Boston University
Electrical and Computer Engineering
EC463 Senior Design Project

1st Prototype Test Plan

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 12th, 2019

1.0 Introduction

For the first prototype, our team will split the overall security camera design into modular components in order to prove that each can work on its own. This modular testing will serve 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 will test three different components. First is the Raspberry Pi 3B+ to make sure it can run motionEyeOS and record videos. Second is the Solar Charging Unit to ensure the solar panel can charge the battery, which will eventually power the Raspberry Pi. Third is the AWS estimated plan cost, which will compare 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 will rely on accessing video feeds from an external device (laptop) that will connect to the motionEyeOS interface using the Raspberry Pi's IP address in a browser.

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

2.1 Measurable Criteria

The tests will be evaluated based on how much storage the recorded videos take up on the pi's micro SD card. As such, measurements will be collected in terms of file size for the videos. In addition, the tests will consider the exact video recording settings in terms of frame rate (frames per second) and resolution (pixels). Measurements will be collected at a 5 minute time interval for each configuration, then linearly scaled to derive an estimate for a 30 day period. This data will provide realistic storage requirements that can then be compared against the theoretical storage requirements that were calculated in the project's parts planning phase.

2.2 Testing Procedure

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

The tests will consider 3 video resolutions (240p, 480p, and 720p) and 5 frame rates (10fps, 15fps, 20fps, 25fps, and 30fps). The Raspberry Pi will record 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 will download the video through the motionEyeOS interface to demonstrate the video can be transferred from the pi. From there, the external device will play back the video in VLC Media Player to ensure the video can be viewed. After confirming the video's viewability, its file size will be 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 will change, some camera settings will remain constant. Specifically, the video file format will be set to H.264 (.mp4) at 50% movie quality as it is a universal file format with a small file size. The camera will 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 will give an upper bound for the necessary video storage.

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

2.3 Test Deliverables

The deliverables for this test will be a table detailing the required video storage for each setting (resolution and frame rate). An optimal video setting can then be obtained from the table, which will then be used for the camera's official settings. The video storage data will also be used for calculating the estimated cost in our AWS cloud server.

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 Measurable Criteria

We can the success of this test by first testing the voltage across the charging port of the battery. Then we connect the battery to the cell and wait a certain amount of time and retest the same port to see if the overall voltage increased thereby proving that some level of charging of the battery did occur. More testing of charging speed will be required, but would need a longer session to prove timing beyond theoretical assumption.

3.2 Testing Procedure

The testing procedure will include 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.

3.3 Test Deliverables

The deliverables of this test will include a fully hooked up system between the battery, charge controller, and solar cell. It will also have a series of theoretical calculations and presumptions made about general power output and charging times.

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 Measurable Criteria

The measurable criteria for AWS cost estimation is the average costs of the services required, as well as the estimated usage of AWS resources. One example will be the cost required for streaming data from the Raspberry Pi to AWS Kinesis. Data costs like these can only be estimated after determining the video frame rate and resolution we will use, which will be determined through the Raspberry Pi test.

4.2 Testing Procedure

The testing procedure will be to collect as much quantitative data as possible on the needs of our project, and using AWS average costs to determine an estimated total monthly cost. One difficulty associated with this procedure is determining the infrastructure we will use in the future. For example, we will need to use AWS Kinesis for streaming video, IoT Core for monitoring external devices, EC2 for web hosting, S3 for storage, Glacier for long-term storage, and Lambda for analytics/video processing. As our prototype evolves, we will have to re-examine the services required, potentially adding more in the future. Part of the testing procedure will also include development of a storage management plan, which will determine the lifecycle of the data we collect.

4.3 Test Deliverables

Test deliverables will include a list of services we believe are necessary for our project and the first version of our AWS storage lifecycle. Additionally, testing will result in the creation of a ‘living’ budget for AWS. This budget will most likely be an Excel document with estimated usage of each service, the average costs of each service, the total for each service on a monthly basis, as well as the total monthly cost. For example, the cost of holding a connection between a device and AWS IoT core is $.08 per million minutes of connectivity, or $0.042 per device per year. Analysis like this is crucial in determining a budget for the operation of the project’s network. This testing will also help us determine if we can reduce costs with AWS free tiers, while maintaining a concept of a commercial budget. Amazon states:

For example, the Free Tier would allow you to run a 50-device workload, where each device:

- Connects for 24 hours a day
- Exchanges 300 messages per day (message size 5 KB or smaller)
- Makes 130 Registry or Device Shadow operations per day (Registry or Device Shadow record size 1 KB or smaller)
- Triggers 150 rule executions per day that invoke one action (processed message size 5 KB or smaller)

If our project can fit within these parameters, we may be able to reduce costs associated with prototype design and testing.