Solar-Powered Smart Wireless Camera Network for Ubiquitous Outdoor Monitoring

Kevin Abas*, Leland Miller *, Katia Obraczka*
*School of Engineering, University of California Santa Cruz, CA 95064 USA
†University of So Paulo, Av. Prof. Almeida Prado, So Paulo, SP 05508-070 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 such that it can 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 on board processing capability allows computer vision software to run locally which contributes to the systems autonomous operation capabilities. Energy efficiency in SlugCam is accomplished both in hardware by using low-power components and software. The systems duty cycles automatically adapt to the current state of the battery in order to balance the trade-off between application-level requirements and power awareness. For instance, SlugCams smart camera changes its monitoring behavior based on how much battery charge remains. Additionally, using its computer vision software, the system only records and transmits information upon event detection which contributes both to the systems energy efficiency as well as its low network bandwidth requirements. Another important contribution of SlugCam is to provide an open-source network software platform that can adapt to future requirements of a variety of visual sensor networks and their applications. For example, SlugCams user interface software maps and tags events to the corresponding video footage captured by camera nodes in order to present collections of fragmented information in a cohesive manner. In addition to a detailed description of SlugCam, this paper presents an extensive power characterization of the systems operation and showcases its deployment in a real-world scenario.**

I. Introduction and Motivation

Wireless visual sensor networks have advanced tremendously over the years and continues to be an exciting area for researchers. Visual sensors provide rich data of a surveyed environment and with new computer vision techniques these visual sensor networks are analyzing more data than ever before. Smart cameras, as they are named, can now not only identify objects in a surveyed space, but determine the objects specific behaviors [1], [2], [3]. Smart wireless visual sensor networks (WVSNs), the nodes referred to as smart cameras, are those which take advantage of these computer vision enhancements for many pervasive applications such as environmental monitoring, monitoring of the elderly, and wildlife monitoring [4].

Solar energy is something that breaks down boundaries for wireless sensor networks and allows researchers to rethink their limitations for remote wireless video sensor network deployment. [5]. Battery operated systems require unnecessary maintenance and frequent battery replacements. Solar systems take advantage of a ubiquitous renewable energy resource, but in turn bring up other new challenges [6]. Systems must adapt to these challenges, such as included low sunlight weather conditions, with innovative and extensible techniques.

The SlugCam project aims to provide a platform upon which any interested future researchers can take advantage of the advances in hardware technology and computer vision in a way that is tailored to their requirements. In order to reach this goal we are constructing open hardware plans and an open source software platform to support a video monitoring network at every level of system design, from the camera nodes to the user interface. Systems in the past have discussed the need for modularity with the rapid advancements of a platforms subcomponents citeFlexi-WVSNP, we hope to address these needs by providing a modular system in which individual components of the network can be replaced without modification to the entire system. Along with a focus on modularity we also focus on using standard and open technologies and building our system to be as developer and user friendly as possible. SlugCams multimodal capabilities will also attract researchers interested in advancing situational awareness applications and will find our system very flexible and cost effective [4].

The rest of the paper is organized as such. In section 2 we provide an overview of the philosophies SlugCam contributes to the research community. In section 3 and 4 we provide summarizations of both SlugCams hardware and software architectures and describe the sub components weve included in the system. In section 5 we include a thorough analysis of the systems energy efficiency and show the usability of the systems management software. Next, we describe similar smart camera platforms and distinguish SlugCam and describe its uniqueness. Finally we conclude the paper with a discussion of future work and a summary of the paper.

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 onboard 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 [7]. 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 its 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 [8] manages the amount of time that the more power consuming Raspberry Pi stays on. This is accomplished by having the MSP430 control the PIR motion 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. Upon motion detection, the MSP430 (triggered by the PIR) will wake up the Pi, who, in turn, will wake up the camera. As will be shown by our power characterization experiments in Section 5, this results in substantial power savings which allows SlugCam not only to use smaller size solar panel and battery, but 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 [9] which runs the nodes main functions including its visual processing task. Due to their high energy cost, 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 goal 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 [10] [11], 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 [10]. 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.

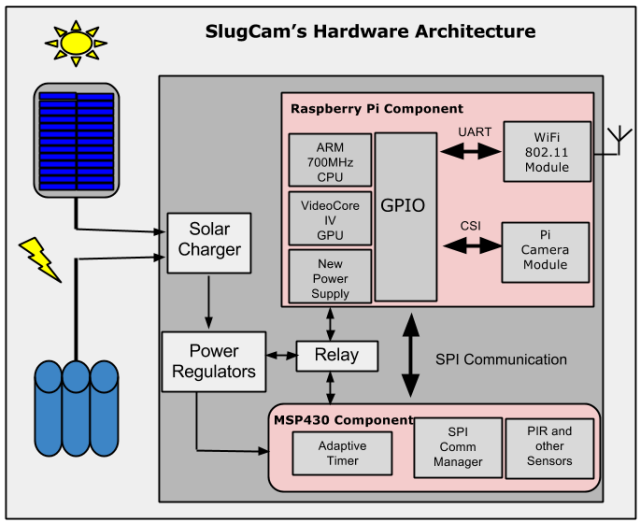

Fig. 1. SlugCam's Hardware Architecture.

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 [**?**]. We have installed our own custom embedded Linux distribution to optimize performance and processing speeds using the Buildroot tool [12]. 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[13], 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 need (and none that we do 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 [14]. 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 .05μA 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. The Raspberry Pi can dynamically change how much time it stays on by communicating to the MSP430 over a Serial Peripheral Interface (SPI) connection (as shown in Figure 1). Both the small msp430 chip and the relay circuit can be viewed in the prototype screenshot from Figure citefig:hardwareScreen. Similar to smart camera systems such as the one described in [15], 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 [**?**] which can be put into a low-power sleep state when not in use [16]. 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 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.

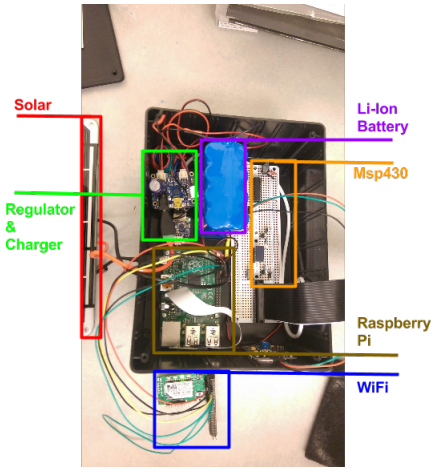

Fig. 2. The SlugCam prototype node.

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 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 [17]. 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.

$$EnergyOutput = SolarEnergy * Panelarea * PVefficiency$$

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) [18]. 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.

$$BatteryCapacity = Consumption(Amps) * DarkUsage(hrs)$$

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 power-trains 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 sub components weve chosen and our adaptive power aware system, SlugCam Will last

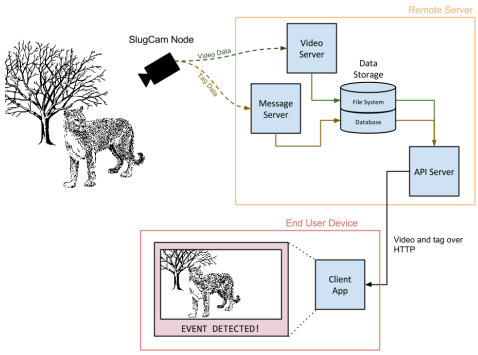

Fig. 3. Overview of the SlugCam system recording an event.

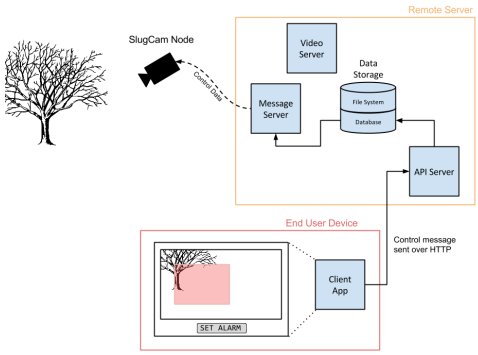

Fig. 4. Overview of the SlugCam system sending a message to the camera, example is setting an alarm region.

longer than this for certain less intensive applications. In order to charge the battery adequately and safely, we chose to use Microchips MCP73871 charger [19] 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. By clearly defining functional units and retaining a separation of duties between separate units of code we help ease the development process. Not only can development on separate modules occur in parallel, but developers can also work on a module without needing knowledge of how other modules are coded. 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 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. Cameras can also be set to come alive on a timer to receive important control data from the network if needed to help manage the communication latency inherent with this type of design.

One of our goals is to make it easy for researchers to augment the camera nodes with additional sensors with minimal modification to our software. We would like these developers to not have to worry about the network at all, and to have the opportunity to focus on their data collection and analysis systems. To implement a new mode of data collection, a developer only needs to establish a connection to the message server and send data. The network will recognize and handle new data types as it does native SlugCam data types. Eventually our video data collection system could be abstracted to generic binary data collection which would allow further opportunities to future developers on our platform without modifying the network itself.

B. SlugCam Node Software

One of the main benefits of running an embedded Linux distributions on our camera nodes any developer who has a moderate understanding of programming in C and using an environment similar to Linux can begin developing custom services for the camera nodes. It also means that we are able to develop our node software more efficiently and securely by using the well tested code bases of several open source projects for much of the lower level hardware interfacing. We also gain compatibility with many existing software libraries and hardware modules.

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. We wrote much of the code to perform these functions as C libraries so that we are able to reuse it in multiple programs. This allows us to do things like develop the long-running processes along side 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.

The service controlling the camera exists of camera interfacing code along with the computer vision algorithms. We use the comprehensive open source computer vision library OpenCV for assist in the construction of algorithms to analyze video. These computer vision algorithms are run against the live video stream from the Raspberry Pi camera module. We use the official library that is provided with the camera module for setting up the camera and filtering the stream.

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. At this time this service is simply responsible for starting the other two services, however as more of the duty cycles are implemented, selecting the duty cycle will become more complex. This service will be responsible for looking at battery level, solar power level, commands from the user, and possibly many other variables and determining an appropriate course of action. It will be where we can facilitate many of the interesting features 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 and the video server receives the video data from the cameras. The API server offers an HTTP interface to the network for end-user applications. All of these 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 that only require access to the video file system and the database. This allows us the ability to scale in the future if necessary while retaining the ability to deploy on a single machine if desired.

In our implementation of the server software we decided to use the Node.js JavaScript framework. We went with this choice for several reasons. Node.js offers many built in tools for writing network software while remaining more lightweight than many alternatives. Facilities for creating TCP and HTTP servers, network security, and many other essentials exist in the standard Node.js libraries. Using these facilities does not require much boilerplate code and is more succinct, and we believe more approachable, than in some alternatives. The developer community around Node.js is active and we were able to find packages for many useful development and network related tasks. In our experience many of the packages we found focused on solving a specific task which allowed us to bring in well-tested packages without adding a large amount of unnecessary code to the project.

1) Data Storage: The SlugCam network's data storage needs can be split into two major sections, textual data and video data. Both of these types of data storage have unique challenges and needs for a storage solution. 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. It is the design of this message format that allows us to be so flexible in our stored data types. All of the native SlugCam messages have a type and 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. This goes along with the transparent network concept. It allows new data type to be introduced into the system without modifying the software running on the remote server.

In our current implementation of the system we chose MongoDB[20] 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. It makes it very easy to implement the flexibility we desire in our message format.

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 with video data each record contains a relatively large amount of data and requires special work to process and search though. 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.

For our project, one of the advantages of storing the video data using the file system is that we retain the ability to use third party video processing tools without requiring a database wrapper to access the video. This helps us in experimenting with different video formats and means that we can easily write scripts to process an entire library of collected video data.

2) Message Server: The message server is a TCP server that is responsible for textual communication with the cameras. It transmits messages to and from camera nodes. 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.

In an attempt to find a standard data format that would be easy to use for future developers we wanted to use a standard data format instead of creating our own. We found that using JSON to structure our messages met most of our requirements. JSON is a standard, well defined, textual data format that is easy to parse and human readable. Libraries for working with JSON exist in many languages and it is not difficult to write libraries in languages that do not already have support. JSON also allows for nested structures which opens up many possibilities in structuring our message data fields.

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.

The protocol used to receive messages is a very simple binary transfer protocol. After a TCP connection is made the camera begins sending any pending videos. Multiple videos can be transferred per connection and like the message server a connection may be ended at any time without causing problems.

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.

It should also be noted that the video ID is a Unix timestamp and the duration of the video is calculated on the receiving server in order to find the video start and end times. This reduces the amount of record keeping necessary on the node, but more importantly it provides us with a simple way to track start and end times for videos even if the node is unexpectedly powered off during video recording.

TABLE I
FIELDS REQUIRED FOR THE VIDEO PROTOCOL

Length	Variable	32 bits	32 bits	$Length_1$	32 bits	...
Field	Node Name	ID1	$Length_1$	$Data_1$	ID_2	...

We have designed several other protocols that work differently than our current protocol. We have looked at adding the ability to break the data into pieces and to send the pieces individually. The protocol we had originally designed worked this way however, due to time constraints, we decided to implement our current protocol first and to experiment with various algorithms to send fragmented videos later. This way we could have a performance analysis of the different algorithms and see how they worked in the system as a whole. One of the main advantages of our current protocol is its simplicity. We will need to balance the value of being able to power off mid-transfer without data loss and the code complexity that a new protocol could add. With our current setup we have the ability for the camera to make sure that a shutdown does not occur mid-way through a video transfer and to prevent video transmission in a low battery situation, so it is possible that other protocols may not offer enough benefit to outweigh the extra complications to the system that they would create.

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. It also allows for the possibility of running custom automated monitoring and video processing applications without modifications to the core network.

D. End-User Device Software

In our system, end-user devices are responsible for presenting collected information to the user. We have designed a client web application that is designed to be a user-friendly interface to the large amounts of fragmented data that this system has the ability to collect. We wanted the client application to act as a showcase for the power of the system and to show many of its unique features. It is also important that the client application is well written and documented so that it can serve as a usage example for the API server.

We decided to build our client application as a single-page web application. This application runs within a web browser, but only requires a connection to the API server. This means that, though the application is built using web technologies, the application itself is fully contained in the client's web browser.

One of the main reasons we chose this approach is its flexibility. By designing our application using standard web

technologies we try to remain as platform agnostic as possible. We hope to build an application that can work well across operating systems and browsers.

Since our application is a web application it can be hosted by a server included with the network software. This approach means that no end-user configuration is needed to run the application and that the application can be accessed from any web browser with access to the server. This could be a very clean and easy design for many uses. This is the method we are currently using in our testing, and we have included hooks in the script that starts the other servers that also starts up a static server hosting or client application.

Since the our application runs entirely in the web browser, there is also the possibility of running the application from a client's local disk. Using frameworks such as node-webkit[21] or Atom Shell[22] we could also include features that are not possible in a normal web application such as the ability to save video files locally without prompting the user for location (could be useful if bulk saving many video files), or deeper desktop integration than web browsers allow for.

Another benefit of using web technologies is mobile support. By sticking close to web standards we have been able to create a client application that is usable on tablets and mobile phones. In fact, we have been able to use the HTML5 geolocation API to allow for setting the location of a camera based on GPS data from a smart phone. This means that when setting up a camera, the user can log into the web application, open the camera configuration, hit a button, and the camera location is automatically adjusted (manual configuration is also available).

We decided to use AngularJS[23] as our main framework 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. Evaluating SlugCams Usability and Performance

In order to prove the systems has enough energy efficiency to deploy the system in real world scenarios we needed to do some accurate power analysis of the system. Using a simple 10 point moving average filter, we were able to smooth out our current consumption data for readability without sacrificing accuracy. We provide a proper current consumption analysis of the system both evaluating the duty cycles we defined and analyzing how the system achieves efficiency through some example application scenarios. We finish SlugCams evaluation by deploying the system on UCSCs campus and simulate some scenarios to demonstrate how SlugCam would be used for monitoring applications. Through these detailed demonstration we also showcase the features we developed in our client video management app.

A. Design Decision evaluation

The most traditional smart camera feature is the ability to put the system in a low powered sleep state when idling for no specific task. SlugCam uses a relay to accomplish this feature and we evaluated the systems current consumption savings as is shown in figure 5. In this simple experiment we run SlugCam for 5 minutes both with the ability to wake with the PIR sensor and immediately sleep with the use of a relay to cut power. For simplicity the system wakes and turns on the camera for approximately 30 secs before returning to a sleep state. Calculating the area under both curves we see that the system saves 30%, and this was with 6 repetitive wake ups imitating a very busy scene to monitor. The troughs of the curve show a sleep state of 60 to 70 mA representing the msp430 idling, but in an actual deployment SlugCam would only be powering the PIR sensor while having the msp430 in a low powered sleep state. With the msp430 only requiring micro amps in its sleep state and the PIR only consuming 3mA the system can sleep for long periods without any activity to detect. We also did our own current consumption comparison of the new modified raspberry pi B+ versus the old Raspberry pi model B. We concluded using our current sensor that the old Raspberry Pi consumed on average 150mA with no connected components and the new B+ model consumed around 95mA. These efficiencies are due to new more efficieny voltage regulators the the new Raspberry Pi B+ uses. This was perfect for SlugCam battery powered design and have proven a good match with the msp430.

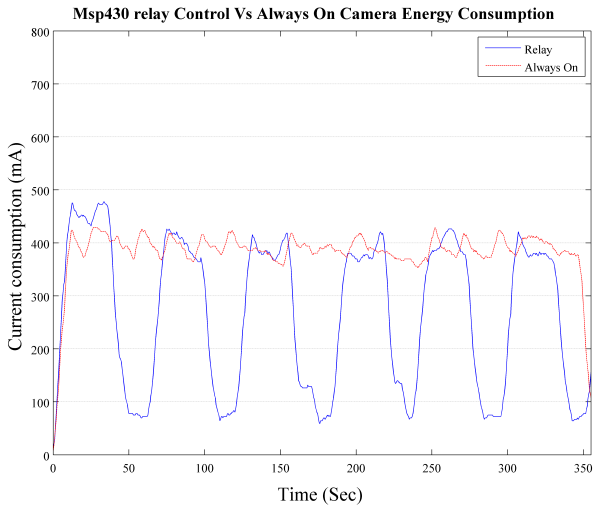

Fig. 5. Power consumption data for SlugCam using the relay and the SlugCam acting as a traditional always on camera.

B. Power characterization

In order to for a system to be adaptive to available battery supply, smart cameras and other battery powered sensor

networks have had to create adaptive duty cycles [10] [11]. Using these duty cycles the system can change how much power is necessary to continue operation. We defined 6 duty cycles for experimentation which can be seen in figure 6, they are described as the following:

1) Duty Cycle A: The first duty cycle was to represent the system needing to move to the low powered sleep state we defined earlier. The system wakes up and runs system services in which it determines the battery remaining will not allow the system to function with the remaining battery supply. We expect the system to idle in much less time but you can see the system only reaches an idling mode and then turns off.

2) Duty Cycle B: The second duty cycle B,along with duty cycle F, are in the next level up in regards to available battery charge. In this duty cycle the camera is turned on with no computer vision. The wifi module is also put to sleep and this duty cycle would only record video locally after being woken by the PIR sensor. Although not defined as a duty cycle, the system could also take a quick high resolution image, reducing the amount of time needing to be on.

3) Duty Cycle C: In duty cycle C the system now uses the computer vision algorithms when recording data. Noticing the data isn't important the system quickly transitions to a sleep state where the msp430 cuts power to the pi with the relay. You can see the rest of the time is spent with the msp430 idling around 70mA, but this will be much lower in an actually deployment as the msp430 will too sleep.

4) Duty Cycle D: Duty Cycle D is more complex in that now it makes use of the computer vision algorithms. A shorter video is recorded than in duty cycle E and in contrast it goes to sleep after the recording finishes.

5) Duty Cycle E: The most complex and longest running duty cycle is E, in which it records video data with computer vision turned on and then immediately transmits the data right after. The first drop signifies the camera turning off and the wifi module sending the video file to our web server. The second drop is the system going back to its sleep state with the msp430 left idling.

6) Duty Cycle F: Duty Cycle F shows the system waking 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 transmission of data. The drop at the end shows the wifi module being put to sleep while leaving the raspberry Pi and msp430 to idle.

After issuing the experiments we noticed little to no difference in the system using the camera with and without computer vision. This is however without the use of the GPU which we plan on using to increase SlugCam's vision processing capabilities. It's also important to notice the change in time these duty cycles take to execute, showing SlugCam's Dynamic adaptive behavior. B and F were put side by side two show the difference in using the camera without computer vision or the wifi module alone. Then D and E were compared to show the difference in needing to transmit or recording and storing the data locally.

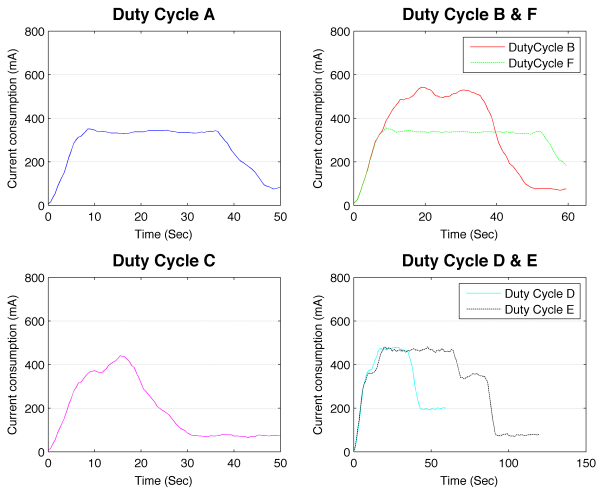

Fig. 6. Current Consumption data for all 6 duty cycles.

Taking this type of analysis further we chose to run the system cycling through different duty cycles as it would normally while showcasing SlugCam's energy efficiency features. The System would operate normally for up to 5 minutes with 3 distinct events occurring in the alloted time. The first event is a false positive example where a normal smart camera system may stay on and try to record data after motion is detected. A false positive event can occur when light or heat radiation causes the PIR sensor to turn on the system, and we simulate this by moving our hand over the PIR sensor. The second event is data that should be recorded for reference later, but the user of the system as deemed this data to not trigger an immediate alarm. The third and final event recorded does trigger an alarm and requires that SlugCam immediately transmit the recorded data.

Shown in figure 7 we show the events played over the 5 minutes, but SlugCam uses different duty cycles in all three scenarios. The First scenario exemplifies a system with no adaptive abilities so show a baseline for comparison. For all three event duty cycle E occurs, as previously described, where data is recorded and transmitted for all events. The second scenario now show cases the efficiency of adding computer vision to capture and analyze the video footage. SlugCam is able to correctly identify the first false positive event and immediately turn off 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. Scenario 3 showcases the system ability to determine that the event is non alarming, record the video, and store the footage locally to be transmitted at a later time. You can see the second even in the figure doesn't drop to where the WiFi is consuming data with a transmission, but it immediately goes into a sleep state using duty cycle d. These example scenarios showcase the largest power consumption reductions using the duty cycles we defined. One could imagine the numerous power efficiency gains should there be even more

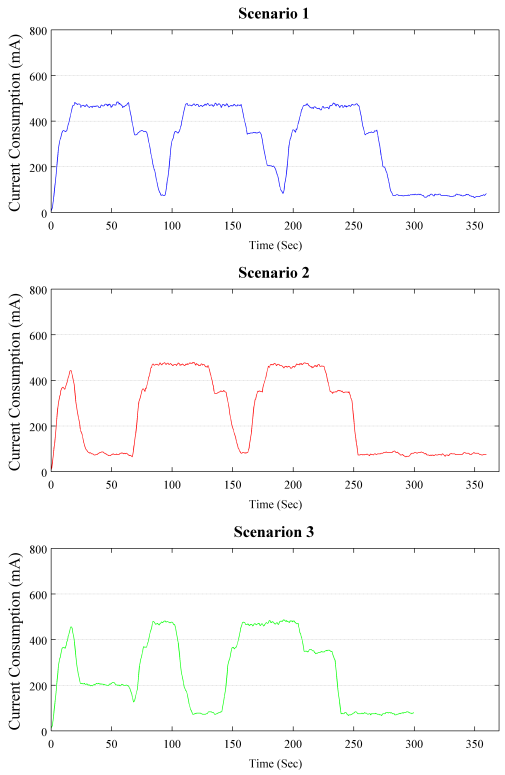

Fig. 7. Current consumption data for the three application scenarios.

duty cycles and smarter order of those duty cycles during normal operation.

C. UI Walkthrough and Application Examples

In order to demonstrate the client application and system usage we will use a simple one camera deployment we have performed. For this experiment we did not use the solar power to avoid the time necessary to charge the battery to capacity before each trial run. At the beginning of the trial all persistent data on the remote server was cleared, we then set up a camera node outdoors and ran through a duty cycle in which the camera would turn on, begin recording and tagging the video, and send the recorded video to the server.

Without configuring anything on the server, we positioned the camera and ran through our duty cycle several times, then we opened the client application to investigate the data collected by the system. When opening the client application from a web browser, we were brought to the main screen and notified that our camera node needed a location set in order to show on the map. We then followed the link to the camera configuration page. On the camera configuration page we were able to successfully set the camera location automatically using our browsers location data. Upon return to the dashboard, we were able to open the video on the activity bar (discussed below) for analysis. We saw all of the sent tags

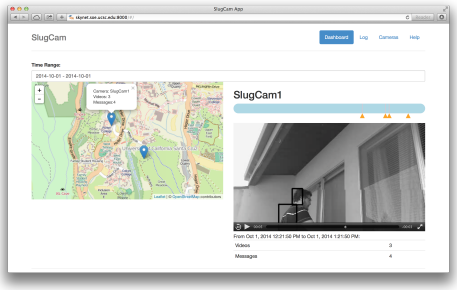

Fig. 8. Screenshot of client application after setup of two cameras and receiving data from actual camera node.

and videos on the activity bar and we were able to view the videos in the video player. We then used a camera simulation script to send a message to the server from a simulated camera to test the map display with multiple cameras. After setting the location for the second camera we returned to the dashboard and were able to switch between the cameras and still view the videos with tags. The videos also show the object recognition algorithm results represented as black squares around detected objects.

In order to present the cameras in an easy to understand way, we decided to allow for the mapping of cameras and the display of system data on maps of the monitored area (shown in figure **??**). 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. Our client application adds location data to cameras records, and is able to then recall it at a later time. This has the drawback that cameras must be renamed when moved to a new location, however, since camera name is easily configurable on the cameras, we did not see this as a major drawback.

We also built what we call the activity bar. 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 to the videos in which activity is happening in an intuitive way, but without hiding any information.This is also demonstrated in figure **??** above the video preview.

Another important feature mentioned earlier 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, though we hope to make a system that allows users to define actions taken when alarms are triggered. The interface we have developed (shown in figure **??** is intuitive and allows the user to visual select an area over recorded video, thus allowing an accurate selection given the current position of the camera.

The application also includes more traditional methods of

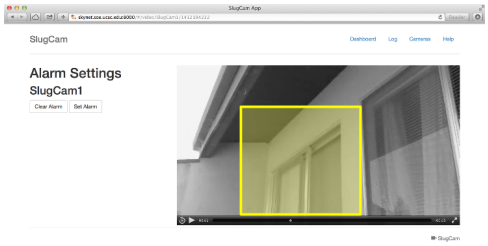

Fig. 9. Proposed interface for selection of alarm areas.

filtering and querying the data collection. There is a log view which allows for querying all collected data in a tabular format. This allows for information to be viewed in a more targeted way, where the other views allow for broader overviews of collected data.

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 [11], a significant body of work has resulted. Rather than an exhaustive survey of the state-of-the-art (which can be found in [11]), this section describes smart camera systems that are more closely related to SlugCam.

A. Selected Smart Camera Systems

Smart cameras typically combine energy efficient hardware with computer vision software which enables them to capture selected data from the scene to either store, transmit, or both [15], [2]. Computer vision techniques have advanced considerably and are able to not only identify objects accurately but also detect object behavior [1]. In order to keep up with advances in computer vision, smart camera systems require greater processing capabilities. This is especially the case in wireless platforms designed to be deployed outdoors sometimes in hard-to-reach areas. Researchers have also been using symbolic information from object motion behavior in the video footage in a similar way as SlugCam to extract more meaningful information from the environments they monitor [3].

We have evaluated recently developed smart camera platforms in a recent survey, and many system accurately track and detect objects in an energy efficient way [11]. A common goal of these systems and smart cameras in general is to have advance on board processing, so that with better video filtering and selection, less high resolution data is transmitted and network bandwidth gains are accomplished. Citric is one of the research smart camera platforms that accomplishes this best in dense smart camera deployments [2]. Using audio rather than PIR sensors, the system also promising long battery like through efficient hardware and smart video enabling and disabling based on the tracked objects location within the visual sensor networks monitored space. Taking a more hardware focused approach to smart camera research, Flexi-WVSN also does an extensive evaluation of older wireless visual sensor networks and some which are still in development today [10]. Realizing it was hard to stay up to date with hardware advances, the Flexi-WVSN has developed a modular system so no sub-component is dependent on any other piece of hardware. Rather than a two microprocessor system as SlugCam has, they chose a two radio system to achieve similar energy efficient gains. When smaller data files need to be sent, the system can switch to a more energy efficient Zigbee radio, otherwise they believe WiFi is the more reliable communication protocol to transmit video data [10].

B. Solar in Visual Sensor Network Research

Many sensor networks have been deployed outdoors with the ability to use solar as a reliable energy resource. Solar with the combination of rechargeable batteries really does allow for sensor networks to be less reliable on grid power [5]. UC Berkley has developed the worlds largest outdoor solar sensor network testbed with 557 nodes on one of their described deployments [6]. With such a massive 802.15.4 Zigbee network and an 802.11 backbone they did report some communication issues with their Zigbee networks. With solar being intermittent as we previously described, they too chose to use different duty cycle operation based on the energy available to run local node processing. The trio paper provides a full solar and battery analysis which is due to the on board power monitoring circuitry. Although visual sensors were included, the focus was on achieving the maximum sensor fusion capabilities possible.

Another more heterogeneous wireless sensor network that was deployed with solar capabilities was the FireWxNet system [24]. They too have to adapt to available energy resources through the ability to change duty cycling. Their vision sensor nodes don't include computer vision, because their auxiliary sensor nodes provide the system with enough information to allow the user to then decides 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 how ever sacrifices the ability to transfer large data packets of information.

A system which doesn't require high resolution video data and makes use of bluetooth and Zigbee radios is a solar wireless smart camera platform from Italy [15]. They throttle their image sensor based on the relevance of the data recorded and therefor uses less power and processing resources. The system also uses computer vision algorithms and like SlugCam chose background subtraction, but the system software as isn't easily accessable as SlugCams Linux based approach is. The system is also able to accurately extract object behavior tracking data through an analog passive infrared sensor. They evaluate their solar capabilities seperately and do not monitor actual deployment scenarios, but make an accurate estimate of how long their system and run consecutively on battery power.

VII. Future work and Conclusion

As a research platform SlugCam has many areas we wish to further develop. We plan on evaluating the system in larger networks and under different deployment scenarios. The solar system, although operational, could be further tested for robustness and operated in different weather conditions. We also will investigate further computer vision techniques to advance SlugCams computer vision capabilities, but we hope this could be accomplished by researchers specializing in computer vision. We believe we created a open and easy to use platform to allow for this. The network software is functional, but we plan on further testing and development to improve robustness and performance, followed by developing an API with more advance features. We would also like to investigate the possibility of using ad hoc wireless networking for camera communication to greatly improve remote deployment possibilities for the system. We also believe that cellular and satellite data transfer may also be used to great advantage with our system and would like to experiment with those technologies.

SlugCam is an wireless solar smart camera that is efficient in adapting to battery conditions while providing a complete user experience through the described Apps and management software we created. We detailed both the way SlugCams smart camera has been designed and operates, as well as the management software. We provide a complete power analysis of the prototype weve built along with a detailed overview of how to use the management software. SlugCam is a more powerful system that uses solar in an adaptive way to meet any conditions its presented with and we hope researchers find our smart camera platform the most cost effective and overall efficient system to date.

Acknowledgment

The authors would like to thank...

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] Thiago Teixeira, Dimitrios Lymberopoulos, Eugenio Culurciello, Yiannis Aloimonos, and Andreas Savvides. A lightweight camera sensor network operating on symbolic information. In *Proceedings of the 1st Workshop on Distributed Smart Cameras*, 2006.

[4] 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.

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

[6] Prabal Dutta, Jonathan Hui, Jaein Jeong, Sukun Kim, Cory Sharp, Jay Taneja, Gilman Tolle, Kamin Whitehouse, and David Culler. Trio: Enabling sustainable and scalable outdoor wireless sensor network deployments. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks*, IPSN '06, pages 407–415, New York, NY, USA, 2006. ACM.

[7] 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.

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

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

[10] 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.

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

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

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

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

[15] 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.

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

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

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

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

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

[21] Roger Wang. node-webkit. [Online; accessed 15-September-2014].

[22] Atom Shell. [Online; accessed 15-September-2014].

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

[24] 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.