Boston University
Electrical and Computer Engineering
EC463 Senior Design Project

1st Prototype Test Report

Team 24
Community Security

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

November 17th, 2019

1.0 Introduction

For the first prototype, our team divided the overall security camera design into modular components in order to prove that each can work on its own. This modular testing serves as a proof of concept that those aspects of the overall project are individually functional, so they can be combined later on into a fully operational product in future prototyping.

In terms of testing, we tested three different components. First was the Raspberry Pi 3B+ to make sure it can run motionEyeOS and record videos. Second was the Solar Charging Unit to ensure the solar panel can charge the battery, which will eventually power the Raspberry Pi. Third was the AWS estimated plan cost, which compares the different AWS services to determine the optimal plan for our project.

2.0 motionEyeOS on Raspberry Pi 3B+

The target module is the Raspberry Pi 3B+ running the motionEyeOS operating system. The pi connects to a camera with an infrared matrix array for recording video. Its overall responsibility is to record footage, and store it locally at a specified frame rate and resolution. Video segments with motion detected should be streamed to a cloud server at a certain frame rate and resolution in the final product. However, the first prototype tests relied on accessing video feeds from an external device (laptop) that connects to the motionEyeOS interface using the Raspberry Pi's IP address in a browser.

The testing for recording video determines how much storage is used when recording at a certain frame rate. This allows us to find a balance between viewability of footage recorded at certain frame rates and resolutions, and allows us to optimize the camera system to store viewable footage while conserving storage usage.

2.1 Test Setup

Prior to testing, the motionEyeOS was installed on the Raspberry Pi, and the pi was connected to the BU Guest (unencrypted) wifi network. Once the pi boots the OS, an external device (laptop) connects to the motionEyeOS interface using the pi's IP address in a web browser. From there, the external device adjusts the video recording configurations in motionEyeOS to set the frame rate and resolution.

The test considers 3 video resolutions (240p, 480p, and 720p) and 5 frame rates (10fps, 15fps, 20fps, 25fps, and 30fps). The Raspberry Pi records a video at each video resolution and frame rate combination for a duration of 5 minutes each. At the end of the 5 minutes, the

external device downloads the video through the motionEyeOS interface to demonstrate the video can be transferred from the pi. From there, the external device plays back the video in VLC Media Player to ensure the video can be viewed. After confirming the video's viewability, its file size is recorded and then linearly scaled to get a calculated storage size for 30 days of constant video under that video's recording configuration.

While the video resolution and frame rates change, some camera settings remain constant. Specifically, the video file format was set to H.264 (.mp4) at 50% movie quality as it is a universal file format with a small file size. The camera also set recording to continuous recording rather than motion detection. Although the final product will store videos based on motion detection, using the continuous recording setting gives an upper bound for the necessary video storage.

Once the videos are stored on the Raspberry Pi's micro SD card, the external device downloads the videos through the motionEyeOS interface to ensure the videos can be transferred and played back to some other device.

2.2 Data

	Trial 1			Trial 2			Trial 3		
Setting	**File Size**	**Duration**	**KB/s**	**File Size**	**Duration**	**KB/s**	**File Size**	**Duration**	**KB/s**
240p @ 10fps	8329 KB	313 s	26.61 KB/s	6689 KB	305 s	21.93 KB/s	7689 KB	311 s	24.72 KB/s
240p @ 15fps	9476 KB	338 s	28.04 KB/s	8288 KB	370 s	22.40 KB/s	8264 KB	309 s	26.74 KB/s
240p @ 20fps	9551 KB	311 s	30.71 KB/s	7261 KB	342 s	21.23 KB/s	7560 KB	313 s	24.15 KB/s
240p @ 25fps	8338 KB	336 s	24.82 KB/s	6762 KB	309 s	21.88 KB/s	7378 KB	308 s	23.95 KB/s
240p @ 30fps	7392 KB	312 s	23.69 KB/s	6111 KB	313 s	19.52 KB/s	7038 KB	305 s	23.08 KB/s
480p @ 10fps	19502 KB	315 s	61.91 KB/s	15426 KB	322 s	47.91 KB/s	16090 KB	328 s	49.05 KB/s
480p @ 15fps	20798 KB	320 s	64.99 KB/s	15016 KB	324 s	46.35 KB/s	15461 KB	309 s	50.04 KB/s
480p @ 20fps	17833 KB	296 s	60.25 KB/s	12896 KB	306 s	42.14 KB/s	14559 KB	328 s	44.39 KB/s
480p @ 25fps	18209 KB	309 s	58.93 KB/s	12007 KB	310 s	38.73 KB/s	13130 KB	320 s	41.03 KB/s
480p @ 30fps	16344 KB	306 s	53.41 KB/s	11905 KB	314 s	37.91 KB/s	11122 KB	306 s	36.35 KB/s
720p @ 10fps	36890 KB	271 s	136.13 KB/s	40551 KB	312 s	129.97 KB/s	41036 KB	306 s	134.10 KB/s
720p @ 15fps	31322 KB	306 s	102.36 KB/s	35541 KB	318 s	110.76 KB/s	33688 KB	319 s	105.61 KB/s
720p @ 20fps	39020 KB	307 s	127.10 KB/s	38053 KB	343 s	110.94 KB/s	37823 KB	306 s	123.60 KB/s
720p @ 25fps	65623 KB	308 s	213.06 KB/s	30547 KB	303 s	100.82 KB/s	62363 KB	308 s	202.48 KB/s
720p @ 30fps	66023 KB	336 s	196.50 KB/s	30357 KB	313 s	96.99 KB/s	63660 KB	303 s	210.10 KB/s

Table 1: Video Storage Size from 5 Minutes of Continuous Recording

	Resolution		
Frame Rate	320 x 240p	640 x 480p	1280 x 720p
10 fps	24.42 KB/s	52.96 KB/s	133.40 KB/s
15 fps	25.73 KB/s	53.79 KB/s	106.24 KB/s
20 fps	25.36 KB/s	48.93 KB/s	120.55 KB/s
25 fps	23.55 KB/s	46.23 KB/s	172.12 KB/s
30 fps	22.10 KB/s	42.56 KB/s	167.86 KB/s

Table 2: Average Video Storage Size per Second

	Resolution		
Frame Rate	320 x 240p	640 x 480p	1280 x 720p
10 fps	63.30 GB	137.27 GB	345.77 GB
15 fps	66.69 GB	139.42 GB	275.37 GB
20 fps	65.74 GB	126.83 GB	312.47 GB
25 fps	62.08 GB	119.83 GB	446.14 GB
30 fps	59.82 GB	110.32 GB	435.10 GB

Table 3: Video Storage Size for 30 Days of Continuous Recording

2.3 Conclusions

The raw collected video data is shown in **Table 1**. Using the file size and video duration, the KB/s rate was calculated for each video configuration. **Table 2** takes all of those rates and finds the average rate across the different trials. From there, **Table 3** calculates the required storage size when those average rates are scaled to 30 days of continuous video recording.

Based on the information gathered in **Table 3**, our 256 GB micro SD card is capable of recording 480p video at 30fps, and still have extra storage space for storing 30 days of continuous video. Although our security camera will rely on motion detection for recording video, knowing that our security camera can still store 30 days of continuous video at 480p without onboard storage concerns is important, as we have an upper bound for what our system can handle.

While the three tables demonstrate that increasing the video resolution increases the video storage size, there isn't an exact trend for the frame rate. In some instances, increasing the frame rate increases the necessary storage size. This trend is to be expected as higher frame rates means more frames must be stored in a video. However, the collected data shows some complications as increasing the frame rate sometimes decreases the file size. During testing, the Raspberry Pi was connected to a monitor and there was an under-voltage throttled

warning displayed. Based on that warning, it is suspected that the Raspberry Pi may have had difficulty in achieving the higher video settings, and ended up cutting on some of the video quality which reduced file size. In order to remove that error, we'll need to adjust the power supply used for these tests (AC wall adapter and USB cable) to see if the issue stems from that.

Despite the file size fluctuations for frame rate adjustments, the general trend for the video resolutions is clear. As a result, we can select 480p as our base resolution. We will run additional tests using a different AC adapter and USB cable to see if the undervoltage error disappears, and see if the frame rate settings affect the file size for our target resolution.

3.0 Solar Charging Unit

The main target of this test was to ensure compatibility between our solar cell and our battery pack which will be used in the camera. The solar cell is what the camera will rely on to remain charged throughout the day and night. Ensuring that the solar panel and charge controller operate as they're supposed to with the battery in paramount to the success of our prototype. While making sure that the connection is functional and charging does take place, it's also important to know the charge time length, and how long the battery will last while being drained under normal circumstances.

3.1 Test Setup

The testing procedure included hooking up a multimeter to the charging port of the battery and measuring the voltage. From there we will hook up the battery to the cell and open a blind to expose the solar panel to light and after 10 minutes we can measure the voltage of the battery once again as well as the ports off of the charge controller to see the changes to the system and the delivering of energy to our system. Prior to setup of the test we soldered a high current rated diode to protect the solar cell from any discharging current generated by the battery.

3.2 Data

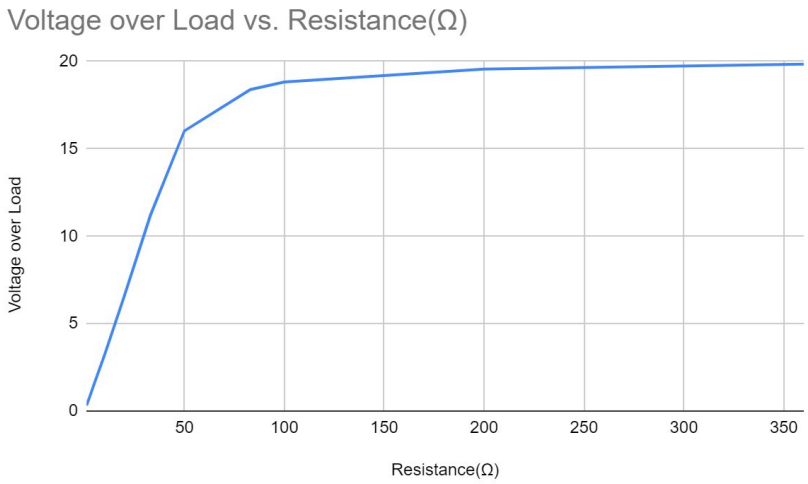

Figure 1: Voltage over Load vs. Resistance

Sunny Day Testing	
Voc = 20.922 V	Is = 0.364 A
Resistance(Ω)	Load Voltage(V)
1000	20.09
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 4: Resistance vs. Load Voltage

Test 1: 25 Min Test	11:45 AM		Test 1: 25 Min Test	2:07 PM
Voc = 21.76	Load = 50 Ω		Voc = 20.922	Load = 50 Ω
VLoad = 18.218	Iload = 0.364		VLoad = 18.431	Iload = 0.368
Vbatt1 = 10.660	Vbatt2 = 10.659		Vbatt1 = 10.657	Vbatt2 = 10.653

Table 5: Voltage Tests

3.3 Conclusion

The final results of the test conducted exposed some glaring issues as well as new direction for the product. The primary takeaway was the fact that charging didn't take place because of an issue with our charge controller. This had lead us to consider a new brand of controller that will allow us to charge the battery and ensure that no damage is done to it. Besides that hiccup, after doing some theoretical calculations based off of actual current recorded we do need a larger solar cell in order to power the system. Due to the overall failure of the system to charge the battery we had to use mostly theoretical numbers to calculate estimates for power consumption and expected longevity of the camera supported by the battery and cell. From these estimates, we found a consumption rate exceeding our recharge rate of the battery using the solar cell. This was disheartening to hear at first, but with some changes to the microprocessor and the size of the cell, we are confident in our ability to overcome this initial issue.

4.0 AWS Plan

Amazon Web Services is a collection of tools and infrastructure used for remote hosting, data management, and processing. For our project, we will be using AWS to host our web-based management platform, store video data for long periods of time, and perform analytics on the data received, such as Automatic License Plate Recognition (ALPR). Each of the services used have variable costs, and are billed on a monthly basis.

The goal of our AWS testing is to develop a budget for our estimated costs of hosting and processing with Amazon Web Services. This is critical, as exceeding the basic storage/processing needs of this project may result in excess costs. At this time, there will be no accurate data to determine the cost of AWS, but we can derive a good estimation to begin using the platform for our prototype.

4.1 Test Setup

The testing procedure began with determination of the general infrastructure requirements of our project, estimation of usage, and finally cost associations. The primary services needed to develop the minimum viable product include AWS Elastic Compute 2 (EC2), S3 storage, and Kinesis data pipeline. Additional services include Glacier, Lambda (impossible to estimate), SES, SNS, and IoT core, as well as the possible additions of Route 52, VPC, and Cloudfront in the future (marked as TDB in **Figure 3**). Using the AWS simple monthly calculator, as well as initial estimates for our needs, we were able to determine a rough budget for the cost of cloud services.

4.2 Data

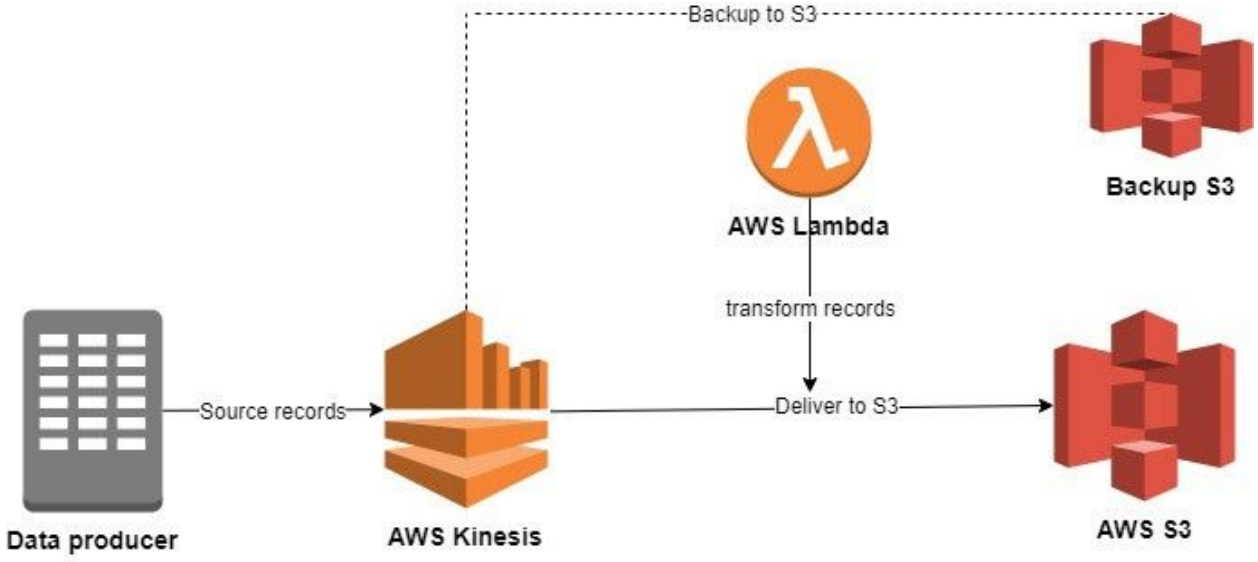

Figure 2: Overview of AWS data input pipeline

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	

Figure 3: Amazon Web Services costs per service, with estimated usage and total price

4.3 Conclusions

Through our testing and estimations, we were able to determine a rough budget of ~$100 per month to operate our project. A major factor in determining this number is the relative size of our project, as well as the use of AWS free tier, which will help reduce some costs. Some services, such as IoT core, were negligible at this scale (for example, IoT connectivity costs $0.04 per year per device). One source of cost reduction we may explore is the use of a smaller instance size, such as a t2.medium, if it can satisfy the needs of our management platform. Additionally, much of the cost of our project is relatively fixed, meaning that if we were to scale to industry levels, costs would reduce over time due to economies of scale.

5.0 Camera Case and Web Application Progress and Updates

While it is not one of the tested categories, this section contains progress updates and concepts for the web application. The web application is being developed with Django, with intentions to use MySQL as the database. The web application has two permission types. A User login portal as well as an administrator login portal. The administer (usually the residential owner) has the ability to add users and cameras to the database. Currently there is a working administrator page with secure login. **Figure 4** below shows the ability to add users and cameras, as well as partial data fields that need to be entered for a camera.

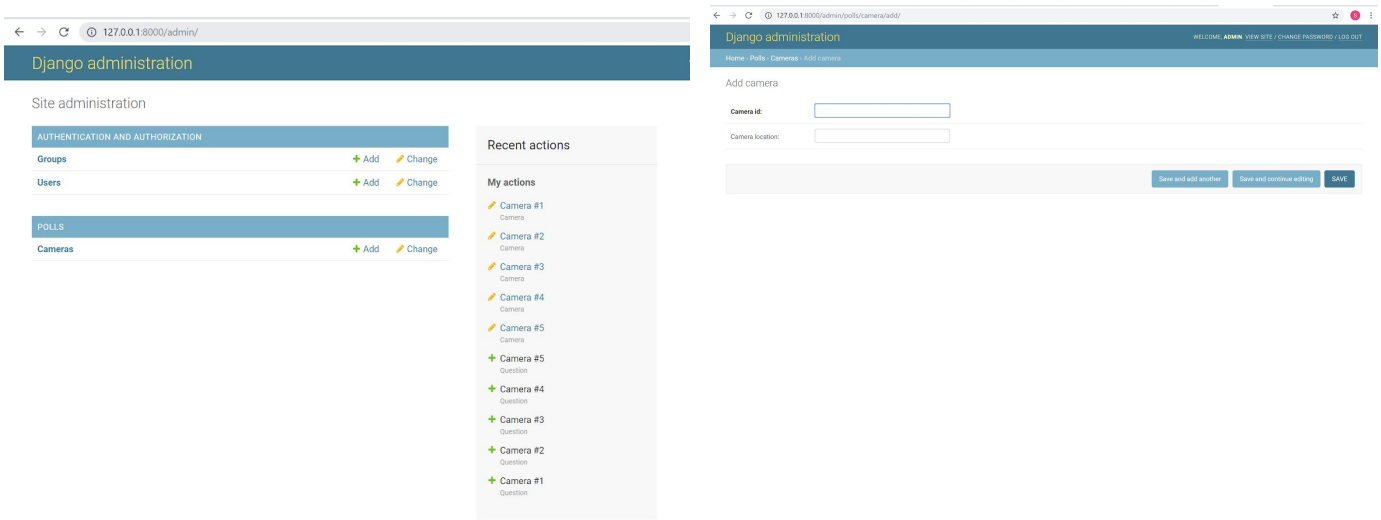

Figure 4: Administrator home page (left) and adding a new camera (right).

The user side of the web application has not yet been developed. On the user side (as well as the admin), the web application should allow someone to view video footage for a certain camera, as well as select what type of footage they want to view, ie. motion detection, license plate recognition, as well as live streaming. They should also be able to access a log that displays which cameras have been accessed by a user and show the date and time, in order to prevent abuse of the system and exessive “spying.” A current database structure design is the have two separate databases. One which contains the information about administrator(s) as well as users within a community. The other is a database of cameras. Each entry in the database for a camera can have different entries under it, such as an entry for motion detected clips, license plate recognition clips, live streaming, and others if necessary. Under each of those entries are video clips sorted by date and time. Currently, the database being used is Sqlite3, which is a locally stored database native to Django. Eventually, once the cloud server is up and running it will be changed to a MySQL database to be hosted on the cloud. In addition, the website is currently running off of a server run from the command line. AWS will take a part in hosting the server instead.

Finally, the team is creating prototype cases for the camera, which will evolve overtime depending on the size of the components used for the final design of the camera. **Figure 5** shows a SolidWorks design that is to be 3D printed as the first case prototype.

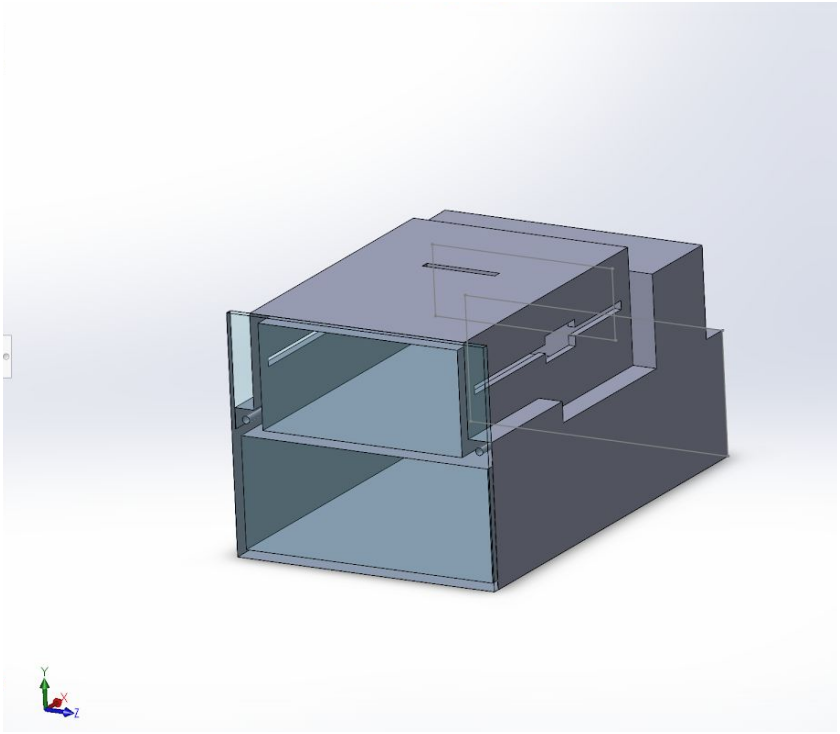

Figure 5: Initial SolidWorks Security Camera Case