Energy Consumption Tradeoffs in Visual Sensor Networks

Cíntia B. Margi[1], **Roberto Manduchi**[1], **Katia Obraczka**[1]

[1]Department of Computer Engineering
University of California, Santa Cruz (USA)

{cintia,manduchi,katia}@soe.ucsc.edu

Abstract. *Visual sensor networks are being increasingly employed as a tool for monitoring and surveillance of wide areas. Due to the relatively high power consumption characteristics of cameras, as well as their more stringent processing, storage, and communication requirements, it is important to carefully evaluate the different modes of operation of these systems, in order to devise energy–aware resource management policies. The ultimate goal is to deliver adequate application–level performance (e.g., maximize the probability of detecting events), yet maximally prolonging the system's operational lifetime. In this paper we present an accurate power consumption analysis for the different elementary tasks forming the duty cycle of a visual sensor node in a wireless camera network testbed. We also present a number of different duty cycle configurations, and provide direct energy consumption measurements for each one of them. Based on the energy consumption characterization we conducted for the elementary visual sensing tasks, we explore the possibility of predicting the lifetime of a visual sensor network system.*

1. Introduction

Continuous and pervasive monitoring of events over wide areas often necessitates a large number of networked, wireless sensors. There is widespread agreement within the sensor network community that multi-tier deployments, comprising both low– and high-level sensors, such as cameras, have great potential for a wide-range of current and upcoming applications. Visual sensors can cover larger fields of view and extract more substantial information about the scene than other simpler sensors such as temperature, humidity, and pressure [Kulkarni et al. 2005].

Sensor networks are typically energy constrained and hard to access once deployed, hence, self–organization and energy management are two fundamental factors determining their performance and lifetime. Accordingly, new mechanisms for network discovery and information flow management have been proposed [Heinzelman et al. 2000, Intanagonwiwat et al. 2000, Solis and Obraczka 2005]. Since the vast majority of networks studied to date employ very simple sensors (providing few bits per measurement and consuming little power) and consequently, the bulk of energy consumption is due to communication-related tasks, most research in power conservation for sensor networks has addressed solely communication issues. Examples include power-aware protocols at the MAC layer, data aggregation mechanisms, and strategies for predictive activation/transmission, topology control, power-aware routing, etc. Another common power conservation approach in sensor network deployments is the use of duty cycles [Mainwaring et al. 2002, Tolle et al. 2005], which alternate nodes between active and idle, low–power periods.

Most studies to date have typically considered the energy cost of sensing and data processing negligible when compared to communication–related tasks. This assumption, however, may not hold for higher-level sensors, such as cameras ([Barr and Asanovic 2003]) which produce large loads of data, requiring considerable processing for analysis (in order to extract semantic information) and/or compression. In addition, sensing itself can be highly power consuming. Analyzing and compressing visual data reduces communication cost but requires more processing, and in some cases may increase net energy consumption. It is clear that traditional network control strategies are not applicable to visual sensing networks.

We argue that a thorough energy consumption characterization is critical for designing effective resource management policies for wireless camera networks. In order to analyze and model power consumption over long periods of time, and therefore predict the lifetime of a node and change the control policy accordingly, it is useful to consider a number of *elementary tasks* whose scheduling and execution are controlled by a *resource manager* [Benini et al. 2000]. Each task has an associated power consumption cost and execution time (where both of these could be random variables). Stochastic models can then be employed for predicting the statistics of the time series of tasks, and therefore the expected energy consumed over a period of time [Mini et al. 2003].

2. Focus and Background

This paper provides a quantitative power consumption and temporal analysis of a set of basic tasks as well as duty cycles representative of activities carried out by wireless camera networks targeting surveillance applications. We focus on the wireless camera network testbed we have been building, dubbed *Meerkats*, which is based on the Crossbow Stargate [Crossbow 2004] board, a relatively powerful single-board computing platform that is well-known in the sensor network community. The 400 MHz XScale processor hosted in a Stargate can perform in real time vision algorithms of moderate complexity, including image compression for reduced transmission load. The board runs the Linux operating system, simplifying code prototyping and testing. Open source drivers for USB cameras and wireless 802.11b cards are available. We should point out that other, less power–hungry camera nodes are available, most notably Cyclops [Rahimi et al. 2005], a single board solution integrating a CMOS camera and micro-controller. However, the current image resolution, computational power, and memory available in Cyclops are not sufficient for the visual tasks required by our project. Furthermore, we should point out that our work on designing effective resource management policies is completely orthogonal, and thus complementary, to efforts that focus on designing low power platforms.

Previous work has provided energy consumption values for other types of simpler sensor network platforms, such as Berkeley's motes [Woo 2000, Shnayder et al. 2004], or for small granularity tasks involving switching the state of a microprocessor [Benini et al. 2000]. A detailed low–level analysis of the power cost for lossless compression and transmission in a StrongARM processor was presented in [Barr and Asanovic 2003]. In our previous work [Margi et al. 2006], we characterized the energy consumption of the Meerkats node using a benchmark that runs "basic" tasks such as processing, flash memory access, image acquisition, and communication over the network. In our characterization, we considered the various hardware states the system switches through as it executes these benchmarks, e.g., different radio modes (sleep, idle,

transmission, reception), and webcam modes (off, on, and acquiring image). We reported both steady-state and transient energy consumption behavior obtained by direct measurements of current with a digital multimeter.

Our goal in this work is to define and characterize a set of elementary tasks representative of visual sensor network activities at a coarse enough granularity to enable modeling different policies by simple task concatenation. For example, one of the tasks considered is image acquisition and compression. Another task is loading the software modules that are necessary for the wireless card to operate, while yet another one is keeping the CPU in sleep mode for a certain period of time.

Furthermore, given that the basic control policy of a Meerkats node is based on regular duty cycles synchronous across all nodes in the network, in this paper we also validate the hypothesis that energy consumption for a given duty cycle can be obtained by the composition of the energy consumed by the corresponding individual tasks. An example of a duty cycle is as follows. The webcam in the node takes snapshots at regular times, and, if it detects something of interest, transmits the data using the node's wireless card to another node or to a sink. Clearly, the image acquisition period and the policy that is used to decide whether to transmit an image or not affect the *miss rate*, i.e., the probability that moving object in the scene went undetected, either because no snapshot was taken of it, or because the image was not transmitted. The power consumption depends on the same parameters: a higher frame and transmission rate drains more current, on average, from the battery.

In fact, the implementation of a duty cycle is a more complex problem than may look at first sight. A number of operational possibilities must be evaluated, each with different characteristics of power consumption and execution time. Consider, as a simple example, the state of the node between two subsequent snapshots. In order to reduce power usage, the CPU should be put to sleep for all the time it is not used. However, putting the system to sleep requires first unloading the software modules used for controlling the webcam and the wireless card, saving the context, setting the interruption to wake up the processor and powering off hardware components. When the system wakes up, these modules need to be reloaded and context restored. If the period between two consecutive image acquisition is too short, there may not be enough time to perform all these operations, and the system would have to be kept in idle mode instead. As another example, consider the case with a camera node in a crowded environment, detecting a new event (e.g., a moving person) in most of the images that it takes. Depending on the power cost of the computation that is necessary for event detection, it may be the case that the node is better off transmitting each image without first analyzing it, rather then spending energy by first processing it.

To validate our hypothesis that the energy consumed in a duty cycle, as well as its execution time requirements, can be estimated by simply adding up the relevant quantities for each elementary task involved, we compare direct measurements (given by a digital multimeter) against predictions based on the concatenation of the corresponding elementary tasks. We present results for a number of duty cycle variations. Our next step is to develop and validate a probabilistic energy consumption model based on task composition in order to predict a node's lifetime.

This paper is organized as follows. Section 3 describes the current Meerkats testbed, including the node hardware and software organization. The elementary tasks a Meerkats node execute are described in Section 4, while the duty cycle analysis is presented in Section 5. Conclusions and future work are discussed in Section 6.

3. The Meerkats Prototype

This section describes the current Meerkats visual sensor network testbed. It presents a detailed description of the Meerkats node's hardware and software organization.

3.1. Hardware Organization

The Meerkats node (shown in Figure 1) is based on the Crossbow's Stargate [Crossbow 2004] platform, which has an XScale PXA255 CPU (400 MHz) with 32MB flash memory and 64MB SDRAM. PCMCIA and Compact Flash connectors are available on the main board. The Stargate also has a daughter board with Ethernet, USB and serial connectors. We equipped each Stargate with an Orinoco Gold 802.11b PCMCIA wireless card and a Logitech QuickCam Pro 4000 webcam connected through the USB. The QuickCam can capture video with resolution of up to 640x480 pixels.

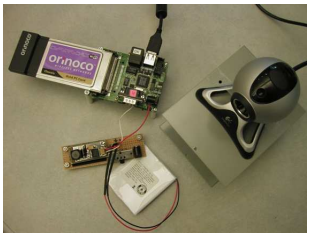


Figure 1. Meerkats node

The Stargate can be powered through a 5V DC adaptor or through a battery. Both the main– and daughter boards have battery input, but only the daughter board has a DC input. Since we are using the USB connector located on the daughter board, we need to power the Stargate through the daughter board with 5V. To achieve this, we use a customized 7.4 Volt, 1000mAh, 2 cell Lithium-Ion (Li-Ion) battery (the white rectangle shown in Figure 1), manufactured by Energy Sales, Inc and an external DC-DC (with efficiency of about 80%) switching regulator. The operating system is Stargate version 7.3 which is an embedded Linux system (kernel 2.4.19).

The choice of Crossbow's Stargate [Crossbow 2004] as the Meerkat's node main component was based on a number of considerations. First, given that our focus was to devise effective resource management policies for visual sensor networks, it made sense to use of-the-shelf components. Choosing a platform that runs an open source operating system was also important in developing an open source system. And, since we selected a webcam as the visual sensor, we needed a board with a USB connector. As previously discussed, we also needed a platform that provided reasonable processing and storage capabilities.

Currently, the Meerkats testbed is composed of eight visual sensor nodes, and one laptop (Dell Inspiron 4000 with PIII CPU running Linux (kernel 2.4.20) and equipped with an Orinoco Gold 802.11b wireless card) acting as the information sink.

We should point out that our approach is completely orthogonal, and thus complementary, to efforts that focus on designing low power platforms. A noteworthy example is Cyclops [Rahimi et al. 2005], a low-power vision sensor node consisting of a CMOS camera, processor, and memory, all integrated on the same board. In the experiments reported in [Rahimi et al. 2005], the Cyclops sensor board was connected to a MICA2 Mote [Woo 2000]. However, as previously discussed, in their current instantiation, Cyclops would not have sufficient capability (in terms of processing power and storage capacity) to run the types of vision algorithms that we implement on the Meerkats node.

3.2. Software Organization

The Meerkats' visual sensor node current software organization, shown in Figure 2, consists of three main components, namely the Resource Manager, Visual Processing, and Communication modules.

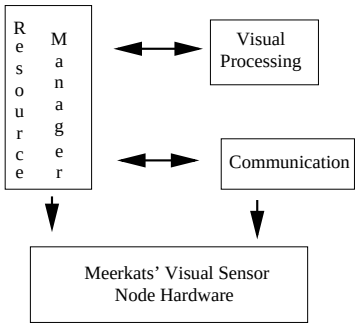


Figure 2. Meerkats software organization

Resource Manager

The Resource Manager is the main thread of control running on the Meerkats node. It controls the activation of the webcam and wireless network card in order to perform image acquisition/processing and communication-related tasks (e.g., transmit an image), respectively, as needed.

For energy conservation, the Resource Manager has the Meerkats sensor node operating on a duty cycle basis, i.e., the node periodically wakes up, performs some task as needed, and goes back to either "idle" or "sleep" mode. The sleep mode has the lowest power consumption, but switching the node to and from sleep mode requires a non-negligible amount of time. This corresponds to the sum of the time it takes to physically switch the operating mode of the CPU, and to load/unload necessary software modules (see below). The node is in "idle" mode when it is not performing visual acquisition, processing, or transmission, and when it is not loading/unloading software modules. Note that this may or may not correspond to the "idle" mode of the processor [Intel 2004]: the operating system determines whether/when to perform this switch, in a way that is transparent to the user. A node's idle mode has in fact a number of variations, depending on which system component (webcam or wireless card) are activated.

Switching the Meerkats node through its state space requires executing specific system calls which load/unload the necessary operating system modules. More specifically, to put the processor in *sleep* mode, one must execute the utility *sys_suspend*, providing as parameter the sleep interval. The node automatically wakes-up when the timer set

with the sleep interval expires. It should be noted that, in the Stargate platform, when the timer for the *sys_suspend* command expires and the node "wakes up", the wireless card goes to idle no matter what its previous state was. This is not the case for the webcam because of the way the *sys_suspend* script is implemented. *cardctl suspend* deactivates the wireless card, while *cardctl resume* activates it. The mechanism we use to deactivate the webcam is to remove the corresponding modules (*rmmod usb-ohci-sa1111*) from the kernel, while activating the webcam requires inserting back the corresponding modules (*insmod usb-ohci-sa1111*). In our experiments, we observed that, after loading the webcam modules, we still have to wait for some time (1 second turned out to be sufficient) until the camera is ready to acquire images. We thus had to explicit add that wait time to the node's duty cycle when appropriate.

Visual Processing

The Visual Processing module performs all vision-related tasks, including image acquisition, compression, and processing. It is invoked by the Resource Manager after the webcam has been activated. Upon completion, Visual Processing returns control to the Resource Manager with a parameter flagging whether an event has been detected. In that case, the Resource Manager will activate the wireless card and transmit the image. Otherwise, it may decide to put the node to sleep. In our experiments, in order to assess different energy consumption behaviors, we also considered just putting the node back to *idle* state without deactivating the wireless card nor the webcam. These experiments are described in detail in Section 5.

The goal is to detect events, in the form of image areas that are different from the background. If an event is detected, the relevant portion of the image is compressed and transmitted to the sink.

The visual processing involved in this experiment is quite simple. A background color image is stored (in a RAM file) and compared to the newly acquired image. When the difference in any color channel at a certain pixel between a new image and the corresponding background is larger than a pre-set threshold, the pixel is labeled as "foreground". The choice of the threshold is rather critical; mechanisms that choose a threshold based on the local color channel variance (or more complex schemes) have been proposed [Stauffer and Grimson 1999], but are computationally demanding, so we opted for a fixed threshold instead. We do, however, update those pixels in the background image that are not labeled as "foreground". The update consists in computing a convex linear combination of the color values of the pixel in the background and in the newly acquire image. This applies only for the pixels that were not labeled as "foreground". After all pixels have been labeled, we run a "blob detector", that is, we identify the connected components of the foreground map. Components that are too small (less than 300 pixels) are rejected from the map. This phase is very important, as it efficiently removes false positive due to noise. We use a fast implementation of a classical blob filter [Haralick and Shapiro 1992]. Finally, if there are remaining blobs with area larger than the threshold, a sub-image is identified by selecting the smallest rectangle that contains all of the remaining blobs. This sub-image is then JPEG–compressed for transmission. Note that Cyclops [Rahimi et al. 2005] also performs background subtraction and updating, however, even a relatively simple operation such as blob filtering will likely be very difficult to implement in Cyclops.

Communication

The third component of the Meerkats software architecture is the Communication module which is responsible for all communication-related tasks between a Meerkats node and the information sink, as well as communication among Meerkats nodes themselves.

The information sink runs a server program that listens for connection requests from sensor nodes, opens a connection, receives image files and renders images on the sink's console. This program is multi-threaded: one main thread listens for connection requests while individual requests are handled by separate threads. In other words, the server is always listening for requests and when a new request is received, the server will invoke a new thread to handle that request and establish the connection with the requesting sensor node. The corresponding image file will be transmitted over this connection. Once the server receives the whole image, it will be displayed on screen.

Currently, the server's screen is divided into four windows, which means that images from four different sensor nodes can be shown simultaneously. Images from a particular Meerkats node will use the same window, and the most recently received image will be displayed. Older images are saved on the server's hard disk.

The client program running on the sensor node initiates a connection with the server program running on the information sink whenever it needs to transmit an image. Note that images are sent one at a time. The current version of the client and server programs use TCP as the transport protocol for image transmission. However, one item for future work is to develop a more streamlined protocol on top of UDP.

Meerkats is a multi-hop wireless sensor network, and thus its nodes need to run a routing protocol in order to send data over the network. We picked the Dynamic Source Routing (DSR) protocol [Johnson and Maltz 1996], an on-demand routing mechanism especially designed for multi-hop wireless ad-hoc networks. The version of DSR run on Meerkats nodes was ported from the DSR kernel module available for the PocketPC Linux [Song 2001] to run on Linux kernel 2.4.20.

4. Elementary Tasks

In this section we define a set of elementary tasks that a Meerkats node executes as part of a duty cycle. The goal is to characterize each task by the amount of energy it consumes, and by the amount of time it takes for its execution. The granularity of each task should be coarse enough that task composition and scheduling is kept reasonably simple, yet fine-grained enough to allow for meaningful modeling.

In order to analyze the energy characteristics of each task, it is important to also consider the current *state* of the node. As discussed earlier, the Meerkats node consists of three basic hardware components, namely the processor core, the sensing core (i.e., the USB and the webcam), and the communication subsystem comprised of the PCMCIA wireless card. Each component is activated by loading the corresponding software module(s). Given its hardware subsystems, the possible states of a Meerkats node are:

- **Sleep** consumes the lowest power. The processor core is in sleep mode and all hardware components are deactivated. The current drawn by the node in this state is about 9 mA.

- In **idle** no specific task is performed, besides background OS processes. As pointed out earlier, the processor core may or may not be in its "idle" mode; this is controlled by the operating system. The current drawn is 139 mA when no additional hardware component is activated, with the wireless card and the webcam components drawing additional 161 mA and 185 mA respectively if active. We have verified that these measurements are additive, i.e., if both hardware components are active, the node draws approximately 139 mA+161 mA+185 mA=485 mA [Margi et al. 2006].
- When **active**, the node performs specific, finite–duration tasks involving the processor core possibly in cooperation with the other hardware components. Note that some of these tasks simply change the state of the node, such as from sleep to idle mode. These "state transition" processes take non-negligible time to execute and consume a non–negligible amount of energy. Therefore, they need to be accounted for, e.g., when planning a node's duty cycle, We assume that at the beginning and at the end of a task, the node is in idle mode (with or without hardware components activated). The only exceptions are: the task that brings the node from sleep to idle mode (wireless card module is automatically activated when the system is awaken, and thus the "wake up" task thus includes wireless card activation); and the task that brings the node from idle (with the webcam and wireless card deactivated) to sleep. We also assume that two tasks are never executed concurrently. This assumption is reasonable since tasks are dependent on each other for most duty cycles, i.e. in order to acquire an image, one needs to load the webcam modules first, and then based on the results from image processing, it will decide whether to transmit or not the image (which would would require wireless card to be activated).

Table 1 shows the cost, in incremental charge and duration, of each elementary task executed by the Meerkats node. The top part of the table summarizes the state transition tasks, while the bottom part of the table lists two other categories of tasks, namely *image acquisition/processing* and *transmission*. As discussed in Section. 3.2, depending on the specific duty cycle organization, a node may take an image at regular intervals and JPEG compress it. Or the node may take an image, analyze it, and if an event is detected, compress only relevant portions of the image. The size of the compressed image (or portion thereof) varies and, for this reason, we considered three different transmission tasks, depending on the size of the transmitted data. These different image sizes were obtained through experiments, considering different distances between the object and the camera.

Each task is characterized in Table 1 by its *duration* and by the *incremental charge* drawn, that is, the total amount of charge drawn by the board when executing the task minus the charge the node would draw if it was idle (with the same hardware modules loaded as at the beginning of task execution). For example, the task of acquiring and compressing an image (with only the webcam activated) draws additional 102 mC with respect to when the node is in idle (with the webcam activated). To compute the total charge drawn by the node during the execution of this task, we need to add this value to the charge drawn by a node in idle mode (with the webcam activated) over the same period of time (1.22 s). Thus the total charge drawn in this case would be $102\ mC + (1.22\ s * (139 + 185)\ mA) = 497\ mC$. Suppose now that the the wireless card was active during the execution of this task. The total charge drawn in this case would be $102\ mC +$

Table 1. Incremental charge drawn (in milli-Coulombs) and duration (in milli-seconds) for the elementary tasks considered. Average values and standard deviations were computed over twenty runs.

Task	Incremental Charge (mC)	Duration (ms)
Activate webcam	126 ± 5	1301 ± 27
Deactivate webcam	79 ± 6	356 ± 11
Activate wifi	104 ± 18	2316 ± 329
Deactivate wifi	54 ± 19	374 ± 80
Put to sleep	75 ± 12	389 ± 33
Wake up	496 ± 3	2798 ± 10
Acquire and compress image	102 ± 4	1227 ± 17
Acquire and process image (no event detected)	88 ± 6	1129 ± 21
Acquire, process and compress image portion(event detected)	105 ± 12	1244 ± 56
Send 7KB (compressed image portion)	12 ± 1	123 ± 12
Send 20KB (compressed image portion)	20 ± 1	207 ± 11
Send 28KB (compressed image portion)	23 ± 2	248 ± 15
Send 48KB (full compressed image)	34 ± 2	356 ± 16

$(1.22\ s * (139 + 185 + 161)\ mA) = 693\ mC$. Obviously, the charge drawn, multiplied by the battery voltage, gives the amount of energy dissipated. Since battery capacities are normally expressed in Ampère-hour, it seems more sensible to use the charge drawn to measure energy consumption (1 Ampère-hour = 3600 Coulombs).

A number of interesting observations can be drawn on inspection of Table 1. For example, we notice that the state transition tasks can take a long time to execute (1.3 seconds for webcam activation and almost 3 seconds to bring up the board from sleep state). Likewise, the overall charge drawn can be relatively high. Also note that acquiring and compressing an image consumes over 2.5 times more than transmitting the compressed image[1]. All these considerations must be taken into account when specifying the system's duty cycle organization.

Another important observation is that, if no event is detected, compressing the full image draws about 14 mC more than analyzing the image with our foreground detection algorithm. In the case an event is detected, analyzing and compressing an image portion draws about 3 mC more than just compressing the full image [2]. This is information is important when comparing the energy cost of two possible strategies, namely: compressing and transmitting each image versus transmitting only compressed portions of images when events are detected. If no event is detected, the second case will allow savings of about 14 mC plus the cost of transmitting the full image (34 mC). Consider now the case an event is detected. The cost of analyzing and compressing an image portion draws about the same charge as just compressing the full image. Depending on the size of the

[1] An equivalent uncompressed image would have about 200KB.

[2] Note that there is a large standard deviation in this case because the size of the compressed image may vary significantly.

compressed image portion, this reduces the transmission cost with respect to transmitting the full compressed image. Intuitively, this means, that, even if the rate of event detection is high, on-board event detection may lead to significant energy savings.

5. Duty Cycle Analysis

The energy consumption characterization of the elementary tasks presented in the previous section makes it possible to compare different duty cycle organizations that yield the same functionality. For example, one possibility is to never put the node to sleep between two consecutive image measurements, thereby avoiding the overhead associated with activating and deactivating hardware modules as well as transitions from idle to sleep mode. Would this be more energy efficient than a cycle that puts the system to sleep after each image acquisition and transmission? Clearly, the answer depends on the interval between two consecutive image acquisitions: if the interval is larger than some critical value, the saving associated with being in sleep mode offsets the overheads due to state transitions. In fact, differences between duty cycles may be more nuanced than this. A duty cycle may branch out into different paths depending on whether an event was detected or not. Additionally, the image acquisition periods needs not be constant. For example, if a moving body was detected in the scene, it may be desirable to take a number of pictures of it while it is still in the field of view of the camera. Hence, the node may wait for a shorter period of time than usual before taking the next shot.

These simple considerations highlight the fact that the energy analysis of a duty cycle is not trivial, and may require appropriate statistical modeling [Mini et al. 2005]. The key insight, however, is that a given duty cycle is a sequence of elementary tasks, possibly with conditional branching points. Hence, the energy consumed by the duty cycle (and its duration), can, in principle, be predicted by simply adding up the energy (and the duration) of each single task in the sequence. In this section we validate this important hypothesis by way of direct current measurements.

We consider six different possible duty cycles: three of them do not include image analysis for event detection (Figure 3), while the other three include visual event detection (Figure 5).

The first three duty cycles, which do not include image analysis for event detection, can be described as follows:

- **Duty cycle (a)**: the node simply takes an image, compresses it and transmits, and then stays in idle for a period $T_1 = 5$ seconds. All hardware components are always activated. Figure 3-(a) summarizes the tasks executed.
- **Duty cycle (b)**: similar to the previous case, but each hardware component is activated only when it is needed, and deactivated as soon as possible. Figure 3-(b) summarizes the tasks executed.
- **Duty cycle (c)**: different from the second case in that the system is put to sleep, rather than kept idle for the same period T_1 of time. This duty cycle includes the additional task of wireless card deactivation after system wake–up. Figure 3-(c) summarizes the tasks executed.

The temporal profile for the current drawn during duty cycle (b) is shown in Figure 4. As depicted in the figure, one can see the variations in current drawn to activate the

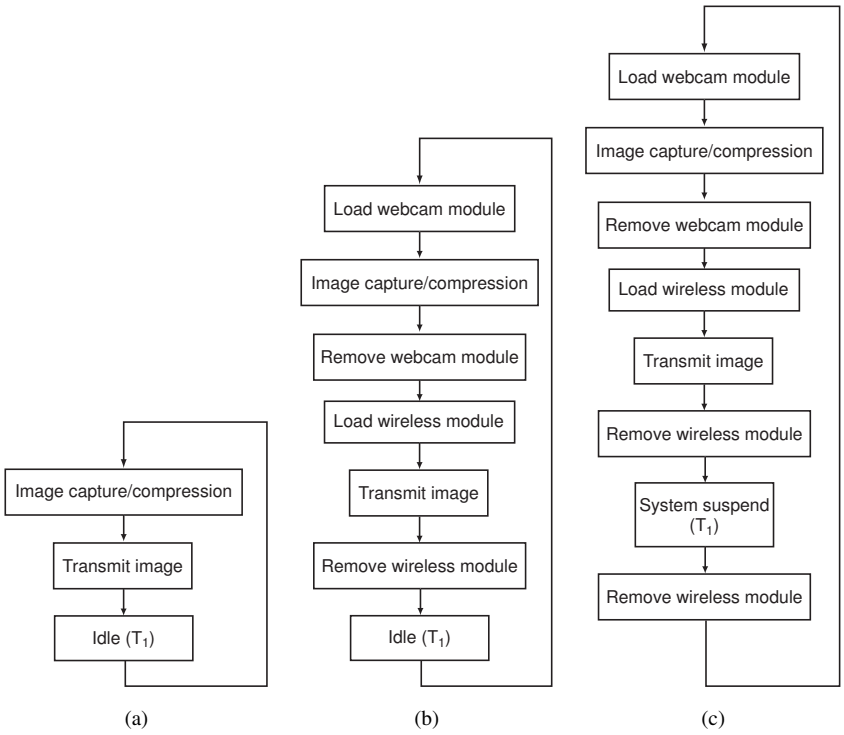


Figure 3. Elementary task sequences of duty cycles that do not include image analysis for event detection, being (a): duty cycle of type (a), (b): duty cycle of type (b), and (c): duty cycle of type (c), respectively.

webcam, followed by image acquisition, webcam deactivation, wireless card activation, image transmission, wireless card deactivation, and finally the node waits in idle mode until next cycle starts.

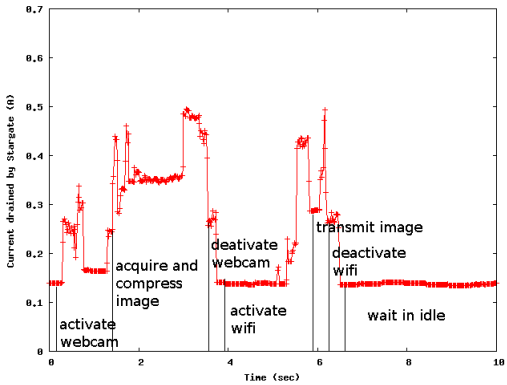


Figure 4. Temporal profile of current drawn in one cycle of duty cycle (b).

Tables 2 and 3 present the measured charge drawn over each duty cycle, together with the cycle duration. We also show the predicted values based on the characterization of each elementary task given in Section 4.

From Table 3, note that the cycles have very different durations. This can be explained by the fact that, while the amount of time (T_1) the system is idle or in sleep mode is constant, the different duty cycles include different transition tasks. The significant

Table 2. Predicted and measured (averaged over twenty tests) charge drawn during the cycle are shown in the table below, along with the relative prediction errors.

	Charge (mC)		
Duty cycle	Predicted	Measured	Relative Error
(a)	3342	3038 ± 19	9%
(b)	2302	2433 ± 43	-6%
(c)	2329	2308 ± 20	1%

Table 3. Predicted and measured (averaged over twenty tests) cycle durations are shown in the table below, along with the relative prediction errors.

	Duration (ms)		
Duty cycle	Predicted	Measured	Relative Error
(a)	6584	6393 ± 38	3%
(b)	10931	11577 ± 78	-6%
(c)	14491	13999 ± 12	3%

differences in the duration of the various duty cycles show that the overhead associated with state transitions contribute significantly to the overall cycle duration. Hence, these values should not be used to measure the efficiency of the different cycles, but simply to validate the prediction based on task composition. Clearly, in order to obtain the same duty cycle period, the amount of idle/sleep time should be carefully tuned in each cycle.

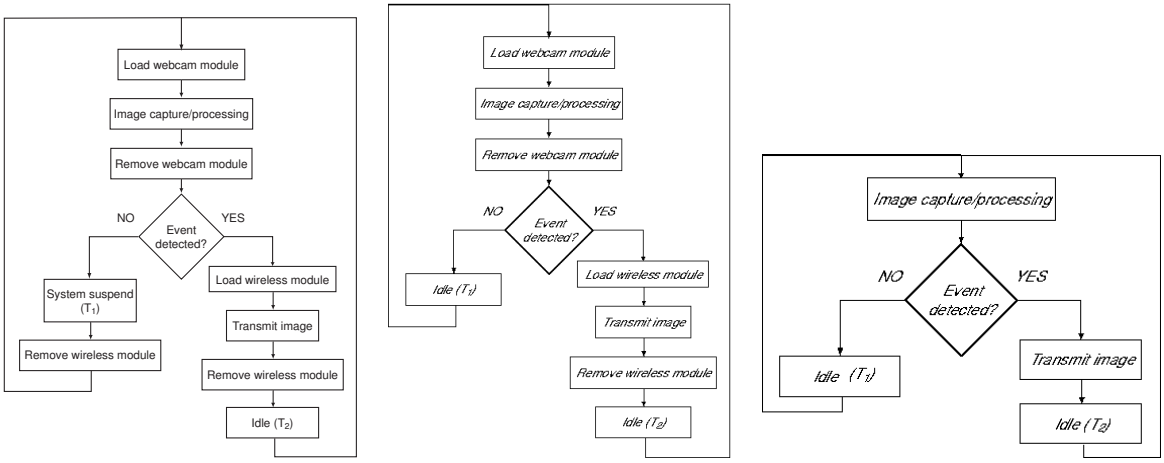
The relative prediction error E_q for the charge drawn during one cycle is defined as 1 minus the ratio between the measured– and the predicted charge (the relative prediction error E_t for the cycle duration is similarly defined). It is seen from Tables 2 and 3 that E_q for all the duty cycles of Figure 3 is less than 10%. During our measurements for the duty cycles, we noticed some variation in the cycle duration (see the standard deviation for the third row on Table 3). By observing the temporal profiles of current drawn, we noticed that some of them presented a (variable) delay between the loading wireless card modules and actual transmission (node spend this delay mostly in idle mode). This delay is introduced by the program that transmits the image trying to connect to the server (it uses TCP) and most likely occurred due to wireless interference. The variation in the duty cycle duration reflects in the charge necessary to execute the duty cycle.

Figure 5 shows three different duty cycles that include visual event detection. The basic idea is that if no event is detected, the system is put in sleep or idle mode for $T_1 = 5$ seconds. Otherwise, the system remains idle for $T_2 = 3$ seconds. The three duty cycles can be described as follows:

- **Duty cycle (d)**: the node starts in idle mode without any modules active. Then it loads the webcam modules, acquires and process an image and then removes the webcam modules. If no event is detected, the system is put into sleep mode for $T_1 = 5$ seconds. Otherwise, the system loads the wireless card modules, transmits the image, removes wireless card modules and then stays in idle for a period T_2 = 3 seconds. Figure 5-(a) summarizes the tasks executed.
- **Duty cycle (e)**: this case is a variation of the previous one, where the node stays

in idle mode instead of going to sleep in case no event is detected. Figure 5-(b) summarizes the tasks executed.

- **Duty cycle (f)**: this is the simplest case, where all modules remain active all the time and the system stays in idle in between image acquisitions. Figure 5-(c) summarizes the tasks executed.



(a) (b) (c)

Figure 5. Elementary task sequences of duty cycles that include image analysis for event detection, being (a): duty cycle of type (d), (b): duty cycle of type (e), and (c): duty cycle of type (f), respectively.

The temporal profile for the current drawn during Duty cycle (d) when no event is detected is shown in Figure 6. The figure depicts the current draw when the node is sleeping (9mA), and then it *wakes up*, going to idle mode which activates the wireless card (as mentioned before, this is done automatically within the kernel), then deactivates the card, activates the webcam, acquires image and processes it. Since no object was detected, it deactivates the webcam, and finally goes into sleep mode.

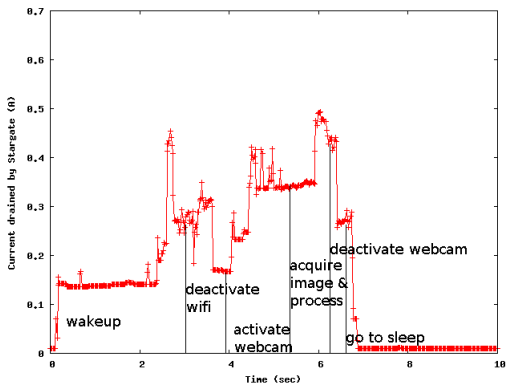


Figure 6. Temporal profile of current drawn during a cycle of duty cycle type (d) when no event is detected.

The predicted and measured charge drawn for each branch of each duty cycle, along with the relative prediction errors, are shown in Table 4. The predicted and measured cycle durations, along with the relative prediction errors, are summarized in Table 5.

Table 4. Predicted and measured (averaged over twenty tests) charge drawn during the cycle are shown in the table below, along with the relative prediction errors. Note that the event detection branches in duty cycles (d) and (e) are the same, and thus values are omitted in this table.

	Charge (mC)		
Duty cycle	Predicted	Measured	Relative Error
(d) - no event	1610	1602 ± 15	1%
(d) - event	1989	2176 ± 115	-9%
(e) - no event	1583	1688 ± 7	-7%
(f) - no event	3072	2915 ± 22	5%
(f) - event	2316	2127 ± 18	8%

Table 5. Predicted and measured (averaged over twenty tests) cycle durations are shown in the table below, along with the relative prediction errors. Note that the event detection branches in duty cycles (d) and (e) are the same, and thus values are omitted in this table.

	Duration (ms)		
Duty cycle	Predicted	Measured	Relative Error
(d) - no event	10973	11000 ± 13	0%
(d) - event	8948	9536 ± 289	-7%
(e) - no event	7786	8300 ± 17	-7%
(f) - no event	6129	6199 ± 48	-1%
(f) - event	4452	4398 ± 34	1%

Note that the relative error for both the charge as well as for the duration presented in Tables 4 and 5 is less than 10%, for all cases. Again, similarly to what happened to our measurements for duty cycle (c), we noticed a lot of variation in the duty cycle duration for (d) and (e) when an event is detected (see the standard deviation for the second row on Table 5). Two aspects contribute to this variation: first, there is a (variable) delay between the loading wireless card modules and actual transmission (node spend this delay mostly in idle mode); and second, there is the variation in the camera perception of the detected object (due to the different positions the object could be relative to the camera), which affects the processing of the image and compression of the sub-image (note the standard deviation presented for *acquire, process and compress image portion (event detected)* on the tenth row of Table 1), and the actual duration of the transmission of the sub-image (note the different transmission duration for the different image sizes presented in Table 1).

6. Conclusions

Wireless networks of embedded visual processing nodes are very attractive for surveillance and monitoring. We have presented an analysis of the power requirement and execution time of the elementary tasks that compose a typical duty cycle of a visual sensing node based on the Crossbow Stargate board. This analysis is necessary to predict the node's lifetime when it is battery–operated, and to choose the most appropriate design parameters. Our study has highlighted the fact that activation and deactivation of hardware components, as well as the transition between idle and sleep mode of the node's

processor core, may require considerable overhead energy and may take substantial time. We also showed how elementary tasks can be combined together to form different duty cycles performing the same function but with rather different energy requirements.

In order to achieve our ultimate goal of conducting a comprehensive study of visual sensor network's lifetime, we divided the work in three main steps: (1) characterize energy consumption of the node in our testbed ([Margi et al. 2006]); (2) determine the cost (in terms of charge and duration) of the main tasks in a visual sensor network and the possible duty cycles, as well as validate the hypothesis that energy consumption for a given duty cycle can be obtained by the composition of the energy consumed by the individual tasks (work reported in this paper); and finally (3) develop and validate an energy consumption model based on task composition to determine a visual sensing node's lifetime (our ongoing work). Future work will apply this energy models for predicting lifetime under more complex and cooperative control strategies.

It is worth to point out that, although our work was done considering a specific visual sensing platform, i.e., the Meerkat's node, the methodology we used could be applied to other platforms and applications. Essentially, the methodology we proposed includes: (1) characterizing energy consumption of the platform; (2) determining the cost (in terms of charge and duration) of the tasks representative of the target application; (3) extend/adapt the energy model based on task composition to determine the node's life time. As part of our future work, we intend to validate this approach by developing a model for lifetime prediction for different hardware platforms and sensor network applications.

7. Acknowledgements

The authors would like to acknowledge Gefan Zhang and Xiaoye Lu, members of the Meerkats project group, for their support in the development of the Meerkats testbed; and also all reviewers for their insightful comments.

The Meerkats project is supported by NASA under contract NNA04CK89A. Cintia Margi was supported by a scholarship from CNPq/Brazil from August/2001 until August/2005.

References

Barr, K. and Asanovic, K. (2003). Energy aware lossless data compression. In *The First International Conference on Mobile Systems, Applications, and Services*, San Francisco, CA.

Benini, L., Bogliolo, A., and Micheli, G. D. (2000). A survey of design techniques for system-level dynamic power management. *IEEE Transactions on VLSI Systems*, 8(3):299–316.

Crossbow (2004). Stargate. http://www.xbow.com/.

Haralick, R. and Shapiro, L., editors (1992). *Computers and Robot Vision*. Addison-Wesley.

Heinzelman, W., Chandrakasan, A., and Balakrishnan, H. (2000). Energy-efficient communication protocol for wireless microsensor networks. In *33rd Hawaii International Conference on System Sciences (HICSS '00)*, Hawaii.

Intanagonwiwat, C., Govindan, R., and Estrin, D. (2000). Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Sixth Annual International Conference on Mobile Computing and Networking, (MobiCom 2000)*, pages 56–67. ACM.

Intel (2004). Intel PXA255 applications processors developer's manual. http://www.intel.com/design/pca/applicationsprocessors/manuals/278693.htm.

Johnson, D. B. and Maltz, D. A. (1996). Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers.

Kulkarni, P., Ganesan, D., and Shenoy, P. (2005). The case for multi-tier camera sensor networks. In *International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2005)*.

Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D., and Anderson, J. (2002). Wireless sensor networks for habitat monitoring. In *First ACM Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA.

Margi, C. B., Petkov, V., Obraczka, K., and Manduchi, R. (2006). Characterizing energy consumption in a visual sensor network testbed. In *2nd International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom 2006)*, Barcelona, Spain.

Mini, R. A., do Val Machado, M., Loureiro, A. A., and Nath, B. (2005). Prediction-based energy map for wireless sensor networks. *Ad Hoc Networks Journal (Special Issue on Ad Hoc Networking for Pervasive Systems)*, 3(2):235–253.

Mini, R. A., Loureiro, A. A., and Nath, B. (2003). Prediction-based energy map for wireless sensor networks. In *Proceedings of IFIP-TC6 8th International on Conference Personal Wireless Communications (PWC 2003)*, pages 12–26.

Rahimi, M., Baer, R., Iroezi, O. I., Garcia, J. C., Warrior, J., Estrin, D., and Srivastava, M. (2005). Cyclops: In situ image sensing and interpretation in wireless sensor networks. In *SenSys 2005)*.

Shnayder, V., Hempstead, M., Chen, B., Werner-Allen, G., and Welsh, M. (2004). Simulating the power consumption of large-scale sensor network applications. In *ACM SenSys 04*, Baltimore, MA.

Solis, I. and Obraczka, K. (2005). Efficient continuous mapping in sensor networks using isolines. In *Mobiquitous 2005*.

Song, A. (2001). Piconet ii - a wireless ad hoc network for mobile handheld devices. http://piconet.sourceforge.net/.

Stauffer, C. and Grimson, W. (1999). Adaptive background mixture models for real-time tracking". In *IEEE Int'l Conf. on Computer Vision and Pattern Recognition*.

Tolle, G., Polastre, J., Szewczyk, R., Turner, N., Tu, K., Buonadonna, P., Burgess, S., Gay, D., Hong, W., Dawson, T., and Culler, D. (2005). A macroscope in the redwoods. In *Third ACM Conference on Embedded Networked Sensor Systems (SenSys)*.

Woo, A. (2000). Mote documentation and development information. http://www.eecs.berkeley.edu/ awoo/smartdust/.