9-30-2009

Power Consumption and Performance Analysis of Object Tracking and Event Detection with Wireless Embedded Smart Cameras

Mauricio Casares
University of Nebraska - Lincoln

Alvaro Pinto
University of Nebraska - Lincoln

Youlu Wang
University of Nebraska - Lincoln

Senem Velipasalar
University of Nebraska - Lincoln, svelipasalar2@unl.edu

3rd International Conference on Signal Processing and Communication Systems
Omaha, Nebraska, 28-30 September 2009

Power Consumption and Performance Analysis of Object Tracking and Event Detection with Wireless Embedded Smart Cameras

Mauricio Casares, Alvaro Pinto, Youlu Wang, Senem Velipasalar
Electrical Engineering Department
University of Nebraska-Lincoln
mauricio.casares@huskers.unl.edu, alvaro.pinto-albuja@huskers.unl.edu
youlu.wang@huskers.unl.edu, velipasa@engr.unl.edu

***Abstract*—An embedded smart camera is a stand-alone unit that not only captures images, but also includes a processor, memory and communication interface. With battery-powered and embedded smart cameras, it has become viable to install many spatially-distributed cameras interconnected by wireless links. Not requiring to have access to electrical outlets and have wired links increase system flexibility. However, wireless and battery-powered smart-camera networks introduce many additional challenges since they have very limited resources, such as power, memory and bandwidth. The algorithms running on the camera boards should be lightweight and efficient. In addition, the frequency of communication between camera nodes, and the content of the message packets should be carefully designed, since communication consumes power. In this paper, we present a wireless embedded smart-camera system that performs peer-to-peer object tracking and event detection. We analyze the power consumption and performance of this system during different parts of the algorithm execution and for different message exchanges between camera nodes. We also present a graph of the energy consumption for different tasks performed in a camera's processor. The number of instructions are also presented. The results demonstrate the importance of the careful choice of when and what data to transfer between cameras, and also the necessity of having lightweight algorithms in these resource-constrained systems.**

I. Introduction

Cameras are employed in military, public and commercial applications for surveillance and statistics gathering. Many systems have been introduced to automatically analyze the large amount of video data generated by multiple cameras. However, most work in this area has considered systems that rely on a back-end server or control center to process multiple video inputs. Yuan et al. [31], Collins et al. [6][7], Nguyen et al. [18], Lo et al. [17] and Krumm et al. [15] present systems where a server/controller performs the coordination and integration of the data from individual nodes. But, these systems have a bandwidth scaling problem, since the central server can quickly become overloaded with the aggregate sum of messages/requests from the nodes. Also, the server is a single point of failure for the whole system.

The aforementioned problems of server-based systems necessitate the use of peer-to-peer (P2P) systems, where individual nodes communicate with each other without going through a centralized server. Atsushi et al. [1] use multiple cameras attached to different PCs connected to a network. They calibrate the cameras, and send message packets between stations. Ellis [9] also uses a network of calibrated cameras. Velipasalar et al. [28] present a P2P multi-camera system for multi-object tracking, wherein each camera is attached to a different CPU. But, cameras and CPUs in these systems are assumed to be wall-powered and/or communication is performed over wired links. These requirements affect the system flexibility and incur additional costs.

Battery-powered embedded smart cameras make it possible to install many spatially-distributed cameras interconnected by wireless links. An embedded smart camera is a stand-alone unit that not only captures images, but also includes a processor, memory and communication interface. However, wireless and battery-powered smart-camera networks introduce many additional challenges including limited processing power, memory, energy and bandwidth. Limited resources necessitate light-weight algorithms to be implemented and run on the embedded cameras, and also careful choice of when and what data to transfer.

Many embedded vision sensor platforms have been developed more recently [23][11][24][8][10][16][14]. The Cyclops platform [23] has a 7.3 MHz processor. The MeshEye [11] integrates three CMOS cameras to provide a surveillance camera module, which has a processor with 50 MHz speed. However, in both of these platforms the processing power is very limited. Panoptes platform [10] hosts a 206 MHz processor, WLAN card and a webcam, but has high energy consumption. SensEye [16] is a multi-tier network of heterogeneous wireless nodes and cameras. Bramberger et al. [2] introduce a highend platform reaching a processing power of 9600 MIPS with onboard memory of 784 MB. However, it requires an average power consumption of 35 Watts.

We have presented a wireless embedded smart camera system for cooperative object tracking [29] and detection of composite and semantically high-level events spanning multiple camera views [30]. Each camera in this system is a CITRIC mote [5] that consists of a camera board and a

This work has been funded by NSF grant CNS0834753.

978-1-4244-4474-8/09/$25.00 ©2009 IEEE

wireless mote. The microprocessor on the camera board is a fixed-point processor with a maximum speed of 624MHz and 256KB of internal SRAM. The wireless mote connected to and powered by the camera board is a TelosB mote. Since local memory for storage, and power is limited in each camera, we designed and implemented light-weight and robust background subtraction and tracking algorithms that are run on the camera boards. Cameras exchange data in a P2P manner over wireless links to track objects consistently, and also to update locations of occluded or lost objects. Due to relatively large power consumption of wireless communication, it is carefully designed when and what data to communicate between camera nodes. With this system, we do not transfer or save every frame or every object trajectory. We detect *events of interest* so that interesting and important video portions and trajectories can be determined.

In this paper, we analyze the power consumption and performance of this system during different parts of the algorithm execution and for different message exchanges between camera nodes. We also present a graph of the energy consumption for different tasks performed in a camera's processor. The number of instructions are also presented. The results demonstrate the importance of the careful choice of when and what data to transfer between cameras, and also the necessity of having lightweight algorithms in these resource-constrained systems.

II. The Wireless Embedded Smart Camera Platform

In this section, we provide a summary of the wireless embedded smart camera platform employed in our system. The details of the platform can be found in [5][30].

The wireless embedded smart camera platform [5] we use consists of a camera board and a wireless mote, and is shown in Fig. 1. The camera board is composed of a CMOS image sensor, a fixed-point microprocessor, external memories and other supporting circuits. The camera is capable of operating at 15 frames per second (fps) in VGA and lower resolutions. The wireless mote connected to the camera board is a TelosB mote from Crossbow Technology. The maximum data rate of the TelosB is 250kbps.

Each camera performs foreground detection, blob forming, object tracking and composite event detection, and all the processing is performed on the camera boards. The camera board is connected to the wireless mote with a serial port. The wireless mote is an idle mode during most of the time while the camera is performing tracking. In other words, no serial communication is performed between the camera board and the wireless mote. When the camera needs necessary information from other cameras, and needs to exchange data, only then it performs serial communication with the wireless mote to send and receive packets. Due to the low radio data rate and the small buffer size of the mote, we also need to buffer and transfer as few and as small-sized packets as possible. The algorithms and the communication protocol have been designed and implemented by taking this fact into account.

III. Light-weight Object Detection and Tracking Algorithms

Limited processing power and limited memory in embedded smart cameras necessitate the design of light-weight vision algorithms for object detection and tracking. Lighting variations and non-static backgrounds make the foreground object detection problem even more challenging. The necessity of eliminating uninteresting motion, such as swaying trees, increases the algorithm complexity, and thus memory requirements.

We presented a light-weight and efficient foreground detection algorithm in [3], [4], and implemented and imported it to the embedded smart camera nodes [30]. This algorithm is highly robust against lighting variations and non-static backgrounds including scenes with swaying trees, water fountains and rain. The memory requirement for the data saved for each pixel is very small compared to many traditional background subtraction methods. This algorithm requires 6.25-byte memory per pixel, whereas original mixture of Gaussians [25], Eigenbackground [19] and Codebook [13] methods require 32, 28 and 91 bytes per pixel, respectively. We also implemented the adaptive Mixture of Gaussians (MoG) [25], which is one of the most commonly used background subtraction methods, on our smart camera board to compare their performances. On

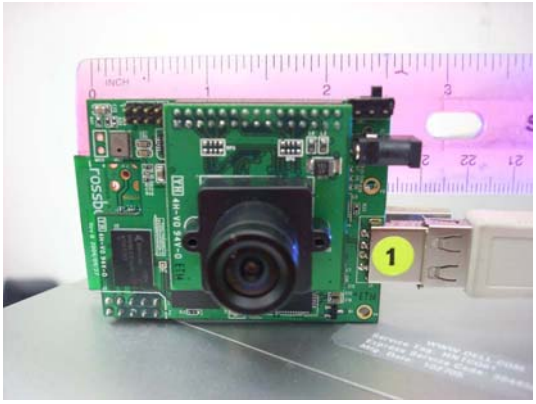


Fig. 1. The wireless embedded smart camera platform employed in the presented system.

our camera boards, the MoG algorithm runs at 1.6 frames per second (fps), and the light-weight algorithm runs at 12.5 fps when there is one foreground object in the scene. The details of our foreground detection algorithm can be found in [4].

We also implemented a reliable and efficient tracking algorithm on the embedded smart camera nodes. The details of this algorithm are described in [28] and [29].

In order to solve the consistent labeling problem, we employ the field of view (FOV) lines, which were introduced by Khan and Shah [12]. A camera also uses the FOV lines to determine which other cameras can see an object that is in its field of view, and thus to figure out to which cameras the request messages should be addressed. We recover the FOV lines off-line as described in [28]. When a new foreground object enters in a camera's field of view, the camera first checks if this object can be seen by any other cameras by employing the FOV lines. The midpoint of the bottom line of the bounding box of the object is used as its location. If this point lies on the visible side of all the FOV lines belonging to camera i, then it is deduced that this object is visible by camera i. In this case, a request for a label addressed to camera i will be sent out by wireless communication to achieve consistent labeling.

IV. Wireless Communication Between Cameras

Due to relatively large power consumption of wireless communication, it is very important to carefully design when to communicate and what to communicate, and to employ algorithms that do not require transfer of large data between cameras. In the presented system, cameras exchange small-size packets in a P2P manner over wireless links.

A camera will need to communicate with other cameras when (a) a new object appears in its FOV, (b) a tracker cannot be matched to its target object, and (c) a primitive event that is part of a pre-defined composite event scenario occurs in its FOV. These three cases are referred to as *New_label*, *Lost_label* and *Primitive_occurred* cases, respectively.

The system can perform detection of semantically high-level events. We define composite and semantically high-level events as a sequence of primitive events, and these primitives can be defined on different camera views [30]. The existing primitive events are motion detection and trip-wire crossing. An example event scenario of interest could be detecting a car entering the scene in the first camera view, and then going through a region of interest in the second camera view, and then parking in a region defined on the second camera view. When a camera detects a primitive event, it sends a message addressed to the next camera in the sequence to let it know the occurrence of this primitive event, and the label of the object performing the primitive event. Since the primitives and their sequence are all pre-defined, we already know which primitive is defined on which camera view. When a primitive event occurs in one camera, it will address the message to the next camera in the sequence.

In the *New_label* and *Lost_label* cases, before sending the request, the current camera checks the visibility of the target by other cameras by employing the FOV lines. If it is deduced that this object is visible by another camera, the ID of that camera will be included in the request message. This way, when a camera receives a broadcasted message, it will drop the message if the target ID in the message does not match its own ID.

A. What data to communicate

Small-sized packets are exchanged between cameras to reduce power consumption and delay. The contents of messages for different scenarios are described in following subsections. In all the messages below, *Target_ID* is the wireless mote ID of the camera to which a message is addressed, and *My_ID* is the ID of the camera sending the message.

1) New_label request: When a new object appears in the current camera view, a tracker is created for it. If it is determined that another camera can see this object, a temporary label is assigned to the object and a request message, addressed to that camera, is created, which has the following format:

$$Target\_ID \quad My\_ID \quad Pkt\_type \quad x \quad y \quad Tmp\_label$$

where *Pkt_type* is an unsigned character indicating that this packet is for a new_label request, and *x* and *y* are the coordinates of the object in the current camera view. *Tmp_label* is the temporary label assigned to this newly found object. When a reply is received, this temporary label is replaced by the received label. In this case, we need 1 byte each for the *Target_ID* and *My_ID*, 1 byte for the *Pkt_type*, 2 bytes for *x*, 1 byte for *y* (the width of the frame is greater than 256 and the height of the frame is less than 256), and 1 byte for the *Tmp_label*. Thus, we only use 7 bytes for a new_label request.

2) New_label reply: When a camera receives a packet that is addressed to itself, and finds that it is for a new_label request, it will calculate the object's corresponding location in its own view by using the received coordinates and the homography matrix calculated off-line. Then, it will find the distance of the closest tracker to the calculated location. If this distance is smaller than a threshold, it will send the label of this tracker as reply in the following packet form:

$$Target\_ID \quad My\_ID \quad Pkt\_type \quad Tmp\_label \quad Ans\_label$$

where *Pkt_type* is an unsigned character indicating that this packet is for a new_label reply. *Tmp_label* is the temporary label the requesting camera is using, and *Ans_label* is the reply label. In this case, we only use 5 bytes in the packet.

3) Lost_label request: For a tracker that cannot be matched to its object, a camera that can see the most recent location of this tracker is found by using the FOV lines. Then, a lost_label request packet is formed, which has the following format:

$$Target\_ID \quad My\_ID \quad Pkt\_type \quad Lost\_label$$

where *Pkt_type* is an unsigned character indicating that this packet is for a lost_label request, and *Lost_label* is the label of the tracker which could not be matched to an object. We only use 4 bytes for a lost_label request.

4) Lost_label reply: When a camera receives a packet that is addressed to itself, and finds that it is a lost_label request, it sends the current location of the tracker, whose label is the same as the *Lost_label* entry of the request message, as reply. The reply packet has the following format:

$$Target\_ID \quad My\_ID \quad Pkt\_type \quad Lost\_label \quad x \quad y$$

where *Pkt_type* is an unsigned character indicating that this packet is for a lost_label reply. *Lost_label* is the label of the tracker received from the requester, and x and y are the coordinates of the object. In this case, we use 7 bytes in the lost_label reply packet.

When the requesting camera receives the reply, it calculates the corresponding location of the object in its own view, and updates the tracker's location.

5) Primitive_occurred message: In the *Primitive_occurred* case, the camera nodes send out messages to inform the next camera node in the defined event sequence. Thus, they do not need replies. The event scenarios of interest are defined beforehand, and they are composed of primitive events. When the first primitive event occurs in the view on which it was defined, this camera sends a message addressed to the camera that is responsible to detect the next primitive event in the sequence. Once the second primitive occurs, and if the third primitive is defined on a different camera view, the second camera sends a message addressed to that camera. The form of the packet for the *Primitive_occurred* message is:

$$Target\_ID \quad My\_ID \quad Pkt\_type \quad Prim\_ID \quad Obj\_label$$

where *Pkt_type* is an unsigned character indicating that this packet is a message informing the occurrence of a primitive event. *Prim_ID* is the order number of the primitive event in the sequence, and *Obj_label* is the label of the object performing this event. In this case, we only use 5 bytes for the *Primitive_occurred* message.

V. Power Consumption Analysis of the System

We analyzed the power consumption of the system over time, where the time and power consumption are measured in milliseconds and watts, respectively.

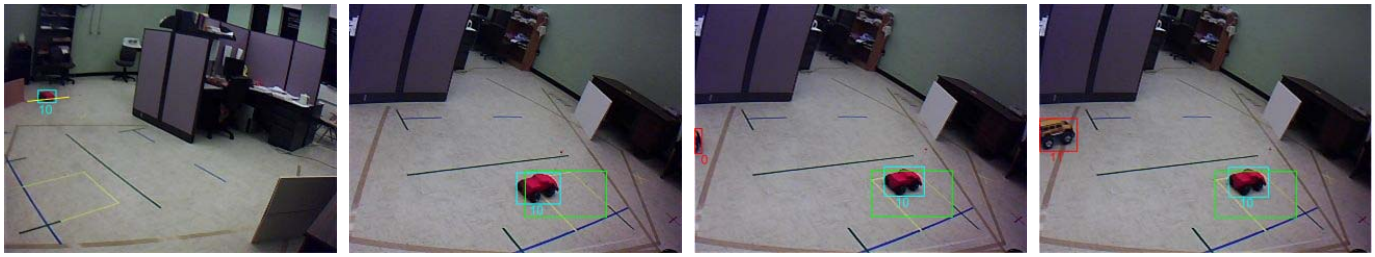
An experiment was performed where remote controlled cars were tracked cooperatively by two embedded smart cameras. A composite event composed of two primitive events was defined on two different camera views. The event of interest is detecting a car crossing a tripwire in the first camera view, *and then* parking in a region defined on the second camera view. Figure 2(a) shows an example of detecting the first primitive event on the first camera view, where the yellow line is the pre-defined trip-wire. When the car crosses this line, the first camera sends a *Primitive_occurred* message to the second camera, and this message includes the label of the car (10) performing the event. The second primitive event occurs when an object is detected in the specified region for longer than a specified period. Figures 2(b) and (c) show the car entering the region of interest and remaining there. The second primitive event occurs, and thus the whole composite event is detected.

When an object enters into the view of a camera, the object is assigned a temporary label. As seen in Fig. 2(c), a new car enters into the FOV of the second camera, and it is assigned the temporary label 0. Then FOV lines are employed to determine if any of the other cameras in the system can see this object. If the object can be seen by another camera, this camera will send a *New_label* request addressed to that camera, and will continue tracking this object with the temporary label until receiving a reply. In the example seen in Fig. 2(c), the second camera sends a *New_label* request to the first one. Once the correct label (11) is received, the requesting camera replaces the temporary label with the correct one, as seen in Fig. 2(d).

During the experiment, there were multiple instances of a new object entering into a camera's FOV, and then a *New_label* request being generated. Figures 3 and 4 show two other cases of successful consistent labeling.

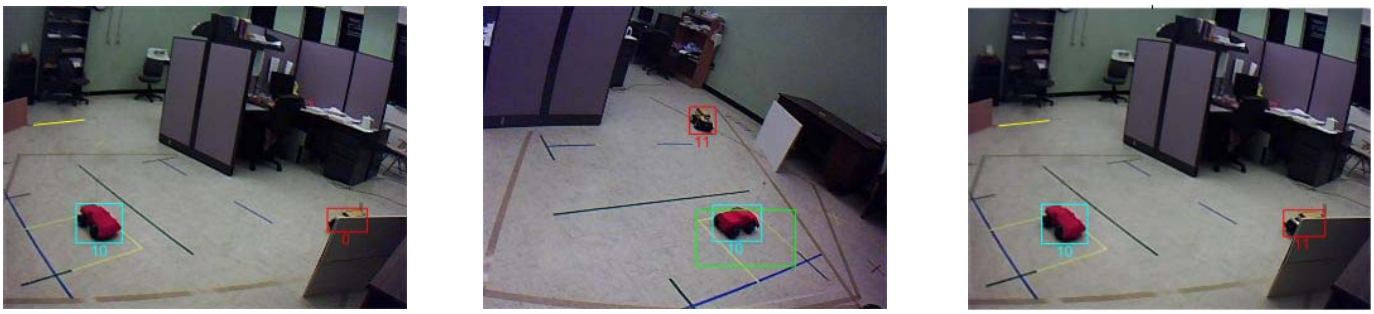
The power consumption of one of the cameras was analyzed over time for this scenario. Fig. 5 shows the plot of the consumed power (in Watts) versus time (milliseconds). In this plot, yellow corresponds to receiving a *Primitive_occurred* message, green corresponds to sending a *New_label* request, while red corresponds to receiving a *New_label* reply. The blue color represents the average power consumed by the camera board when tracking different number of objects. The power consumed by the camera board increases with increasing number of tracked objects or vice versa.

As stated before, two primitive events were defined in this scenario. When the first camera detects the first primitive event, it sends a *Primitive_occurred* message to the second camera. The yellow peak in Fig. 5 indicates the reception of this message in the second camera. During this time, there



(a) Camera 1 (b) Camera 2 (c) Camera 2 (d) Camera 2

Fig. 2. Composite event detection and consistent labeling: (a) The first primitive event is detected on the first camera view, (b)-(c) second primitive event, and thus the composite event scenario is detected, (d) the correct label of the new object is received from the first camera.



(a) Camera 1 (b) Camera 2 (c) Camera 1

Fig. 3. Receiving the correct label of a new object entering the FOV: (a) The yellow car enters the FOV of the first camera, it is assigned a temporary label, and a New_label request is sent to the second camera; (b) the object is being tracked by the second camera, which sends the correct label 11 to the first camera; (c) the temporary label is replaced by the correct label 11 in the first camera.

are no objects in the view of the second camera yet, and the camera board consumes an average of 1.515 watts by grabbing a frame and doing background subtraction and model update. Then a car enters into the FOV of the second camera, and the second camera sends a *New_label* request to the first camera at time $t = 25486$ ms. The reply of the first camera arrives in approximately 90ms to the second camera. As can be seen in the figure, after the car enters into the FOV of the second camera, the power consumption of the camera board increases from 1.515 to 1.627 watts, since the camera has started to track this object. Subsequently, at time $t = 97286$ ms, another car enters into the FOV of the second camera. Once more, a *New_label* request is sent to the first camera, and the corresponding reply is received. Since there are now two objects in the FOV of this camera, the consumed power increases to 1.657 watts. At time $t = 199317$ ms, a third car enters into the view, and the required power increases to 1.787 watts.

The green peaks in Fig. 5 correspond to the power that is required to transmit a *New_label* request message, whereas the red peaks correspond to the power that is consumed to parse a received message. Power formulas have been used to calculate the power consumption of the wireless communication. As stated above, in the presented system, small-sized message packets are sent between camera nodes for different scenarios, namely *New_label* request, *New_label* reply, *Lost_label* request, *Lost_label* reply, and *Primitive_occurred* cases. Table I summarizes the number of bytes in each of these messages. Additionally, a default header of 14 bytes, due to the MAC protocol, is added to encapsulate the information.

TABLE I
NUMBER OF BYTES IN DIFFERENT MESSAGE PACKETS

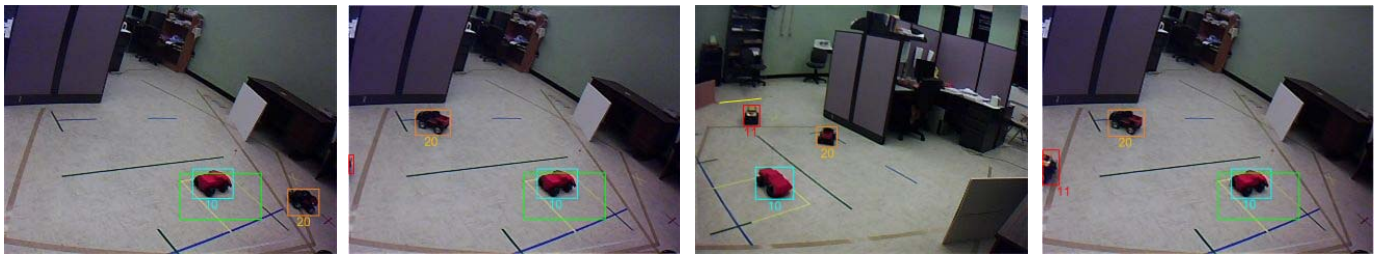
	New_label	$Lost\_label$	$Primitive\_event$
Request	7 bytes	4 bytes	5 bytes
Reply	5 bytes	7 bytes	

Figure 6 shows the energy consumed for transmitting request and receiving reply messages. The energy is calculated based on the packet size. $Power = V \times I$ where V is the nominal voltage, which is 3V for the TelosB. The current I is taken from the data sheet [22]. The typical current for reception (21.8 mA) is larger than the current for transmission (19.5 mA). The time that takes to send each packet is computed using the following formula:

$$\text{time} = \frac{N_T \times 8}{D} \tag{1}$$

where N_T is the number of bytes to be send, D is the nominal data rate, which is 250 Kbps for the CC2420 radio chip. Thus, based on this formula, it takes 0.672ms to transmit a 21-byte (14+7 bytes) *New_label* request message. The energy required is $E = P \times T = 19.5mA \times 3V \times 0.672ms = 39.312\mu$J.

On the other hand, the power consumption of the camera board depends highly on the amount of activity in the scene.



(a) Camera 2 (b) Camera 2 (c) Camera 1 (d) Camera 1

Fig. 4. (a) The second camera labels a new object entering in its FOV, (b) the second camera issues a New_label request after assigning a temporary label to the object entering from the left, (c) the first camera replies with the correct label 11, (d) the second camera updates the label of the target object.

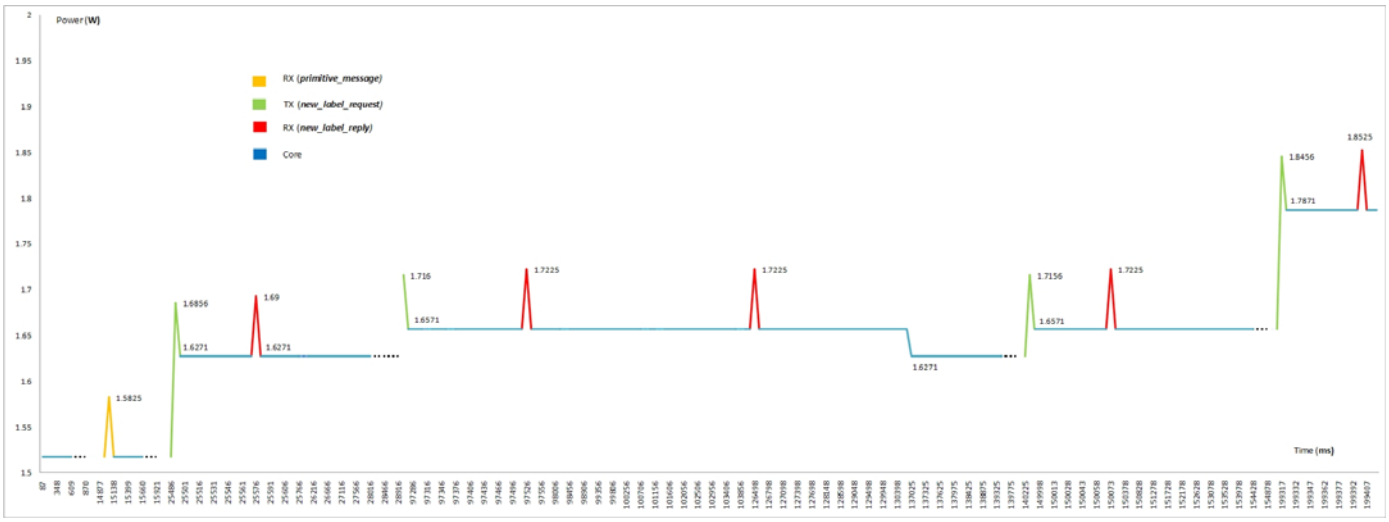


Fig. 5. Power consumption over time.

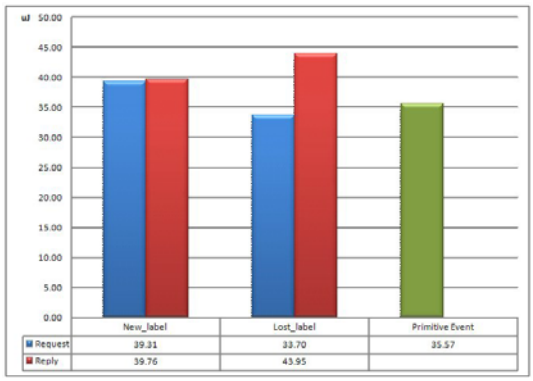


Fig. 6. Energy required to transmit and receive different message types.

Different number of objects in the scene causes variations in the operating current of the camera board. The current increases or decreases based on the workload of the processor (number of instructions per task), the supply voltage source and the frequency at which the processor is working. Thus, we measured the operating current when there were different number of objects in the scene. To measure the current, we used a precise oscilloscope and a 1-ohm resistor configuration placed at the input of the supply source. Figure 7 shows the variations in the current during the *processing* of frames containing different number of tracked objects. The measures in the oscilloscope have to be scaled by 10.

The system takes an average of $87ms$ to process one frame. Figure 8 shows a bar graph of the energy consumption for different tasks performed in the camera's processor. The number of instructions are also presented. All the computations were done when there is one tracker. This plot can help us to extrapolate the energy consumption for more than one tracker if needed.

In the above discussion, we did not consider the power consumption of the UART protocol between the camera mote and the TelosB, since this process only requires a current in the order of 15 to 50 μA and the time to send the packets is in the order of 20μs. Thus, the energy consumed is very small.

VI. Conclusions

In this paper, we analyzed the power consumption and performance of a wireless embedded smart camera system performing object detection, tracking and event detection. We made the analysis during different parts of the algorithm execution and for different message exchanges between camera nodes. The power consumption of the camera board depends highly on the amount of activity in the scene. Thus, we measured the operating current when there were different number of objects in the scene. To measure the current, we used a precise oscilloscope and a 1-ohm resistor configuration. We also presented a graph of the energy consumption for different tasks performed in a camera's processor together with the number of instructions.

References

[1] N. Atsushi, K. Hirokazu, H. Shinsaku, and I. Seiji, *Tracking multiple people using distributed vision systems*, Proc. of IEEE Int'l Conf. on Robotics and Automation, vol. 3, pp. 2974-2981, 2002.

[2] M. Bramberger, A. Doblander, A. Maier, B. Rinner, and H. Schwabach, *Distributed embedded smart cameras for surveillance applications*, IEEE Computer, vol. 39, pp. 68–75, 2006.

(a) One tracker (b) Two trackers (c) Three trackers

Fig. 7. Measured voltages when different number of objects are being tracked.

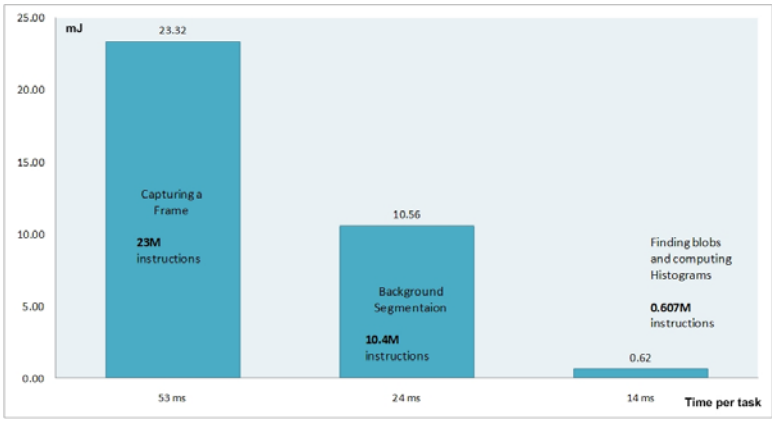


Fig. 8. Energy consumption (mJ) and the number of instructions for different tasks.

[3] M. Casares and S. Velipasalar, *Light-weight salient foreground detection for embedded smart cameras*, Proc. of the ACM/IEEE International Conference on Distributed Smart Cameras, 2008.

[4] M. Casares and S. Velipasalar, *Light-weight Salient Foreground Detection with Adaptive Memory Requirement*, to appear in the Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2009.

[5] P. Chen and et al., *Citric: A low-bandwidth wireless camera network platform*, Proc. of the ACM/IEEE International Conference on Distributed Smart Cameras, 2008.

[6] R. T. Collins, A. J. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, P. Burt and L. Wixson, *A system for video surveillance and monitoring: VSAM final report*, Technical report CMU-RI-TR-00-12, Robotics Institute, Carnegie Mellon University, May, 2000.

[7] R. T. Collins, A.J. Lipton, H. Fujiyoshi and T. Kanade, *Algorithms for cooperative multisensor surveillance*, Proceedings of the IEEE, vol. 89, no. 10, pp. 1456–1477, October 2001.

[8] I. Downes, L.B. Rad and H. Aghajan, *Development of a mote for wireless image sensor networks*, Proc. of Cognitive systems with Interactive Sensors, 2006.

[9] T. Ellis, *Multi-camera Video Surveillance*, Proc. of the Int'l Carnahan Conf. on Security Technology, pp. 228-233, 2002.

[10] Wu-chi Feng, Wu-chang Feng, M. L. Baillif, *Panoptes: Scalable low-power video sensor networking technologies*, Proc. of the ACM Conference on Multimedia, pp. 562–571, 2003.

[11] S. Hengstler, D. Prashanth, S. Fong, and H. Aghajan, *Mesheye: A hybrid-resolution smart camera mote for applications in distributed intelligent surveillance*, Proc. of the International Symposium on Information Processing in Sensor Networks, 2007.

[12] S. Khan and M. Shah, *Consistent labeling of tracked objects in multiple cameras with overlapping fields of view*, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 25, pp. 1355–1360, 2003.

[13] K. Kim, T. H. Chalidabhongse, D. Harwood, and L. Davis, *Real-time foreground-background segmentation using codebook model*, Journal of Real-time Imaging, vol. 11, pp. 172–185, 2005.

[14] R. Kleihorst, A. Abbo, B. Schueler, and A. Danilin, *Camera mote with a high-performance parallel processor for real-time frame-based video processing*, Proc. of the ACM/IEEE Int'l Conf. on Distributed Smart Cameras, pp. 106–116, 2007.

[15] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale and S. Shafer, *Multi-camera multi-person tracking for EasyLiving*, Proc. IEEE Int'l Workshop on Visual Surveillance, pp. 3–10, July 2000.

[16] P. Kulkarni, D. Ganesan and P. Shenoy, *The case for multi-tier camera sensor network*, Proc. of the ACM Workshop on Network and Operating System Support for Digital Audio and Video, 2005.

[17] B. P. Lai Lo, J. Sun and S. A. Velastin, *Fusing visual and audio information in a distributed intelligent surveillance system for public transport systems*, Acta Automatica Sinica, pp. 393-407, May 2003.

[18] K. Nguyen, G. Yeung, S. Ghiasi, and M. Sarrafzadeh, *A general framework for tracking objects in a multi-camera environment*, Proc. of the Int'l Workshop on Digital and Computational Video, pp. 200–204, 2002.

[19] N. Oliver, B. Rosario, and A. Pentland, *A bayesian computer vision system for modeling human interactions*, IEEE Trans. on Pattern Analysis and Machine Intelligence, pp. 831–834, 2000.

[20] Y. Panthachai, P. Keeratiwintakorn, *An Energy Model for Transmission in Telos-Based Wireless Sensor Network*, King Mongkut's Institute of Technology North Bangkok.

[21] M. Piccardi, *Background subtraction techniques: a review*, Proc. of the IEEE Int'l Conf. on Systems, Man and Cybernetics, vol. 4, pp. 3099–3104, 2004.

[22] J. Polastre, R. Szewczyk, and D. Culler, *Telos: Enabling Ultra-Low Power Wireless Research*, Porc. of the International Symposium on Information Processing in Sensor Networks, pp. 364–369, 2005.

[23] M.H. Rahimi, R. Baer, O.I. Iroezi, J.C. García, J. Warrior, D. Estrin

and M.B. Srivastava, *Cyclops: In situ image sensing and interpretation in wireless sensor networks*, Proc. of the Int'l Conf. on Embedded Networked Sensor Systems, pp. 192–204, 2005.

[24] A. Rowe, C. Rosenberg and I. Nourbakhsh, *A low cost embedded color vision system*, Proc. of the IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems, 2002.

[25] C. Stauffer and W. E. L. Grimson, *Learning patterns of activity using real-time tracking*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, pp. 747–757, 2000.

[26] T. Teixeira, D. Lymberopoulos, e. Culurciello, Y. Aloimonos, and A. Savvides, *A lightweight camera sensor network operating on symbolic information*, Proc. of the First Workshop on Distributed Smart Cameras, 2006.

[27] V. Tiwari, S. Malik and A. Wolfe, *Power Analysis of Embedded Software: A First Step towards Software Power Minimization*, IEEE Trans. on VLSI Systems, vol. 2, no. 4, 1994.

[28] S. Velipasalar, J. Schlessman, C. Chen, W. Wolf, and J. Singh, *A scalable clustered camera system for multiple object tracking*, EURASIP Journal on Image and Video Processing, 542808, 2008.

[29] Y. Wang, M. Casares and S. Velipasalar *Cooperative Object Tracking and Event Detection with Wireless Smart Cameras*, to appear in the Proc. of the IEEE International Conference on Advanced Video and Signal Based Surveillance, 2009.

[30] Y. Wang, M. Casares and S. Velipasalar *Detection of Composite Events Spanning Multiple Camera Views with Wireless Embedded Smart Cameras*, to appear in the Proc. of the ACM/IEEE International Conference on Distributed Smart Cameras, 2009.

[31] X. Yuan, Z. Sun, Y. Varol and G. Bebis, '*A distributed visual surveillance system*, Proc. of the IEEE Conf. on Advanced Video and Signal Based Surveillance, pp. 199 - 204, July 2003.

[32] Texas Instruments, *Intel PXA270 Processor*, datasheet, April 2005. 1997 [Revision 005].