.

Boston University
Electrical & Computer Engineering
EC463 Senior Design Project

First Semester Report

Community Managed Security System

Submitted to

Dan Ryan
VergeSense
724 Brannan Street
San Francisco, CA 94103 United States
1-617-618-5006
dan@vergesense.com

by

Team 24
Community Security

Team Members

Samuel Evans evanss@bu.edu
Esteban Hernandez hesteban@bu.edu
Daniel-John Morel djmorel@bu.edu
Steve Numata numatas8@bu.edu
Cameron MacDonald Surman csms@bu.edu

Submitted: December 10th, 2019

Table of Contents

Executive Summary

Community Managed Security System
Team 24 – Community Security

With recent advances in machine learning technologies, there is a growing concern over video data tampering. Currently, most privatized security systems are owned and operated by third-party companies with little accountability. Our project aims to get rid of third party involvement and provide accountable security through a community audible security. The final deliverable will be a video surveillance system that can be accessed through a web-based interface. The website will allow the user to view the live stream of the cameras and the recorded video data uploaded to the cloud storage. All actions taken by the users account will be recorded in a log that can be viewed by all users. This will ensure the privacy and security of the community.

1.0 Introduction

Dan's main concern when creating this project was the over privatization of data, specifically security feeds. As information about citizens expands with our ever increasing reliance on technology, the control over this data becomes more and more valuable. Dan is looking to create a security solution that doesn't forfeit the user's privacy and keeps it out of the hands of private companies and government bodies.

In order to fully understand the value of a community-driven security system, it's best to look at current statistics regarding concerns for online privacy. In the United States, over 75% of Americans are at least somewhat concerned about data vulnerability[1]. An estimated "$19 billion in 2018" was spent in the acquisition of consumer information[2]. As this market for data increases with time, so too will the desire for individuals to protect and keep that information out of private companies' hands.

Our group's approach to this project revolved around two concepts. We were looking to create a competitive security camera that is as close to industry standards as possible, all the while being completely unreliant upon the grid. We also needed to formulate an approach to the democratic holding of power and data. The idea was that not one person within the community can hold any real power over any of the other members. We also wanted to ensure that all data access is transparent, and all usage/downloads is visible to the entire community. This led to the idea of a logbook. It would be an immutable file, available to all residents.

Our aim to provide an industry standard camera will allow us to meet various requirements including strength and durability, visual capabilities, and ease of data transfer. Beyond that, the solar cell will allow us to have an autonomous and self-reliant product. The final design decision, the logbook, is likely the most important feature of our project. Ensuring its integrity is what will breathe life into the marketability of this camera system. We believe a logbook will provide much needed accountability among the user base and expose any possible abuse of the system.

While it is a required feature, the developed logbook should be an interesting mechanic, especially seeing how vital it is to this idea of a shared community managed security system. The system's sole charging mechanism, the solar panel, will be a great testament to the overall system's ability to operate with little-to-no advanced setup and easy integration into a community.

2.0 Concept Development

The creation of a community managed security system depends on three separate components coming together in synchronization. The three components are broken down into hardware, cloud, and web application. Information is streamed from hardware, managed by and stored in the cloud, and delivered by the web application. The top down technical description of the project is a deprivatized security system that supports the dynamic inclusion of cameras into the network (i.e. a camera can be automatically added to the camera network for a specific community). The video footage from the security camera is streamed to a cloud storage when motion is detected through a 4G cellular service, stored and processed by Amazon Web Services (AWS), and extracted from the web application. The concept development problem and approach has been divided into three separate sections.

The hardware component of the project encompasses the power supply, the physical camera module, and any preprocessing that has to be done with video footage. It is also responsible for transmitting data to AWS. The specific engineering requirements for the hardware portion are to be able to record footage with adequate visibility (should be able to see facial features), store footage onboard the camera in local storage for up to 30 days. The camera module must be solar powered, be able to record at night, and run 24/7 for up to a year without maintenance, and be able to transmit footage through a cellular 4G network. Finally, each camera module must cost less than $1000. In order to supply power to the camera, a 2x1x4 in solar panel was used. The size of the solar panel is kept at a moderate size to prevent it from becoming too noticeable. Originally a 25W 12V solar panel was used, however in order to increase the supply of power in suboptimal lighting conditions, a 35W 12V monocrystalline solar cell was used instead. As this system is intended to function at night as well, a 26000 maH battery is used to sustain the system overnight without sunlight. In addition to that, since the 12V input port on the battery is also used as an output source, a current controller is used in between the solar panel and the battery to ensure that the batter is not discharging. In selecting a microprocessor, our initial three options were narrowed down to a Raspberry Pi 3, a Beaglebone Black, and a Jetson Nano. The team selected to use the Raspberry Pi 3 for several reasons. While the Jetson Nano is all around a superior microprocessor for video analytics, most of the video analytics is being done within the cloud server. This made the cost and power consumption of the Jetson Nano outweigh the benefits. The Raspberry Pi 3 was selected alongside a 256GB SD card originally due to the fact that it had a compatible cellular shield, as well as a compatible camera with a night vision IR array, and an installable operating system capable of tracking motion to customizable degrees called motionEyeOS. MotionEyeOS has the ability to select frame rate, resolution, and motion detection sensitivity, and save data based on motion detection, making it ideal to select the Raspberry Pi, as that is the extent of video processing required onboard the camera module. The camera module only records data based on motion detection. This saves both power and storage space, while simplifying the amount of data transmitted to the cloud. The rationale behind it is that footage without motion will not capture anything of meaning, and can be omitted. This was confirmed to be acceptable by the client. The

Raspberry Pi 3 was eventually upgraded to a Raspberry Pi 4 in order to save power by underclocking the processor. Twilio was used as the sim card over Hologram due to its cheaper data transmission service. Finally, the complete camera module will require a case to hold the components. Solidworks has been used to design and 3D print prototype cases for the camera module, containing the battery, microcontroller, camera, ect. The final casing will need to be IP65 rated and be able to expel heat from the Raspberry Pi 4's microprocessor.

The next portion of the Community Security Project is the cloud infrastructure. Our client requires scalable infrastructure, video analysis, and redundant storage and we have determined that the optimal solution to these criteria is the usage of cloud services. We have decided to use Amazon Web Services as our cloud service provider as it is the industry standard and offers all of the tools and functionality we require. The most important AWS tool we will use is AWS Kinesis, which will act as both our data intake and processing tool. AWS Kinesis can process massive amounts of information each second, and video streams to HTTP Live Streaming (HLS) or S3 storage for later retrieval. It also has built in tools to assist in machine learning, which will be utilized later in the project. We will also use Elastic Compute 2, or EC2 in order to host our management platform. Kinesis, EC2, and additional tools are explained in more depth below.

The final portion of the Community Security Project is the user interface. The user interface's role is to provide a method of allowing users to view footage captured on security cameras. The specific requirements of the user interface is that the user should be able to view footage from various cameras including motion detected footage, and feature based footage such as person detection and license plate recognition. There should be a secure login, and privileges given to any "administrator" users should be extremely limited. Footage viewed by user to be limited to those recorded within the community, ie. a user in one neighborhood should not be able to access footage from another community. Finally a read only audit trail with information on which cameras were accessed by which users at a certain time should be available. The method of delivery to the users chosen was to create a web application rather than a mobile application. This was based on the idea that most users would likely be viewing security footage within their own homes on a laptop rather than on a mobile device. The platform for creating the web application was Django. Django was choses due to its easy use of python, scalability, and ability to create databases within its framework automatically. Html generators are being used to generate code for the html pages in order to save on development time. This entails using an online tool to generate html code based on an interactive user interface. The web application will allow the user to select a desired camera. Footage captured from the selected camera is divided into several categories, such as live streaming, license and person recognition, and motion detection, of which the latter has subcategories ordered by time and date. In addition to that, camera, user, and video footage should be segregated into different communities. In order to tackle this problem, and allow the dynamic integration of additional cameras into the network by entering a camera ID, a relational database will be used. This automatically links the ID of a camera to a community, as well as to its various collected video data. The database

will include camera and user data within a community within the database. Within each bin for a camera within a database, there will be links to various video footages, including livestream, which can be displayed in a web page either using an iframe or an alternative video browser. The database that will be used is MySQL, chosen for its compatibility with the Django framework, as well as its compatibility with Amazon AWS. Finally, the audit log with be a read only web page in which the user can select a camera, and see when it has been accessed and by who. This page will have no user interaction aside from scrolling and hitting the back/home button in order to prevent tampering.

3.0 System Description

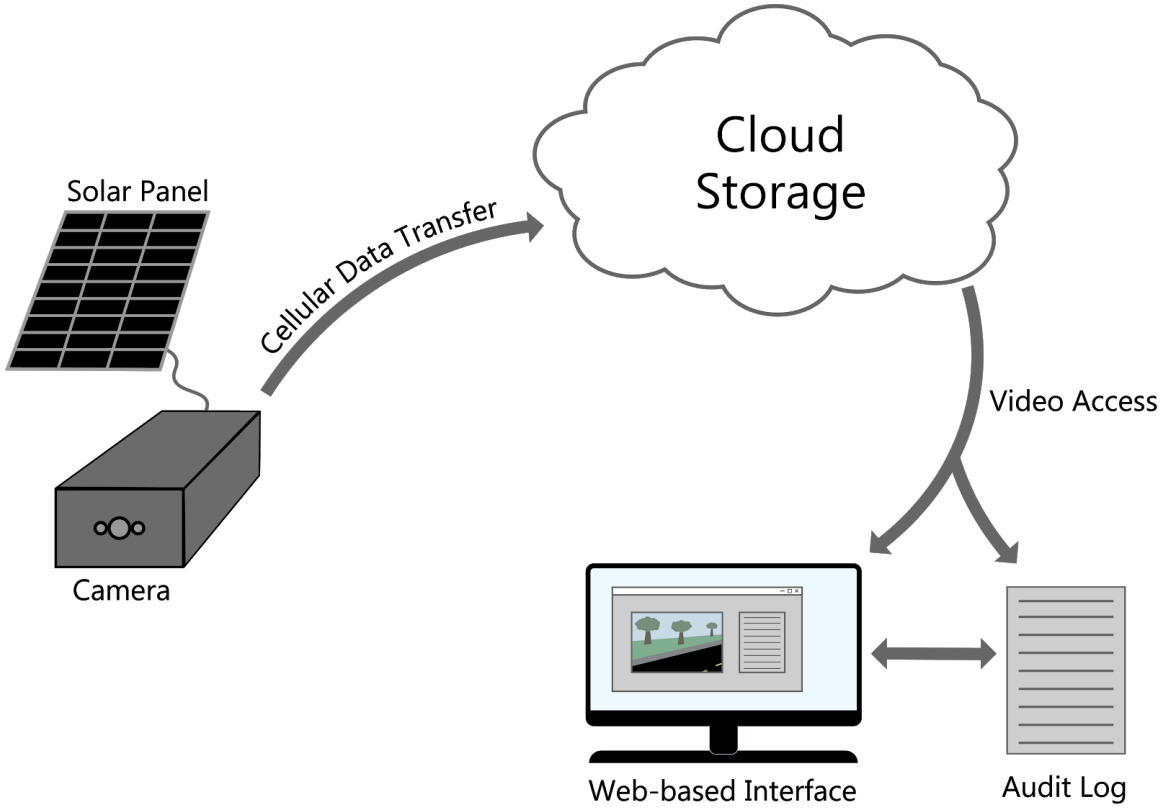


Figure 1: System Design Overview

3.1 Camera System

The current camera system is composed of two main hardware modules: the camera unit and the solar charging unit. A Raspberry Pi 3B+ microcontroller serves as the base for the camera unit. On top of the pi, there is an IR-CUT camera module to record video, and a cellular transmitter module to transfer video footage to cloud storage. The pi runs a security camera operating system, motionEyeOS, that offers camera configurations such as setting resolution, frame rate, and motion detection. The entire Raspberry Pi configuration sits inside an IP65, 3D printed box to protect the electronics.

The other aspect of the camera system, the solar charging unit, can be broken down into three pieces. The solar panel, charge controller, and battery connect in a chain off of the microcontroller to power the system. The overall connections are quite straightforward, and the charging mechanism insurance (current regulator) protects from overcharging the battery or battery discharge that makes it to the cell. This breakdown can be visualized by the figure below.

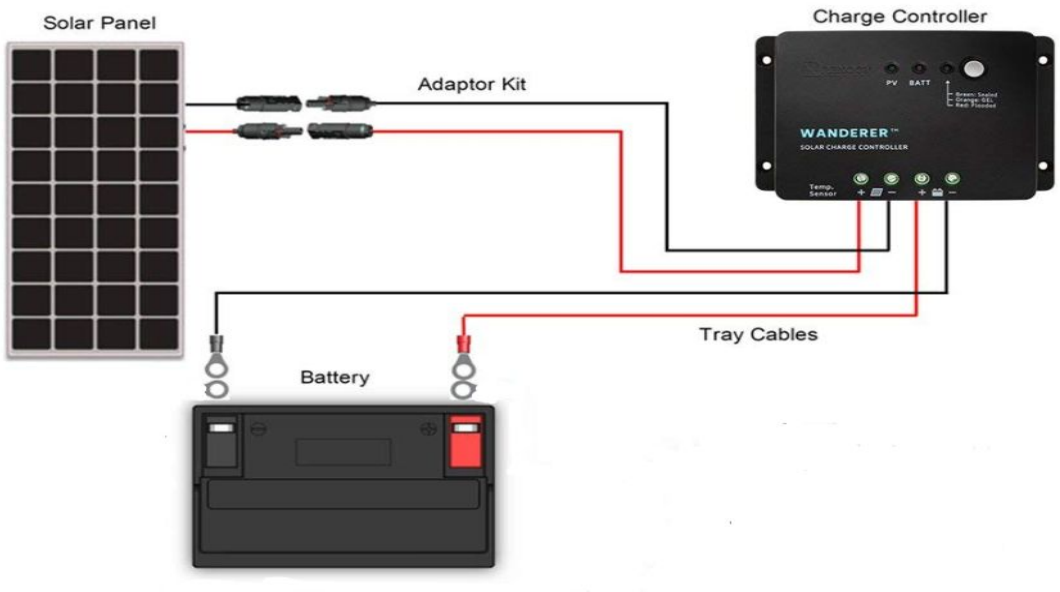


Figure 2: Charging System Overview [3]

3.2 Cloud Storage

Our initial concept for storing and serving video data to our management platform is outlined in **Figure 3** below. Utilizing AWS Kinesis Video Streams Producer SDK, we will package video data and send it to Kinesis via Twilio 4G LTE cellular data. From there, we can either serve the data directly to our management platform via HTTP Live Streaming (HLS), or use Kinesis Data Analysis and Lambda to draw insights from video data before being stored in S3 Storage. We can also choose to alert individuals or organizations via Simple Email Service (SES) and Simple Notification Service (SNS) based on thresholds we will define later. S3 will store data for up to 30 days before our data lifecycle moves it to Glacier for long term storage. We can then serve on-demand video using CloudFront to our platform as well. Elastic Compute 2 (EC2) will host our management platform, and enable us to distribute data to our customers. Another feature to be implemented later is the use of CloudWatch and CloudTrail for our audit logs, allowing users to view aggregated or specific data related to stream and VOD access.

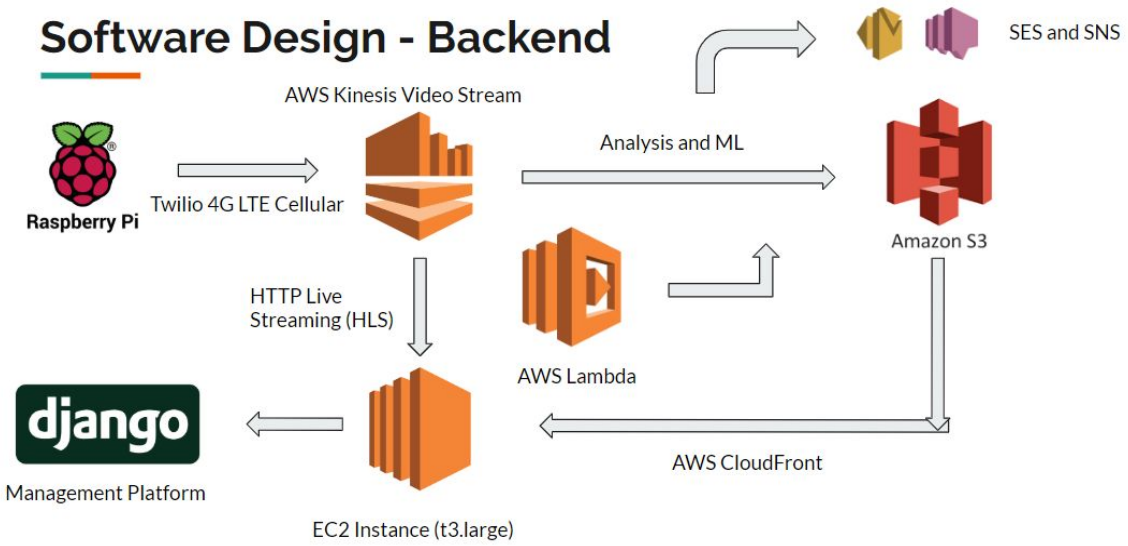


Figure 3: Backend Design Overview

3.3 Web Development

The complete web development application will have a secure user login. Upon registration, a community ID link will be entered to ensure that the user gets added to the correct group. Administrator users will have the ability to add cameras to the community. The home page of the web application will give the user a display various camera IDs. Each link will take the user to a camera specific to it, and give the option to allow the user to view live stream, motion detection, video analytics footage, or camera logs by clicking on links. For motion detection and video analytics, a popup or another page will allow the user to select from a list of footages sorted by date. Footage will be displayed either on a browser or an iframe. The web application will note the user's activity in viewing footage, and write it into the audit trail. The database for the web application will contain user and camera information, as well as links to various instances of recorded video. The database will be relational, and items within it will be linked by an ID, essentially sorting out the issue of making sure each community have access to footage from its respective cameras.

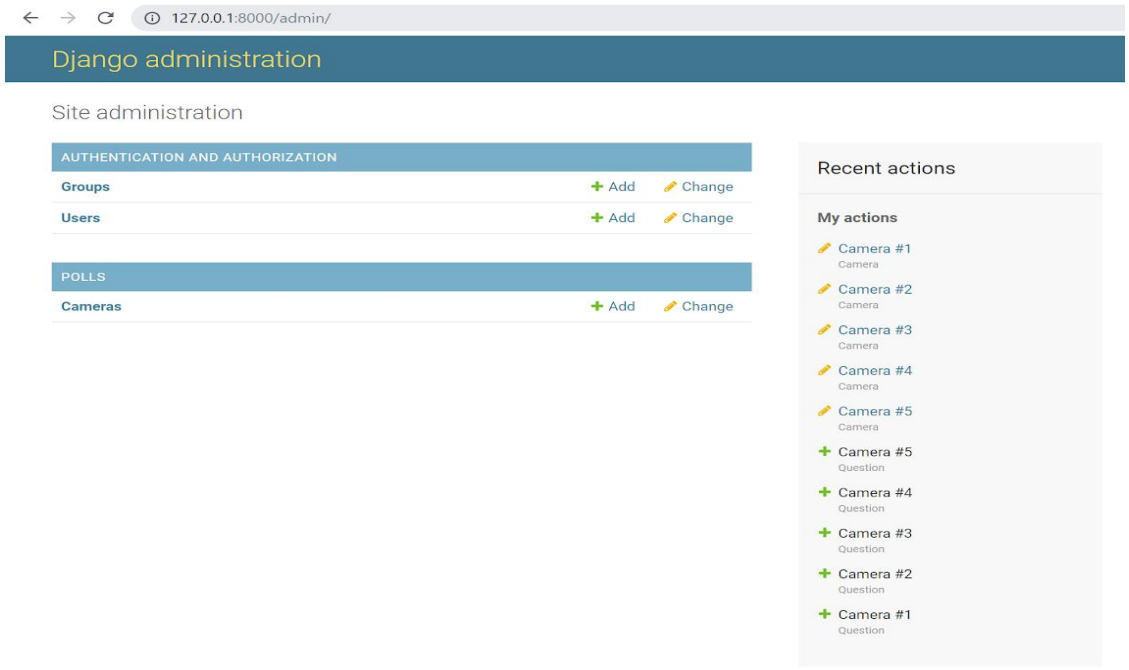


Figure 4: Early concept of administrator page with the ability to add cameras to the network.

4.0 First Semester Progress

This semester, the team primarily focused on the hardware aspect of the project, but also began web development. When testing, the team divided the security camera design into modular parts that way each component could be tested in parallel. For hardware, the team tested the solar charging unit and Raspberry Pi with motionEyeOS. For software, the team began the customer web interface and developed a budget for AWS cloud services.

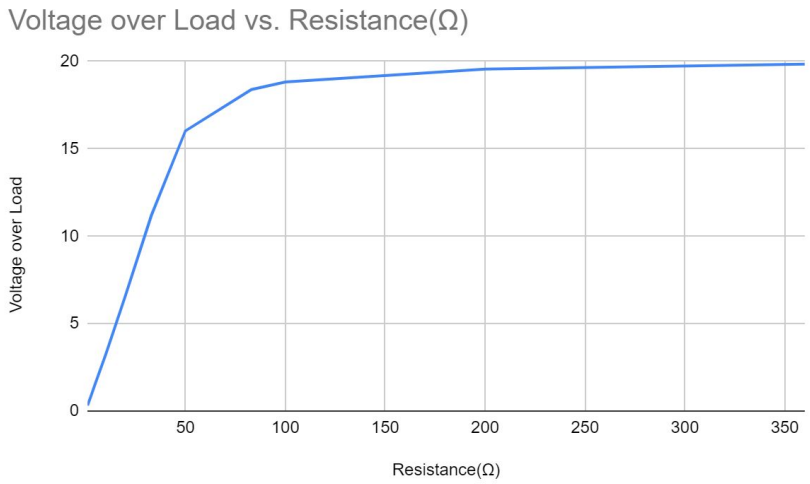
4.1 Solar Charging Unit

As it currently stands, the system's only charging mechanism is its PV cell connected off of the battery. Based on the rough estimates, we believe a singular 25 Watt cell connected to a high capacity Lithium Ion battery would suffice to keep the system online at all times through the year. The type of battery we chose had more to do with maximum capacity and weight. We wanted to find the highest capacity battery without weighing down the camera or expanding its overall size too much. Once we had these determined, we did connection testing and charging capabilities of the pair.

To ensure the two components don't break each other either due to overcharging of the battery or if the battery discharged into the cell, we used a charge controller included in the purchase of the solar cell. As one extra level of protection we soldered a high current rated diode forward facing into the charge controller to protect the panel from any battery discharge. Testing began once these additional safety measures were implemented.

The first performed test was simple voltage testing of the cell during various weather conditions. We then tested over various resistor loads during peak sunlight hours to get an idea of what current values we would be looking at going into the battery. After all of the basic solar panel tests were finished, we connected it to the charge controller and battery to begin charge testing. This is where some issues began.

The setup for charge testing was quite simple, we first got the Voc for the battery then plugged into into the charge controller waited around 30 min and then tested the Voc again. When this basic test was conducted, we got results that were rather unexpected. The battery simple didn't charge at all. Even though the cell wasn't going to recharge the battery at a very high rate, there still should've been a change to the Voc which there wasn't. This led to further testing of the components and opening up the battery to investigate the circuitry. In the end, we figured out it was the free charge controller that was the problem.



Resistance (Ω)	Load Voltage(V)	Voc	Isc
1000	20.09	20.92 V	0.364
510	20.11		
360	19.845		
200	19.56		
100	18.82		
83	18.388		
50	16.02		
33	11.203		
19.88	6.56		
10	3.201		
5	1.598		
2.5	0.782		
1	0.314		

Table 1: Basic Voltage and Current Testing of 25 Watt PV Cell

Our other power balance testing involved the current draw from the Raspberry Pi. At first we were very concerned about the overall power consumption of the Pi because online sources pointed to a very high draw. These numbers were so high, in fact, that over the course of just 24 hours it was estimated that nearly 50% of the battery's total charge would be consumed. Based off of our recharge rates that we calculated using the theoretical current of the panel, we would only get about 42% of the battery charged under good conditions (8 hours of sunlight). These were obviously incredibly poor results and further testing had to be done regarding the overall draw of the Raspberry Pi.

Resolution	Frame Rate	Vinitial	Vpost	ΔV	Test Duration
240	30	10.593	10.534	0.059	26
480	30	10.534	10.467	0.067	26
720	30	10.467	10.4	0.067	26
1080	30	10.395	10.306	0.089	26

Table 2: Battery Voltage Drop After Recording Video w/ Varying Settings

To do this testing, we bought a power meter that sat between a wall outlet and the AC/DC charger for the Raspberry Pi. This power meter gave us live current draw values as we changed things such as resolution, framerate, and environment lighting (for IR testing). Our results were drastically better than those found from online results. The consumption went from nearly 50% to closer to 9.2%. With these new values, even with Boston's average of 4 hours of sunlight during the winter, the system will easily be able to stay on and last. This test proved that the solar panel was still a viable charging mechanism.

After these various tests and discoveries, a few changes were made. We ordered a more powerful PV cell (35 Watt) as well as a new charge regulator. When these parts

arrived, we reconducted the old testing with much greater success. The new parts work incredibly well and will be sufficient for the remainder of the project.

4.2 Raspberry Pi with motionEyeOS

The current security camera uses a Raspberry Pi 3B+ as the microcontroller and an IR-CUT camera module for recording video. The Raspberry Pi runs on the open source operating system, motionEyeOS, which is a useful tool for adjusting camera settings such as resolution, frame rate, motion detection, video file format, and more. The Raspberry Pi also has a 256GB micro SD card for local storage, which was estimated to be a large enough size during the initial project research.

Since the final product must store 30 days of recorded video on a micro SD card, the Raspberry Pi setup was tested to determine the optimal video resolution and frame rate combination. There were two rounds of testing to (1) find the optimal recording settings, and (2) determine how motion affects video file sizes. Both rounds of testing set the video file format to H.264 (.mp4) at 50% movie quality within motionEyeOS.

The first round of testing considered 3 video resolutions (240p, 480p, and 720p) each at 5 different frame rates (10fps, 15fps, 20fps, 25fps, and 30fsp). The Pi recorded 5 minutes of video at each video resolution and frame rate combination. Upon completion, the video's file size and duration was recorded to determine its video storage rate in KB/s. That rate was then linearly scaled to calculate the necessary storage size for 30 days of continuous recording as seen in **Table 3**. Based on the collected results, 480p at 30fps is the optimal combination as it provides the best video quality settings while not exceeding the 256GB micro SD card limit.

	Resolution		
Frame Rate	320 x 240p	640 x 480p	1280 x 720p
10 fps	60.37 GB	130.92 GB	329.76 GB
15 fps	63.60 GB	132.97 GB	262.63 GB
20 fps	62.69 GB	120.95 GB	298.00 GB
25 fps	58.22 GB	114.28 GB	425.48 GB
30 fps	54.63 GB	105.21 GB	414.95 GB

Table 3: Video Storage for 30 Days of Continuous Recording

The second round of testing considered how motion affects video storage. This round took inspiration from **Table 3**'s results as the video storage sizes had a strange correlation when increasing the frame rate. Specifically, **Table 3** illustrates that increasing the frame rate can either increase or decrease the storage rate, which is counterintuitive. For the second round of tests, the resolution was fixed at 480p, but the 5 different frame rates were still tested. The main difference for these tests was that the *Motion* trials had a hand wave one foot in front of the camera for the entire testing

duration. This ensured the entire recorded frame changed. The *No Motion* tests, however, just had the camera record a wall, so to resemble a static environment.

	Resolution (640x480p)	
Frame Rate	No Motion	Motion
10 fps	144.76 GB	274.47 GB
15 fps	147.28 GB	322.13 GB
20 fps	121.80 GB	370.60 GB
25 fps	113.14 GB	416.53 GB
30 fps	143.18 GB	425.98 GB

Table 4: Video Storage for 30 Days of Continuous Recording (Motion Tests)

According to the *Motion* results in **Table 4**, increasing the frame rate directly increases the necessary storage. The *No Motion* results, however, do not have a direct correlation, so the video compression mechanism likely optimizes the frame data information when recording a static environment to reduce file size. While the results in **Table 4** suggest that 480p at 30fps will go over the 256GB local storage, it should be noted that the results show the storage size for continuous recording. The final camera system will only save motion-detected clips for local storage, so the required storage per month will be significantly less. Additionally, the actual recording environments will more likely resemble the *No Motion* trials. Since the cameras will be mounted on elevated objects such as buildings, trees, and lampposts, motion in the environment (be that people or cars) will encompass a relatively small percent of the overall frame. This means that most of the recorded frame will remain the same, so the recorded videos will resemble the *No Motion* data rather than the *Motion* data.

Actual field test data is necessary to confirm the camera settings and their impact on video storage size, which will be gathered in the Spring semester. However, the results from the Raspberry Pi tests give an idea of what recording settings can be used to initially test the system, and provide an idea of video quality for cellular transmission and machine vision analysis.

4.3 Web Interface

The with the Django web development application so far has been to create a portion of the web application with limited functionality. The web application so far is divided into two categories, an admin page and a user page. The admin page has user verification, where an account can be registered as an admin and login securely. The admin has the ability to add users and cameras, along with sample data fields. So far on the user end, the user has the ability to click on a display of existing cameras, and display meta-information such as camera ID and its location (it is just dummy data since it is not setup for footage yet. This portion of the project is functional, and can be demonstrated. The templates for the web pages so far are very primitive, however using online html

code generators we have been able to create better templates, such as ones which include iframes which are capable of displaying a browser. We have created a basic database for the cameras and users on sqlite3, within Django. It is a locally stored database that will eventually be switched to MySQL on AWS. The database is important as it will store links to video footage as well as separate users and cameras into different communities, and allow for the dynamic addition of cameras into the network. The database itself at the moment can be viewed using an online SQL browser. The server for the web application is currently installed as a virtual environment a laptop, meaning that the server must be running from my laptop in order to function properly on a browser. This will eventually be hosted by Amazon EC2.

4.4 AWS Budget

One major requirement of our project is the utilization of cloud services, and we have decided to use Amazon Web Services as it is the industry standard, is used by our client, and is well documented. Some of the features required of our cloud service include managing our project's infrastructure, content delivery to end-users, and built-in analytics and cloud computing tools. Due to the recurring and consistent nature of cloud service payments, we developed an estimated AWS budget as one of our tests in order to better estimate our fixed costs.

AWS Service	Specs	Quantity	Cost/Unit	Estimated Cost	Description
Elastic Compute 2 (EC2)	t2/t3.large - 2 CPU, 8 GB	732 hr (1 month)	$0.0928/hr	$67.93	For hosting web server
S3	Standard storage	1000 GB	$.023/GB	$23	Video storage, 30 days
Kinesis Stream	30 records/sec - 4 KB/record	1 shard	$12.09/shard	$12.09	From Amazon estimate of 7.5 MB/min for video streams
Glacier	Longterm storage	1000 GB	$0.004/GB	$4.00	Long term storage
SES	Negligible	0	$0	$0.00	Simple Email Service
SNS	Negligible	0	$0	$0.00	Simple Notification Service
IoT Core	Negligible	0	$0	$0.00	Connectivity, messaging, shadow operations
VPC	TBD	0	$0	$0.00	Virtual Private Cloud
Route 53	TBD	0	$0	$0.00	DNS
Cloudfront	TBD	0	$0	$0.00	Media delivery - similar to kinesis
			TOTAL:	$107.02	

Table 5: AWS Initial Budget
(Note: An expanded view of this table is located in part 6.0 Budget Estimate)

Our total fixed costs are estimated to be about $100/month, with our primary costs being EC2 uptime and Kinesis shards. Some of our estimates also include S3 and Glacier storage costs, which will be lower as we start to build and test our infrastructure. Another point to mention is the 'Negligible' and 'TBD' costs in the budget. Negligible costs, such as IoT Core, are listed as such due to their small or infrequent nature. For example, IoT Core connectivity costs $0.04 per year per device, and is irrelevant on a monthly basis. Also, some of these features will take advantage of AWS Free Tier, which lasts for one year and will reduce costs. Other 'TBD' features, such as Virtual Private Cloud, Route 53, and CloudFront, are production-level features that are not required at this time, or are unable to be estimated as they are tied closely to end-user usage. Another point to mention is that many of these costs are higher initially, and are subject to economies of scale. This means that adding additional cameras/networks to our project should cost less

as our infrastructure grows (ex. only need enough EC2 instances to support current network traffic, all networks can use the same instance).

One infrastructure-related cost not included in our budget is that associated with Twilio 4G LTE cellular data as it is too variable to estimate at this time. This cost is advertised as $.01/MB, and will be a vital resource to control in order to reduce our overall operational costs. One way we plan on mitigating this is to use less computationally intensive forms of edge video analysis (such as motion detection) in order to only stream video when necessary or requested by a user.

This budget will help us more efficiently allocate cloud resources, as well as help us plan for future costs. As we continue to develop our infrastructure next semester, we will update this budget and deliver a final budget for long-term production costs associated with our platform. We will also be able to estimate the costs associated with bringing multiple networks or ‘clients’ online, helping us to determine the operational costs to be handled by the end consumer.

5.0 Technical Plan

Task 1. Cellular Shield Installation
Lead: DJ; Assisting: Sam
The cellular shield shall be installed onto the Raspberry Pi 3B+, and configured to have the Pi connect to the internet via cellular data.

Task 2. Dummy Data Transfer
Lead: DJ
Test the cellular shield configuration to ensure videos can be transferred from the Pi to an external device via the motionEyeOS interface. Also test transferring videos from the Pi to the AWS cloud storage.

Task 3. Setup Consistent Data Stream
Lead: Cam
Set up AWS Kinesis shards (throughput units), and integrate with the AWS Video Streams Producer SDK on the Raspberry Pi via wifi. This should enable simple transfer of video data to AWS for later analysis and distribution.

Task 4. Continuous AWS Data Sorting
Lead: Cam
Set up AWS Kinesis Data Analysis and Lambda to analyze and sort input video data before storing in S3. Setting up this framework of input data flow control will enable us to perform a more detailed analysis and use machine learning in the future.

Task 5. Pull Dummy Data from AWS
Lead: Steve
Links to sample online videos will be pushed to several cameras within the relational database and displayed on a webpage within the web application.

Task 6. Pull Video Data from Camera
Lead: Steve
Once video footage is transmitted to AWS from a camera, extract it using the web application and view the footage.

Task 7. Complete User Interface
Lead: Steve
Complete a full web application user interface which allows a user to view specific types video footage from selected cameras by clicking on links.

Task 8. Log User Interaction with Data
Lead: Steve; Assisting: Cam
Enable integration of AWS CloudWatch logs with management platform. Task will include the partitioning of data using AWS security groups, as well as filtering and formatting of relevant information.

Task 9. Handle Livestream requests from Kinesis
Lead: Steve; Assisting: Cam
Give users the ability to start a live stream on a specific camera, then display live stream via Kinesis and HTTP live streams (HLS).

Task 10. Complete Backend Database MySQL
Lead: Steve; Assisting: Cam
In progress, use MySQL to support management platform data and content distribution.

Task 11. Host Webdev on EC2 Instance
Lead: Cam
Create EC2 instance with DNS, and upload current version of management platform via SSH. Can integrate with github for accelerated development cycle.

Task 12. Testing Machine Vision Libraries
Lead: Esteban
Install MotionEye library and begin connecting the Rpi, camera, SD card, and ethernet cable. Then test machine vision by sending commands to the camera via Raspberry Pi.

Task 13. Develop Machine Analytics Web Interface
Lead: Steve
Create an option under each camera's web page to take users to a page to view footage processed with machine vision.

Task 14. Device Backend Processing of Video Clips
Lead: Cam
Use Kinesis and Lambda to draw insights from video clips, and use that data to develop aggregated or clip-specific analysis to add value to the platform.

Task 15. Present Machine Vision Outputs to Users
Lead: Steve
The ability to use person or license plate recognition will be an option under each camera displayed on one of the pages of the web application. Should be able to pull link to footage from database.

Task 16. Log User Machine Analysis Requests
Lead: Steve; Assisting: Cam
Integrate CloudWatch and CloudTrail to determine usage and access to analytics data.

Task 17. Design and Produce IP65 Casing
Lead: Sam; Assisting: Steve
After researching proper sealing and protection measures for similar security cameras we will then use CAD to design our casing before integrating our various parts into the container. High strength plastics will be a focus for materials choice as they are lightweight and affordable.

Task 18 & 19. Test Product on Cummington Mall
Lead: Sam; Assisting: DJ
Combine all electronics and casing into one unit, and mount the camera system to a location on Cummington Mall. Have the unit record video for a week, and make necessary adjustments to the system as needed.

6.0 Budget Estimate

Hardware Costs

Item	Description	Cost
1	TalentCell Lithium Ion Battery	$65
2	Raspberry Pi 3B+	$35
3	Raspberry Pi 3B+ Case	$10.99
4	Raspberry Pi 4	$47.69
5	Raspberry Pi IR-CUT Camera Module	$25.99
6	25 W 12V Solar Cell	$59.96
7	256GB Micro SD Card	$38.65
8	Power Meter Monitor	$19.99
9	35W Monocrystalline Solar Cell	$40.55
10	Charge Controller Module	$17.99
11	USB 2.0 A Male to Micro B Cables	$7.31
12	Micro HDMI to HDMI Cable Adapter	$7.99
13	USB Type-C to USB 2.0 A Male Cable	$6.99
	Total Cost	$384.10

AWS Budget and Resource Estimates

AWS Service	Specs	Quantity	Cost/Unit	Estimated Cost	Description
Elastic Compute 2 (EC2)	t2/t3.large - 2 CPU, 8 GB	732 hr (1 month)	$0.0928/hr	$67.93	For hosting web server
S3	Standard storage	1000 GB	$.023/GB	$23	Video storage, 30 days
Kinesis Stream	30 records/sec - 4 KB/record	1 shard	$12.09/shard	$12.09	From Amazon estimate of 7.5 MB/min for video streams
Glacier	Long Term storage	1000 GB	$0.004/GB	$4.00	Long term storage
SES	Negligible	0	$0	$0.00	Simple Email Service
SNS	Negligible	0	$0	$0.00	Simple Notification Service
IoT Core	Negligible	0	$0	$0.00	Connectivity, messaging, shadow operations
VPC	TBD	0	$0	$0.00	Virtual Private Cloud
Route 53	TBD	0	$0	$0.00	DNS
Cloudfront	TBD	0	$0	$0.00	Media delivery - similar to kinesis
			TOTAL:	$107.02	

Cellular Variable Costs

Twilio 4G LTE Cellular Data	$0.1/MB

7.0 Attachments

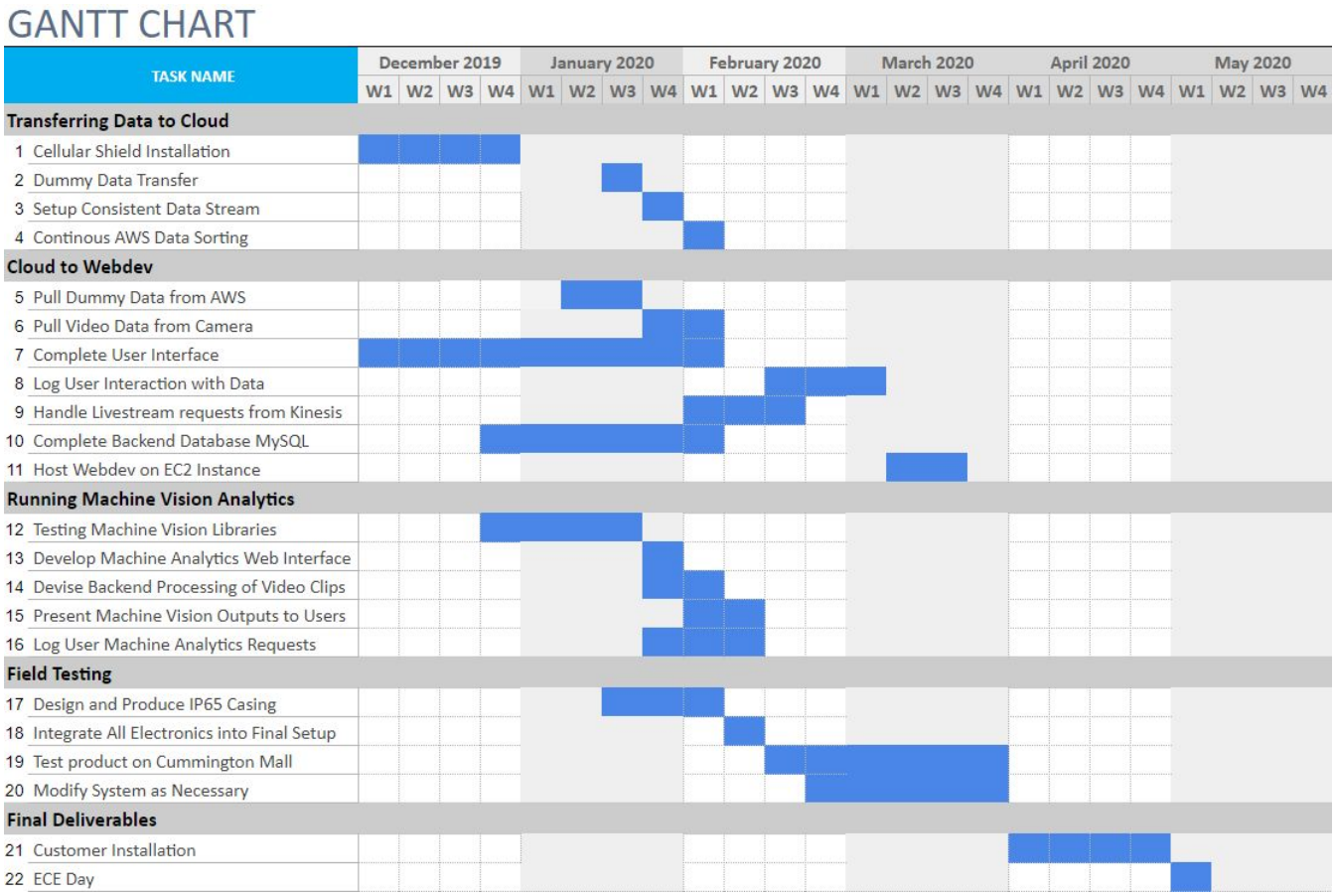
7.1 Appendix 1 – Engineering Requirements

Team 24 Team Name: Community Security

Project Name: Community Managed Security System

Requirement	Value, range, tolerance, units
Case dimensions	6 in x 4 in x 3.5 in
Case protection	IP65 rating
Camera Weight	< 6 lbs
Power Source	Solar Cell Optional: Power over Ethernet
PV Cell Sizing	2 ft x 1 ft x 5 in
PV Cell Weight Limit	< 8 lbs
Data Transfer	Cellular Data 4G LTE
Local Storage	30 days of local storage
Cloud Storage	30 days on AWS S3, 1+ years in AWS Glacier
Sustainability	24/7 operation for a full year without maintenance
Livestream Latency	< 1 minute
Camera System Budget	< $1000 per Camera
Audit Log	Only Read Access No Ability to Write to file
Machine Vision	Detect: People, Cars, License Plate Numbers Trigger event when detection occurs
Camera Module	≥ 5 Megapixels Night Vision Capabilities

7.2 Appendix 2 – Gantt Chart



7.3 Appendix 3 – Other Appendices

Bibliography

[1] J. Clement, “U.S. online users who feel their data is vulnerable to hackers 2019 l Statistic,” *Statista*, 01-Aug-2019. [Online]. Available: https://www.statista.com/statistics/972911/adults-feel-data-personal-information-vulnerable-hackers-usa/. [Accessed: 10-Dec-2019].

[2] L. Matsakis, “The WIRED Guide to Your Personal Data (and Who Is Using It),” *Wired*, 19-Feb-2019. [Online]. Available: https://www.wired.com/story/wired-guide-personal-data-collection/. [Accessed: 10-Dec-2019].

[3] Randy, “Renogy 200W 12V Solar Panel Monocrystalline Off Grid Starter Kit with 30A Wanderer Charger Controller,” *Walmart.com*, 16-Jul-2018. [Online]. Available: https://www.walmart.com/ip/Renogy-200W-12V-Solar-Panel-Monocrystalline-Off-Grid-Starter-Kit-with-30A-Wanderer-Charger-Controller/118614164. [Accessed: 10-Dec-2019]. Figure (2)