Retrofittable Apartment Access Device Leveraging Facial Recognition

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Andrew Palmer

MEng Field Advisor: Bruce Land

Degree Date: January 2017

Abstract

Master of Engineering Program
School of Electrical and Computer Engineering
Cornell University
Design Project Report

Project Title: Retrofittable Apartment Access Device Leveraging Facial Recognition

Author: Andrew Palmer

Abstract: The aim of this design project was to explore the facial recognition capabilities of embedded and compute resource limited devices, such as the Raspberry Pi. Through this exploration, an application was developed using the facial recognition features of the OpenCV library as its pivotal subsystem. This design project's application was a retrofittable facial recognition enabled apartment access device. By analyzing a user's face, the device allows or prohibits entry into one's home or apartment. Additionally, the device also hosted a web server that allowed a user to grant access without needing facial recognition. When access was granted from either the facial recognition application or the web server, the device transmitted a radio frequency message to a secondary device attached to the user's buzzer. The buzzer was then activated, which opened the door. Using this application, a user could register common visitors and delivery services, which could alleviate the frustration of missed deliveries from not being at the buzzer. Additionally, the facial recognition application was implemented in Python and C++, which offered comparison between the two implementations on various metrics.

Executive Summary

The goal of this project is to provide a solution to missed deliveries in a shared living space, such as an apartment. Currently, there is a frustrating disconnect between delivery services and customers who do not have access to concierge who can accept deliveries on customers' behalf. When a delivery service arrives at an apartment without physical or electronic concierge, the service will ring the buzzer a few times and wait for a response. When no response is given because the customer is not home, the delivery service leaves a note and a time that it will try to redeliver the package another day, causing frustration and inefficiencies for both parties.

This project looks to solve this inefficiency by allowing delivery personnel to gain access to an apartment building using the facial recognition capabilities of an embedded device. Using this application, named DeliverEZ, a delivery service can register its delivery personnel to the system's allowed users database. Now, when delivery personnel attempt to deliver a package, his picture will be taken and analyzed to determine if he is allowed access or not. If access is allowed, a wireless message is sent to the resident's buzzer, which will then unlock the main entrance and allow the delivery service to leave the package safely inside the apartment building. Additionally, a user may bypass the facial recognition requirements by accessing a web server hosted by the system for cases when an unregistered but valid delivery person requests access, such as a substitute for the usual delivery person.

The final system contains two major hardware components. The first is a base station, which includes a Raspberry Pi 2 as the main processing unit, a Raspberry Pi camera to take pictures of the user, and a Nordic radio transceiver to communicate with the receiving station. The secondary hardware component is the receiving station, which includes a Nordic radio transceiver, which receives messages from the base station, and an Arduino, which processes incoming wireless messages and actuates the resident's buzzer.

The overall goal to implement a complex facial recognition system on an embedded device was achieved by the end of this project. DeliverEZ effectively registered new users to the system, tested a current user against the database of allowed users, restricted unregistered users from access, and granted access to recognized users. Using OpenCV's facial recognition libraries, high recognition accuracy was achieved in constrained environments. The system was also capable of wirelessly transmitting the result of the facial recognition screen or web server command to a secondary hardware unit that then gave a visual cue of the buzzer being activated.

Additionally, as OpenCV provides multiple programming language interfaces, DeliverEZ's facial recognition implementation was written in both Python and C++. Execution metrics were introduced to evaluate and compare the performance between the two implementations.

1 Introduction

The internet of things (IoT) is a popular buzzword that has created a lot of hype in the technology industry. Simply, it is a proposed development of the Internet where network connectivity is given to everyday objects [9]. There were 10 billion connected devices in 2015 with an expected growth of 24 billion more devices by 2020 [6]. Creating these smart objects can have a positive effect on peoples' everyday lives if they are used to efficiently solve problems in society. However, the adoption of IoT products has been slow because they often do not provide any great benefit to an individual's life. The key is to develop exciting and also useful products that will solve an issue that consumers are passionately frustrated about. In the current landscape of ordering everything online, repeatedly missing a delivery elicits the type of frustration that is a perfect place for a smart device enter and solve the problem.

Facial recognition has the technological appeal that used to only exist in science fiction movies. Through continued advances in software and hardware, facial recognition applications are now becoming more common in consumer products. Additionally, open-source computer vision packages, such as OpenCV, offer powerful computer vision development frameworks to the public, allowing anyone the ability to develop computer vision applications. However, computer vision is a computationally demanding task as it often involves computation on large pieces of data and machine learning.

In recent history, facial recognition was reserved for powerful computation platforms and was too demanding for most embedded devices. However, through the technological advances in chip design, embedded devices now contain complex and powerful multi-core architectures coupled with large amounts of memory. These advances allow embedded devices, such as the Raspberry Pi 2, to function more as a full-blown computer than as simple microcontrollers. As a product of this advancement, a $35 dollar Raspberry Pi 2 is more than capable for embedded facial recognition applications.

2 Computer Vision Background

2.1 OpenCV

OpenCV is a library of programming functions mainly aimed at real-time computer vision applications. It is released under a BSD license, which makes it widely available for commercial and academic use. According to the current maintainers of the software, OpenCV has more than 47 thousand

users in its community and has been downloaded over 9 million times, making it an exciting and effective open-source software [10].

Initially developed through an Intel research initiative, OpenCV's main goal was to provide advanced vision research through optimized C++ code for basic vision infrastructure. The initial research team, which consisted of optimization experts in Intel Russia and Intel's Performance Library Team, wanted to provide a common infrastructure for developing computer vision software. They believed that through this common infrastructure, vision-based commercial applications could be advanced technologically.

OpenCV has also given ample opportunity to the academic community to learn about computer vision and develop software applications using the library. As an added bonus, OpenCV is offered for multiple programming languages such as C++, Python, Java, and MATLAB, making it available to an even wider audience.

Beginning with OpenCV version 2.4, facial recognition capabilities were added to the software application programming interface (API). There are currently three available facial recognition algorithms that each use a different process to perform the recognition: Eigenfaces, Fisherfaces, and Local Binary Patterns Histograms. For this design project, the Eigenface facial recognition algorithm was used.

2.2 Facial Detection

A pivotal step in the facial recognition process is facial detection. The processing unit must first be able to determine if there is a face or not before attempting to perform recognition. There are various ways that a computer can detect objects in an image. For this design project, object detection using Haar feature-based cascade classifiers was used to detect a face in an image.

2.2.1 Haar Feature-Based Cascade Classifiers

Object detection using Haar feature-based cascade classifiers was an object detection method proposed by Viola and Jones in 2001 [11]. Their paper claimed that they could implemented a machine learning approach for visual object detection capable of processing images extremely rapidly with high detection rates [17]. At its core, it is a machine learning based approach which trains a cascade function from a large amount of positive (valid) and negative (invalid) images. For example, a positive image would be an image with a person's face, while a negative image could be an image of a shark. A trained classifier will then be able to determine if a face exists in an image.

Initially, the algorithm needs large amounts of positive and negative images to train the classifier. From these training images, Haar features are extracted using the convolutional kernels shown in Figure 1 [11].

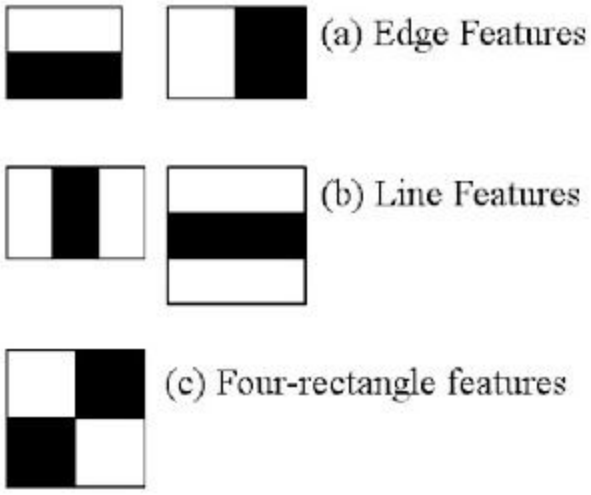

Figure 1 [11]: Three different convolution kernels for Haar features.

All possible sizes and locations of these kernels are used to calculate large amounts of features. In terms of facial features, the convolutional kernels shown in Figure 1 [11] focus on certain properties of human faces. For example, the two kernels in Figure 2 [11] take advantage of different facial properties. The first kernel on the left focuses on the property that the region of the eyes is darker than the region of the nose and cheeks. Also seen in the image, the second kernel focuses on the property that the eyes are darker than the nose's bridge. Viola and Jones had a final setup of around 6000 features to use for facial detection. They also claimed that even 200 features provided detection with 95% accuracy [17].

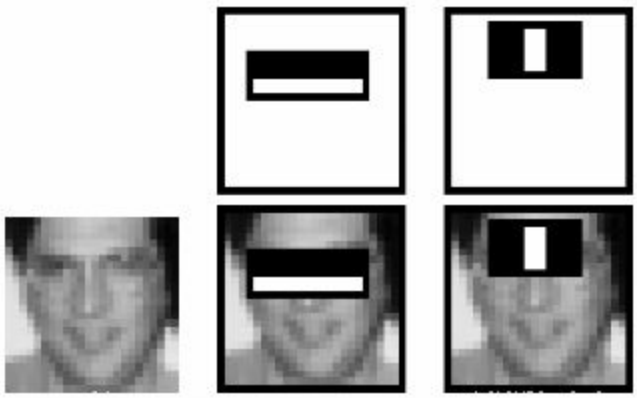

Figure 2 [11]: Different facial properties and associated kernels.

Applying 6000 features to each window in the image, say a 24x24 pixel window, can be extremely time consuming and inefficient to process for a large image. In order to account for this, Viola and Jones took advantage of the fact that most of the area in the image is a non-face region. Thus, it is more efficient to have a simple method to check if a window is not a face region than applying all 6000 features, which would be a waste of time and computation. If the window is not a face region, discard it and do not process again.

To implement this time-saving process, the authors introduced the concept of Cascade of Classifiers. Instead of applying 6000 features on a window, the different features are grouped into different stages of the classifier and applied. If the window fails a stage, discard it. As the window passes different stages, it continues through the cascade. A window that passes all stages is determined to be a face region.

In OpenCV, custom classifiers can be trained to use in later detection. Classifiers for planes, cars, fruit, etc. can be trained for custom applications. For facial detection, OpenCV provides a Haar cascade classifier that is already trained and offered in an XML file. A developer can use this pre-trained classifier by simply loading the XML file into the facial detection program. This design project took full advantage of the pre-trained Haar cascade classifier for the facial detection process.

2.3 Facial Recognition

Facial recognition is an application of machine learning that is capable of identifying or verifying a person from an input video or image source. For humans, face recognition is a simple task that even infants are capable of processing [5]. The process for computers, however, is more involved and complicated as human recognition is actually very difficult to model.

Recognition of human faces based on the geometric features of the face is an intuitive approach that was developed in the 1970s by T. Kanade at Kyoto University [7]. Kanade's approach was to user marker points, such as positions of the eyes, nose, and ears, to build a feature vector. The Euclidean distance between the various marker points in a test and reference image was used to perform the recognition. As a positive, this method was robust against changes in illumination. The most concerning drawback about this method is the inability to accurately register the marker points. Further research in geometric face recognition on large datasets has proven that geometric features alone may not carry enough information for accurate facial recognition [1].

2.3.1 Eigenfaces

The Eigenfaces method, developed by Sirovich and Kirby, takes a different approach to facial recognition. Instead of focusing on specific facial markers, the face images are analyzed more holistically by using principal component analysis (PCA) to construct a set of basis features, called Eigenpictures [14]. The authors found that these Eigenpictures could be linearly combined to reconstruct images in the original training set. Additionally, the reconstruction error was reduced by increasing the number of Eigenpictures.

Expanding upon the Eigenpicture research done by Sirovich and Kirby, authors Turk and Pentland presented Eigenfaces as a method for face recognition. By calculating the eigenvectors of a covariance matrix, the gave computers a way to perform eigen-decomposition on a large number of face images [16]. Various linear algebra concepts are pivotal in creating Eigenfaces from a set of images; however, the derivations and proofs are not within the scope of this design project and can be reviewed by the interested reader.

2.3.2 Training

Part of the facial recognition process is training the system in order to give it the ability to distinguish between different faces. By providing a large amount of training images with classification labels, the system can use this information to make predictions on a test image's class. It is imperative to note that training and testing images should be completely separate. If the system is tested with the same images that it used for training, it will always have 100% accuracy. Additionally, all face images used in the training process must be of the same size.

In the case of training Eigenfaces, a large amount of positive and negative images are required to create robust facial recognition models. For this design project, the AT&T Facedatabase was used as the primary training image set [2]. This database contains ten different images of 40 distinct subjects. Lighting conditions, facial expressions, and facial details vary across the 40 different training sets, making it a valid database for initial training and tests.

When training the model, an average Eigenface is generated from the training data. The average Eigenface is what an average human face looks like according to the training set. Additionally, a positive Eigenface is developed from the training data that is a summary of features that differentiate that user's face from the mean face. The negative Eigenface then represents all the other faces in the training set that will not be the user's face. Figure 3 [3] offers an illustration of how these Eigenfaces appear.

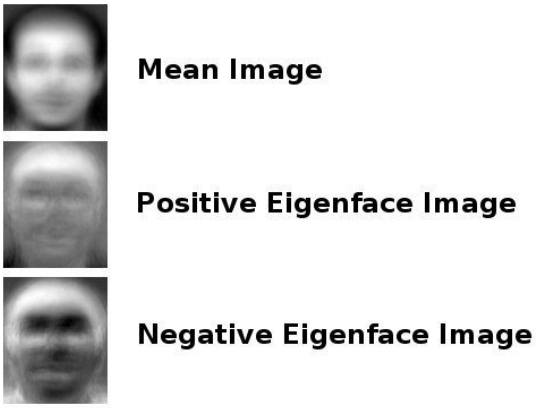

Figure 3 [3]: Illustration of different Eigenface images.

2.4 Existing products, projects, research

The initial inspiration for this design project came from Tony Dicola's Raspberry Pi Face Recognition Treasure Box project on Adafruit [3]. His project provided the background and framework for how to develop a facial recognition system in Python. However, his treasure box was only set up to allow one user for the system. My major development for this design project was to expand the system architecture to allow multiple users, which meant multiple positive Eigenfaces.

As a precursor to this design project, I developed a different facial recognition system, named Face Cafe, for a graduate level course's final design project. Face Cafe was a facial recognition enabled device that dispensed cream and sugar into a user's coffee based on the user's predefined preferences in the device's user database. The system successfully leveraged facial recognition using OpenCV and Python and was able to account for multiple concurrent users of the system. The Python code developed for Face Cafe provided the backbone for this design project's Python software architecture. When developing the C++ version of this design project, my original Python implementation was used as a model, which greatly aided the development.

2.5 Design Alternatives

As discussed in Section 2.1, OpenCV provides 2 facial recognition algorithms other than Eigenfaces. Fisherfaces and Local Binary Patterns Histograms (LBPH) are both proven facial recognition algorithms and were looked at as design alternatives. These algorithms both have their benefits, including being more robust against changes in scene lighting; however, Eigenfaces was chosen as the algorithm of choice because of the initial work done by Tony Dicola, which used Eigenfaces.

Additionally, a face image database other than the AT&T Facedatabase could have been used for this project. The Facedatabase was used in Tony Dicola's project as well as my Face Cafe design, making the transition to a new project simple. However, this database is rather simple and good for initial algorithm tests. Other databases, such as the Yale Facedatabase, provides more variation in the face images, which makes the recognition problem more difficult. As this design project's main goal was to successfully develop a facial recognition system and not evaluate the effectiveness of different recognition algorithms, the AT&T Facedatabase was a valid design choice.

3 System Design

3.1 High-Level Design

DeliverEZ is broken down into two main systems. The first system is the main processing unit and transmitter. This system captures images, trains facial recognition models on images, manages the different users of the system, performs facial recognition analysis, hosts the web server, and transmits facial recognition results to a receiving unit. The second system is a receiving unit that resides on the resident's apartment buzzer. It takes in radio frequency messages from the main unit and gives a visual cue about when the apartment's buzzer is actuated.

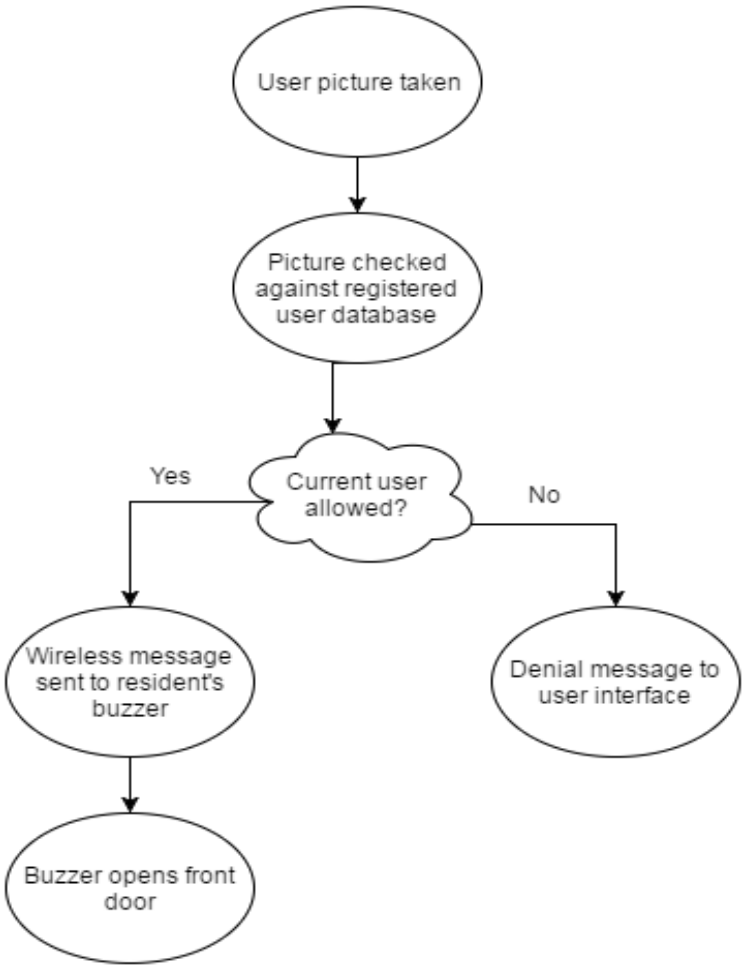

Figure 4: High-level flowchart of DeliverEZ system processes.

Figure 4 shows the high-level flow of processes in the DeliverEZ system when a user wants to access the building. As seen in the figure, the user's picture is taken and passed to the main processing unit for evaluation. During this evaluation, the user's face is analyzed and compared against the registry of allowed user faces. If the current user's face is in that allowed registry, the main processing unit sends a message to the receiving unit granting access to the building. In contrast, if the current user's face is not allowed, a denial message is presented on the user interface alerting the user of the result.

3.1.1 Central Processing Unit

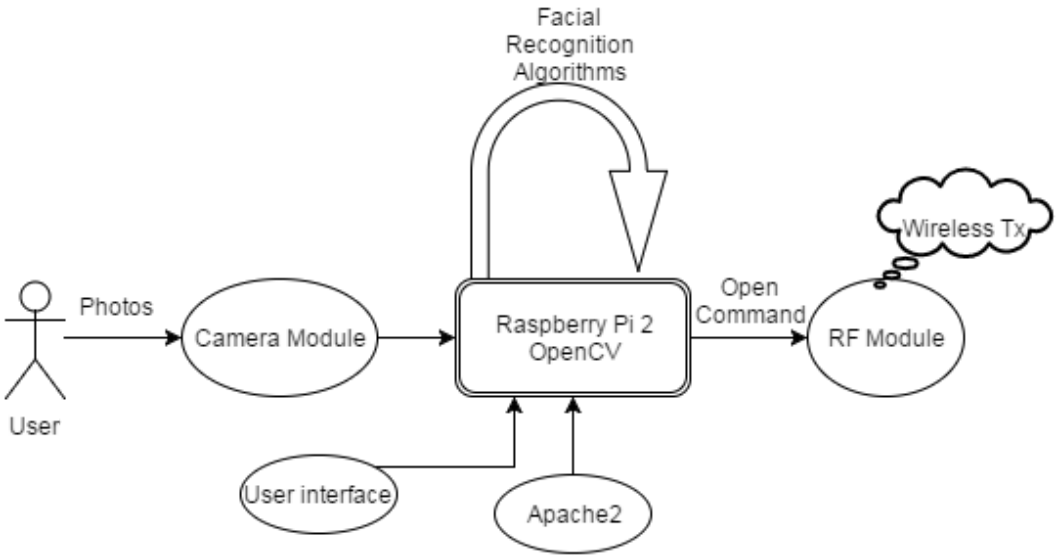

Figure 5: Central processing unit high-level design flowchart.

The central processing unit, or base station, performs the vast majority of the computation and logic in the DeliverEZ system. Figure 5 provides a flowchart for the actions performed by this unit, which is a Raspberry Pi 2. As can be seen in the flowchart, the Raspberry Pi takes inputs from the Raspberry Pi camera, user interface, and Apache2 web server. Additionally, there is an acknowledgement included in the wireless design to give a confirmation message to the Raspberry Pi on a successful wireless transmission.

When user photos are captured by the camera, the Raspberry Pi performs facial recognition operations, such as training new users or predicting current users. The facial recognition operation is determined by the current state of the system, which is controlled by the user interface. Training images are saved into the Raspberry Pi's filesystem while prediction images are discarded after use.

From the web server, the Raspberry Pi can receive a request to transmit a command to the secondary, receiving unit. This command is initiated by the user requesting a specific URL, which contains a script on the web server to transmit the command.

Its main source of output, the Nordic RF24L01 wireless transceiver, transmits information to the receiving unit on the resident's buzzer. The Raspberry Pi 2 base station also delivers system details as an output to the user interface. An alternate design decision was to use a touch screen, like in the previous project Face Cafe, to improve the user interface and provide more output information to the user.

3.1.2 Receiving Unit

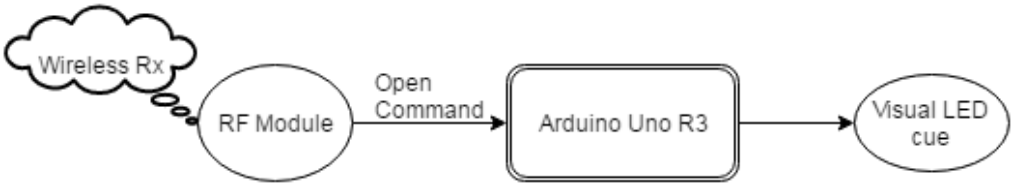

Figure 6: Receiving unit high-level design flowchart.

The receiving unit, which lies on the resident's apartment buzzer, has limited computation in the DeliverEZ system. As seen in Figure 6, the receiving unit consists of an Arduino Uno as its computational brain. The Arduino takes wireless messages as inputs that were received from the Raspberry Pi base station using a second Nordic RF24L01 transceiver. As an output, the Arduino illuminates an LED when an open command has been received to let the resident know that the buzzer is currently being actuated. Additionally, the Arduino explicitly acknowledges messages received from the Raspberry Pi by sending a response wireless message back to the Raspberry Pi.

3.2 System Requirements

There are a few important system requirements for implementing the DeliverEZ system. OpenCV version 2.4 or above is required as OpenCV 2.4 is the first version to offer facial recognition algorithms. There are additional OpenCV dependencies that must also be installed and can be found on the OpenCV website.

For the Python implementation of this design project, there are also dependencies that must be installed. The picamera library is required for interfacing with the Raspberry Pi camera in Python. Since GPIO pins on the Raspberry Pi are being used, the RPIO library must also be installed.

The C++ implementation of this design project could not take advantage of the Raspberry Pi camera library written for interfacing in Python. The RaspiCam C++ API provided by Rafael Salinas offered a great way to capture images using the Raspberry Pi camera hardware and was used for this project's C++ implementation [13].

DeliverEZ's wireless application code was written entirely in C++ for the Nordic RF24L01 2.4 GHz transceivers. GitHub user TMRh20 provided an optimized library that was used to develop this project's wireless application [15].

Lastly, the two hardware units consisted of a Raspberry Pi 2 for the main processing unit and an Arduino Uno for the receiving unit. The Raspberry Pi 2 runs the Jessie version of Raspbian.

There are also environmental requirements and constraints for the DeliverEZ system that should be noted. First, DeliverEZ assumes the user is in an apartment dwelling. The apartment dwelling must also have a buzzer system that is capable of being retrofitted with this system. Retrofitting a buzzer system may require removing the panel and moderate experience with electrical systems and wiring. Additionally, there is a range constraint between the front door of the apartment, which houses the main processing unit, and the resident's buzzer, which houses the receiving unit. The max range of the radio modules used in this system is 100 meters. Finally, if the user wishes to use the web server functionality of DeliverEZ, the main processing unit must be connected to the internet either through the ethernet port on the Raspberry Pi 2 or a WiFi dongle.

3.3 Hardware Design

Being a software-heavy design project, DeliverEZ did not contain custom hardware components. Nearly all of the hardware components, which are described in the following subsections, were prefabricated and ready for use.

3.3.1 Central Processing Unit

3.3.1.1 Raspberry Pi 2

The Raspberry Pi 2 is an excitingly capable and low-cost embedded development platform. Being able to run the Linux operating system, the Raspberry Pi can accomplish much more complicated tasks and computations than a simple microcontroller. Equipped with a Broadcom BCM2836 SoC 32-bit quad-core ARM Cortex-A7 processor running at 900 MHz, the Raspberry Pi 2 has as much processing power as a smartphone, such as the 2014 Moto G [12]**.**

Raspberry Pi's are geared towards academia and the open-source community. Because of this, there are ample free libraries readily available. Additionally, there are countless forums and previous projects published online that make the Raspberry Pi a fantastic platform for any embedded project that demands more computational power than a microcontroller. Figure 7 [3] illustrates the basic anatomy of the Raspberry Pi 2 development platform.

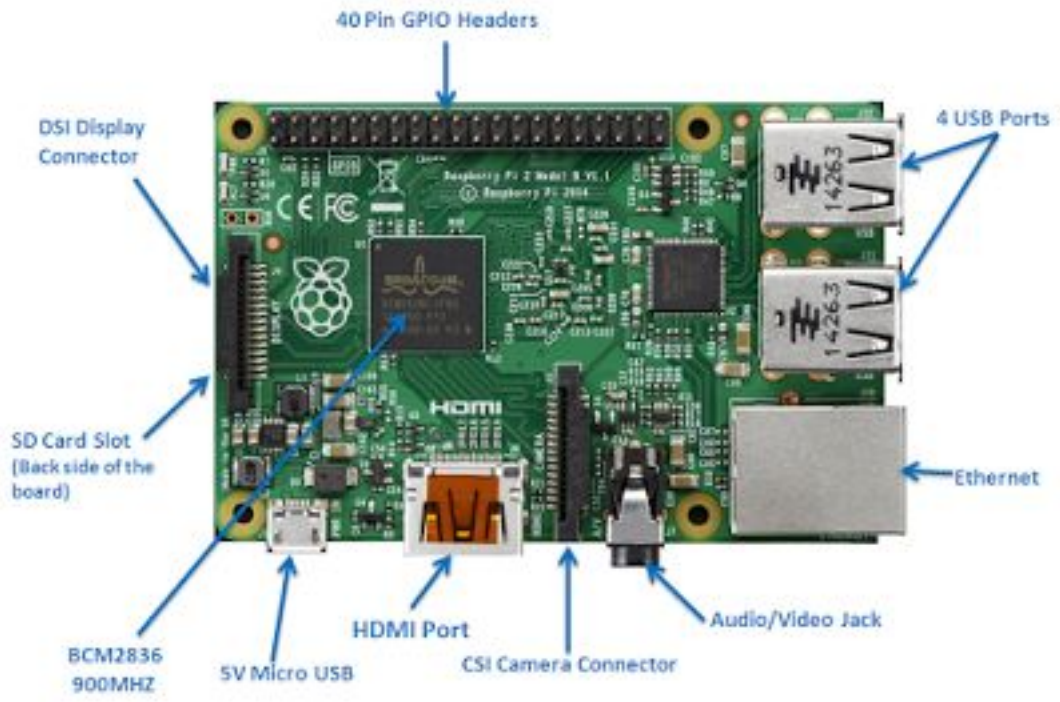

Figure 7 [3]: Basic anatomy and components of the Raspberry Pi 2.

Another positive aspect of the Raspberry Pi 2 as a development platform is its wealth of GPIO pins. On this model, there are 40 GPIO pins as shown in Figure 7**.** Many of the 40 pins can be used freely for general input and output. However, some of these pins are reserved for special functions such as SPI and I2C communication protocols. Figure 8 shows the Raspberry Pi 2 pinout and key for what each pin can be used for. Table 1 gives a summary of the hardware specifications for the Raspberry Pi 2.

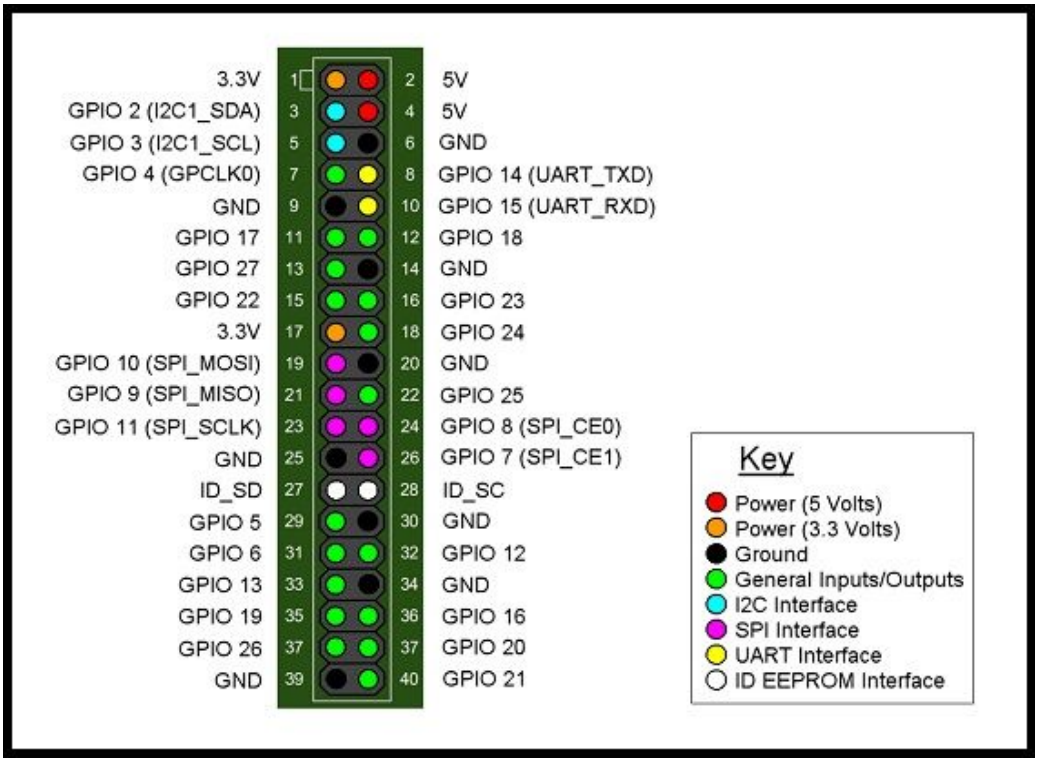

Figure 8: Raspberry Pi 2 pinout and pin key.

Raspberry Pi 2 Summary	
SoC	Broadcom BCM2836
Core Type	Cortex-A7
Number of Cores	4
CPU Clock	900 MHz
RAM	1 GB
Current Output	800 mA
Operating Voltage	3.3 V

Table 1: Summary of Raspberry Pi 2.

Because this hardware unit is constantly connected to a power source, low power consumption was not design concern. Therefore, the optimal hardware properties for this design project were the 900 MHz quad-core Broadcom SoC and the 1GB of RAM. Another important point to note is the Raspberry Pi 2's operating voltage of 3.3 Volts. Although the hardware components in DeliverEZ operated at this same voltage, care must be taken when connecting components to the Raspberry Pi 2's GPIO pins. The maximum input voltage to a GPIO pin is 3.3 Volts, and anything over can cause damage to the device.

3.3.1.2 Nordic RF24L01 2.4 GHz Transceiver

There are many number of wireless communication devices that could have been used for this design project. Different communication protocols, such as WiFi and Bluetooth, were also explored as being a potential candidate for DeliverEZ's wireless transceivers. However, the Nordic RF24L01 transceiver was chosen for its ease of use, price, and range. Additionally, there are multiple optimized libraries for this transceiver written in C++ that were freely available for use, making it a great wireless module choice. Figure 9 [4] shows the Nordic RF24L01 transceiver anatomy and pinout and Table 2 gives major characteristics of the device.

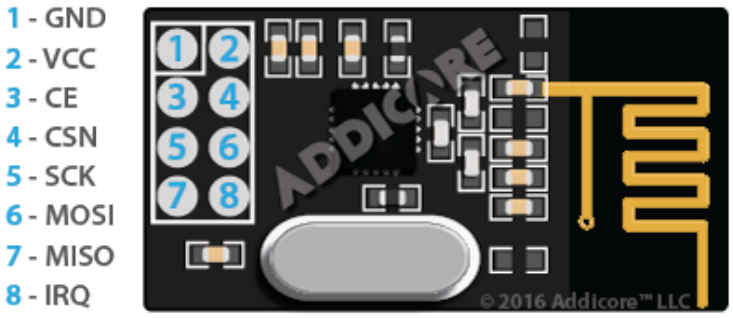

Figure 9 [4]: Nordic RF24L01 anatomy and pinout.

Nordic RF24L01 Summary	
Operating Voltage	1.9-3.6 V
Operating Frequency	2.4 GHz
Power Consumption	13.5 mA
Emission Distance	70-100 m

Table 2: Summary of Nordic RF24L01 radio module.

3.3.1.3 Camera Module

The Raspberry Pi 2 has a specialized camera interface, called Camera Serial Interface (CSI), that provides faster processing than using a USB-connected camera. The specialized interface attaches directly to the GPU, while a USB camera requires a lot more CPU processing, which increases the overhead of capturing an image. Taking advantage of this performance gain, the Raspberry Pi Camera Revision 1.3, using CSI, was used as the source of capturing images. The Raspberry Pi camera module is shown in Figure 10 along with some important specifications given in Table 3.

Figure 10: Raspberry Pi camera module.

Raspberry Pi Camera Summary	
Resolution	5 MP
Video	1080p30, 720p60, VGA90
Interface	Camera Serial Interface (CSI)

Table 3: Summary of Raspberry Pi Camera Module.

3.3.1.4 Main Processing Unit Wiring Diagram

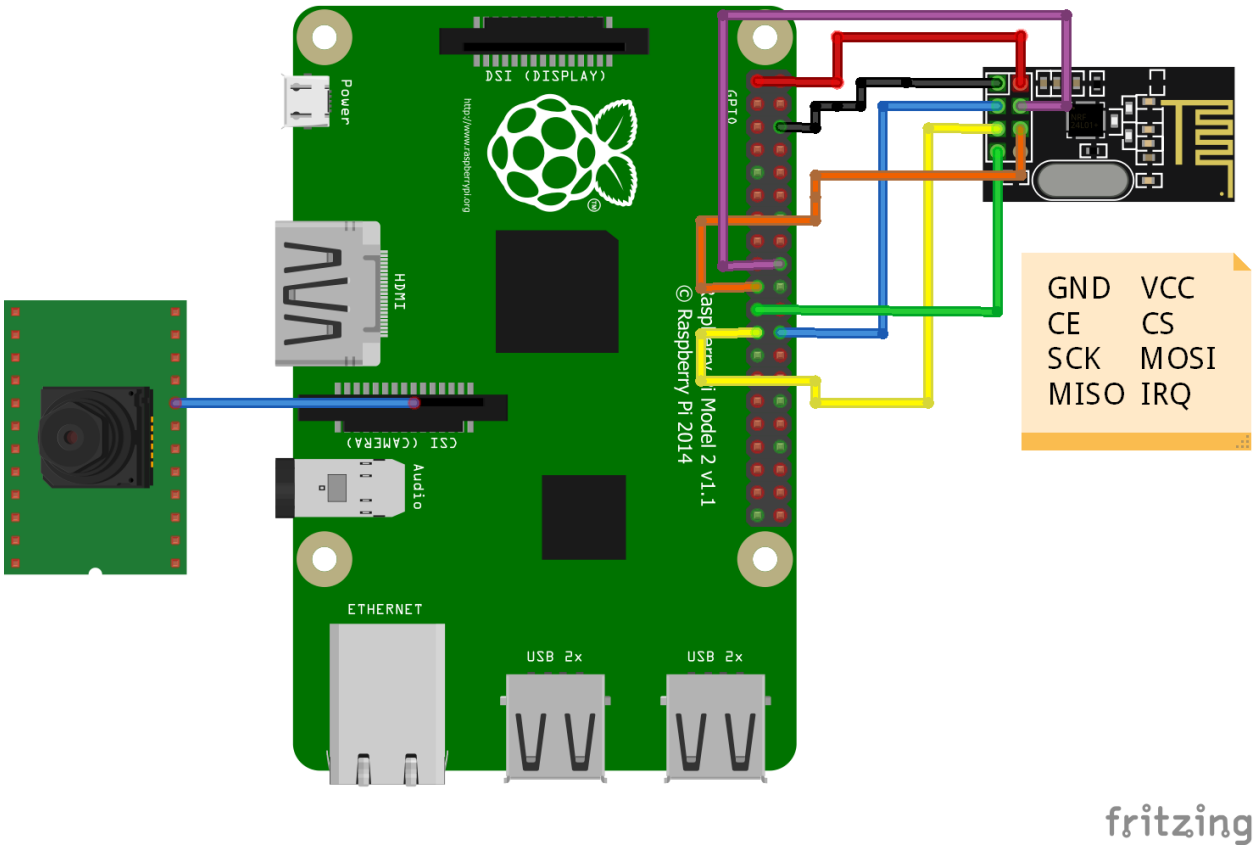

Figure 11: Wiring connections between the Raspberry Pi 2, RF transceiver, and camera.

Figure 11 illustrates the wiring connections between the components in the main processing unit. The Nordic RF24L01 radio module is connected to the Raspberry Pi 2's SPI0 pins. The Raspberry Pi camera comes with a ribbon cable, as seen in Figure 10 that connects to the CSI interface on the Raspberry Pi. Additionally, an ethernet cable is connected to the Raspberry Pi 2 to give it internet connectivity. A WiFi USB dongle can also be used in the absence of ethernet; however, ethernet provides faster internet connectivity for certain DeliverEZ features.

3.3.2 Receiving Unit

3.3.2.1 Arduino Uno R3

Because of the receiving unit's limited amount of required computation for this design project, it does not require as powerful of a development platform as the Raspberry Pi 2 used in the central processing unit. As the brains for the receiving unit, an Arduino Uno R3 development platform is used.

Equipped with the ATmega 328P microcontroller, Arduino development platforms are known for their ease of use, cost, and large development community. The large amount of previous projects and open-source libraries, just as with the Raspberry Pi 2, make it a great candidate for many projects.

Figure 12 offers a diagram of the Arduino's pinout and basic anatomy. As seen in the figure, there are plenty of different types of GPIO pins along with specialized pins for interrupts, PWM, SPI, and I2C. Table 4 provides a summary of the Arduino Uno R3 specifications.

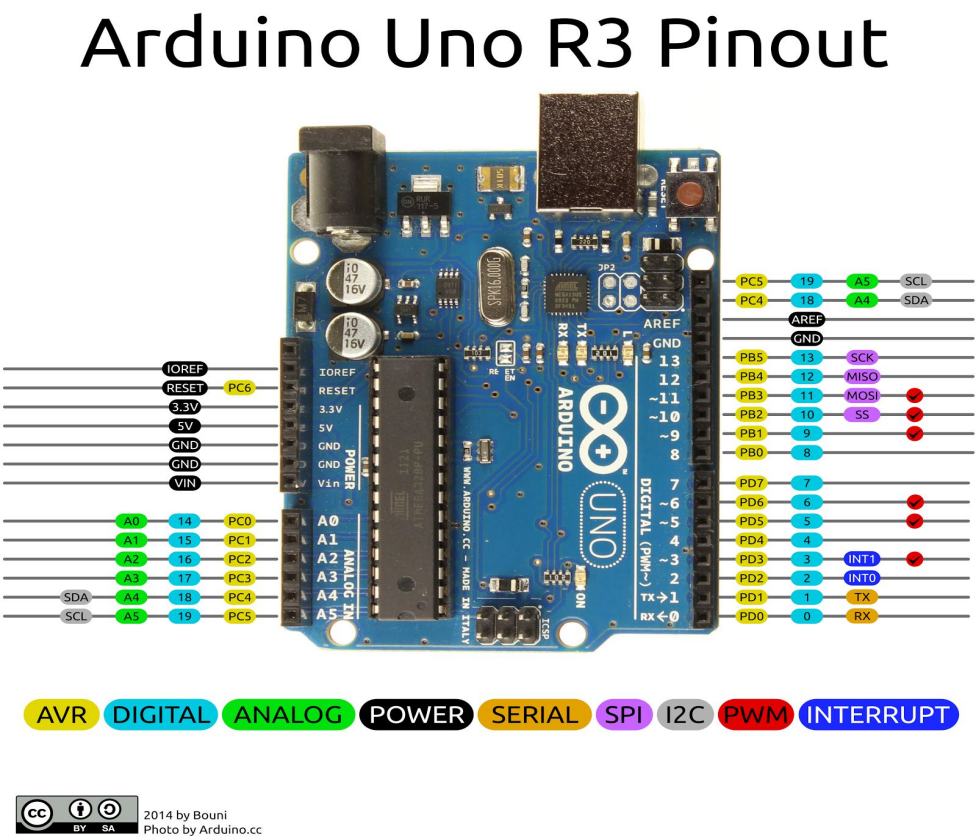

Figure 12: Diagram of Arduino Uno R3 Pinout and anatomy.

Arduino Uno R3 Summary	
Microcontroller	ATmega328
CPU Clock	16 MHz
Digital I/O pins	14
Analog Inputs	6
Operating Voltage	5 V
Current Output	200 mA

Table 4: Summary of the Arduino Uno R3 development platform.

3.3.2.2 Nordic RF24L01 2.4 GHz Transceiver

The receiving unit contains the same Nordic RF24L01 radio transceiver as the central processing unit. Please refer to Section 2.3.1.2 for information on this device.

3.3.2.3 Receiving Unit Wiring Diagram

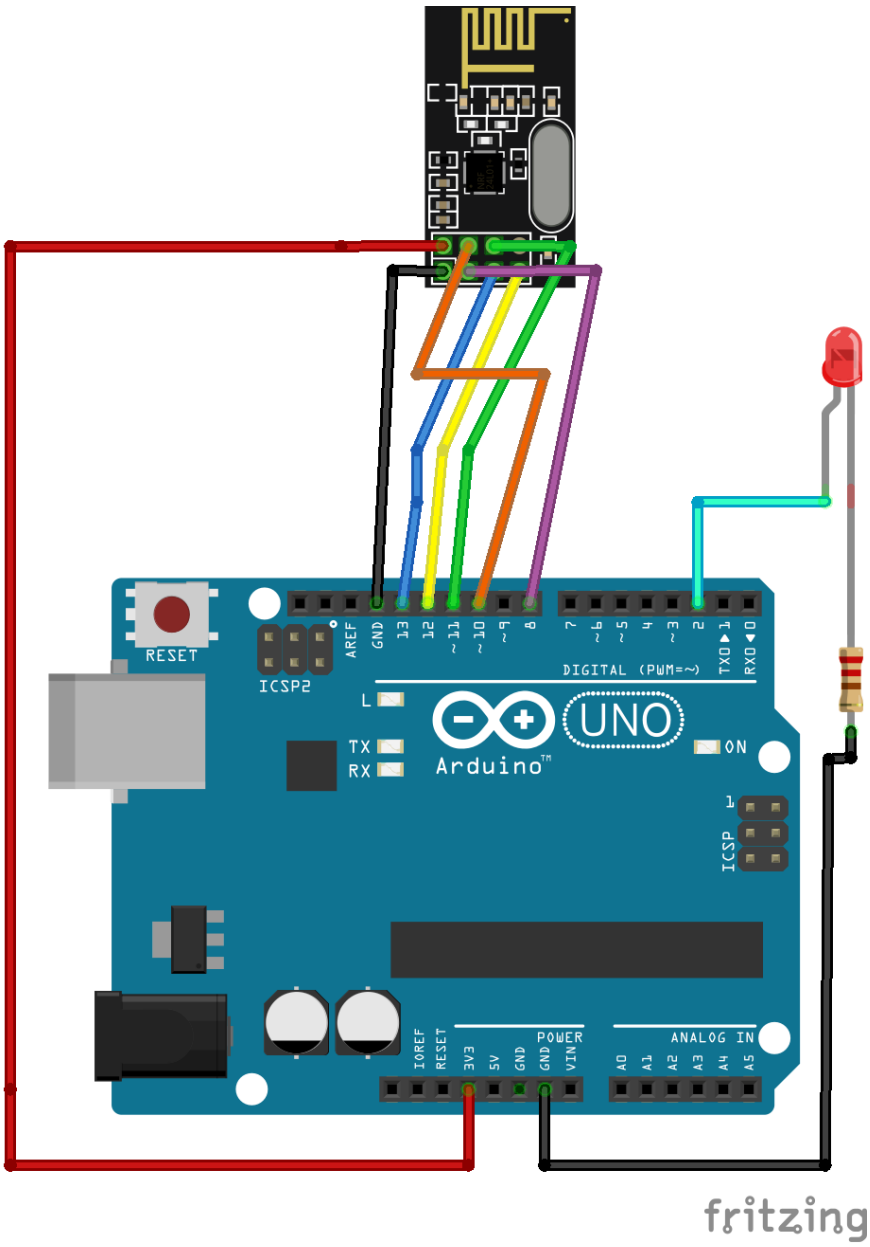

Figure 13: Wiring diagram of Arduino Uno, RF transceiver, and LED.

The wiring diagram shown in Figure 13 illustrates the connections on the receiving unit. The Arduino is connected to the Nordic RF24L01 transceiver and a light emitting diode. Utilizing SPI, the Nordic transceiver connects to the Arduino's SPI connections as follows: Pin 13 to SCK, pin 12 to MISO, pin 11 to MOSI, pin 10 to SS, and pin 8 to CE. Additionally, the Nordic transceiver uses 3.3 V, thus has its VCC connected to Arduino's 3.3 V output supply port. The LED has a simple connection with its

anode connected to Arduino pin 2, which is digital output, and its cathode connected to a 220 Ohm resistor, which is in turn connected to the Arduino's GND port.

3.4 Software Design

The software design on the main processing unit for both the Python and C++ implementations of DeliverEZ followed the same overall architecture. The architecture was split into multiple functional modules that were tested individually before integration with the entire system. For each implementation, the major functional blocks were as follows: A main application that runs indefinitely and waits for user input to perform other actions, an application to register a new user's face to the system, and an application for a user to request access. Inside each of these main function modules, there are multiple other modules to facilitate the desired operation. Additionally, there is the wireless transmission application and the web server application running on the main processing unit. Each module's software design is explained in more detail in the following sections.

On the receiving unit, the Arduino software design is simpler as the Arduino is only running one sketch indefinitely. This sketch continually waits for incoming wireless messages for the main processing unit and performs actions based on the incoming message.

3.4.1 Main Application

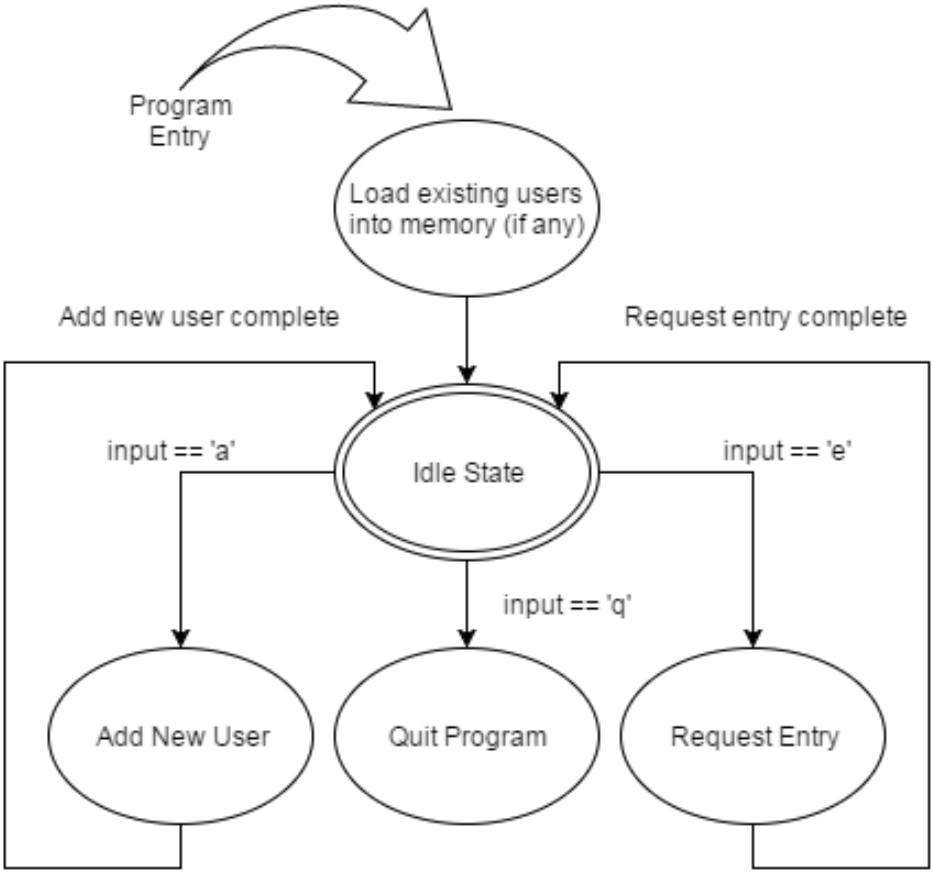

Figure 14: Control flow graph of main DeliverEZ application.

Figure 14 illustrates the control flow graph of DeliverEZ's main application. Once started, the main application runs indefinitely on the Raspberry Pi. From here, the user can perform different actions based on input into the command line user interface. When the user gives this application input, main calls the appropriate code from different functional blocks to complete the request. Upon completion, main continues to wait for user input in the idle state.

When DeliverEZ is started, existing users, and their facial recognition models, are loaded into program memory. If the system is fresh and no users currently exist, this step is skipped and main enters an infinite polling loop represented in the figure as the Idle State. Additionally, a vector of user names and a vector of user facial recognition models are initialized for later use.

From the Idle State, there are 3 different control paths that the system can take. If the user wishes to add themselves to the system, the user enters ‘a’ into the command line user interface. From here, the main application will begin the process for registering a new user to the system and creating the new user’s facial recognition model. If a pre-existing user wishes to gain entry into the building, the user enters ‘e’ into the system. The main application will then initiate the required steps to analyze the current user and determine if that user is allowed access. To quit DeliverEZ, the user enters ‘q’ to the system, which will terminate the program and release the resources used by DeliverEZ.

3.4.2 Registering a New User

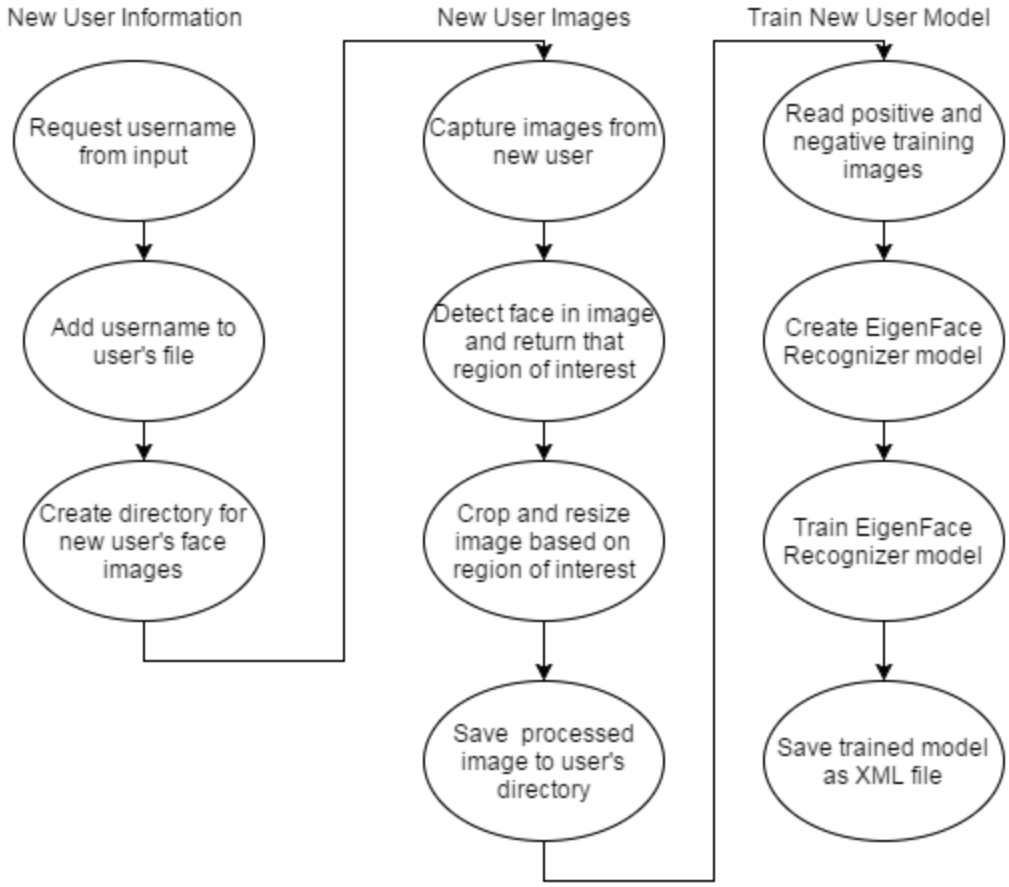

Figure 15: Flowchart of control flow in registering new user software process.

Figure 15 provides a flowchart illustrating the different operations of the functional module that handles adding a new user to the DeliverEZ system. When the main application first transitions to this functional module, the processes under the “New User Information” pipeline will be the first to take place. As a first process, the user will be prompted to register to the DeliverEZ system by providing a

username. This username is added to a file that contains all the currently registered users of the system. Next, the username is used to created a personal directory that will soon contain face images of the new user. Giving each user a personal directory helps the organization of DeliverEZ filesystem and also aids in the training process.

Under the "New User Images" pipeline in the above figure, the shown processes are involved in capturing face images and processing them for future use. Once the user has completed the steps in the "New User Information" pipeline, the user will be prompted to take face images using the Raspberry Pi camera. When a picture is taken, the program then checks to make sure that there is a valid face in the image by running the face detection program.

The face detection program follows the processes described in Section 2.2.1 by utilizing the Haar cascade provided by OpenCV for facial detection. When an image is passed to the face detection program, the face detect Haar cascade XML file is loaded into memory. Then, an OpenCV API is called that uses the Haar cascade to detect a single face in the image. If a face is detected, the API returns a rectangle object that contains the 4 corner points of the rectangle that surrounds the face. This rectangle is important, as it is the region of interest (ROI) that the system needs for further processing. The ROI is returned and used as a parameter for the crop and resize function.

The crop and resize function is crucial for the entire facial recognition system. As discussed in Section 2.3.2, all images used for facial recognition training and processing must be the same size. The negative training images provided by the AT&T Facedatabase have the dimension 92x112. Therefore, all face images captured by DeliverEZ are resized to 92x112 pixels. Before resizing the image to the appropriate size, the image is cropped using the rectangular ROI that was returned from the face detection code. Once the image only contains the relevant face data, it is resized and returned.

The final step in the "New User Images" pipeline is to save the processed face image into the appropriate user file. After completing this step, the new user's face image is ready to be used for training a unique facial recognition model. As using more pictures improves the robustness of the user's facial recognition model, the user may repeat the "New User Images" pipeline a number of times to capture multiple face images for use in training.

Training a new user facial recognition model is the most computationally intensive task in a facial recognition system. This process accumulates hundreds of images, both positive and negative, in the DeliverEZ system to run the training algorithms. Once the images are loaded into an array, an EigenFaceRecognizer class object is initialized. As DeliverEZ uses Eigenfaces for facial recognition, this is the object of choice. Once the object is created, the OpenCV API to train the facial recognition model is called. This process can take around 10 minutes depending on how many images are used in the

training process. When finished, the XML file representing the facial recognition model is saved to the DeliverEZ filesystem. The EigenFaceRecognizer object that was created in this training process is returned by this functional module to the main application where it is appended to the vector of current user models registered to the system.

3.4.3 Requesting Access

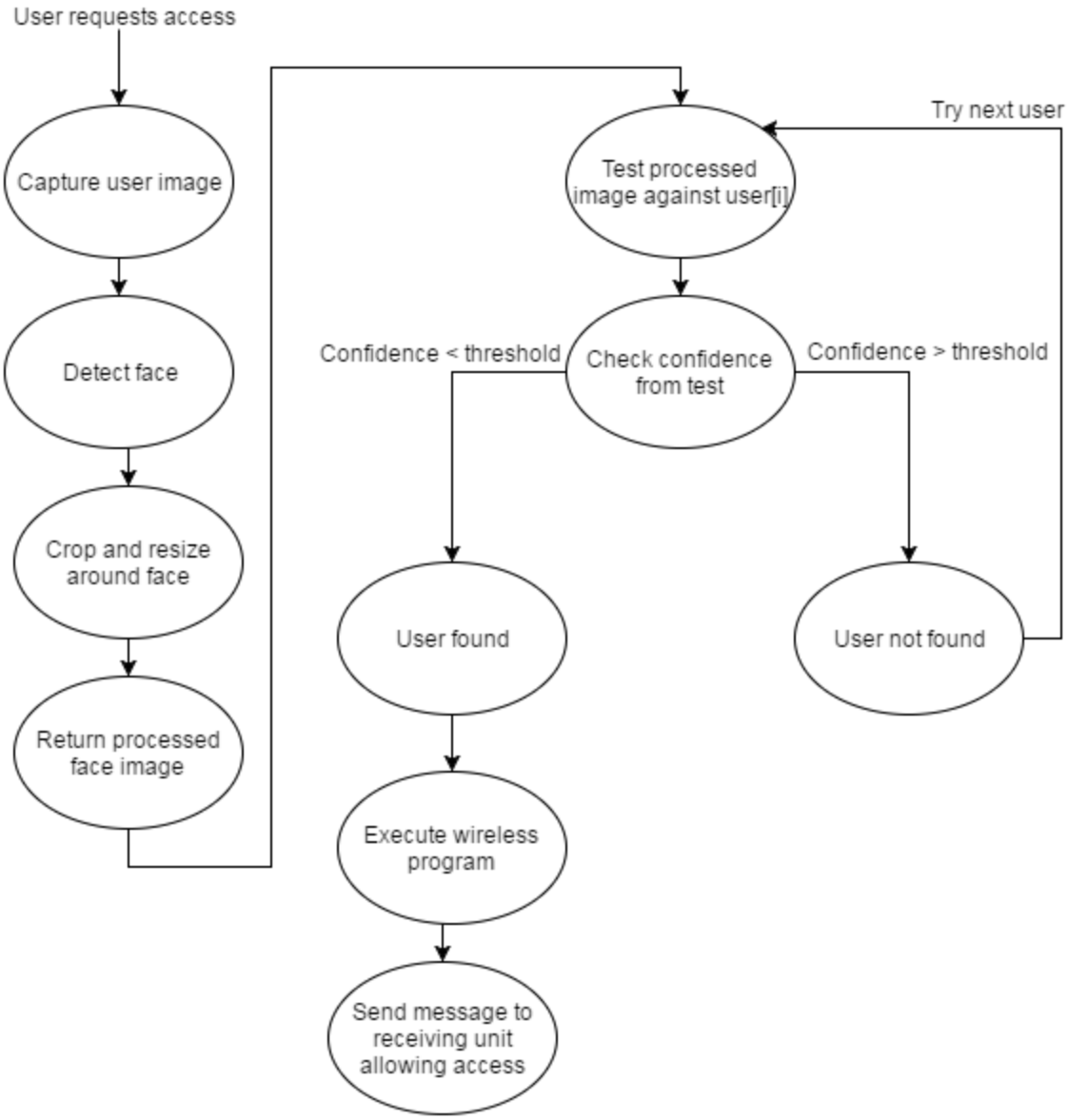

Figure 16: Flowchart of control flow in requesting access software process.

When a user wishes to enter the building, he will enter ‘e’ into the command line user interface to initiate the process. Figure 16 provides a flowchart for the different steps when a user requests access.

As shown in the flowchart, the first step after a user initiates an access request is to capture an image of the user. Just as with the facial recognition training process, each captured image that is used for recognizing a user must be resized to the same size as the training set, which is 92x112 pixels. After a picture is captured of the current user, it is processed before passing it into the facial prediction API provided by OpenCV for the EigenFaceRecognizer class.

The first step in processing an input image of a user is to detect if there is a single face in the image. This process uses the same Haar cascade algorithm as outlined in Section 3.4.2 for face detection. If a face is detected, the rectangular region of interest is returned and used to crop the image to focus on the user's face. After cropping, the image is resized to 92x112. Once the image has been processed appropriately, it can be used to make a recognition prediction.

The next part of the recognition program contains a loop that iterates through each user that is registered to the DeliverEZ system. On each iteration, an OpenCV API is called that takes in the processed image and a predicted result. The API then compares the processed image to the predicted result's facial recognition model and returns a confidence value. In OpenCV's facial recognition framework, a lower confidence value returned from the prediction API equates to a more confident prediction. Therefore, if the prediction API returns a confidence value of 1000 for one case and 3000 for another case, the 1000 case is said to be more confident and will more likely be a recognition match.

As part of the configuration for DeliverEZ, a confidence threshold is defined for use throughout the application. If the confidence level is lower than the threshold, meaning the API is confident in its facial recognition prediction, then a boolean variable is set denoting that the user has been found. When a user has been recognized, the program breaks out of the loop that iterates through the registered users as it no longer needs to search for a possible user.

Once the program is out of the loop, it is time to send a wireless message to the receiving unit announcing that a user has been granted access to the building. To send the message, this functional module calls an executable that contains the wireless transmission program. This executable then sends a command to the receiving unit alerting it to activate the resident's buzzer and open the door.

3.4.4 Main Processing Unit Wireless Transmission

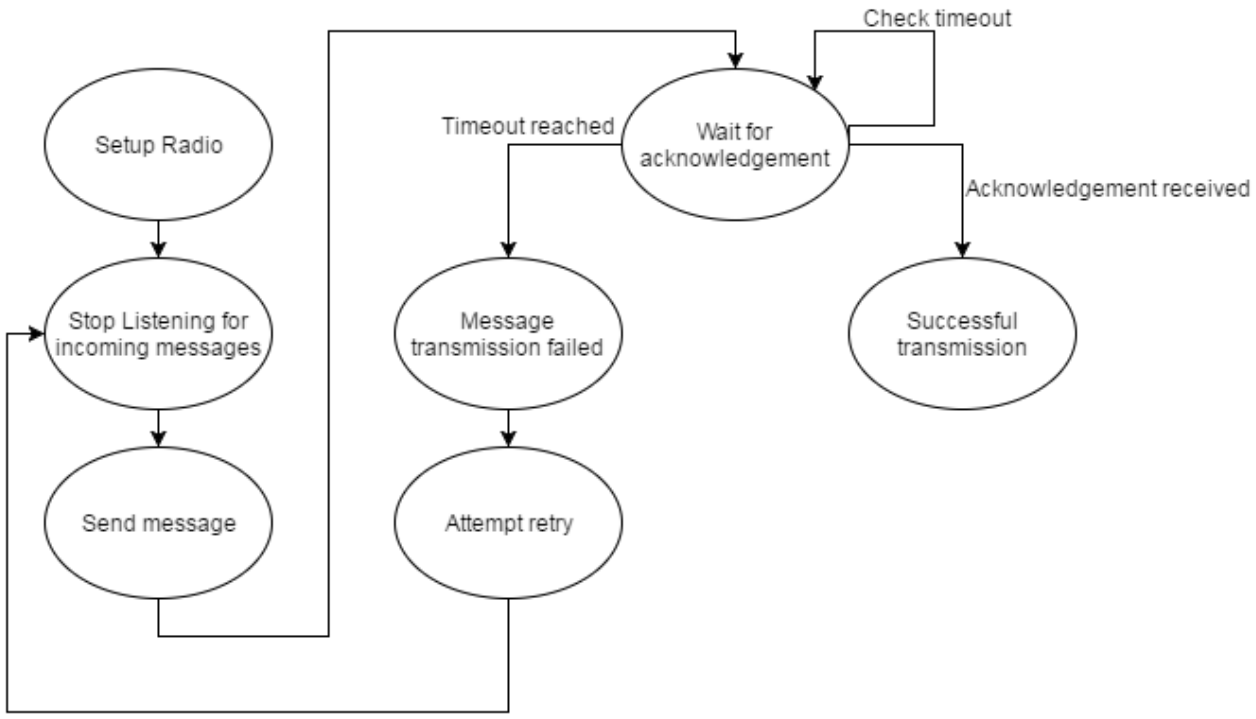

Figure 17: Flowchart of software design for wireless transmission application.

The wireless transmission application is written in C++ following the ping / pong example on the Nordic RF24L01 transceiver library github page provided by maniacbug [8]. The application was modified to fit the needs of the DeliverEZ system. Figure 17 provides a flowchart for the wireless transmission application software design. When the wireless transmission application is executed, the first step is to set up the Nordic RF24L01 radio with the correct settings for this system. Two pipes are opened: one pipe is for reading incoming messages from the receiving station, and another pipe is for writing data to the receiving station. It is imperative to match these two pipe names between the receiving and transmitting radio applications so that the radios can communicate between each other correctly. Additionally, the number of retries are Raspberry Pi board numberings are defined so that the radio transceiver knows the correct SPI pins to use on the Raspberry Pi.

After initializing the radio, the main radio application attempts to send a message. This portion of the application calls a send message function that takes in a command as its argument that the system wishes to send. The command that is passed into this function is the “access” command, which tells the receiving unit to activate the buzzer. A boolean is returned from the send message function that tells the application whether or not the message was successfully received by the receiving unit in the resident’s

apartment. If the send message function returns false, the main application will attempt to retry a certain number of times. The number of retries can be defined by the user based on the user's environmental constraints.

The send message function takes in an action as its argument and converts this into an unsigned long type to prepare for transmitting the action in the message payload. Additionally, an API is called at the beginning of the send function to tell the radio to step listening as it prepares to send the message. Once the radio has stopped listening, an API is called to send the message payload. The send API returns a boolean status that tells the application if the program was not able to communicate with the radio to send the message, which could indicate a potential hardware or wiring issue.

After sending the message and checking the send status, the radio is instructed to begin listening again in order to hear the acknowledgement from the receiving unit. If a message is received successfully, and the receiving unit activates the buzzer, it will send a message with a success status code back to the main processing unit. The main processing unit listens for this message, and if it receives the expected success status code, it will return success back to the main application and no further message transmission attempts will be needed. If the expected status code is not received, the application will return false and the main application will attempt to retry until the user defined retry limit is reached. At this point, the wireless transmission application completes execution and returns back to the main DeliverEZ application loop.

3.4.5 Web Server

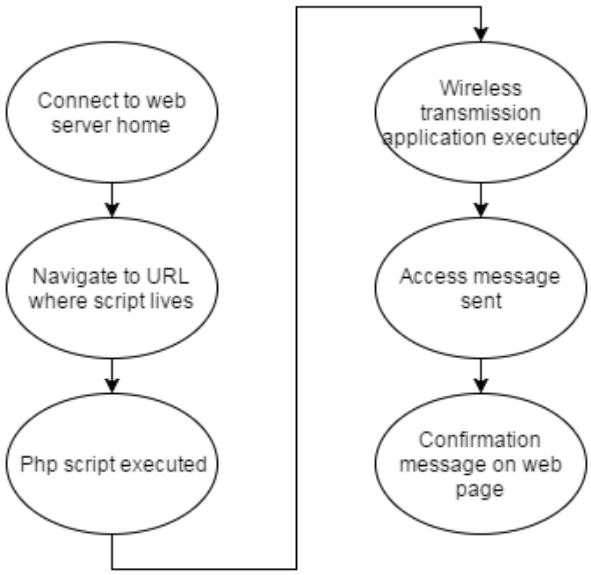

Figure 18: Flowchart of software design processes in web server application.

The Apache2 web server is configured to execute the wireless transmission application when the appropriate php script is requested from the server. Figure 18 gives a flowchart of the software design used for the web server. Hosted on the home IP address of the web server is a simple homepage that welcomes the user to the page. In order to force a wireless transmission to the receiving unit to activate the buzzer, the user must enter the URL that contains the path to the php script that sends the message. When the user executes this script, the status of the transmission is outputted onto the web page to alert the user of a successful or failed wireless transmission.

In order to execute a script on the Apache2 web server, it must be configured to allow dynamic content with the Common Gateway Interface (CGI). Using CGI, the web server will execute the code in the script instead of just printing the script contents to the webpage. This is a powerful tool in web programming. By modifying the Apache2 configuration file, we explicitly declare that CGI programs are allowed to be executed and, for additional security, define a path to the CGI program's location, which tells Apache2 that this is the only location in the Raspberry Pi's filesystem that is allowed to have executable scripts on the web server.

Although the wireless transmission program is written in C++, Apache2 cannot call this executable directly. A workaround was implemented to solve this issue. Instead of having the web server call the executable itself, a PHP script was written that uses the PHP API to execute an external program. When the web server executes the PHP script, PHP will then execute the C++ executable and print its output to the webpage. This workaround was used to successfully execute the C++ wireless transmission application to send access commands to the receiving unit.

3.4.6 Arduino Receiving Unit

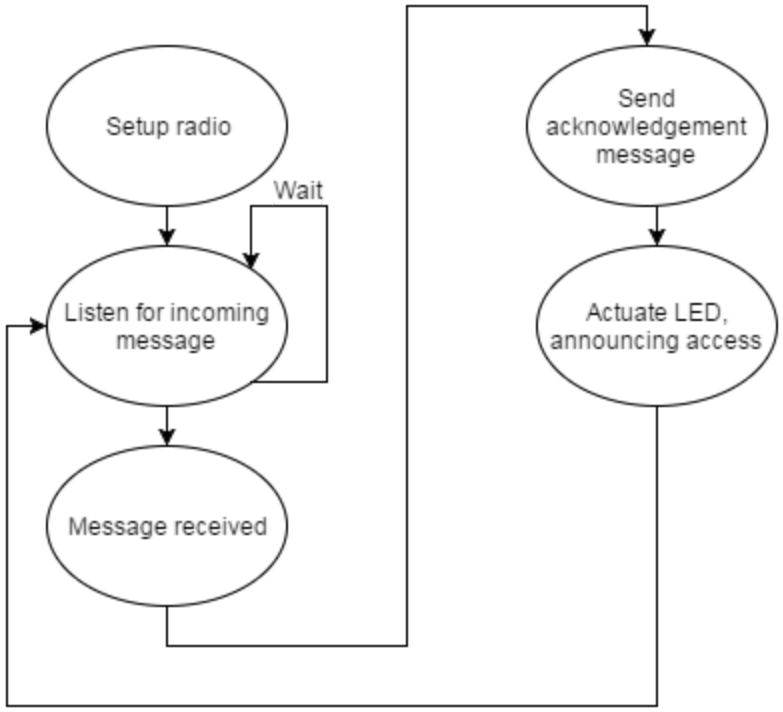

Figure 19: Flowchart of wireless receive application software design running on Arduino.

The receiving unit constantly runs the same Arduino sketch, which waits for incoming messages from the main processing unit. This sketch is also modified from the ping / pong example provided by maniacbug [8]. Figure 19 provides a flowchart for the Arduino sketch. Just as with the transmission application, the radio is initialized with the correct settings, reading pipe, and writing pipe. The reading pipe and writing pipe must match the appropriate pipe names from the main processing unit in order for the two radios to communicate with each other successfully.

After initialization, the Arduino receiving sketch enters an infinite loop that continually calls the API to listen for incoming messages. While listening, if the read API returns true, meaning the radio received a message, the radio is told to stop listening and the received message is digested and saved to memory. Once the message contents are digested, the Arduino attempts to send an acknowledgement to the main processing unit to alert it that the transmission was successful. This provides a handshake mechanism to increase the reliability of the wireless system in DeliverEZ. The final step after receiving an access command from the main processing unit is to actuate the buzzer. Since there is no buzzer in this prototype system, an LED is used to give a visual indicator of when the buzzer would be activated.

When it is time to activate the buzzer, a function is called that blinks the LED and keeps it illuminated for a user defined amount of seconds. Once the visual cue has completed, the receiving unit is instructed to listen for incoming wireless messages once more.

4 Testing Strategy

After developing the DeliverEZ system for this design project, there were various metrics that were used to test the effectiveness of the system. The most important metric in a facial recognition system, accuracy, was thoroughly tested with different parameters to determine how effective DeliverEZ was at recognizing allowed and restricted users. Additionally, a major design comparison point in this design project was comparing the Python and C++ implementations of DeliverEZ. The execution time of different computationally intensive facial recognition operations was compared between Python and C++. Also included in the comparison between the two software implementations, hardware resource constraints were used for comparison. Being an embedded platform, resources can quickly be consumed on the Raspberry Pi for large applications such as DeliverEZ.

4.1 Accuracy

In order to determine how accurate the DeliverEZ system was, there were various experiments that were done. As the prediction algorithm is not an exact, binary output algorithm, part of the machine learning process for facial recognition is to give an indication of how sure you are of the result. As discussed in Section 3.4.3, a confidence threshold is used to tell the system if we are going to accept the prediction as truth or not. This process requires tuning to determine the best confidence threshold. The best confidence threshold will give the lowest number of false positives and false negatives. False positives are defined as when the facial recognition prediction believes it has found a match when it actually has not. False negatives occur when a registered user is not recognized by the system, even though he should be. By changing the confidence threshold, the number of false positives and negatives will change. The accuracy was tested using confidence thresholds of 1500, 2000, 2500, 3000, and 3500. From the results of this experiment, an effective prediction threshold to use in the rest of the system's experimentation should become clear. Sensitivity and Specificity were calculated for this experiment with Sensitivity defined as the probability of a registered user granted access and Specificity as the probability of a non-registered user denied access. For this experiment, the number of positive images used in training was fixed at 10 positive images.

A known drawback with the Eigenface facial recognition algorithm is its susceptibility to different lighting conditions. If the user trained his facial recognition model in a specific lighting condition, that lighting will be carried over and saved as part of his model. For example, if the user had a spotlight on the right side of his face, the training process would have saved this illumination as part of the user's face. Therefore, when performing the recognition, the Eigenface algorithm will look for that illumination as a feature of the user's face. To test how susceptible lighting conditions are to DeliverEZ, the confidence value of the prediction algorithm was analyzed when using different lighting conditions in the image under test. The model used for testing lighting susceptibility was trained using 10 positive training images for this experiment.

The confidence level determined by the prediction algorithm is also dependent on how robust the trained facial recognition model is. By using more training images with different lighting, facial expressions, and angles, the model will be improved. As an additional metric for testing DeliverEZ's accuracy, the confidence value from the prediction algorithm was analyzed as a factor of the number of positive training images used. The number of positive training images used was varied from 1, 5, 10, and 20 images. The expectation is that the confidence will improve (a lower numerical confidence value) with an increased number of positive training images.

As a final metric for evaluating the system's accuracy, different facial expressions and their respective confidence was explored. A dataset of 10 positive training images was used to train the facial recognition model used in this experiment. The training images had varied facial angles with a relatively constant facial expression. Both eyes were always open, and the training face's mouth was always closed. During the experiment, different test images were captured with different facial expressions of a registered user. Each facial expression's resulting confidence level was noted for later analysis.

4.2 Execution Time

To compare the Python and C++ implementations of DeliverEZ, different execution times were measured for computationally intensive tasks. There were 4 major tasks whose execution speeds were noted while testing the system. First, being the most computationally intensive task, the facial recognition model training algorithm's execution speed was analyzed. The time that it took to build the facial recognition model for varying number of input images should give an indication of the computational cost of having a more robust model with more training images. Additionally, loading the models into memory can take a significant amount of time and will increase with a larger number of users in the system. Detecting a face and making a recognition prediction are two other major tasks performed in DeliverEZ whose execution time was worth exploring. However, these two operations are much less intensive than

training and loading facial models. Each task's execution time was measured for both the Python and C++ implementations in this experiment. The initial hypothesis for execution time is that the C++ implementation should have higher performance and complete its tasks in less time than the Python version.

4.3 Resource Constraints

The last major testing metric for this design project was the resource constraint of the system for both the Python and C++ implementations. Processor load and memory footprint were the two resources that were analyzed in the resource constraint experiment. Even though the Raspberry Pi 2 has 1 GB of memory, which is substantial for an embedded system, this is much smaller than a middle range computer available for consumers. Additionally, the CPU on the Raspberry Pi 2 is similar to a mobile device from 2014 [12], which could hamper the performance of the facial recognition algorithms.

To test processor load, intensive operations, such as model training and loading, were run while taking note of the system's state as the operations progressed. Memory usage should increase as the number of registered users and their respective facial recognition models increase in the system. Keeping track of more facial models will require more memory resources and could eventually reach an upper limit of the number of possible users in the system. Additionally, there should be a distinct difference in memory usage between the Python and C++ versions of DeliverEZ.

5 Results and Discussion

5.1 Accuracy

5.1.1 Confidence Threshold Variation

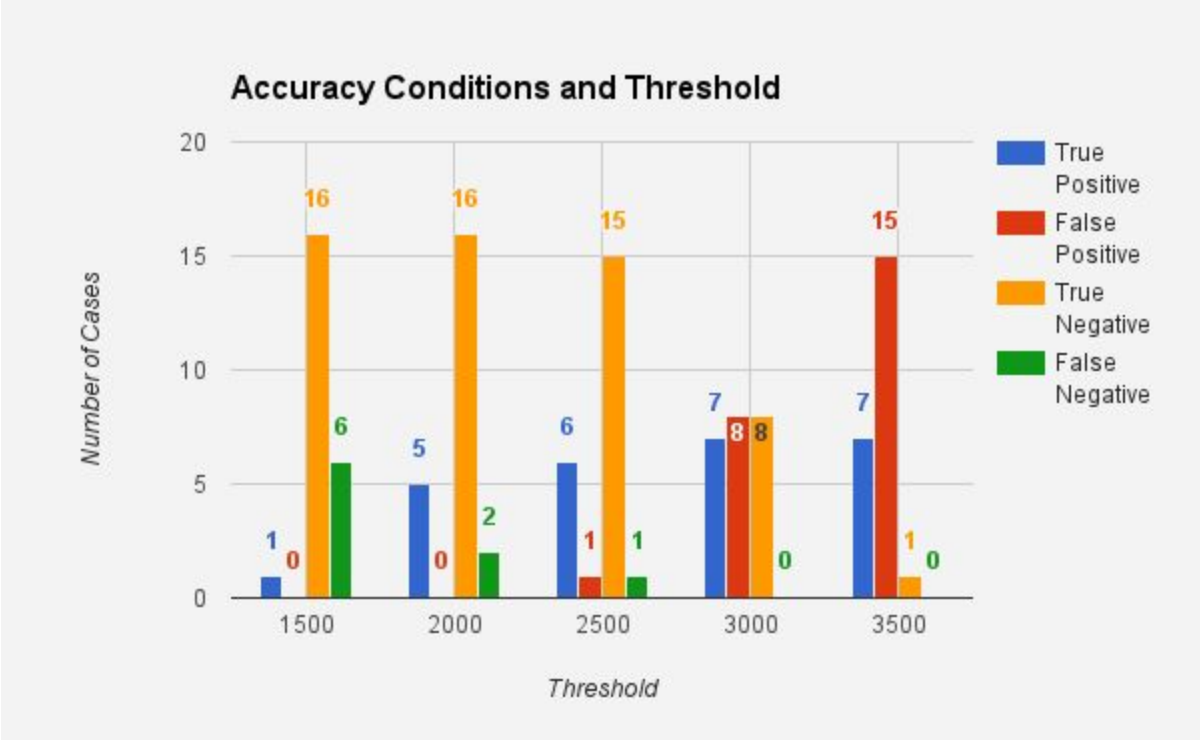

Figure 20: Results showing accuracy metrics when using different prediction thresholds.

Figure 20 provides results for the accuracy threshold testing. In this experiment, multiple subjects' faces were tested with the DeliverEZ system. Shown in the results are the number of true positive, false positive, true negative, and false negative cases using the same set of images but changing the confidence threshold that determines a recognized face or not. In the case of using a threshold of 1500, it can be seen in the results that only 1 true positive case occurred. Six cases of registered users not being allowed access, false negatives, existed with a threshold of 1500. Therefore, a threshold of 1500 is likely too strict for this application. That many false negatives would likely cause frustration with the system.

The best performing threshold, in terms of highest security, is a threshold of 2000. This case has a sensitivity of 71.43% and a specificity of 100.00%. Therefore, with this specificity, an unregistered user was not granted access. However, 2 registered users were still denied access, which could cause some frustration. These denied registered users would be required to repeat the access procedure in order to gain access as the confidence values can vary over multiple cases.

A threshold of 2500 also performed adequately. This case had a sensitivity of 85.71% and specificity of 93.75%. Even though this test had fewer false negatives than a threshold of 2000, a false positive case emerged, which dropped the specificity from 100% in the 2000 case to 93.75% in this case. In order to provide the best security for the system, false negatives should be eliminated. Therefore, a threshold of 2500 is not an acceptable threshold. Either a threshold of 2000 must be used and the small number of false negatives exist, or a smaller increase of thresholds should be chosen from 2000, such as 2100 or 2200.

Thresholds of 3000 and 3500 were the poorest performers with sensitivities of 100% and 100% and specificities of 50% and 6.25%, respectively. Even though both cases had no false negatives, thus sensitivities of 100%, the high rate of false positives is not acceptable for this system. The 3000 case allowed access to 8 unregistered users, and the 3500 case allowed access to all but one tested user. From these experimental results, a threshold value between 2000 and 2500 was determined to be the most effective.

5.1.2 Environment Illumination Variation

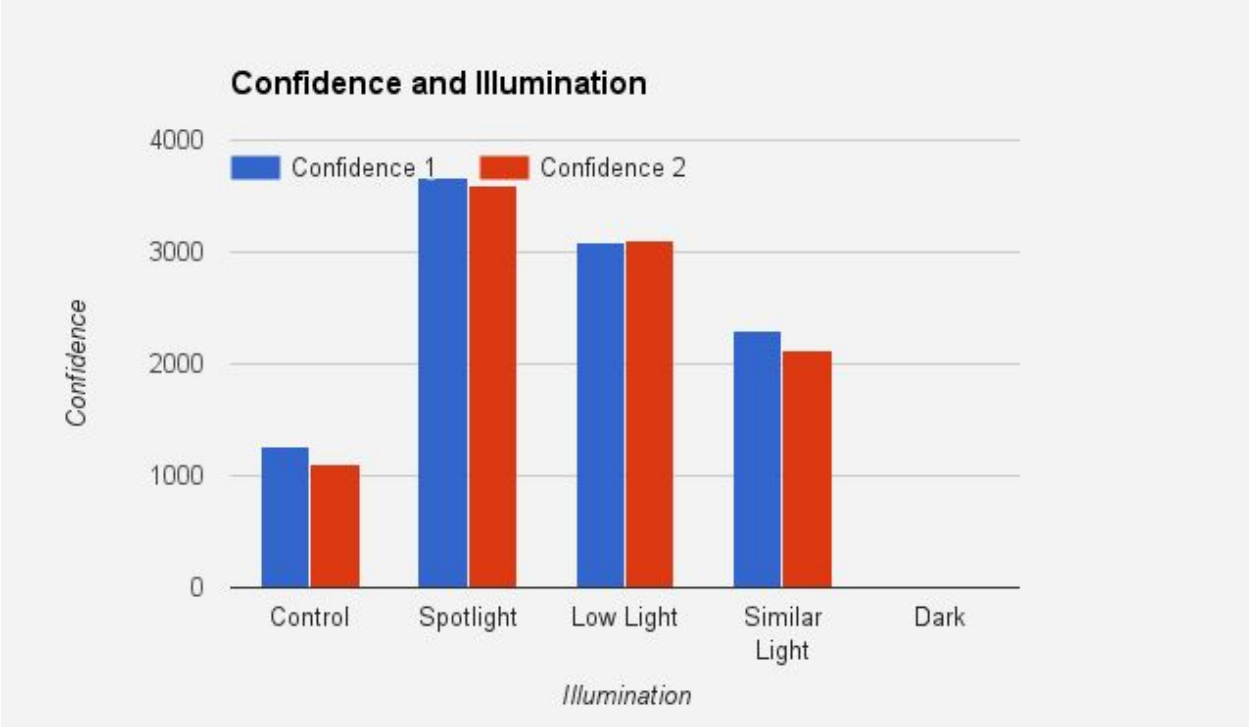

Figure 21: Results showing the change in prediction confidence under different lighting conditions.

As previously discussed, the Eigenface algorithm is susceptible to different illumination conditions as the lighting environment is built into the facial recognition model. As shown in Figure 21, different lighting conditions and their respective prediction confidence values were tested and recorded. Discussed previously, a lower confidence value represents a more confident prediction. As a baseline, the Control case is a lighting environment in exactly the same conditions as existed for training the facial model. Looking at the results, there is a large discrepancy in the face prediction confidence value for different illumination cases. In the worst case, the Dark experiment shows no results as the algorithm was unable to detect a face in a dark room. Therefore, no face was captured for making a face prediction.

The three other lighting conditions tested were a spotlight on the user's face with a flashlight, a low light room, and a room with similar lighting to the training set. The Spotlight condition provided the worst results compared to the control case. With confidence values greater than 3500, the Spotlight lighting condition would yield a false negative prediction even with the highest tested threshold value from the results in Section 5.1.1. Therefore, this case would be unacceptable for an effective facial recognition application.

The Low Light environment also yielded unacceptable confidence results. With both tests in this environment giving a confidence value of over 3000, the prediction algorithm would generate false negative predictions even if a threshold of 3000 is used, which is a higher than desired threshold to begin with. With a similar lighting condition to the training set, the Similar Light case should provide confidence values close to that of the Control case. However, there is still a substantial discrepancy as the confidence values in this case are slightly above 2000. If a prediction threshold of just over 2000 was used in this system, then the prediction algorithm with yield true positives for these faces, which could be acceptable for the system. Looking at the results in Section 5.1.1, 1 false positive case existed with a threshold of 2500. Therefore, care must be taken to choose a threshold that does not give access to any users it should not.

5.1.3 Confidence and Size of Training Set

Figure 22: Results testing the confidence value vs. the number of positive training images used.

Increasing the number of positive training images in the training set is a known benefit of a machine learning application. For the DeliverEZ application, the prediction confidence and effectiveness was analyzed with different size training sets to train the facial model. The facial model was trained with 1, 5, 10, and 20 positive images in a controlled lighting and facial expression environment. As shown in

the Figure 22's data, there is a significant change in confidence due to the facial model being more robust with a larger data set. Going from 1 positive image to 10 provides a substantial benefit to confidence, since a lower confidence equates to a more confident prediction. However, because more training images take up a larger amount of space in the system and increase the time of training the user model (Section 5.2.1), care must be taken to choose the correct training set size. The desired training set size will provide the best confidence increase without taking a large toll on system size and execution speed. Therefore, based on this experiment, a training set size of 10 positive images was used as the default for other experimentation and system execution.

5.1.4 Confidence and Facial Expression

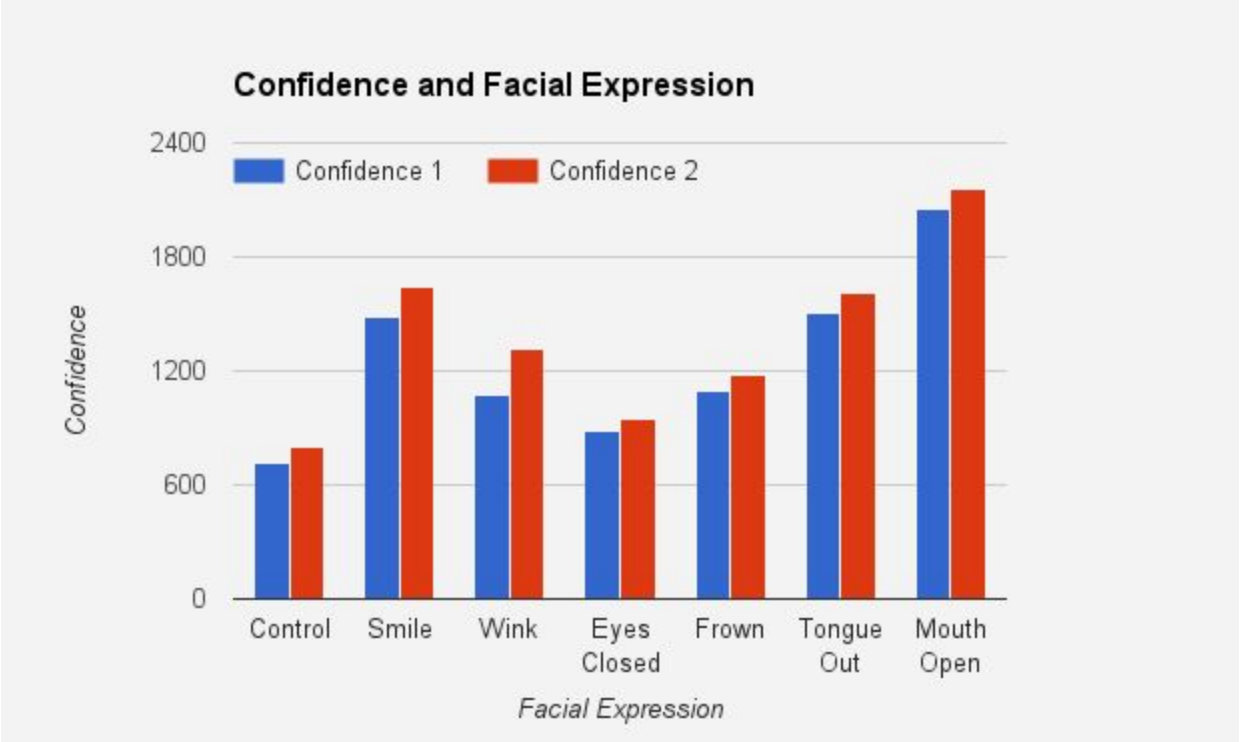

Figure 23: Graph illustrating confidence value changes with different facial expressions.

Illustrated in Figure 23, the resulting confidence values for the experiment varying the user's facial expression are shown. Two confidence values were taken for each facial expression in order to ensure that the confidence values were consistent and no outliers negatively affected the data. A Control case is given to give a comparison between a best case confidence value and those of varying facial expressions. As discussed previously, a lower confidence value equates to a more confidence prediction by the facial recognition prediction algorithm.

The data provides some interesting conclusions about the accuracy of the facial recognition prediction for the Eigenface algorithm. Initially, one would think that closing one or both of the user's eyes would give a significant worsening of confidence. However, looking at the data, eyes closed and wink were two of the better performing facial expressions. Eyes are significant features on the human face. From these results, it looks as if the Eigenface algorithm is more concerned with the presence and location of the eyes rather than the detailed features of each individual eye. As discussed in Section 2.2.1 about the Haar features used in DeliverEZ, the eye features of a face are simply detected as darker regions from being deeper than other parts of the face. This provides conclusion to the results as even though the one or both eyes were closed in this test, these areas of the face would still be darker and detected as eye regions.

Another facial expression with a small worsening of confidence was the frowning face. A frown more or less keeps the same shape and size of the human face. The main facial feature changing in a frown is the position and shape of the lips. As the algorithm would still be able to detect human lips, there is not an overly large change in confidence for a different lip position in a frowning versus a straight face.

The worse performing facial expressions, with a high confidence value of 2163, were facial expressions that greatly changed large features of the human face. Smiling, for example, separates the lips and exposes the teeth. These features were not included in the training set for this experiment. Thus, these large features deteriorated the accuracy of the facial recognition prediction algorithm. Additionally, the facial expressions Mouth Open and Tongue Out also greatly changed the large features of the human face. Opening the mouth, with and without the tongue out, changes the shape and size of the human face since the jaw line moves down. Having the mouth open also provides a large dark area in the image where the open mouth is. Since this large dark feature does not exist in the training set, the prediction algorithm gives a much lower confidence value than would be normally acceptable. For a confidence threshold of 2000, which is an empirically defined appropriate confidence, the mouth open facial expression would give a false negative prediction and not allow a registered user access.

5.2 Execution Time

5.2.1 Training the Facial Model

Figure 24: Graph comparing the time to train the facial model versus the number of images used for Python and C++ implementations.

As shown in Figure 24, the number of positive images used in the training image set has an impact on the execution time for training a facial recognition model. This follows the initial hypothesis that increasing the size of the training set also increases the time it takes for the system to train the model. For both the Python and C++ implementations, the execution time increases linearly versus the number of positive images in the training set.

Interestingly, the C++ implementation has a larger execution time than the Python implementation for training the facial recognition model. Over time, the two implementations execution time begins to become equivalent. However, the initial hypothesis for this design project was that the C++ implementation would have higher performance, thus shorter execution times, than the Python implementation for each tested process. As will be discussed throughout the results section, the execution times for both the Python and C++ implementations were largely the same. Although C++ has faster

execution than Python in theory, the OpenCV framework for Python really just calls the underlying C++ functions for each of the processes tested. Therefore, after performing experimentation, both the Python and C++ implementations were deemed to be similar or equivalent in terms of execution time.

5.2.2 Loading User Models

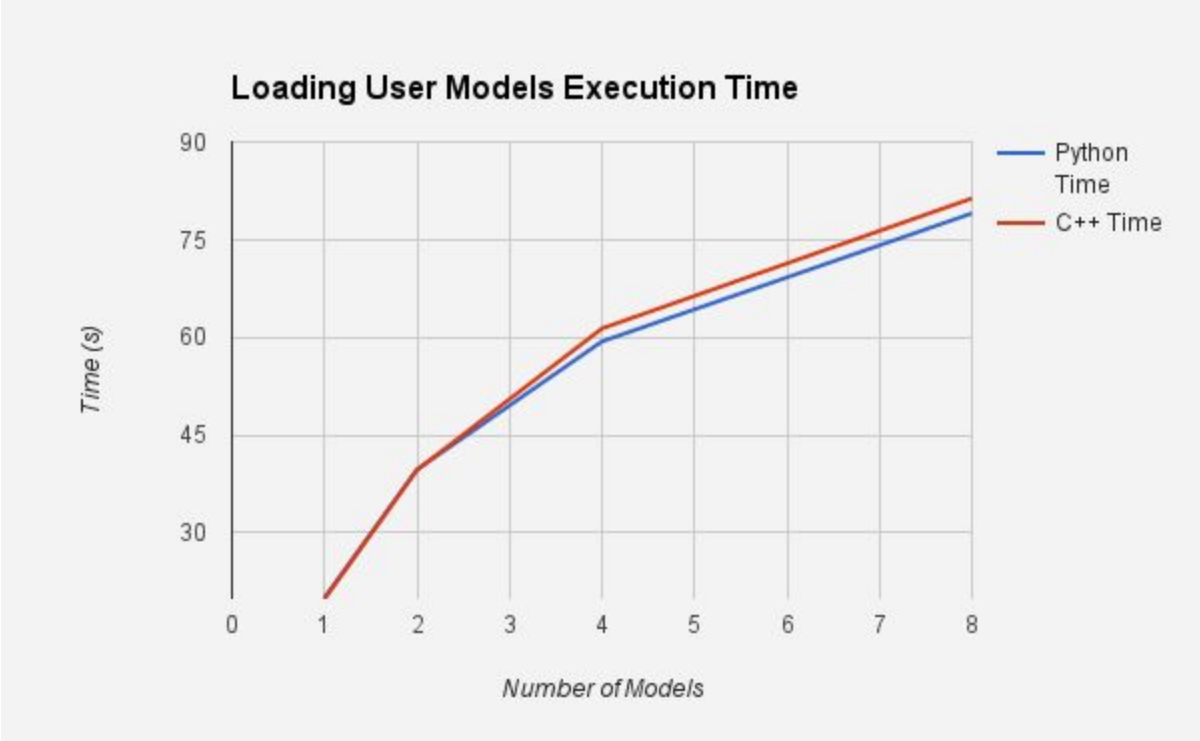

Figure 25: Execution time to load user models for Python and C++ implementations.

Figure 25 shows how the number of user models in the DeliverEZ system affected the time it took to load these models into memory. The resulting data shows a linear increase in execution time versus the number of user models loaded. This result is expected as the process to load user models is done sequentially and one at a time. As discussed earlier, the Python and C++ implementations have equivalent execution times for OpenCV processes. This conclusion is illustrated by the data from this experiment.

5.2.3 Face Detection

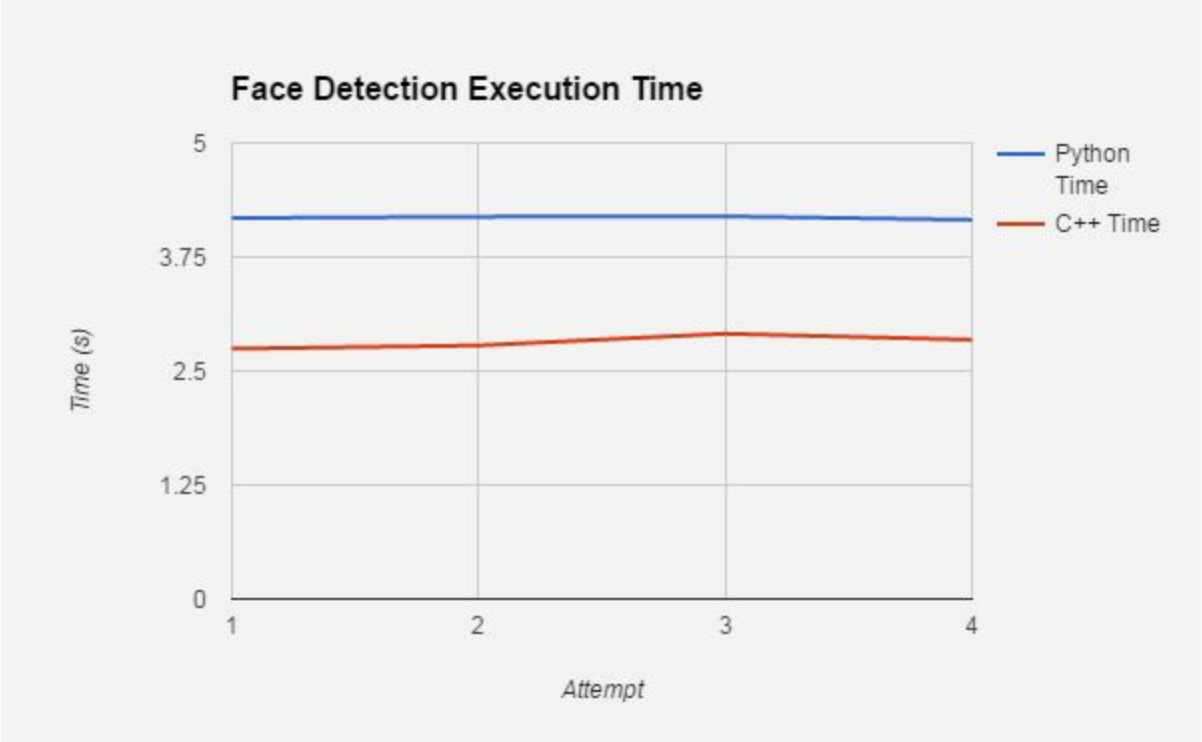

Figure 26: Face detection execution time for Python and C++ implementations.

The execution time for the face detection algorithm is shown in Figure 26. As can be seen in the figure, each attempt for each implementation has largely the same execution time. Thus, the face detection algorithm had a consistent execution time even for multiple attempts. Face detection used a combination of different OpenCV functions and native language constructs. As shown in the results, the C++ implementation was consistently slightly faster than the Python implementation. Although the two execution times are expected to be equivalent for OpenCV functions, other aspects of the face detection algorithm that used native language constructs attributed to C++ having a faster detection time than Python.

5.2.4 Face Prediction

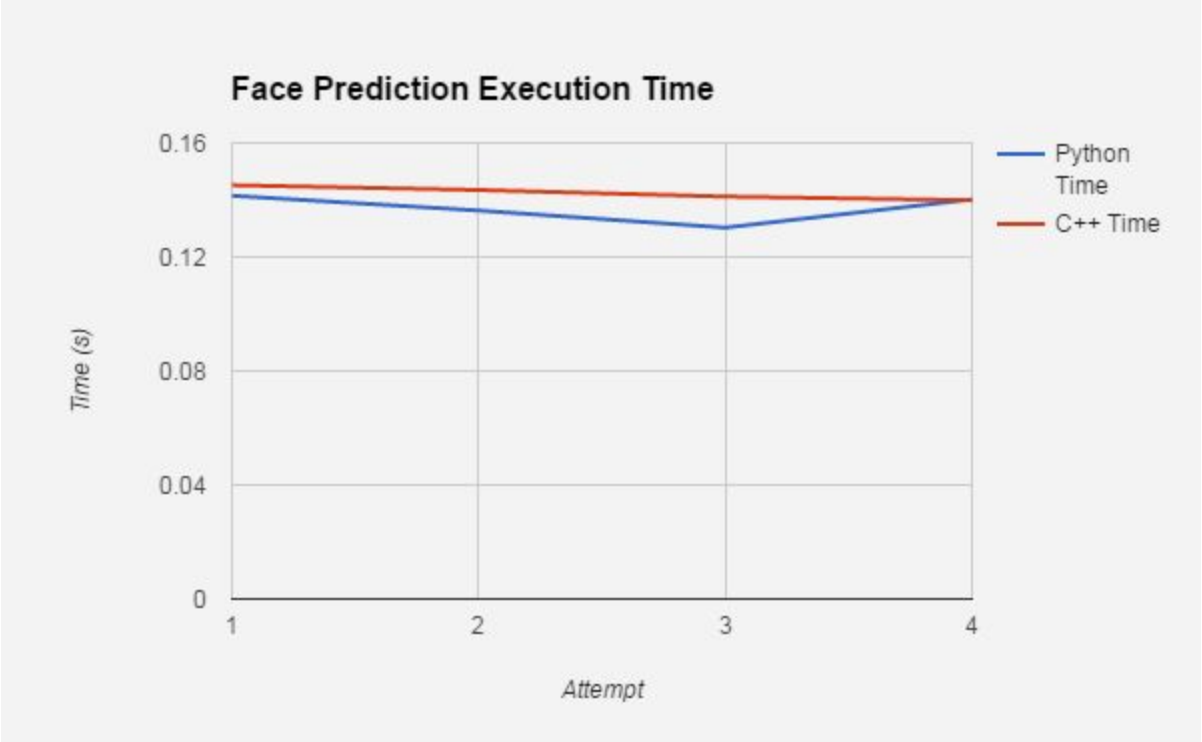

Figure 27: Execution time for prediction algorithm for Python and C++ implementations.

The face prediction algorithm was another OpenCV function that gave the same execution time for both Python and C++ implementations. As shown in Figure 27, the execution time for facial prediction was consistent and near equivalent on multiple attempts for each design. These results further strengthen the conclusion that OpenCV functions have equivalent execution times regardless of the language being used. Even though the Python implementation was hypothesized to be slower than C++, the OpenCV functions in Python execute the same underlying C++ function that the C++ design used. Thus, their execution times are equivalent.

5.3 Resource Constraints

5.3.1 Memory Footprint

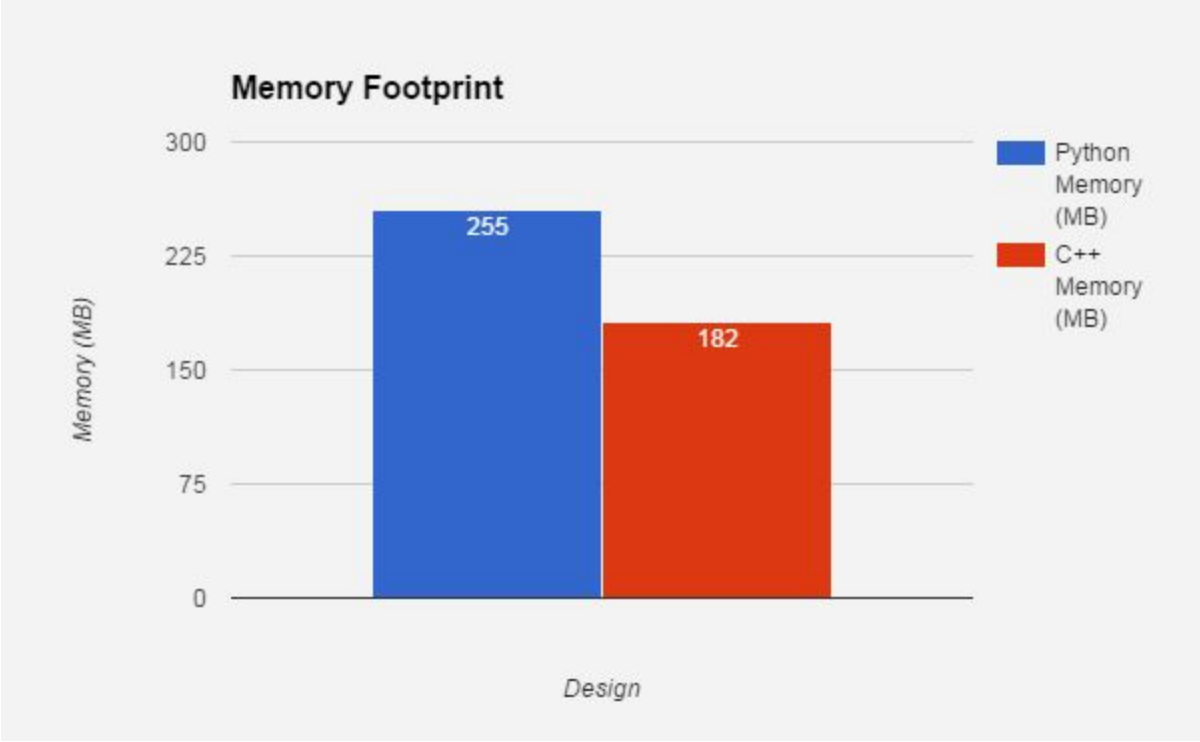

Figure 28: Memory footprint comparison between Python and C++ designs.

Figure 28 shows the memory footprint while running the DeliverEZ system for both the Python and C++ versions. The system, in both implementations, is currently waiting for user input (idle state) and has a total of 5 users registered with their facial models loaded into program memory. As can be seen in the figure, the C++ design has a smaller memory footprint than the Python design even with the same number of users in the system.. This follows the hypothesis that C++ should consume less memory than Python while running the DeliverEZ application. The main reason behind this result is that Python, in general, consumes more memory while running an application. Because it is a dynamic, interpreted language, Python cannot benefit from compiler optimizations like C++ can. Therefore, its dynamic nature forces Python applications to consume more memory than equivalent C++ applications.

For an embedded system, memory usage is a big design point. On the Raspberry Pi 2, there is 1 GB of RAM available. If a developer is creating a large application that needs to use lots of RAM, he

will want to choose a development framework that minimizes the memory footprint on the embedded platform. As the number of registered users and user models increases in DelivereEZ, the amount of RAM required for the application will also increase. Therefore, a C++ version of DeliverEZ can accommodate for a larger number of users in this facial recognition system.

5.3.2 Processor Load

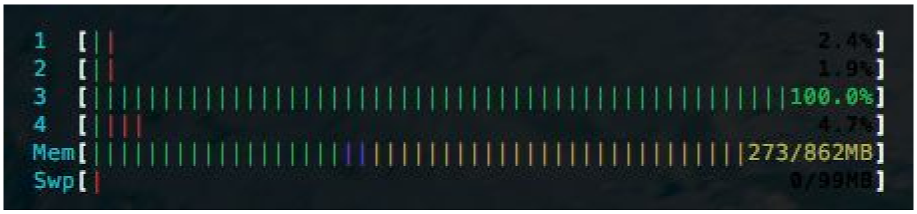

Figure 29: Screenshot of *htop* process viewer application showing full CPU load on CPU 3.

Figure 29, shows a screenshot of the *htop* process viewer application on the Raspberry Pi while running the training algorithm. As seen in the image, CPU 3, is at full load (100%) during this process. Also visible, CPUs 1, 2, and 4 are mostly unused by the system. The training algorithm is the most computationally intensive process in the DeliverEZ system. From the results found in Section 5.2.1, the training algorithm takes over 2 minutes even in the fastest case. Thus, during this time, one of the 4 cores is completely loaded. Fortunately, the Raspberry Pi 2 is equipped with a quad-core processor so other processes running on the system are free to use the available 3 cores.

The single processor at full load is a drawback of the DeliverEZ software implementation. A possible improved design would be to take advantage of the multi-core processor and parallelize the computationally intensive algorithms such as training and loading user face models. Distributing the computation across the 4 cores could improve the execution time for these processes, thus improving the system's user experience. However, the process in question must be parallelizable in order to take advantage of the multiple cores.

6 Conclusions

This design project, DeliverEZ, accomplished its goal of developing two implementations of a multiple user facial recognition system. On top of the facial recognition application, wireless communication between the main processing unit of the Raspberry Pi and receiving unit of the Arduino was achieved. Additionally, a web server was successfully created to allow the user to bypass facial recognition for special circumstances.

The Python and C++ implementations also underwent thorough testing for accuracy, execution time, and resource constraints. From the accuracy experiments, the susceptibility of the EigneFace algorithm to different lighting environments, training set sizes, and facial expressions was explored. These weaknesses in the facial recognition algorithm are important to thoroughly understand when creating a robust system for use in the real world as the same constraints used in DeliverEZ will not always be applicable in every case. However, for the scope of this design project, the experimental results validated the success of the project.

Although the initial hypothesis that the C++ implementation would out perform the Python implementation in terms of execution speed was not found to be true, developing the system in two languages provided experience in architecting and developing large applications for multiple languages. A major benefit of reaching the conclusion that the Python and C++ implementations performed equivalently for execution speed is that developers wishing to implement a system using OpenCV are not restricted to certain languages in order to make the best system. Especially for the Raspberry Pi, where Python is one of the official languages supported, a developer can expect great results when using OpenCV.

The DeliverEZ system's accomplishments gave exciting results and expanded upon the possible facial recognition systems that can be implemented on embedded devices. As current delivery services are unable to complete deliveries for many customers in apartments without a concierge, DeliverEZ provides a possible solution. By allowing delivery services to complete their deliveries the first time, major inefficiencies and frustrations are taken out of the delivery process. This has the possibility of saving time and money for both the delivery services and their customers.

7 Acknowledgements

I would like to thank Dr. Bruce Land for all his help and guidance throughout my design project. His wealth of knowledge, constant availability for questions, and excitement about my work made this design project a truly rewarding experience. I would also like to thank Professor Joseph Skovira for inspiring my initial interest in the Raspberry Pi. Finally, I would like to thank my parents, Sylvain and Rosemary Palmer, for their constant support throughout my entire career at Cornell University.

8 References

[1] Brunelli, R., Poggio, T. *Face Recognition through Geometrical Features.* European Conference on Computer Vision (ECCV) 1992, S. 792–800.

[2] "The Database of Faces." *The Database of Faces*. AT&T Laboratories Cambridge, 2002. Web.

[3] Dicola, Tony. "Raspberry Pi Face Recognition Treasure Box." *Training | Raspberry Pi Face Recognition Treasure Box | Adafruit Learning System*. N.p., 24 Jan. 2014. Web.

[4] Erel. "RF24." *B4R Library - RF24 - NRF24L01 Radio Modules | B4X Community - Android, IOS, Desktop, Server and IoT Programming Tools*. N.p., 17 May 2016. Web.

[5] "Face Recognition with OpenCV." *Face Recognition with OpenCV — OpenCV 2.4.13.1 Documentation*. Web.

[6] Greenough, John. "How the 'Internet of Things' Will Impact Consumers, Businesses, and Governments in 2016 and beyond." *Business Insider*. Business Insider, 18 July 2016. Web. 07 Dec. 2016.

[7] Kanade, T. *Picture processing system by computer complex and recognition of human faces.* PhD thesis, Kyoto University, November 1973

[8] Maniacbug. "Maniacbug/RF24." *GitHub*. N.p., 03 Oct. 2013. Web.

[9] Morgan, Jacob. "A Simple Explanation Of 'The Internet Of Things'" *Forbes*. Forbes Magazine, 14 May 2014. Web.

[10] "OpenCV | OpenCV." *OpenCV | OpenCV*. 2016. Web.

[11] "OpenCV: Face Detection Using Haar Cascades." *OpenCV: Face Detection Using Haar Cascades*. Web.

[12] Ricknäs, Mikael. "Hands On: The Raspberry Pi 2 Is Powerful, but You Still Get What You Pay for." *PCWorld*. IDG News Service, 02 Feb. 2015. Web.

[13] Salinas, Rafael. "RaspiCam: C++ API for Using Raspberry Camera With/without OpenCv." *RaspiCam: C++ API for Using Raspberry Camera With/without OpenCv | Aplicaciones De La Visión Artificial*. 28 Feb. 2015. Web.

[14] Sirovich, Lawrence, and Michael Kirby. "Low-dimensional procedure for the characterization of human faces." *Josa a* 4.3 (1987): 519-524.

[15] TMRh20. "Design Goals." *Optimized High Speed NRF24L01+ Driver Class Documenation: Optimized High Speed Driver for NRF24L01(+) 2.4GHz Wireless Transceiver*. 2014. Web.

[16] Turk, M., and Pentland, A. *Eigenfaces for recognition.* Journal of Cognitive Neuroscience 3 (1991), 71–86.

[17] Viola, Paul, and Michael Jones. "Generic 3D Object Recognition From Time-Of-Flight Images Using Boosted Combined Shape Features." *Proceedings of the Fourth International Conference on Computer Vision Theory and Applications* (2009): Print.

9 Appendix

All software for the DeliverEZ system can be made available upon request.