Technical Report

ABSTRACT

The purpose of this project was to investigate, design and implement an indoor security system that involved wireless communication, motion detection, image processing as well as additional features including a website with a video stream and control features. The projects objective has been to innovate an easy to use and affordable system ready for consumer use. This technical report provides and overview of the investigation, design and implementation done to deploy the project prototype.

GROUP 5: Dane Lennon, Michael Grasso, Stuart Hinchliff, Stephanie Baker

CC3501

Contents

1. Project Overview

1.1 Project Approach

The investigative approach of the *Indoor Security System* Project began with researching systems currently on the market and analysing their specifications. It was found that most existing systems integrate motion detection algorithms with a continuously running camera to detect motion. One of the design criteria of this project was to utilise a low-power sensor network to implement motion detection over a large area with minimal cost. Therefore, using the camera continuously for motion detection was not suitable.

To minimize the use of the camera, a Photoelectric ("*Passive")* Infrared Sensor was implemented. It was chosen for several reasons, including its ability to detect changes in infrared radiation as well as its low cost and low power consumption. If there is a differential change within the scope of the sensor, the sensors convert it into an output voltage, which represents a detection, illustrated in **Figure 1**.

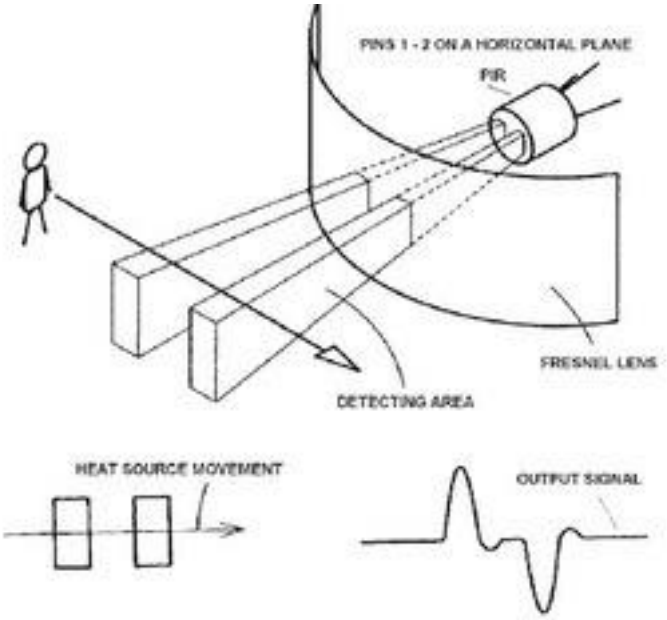

Figure 1: PIR SENSOR [2]

1.2 Project Features and Functionality

This system involves the integration of both hardware and software. It features a custom microprocessor PCB with a PIR motion detecting sensor communicating with a Raspberry Pi via XBee communications.

The features of this project include:

- Webserver application interface with full functionality
- The ability to arm and disarm the system
- Low power initial motion detection using a PIR sensor
- A second check for motion detection with complex image detection algorithms
- Voice configuration mode
- Ability to email and save screenshots

Included in **Figure 2** is a flow chart illustrating how the system works when being controlled by the user.

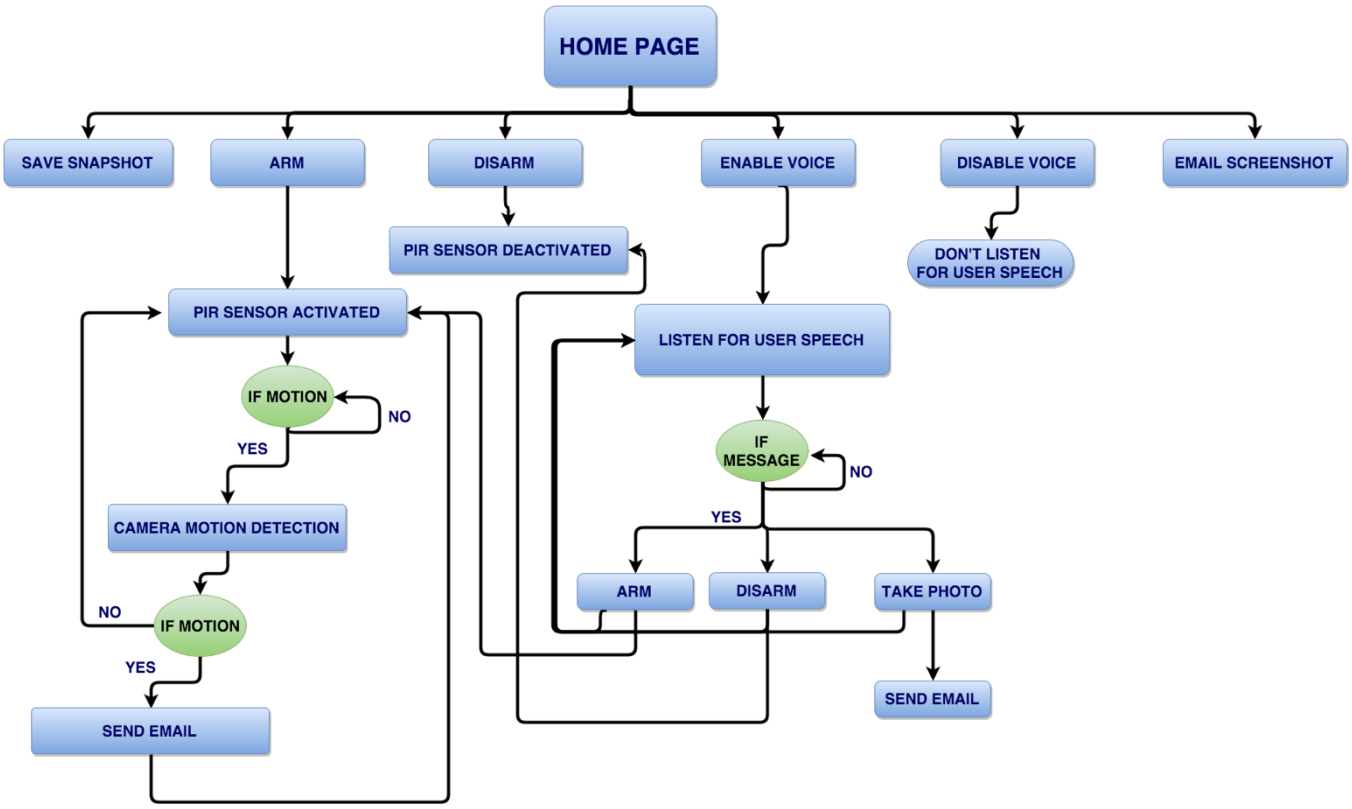

Figure 2: Flow Chart- System Design

2. Hardware Design

2.1 Printed Circuit Board Design

When designing this system, it was a priority that it must be as small and efficient as possible and only include the minimum number of necessary hardware features on the PCB, these included:

- PIR sensor for motion detection,
- LED to indicate motion detection
- USB 2.0 port for power and necessary power supply regulations
- Xbee for Zigbee communication
- JTAG header for SEGA programming
- Battery holder for backup power

To reduce the size of this board further, a custom microprocessor board was designed to replace the standard shield for a Freescale Board. The Freescale FRDM-K20D50M schematic was used as a reference through the design process. Therefore, the microprocessor design, including the clock source and decoupling capacitor designs, was executed similarly to a professional board design. **Figure 3** shows the completed PCB.

Figure 3: PCB Design

2.2 Communications Network

The wireless communications network utilised in this system is a ZigBee mesh network. The ZigBee network used consists of one Coordinator connected to the Raspberry Pi and one Router connected to PCB. This was designed so multiple PCBs can be distributed, automatically forming a mesh network with the Coordinator at the central hub. **Figure 4** illustrates the ZigBee communication network outlined above.

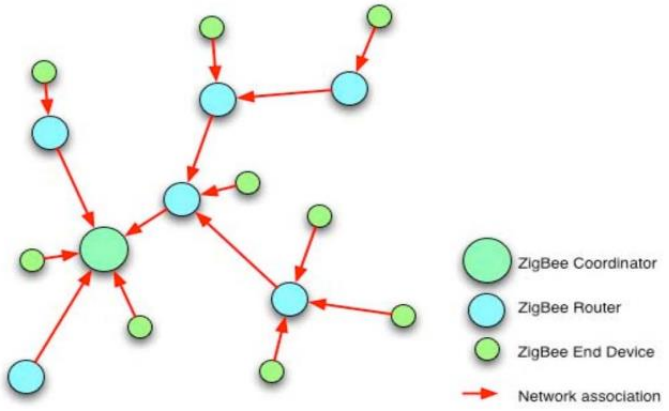

Figure 4: ZigBee Network [5]

2.3 Central Hub and Peripherals

Implementing a security system with advanced features such as image processing and voice control was a task that could only be performed using a highly functional and heavily customizable device. The obvious choice was a Raspberry Pi, with capabilities of a small computer running the Linux operating system. The Raspberry Pi offers all of the features required by the group to design the central hub. Additionally, many open-source libraries are available for Linux to assist with the necessary features of the system such as the development of image processing, voice control and video streaming.

The Raspberry Pi B+ model features two USB ports and a 3.5mm audio jack, allowing the group to attach all required devices with ease. Additionally, a ribbon camera designed specifically for the Raspberry Pi is available, allowed for easy connection of a camera. The list of peripherals used in this project are:

- Raspberry Pi Camera Board

- USB 2.0 Microphone
- Wi-Pi USB 2.0 Connector
- Powered USB 2.0 Hub

These peripherals were connected to the powered USB 2.0 hub to ensure devices would receive enough power and not switch off.

3.0 Software Design

Robust was fundamental to the project. The system integrates scripts in C, C++, Python, HTML, JavaScript, and many external frameworks and libraries. The two main scripts were the C++ code running the main hub, and the Python code operating the webserver. Details of the functionality are described below.

3.1 Main Loop

The main loop controls the system and is responsible for automation of the security system. The structure of the code is demonstrated **Figure 5.**

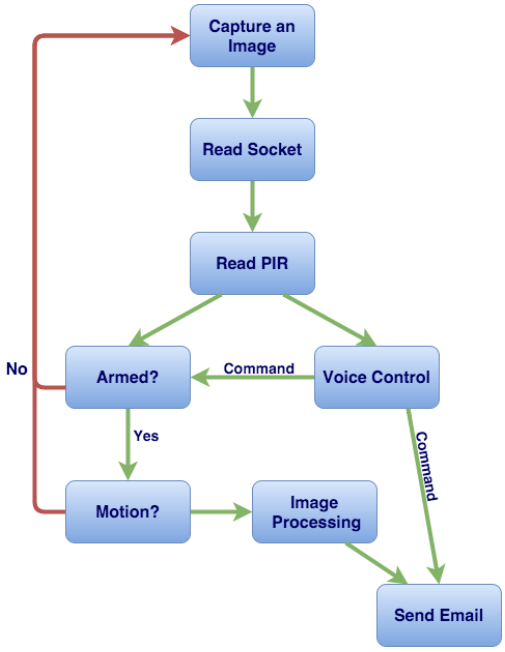

Figure 5: Main Loop Logic

3.2 Server and Web-application

In order to appropriately develop the project, the target audience and their needs were carefully considered. It was assumed that average users would have no prior technical experience with embedded systems, so the software was designed to be user friendly.

A number of control methods were contemplated in the design process, including the use of a Bluetooth or web application. Upon review of similar products available it was determined that a web application provided the flexibility desired with:

- Range of use (only an internet connection required)
- Security (authentication methods)
- Interface (intuitive GUI)
- Accessibility on a range of devices
- Potential for expansion

Various frameworks were considered to host the server. Due to its popularity and power, the lightweight, Python-based, framework called Flask was chosen. Further, a Bootstrap template was used for the webpage structure to improve the visual appeal and enhance user experience.

3.2.1. Video Streaming

Surveillance is becoming increasingly prevalent in security systems and was the main drive for developing a web application. The implementation of this was inspired by *Miguel Grinberg's live video stream [1]*, which demonstrated how versatile Flask could be.

The main complication with using Python was its compatibility with the existing C++ code. It was found that a conflict occurred when attempting to access the camera from Python (for video streaming) and C++ (for image processing) scripts simultaneously. To overcome this, it was decided that saving an image from the C++ code and loading it to Python was the most feasible option. This came with a variety of advantages and disadvantages. It reduced the complexity of the code and it also provided a saved image for use in other processes, but the continuous read/write of images does consume system time. Sockets were also considered, but for image matrices proved complicated and prone to problem.

3.2.2 Web-Application Control

The website comprises of three main pages: dashboard, view screenshots, and email configuration. The functionality of the website is as follows: (see **Figure 2** for flowchart):

- **Dashboard:** View the live stream and execute available commands.
    - *Arm (Default):* Receive data from the PCB. If a 1 (motion detected) is received image processing code is executed and an email is sent to the saved address.
    - *Disarm:* Stop receiving data from the PCB.
    - *Save a SnapShot:* Capture image of the live stream.
    - *Email a SnapShot:* Send an email of the current image to the saved address.
    - *Enable Voice:* Turn on voice control, allowing the user to configure via voice.
    - *Disable Voice (Default):* Turn off voice control.

Figure 6: Dashboard

- *View SnapShot:* View all screenshots taken from the Dashboard for up to 12 saved photos.

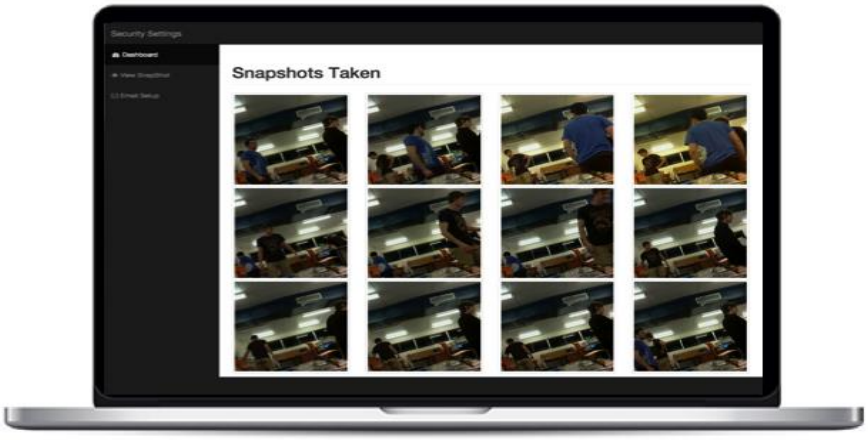

Figure 7: View Snapshot

- Email Setup: Change the current email address to forward notifications to.

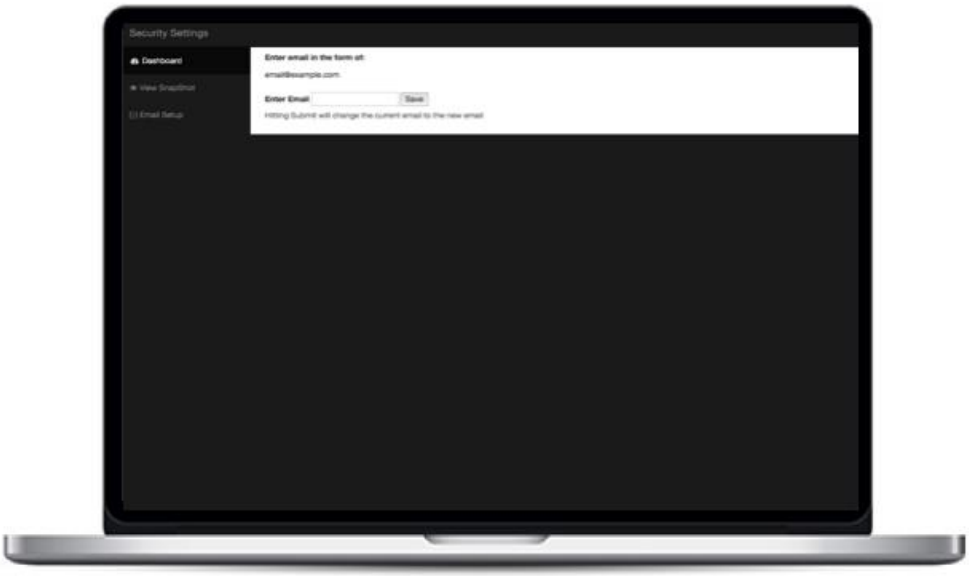

Figure 8: Email Setup

3.2.3 Integration

Multilingual programming was an important aspect of the overall project. In order to ensure various scripts cooperated together, careful consideration of individual tasks and the overall product was taken in design.

The ability to email from the application was a significant feature, with details on its implementation available in Section 3.5. The Python script containing the code is called using a system call from either the C++ or Python code. Due to the sequential nature of the languages used, this caused a halt in the program preventing streaming, voice control and most importantly motion detection. To overcome this, forks were implemented.

Forks are a system call used in Unix-based systems in which a process creates a copy of itself. This process is best explained through the use of a flowchart:

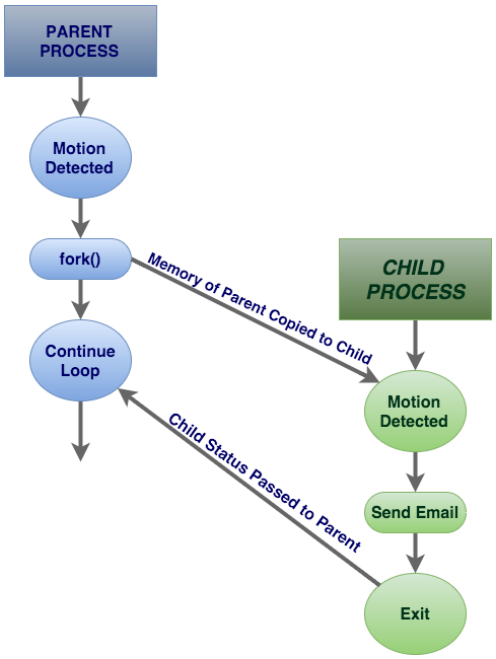

Figure 9: Flow Chart- Fork

The C++ script creates a fork when motion is detected, prior to calling the email function. The parent continues looking for motion while the child process simultaneously executes the email script and is then destroyed.

Acquiring information from the website and sharing it between processes was a crucial part of controlling the system. Flask handled routes from the HTML script, so button clicks were accessible from Python. Non-blocking UDP/IP sockets were then used to transfer the information to the C++ code. UDP/IP was chosen as only two Boolean numbers needed to be sent, and the lack of error checking and overhead increased speed. Additionally, as the sockets were operating on the same system, there would be no loss of data and no need for retransmission.

3.3 Voice Control

The concept of voice control is a very marketable design and could potentially be implemented a variety of ways into any system in order to create an easy-to-use interface for the consumer. It was introduced to this security system as a prototype level feature. In future, it could be expanded to have more complete control.

3.3.1 Speech-To-Text Input

There are several speech-to-text engines available for the Raspberry Pi. The two engines considered were Pocketsphinx and Julius, both well known for their voice recognition capabilities. After research on these two speech engines it was determined that Pocketsphinx was the engine best suited to a project with small sentence structures. It is designed for small systems and implements natural language processing, including advanced algorithms such as n-grams and Hidden Markov Models. A logic diagram for Pocketsphinx can be seen in **Figure 10.**

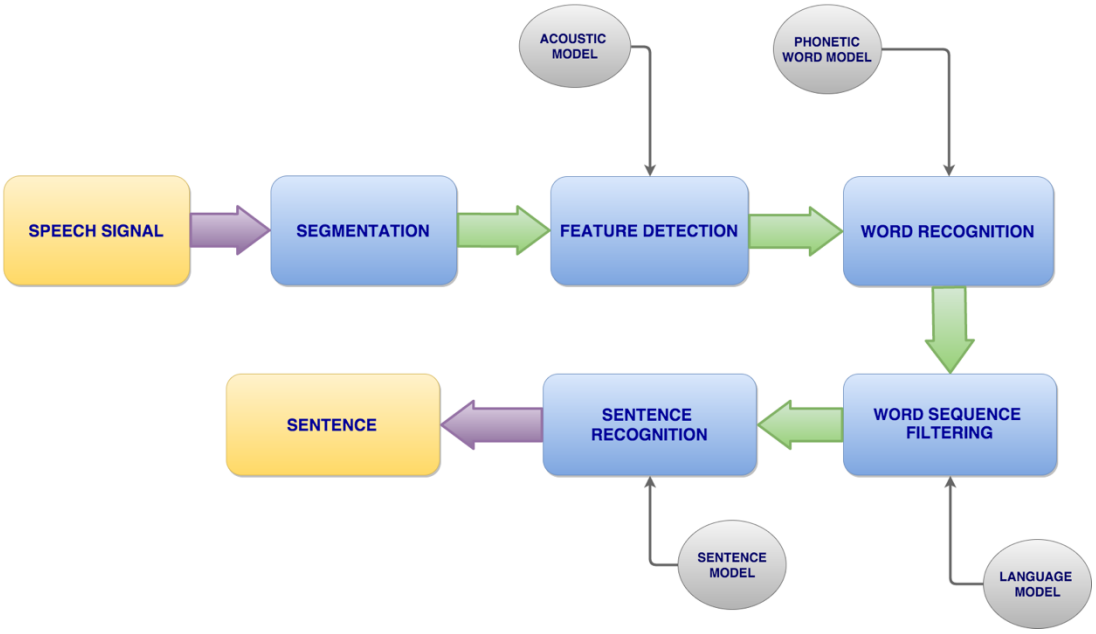

Figure 10: Flow Chart- Pocketsphinx

This speech engine required extensive configuration and modification to suit our purposes. Word recognition was improved using a grammar model tool by Carnegie Mellon University, which involved creating a .txt file containing required commands. The tool generated language and dictation models which were implemented in the C++ code. The dictation model file was then manually modified to include pronunciations consistent with the Australian accent, as initially it contained only American pronunciations.

3.3.2. Text-to-Speech Output

In order to complete a full speech structure, speech synthesis allows the system to talk back to the user. After researching several speech synthesis engines, eSpeak was implemented due to its usability. It was implemented in the C++ code through a system call to pass a string to command line which would generate the desired speech response. Text-to-speech output was used to interact with the user while voice control configuration was active.

3.4. Image Processing for Motion Detection

3.4.1. Image Processing Techniques

There are a number of ways to approach detecting motion using a camera. After extensive research the method which was utilised in this algorithm was the comparison of three frames captured after slight time delay in order optimise the reduction of would background noise. The images are stored as 512 x 512 pixel matrices and processed in C++ with the implementation of OpenCV libraries. The absdiff() function was used twice to take the element-by-element absolute difference between two matrices and return a resultant matrix. Example output of absdiff() is seen in **Figure 11.**

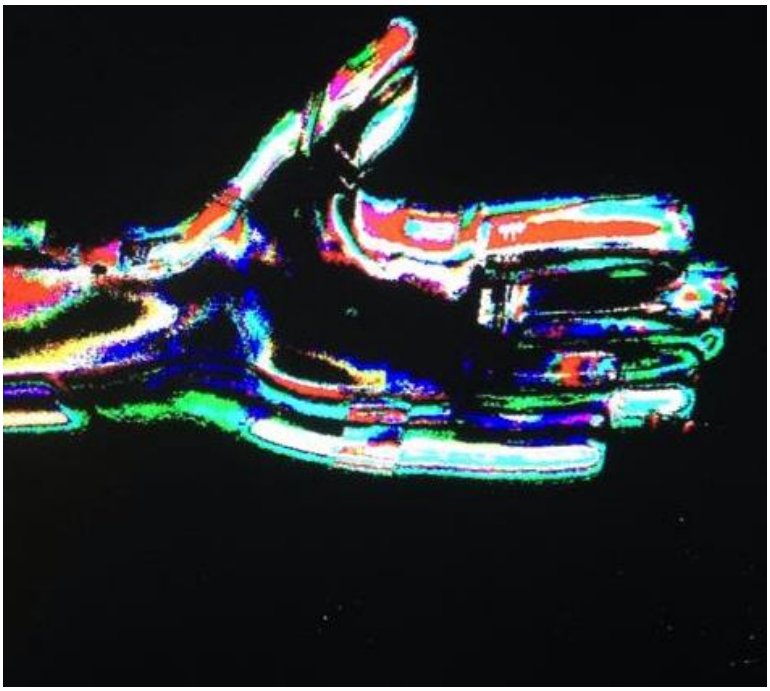

Figure 11 - Image of moving hand after absdiff() is applied

Figure 11 shows that there are some undesired image artefacts, such as the cluster of white pixels in the bottom right corner. In the application of motion detection, it is undesirable to have this background noise in difference images as it triggers a false positive for movement.

To remove artifacts in images, the frames were compared using an OpenCV function bitwise_and(), which takes two image matrices as input and performs a logical 'and' operation on each pixel, resulting in a significant reduction of background artifacts. This function produces a resultant matrix where pixels which are the same colour in image 1and image 2 are returned unchanged, while all other pixels are converted to black.

After artefact removal, the image is converted to greyscale and binary thresholding is performed to eliminate remaining background noise while preserving the foreground image. Binary thresholding returned an image whereby white pixels represent pixels that changed colour between frames, while black pixels are those that remained the same colour.

It was determined through manual testing that if the white pixel count was over 20 pixels, motion had occurred. A bounding rectangle was drawn around the movement region by generating a rectangle between minimum and maximum (x, y) coordinates of the detected movement and overlaying it on captured frames. This makes it easier to pinpoint the source of movement.

3.5 Emailing System

The security system has been implemented with an emailing system with two methods of activation. Emailing of images can be triggered manually from the webpage, or automatically by the C++ code if motion is detected.

3.5.1 Manually Capture and Email a Snapshot

The email system was written in Python, using *StackOverflow* as a guide. The script is run from the command line and contains information about the location of the images to attach and the addresses of sender and receiver. The website dashboard includes a button to manually email a snapshot. The onClick listener of the button calls a function attached to an xhttp route which executes the Python script on the server side.

3.5.2 Automatic Email Alert

When motion is detected by the C++ code, a fork is created and the child process sends an email via a system call to the Python Script.

4.0 Limitations and Recommendations

4.1 Hardware

4.1.1 Component Limitations

The design of the Printed Circuit Board (PCB) was derived from the Freescale Freedom schematic and similar components chosen to match the design. However, during implementation, some components intermittently failed. This included an unresolved voltage output error from the buck boost regulator, which returned approximately 3.7V instead of the expected 5V. The system was redesigned using a though-hole 3.3V linear regulator to overcome this, but the required component could not be purchased before project completion due to time constraints. Further, the use of the linear regulator would have resolved the first issue but in turn produced a second issue where the battery could not have been used.

4.1.2 Design Flaws

During the design stage, flaws occurred in the PCB schematic layout when the USB footprint was placed backwards on the Eagle board layout.

This design flaw was overcome by soldering the USB facedown and wiring the pins of the USB to the tracks of the PCB as shown below. A copper sheet was used to ensure the USB did not make any undesired electrical connections with other PCB tracks. This worked well as a fix, but in the future the design could be easily corrected.

Lastly, the design made use of a surface mounted microprocessor. After the components were placed on the board, it became evident that too much solder paste had been used around the terminals of the processor, as shown below. This resulted in time-consuming re-soldering before the PCB was at a satisfactory level.

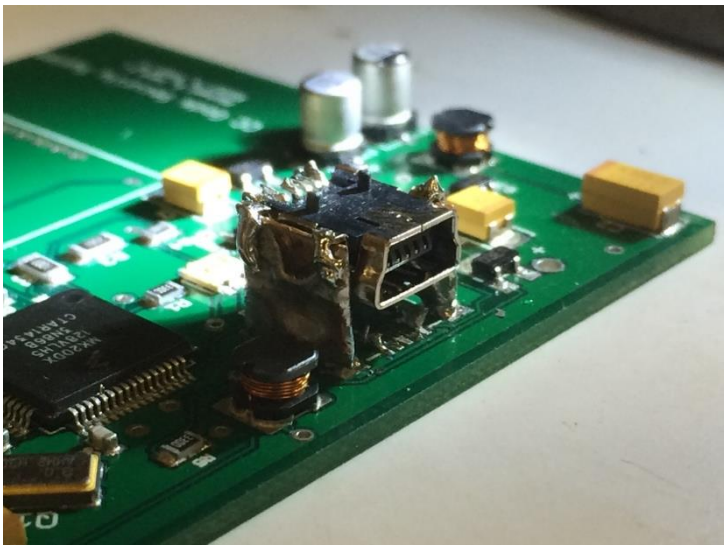

Figure 12: USB Error

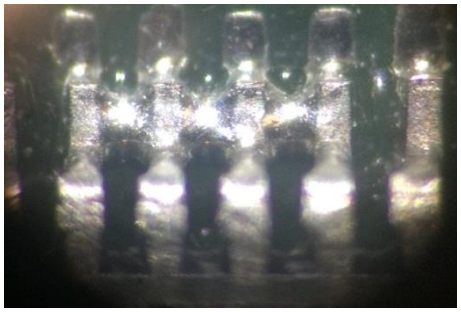

Figure 13: Solder Paste Error

4.1.3 Future Design Implementations

The initial scope of the project included a custom made 3D printed case to house the PCB. However, due to the time constraints of the project the implementation of this feature was discarded in order to prioritized the technical design. Future designs would also resolve the issues outlined above, and off-the-shelf installation and implementation would be implemented for increased usability.

4.2 Software

4.2.1 Voice Control

The implementation of voice control was limited by the recognition engine and microphone quality. Background noise needed to be removed for full accuracy, and the engine did not process all voices with equal effectiveness. While this is a software limitation, it could be overcome by improving the engine to accept a wider variety of voice pitches, dialects and accents. However, this was outside the reasonable scope of a time-constrained project. Voice control also needed to be listening at all times to recognize words, which became problematic when the C++ code also needed to listen to a socket and the XBee.

4.2.3 Saving images

Currently, the system has been designed to save the images to the Pi and load the images to the webpage. This is not an ideal implementation and could be improved using sockets. This would involve sending the data from the C++ code to the Python code in matrix format via UDP/IP. The client side (Python) would receive the matrix and format the image for webpage display.

4.2.4 Image Detection

The image detection algorithm was optimized for an indoor environment, where movement is minimal and lighting is consistent. For an outdoor environment with a dynamic background, new algorithms would need to be developed to accurately detect only relevant movement.

5.0 Conclusion

The scope of the project included the design and implementation of a security system that made use of wireless communication between microcontrollers, motion detection and multiple methods of control such a web app and voice control. These design requirements were successfully implemented with only a few limitations. While most limitations were due to uncontrollable parameters, some flaws were introduced that could have been avoided and these have been documented for future designs. The final product provides a suitable working prototype of an affordable system that implements complex security measures.

6.0 References

[1]M. Grinberg, 'Video Streaming with Flask - miguelgrinberg.com', *Blog.miguelgrinberg.com*, 2014. [Online]. Available: http://blog.miguelgrinberg.com/post/video-streaming-with-flask. [Accessed: 27- Oct- 2015].

[2] Learn.adafruit.com, 'How PIRs Work | PIR Motion Sensor | Adafruit Learning System', 2015. [Online]. Available: https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/how-pirs-work. [Accessed: 27- Oct- 2015].

[3] Speech.cs.cmu.edu, 'The CMU SLM Toolkit', 2015. [Online]. Available: http://www.speech.cs.cmu.edu/SLM_info.html. [Accessed: 27- Oct- 2015].

[4] Flask.pocoo.org, 'Welcome | Flask (A Python Microframework)', 2015. [Online]. Available: http://flask.pocoo.org/. [Accessed: 27- Oct- 2015].

[5] Lecture Notes; Bronson Philippa-CC3501