shmPi: Smart Home Monitoring Using the Internet of Things

Computer Science Honors Project (COMP4905)
Jean-Elie Jean-Gilles
100856860
Presented to Professor Michel Barbeau
Carleton University, Ottawa Ontario
December 15, 2017
Table of Contents

Abstract ............................................................................................................................................. 3

1. Introduction .................................................................................................................................. 4
   1.1 Context ...................................................................................................................................... 4
   1.2 Problem Definition .................................................................................................................. 5
   1.3 Result ........................................................................................................................................ 6
   1.4 Motivation .................................................................................................................................. 8
   1.5 Outline ..................................................................................................................................... 9

2 Background Information (Development) ...................................................................................... 9
   2.1 Hardware .................................................................................................................................. 9
   2.2 Software ................................................................................................................................... 10
   2.3 APIs ........................................................................................................................................ 12
       2.3.1 Sonos API ...................................................................................................................... 12
       2.3.2 Flask API ...................................................................................................................... 13
       2.3.3 Lightify API .................................................................................................................. 13
       2.3.4 Twilio API ...................................................................................................................... 14
       2.3.5 Smtplib .......................................................................................................................... 14
       2.3.6 GPIO API ....................................................................................................................... 14
       2.3.7 PiCamera ...................................................................................................................... 16
       2.3.8 Open CV ...................................................................................................................... 16
   2.4 Interoperability ....................................................................................................................... 19

3. Evaluation ..................................................................................................................................... 20
   3.1 Expected results ...................................................................................................................... 20
   3.2 Difficulties/issues/setbacks ..................................................................................................... 23
   3.3 Usability ................................................................................................................................... 27

4. Conclusion .................................................................................................................................... 33
   4.1 Results Received ...................................................................................................................... 33
   4.2 Contrast ................................................................................................................................... 34
   4.3 Future Work ............................................................................................................................. 35
   4.4 Final Conclusion ...................................................................................................................... 36

Acknowledgments ............................................................................................................................ 38

Works Cited ....................................................................................................................................... 39
Abstract

As the internet of things becomes in-vogue and the concern for physical security moves more and more to the forefront of this society, many ideas and solutions to these concerns have begun to surface. One of the biggest concerns regarding this from a software perspective is the lack of cost friendly, open-source solutions, which have been instead substituted for proprietary SaaS systems with little to no control allowed from the end-user. ShmPi attempts to rectify this gap by presenting a proof of concept for a highly scalable smart home monitoring system using the internet of things which can perform at a rate similar to its contemporaries, while remaining open-source and user-friendly.
1. Introduction

1.1 Context

In the past few years, the internet of things, otherwise known as IoT, has become increasingly popular. With a multitude of applications, including but not limited to, automotive, health and manufacturing, IoT has been pegged as the solution for our society’s most complex personal and commercial problems. With that being said, IoT is actively being used to provide an added value to an otherwise considerably saturated market. The home monitoring/alarm industry has long used many different methods to effectively provide solutions to customers, such as intelligent door sensors and premium concierge services, but in the advent of IoT they have begun to take a new direction, intelligent home surveillance. Most of these solutions provide the consumer with a quick and easy way to monitor their homes without ever needing to be in the actual vicinity of it. They also offer additional features such as motion tracking and night vision video capturing.

Companies such as Ring, Nest and many other computer networking companies alike have capitalized on this rise in popularity and have begun to offer their own versions on what would be smart home monitoring. Ring provides users with a single camera, which can be integrated into its full home security suite, for 249.99$CAD\textsuperscript{1}. Nest on the other hand, requires that the user have their in-home learning thermostat as well as a monthly subscription to their

\textsuperscript{1} https://www.bestbuy.ca/en-ca/product/ring-spotlight-wired-outdoor-1080p-ip-camera-black/11428076.aspx?
services in order to be able to use their 250$CAD\textsuperscript{2} camera to its full potential. Other companies such as Netgear offer non-subscription based solutions at 199.99$CAD\textsuperscript{3} for the unit but unfortunately have very little support from consumers\textsuperscript{4}.

Further to solutions offered by these home automation companies, traditional home monitoring companies have also taken the leap into IoT. In Canada, companies such as AlarmForce and Rogers communications offer solutions for home monitoring using smart internet connected cameras, as well as 24/7 concierge services. These solutions, coupled with their traditional implementations of home monitoring, are all offered to end-users at a monthly cost. These costs typically range between 29.99$ a month to 199.99$ a month while also charging set up and hardware fees that can reach over 200$ upfront.\textsuperscript{5}

1.2 Problem Definition

Originally, IoT solutions were meant to be a rather inexpensive solution to otherwise complex issues\textsuperscript{6}. As more and more connected home surveillance companies release their solutions, which rely heavily on the internet of things, there does not seem to be any kind of reduction of cost in terms of benefit for the end-user. These solutions also all rely on proprietary software, disallowing any kind of user intervention in order to either ameliorate or

\textsuperscript{2} https://www.bestbuy.ca/en-ca/product/nest-nest-cam-wi-fi-outdoor-1080p-ip-camera-white-nc2100ef/10450519.aspx?
\textsuperscript{5} https://www.rogers.com/consumer/home-monitoring
\textsuperscript{6} http://www.itu.int/net/pressoffice/press_releases/2016/02.aspx#.WjSNgBiZNE5
adapt the system to their own needs. Due to this, and the original stipulation that IoT was meant to be a less costly alternative to traditional solutions, I believe that an efficient home surveillance system can be achieved using inexpensive commodity hardware. That, coupled with a properly automated program, should be capable of providing the same, if not close to, solution than its paid counterparts. This solution would also address the question of proprietary software by providing an open-source platform, allowing it to be self-evolving. By allowing the system to remain open-source, this could give way to allow users to experiment with the system and adjust anything revolving it to their liking in order to produce the solution that they would not only need, but would want as well.

1.3 Result

My goal was to create an inexpensive, modular and open-source home surveillance system using the Raspberry Pi single board computer, an onboard camera, a reed switch and an alarm program written in the Python programming language. Using widely available third party libraries, otherwise known as API’s, I utilized the Raspberry Pi’s processing power to control various components within a smart home, such as wifi-connected speakers and internet connected lights. I also utilized the devices processing power in order to implement an image processing algorithm, capable of detecting motion within moments, which would then, using the aforementioned API’s, notify the user of an intrusion via picture text message and/or email. This was achieved in order to highlight how inexpensive it could be to implement a full home monitoring system. As a level of automation is implemented using the motion capturing
algorithm, the need for an otherwise costly concierge monitoring system would no longer be valid. Including all primary parts, such as the on-board camera and wires, the Raspberry Pi currently retails for a fraction of the price of the more popular smart home monitoring offerings mentioned above. As this system was intended to be open-source, the intention would be for the system to offer a highly scalable solution in the face of what would otherwise be a market saturated with proprietary solutions. As an open-source system, this solution would allow end-users to have freedom over the functionality of the system, letting them choose what they wish to implement or leave out when rolling out the program. It would also give them the flexibility to adapt the system to their needs and adjust based on computer security concerns while giving any contributor a sense of accountability when programming this system. These factors allow an open-source implementation of smart home monitoring to remain on the bleeding edge of IoT as long as the solution remains viable.

In order to achieve this, quite a bit of work had to be done in order validate the problem (whether or not cost could be scaled down while functionality remained the same). By breaking down the original goal, focussing first on cost, then on functionality, I was able to better understand the problem at hand through brainstorming, research and application. Following these phases, I had to go through multiple stages of testing in order to confirm whether or not my result was what was desired or if more work was required in order to achieve my desired end-result. The chart below shows the issues encountered, along with their specific goals.

<table>
<thead>
<tr>
<th>Issues</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brainstorm</td>
<td>Break up the individual goals, and understand the problem</td>
</tr>
</tbody>
</table>
1.4 Motivation

My initial motivation for this project came from a problem I experienced at one point last year. I had been in the market to purchase home insurance and was told that a significant reduction in cost would be levied if I in fact did have an alarm system. As I was moving into a new place and was hoping to reduce as many costs as possible, I took the bait and began looking for reputable alarm companies who could provide their services to me. Upon a closer look, I noticed that all of the solutions currently available, whether it be from an alarm company or from a home automation platform, were quite costly if I were to procure what I desired to have in my new home. While looking at solutions like Nest and Rogers Smart Home Monitoring, I noticed that the extra cost I was looking to avoid when applying for home insurance would have just been transferred to the cost I would have paid for these solutions. Due to this, I set out to research possible alternatives, and whether or not they would be accepted by my insurance provider. Through this research, I concluded that although risky, I could write an open-sourced system myself that would not only provide a similar solution as to what these more proprietary security companies have, but also allow for a level of modularity, which would help this aforementioned system scale to any level needed as long as the base hardware is available.
1.5 Outline

The report in its entirety covers a vast range of concepts. Section 2 provides background information about the Raspberry Pi’s hardware, the software used to create this system and how they interface with specific API’s in order to address certain requirements. Some basic circuit principles are also provided in order to better showcase some of the basic functionalities of the system. Section 3 evaluates my final system and gives insight on the problems I faced while developing our solution, the steps I took to overcome them and the obtained result. Lastly, I conclude with my final thoughts and future work in section 5.

2 Background Information (Development)

2.1 Hardware

At the core of the system, all computations and processing is completed on a Raspberry Pi single board computer. The Raspberry Pi is a small barebones computer that is about the size of a credit card. The device possesses 4 on-board USB ports along with a built-in network card for everyday use. Despite that, fast access to the computers Broadcom BCM2836 processor is available through its on-board general purpose input output (GPIO) pins. The device also allows for fast camera access through a dedicated serial port. The camera used in this system is the Raspberry Pi Camera Module V2. The camera possesses an 8-megapixel sensor and is capable of recording video with a resolution up to 1080p at 30 frames per second.
Lastly, a magnetic reed switch is connected to the GPIO pins to be used as a door sensor, by connecting to the GPIO headers, this overrides the connection speeds of USB or any other serial connection.

In order to achieve a high level of modularity, all that is necessary in order to maintain the baseline of this system is the Raspberry Pi, an onboard camera, and a reed switch. All other functionality can be omitted if the hardware is unavailable. As a proof of concept for my testbed, I made use of wifi-enabled speakers, as well as internet connected lightbulbs in order to showcase some other functionalities of my system.

2.2 Software

Using the Raspberry Pi’s natively supported operating system, Raspbian, I utilized specific programming languages and frameworks in order to complete this system. Raspbian is the Raspberry Pi’s native Linux distribution. The OS possesses the same capabilities as its other Debian based counterparts (such as aptitude for package management, a standard file browser and web browser). What sets Raspbian apart from other Linux distros is its enhanced features made to interface specifically with the Raspberry Pi. As the Raspberry Pi’s processor is ARM based, its reduced instruction set computing based processor requires a specific type of OS architecture, one that is not available in most distributions as they are more commonly built for complex instruction set computing processors such as x86 or x64 processors. Using
an ARM based system allows for better power consumption\textsuperscript{7}, while providing a smaller form factor in order to maintain the compactness of this project.

The programs written within Raspbian for this project are written in Python. Python is considered a high-level programming language used for general purpose programming. As an object-orientated programming language, Python code is interpreted as a script, with indentation being used in lieu of curly brackets for function definition and statement declarations. This allows for easily readable code and clear syntax. Difficulties may be encountered when attempting to use Python for parallel or concurrent programming due to a Global Interpreter Lock that prevents multiple threads of Python code from running simultaneously\textsuperscript{8}. This can be solved by taking a multi-process approach to programming software when using python. By doing so, this allows Python to be used as a very dynamic programming language on all scales.

In this project, Hypertext Markup Language (HTML) is also used. HTML is the standard language of choice when creating webpages. As this projects solution relies on a simple web UI, HTML was incorporated in order to interface with the systems back end. This is done using a specific API, provided as an enhancement to Python. By doing so, the system is able to render webpages written in HTML that can interact with the back end written in Python.

\textsuperscript{7} http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/DEN0024A_v8_architecture_PG.pdf
2.3 APIs

In order to mesh the hardware and software efficiently, the use of open-source, widely available application programming interfaces, or APIs, were needed. These API’s provided multiple different sets of subroutines and functions that allowed for increased functionality and interoperability within my system. By providing a layer of abstraction, the APIs used were able to be implemented seamlessly and effectively without having to loop back to any other program to complete a task. The use of API’s also allows for a high level of modularity within the system in order to pick and choose which API’s should be active within the system, depending on whether or not the necessary hardware is available. For my solution, I made use of the Sonos API, the Lightify API, the GPIO library and the PiCamera library in order to interface and interact with real world components. I also used the Twilio API, the SMTPlib library, the Flask API and the OpenCV library in order to make use of the inter-system capabilities of my solution. Below are the detailed descriptions of each API and their specific use.

2.3.1 Sonos API

Sonos is a brand of internet connected high fidelity wireless speakers. The speakers are connected to a home network through either a wireless or wired Ethernet connection, thus having an accessible IP assigned to each speaker. The system itself is capable of reading music files from a multitude of sources, whether it be webpages, network attached storage, streaming services or devices who may have the Sonos application loaded onto it. This allows for the speaker to be a highly versatile solution when attempting to use it for alerts.
SoCo is a high-level open-source Python library that can be used to control any Sonos speaker on a network. The library can either be configured to automatically discover a Sonos system on the network, or can be manually configured by specifying the IP of a given Sonos speaker. SoCo can then be configured to select which source to use to play audio. In this case, SoCo is configured using manual configuration for speaker discovery and a URL is specified to be played when needed.

2.3.2 Flask API

Flask is a lightweight micro-framework created for simple Python web application development. The framework relies on HTML, Javascript, as well as simple route tags in order to render and interact with webpages. Commands are sent through Python scripts via Ajax and handled at the HTML layer when rendering webpages, this abstraction allows for very simple implementation of the framework into projects.

2.3.3 Lightify API

Lightify is a platform of internet connected smart home devices made by Osram. The platform provides high scalability due to it being modular by design when implementing components such as smart lighting and intelligent power sockets. All components are routed through a central physical hub, programmed to interface with the Lightify mobile application. The API available for use in Python provides a way to discover a Lightify hub, including all of its components, by providing the IP for the hub. Once configured, the API can be used to
control the lights and sockets, set dimmers and timers, while monitoring the status of the components active of the network.

2.3.4 Twilio API

Twilio is a lightweight cloud platform as a service that provides a web service API, allowing for applications to be able to send SMS/MMS messages and make phone calls using the service. Messages are sent using HTTP post requests, which are routed to the services account using a unique subscriber ID and authentication token. Once routed, the service then communicates with the public switched telephone network in order to send an SMS or MMS to the designated receiver.

2.3.5 Smtplib

In order to send emails, the Smtplib library is used, which provides an encapsulated wrapper to an SMTP connection client. Emails are sent by first specifying and connecting the client to the SMTP server of the email provider, then creating the email content by specifying its multipurpose internet mail extensions, or MIME. MIME is the standard that provides a format for all emails being sent over SMTP. Using the library’s built-in MIMEMultipart() function, an array is created to specify the specific portions of an email, such as the subject, receiver, date and the emails body. Once the information is completed, the sendMail() function is used to send the array created by MIMEMultipart(), the receivers email, as well as the senders information.

2.3.6 GPIO API
The Raspberry Pi is capable of receiving physical input and output using its built-in general purpose input output (GPIO) pins. Using these pins, interfacing can be done with the device for a multitude of things. The Raspberry Pi foundation offers a simple yet intuitive library for Python in order to write applications to communicate with these pins for specific uses.

As figure 2 illustrates above, each pin has its own dedicated function to either control, power or ground a device connected to the system. The figure above utilizes what is called the physical BOARD numbering system. As the pins can be numbered in different ways depending on the wiring framework used, the pin numbers must be specified as well as which framework used in order to avoid any ambiguity when wiring a system. For my solution, I chose to use BOARD pin 23 to control the input and pin Y for ground. Once connected, a circuit is created using a magnetic reed switch as an interrupt, as shown in figure 3. When programming the

Figure 2: Raspberry Pi GPIO pinout diagram

Figure 3: Simple schematic of GPIO pinout.
reed switch, a Python function is used to import the GPIO header pins, and check if the switch is open or closed. Checking pin 23 in this implementation will notify the system on if the pin is true, meaning if the switch is open, or false, meaning the switch is closed. After verifying the switches activity, the program can then make its decision based on the case it receives from the switch.

2.3.7 PiCamera

The Raspberry Pi allows images to be captured through any USB camera but as well from its own onboard camera interface. As an addition to the system, a camera can be connected via a ribbon cable directly to the devices onboard camera serial interface port to provide fast and efficient digital stills to be sent to the device for processing. Images can then be captured in Python using the PiCamera library. This library provides developers with a multitude of resources to be able to interface the camera with the Raspberry Pi using Python. Using functions that allow for recording, such as start_recording() and stop_recording(), video capturing can be done using the now onboard camera. This can be useful to record to a file if specified by the user within the program. The PiCamera library also allows for continuous image capturing using the capture_continuous() function. Frames can be taken constantly from the camera and processed through the device in order to implement the specific use case of this solution using other libraries.

2.3.8 Open CV

The Open Source Computer Vision Library (OpenCV) is an open source library used for computer vision and machine learning. It can be implemented for a multitude of use cases, such as image manipulation, facial recognition, augmented reality and motion tracking. Using
both classic and state-of-the-art computer vision and machine learning functions, I created an algorithm that can be used to detect motion from any given camera when implemented while capturing from the device. The algorithm works as follows:

1. For each frame captured by the camera
2. Convert the frame to greyscale
3. Perform a Gaussian blur on the frame with a specific standard deviation (blur size)
4. Set this initial frame as the background model if no background model has been set (original image to be compared)
5. Verify the frame delta between the initial frame and the following frame
6. Create a threshold delta between the initial frame and the following frame
7. Dilate the threshold frame
8. Find the contours in this threshold
9. Verify if the contours of the following frame are the same as the initial frame
10. If the contours are the same
11. Continue to the next frame
12. Else
13. Trigger the alarm

In this algorithm, many functions specific to the OpenCV library are used. After converting the image to a greyscale image, a Gaussian Blur is applied. A Gaussian Blur is an image processing method that smoothens (or blurs) an image using a Gaussian function. This is done in an effort to reduce any kind of noise surrounding the image when reducing it to simple shapes using a specific standard deviation, or blur size, as opposed to the well-defined images originally produced by the camera.
The delta between both frames is verified by calculating the absolute difference of each matrix element produced by the image frame and the initial image (as each frame is essentially a matrix of pixels). Once the delta is computed, an image threshold is created from the frame delta using the built-in function threshold(). Threshold() takes the image, verifies each pixel to see if its below or above the specified value passed into it, then changes the specific value of that pixel to a pre-set value (in this case, 255, or white). Once the threshold is made, the new image is dilated in order to increase the object area in an effort to avoid false-positives.

Once the threshold is created and dilated, findContours() is used to generate the contours of the dilated image. When the contours are generated, they are then iterated through.
and compared to a minimum area of movement. If the contours are the same, then continue to monitor the next frame, otherwise the alarm is triggered by changing a Boolean value to true.

2.4 Interoperability

Altogether, the API’s interact with the software layer in order to produce the final solution. Through multiple Python scripts, the system interacts with the Raspberry Pi hardware, its camera, as well as the Sonos speakers and the Lightify hub in order to provide a full security system with the Raspberry Pi at the helm. A Python script called __init__.py initializes a webpage using the Flask library visible to the end-user via any web browser when entering the address. From there the end-user can then control the alarm system, allowing them to arm it, whether it be locally or remotely, and also enable video capturing using the PiCamera library. Once armed, the script executes a new process by launching the shmPi.py Python script. This script is where the majority of the backend logic is held for the system. Once launched, the shmPi.py script will instantiate and activate the necessary GPIO pins using the GPIO library to be able to monitor the reed switch status, while also activating the onboard camera in order to begin capturing stills once armed. As each frame is captured, the system checks whether or not the reed switch has been tripped, if not it will move onto checking the frames for motion using the algorithm created with OpenCV. If motion is detected or the reed switch state is changed, the system will then break from its loop and go into an alert mode.

Depending on what’s configured, the system will check a configuration file to see what it can and cannot do. Once triggered, a rolling capture begins using the PiCamera, saving all footage to the device. If Sonos is activated, the system will use the SoCo API to sound a loud
alarm over the connected speakers. If Lightify is activated, the system will interact with the hub to turn on the lights, allowing the camera to have better vision and higher fidelity recording. If a phone number was supplied, the system will then use the Twilio API to send a text message to the end-user with an attached image of what caused the system to be triggered. Lastly, if an email address was given, then the system will contact the end-user via email with a notification including the attached image of what activated the alarm. Following this alert action, the system will continue to record video of the environment until the user stops the recording or the filesystem begins to run out of space.

With the API’s and the baseline code working in unison, the system is capable of detecting motion and breaches of environment and notify the end-user of such things.

3. Evaluation

3.1 Expected results

Over the course of the implementation of this project, despite a handful of difficulties that plagued me in the early stages, I never forgot the main goal. I managed to build a simple circuit in order to programatically detect intrusions by physical intervention when armed, while implementing the majority of my design goals and requirements. The expected end result of this implementation process was intended to be a lightweight, highly modular smart home monitoring system. This system would utilize inexpensive, widely available hardware and specific open-source software in order to achieve the goals I had originally set out. The system
is to be easy to use, with a simple web interface, allowing for easy configuration, which should be accessible to users from all spectrums.

The goal was to build this system as an alternative to widely available solutions from companies such as Nest, Rogers and Ring. Due to this, close attention was paid to some details when completing the implementation. Alternative smart home monitoring solutions such as Rogers Smart Home Monitoring and Nest offer a rather modular experience, a methodology I sought to replicate in my project. The service provided by Rogers Communications, allows users to pick and choose which features they would like to have in their home automation system. At the base, users pay 19.99$ a month for a self-monitored service, meaning they do not receive any 24/7 security alerts other than the ones produced by the system. Along with the monthly fees, comes a one-time equipment purchase price which can cost up to 150$ in a barebones configuration. In a typical configuration of Rogers’ solution, a user must purchase many motion sensors, indoor and outdoor motion sensor equipped cameras if needed, as well as a proprietary hub, which is necessary in order for the system to work. Users can then take advantage of the unmonitored solution, which is provided under a 2-year contract, but only
provides them with live video streaming. In order to receive the 24/7 security service, users must pay an extra 20$ for the next best package, which doesn’t provide a recording camera out of the box but can be added to the account for 170$ as well as a monthly surcharge of 7$ for the live video streaming service. After setup fees, users typically spend close to 450$ on equipment, and 45$-65$ a month on the service. These monthly fees include live video streaming, sensor monitoring and a 24/7 security service. Nest also follows a similar module based pricing model. Nest, originally being a learning thermostat home automation solution, offers an internet connected camera for 250$. This camera provides users with a live video stream as well as motion detection notifications while being integrated into Nest’s suite of home monitoring peripherals. Despite that, in order to make proper use of the camera, Nest requires that the data be stored on their servers, which comes at a charge of about 10$ a month. Once stored, Nest makes use of learning motion detection algorithms in order to provide customers with a more accurate security update if ever the system is triggered. Due to this, Nest’s camera is proprietary by nature, only allowing the company to have access to it in order to make ameliorations without any intervention from the end-user.

As both solutions mentioned possessed a rather modular implementation, I chose to do the same with my system. The main difference is that despite being modular, my system would be modular by desire rather by necessity. Systems by Nest and/or Rogers can be considered modular by necessity as in order to achieve the original objective of the system, users must purchase additional services and hardware. These additional add-ons, available only from the vendor, disallow users from having any real control of their systems if they were to want to
add increased functionality or remove functionality based on their needs. Although some additional services prove to be very useful (such as 24/7 security concierge services) others that would have otherwise been thought to be included in the cost end up coming at a hefty premium (such as video storage and live streaming). Although my solution does not provide users with any kind of live security concierge, things such as storage and streaming remain in the control of the end user. These options are included as the base configuration of my system. Users can then decide to add any extra functionality as they desire. These functionalities can be added and configured through the web application launched when running the security program. Once configured, these additions are recorded in a configuration file to provide persistent storage of the configuration. Due to this level of user-centric modularity, my system is capable to make use of many other open source libraries if one were to want to add any other kind of functionality of their own, something that is not available when using proprietary software from the likes of Rogers or Nest.

3.2 Difficulties/issues/setbacks

When implementing my solution, many things were taken into account. In an effort to mitigate any kind of delay in the design and development process, I ensured to research all aspects of the project from the hardware to be used to the specific programming languages and APIs to be implemented. Despite this, there were still some difficulties that needed to be overcome throughout the course of this project in order to complete it in its entirety.

Although being familiar with objective-oriented programming as well as scripting, before this project I possessed little to no knowledge regarding my programming language of
choice, Python. Originally planned to be written as a Java web application, as the Raspberry Pi provided a very intuitive and easy to use library for interacting with the device using Python, I felt obliged to attempt to complete the project in that language. Due to my original lack of knowledge, quite some time was spent learning about the language and some of its limitations when dealing with a project as specific as mine. For one, it has been stipulated that concurrent programming in Python is rather difficult to implement. This is due to what was mentioned as a Global Interpreter Lock (GIL). This lock ultimately prevents multiple threads of Python code from running simultaneously. Because of this limitation, threading in Python does not necessarily behave in the same manner as other programming languages I would’ve otherwise been knowledgeable in. In an effort to avoid any possible ambiguities when writing this project due to this limitation, I chose to take another approach that would still provide me with the desired results. By opting to launch separate processes as opposed to multiple threads, I was able to avoid having to implement threads that may or may not work to my liking in Python. The web UI executes a process for the backend security system, and kills the process on shutdown. This allows for monitoring to occur at all times while the program is running.

Another limitation was identified due to a lack of knowledge. When originally planning this project, a content rich web user interface had been thought of as the solution of choice for user interaction. Despite this being one of the design requirements, it also required extensive research into web based programming languages that would provide such an experience. As a domino effect of deciding to implement the project using Python, a language that itself required some learning on my part, the design goal of implementing a rich user interface had become
increasingly difficult to meet. Due to this, I opted for a more barebones user interface, that mainly focussed on what was necessary for the end user, starting the alarm system, stopping it, and configuring it. I also managed to add a simple video stream to the UI so that users can log on and see what is being captured in real time.

Although very basic, the final web user interface seemed to meet the majority of my design goals. The interface allowed the system to interact with the back end through the web layer, while also providing users with a real-time view of their environment.

Lastly, there were some issues during the implementation of some APIs over the course of the project. One major design goal that I had set out for was the ability to notify users over text message of a breach in the security system. This was achieved using the Twilio API, which
provided me with a library to make RESTful calls to their service, which would then send a
text message to any designated cellphone number. I had intended to make use of their MMS
service in order to send to the user an image of the event that caused the trigger. I had
successfully implemented the simple SMS capabilities to send text messages but when
attempting to use the MMS functionality, I was met with some issues. As Twilio’s API requires
images to be stored on the web in order for the service to make an HTTP request to retrieve
the image, images that were taken and stored locally on the device were unable to be accessed
by the API’s built-in media_url parameter. As a result, I was unable to attach images directly
to my API call to send to the end user, causing me to almost have to stray away from this major
design goal. At that point, I managed to brainstorm two options, one of which would cause a
fee to be incurred if this project were to ever scale to a larger platform, the other being a
solution that would cause more strain to be put on processing power. The first option was to
rely on cloud services to store images of the intrusions as they were being taken. By leveraging
services such as Amazon AWS S3 or personal storage services such as Dropbox, the Twilio
API would then be able to access the storage through a web address in order to send the
necessary stored image to the end-user. The only setback is that by using any of these services,
a monthly cost would then be incurred. Although the costs would be minute in the case of my
testbed (Amazon S3 services provides the first 50GB of storage at a measly 1.25$ per month),
if this project were to scale to the likes of companies such as Nest or Ring, the fees involved
would definitely surge to a price that would then set my project well outside of my design goal
of providing users with an inexpensive alternative. I would also then have to rely on proprietary
storage solutions from a third party, at which point my goal of having a fully open-source
The solution will then unfortunately not be met. The second option revolved around keeping all of the data stored in-house, but accessing it externally using a loopback mechanism. By implementing a simple webserver, such as Apache 2.4.10, I would be able to feed the images to Twilio using the server running on my public IP. This would then allow Twilio to send the image to the end user using its URL, avoiding any issues that could have been faced when trying to access the images locally. The second option seemed a lot more interesting, as it required no extra services to be procured, while delivering the same solution. The only concern that came to mind was regarding resources. As the device running all of this software is rather barebones, maintaining efficiency when dealing with computations would prove to be a primordial factor to the success of my system.

<table>
<thead>
<tr>
<th>JEAANN@shmp1:~</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>862</td>
<td>0.0</td>
<td>0.4</td>
<td>7120</td>
</tr>
<tr>
<td>www-data</td>
<td>864</td>
<td>0.0</td>
<td>0.3</td>
<td>229728</td>
</tr>
<tr>
<td>www-data</td>
<td>865</td>
<td>0.0</td>
<td>0.3</td>
<td>229728</td>
</tr>
</tbody>
</table>

*Figure 8: Printout of system resources allocated for Apache*

Luckily, on average the Apache service took only 10mb of memory to run in the background at all times. This low cost of resources is what drove me to choose it as an alternative, allowing Twilio to access the system and to feed the images back to the user after any event.

### 3.3 Usability

Following the completion of the first iteration of my system, I then began to initiate tests to confirm and validate specific components. Components such as image capturing speed in the event of a breach, time between breach and end user notification as well as add-on interaction were all taken into account when testing. In order to streamline and avoid redundancy during testing, these performance tests were done during the usability tests.
Usability tests were performed on the system using a sample space of 4 users, each varying in technical ability. The users were chosen specifically to showcase their different levels of technical knowledge, in order to remain consistent to the user-base of the more widely available alternatives. The users were to complete a specific set of scenarios, in a specific order, then they were asked to complete a short questionnaire and to provide comments. The scenarios were as follows:

Scenario 1: Testing Reed Switch Sensor

1. Set your phone number using the configuration utility
2. Launch the alarm program (__init__.py)
3. Leave the area occupied by the system (My apartment)
4. Arm the system using your mobile device and the online portal
5. Once confirmed that the system is armed, open the door the sensor is connected to

Scenario 2: Testing Motion Sensor Camera

1. Set your phone number using the configuration utility
2. Launch the alarm program (__init__.py)
3. While not in-front of the system, arm the alarm
4. Once confirmed that the system is armed, attempt to put yourself in-front of the on-board camera sensors field of view

Both scenarios were meant to be completed independently in order to provide as consistent of data as possible. As both tests were relatively simple, all 4 subjects were able to complete the tests as specified with little to no intervention by me.
During the execution of the scenarios, I was able to pinpoint the responsiveness of the system by timing the reactions based on the user’s actions. These tests allowed me to deduce that the system was consistently performing all tasks during all tests in the same amount of time. On average, it took the system 7.4 seconds to detect a breach, this was determined by timing the users breach and the time it took for the system to make a call to the SoCo API. As this API is the first action the system takes in my implementation after a breach, it also tests the add-on interaction component of my test bed, which in itself fared very well as well considering the fact that it also only took the Lightify API 7.6 seconds to make its calls following a breach. The system took minimal, but notable amount of time to send a text message to the user following an intrusion detection. The users received a notification with picture message about 45 seconds following the breach. This could be due to the time it takes to make an HTTP call to Twilio, have Twilio loop back and download the media from our local server, then send that media to the end-user as a picture message. These time-consuming calls are unfortunately the by-product of the solution implemented in order to allow for multimedia messaging when an intrusion is detected. Such a delay in notification also lands the system to trail behind the average time taken by more traditional systems such as Alarmforce to set up a two way communication between the intruded location and their security desk.

Once the scenarios were completed, I requested that the users complete a short questionnaire, in order to provide me with background information in order to build personas. The questions
were rather basic in nature but related directly to the scenarios. As the users completed the set of scenarios, I had them answer the following questions:

1. On a scale of 1 to 5, How familiar are you with smart home automation solutions like Nest, Rogers SHM and Ring?
2. On a scale of 1 to 5, How easy was it to navigate through the online portal?
3. On a scale of 1 to 5, How easy was it to configure the system?
4. On a scale of 1 to 5, How easy was it to arm/disarm the system?

Questions were provided to users on a Likert scale going from 1 to 5, 1 meaning “not at all” and 5 meaning “very much”.

The results were rather consistent with what I expected from the system implemented. Based on the survey data, the users were all not necessarily experts regarding smart home monitoring but found this solution easy to use. Due to this I had planned to ask the users to provide me
with more feedback regarding what they liked and what they thought could be improved. The users pro’s almost all varied, each pointing out various different aspects of the system that they found useful. User 1 mentioned that they liked the portability, due to its small form factor and low requirement of wiring while also enjoying the use of picture as they can receive them without needing internet. User 2 liked the fact that a picture is taken in the event of breach as well as the use of the intelligent lights and audible alarm add-ons. User 3 enjoyed how easy the system was to use while user 4 also enjoyed the portability of the system. The positive feedback was different in terms of what each user liked but it was quite interesting what occurred when asked what could be ameliorated. The users all had concerns regarding the user interface. Despite all mentioning that it was simple and easy to use, they would have liked for it to be a little more appealing to the end user. This echoed my concerns from my difficulties mentioned, validating that some work should have been put into the user interface. Other than the UI concerns, each user also had their own set of comments regarding the system and what they believe could be done better. User 1 mentioned that they would've liked for the system to be able to control a mesh of devices in order to monitor bigger spaces while also allowing image sensor to detect and omit small intrusions like domestic pets. User 2 had mentioned that when receiving their notification, the image taken was of their back. This was due to where the system was mounted, but as the system was built to be highly portable, this would not be a constant concern. User 3 mentioned that they would've liked to have the system record sound as well as video, while also having the possibility of adding more cameras as well as a stay function similar to more traditional systems like Rogers SHM & ADT. Lastly user 4 noticed in their scenario that the system detected shadows when intrusions occurred, causing an empty
image to be sent to user. All of these suggestions were very valid and taken into consideration through-out the final stages of the project, where refinements and contrasts were done based on the feedback received.

Once all the surveys were completed and compiled, the data was used to create user personas. These personas helped map out where specific refinements can be implemented based on priorities and user types. Based on the 4 surveys, 3 personas were created. The personas were as follows:

**Persona 1: UserA**

UserA is a rather tech savvy individual who benefits heavily from smart home monitoring as it provides them with a way to access and monitor their home at any time. As they are never sure as to when they’ll be out or at home, they require a solution that allows them to configure the system anywhere that they deem fit. They are knowledgeable as to what’s available in the market but would prefer to have full control over their system.

**Persona 2: UserB**

UserB is a somewhat tech savvy individual who currently has a more traditional alarm system set up but is aware of the new solutions being available. As they have a specific schedule, they know exactly when they’ll need to have access to their system, but they would like to make sure that the interaction with the system is as seamless and easy as possible.

**Persona 3: UserC**
UserC is not nearly as tech savvy as the others, and therefore has no knowledge of neither smart home monitoring solutions, nor the more traditional contemporaries. They would require a home monitoring system in order to ensure that their home is secure whenever they are not there as they are never too sure when that will be. They would like a system that is capable of being accessible whenever they need it, while being able to keep in mind any kind of negligible intrusion such as a pet or trusted individual entering their space.

4. Conclusion

4.1 Results Received

As there was a slight time constraint following testing, there was not much more time to incorporate the changes suggested by the participants. Only slight changes were done to the system, in order to refine the backend components of the system and to allow for better future scalability. The final system consists of 2 web servers, one for hosting the web application and another for storing images and data, as well as a back end program that controls all of the system logic, a utility script that possesses all of the notification data and a configuration JSON file. These components all work together by being launched by an initializing script, which launches a web server for the end user system. The backend security application is launched in another process using the web application and stores all pertinent data to the project in the systems JSON configuration file. Once any breach is detected, the system then starts to record the ensuing events while also sending to the end-user an image of what caused the breach via picture message and email. Users can specify how long they would like the system to record
the event based on how much space they may have available on their system. Once recorded, the users can log onto the server storing the intrusion images to also view the video of the intrusion, allowing them to save the video and provide it to the authorities if needed.

4.2 Contrast

Based on what I intended to achieve and what the actual results were, there were some differences which would need to be pointed out as they may have affected the overall design goals of my system. The first and main difference between my solution and what was originally expected is the web UI. When originally outlining my design goals, I had planned for a rich user interface, capable of linking the end user to the back-end security system. This UI was to serve as a dashboard to provide users with system status, configuration and commands to the alarm system. What was produced was unfortunately not that. The final web user interface seemed to serve more as a visual proof of concept to bridge the web application to the back end as opposed to the rich user-centric interface that was originally planned. Despite having not met one of my design goals, I still managed to meet a major one. When planning this project out originally, I had aimed to produce a scalable, highly modular open-source alternative to what was an otherwise fee based, modular yet proprietary marketplace. I managed to keep this design goal in mind throughout the entirety of this project, while striving to maintain it even during times when met with difficulties. Proprietary solutions from the likes of companies like Nest offer added value, such as cameras that can operate in any element while being able to continue learning and evolving its motion capture algorithms (through updates supplied solely by the vendor). This in itself could make one consider the Nest camera
a superior product when compared to my system. This is due to the amount of research and development put into their product as well as many other things, including backing capital. Although my system would be capable of withstanding the environmental elements with the right enclosure, its built-in motion capturing algorithm has not been tested outdoors. This is where it differs from other commercial solutions as its ability to be refined and tweaked by anyone with a level of background knowledge due to its open-source platform helps users regain a level of control over their system. Due to this, users can modify and update the system as they deem fit for their implementation, omitting or adding to it as they please. Based on testing, the system also meets other design goals set out at the beginning of the project to a certain degree. I had hoped for the system to be capable of notifying users of any intrusion in a fast and timely manner. Based on my tests, showing how fast the participants received a notification, I was able to see that despite not being as fast as a two-way communication is set up by a major company such as Alarmforce, my system fared rather well by being able to detect an intrusion and notify the user within 45 seconds.

4.3 Future Work

Going forward, if I were to choose to continue with this project, I will definitely consider the concerns raised during the evaluations, as well as my own enhancements that I’d like to add. The first major change, taken directly from the difficulties and evaluations, would be the web interface. As it is extremely barebones and provides the user with no real visually appealing properties, I believe that this would be the first thing to be modified and enhanced if I were to revisit this project. By providing a more appealing UI, the users described within my personas would have a good portion of their requirements met. Secondly, I would look into the
limitations and ameliorations of my current camera. The Raspberry Pi camera offers a relatively good resolution, but with newer solutions on the rise, offering night vision as well as even higher resolutions, this would help with further developing my system to offer more than at its base. A higher picture resolution would help with things like better motion detection but would also allow to implement more enhanced functionalities like facial recognition or detection omission as mentioned in the evaluations. A better physical camera would also put the system on par with the more commercial solutions such as Nest and Ring, as their solutions already offer high definition video with certain degrees of computer vision intelligence, but at a fee. Lastly, in an effort to avoid dealing with storage limits and siphoning processor power, I would like to implement the system so that it relies on network attached storage as opposed to its own back end web server. By doing so, I would be able to take the load of this server off of the device, although being small, it could be put to good use if developing more intelligent motion detection algorithms. The images could then be sent to a remote, in-house server that could also still be accessible anywhere around the world, but could also offer some sort of panic protection, in the event that any would be intruders spot the smart home monitoring device and decide to tamper with it. These are just a few ways I could make use of future work on this project, but I’m sure that as it would go by, more ideas would come.

4.4 Final Conclusion

Overall, despite a few minor setbacks, the process of building a smart home monitoring device, making use of the internet of things in an effort to not only be modular, but scalable and open-source, was in my opinion, a success. I managed to meet the majority of my design goals, by
developing a system whom at its base provides similar functionality to that of other widely available smart home automation and security solutions. From researching the solution, to actually implementing and conducting the surveys, everything step I took in this project was primordial to the successful completion of the solution. I managed to apply important concepts learned throughout my degree to a fairly new area of computer science, providing a solution to those who would like to take advantage of these new technological advancements to rely on more open-source systems.

This paper described shmPi, a system that utilizes inexpensive IoT enable components in order to provide a scalable, highly modular open-source solution to home monitoring and automation. ShmPi is incrementally scalable, allowing users to scale the system based on their needs through open-source modification and modular additions of hardware. This project serves as a proof of concept that alternative, open-source and free of charge home monitoring solutions can be put in place that provide similar functionality as more widely available commercial proprietary systems do. This system could be useful in many cases whether it be simple home monitoring or scaling to monitor major spaces.
Acknowledgments

I would like to thank Professor Michel Barbeau for agreeing to supervise me this term for this honors project. His insight and suggestions allowed me to be able to confidently and efficiently
complete this project. I would also like to thank the Raspberry Pi foundation, as their website, laden with helpful discussions and tutorials, aided me when I was in search of solutions regarding configuring the device and parsing information from the inputs received.

Works Cited

Accessed: November 24, 2017

https://docs.python.org/2/library/smtplib.html#module-smtplib

https://www.osram.com/cb/lightify/index.jsp


http://opencvpython.blogspot.ca/2012/06/hi-this-article-is-tutorial-which-try.html

https://www.learnopencv.com/opencv-threshold-python-cpp/

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

[9] Image Thresholding (n.d.) Retrieved October 24, 2017,
https://docs.opencv.org/3.3.1/d7/d4d/tutorial_py_thresholding.html


https://soco.readthedocs.io/en/latest/#


https://www.alarmforce.com/?utm_source=google&utm_medium=cpc&utm_campaign=afcaon&type=CAONBR&gclid=CIriskoDOjdgCFQ6ywZUAC5w&gclsrc=ds

https://raspberrypi.stackexchange.com/questions/34947/reed-switch-wiring