp. 693–698.

[9] H. Li and M. Flierl, "Sift-based multi-view cooperative tracking for soccer video," in *ICASSP*. IEEE, 2012, pp. 1001–1004. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6268628

[10] M. Teutsch and W. Krüger, "Detection, segmentation, and tracking of moving objects in UAV videos," in *AVSS*. IEEE Computer Society, 2012, pp. 313–318. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6327837

[11] Y. Park, V. Lepetit, and W. Woo, "Handling motion-blur in 3D tracking and rendering for augmented reality," *IEEE Trans. Vis. Comput. Graph*, vol. 18, no. 9, pp. 1449–1459, 2012. [Online]. Available: http://dx.doi.org/10.1109/TVCG.2011.158

[12] C. Xie, J. Tan, J. Zhang, Y. Bu, and J. Xie, "Adaptive bandwidth object tracking using sparse approximation," in *Computer Science and Automation Engineering (CSAE), 2012 IEEE International Conference on*, vol. 1, 2012, pp. 618–622.

[13] W. Nie, A. Liu, and Y. Su, "Multiple person tracking by spatiotemporal tracklet association," in *AVSS*. IEEE Computer Society, 2012, pp. 481–486. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6327837

[14] M. Mirabi and S. Javadi, "People tracking in outdoor environment using kalman filter," in *Intelligent Systems, Modelling and Simulation (ISMS), 2012 Third International Conference on*, 2012, pp. 303–307.

[15] V. Lasdas, R. Timofte, and L. J. V. Gool, "Non-parametric motion-priors for flow understanding," in *WACV*. IEEE, 2012, pp. 417–424. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6156698

[16] P. Ekler, J. K. Nurminen, and A. Kiss, "Experiences of implementing bittorrent on java me platform," 2008.

[17] I. Kelnyi and B. Forstner, *SymTorrent and GridTorrent: Developing BitTorrent Clients on the Symbian Platform*. John Wiley & Sons, Ltd, 2009, pp. 101–142. [Online]. Available: http://dx.doi.org/10.1002/9780470747889.ch7

[18] P. Saastamoinen, S. Huttunen, V. Takala, M. Heikkila, and J. Heikkila, "Scallop: An open peer-to-peer framework for distributed sensor networks," in *International Conference on Distributed Smart Cameras*, 2008, pp. 1–9. [Online]. Available: http://dx.doi.org/10.1109/ICDSC.2008.4635712

[19] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, no. 4, dec 2006. [Online]. Available: http://doi.acm.org/10.1145/1177352.1177355

[20] D. Kiss and G. Tevesz, "Advanced dynamic window based navigation approach using model predictive control," in *Methods and Models in Automation and Robotics (MMAR), 2012 17th International Conference on*, 2012, pp. 148–153.

[21] P. I. Corke, R. A. Peterson, and D. Rus, "Localization and navigation assisted by networked cooperating sensors and robots," *I. J. Robotic Res*, vol. 24, no. 9, pp. 771–786, 2005. [Online]. Available: http://dx.doi.org/10.1177/0278364905057118

[22] Google Street View. [Online]. Available: www.google.com/streetview

[23] Q. Wang, A. Lobzhanidze, H. I. Jang, W. Zeng, and Y. Shang, "Video based real-world remote target tracking on smartphones," in *ICME*. IEEE, 2012, pp. 693–698. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6297631