Solar-Powered Smart Wireless Camera Network for Ubiquitous Outdoor Monitoring

Kevin Abas*, Leland Miller *, Katia Obraczka*
, Guilherme Silva†
*School of Engineering, University of California Santa Cruz, CA USA
†University of S$\tilde{a}$o Paulo, SP Brazil

***Abstract*—In this paper, we present SlugCam, a solar-powered wireless smart camera network platform that can be used in a variety of outdoor applications including surveillance of public spaces, habitat and environmental monitoring, wildfire prevention and detection, to name a few. SlugCam was designed to be deployed and left unattended for extended periods without requiring regular maintenance, e.g., frequent battery replacement. The system is built with off-the-shelf components which not only keeps it modular and low cost, but also facilitates its prototyping, reproducibility, and evolution. SlugCams onboard processing capability allows computer vision software to run locally contributing to the systems autonomy. In addition to a detailed description of SlugCam, this paper presents an extensive power characterization of the systems operation and showcases its deployment in an outdoor scenario.**

I. Introduction and Motivation

Smart wireless visual sensor networks are at the crossroads of wireless sensor networks, embedded systems, and computer vision. They have become increasingly more pervasive as a direct result of advances in wireless communication technology as well as visual sensor processing. Some smart camera platforms are able to not only identify objects in the scene, but also determine an objects behavior [1], [2]. Motivated by their new capabilities, which are enabled by their greater processing power and lower cost, smart wireless camera networks have found a wide range of application including environmental and habitat monitoring, surveillance of outdoor spaces, wildfire detection and prevention, to name a few [3]. Another important breakthrough is the availability of affordable energy harvesting sources, such as solar energy [4]. Energy harvesting techniques allows wireless smart camera network systems to be deployed for extended periods of time in remote areas without requiring frequent maintenance (e.g., battery replacement).

In this paper, we introduce SlugCam, an open-source smart wireless visual sensor network system built with low-power, off-the-shelf components. SlugCam offers a complete system solution from the hardware to its Web-based user interface. It targets modularity and extensibility so that the system can adapt to the needs of existing applications as well as evolve to fulfill the requirements of emerging ones.

The rest of the paper is organized as follows. In Section 2 we provide an overview of SlugCam. Sections 3 and 4 describe SlugCams hardware and software architectures, respectively. In Section 5, we present a thorough characterization of the systems power consumption as well as our experience deploying SlugCam outdoors. Section 6 presents a discussion of related work and a summary of the paper and Section 7 concludes the paper with some directions for future work.

II. System Overview

SlugCam has been designed with readily available off-the-shelf devices to reduce cost and allow for rapid development. Each device has been chosen carefully to meet our power consumption efficiency requirements. Another important design consideration was to ensure sufficient on-board processing capabilities which allows SlugCam to perform visual processing tasks locally, and thus be more selective of the video it records. The onboard camera is normally off and gets turned on by a passive infrared (PIR) sensor when motion is detected. Additionally, SlugCam is able to adapt its operation to the available energy remaining in its rechargeable battery. By combining a low-cost and low-power hardware design with energy-aware software, we believe SlugCam adequately balances energy efficiency and application-level requirements, an essential feature in wireless smart cameras [5]. Unlike previously designed smart cameras that run solely on battery as an energy resource, weve chosen to focus on having SlugCam run efficiently on solar power. Developing an open-source platform that could be used and adapted to other applications was an important design consideration. As a result, besides being a low-cost, open-hardware wireless camera node, SlugCam also includes an open-source web-based management system which we describe in detail in Section 4.

A. Energy-Efficient Hardware

One of SlugCams distinguishing features is that it includes two processing units: an MSP430 and a Raspberry Pi. The low-power MSP430 microcontroller [6] manages the amount of time that the more power consuming Raspberry Pi stays on. This is accomplished by having the MSP430 control PIR motion sensing while the Pi controls the rest of the system. When no motion is detected, only the MSP430 is on and all other components of the system, including the Pi, are off. As will be shown by our power characterization experiments in Section 5, this approach in substantial power savings, which allows SlugCam to use smaller size solar panel and battery, and also to run more processing-intensive computer vision algorithms on board.

Every external module used including the PIR, camera, and WiFi have the ability to be powered down as well, which allows SlugCam to adapt its operation according to the current state of its battery. For example, when not active, the WiFi module is in low-power sleep state until it receives a signal from the Pi to wake up. SlugCams different duty cycles (described in detail in Section 5) define the sequence of tasks executed by each SlugCam component and their corresponding power states. As previously pointed out, SlugCam's main processing device is the Raspberry Pi [7] which runs the nodes main functions including its visual processing task. Due to their high system resource usage, visual processing tasks are not always executed during SlugCams duty cycles. Depending on the current state of its battery, a SlugCam node will decide whether to perform visual processing onboard. For example, suppose that during the night there was a lot of activity and thus the system was frequently active and thus consuming more energy. The next day is cloudy/rainy so the solar power scavenging could not charge the battery sufficiently. Consequently, SlugCam will not run its vision software until its battery is charged beyond a certain level. This battery charge threshold is a configurable system parameter and influences the trade-off between energy efficiency and event detection requirements.

B. Outdoor, Off-Grid Requirements

As previously mentioned, one of SlugCam's main goals is the ability to not rely on the power grid or wired communication infrastructure, and operate autonomously and unattended for extended periods of time. Equipped with solar power scavenging and directional WiFi antennas, SlugCam is completely wireless and can be deployed in remote areas with readily available sunlight. Unlike a number of existing sensor network deployments [8] [9], SlugCam should very rarely need to be serviced for battery as shown by our experiments reported in Section 5. Pervasive wireless communication and lower-cost solar power technologies make SlugCam increasingly more technically viable and affordable. Unlike systems that offload large amounts of video footage to a more powerful central server, onboard visual processing capabilities contributes to SlugCams autonomy and lower communication bandwidth requirements. These are critical features to enable outdoor deployments, possibly in remote environments, and significantly increase the range of applications for SlugCam.

C. Flexibility and Open-Source by Design

One of our main goals in the SlugCam project is to provide a platform that other researchers and the community at large can use and extend. As such, our system is both flexible and easily reproducible. As previously pointed out, the current SlugCam node is equipped with two sensors, namely the camera and the PIR, but can be extended with other sensors in order to adapt to different applications and their requirements. As will become clear from our description of SlugCams software system, its design allows it to easily adapt to new data collection and processing methods in a way that does not require modifications to the core components of the system. We also designed our system in a modular fashion so that when modifications to the core components must be made, they are constrained to a module of the system and do not require knowledge of how the rest of the system works. This has been proven to be an effective strategy in designing smart cameras as they need to stay up-to-date with performance and technology upgrades [8].

Another one of our main design decisions was choosing to keep both hardware and software designs as open as possible. Not only do we make our code available on a public online repository, but we have also used hardware and software that are both open source and very friendly to modification. We ensure that every part of the software we have developed is free and open source, including the network software platform, the database software, the Web frameworks, and utility libraries. In the same spirit we have also used off-the-shelf components whenever possible and will release schematics for any custom hardware so that the hardware can be easily reproduced. This helps in increasing both the flexibility and reproducibility of our system as well as encouraging developers to extend it and apply it to other applications.

D. Management Tool

SlugCam also includes a Web-based tool for managing its network of wireless camera nodes and an innovative user interface for managing the captured data. We consider this to be an essential part of the system and something that is often overlooked by smart camera system designers.

Smart camera networks such as SlugCam have the potential to collect large amounts of data that is fragmented over time and space. This presents unique challenges in the design of an effective, yet easy-to-use interface. As described in more detail in Section 4, SlugCams management tool allows end users to configure nodes, send them control signals, and access collected data. Its graphical interface makes the tool easy to use and very effective as a way to access and visualize collected data.

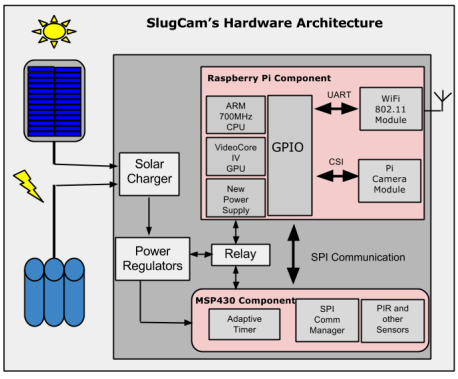

Fig. 1. SlugCam's Hardware Architecture.

III. SlugCams Hardware System

As shown in Figure 1, SlugCams hardware is organized as three sub-systems, namely: (1) the Rasperry-Pi controlled component, (2) the MSP-430 controlled component, and (3) the solar harvesting component. In the remainder of this section, we describe each sub-system in detail.

A. The Raspberry-Pi Component

At the core of SlugCam is the Raspberry Pi (or sometimes referred to as the Pi), an open-hardware device intended to be used by the educational- and research community [7]. We have installed our own custom embedded Linux distribution to optimize performance and processing speeds using the Buildroot tool [10]. Buildroot allows users to build a customized embedded Linux operating system to fit the needs of an application. From Linux packages and tools such as openCV[11], all the way to more complex hardware specific configurations required by the devices hardware architecture, buildroot provides complete customizability. Creating a SlugCam-specific Linux operating system with all the packages we needed (and some that we did not need) really boosted the Raspberry Pis ability to process tasks quickly and efficiently. Last year the Raspberry Pi Foundation also released an advanced camera module that communicates to the Pi using the onboard Camera Serial Interface (CSI). The camera has the ability to take high resolution (2592 x 1944 pixels) images and capture 1080-pixel video at 30 frames per second [12]. Coupling the newly-released Pi camera with computer vision software (e.g., openCV) in order to record only relevant data aims at addressing the energy consumption-, bandwidth efficiency-, and application requirement (e.g., event detection accuracy) tradeoff. Just recently we have upgraded to the latest Raspberry Pi B+ due to its improved power efficiency which is demonstrated through experiments reported in Section 5. SlugCam includes a custom PCB daughter-card to house its external devices and available I/O ports. The daughter-card plugs directly into the Raspberry Pi and improves robustness and reliability of the systems hardware; it also reduces SlugCams form factor. Future revisions of SlugCam's daughter-card may include additional sensors to address the needs of a wider range of outdoor monitoring applications.

The Pi also records live current consumption data to be processed by SlugCam, allowing it to be a more power-aware system, and to adapt to what battery charge remains. Knowing its current usage and available battery level, the system can estimate how much time is available before it needs to enter a sleep state waiting for available solar power. The current sensor also allows us to determine what generic power consuming state the system is in. The idling state is when both microprocessors are on and idling, but all auxiliary sensors and the WiFi module have been disabled or put to sleep. When we discuss SlugCam being in a high powered state, the system is using both the WiFi module and auxiliary sensors like the camera module actively. The low-powered sleep state, refers to when only the PIR sensor is idling and the Pi is powered off, and the MSP430 has been put into its own sleep state waiting for a hardware interrupt from the PIR sensor to wake SlugCam up.

B. The Low-Power MSP430 Unit

Its ability to run on 0.5μA on standby and nearly 5 times less current when sleeping makes the MSP430 quite attractive in the context of systems where power-efficiency is of critical importance. In the case of SlugCam, by controlling how long the Raspberry Pi and its peripherals stay on, the MSP430 is able to decrease the overall power consumed by SlugCams nodes substantially (as illustrated by our power characterization experiments described in Section 5). It does so using an onboard timer and an external relay acting as a switch to completely cut power power to the Pi. The Raspberry Pi can dynamically change how much time it stays on by communicating to the MSP430, over a Serial Peripheral Interface (SPI) connection, how long it needs to wait before it sends an off signal to the relay(as shown in Figure 1). Both the small MSP430 chip and the relay circuit can be viewed in the prototype screenshot from Figure 2. Similar to smart camera systems such as the one described in [13], SlugCams multimodal sensing node uses a passive infrared sensor to decide when to wake up the rest of the system, i.e., the Pi and the camera. One direction of future work we plan to pursue is to augment SlugCams current sensor fusion technique with other sensors, e.g., an audio sensor similar to the one the Citric platform [2] uses to determine if video needs to be recorded.

While WiFi has been one of the most ubiquitous wireless communication technology, its energy efficiency has always been an obstacle for its use in embedded systems. Recently, however, energy efficient WiFi modules have been developed specifically for embedded applications. SlugCam uses the RN-171 WiFi module, as seen in the prototype photo Figure 2 which can be put into a low-power sleep state when not in use [14]. We chose the RN-171 not only for its low power consumption and high bandwidth, but also because it has the capability of operating in ad-hoc mode. This ad-hoc feature will allow for networks to be extended farther distances, further assisting the systems ability to operate in more remote locations. One of the shortcomings of low-powered WiFi modules is the maximum bandwidth allowed by the UART serial communication line, which prevents us from making full use of WiFis transmission speeds. We believe, however, that favoring energy efficiency over transmission speed is more beneficial in the case of SlugCam. In addition to the module efficiencies, the raspberry Pi current consumption drops on average around 200 mA when the USB hub is not being used for external components.

C. SlugCams Power System: Going Solar

Unlike traditional battery powered smart cameras, SlugCam will rarely require battery replacement as it uses solar harvesting techniques and rechargeable batteries. In order to size the solar panel and battery for our power requirements while keeping cost and form factor low, we researched information

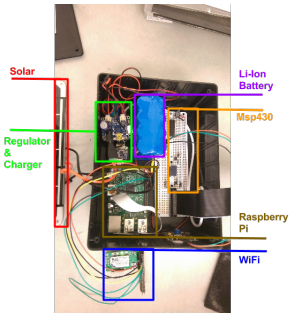

Fig. 2. The SlugCam prototype node.

on solar radiation levels in the USA and made accurate estimations of SlugCams power requirements. More specifically, for our planned SlugCam deployment, we looked up solar radiation in Santa Cruz, CA. This data was found on the National Renewable Energy Laboratory (NREL) website [15]. The NREL Web page shows solar radiation for every month at peak, average, and minimum levels and at every possible orientation of PV cells relative to the sun. Since one of our main goals is for the system is to operate unattended for long periods, we conducted worst case analysis, i.e., we needed to make sure that even with extended periods of little sunlight, SlugCam could still operate. SlugCam even if in degraded mode, i.e., offering only minimal service will still be effective for its needed applications. Knowing that the worst case sun exposure is in the winter months, we chose the month of January with a PV cell orientation facing south as the basis for our analysis which was the most effective for our location.

The energy produced by a photovoltaic cell is directly proportional to the active area of the cell, the amount of light falling in that area, and the conversion efficiency of the cell.

$$\text{Energy Output} = \text{Solar Energy} \times \text{PanelArea} \times \text{PVefficiency}$$

Current PV cell technology differs in efficiency, construction material, and cost. Today the most known PV cells materials are Crystalline silicon (e.g., monocrystalline silicon and polycrystalline silicon) and Thin Film (e.g., amorphous silicon, cadmium telluride, and copper indium gallium selenide) [16]. We found a monocrystalline solar panel with 6V and 19% of efficiency. Using the data from our solar radiation research we found Santa Cruz in the month of January receives 2-3 kWh per day.This unit of measurement represents the amount of energy accumulated over 1 square meter in a 24 hour period. Calculating how much energy can be converted to energy with the panel chosen of size 220mm by 175mm gave us the total energy produced for a whole day in our area.

$$\text{Battery Capacity(Amphrs)} = \text{Current Consumption(Amps)} \times \text{No Light Time(hrs)}$$

Now using the above equation and considering the Raspberry Pi consumes around 300–400 mA and during 8 hours per day on average, we have estimated the Pi taking 3200mAh total per day. Converting our current consumption per hour to joules showed our system needs less than what would be provided by a solar panel with all day direct sunlight.

Since PV cells can only absorb energy when there is sunlight and an efficient provided angle, batteries are needed to power the system when there is little to no sun. Lithium-ion (Li-ion) is currently the most promising battery system: it is used for portable consumer products as well as electric powertrains for electric vehicles. Its main drawbacks are that it is more expensive than other types of batteries (e.g., Nickel and lead acid systems) and needs protection circuits for safety. Another critical factor when choosing the battery was its capacity, i.e., how much energy it can store in ampere-hours (Ah). This is important because SlugCam will rely on its battery as its power source when there is no sun. Thus, our battery capacity estimation was based on how much time the target location for our deployment receives no solar radiation. Using on our capacity estimations and battery type comparison, we chose a 3.7V 17600mAh pack of Li-Ion batteries which will allow the system to last up to 4 days without sun and using the maximum performance previously specified. With the subcomponents weve chosen and our adaptive power aware system, SlugCam will last longer than this for certain less intensive applications.

In order to charge the battery adequately and safely, we chose to use Microchips MCP73871 charger [17] which allows us to both charge and make use of the extra energy available when full sunlight is available. The charger also had other useful features such as temperature monitoring for the battery which prevents charging the battery under extreme weather conditions. Status signals are given for current battery conditions like when the battery has finished charging or when the battery has a very low charge level.

IV. Software System

A. Overview

The software for the SlugCam project can be divided into three main areas as illustrated in Figure 3. There is the software that runs on the SlugCam node, the software that runs on a remote server, and the software that runs on the end-user device. The software running on the node is responsible for managing camera functions, running the computer vision algorithms, and communicating with the software on the remote server. The remote server runs software that is responsible for storing and managing data, and facilitating communication with both the camera nodes and the end-user devices. The user is able to view collected data and interact with the network from software running on end-user devices.

We decided early on that designing our software in a modular way would afford us several benefits, including ease of extensibility, ease of deployment, and the testability of our system. This aspect becomes more important in our desire to create a system that is easily extensible by future researchers who may not have the time or desire to fully understand our code, but would like to add modules for their unique needs.

Since various factors could inhibit communication to and from the camera nodes, we are required to design a delay-tolerant system. To deal with these issues, we built all communication protocols in an opportunistic way. The video and

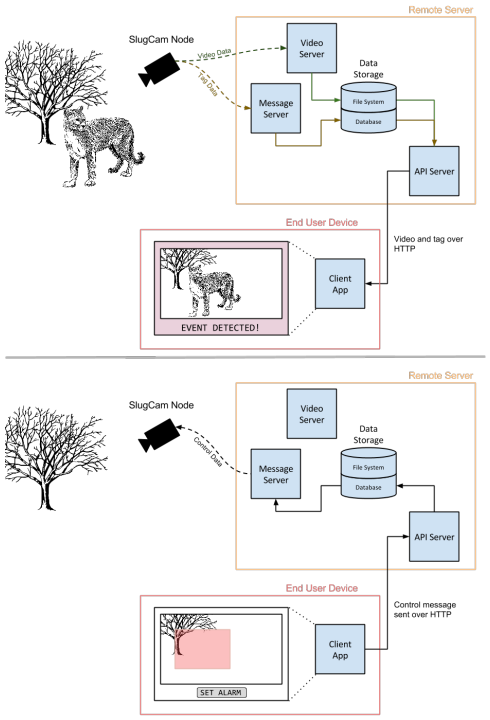

Fig. 3. Overview of the SlugCam system recording an event and SlugCam sending a message to the camera, example is setting an alarm region.

message servers constantly wait for incoming connections from camera nodes and expect that those connections could be dropped at any time. Any messages that need to be sent to the camera must wait until that camera comes online and as soon as the camera does come online we begin to send all outgoing messages. This style of network further increases camera autonomy at the cost immediate communication ability. It allows for many interesting optimizations, such as reducing communication in low power situations and when only non-interesting data has been collected.

B. SlugCam Node Software

The software on the camera node is required to fulfill three main responsibilities: controlling the camera, controlling wireless communication, and controlling the camera node itself. Each of these functions is implemented as a service on our system. This allows us to do things like develop the long-running processes alongside shorter running processes that can be scripted more easily for testing purposes without duplicating code. The ability to write scripts to perform our various duty cycles has been invaluable in testing of our system and allows for the rapid implementation of new ideas during experimentation.

We decided that all of our communication handling should be handled in a single process due to the nature of the WiFly module that we are using for wireless communication. Since all communication must happen over a serial line and only one TCP connection can be open at a time, having one communication service allows us to better manage our communication needs.

Since the node is also responsible for intelligently managing duty cycles, our system also requires a control service. This service will be responsible for looking at battery level, solar power level, commands from the user, and other variables then determining an appropriate course of action. It will be where we can facilitate many of the interesting features of our system such as immediately sending video with action in preset areas, allowing video recording on a timer, and taking still shots instead of video when battery levels are low.

C. Remote Server Software

We chose to break apart the software that runs on the remote server into several main areas as shown in figure 3. The message server is the main server responsible for communicating all textual data between the network and the cameras, the video server receives all video data from the cameras, and the API server offers an HTTP interface to the network for end-user applications. All of these server modules interact independently and communicate through shared data storage. This means that all of these modules can be run on a single physical server, or split across multiple servers. This allows our system to scale well if necessary while retaining the ability to deploy on a single machine.

In our implementation of the server software we decided to use the Node.js JavaScript framework. We went with this choice for several reasons. Facilities for creating TCP and HTTP servers, network security, and many other essentials exist in the standard Node.js libraries and the code written using these libraries remains succinct, and we believe easier to understand than in many other frameworks. The developer community around Node.js is active and we were able to find packages for many useful development and network related tasks.

1) Data Storage: The SlugCam network's data storage needs can be split into two major sections, textual data and video data. In order to consolidate storage code in a meaningful way, the data storage code exists as a Node.js module and contains all the functions required to access the data layer.

Textual data in our system takes the form of what we call messages. Messages encapsulate data communicated both to and from the cameras. Each message has several required metadata fields, a type, and a data section. All of the native SlugCam messages have a type and each type has a specified structure for the data section. The data layer is responsible for allowing us to query a specific message type and it is up to the client to decide how to interpret the data in a message. It allows new data types to be introduced into the system without modifying the software running on the remote server, a developer simply has to find a unique type name and decide how the data field of this new type will be structured.

In our current implementation of the system we chose MongoDB[18] for textual data storage. MongoDB offers several advantages in our situation. First of all, MongoDB stores

records as documents, which are very similar to JSON objects. This closely matches our desire to store our messages in a flexible way and the use of JSON as a communication format. Since MongoDB is designed around this style of data storage, and offers tools to easily and efficiently query data stored in this manner, it is a choice that has allowed us good performance and rapid development time.

The second portion of our storage layer is video storage. Video data and the textual data are very different. We often want to search through and query against large amounts of small messages whereas video data contains a smaller number of large data records that are not usually queried by content. In looking at methods we could use to store video data we determined that simply storing video data on the file system of the operating system would suit this project better than the alternatives, such as using a database system.

Since all metadata pertaining to a videos is textual data, it is stored in the database. This means that there is a set of data in the textual data storage system containing all the metadata for a video. This set will contain a video ID, which along with the camera name can be used to recall the required video data from the file system. This means that we are able to retain the advantages of a database for querying and searching through video data.

One of the most important textual data messages are what we refer to as tags. These tags are specific events that occurred while recording the data and are uniquely identified using the on-board computer vision software. This allows users to query video files with detailed events for their application. In section 5.C. we show hows these tags are presented to the user in a meaningful way.

2) Message Server: The message server is a TCP server that is responsible for textual communication with the cameras. This server operates in an opportunistic manner, as the rest of the network software does. This means that the server keeps a TCP port open ready to receive data and on the first message it receives it will begin sending outgoing messages until there are no more messages or the connection has been lost.

One of these outgoing messages could be a request for a live stream of the area being monitored. Currently we have implemented a feature to allow the user to set what we call alarms in a viewed space. This too is showcased and detailed in section 5.C.

3) Video Server: The video server is a TCP server that is responsible for receiving video data from the cameras. Unlike the message server, the video server only allows for communication from cameras, thus it is simpler in design than the message server.

Currently after a connection is made the camera first sends its name as a null terminated string, the camera then sends the video ID and video byte length as unsigned 32bit integers, then sends the video data itself. The camera can then repeatedly send IDs, byte lengths, and video data for as many videos as it likes. This is shown in table I.

4) API Server: The API server provides a clearly defined interface into this potentially complicated network. The API server is one of the points at which we were able to create a more transparent design for future developers. Our API server provides an HTTP interface to the network software that can be used by multiple client types. We chose to use an HTTP interface due to its ubiquity in modern computer systems. Future developers of the system have the ability to develop clients in many different languages and frameworks.

TABLE I
FIELDS REQUIRED FOR THE VIDEO PROTOCOL

Length	Variable	32 bits	32 bits	$Length_1$	32 bits	...
Field	NodeName	ID1	$Length_1$	$Data_1$	ID_2	...

D. End-User Device Software

We decided to build our client application as a single-page web application(SPA). This application runs within a web browser, but only requires a connection to the API server. AngularJS[19] is our main framework used in the development of this application. AngularJS is a JavaScript framework developed by Google. It aids in the creation of single-page client based web applications and allows us to build user interfaces in a more declarative way than in traditional web programming. This declarative approach to user interface design makes it much easier for us to experiment with different interface ideas rapidly. AngularJS is well documented and supported by Google, and has an active community of users outside of Google. Using the AngularJS framework has helped us create efficient, modular code that is easy to maintain.

V. DEPLOYING AND EVALUATING SLUGCAM

In this section, we describe our experience deploying SlugCam in an outdoor scenario as well as experiments we conducted to characterize SlugCams power consumption. SlugCams on-board passive current sensor proved to be invaluable to obtain accurate and adequate power consumption information.

A. Energy Efficiency

One well-known and widely-used sensor network power savings techniques is to keep the system in a low-power state whenever it is idle, i.e., not executing any specific task. In SlugCam, we go a step further and use a low-power microcontroller (the MSP430) to control: (1) a PIR motion sensor to wake-up the Pi when it detects motion and (2) a relay circuit to cut power to the Pi based on an onboard MSP430 timer, which can be dynamically set by the Pi. In order to show the power savings achieved through our design, we run SlugCam for 5 minutes during which the system wakes up 6 times. When it wakes up (triggered by motion detected by the PIR), the Pi turns on the camera for 30 seconds before going back to sleep. As baseline, we run the system uninterruptedly for 5 minutes. Figure 4 shows the current consumed in the two runs. We use a 10-point moving average filter to smooth out the sampled current consumption data for readability without sacrificing accuracy. Calculating the area under both curves,

we observe that the system saves around 30% when monitoring a relatively busy scene. The dips in the curve show that in the low power state, the system is consuming between 60 and 70 mA, while in the full on state, it consumes around 400mA. We note that the power savings can be even greater if we take advantage of the MSP430 deep sleep mode when it only requires current in the order of microAmps. We also performed a comparison between the recent Raspberry Pi B+, which SlugCam is using, and its precursor, the Raspberry Pi B. We found that latter consumes on average 150mA with no connected components and the new B+ model consumed around 95mA.

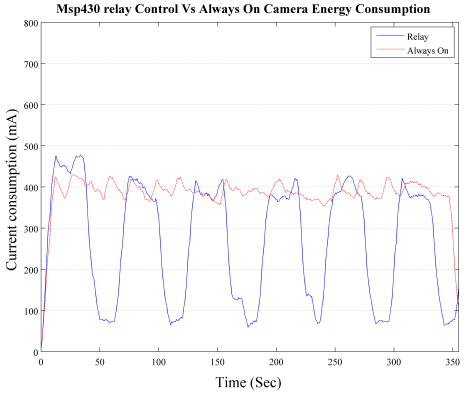

Fig. 4. SlugCam power consumption showing power savings by using the MSP430 to control the PIR motion sensor and power relay.

B. Adaptive Duty Cycling

Duty cycling has been adopted by wireless sensor networks in general, and visual sensor networks in particular, as an effective energy savings technique [8] [9]. It typically works by having sensor network nodes switch to low-power states as much as possible in order to save overall energy consumed. SlugCam uses what we call adaptive duty cycling which allows the system to use current battery level information to guide its choice of duty cycles. In particular, when battery levels are too low (i.e., lower than a given threshold), the system avoids duty cycles that power hungry. Clearly, this typically comes at the price of degrading performance. This trade-off between performance and energy efficiency can be adjusted to meet application requirements by controlling the battery level threshold parameter.

We show SlugCams power consumption characteristics for the six different duty cycles described below:

1) Duty Cycle A: This duty cycle illustrates the case of the remaining battery level being below the threshold. In this particular experiment, the system wakes up and runs system services in which it determines the battery remaining is below the threshold.

2) Duty Cycle B: In duty cycle B, the Pi is turned on by the PIR sensor and triggers the camera. The camera is turned on but does not run any computer vision functions. The WiFi module is also asleep as this duty cycle only records video and stores it locally. An alternate, lower power duty cycle to this one could take a high resolution image, instead of recording a video.

3) Duty Cycle C: Duty cycle C is the same as duty cycle B except that the system uses computer vision algorithms when recording data. In this particular instance, noticing the data is not important (based on the computer vision results), the system quickly transitions to low power state and the MSP430 cuts power to the Pi via the relay. As shown in the bottom left graph of Figure 5, at low-power state, the system (essentially the MSP430 in idle mode) consumes around 70mA. However, as previously pointed out, power consumption can be substantially decreased by having the MSP430 go to sleep.

4) Duty Cycle D: Duty Cycle D is very similar to C with the only difference being that in C nothing is recorded while in D the video is stored locally. This explains the difference, albeit small, in power consumption between the C current consumption curve (in the bottom left graph) and Ds (in the bottom left and right graphs of Figure 5, respectively).

5) Duty Cycle E: The most complex and longest running duty cycle is E, during which SlugCam records video (longer than D) with computer vision turned on and then immediately transmits the data. In Figure 5, for the E current consumption curve in the bottom right graph, the first drop in current consumption happens when the camera turns off and the WiFi module is on in order to transmit the video file to our Web server. The second in current consumption drop corresponds to the system going back to its sleep state with the MSP430 in idle state.

6) Duty Cycle F: In duty Cycle F, the system wakes up on its own to ping the Web server for any requests or to report status updates. This time is also used to transmit any unsent video data. Notice though that the camera is never turned on and SlugCam consumes around 350mA during data transmission. The drop at the end shows the WiFi module and the Pi being put to sleep and the Raspberry Pi turning to idle.

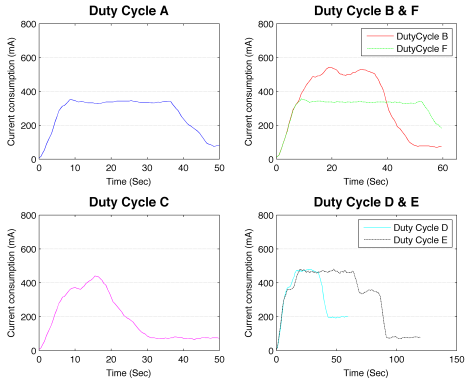

Fig. 5. Power consumption characterization for SlugCam duty cycles.

Another interesting feature to notice from the current consumption curves in Figure 5 is there is very little difference in power consumption when the camera is on and computer vision software is executed. This is the case without using

the Pis GPU which we plan to use in the future. Another interesting observation when looking at the upper right graph comparing duty cycles B and F is that the camera consumes more power than the WiFi module. Comparing the curves in the bottom right graph for duty cycles D and E, we observe the difference in duty cycle duration; more specifically, duty cycle Es active period, i.e., when the system is not in idle state, is almost twice as long as duty cycle D. In this particular comparison, it shows the difference between storing the data locally (duty cycle D) and transmitting to a remote server (duty cycle E).

In the next set of experiments, we have SlugCam cycle through different duty cycles as it would in normal operation. More specifically, we run the system for 5 minutes during which 3 distinct events occur. The first event is a false positive, e.g., light or heat radiation trigger the PIR, which in turn wakes up the camera and video is recorded and transmitted. The second event is data that should be recorded for reference later but does not get transmitted, because it is deemed not urgent, or the battery level is too low for transmission. The third and final event records video and transmits it to the remote server as it is considered by the user to be an urgent event. The users ability to set these alarms is described in detail in Section 5.C.

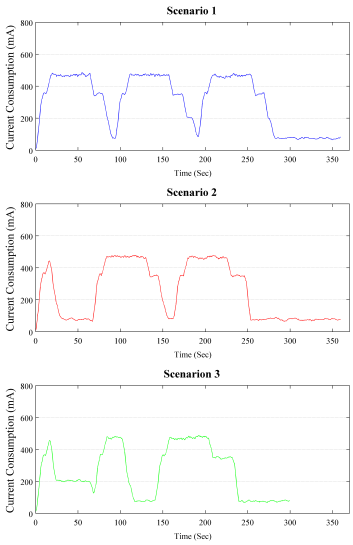

Fig. 6. Current consumption data for the three application scenarios.

In Figure 6 we show the events played over 5 minutes, while SlugCam uses different duty cycles in all three scenarios. The first scenario exemplifies a system with no adaptive abilities as baseline for comparison. In this scenario, all three events are handled by duty cycle E, as previously described, where data is recorded and transmitted for all events. The second scenario showcases the power saving benefits of adding computer vision to capture and analyze video footage. SlugCam is able to correctly identify the first false positive event and immediately go back to low power state. This operation is represented by duty cycle C. In this scenario however, SlugCam then decides that the next two events should not only be recorded but immediately transmitted as well, which is handled by duty cycle E. Scenario 3 showcases the system ability to ignore the first event (duty cycle C), record and not transmit the second event (duty cycle D), and record and transmit the third event (duty cycle E).

C. Outdoor Deployment

We deployed SlugCam in the UCSC campus by setting up a node in an outdoor area and running both the client and server applications on a laptop. We initially cleared all persistent data on the server and ran a duty cycle in which the camera turns on, begins recording and tagging the video, and then sends the recorded video to the server. Without configuring the server, we positioned the SlugCam node and ran through the duty cycle several times. We then started the client application to look at the data collected by the system. When we first start the client application from a Web browser, we are presented with the main screen and notified that we need to set a location for the node in order for it to show up on the map.

Then, following the link to the camera configuration page, the node location can be set automatically using the browsers location data. The map allows the user to better understand the area being monitored by the deployed cameras. Also since this system is designed to work with a large number of cameras, having a list of cameras is not as intuitive as being able to see the location of cameras and recognize where activity is happening on a more global scale. The ability to implement mapping is an interesting application of our transparent network feature. The mapping data is implemented only at the client level, not requiring any input from the cameras or change to the networking software. Upon return to the dashboard, we were able to open the video on what we call the activity bar for analysis. The activity bar displays the activity over a period of time. This is an important aspect of our visual presentation as it allows the user to correlate collected tags (previously discussed in section 4) to the videos in which activity is happening in an intuitive way, but without hiding any information.This is also demonstrated in figure 7 above the video preview. We saw all of the sent tags and videos on the activity bar and we were able to view the videos in the video player. The videos also show the object recognition algorithm results represented as black squares around detected objects.

Another important feature mentioned earlier in section 4 is the ability to set certain areas within view of a camera that will trigger alarms. When an alarm is triggered it could mean that a user is notified and the video recorded is instantly sent to the server. The interface we have developed (shown in figure 8 is intuitive and allows the user to visually select an area over the monitored space, thus allowing an accurate selection given the current position of the camera.

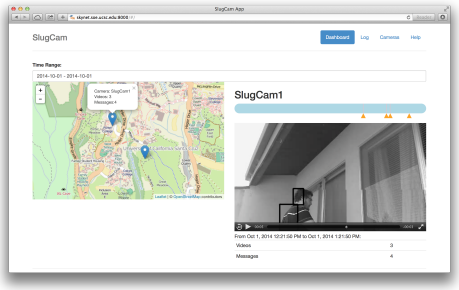

Fig. 7. Screenshot of the client application.

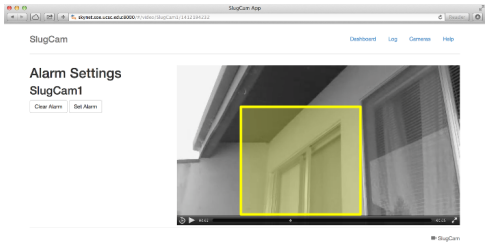

Fig. 8. Proposed interface for selection of alarm areas.

VI. Background and Related Work

Smart camera systems have received considerable attention from academia as well as industry. Consequently, as reported in our recent survey of the state-of-the-art in smart camera systems [9], a significant body of work has resulted. Rather than an exhaustive survey of the state-of-the-art (which can be found in [9]), this section describes smart camera systems that use solar power.The FireWxNet system [20] is a solar-powered wireless sensor network that, instead of computer vision capabilities, use auxiliary sensor nodes to provide information to allow the user to decide if they wish to receive a live video stream of the monitored area. In contrast to SlugCam's use of 802.11 WiFi to transmit high resolution video data, FireWxNet uses 900Mhz radios to achieve long range communication. Doing this however sacrifices the ability to transfer larger amounts of information as 900Mhz radios dont achieve the same bandwidth availability as WiFi. SlugCam does not have a range issue with WiFi directional antennas and its ability to be deployed as an ad-hoc network.The system described in [13] does not require high resolution video data and makes use of bluetooth and Zigbee radios. It throttles its image sensor based on the relevance of the data recorded and therefore uses less power and processing resources. The system also uses computer vision algorithms and like SlugCam performs background subtraction. However, the system software is proprietary. The authors evaluate their solar capabilities separately and do not describe actual deployment scenarios. They do however provide an accurate estimate of how long their system can run uninterruptedly on battery power.

VII. Conclusions

In this paper we introduced SlugCam, an open-source smart wireless visual sensor network system built with low-power, off-the-shelf components. SlugCam offers a complete system solution from the hardware to its Web-based user interface. It targets modularity and extensibility so that the system can adapt to the needs of existing applications as well as evolve to fulfill the requirements of emerging ones. We demonstrate SlugCams energy efficiency through a thorough power consumption characterization of the system and describe our experience deploying it in an outdoor scenario.

References

[1] Weiming Hu, Tieniu Tan, Liang Wang, and S. Maybank. A survey on visual surveillance of object motion and behaviors. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 34(3):334–352, Aug 2004.

[2] Phoebus Chen, Kirak Hong, Nikhil Naikal, S Shankar Sastry, Doug Tygar, Posu Yan, Allen Y Yang, Lung-Chung Chang, Leon Lin, Simon Wang, et al. A low-bandwidth camera sensor platform with applications in smart camera networks. *ACM Transactions on Sensor Networks (TOSN)*, 9(2):21, 2013.

[3] Umakishore Ramachandran, K. Hong, L. Iftode, R. Jain, R. Kumar, K. Rothermel, Junsuk Shin, and R. Sivakumar. Large-scale situation awareness with camera networks and multimodal sensing. *Proceedings of the IEEE*, 100(4):878–892, April 2012.

[4] Sujesha Sudevalayam and Purushottam Kulkarni. Energy harvesting sensor nodes: Survey and implications. *Communications Surveys & Tutorials, IEEE*, 13(3):443–461, 2011.

[5] Alvaro Pinto, Zhe Zhang, Xin Dong, Senem Velipasalar, Mehmet Can Vuran, and Mustafa Cenk Gursoy. Analysis of the accuracy-latency-energy tradeoff for wireless embedded camera networks. In *Wireless Communications and Networking Conference (WCNC), 2011 IEEE*, pages 2101–2106. IEEE, 2011.

[6] Msp430. http://www.ti.com/msp430launchpad.

[7] Raspberry Pi Foundation. Raspberry pi. http://www.raspberrypi.org/introducing-raspberry-pi-model-b-plus/, 2014.

[8] Adolph Seema and Martin Reisslein. Towards efficient wireless video sensor networks: A survey of existing node architectures and proposal for a flexi-wvsnp design. *Communications Surveys & Tutorials, IEEE*, 13(3):462–486, 2011.

[9] K. Abas, C. Porto, and K. Obraczka. Wireless smart camera networks for the surveillance of public spaces. *Computer*, 47(5):37–44, May 2014.

[10] Buildroot. http://buildroot.uclibc.org/.

[11] Itseez. Opencv. http://www.opencv.org.

[12] Raspberry pi camera module. http://elinux.org/Rpi_Camera_Module, 2013.

[13] M. Magno, D. Brunelli, P. Zappi, and L. Benini. A solar-powered video sensor node for energy efficient multimodal surveillance. In *Digital System Design Architectures, Methods and Tools, 2008. DSD '08. 11th EUROMICRO Conference on*, pages 512–519, Sept 2008.

[14] Rn-171 wifi module. http://www.microchip.com/rn171, 2014.

[15] National renewable energy laboratory. http://www.nrel.gov/rredc/.

[16] Mat Dirjish. Whats the difference between thin-film and crystalline-silicon solar panels @ONLINE, May 2012.

[17] Li-ion/li-polymer battery charge management controller. http://www.microchip.com/mcp73871.

[18] MongoDB, 2013. [Online; accessed 15-September-2014].

[19] AngularJS – Superheroic JavaScript MVW Framework. [Online; accessed 15-September-2014].

[20] Carl Hartung, Richard Han, Carl Seielstad, and Saxon Holbrook. Firewxnet: A multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services*, MobiSys '06, pages 28–41, New York, NY, USA, 2006. ACM.