FireFly Mosaic: A Vision-Enabled Wireless Sensor Networking System

Anthony Rowe Dhiraj Goel Raj Rajkumar
Dept. of Electrical & Computer Engineering
Carnegie Mellon University, U.S.A.
{agr, dgoel, raj}@ece.cmu.edu

***Abstract*— With the advent of CMOS cameras, it is now possible to make compact, cheap and low-power image sensors capable of on-board image processing. These embedded vision sensors provide a rich new sensing modality enabling new classes of wireless sensor networking applications. In order to build these applications, system designers need to overcome challanges associated with limited bandwith, limited power, group coordination and fusing of multiple camera views with various other sensory inputs. Real-time properties must be upheld if multiple vision sensors are to process data, communicate with each other and make a group decision before the measured environmental feature changes. In this paper, we present FireFly Mosaic, a wireless sensor network image processing framework with operating system, networking and image processing primitives that assist in the development of distributed vision-sensing tasks. Each FireFly Mosaic wireless camera consists of a FireFly [1] node coupled with a CMUcam3 [2] embedded vision processor. The FireFly nodes run the Nano-RK [3] real-time operating system and communicate using the RT-Link [4] collision-free TDMA link protocol. Using FireFly Mosaic, we demonstrate an assisted living application capable of fusing multiple cameras with overlapping views to discover and monitor daily activities in a home. Using this application, we show how an integrated platform with support for time synchronization, a collision-free TDMA link layer, an underlying RTOS and an interface to an embedded vision sensor provides a stable framework for distributed real-time vision processing. To the best of our knowledge, this is the first wireless sensor networking system to integrate multiple coordinating cameras performing local processing.**

I. Introduction

Wireless sensor networks (WSNs) provide a simple-to-deploy and versatile platform for sensing and interacting with physical environments. These devices can support multi-hop communication forming mesh networks capable of self-configuration, self-healing and automatic management. These properties make WSNs suitable for various monitoring and actuating application domains such as industrial control, surveillance, asset tracking and assisted living. Current sensor networks focus on measuring low-bandwidth physical properties such as light, sound, temperature, motion and humidity. The inclusion of image processing sensor nodes provides a rich new information source capable of enhancing current applications as well as enabling new ones. More so than with single camera systems, real-time properties must be upheld if multiple vision sensors are to process data, communicate with each other and make a group decision before the measured environmental feature changes. In this paper, we present FireFly Mosaic, a real-time distributed image processing infrastructure built upon the FireFly wireless sensor networking platform.

Distributed vision processing in a sensor network has three main advantages over a stand-alone vision system. Multiple cameras have greater coverage of a scene, can address difficulties associated with view obstruction, and can increase decision confidence based on mutual information. FireFly Mosaic is a framework with supporting hardware platform that adds vision processing capabilities to the FireFly [1] sensor nodes with the addition of the CMUcam3 [2] embedded vision processor.

The FireFly platform is designed around tight global time synchronization which enables collision-free, energy-efficient TDMA-based communication with bounded latencies. Each node runs Nano-RK [3], a real-time resource centric operating system, that provides hooks for globally synchronized task processing. This framework enables capturing synchronous images as well as scheduling appropriate communications patterns while meeting system timing constraints. With deadline guarantees in place, it is possible to use the bounded time between consecutive frames to provide video information about the velocity of objects in the scene.

This paper discusses the system design principles that aid in the deployment of real-time vision-enabled sensor networks. Through an example assisted living application, we show how to apply these principles in the form of FireFly Mosaic primitives. We describe the system hardware, software and various operating system, networking and image processing techniques which can be extended to develop future vision based sensor networking applications.

We now describe an assisted living application built using the core FireFly Mosaic designed primitives. According to the 2000 US census [5], more than 10 million elderly over the age of 65 live at home alone. Projections estimate that the elderly population will more than double by the year 2050. Modern lifestyles have made it difficult to stay at home and care for family members. Both from an economic and a quality-of-living perspective our elders wish to remain in their homes as long as possible. This brings the demand for in-home monitoring systems that can provide accurate feedback on daily activities, detect if family members are in distress and at the same time maintain an appropriate level of privacy. Though policy is very important with respect to camera systems, it is outside the scope of this paper.

We illustrate an application using FireFly Mosaic to monitor people's daily activities in the home. The system automatically combines information extracted from multiple

Fig. 1. Eight vision-enabled sensor nodes used to perform distributed activity clustering in an active apartment. The node on the left side of the image is a standalone FireFly node that can be used to forward messages of connect the network to a PC.

overlapping cameras to recognize various regions in the house where particular activities frequently occur. Examples of such activities include washing dishes, preparing food, eating dinner, sitting on the couch or sleeping.

Once these daily activity clusters are defined, the system builds a model which tracks the duration and transition frequency between various tasks. This information can be monitored over time to detect changes in behavior, or it can be used to give contextual clues to other in-home systems. For example, a fall detection system could use this type of context information to lower its probability of triggering if the user is sleeping or has a visitor in a nearby location. We show a working prototype of this system running on eight battery-operated embedded vision cameras in an active apartment.

The organization of this paper is as follows. Section II discusses related work. Section III presents FireFly Mosaic, our vision-enabled sensor network platform. Section IV evaluates various peformance aspects of this system. Section V discusses the operation of our activity clustering algorithm and Section VI provides concluding remarks.

II. Related Work

Link and network support for real-time communications over sensor networks has attracted much attention in recent years. Constraints on computing power, bandwidth, memory and energy supply in sensor networks make the problem of delivering timeliness guarantees across multiple hops especially challenging. In [10], the authors present the capacity bounds on how much real-time data a sensor network can transfer by the imposed deadlines. They derive a sufficient schedulability condition for a class of fixed-priority packet scheduling algorithms and also provide a theoretical basis for capacity planning. Several media access schemes have been proposed for real-time communication over sensor networks, with the goal of bounding the end-to-end delay. In [9], the authors present velocity-monotonic scheduling that accounts for both time and distance in large-scale networks. Simulation studies show that such a scheme is effective in minimizing the deadline-miss ratios in multi-hop sensor networks. [8] and [11] present simulation studies on resource reservation and routing protocols for applications with end-to-end timeliness constraints. [4] shows how TDMA can facilitate bounded latency in communication given an appropriate schedule. We build upon this work by extending the spanning tree aggregation-centric scheduling, through the use of camera topology information, to support efficient local group communication as well.

Various vision sensor nodes have been developed with sensor networking in mind. However, these efforts have focused on the development of image processors alone and little is said about their integration with the sensor network. The Stanford MeshEye [6] system was designed for use in low-power sensor networks. The design uses two different sets of image sensors, a low resolution pair of sensors is used to wake the device in the presence of motion, while the second VGA CMOS camera performs image processing. The UCLA Cyclops [7] camera uses an 8-bit microprocessor and an FPGA to capture and process images. The main drawbacks to the Cyclops system are low image resolution (128×128) due to limited RAM and slow processing of images (1 to 2 FPS). FireFly Mosaic uses CMUcam3 [2], an open-source programmable embedded vision processor. CMUcam3 is accompanied by a suite of open-source image processing libraries and sample applications that can be used as templates for custom image-processing algorithms.

Multiple groups have addressed the problem of determining camera topologies and doing in-home activity clustering. [13] uses stocastic sampling of plausable trajectories to determine which cameras are correlated with each other. We take a simpler approach by allowing a training period immediately after installation to generate our camera topology graphs (called camera network graphs). This system also uses a reliable high speed wired network as a communication medium, whereas we use a low speed wireless link. [14] shows activity clustering computed from a single wide angle camera attached to the ceiling. In our approach, both the model and the image processing is done in a distributed network containing multiple cameras. Both of these previous papers focus on algorithmic development and hence use desktop class computers that are rather expensive, power-hungry and often too large for WSNs.

In [12], the authors present WiSNAP, a Matlab-based platform for vision-enabled WSNs. The high-level Matlab code can be deployed using TinyOS on MeshEye sensor nodes. The authors mention that a major design limitation they suffer from is insufficient real-time support with respect to image processing. The combination of global time synchronization, a collision-free link layer, and synchronized task scheduling on each node, makes our system a stable framework for real-time distributed vision processing.

III. System Design

The FireFly Mosaic framework relies on various system-level principles which should be adopted as design guidelines for multiple classes of sensor networking applications. These key design principles include:

1) Physical Distribution of Sensors,
2) Tight Time Synchronization,
3) Integration of Multiple Sensing Perspectives,
4) High-Powered Local Processing to Reduce Data Transmission,

5) Increased Decision Confidence Resulting from Group Coordination, and
6) Efficient Communication in the Presence of Recurring Events.

The physical distribution of sensors along with the integration of multiple sensor perspectives allows coverage of large physical areas as well as the ability to correlate events. In visual sensor networks, this can provide a simple means to avoid problems associated with occlusion. Many complex tasks like evaluating the well-being of a person in their home require using a diverse set of sensors. Integrating devices like body-worn accelerometers with cameras and audio sensors provides a simple and reliable way of correctly classifying events without relying on single sensing sources. In order to maintain extensibility, a good sensor network needs to support the communication needs of all of these devices. Adding a light sensor to a sensor network designed around real-time communication is conceptually simpler than adding image streaming to a network without timing guarantees. The key to efficiently supporting these guarantees in a wireless multi-hop environment is time synchronization. Time synchronization provides both event ordering and the ability for nodes to efficiently coordinate communication as well as processing. As a simple example, we show later that object tracking in our network degenerates substantially as synchronization drifts.

Contention based sensor networks perform better if data are collected sporadically keeping the instantaneous load on the network relatively low. Many sensors like smart cameras require periodic and synchronous sampling of the environment. This in turn generates periodic bursts of high-load network traffic. They also require streaming large blocks of data (e.g. images). These communication patterns require efficiently scheduled messages. Over-provisioning bandwidth will go only so far since cameras and other high-bandwidth sensors will always be able to consume the additional resources. Sophisticated local processing of data also helps reduce communication requirements. Image sensors are an example of where constantly sending images is not nearly as efficient as local processing if possible.

Limited power and bandwidth are common problems found in sensor networks that are made worse with the addition of image processors. Cameras tend to consume large amounts of energy because they often operate at high frequencies and require the constant integrating and clearing of charge as light enters a large array of pixels. Even more problematic is that cameras have a much longer setup time because of autogains and whitebalance ($> 100ms$) making them harder to effectively run at low duty-cycles. Bandwidth can be problematic if raw images are transmitted. However, if images are processed locally, the meta-information sent over the network can be as small as a few bytes.

From an image-processing perspective, group coordination is a new challenge since now the cameras can communicate with one another over a network. In dynamic environments, if cameras are required to make group decisions, there needs to be a common notion of time so that events in frames can be correctly associated with one another. There also needs to be a communication mechanism in place to allow for efficient local message passing. Most previous image-processing algorithms are also not designed to utilize additional sensor information collected from the environment. In a sensor network, vision processing applications will likely work closely with other sensors like audio, temperature and body-worn accelerometers.

Detection of moving objects, such as people, in a scene is typically achieved using passive infrared (PIR) motion detectors. PIR devices can be thought of as single pixel cameras viewing infrared light. These types of sensors are good at detecting motion, however they are not able to localize motion within their field of view, distinguish between multiple sources of motion and they provide little information that can be used to intelligently filter readings. Cameras on the other hand can very easily locate and filter data based on various image features such as object size or location in the scene. Cameras also provide information like the color, shape and texture of objects which is lost in single pixel sensors. These properties can be used for smart filtering of moving machinery or small pets in a house. The cost and power consumption of cameras is rapidly being improved by the thriving cellular camera-phone market. With increased processing possible in cameras, they will soon become a more generic replacement for many current sensing devices like PIR detectors. Currently, PIR sensors consume less power than cameras by up to two orders of magnitude. In order to get the best of both worlds, PIR sensors can be used as a low-power wakeup mechanism for cameras.

A. FireFly Mosaic System Primitives

FireFly Mosaic provides the following system primitives to aid in system construction of distributed wireless sensor networks with vision-processing capabilities:

1) Efficient Camera Communication Layer: Communication of camera information involves two parts. The camera needs to communicate locally with the sensor node, and then the sensor nodes need to relay the data multiple hops over the network. The network communication relies on a TDMA-based link layer [4] with schedules and routing tables constructed based on how the cameras are deployed. The local communication between the camera and the sensor node currently uses a protocol based loosely on the Serial Line IP (SLIP) protocol. An application running on each FireFly node facilitates message passing between cameras acting as a transparent transport layer. This allows a camera to query the network for other cameras, send messages addressed to other cameras or the gateway without being aware of the underlying sensor network. This layer of abstraction simplifies the prototyping of distributed image processing algorithms by largely isolating the cameras from the underlying networking concerns.

2) Time Synchronization: Nano-RK running RT-Link provides sub-millisecond time synchronization. This is crucial for TDMA-based network packet transmission, and can also be used by tasks. With a camera operating at 30 frames per

second, even if the nodes are perfectly time synchronized the frame capture could experience a worse-case jitter of $33ms$. Once synchronization deviates beyond that point in a distributed system, there will likely be a decrease in system quality.

We use the hooks provided by Nano-RK for global task synchronizion to synchronize camera frame capturing. As the system executes, each FireFly node increments a global frame counter based on its globally synchronous wall clock time and passes this to the camera. The camera then blocks until the next globally synchronized frame by calling the `wait_until_next_frame()` function. This function returns the current global frame counter number shared among all nodes in the system. If the time synchronization is known to be not operational, the function returns an error code.

3) Camera Network Graph: A Camera Network Graph (CNG) captures the relationships between camera fields of view. FireFly Mosaic uses the CNG to optimize communication routes. Given a set of cameras, we create a CNG $G = \{V, E\}$, where V represents the camera nodes and E represents the set of edges between any two nodes if the cameras share information. Edges can be established in one of two ways, either based strictly on overlapping camera field of view, or based on the sequence at which nodes would be activated as an object moves through the scene. Figure 2 illustrates the two classes of CNGs generated for the camera topology shown in (a). Figure 2 (b) shows the resulting non-overlapping CNG where edges are added in relation to how cameras could be triggered by a moving object in the scene. An object moving from *cam-1* to *cam-3* would pass *cam-2* and hence be detected. In this case, the *cam-2* followed by *cam-3* transaction warrents an edge. Figure 2 (c) shows the CNG resulting when edges are added only when camera views overlap. In this case, there is no link from *cam-2* and *cam-3* because they do not share any common image regions. Unlike the non-overlapping camera graph, this graph may be disconnected. The non-overlapping CNGs are useful in security applications where the system must determine which cameras are nearby an event. For example, a surveillance application where the system tracks people moving from one camera zone to another. Overlapping CNGs are important for algorithms that correlate events viewed by multiple cameras from different perspectives like in our assisted living application. In many systems, one would use both graphs for different purposes.

4) Image Processing Tools: CMUcam3 comes with numerous open-source example applications and libraries including JPEG compression, frame differencing, color tracking, convolutions, histogramming, edge detection, connected component analysis, and a face detector. Since the entire system is open-source, these basic libraries can easily be extended to support custom applications.

5) Image Transfer: One of the most basic functions of a vision sensor is the ability to capture images. In battery-operated networks, transmitting these images takes a heavy toll on nodes generating or forwarding the data. A JPEG-compressed 176x144 (QCIF) image requires 7KB of memory

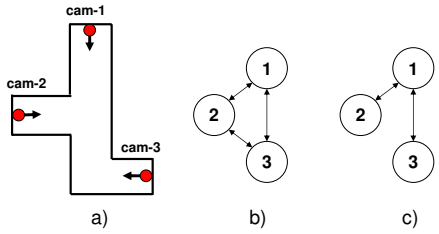


Fig. 2. This figure shows the two classes of Camera Network Graphs (CNG). (a) shows a scene with three cameras. (b) shows a non-overlapping CNG. (c) shows a shared view CNG.

Fig. 3. A CMUcam3 mounted on top of a four cell AA battery back with a FireFly wireless sensor node connected on the back.

which is more than 50 128 byte 802.15.4 packets. However, in many situations, the option to transmit an image may be in response to a crucial situation in which the network decides that it is worth the energy. For example, visual confirmation of a fire in room, or an image verifying that an elderly person is in distress would in most cases be worth the energy.

Using the camera communication layer, we provide a `transfer_jpeg_frame( ss_x,ss_y, x0, y0, x1, y1, quality)` function. This function allows the user to configure the amount of sub-sampling in the X and Y direction, the image window clip as well as the jpeg quality level. In section IV-B, we discuss balancing I/O overhead as well as CPU consumption to tune these parameters for transfer of the best perceived image quality in the least amount of time.

B. Hardware Components

The FireFly Mosaic hardware is composed of three main components; the CMUcam3 vision processing board, the FireFly sensor networking node and the FireFly gateway to PC interface board. Figure 3 shows the front and back of a battery-operated FireFly Mosaic node. The largest centrally located board is CMUcam3, the smaller board with the antenna is the FireFly node and the battery pack is located at the bottom.

The bottom section of Figure 4 shows the hardware architecture of CMUcam3. It consists of three main components: a CMOS camera chip, a frame buffer, and a microcontroller. The CMOS sensor is an OmniVision OV6620 camera capable of producing fifty 352x288 color images per second. These images are buffered using an Averlogic AL440b FIFO chip that effectively decouples pixel-level timing details from the micro-controller. Once a frame capture is enabled by the main processor, on-board logic manages loading the image

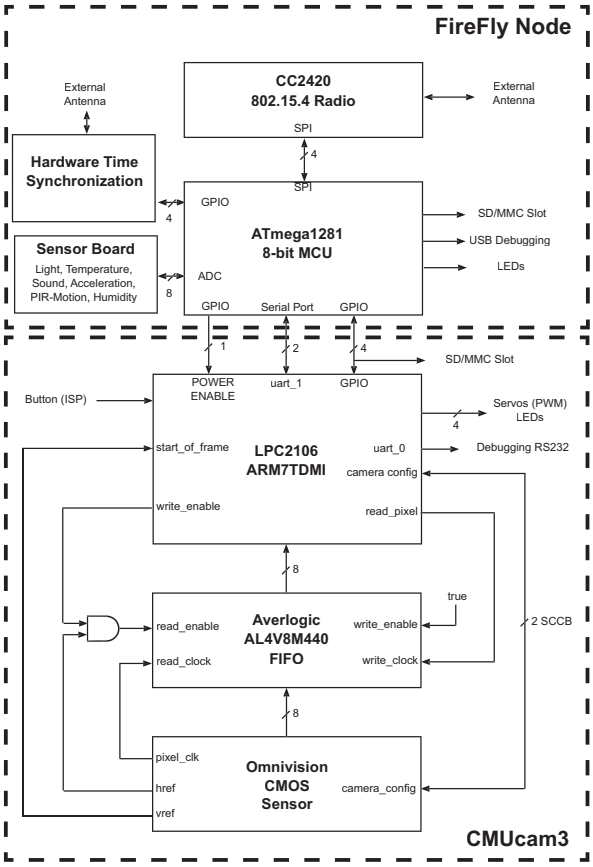


Fig. 4. FireFly Mosaic Vision-Enabled Node Hardware

into the FIFO. Processing of images is done using a low-cost 32-bit LPC2106 ARM7TDMI micro-controller running at 60MHz with 64KB of on-chip RAM and 128KB of on-chip FLASH memory. CMUcam3 has a serial boot-loader allowing executables to be flashed onto the board using the serial port without external downloading hardware. The interface between the FireFly node and CMUcam3 includes an in-circuit programming interface, making wireless updates of the camera software over the sensor network possible. CMUcam3 also has four built-in servo controller outputs which can be used to actuate a pan-tilt head.

The FireFly sensor nodes have a low-power Atmel Atmega1281 8-bit processor coupled with a Chipcon CC2420 802.15.4 radio. The main processor has 8KB of RAM and 128KB of FLASH memory. The radio is capable of transmitting at 250Kbps for up to 100 meters. The FireFly sensor nodes have a mini-SD slot which can be used for data storage as well as a hardware expansion. Any device designed to use this expansion slot can then also be tested on a PC given appropriate SDIO drivers. Unique to the FireFly sensor networking platform is the ability to add external time synchronization hardware. As described in [4], this support is provided by an out-of-band AM carrier current radio transmitter and on-board AM radio receiver. This time synchronization hardware provides as good as $20\mu S$ global time synchronization without the use of any in-band message passing. The FireFly boards can be interfaced with a sensor card that senses multiple variables including

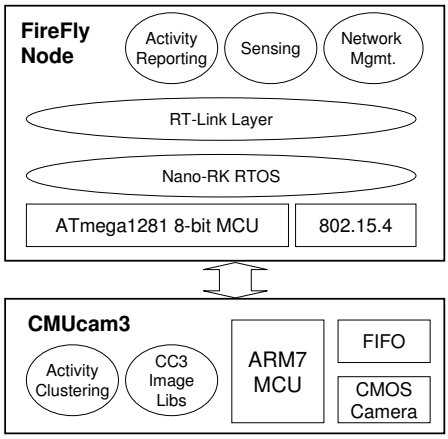


Fig. 5. FireFly Mosaic Vision-Enabled Sensor Node block diagram. Hardware components are shown in rectangles while software components are shown in ovals.

light, temperature, acceleration and audio.

The primary communication channel between CMUcam3 and the FireFly node is over TTL serial, with various extra GPIO pins that can be used for signalling. Using a 4 AA battery split voltage supply, the FireFly node can operate on an unregulated 3 volts, while CMUcam3 can use its onboard regulator to step the 6 volt output down to 5 volts. When AC wall power is available, CMUcam3 can use its internal regulator to power the FireFly board.

C. Wireless Sensor Network Setup

We now briefly describe the wireless sensor network and software infrastructure running on the FireFly Mosaic system.

As shown in Figure 5, each FireFly node uses the Nano-RK operating system [3], and runs the RT-link communication protocol [4]. Nano-RK is a real-time multi-tasking priority-driven reservation-based power-aware OS specifically designed for sensor nodes. RT-Link is a globally time-synchronized link layer protocol that communicates using scheduled TDMA slots. Mobile nodes communicate in a slotted aloha contention period or can be temporarily leased scheduled slots. Nano-RK with RT-link supports a `nrk_wait_until_tdma_slot()` syscall which suspends a task until a particular TDMA time slot. Since the link layer is globally time-synchronized, it offers a simple and powerful means for running distributed tasks and taking (near-)simultaneous image snapshots across multiple nodes.

D. Communication Scheduling

Given a network of cameras, our goal is to schedule communication such that each adjacent camera in the camera network graph (CNG) can communicate with all adjacent cameras before reporting data back to the gateway. This communication pattern allows all cameras with overlapping views to share them with one another before returning a final result. This can be achieved using a statically computed TDMA communication schedule. Reducing the number of slots in the schedule increases transmission concurrency and allows for a higher number of frames to be processed.

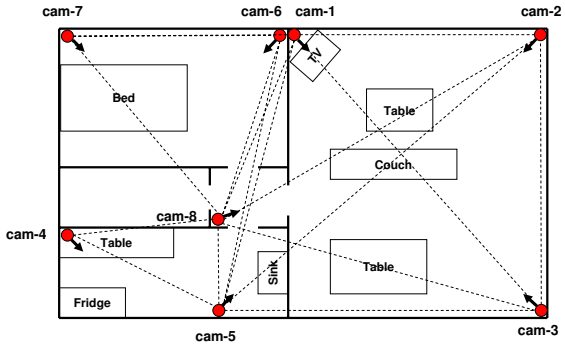


Fig. 6. Testbed Topology with eight cameras mounted in an apartment. The dotted lines denote communication links.

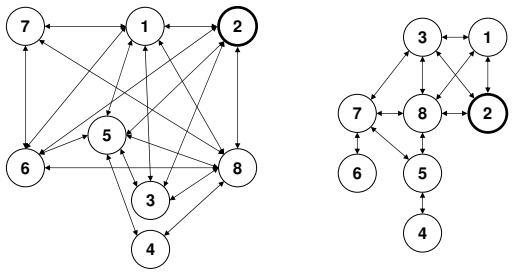


Fig. 7. Network Connectivity graph (a) and CNG (b) for the apartment deployment. Node 2 is bold indicating that it is the gateway.

Figure 6 shows the position and radio connectivity of eight cameras deployed in an apartment. Figure 8 (a) shows the wireless network topology where each node in the graph is connected to nodes that are within communication range. Figure 8 (b) shows the overlapping field of view CNG. Adjacent links connecting nodes indicate that the cameras share at least a portion of their field of view with each other. Previous methods for allocating TDMA slots given a network topology formed a spanning tree over the network graph, which is colored such that nodes within two-hops of each other have unique values. The spanning tree ensures total network connectivity while the two-hop coloring guarantees collision-free communication in the presence of hidden terminals. With vision nodes, we have the additional requirement of facilitating camera group communication. In order to accommodate camera flows, we specially mark links from the camera network graph found in the network topology graph. These will be used first when building a spanning tree to connect the network. Figure 2 (a) shows

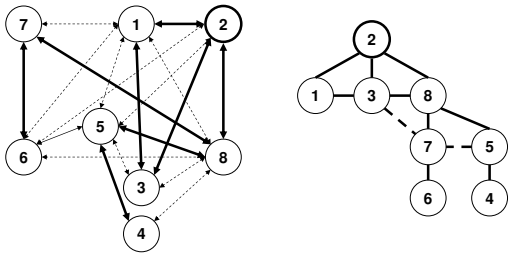


Fig. 8. This figure shows how the Camera Network Graph is used to construct the primary links in the scheduled communication graph.

Node	TX	RX
Node 1	1 | 17	0,2 | 16
Node 2	0	1,2 | 15,16,17
Node 3	2 | 16	0,6 | 14,15
Node 4	9	1,7
Node 5	7 | 11	4,6,9
Node 6	10	5
Node 7	5 | 13	3,4,8,10 | 12
Node 8	3,4,6,8 | 12,14,15	0,2,5,7 | 11,13

TABLE I

THIS TABLE SHOWS THE TRANSMIT AND RECEIVE SCHEDULE FOR EACH NODE IN THE APARTMENT DEPLOYMENT. THE SCHEDULE USES 18 OF THE 32 TOTAL TDMA SLOTS PER CYCLE.

the network connectivity graph with bold links that denote required camera communication. The next step requires adding additional links to ensure full network connectivity as well as including any CNG links that did not exist in the communication graph. Figure 2 (b) shows the network rooted at the gateway (Node 2) which is now ready to be scheduled. The dotted links between nodes 3,7 and 5 show instances where the cameras shared an overlapping view, but the radios cannot communicate directly. The schedule must accommodate message forwarding between these nodes.

Once a tree with all required connections has been established, we perform a two-phase scheduling heuristic. First, a breadth-first search schedules nodes starting from the root of the tree downwards. Any nodes that require message forwarding to share adjacent camera information are scheduled immediately after their adjacent partner in the camera graph is scheduled. For example, after Node 3 is scheduled to transmit, Node 8 is scheduled to receive and then forward the data to Node 7 on the next time slot. Once all nodes have been scheduled in this manner, the second phase of scheduling begins assigning slots in the reverse order to facilitate upstream communication. By the time the leaf nodes have been reached on the first pass of scheduling, all cameras should have heard from their neighbors. The second pass of scheduling data allows each node, if they require, to send data back to the gateway. Table I shows the complete schedule used in our apartment experiments. The vertical lines show the separation between the upstream and the downstream scheduling phases.

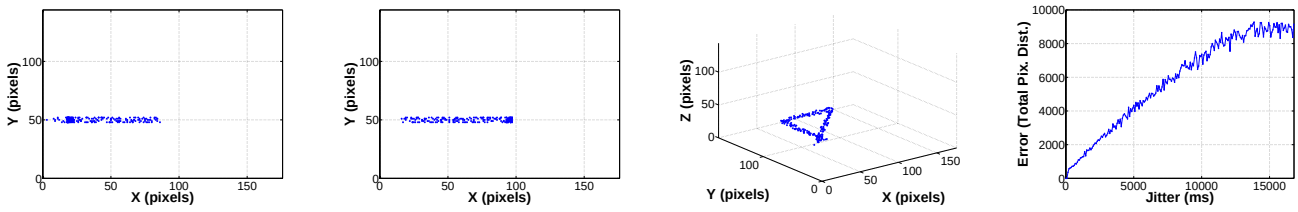
IV. System Performance

In this section, we evaluate the performance of FireFly Mosaic with respect to timing jitter, the trade-offs associated with CPU and network bandwidth when transferring images, and the energy of various system components.

A. Timing Jitter

Nano-RK running RT-Link maintains a network-wide time-synchronization of less than $100\mu S$. CMUcam3 has a continuously running asynchronous clock which generates image data. Since the camera captures frames at a maximum of 30Hz, this means that in the worst case, the timestamps between two images differ by as much as $33ms$.

To demonstrate the effects of jitter in our system, we conducted a simple multi-camera tracking experiment. We mounted four cameras on each wall of a 10 meter x 10 meter room that were setup to track the location of a bright



(a) (b) (c) (d)

Fig. 9. This figure illustrates how jitter in camera sampling could affect tracking accuracy. (a) and (b) show the result of tracking a bright object in a room from two cameras placed orthogonal to one another. (c) shows how these two images can be combined to determine the 3-dimensional location of the object. (d) shows how the tracking error increases with added timing jitter between when the cameras sample data and communicate.

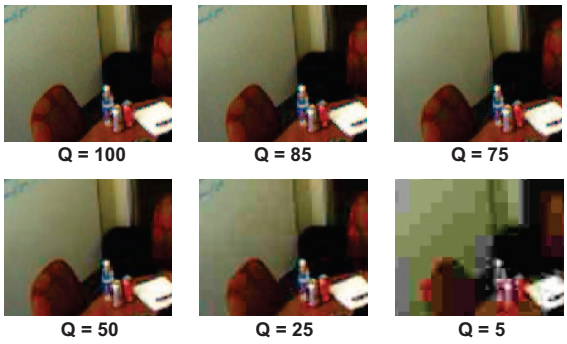


Fig. 10. Detail of a section of an image at different JPEG quality levels.

light. Each camera was placed 1 meter off the ground and facing directly towards the center of the room. The light was mounted on a wheeled cart that could be pushed along tape marks on the floor. The cameras used the FireFly Mosaic API call to load time-synchronized frames into their buffers followed by CMUcam3's `track-color` API call. Each camera returned the X and Y location of the light back to a gateway. Figure 9(a) and Figure 9(b) show the view from two orthogonal cameras. Figure 9(c) shows that using the X and Y axis of the two orthogonal cameras it is possible to plot the 3-dimensional location of the object being tracked. In this plot, the triangle path that the light followed can be seen. Such information cannot be determined from a single camera alone. After collecting the data, we compared the tracking performance of the system while adding random jitter to the time when points were sampled. Figure 9(d) shows how the total error increases as the timing jitter increases. The error eventually levels off as the timing is so skewed that the object could be in any location bounded by its path.

B. CPU and Network Bandwidth

The various processing stages involved in communication and processing must be well-understood in order to achieve the peak performance from the system. For example, in some cases, it might be advantageous to offload processing to a gateway machine if it takes longer or consumes more energy than transferring the raw data. To illustrate this point, we show the design space trade-offs associated with sending JPEG-compressed images over multiple hops to a gateway.

JPEG compression is a common technique for reducing the size of an image for transmission over a network. JPEG provides control over image size using a quality parameter given a particular resolution. Unfortunately, the popular version available does not have a mechanism to change the resolution and quality to match a particular network bit-rate. These features are now emerging in compression schemes like JPEG 2000. However, as shown in [16], these algorithms are typically too CPU-intensive for the class of devices used in wireless sensor networks. Figure 12 shows the size of a CIF resolution image with respect to the JPEG quality parameter as compressed by CMUcam3. Figure 10 shows a detail of the resulting images as the quality changes. We see that image quality is maintained below quality level 85 with a significantly smaller image size. Figure 13(a) shows the image transfer times given different quality levels and image resolutions required by an ideal MAC protocol using an 802.15.4 radio over multiple hops. This transfer time is calculated by using the size of the compressed image and the maximum 802.15.4 data rate divided by three to remove hidden terminals. Real protocols will have additional overheads as well as packet loss making these values the minimum transfer time.

Figure 13(b) shows the processing time taken by a node when capturing and compressing images at different quality levels and resolutions. We see that JPEG's processing time varies little with the quality level, but greatly with the resolution of the image. This is partially due to the increased CPU time, but largely an effect of the increased I/O transfer time between the camera and the main CPU. At lower resolutions, the camera skips reading pixels from the image buffer greatly decreasing the overall I/O bottleneck. Figure 13(c) shows the combination of the network transfer time with the CPU and I/O time. Since the I/O and CPU bottleneck can be on the same order as the network transfer time, it is important to understand how the perceived quality of an image changes as it is down-sampled to lower resolutions. Figure 11 shows two images of equal memory size. Image (a) shows a 352x288 (CIF) JPEG compressed at quality 25, while image (b) shows a 140x115 resolution image at quality 85 that has been scaled to equal the size of the first image. We see that images of the same size tend to be of nearly the same perceived quality. Due to the computation and I/O overhead of reading and compressing a larger image, the image on the left would take 2.5 times longer to transmit over the network. This notion of equal perceived image quality based on image size is captured by the dotted lines in Figure 13(c).

Based on Figure 13(c), if the objective is to achieve a particular frame rate, one should pick the lowest solid curve at the desired time interval. For example, to stream frames at 2 fps, one should pick the 0.3 resolution operating point on the quality 85 line. If instead, the objective is to transfer an image with the same perceived quality as a full resolution image at a particular JPEG quality level, then one should pick the farthest left point along the dotted line starting from

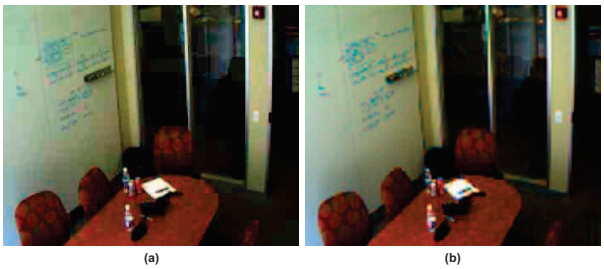


Fig. 11. Two equally sized images with similar perceived quality. The image on the left is a 352x288 JPEG compressed at quality 25. The image on the right is a 140x115 at quality 85. Due to the computation and I/O overhead of reading a larger image into memory, the image on the left would take 2.5 times longer to transmit over the network.

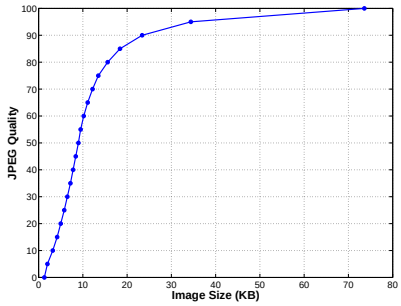


Fig. 12. This figure shows the image size with respect to JPEG quality.

your desired quality level. For instance, to transmit a full-sized quality 50 image, it is better to down-sample to .6 the original size and compress at quality 85. This reduces the computing bottleneck while maintaining image information, thereby decreasing the image transfer time from 1.9 seconds to 1.25 seconds. With different camera architectures, the particular values associated with these curves will change, but the notion of balancing CPU, I/O transfer times and networking throughput will always apply.

C. Energy

Table II shows the energy consumption of various components in the system during different power states. During our assisted living experiments, the nodes were operating at approximately a 20% duty cycle. At this load, FireFly Mosaic will run for just over five days from AA batteries. There are many simple optimizations that can be made to improve system lifetime. For example, at night or during periods of inactivity, the system could lower the rate of frame processing. One could use the CNG along with a predictive wakeup scheme that only activates nearby nodes when it sees motion about to enter the field of view. In most practical deployments, the effort should focus on optimizing the network power consumption since in long-term surveillence applications, cameras still require too much power and would need wall-outlet sources. This may not necessarily be the case for network forwarding nodes.

	Active(mW)	Idle(mW)	Sleep(mW)
CPU core	108	10	.25
CPU peripherals	49.5	2	.01
Frame Buffer	171	52	n/a
Camera	125	5	n/a
MMC	13.2	1	n/a
Atmega1281	6.6	.02	.02
CC2420	66.0	15	.01
Total	572.3	132.52	.29

TABLE II

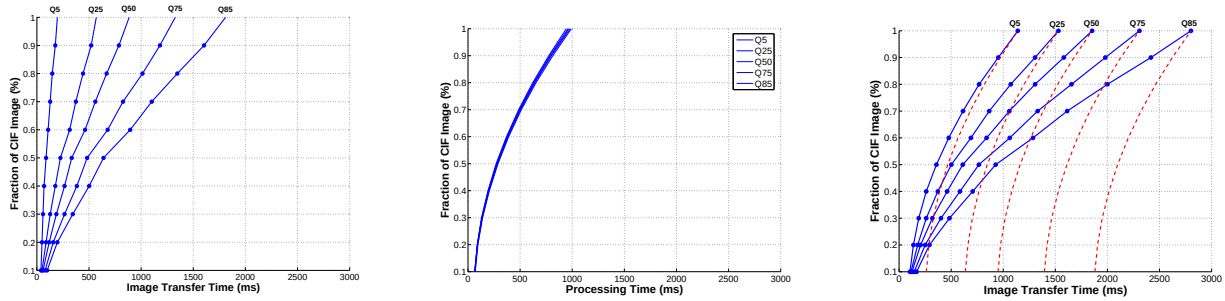
THIS TABLE SHOWS A BREAKDOWN OF THE POWER CONSUMPTION OF VARIOUS COMPONENTS WHILE THE SYSTEM IS IN DIFFERENT STATES OF OPERATION. WHEN THE CAMERA IS DISABLED BY THE FIREFLY NODE, THE ONLY POWER CONSUMED IS LEAKAGE CURRENT FROM THE MAIN VOLTAGE REGULATOR.

V. HOME ACTIVITY CLUSTERING

In this section, we briefly describe the home activity clustering application deployed using FireFly Mosaic. The system automatically combines information extracted from multiple potentially overlapping cameras to recognize various regions in the house where particular activities frequently occur. The final output of the system is a Markov model that shows regions of activity with transition probabilities based on the collected data. In a full system, these activity states could easily be fused with other sensors to more accurately determine the context of occupants in a living space. Once the context of the user is accurately determined, an elderly monitoring system could notify authorities or family members if they are in distress. The system could also log how their activities are changing over time in order to detect declines in activity or other more subtle problems. Beyond simply activity clustering, embedded vision-processing could begin to recognize more detailed events in the environment such as a person falling or a pot being left on a lit stove for several hours.

We now describe an experiment where we deployed eight cameras in an active apartment. During these tests, the vision nodes exchanged messages over multiple hops wirelessly. Figure 14 shows a timeline of different processing and communication transactions that occur between the sensor node and the camera. Each TDMA communication cycle completes every $192ms$. During each cycle, all cameras capture and processing a single frame which is then used by the sensor network in the next TDMA cycle. Pipelining the processing in this way yields an overall frame rate of 5.2 frames per second. Each camera initiates a frame capture at the start of the TDMA cycle. Figure 14 shows the timeline with various stages required to capture and process the image. Block (a) shows the jitter time between the start of the cycle and the start of the next frame. As previously mentioned, at 30FPS this can be as long as 33ms. Block (b) shows a fixed $20ms$ required for the image to load into the FIFO before it can be processed. Block (c) shows approximately $80ms$ of image processing time required by our application. We see that the camera requires in the worst case $133ms$ out of the $192ms$ available in a TDMA cycle. The longest latency between when a frame is captured and when data is returned over multiple hops to the gateway corresponds to the $102ms$ critical path in the TDMA schedule.

We now describe the various image processing stages of the algorithm used to build the final activity model. Many of the image processing details are beyond the scope of this paper and can be found in [15].



(a) (b) (c)

Fig. 13. This figure shows how the CPU, I/O and network affects the transmission of JPEG-compressed images. (a) shows the minimum possible time to transfer an image at different resolutions and qualities given an ideal MAC protocol running on 802.15.4 hardware. (b) shows the processing and I/O waiting time required to load and compress a JPEG image at different resolutions and qualities. (c) shows the end to end image transfer time including the compression and network transmission of an image. The dotted lines indicated where the perceivable image quality is identical.

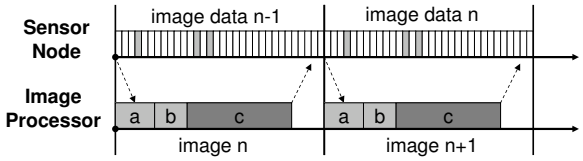


Fig. 14. This figure shows a timeline of the transactions that occur between the sensor node and the image processor. The time-slices on the sensor node timeline represent TDMA slots. The dark vertical lines show the seperation between TDMA cycles. Darker time-slots denote when that particular node communicates data.Block (a) on the image processor timeline shows the jitter associated with grabbing a frame. Block (b) shows the time take to capture a frame before processing can begin. Block (c) shows the image processing time followed by a communication of the result back to the sensor node.

The activity clustering algorithm consists of five main steps:

1) Generate CNG and Correlate Image Regions
2) Gaussian Mixture Model Activity Detection
3) Local Activity Clustering
4) Global Activity Cluster Merging
5) Generate Model

The first step in the process takes place during an initial setup phase. After the cameras have been placed in the house, a single person wearing a brightly colored outfit walks around the environment. Using time-synchronized frame differencing, simple correlation can be used between cameras to build the CNG as well as determine overlapping regions of the cameras. Each camera builds two four-by-four matrices capturing the image overlap between any moving regions during the training phase. Images from eight different cameras deployed as shown in Figure 6 can be seen in the first row of Figure 15. The resulting network, CNG and communication schedule are shown in Figure 8 and Table I.

After the training phase, the system switches into an observation mode where it begins to monitor the location of occupants over time. To separate the foreground from the background, we use a Gaussian Mixture Model (GMM) approach. Unlike a passive infrared motion detector, the GMM has the advantage of using memory to detect the presence of an object even if the object remains still. For example, if a person sits on a couch or lays in a bed, they may not move, but they should still be detected. Only after a person remains completely motionless for a very long period of time will they merge into the background. Practically speaking we very rarely saw this happening, even while occupants were sleeping. The second row of Figure 15 shows the output of the GMM averaged over the duration of the experiment. Lighter regions in the image represent more activity. At this point in the system's operation, multiple people can be in the environment. This causes problems with activity transition probabilities, but does not affect the performance of the clustering or the generation of activity histograms. Any higher-level processing should be able to accommodate multiple active states due to more than one person in the environment. As future work, it may be possible to track multiple occupants using both motion and feature matching such as color or texture of moving objects.

The next step in the algorithm is to cluster activity regions and merge regions viewed by multiple cameras. In order to cluster the activity groups, we first down-sample the GMM to a 16x12 pixel matrix and then remove any unconnected components that are smaller than two blocks. Row three of Figure 15 shows the clustered output. Any connected blob remaining in the image is considered an activity region for that single camera. Finally, we merge activity clusters on different cameras that are located in the same region in space. This can be done using the CNG and overlapping regions of the camera derived during the training period. The final row in Figure 15 shows a single instance of the overlapping regions matrix where each blob is uniquely identified. The numbers in each cell represent the global number of each activity cluster. Notice that the 17 blobs detected in all cameras are reduced to seven activities.

Figure 16 shows the seven main activity states detected during the apartment experiment. The transitions on the model represent probabilities that were greater than 60% when looking at the activity sequence. Each activity was labeled offline by a person using images captured by the system while a particular activity was occurring. As can be seen by activities like $a3$, an oscillating bag hanging on the door, some moving objects in the scene confused the system. These can be removed when the states are labeled. For the most part, the system detected only core occupant activities that with additional sensor information could be even more finely categorized.

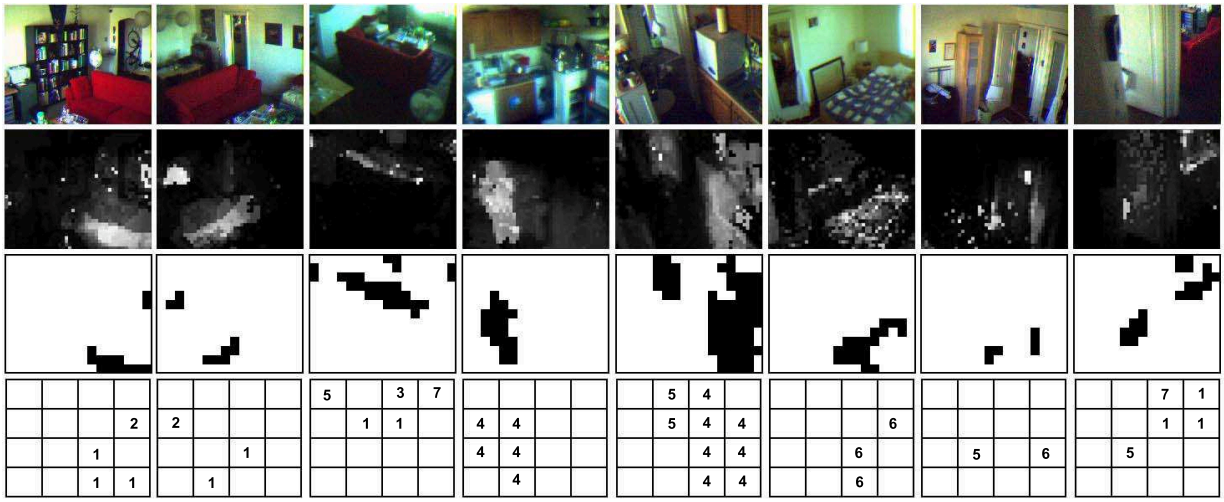
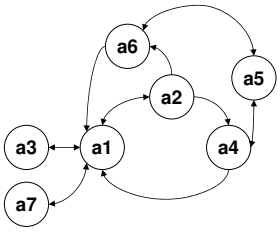


Fig. 15. This figure shows various steps of the activity clustering algorithm. The first row shows the view from the eight cameras mounted in the apartment. The second row shows the average result of the Gaussian Mixture Model. Lighter regions indicate more presence of foreground objects. The third row shows the clustered and thresholded activity regions. The final row shows the global set of activities that have been merged together based on the CNG and overlapping camera view matrices extracted during the camera setup phase.



State	Labeled Action
a1	Sitting on Couch
a2	Working at Table
a3	Television
a4	Working at Sink
a5	Bag Moving on Door
a6	Sleeping in Bed
a7	Checking Gateway CPU

a) b)

Fig. 16. Activity State Model extracted from observing 24 hours in an apartment.

VI. Conclusions

With new CMOS camera sensors emerging, many useful computer vision techniques can now execute on smaller, low-cost embedded packages. In the WSN domain, this enables a rich new sensing modality that can benefit a wide range of applications from security and process monitoring to smart spaces and assisted living. This paper presents FireFly Mosaic, a framework for integrating vision processing into WSNs. To the best of our knowledge, this is the first fully integrated wireless sensor network with mulitiple coordinating camera nodes performing local processing.

FireFly Mosaic builds upon the FireFly real-time sensor networking platform by integrating a sensor node with a programmable embedded image processer. We address various design principles that are applied to an assisted living application where occupant activities are automatically discovered and clustered using multiple cameras with overlapping views. This system was successfully deployed in an active apartment where eight vision-enabled nodes communicated in a mesh network identifying various activities such as washing the dishes or sitting on the couch. In the future, we also plan to extend this work to support detection of particular types of activities. This could be used in the home to detect events such as an elderly person falling or in surveillance systems to isolate suspicious events.

References

[1] R. Mangharam, A. Rowe, R. Rajkumar, "FireFly: A Cross-Layer Platform for Wireless Sensor Networks", *Real Time Systems Journal, Special Issue on Real-Time Wireless Sensor Networks* , Nov. 2006.

[2] A. Rowe, A. Goode, D. Goel, I. Nourbakhsh, "CMUcam3: An Open Programmable Embedded Vision Sensor", *International Conferences on Intelligent Robots and Systems*, Oct. 2007.

[3] A. Eswaran, A. Rowe, R, Rajkumar, "Nano-RK: an Energy-aware Resource-centric RTOS for Sensor Networks", *IEEE Real-Time Systems Symposium*, Dec. 2005.

[4] A. Rowe, R. Mangharam, R. Rajkumar, "RT-Link: A Time-Synchronized Link Protocol for Energy-Constrained Multi-hop Wireless Networks", *Third IEEE International Conference on Sensors, Mesh and Ad Hoc Communications and Networks (IEEE SECON)*, Sep. 2006.

[5] "US Census Bureau", `http://www.census.gov`, Viewed on March 25, 2007.

[6] S. Hengstler and H. Aghajan, "A Smart Camera Mote Architecture for Distributed Intelligent Surveillance", *ACM SenSys Workshop on Distributed Smart Cameras*, Oct. 2006.

[7] M. Rahimi, R. Baer, O. Iroezi, J. Garcia, J. Warrior, D. Estrin, M. Srivastava, "Cyclops: In Situ Image Sensing and Interpretation in Wireless Sensor Networks", *ACM SenSys*, Nov. 2005.

[8] T. He and J. A. Stankovic and C. Lu, T. F. Abdelzaher, "SPEED: A Stateless Protocol for Real-Time Communication in Sensor Networks", *ICDCS*, 2003.

[9] C. Lu and B. M. Blum and T. F. Abdelzaher and J. A. Stankovic and T. He, "RAP: A Real-Time Communication Architecture for Large-Scale Wireless Sensor Networks", *RTAS*,2002.

[10] T. F. Abdelzaher and S. Prabh and R. Kiran, "On Real-Time Capacity Limits of Multihop Wireless Sensor Networks", *RTSS*,2004.

[11] T. Facchinetti and L. Almeida and G. C. Buttazzo and C. Marchini, "Real-Time Resource Reservation Protocol for Wireless Mobile Ad Hoc Networks", *RTSS*, 2004

[12] S. Hengstler, H. Aghajan, "Application Development in Vision-Enabled Wireless Sensor Networks", *International Conference on Systems and Networks Communications*, 2006.

[13] D. Marinakis, G. Dudek, "Topology Inference for a Vision-Based Sensor Network", *Canadian Conference on Computer Vision and Robotics*, 2005.

[14] T. Teixeria, D. Lymberopoulos, E. Culurciello, Y. Aloimonos, A. Savvides, "A Lightweight Camera Sensor Network Operating on Symbolic Information", *ACM SenSys Workshop on Distributed Smart Cameras*, 2006.

[15] D. Goel, A. Rowe, R. Rajkumar "Sensor Network Activity Cluster using Embedded Vision Processors", *Carnegie Mellon ECE Technical Report*, 2007.

[16] G. Pekhteryev, Z. Sahinoglu, P. Orlik, G. Bhatti "Image Transmission over IEEE 802.15.4 and ZigBee Networks", *Mitsubishi Electric Research Labs Technical Report*, Nov. 2004.