8-2016

ADDSMART: Address Digitization and Smart Mailbox with RFID Technology

Jonathon Ross Tew

ADDSMART:
ADDRESS DIGITIZATION AND SMART MAILBOX WITH RFID TECHNOLOGY

Jonathan Ross Tew

Columbus State University

D. Abbott Turner College of Business and Computer Science

The Graduate Program in Applied Computer Science

ADDSMART:
Address Digitization and Smart Mailbox with RFID Technology

A Thesis in

Applied Computer Science

By:

Jonathan Ross Tew

Submitted in Partial Fulfillment
of the Requirements
for the Degree of:

Master of Science

August 2016

I have submitted this thesis in partial fulfillment of the requirements for the degree of Master Science.

8/5/16	Jonathan Ross Tew
Date	Jonathan Ross Tew

We approve the thesis of Jonathan Ross Tew as presented here.

8/9/2016	Lydia Ray
Date	Dr. Lydia Ray, Associate Professor of Computer Science, Thesis Advisor
08/10/2016	
Date	Dr. Alfredo Perez, Assistant Professor of Computer Science
8/8/16	Y.S.
Date	Dr. Sumanth Yenduri, Associate Professor of Computer Science
8/5/2016	
Date	Dr. Wayne Summers, Distinguished Chairperson, Professor of Computer Science

Abstract:

ADDSMART is a research project focused on digitizing addresses of locations and building a smart mailbox by combining wireless sensors, cameras, locks, and RFID readers and tags into a system controlled by an Arduino board. The aim of the project is to explore the idea of address digitization (using RFID tags to store addresses) and incorporate it into a mailbox that can communicate wirelessly with the homeowner to provide mail status updates and home security footage through digital photographs. This paper demonstrates the proposed ideas, describes the design of a smart mailbox, the technology that has been used, and the current results of the work along with future research ideas.

Table of Contents:

List of Figures:

List of Tables:

Acknowledgments:

I want to thank my research advisor, Dr. Lydia Ray, for guiding me throughout this process and being there for me every step of the way. Dr. Ray's willingness to help, valuable feedback, and patience with me was very much appreciated as I would not have been able to accomplish my goals without all of her help.

I would also like to thank my committee members Dr. Alfredo Perez and Dr. Sumanth Yenduri. Dr. Perez and Dr. Yenduri provided me with valuable advice and comments along with helping me to navigate several challenging "crossroads" along my journey.

Finally, I want to thank my family – Mom, Dad, and Sam – for all the love and support they provided me along the way. I would not have been able to do any of this without them. Thank you all so much!

Chapter 1: Introduction

The ADDSMART project aims to create an Internet of Things, or IoT, device by combining RFID technology and wireless sensor networking technology in a physical system, i.e., a mailbox, to provide a wide variety of services. The project has two components: address digitization and smart mailbox. The descriptions of the two components are given below.

1.1 Address Digitization

The goal of address digitization is to digitize the address of a building or any location with a physical address. The information written to the tag includes the address, the names associated with the address, phone numbers, and email addresses. The RFID tag with the digitized address would be used in conjunction with Global Positioning System (GPS) technology to provide an accurate identification of an address. Digitized addresses have the potential to be extremely beneficial when driverless vehicles become a reality [7], and drones are used to deliver goods [2] [8] in the future. A RFID tag storing a digital address will provide a way for these automated systems to scan the tag and confirm or deny that they are at the correct location. In addition to this location verification, the RFID tags have the potential to help optimize routes, similar to how various cities are beginning to use the same technology for optimizing waste collection routes [1].

1.2 Smart Mailbox

The goal of this component is to design and build a smart mailbox. The smart mailbox incorporates the address digitization technology mentioned

previously, along with several other technologies. An Arduino board is used to control a RFID reader, camera, motion sensor, locking solenoid, and WiFi module. The main functions of this mailbox are as follows:

- Notifying the home owner when a postal worker has opened the mailbox/new mail has arrived
- Acting as a driveway monitor and notifying the home owner by sending pictures whenever a car or a human crosses a threshold of the owner's driveway

Thus, the smart mailbox will enhance home security at a very low cost compared to surveillance video systems. Surveillance video systems are complex to install and capture video continuously, consuming a lot of electricity and creating large amounts of digital data, much of which is redundant for a home owner. A smart mailbox will consume significantly less energy and will create only relevant data (only when somebody or some vehicle crosses a threshold).

The prototype of the smart mailbox has been built and this paper will focus on the equipment used to accomplish this, how it was set up, and how the different components work together. In the next chapter, the previous work for this project will be briefly outlined, then a discussion of related works will follow. Chapter three will describe the technology used to create the prototype, then chapter four will go over how the components are combined to create the smart mailbox. Chapter five

will describe the testing of the prototype and the results, while the last chapter will conclude the paper.

Chapter 2: Previous and Related Work

The Previous and Related Work section will be separated into three main sections: previous work completed, projects that are similar in nature that might be considered some form of a "smart mailbox", and then related projects that have similar elements to the smart mailbox. The Previous Work section will briefly describe prior attempts at creating a smart mailbox, which were unsuccessful but lead to the working prototype that is presented in this paper. The Similar Concepts section will show that the smart mailbox presented here appears to be the first of its kind. The following section, Related Projects, will outline several projects found on the web that provided guidance for this project and how certain components worked, for example: how to use USB cameras with the Arduino Yun.

2.1 Previous Work

The first iteration of the smart mailbox prototype began with slightly different hardware than what is presented in chapter 3. The main hardware used in the first prototype was an Arduino Uno R3 microcontroller with an Adafruit CC3000 WiFi shield [18] and a small Adafruit camera [19] that worked through TTL (transistor-transistor logic) serial communication [20]. The same motion sensor that is described in chapter 3 was also used with these components.

The first issue with this set up was the camera. The picture resolution had a maximum setting of 640x480 [19] pixels and image quality was low. The last issue with the camera was the price – at $54.95 it was the most expensive component used.

The Arduino Uno R3 paired with the WiFi shield allowed the Uno to connect to wireless networks. The Uno with the WiFi shield was successfully connected to a WiFi network, but there were many issues that started popping up with this setup. First, the reliability was not good at all. One of the simplest test sketches for the WiFi shield involved taking a postal address as input, then connecting to a site and searching the address to gather the basic weather information for that area and then displaying it in the console. This test was hit or miss, often times not working. When trying to send test emails to set up the email communication portion of the project, similar results were seen. Finally, one of the key aspects of the project was the ability to email and upload a photograph. Multiple tests and configurations all lead to unsuccessful attempts at these tasks.

It was at this point the decision was made to begin researching new hardware. This research lead to the hardware that will be outlined in chapter 3, which ultimately resulted in a working prototype.

2.2 Similar Concepts

The proposed idea of address digitization appears to be a novel concept as no previous work or research related to the idea of digitizing addresses using RFID tags has been found at the this time. As such, the research on this topic will provide new direction.

The idea of a smart mailbox is not entirely new, however; two examples have been found with limited features. The first example is a patent filed December 17, 2009 and titled “Method and system for tracking and processing items in

personal mailbox" [5]. The patent describes an invention that would have a small robotic arm inside a mailbox that would pick up and move mail to a scanner. The scanner would collect information off the mail, sending the owner a notification or either sorting it as junk if there is no sender address. No information has been found about this invention outside of this patent, however.

The second example is a project titled "Mr. Postman" by a company named Simple Elements [4] which first appeared as a Kickstarter campaign [21] in 2014 that ultimately never reached its funding goal. Following the Kickstarter, their website began taking pre-orders for the mailbox, and still shows a pre-order button at the time of the last visit. Mr. Postman had a locking system that was controlled by a mobile application that also received notifications when mail was delivered. The locking system would open up during a certain window of time, allowing the postal worker to deliver mail to the mailbox [4]. Neither of these systems have the security monitoring system of the smart mailbox presented in this paper. Additionally, both systems lack any form of digitized addressing or an RFID enabled locking system.

2.3 Related Projects

This section will outline several projects found on the web that were a great help to the work presented in this paper. A brief description of the projects, along with how they were helpful will be included. The smart mailbox created for this project was built in a piecemeal fashion, and many of the projects in this section were used as building blocks.

The first project to be outlined is from Adafruit, an online vendor of hobby electronics specializing in Arduino and Raspberry Pi related goods that also provides many great tutorials and informative guides. One of these guides is titled "Wireless Security Camera with Arduino Yun" and goes through how to set up a simple USB camera with a Yun that is connected to the internet [22]. The guide also details how to set up a service called Temboo to work with the Yun and Dropbox to store pictures captured by the camera. The idea to use Temboo in this project came from reading through this guide early on in the research phase of the project. A more detailed explanation of Temboo can be found in chapter 4. Some of the Arduino and Python code [22] that the Adafruit guide goes over were perfect building blocks for the smart mailbox project. Another useful resource from this guide was the set up process outlined for the USB camera and Yun interface.

Adafruit's guide to PIR motion sensors [23] was another online resource that provided excellent information which helped to lay the ground work for this project. Adafruit's PIR sensor guide has a wealth of knowledge on these sensors; going over how they work, connecting them, testing them, and example projects. The sensor Adafruit uses in their guide is very similar to the sensors that were used in this project; however, there were a few slight differences. To account for these small differences, the manufacturer's data sheet [24] was used in conjunction with the Adafruit guide to obtain a firm grasp on the workings of these sensors. Additionally, some of the Arduino code [23] provided in the guide turned out to be useful for testing the sensors used in the smart mailbox.

The RFID locking solenoid was inspired by projects like Robot Geek's "RFIDuino Lock Box Getting Started Guide", where RFID readers were being used to control access to boxes locked with a solenoid [25]. The Robot Geek guide goes over how to build a small trinket box that is locked and controlled by RFID. They go over the hardware needed, how to connect everything, and how to write the code to control the box. Ultimately, the smart mailbox project ended up going in a different direction than what they outline, and instead took advantage of some of the test sketches that came with the Adafruit library for their RFID reader (the reader used in the smart mailbox).

Chapter 3: Description of Technology

3.1 Address Digitization

To create the digitized addresses, the Android application TagWriter created by NXP [10] was used to write the information to the RFID tag. The TagWriter application helped to streamline the work in this area, as it is easy to use and performs exactly the tasks that were needed for this phase of the project. The RFID system in the smart mailbox works on the high frequency range which allows smart phones with Near Field Communication (NFC) chips to communicate with the tags used on the mailbox – this will be discussed more in section 3.2. The type of tag used for the prototype was a simple sticker tag that was affixed to the front of the mailbox door after the necessary data had been written to it. Another benefit of using the TagWriter Android application was that it allowed the tag to be set up to open a message with the data read from the tag when scanned with any NFC enabled smartphone. A screenshot from an Android phone reading the tag is shown in Figure 1. This allows the digital address to be read quickly and easily.

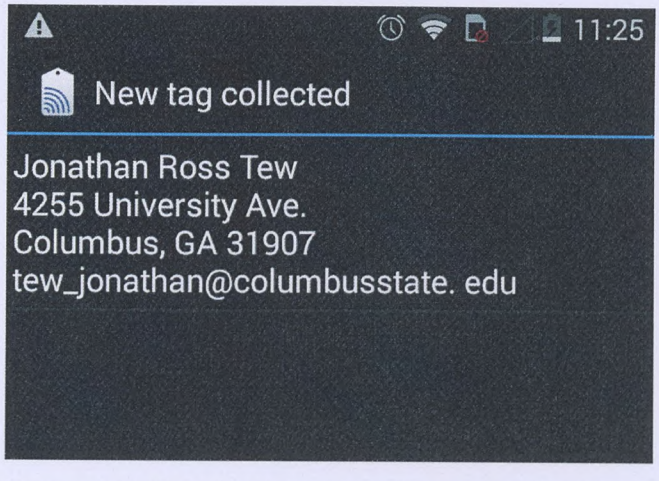

Figure 1: Screenshot of Android phone reading a digital address RFID tag.

3.2 Smart Mailbox

The smart mailbox prototype has an Arduino board as its brain. The Arduino controls a RFID reader, camera, motion sensor, locking solenoid, and onboard WiFi module. A brief description of each technology is given below:

Arduino: The Arduino board being used for this project is an Arduino Yun (Figure 2, Item A), one of the newest Arduino boards. The Yun is different from other Arduino boards as it has both a microcontroller and a microprocessor. The Yun uses an ATmega32u4 for its microcontroller and an Atheros AR9331 for its microprocessor [26]. The use of these two different chips allows the Yun to run the Arduino environment on the microcontroller and a special Linux distribution known as OpenWrt-Yun on the microprocessor [26]. The AR9221 has 802.11b/g/n WiFi built in, along with Ethernet support. The board also features a Micro USB port for hard-wire programing of the board, and a Micro SD card slot for expandable on-board storage. Finally, there is a standard USB Type A port on the board, which is important as this is how the camera will be connected to the system.

RFID Shield: An RFID shield (Figure 2, Item B) built for Arduino by Adafruit is being used as the RFID reader for this project. The shield uses a PN532 chip-set, which works as a reader/writer operating on the 13.56MHz High Frequency (HF) range. RFID devices available to consumers typically fall within one of three frequency ranges: low frequency (LF), high frequency (HF), and ultra-high frequency (UHF). The low and high frequency range devices offer similar performance at similar price points, while the UHF devices offer more powerful

performance – much farther read ranges – at a significantly higher price point. The price of the UHF devices would be too prohibitive for this project, so the choice was narrowed down to either the low or high frequency RFID systems.

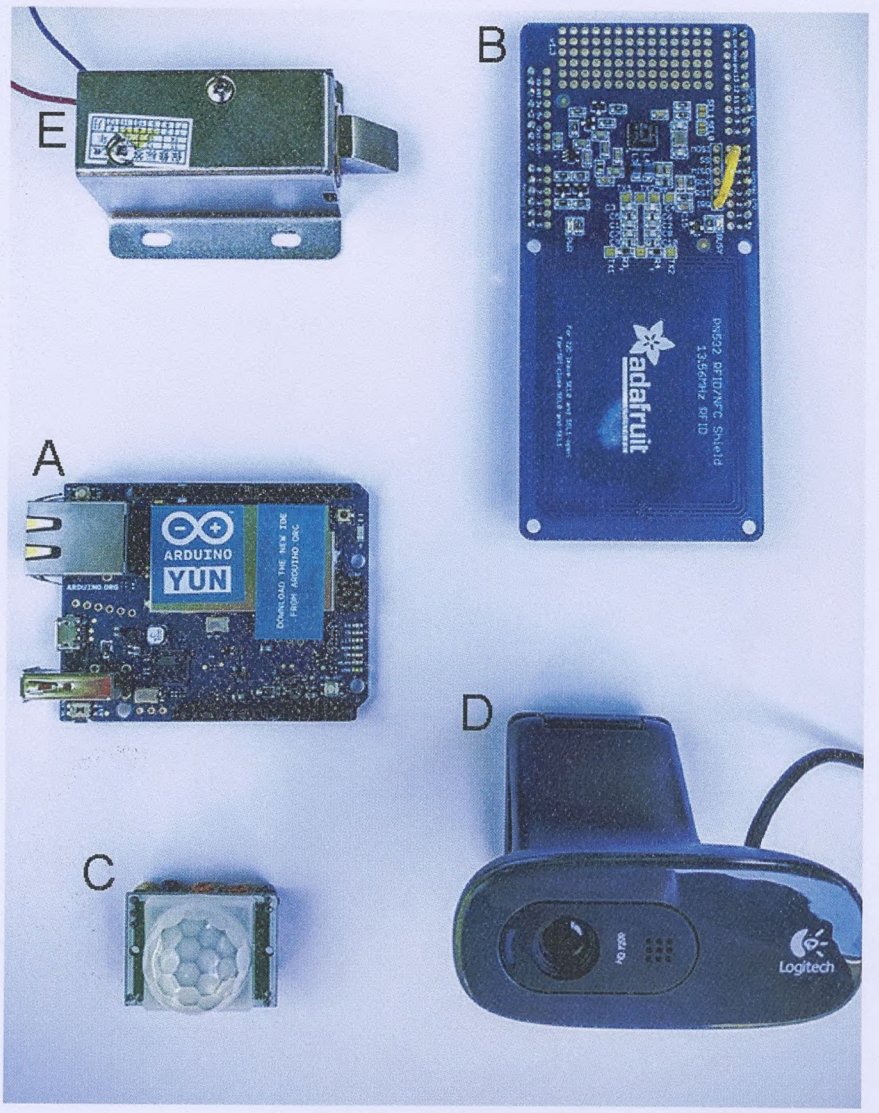

Figure 2: Smart mailbox hardware. A: Arduino Yun, B: RFID shield, C: PIR sensor, D: Logitech USB webcam, E: Lock solenoid

As mentioned in section 3.1, one of the benefits of HF based RFID systems is their compatibility with NFC. NFC is an emerging technology that appears to be

gaining traction, particularly in the smartphone sector: it is estimated two in three phones will have NFC capabilities by 2018 [11]. NFC is a sub-technology of HF RFID, built on the same specifications as its parent technology. The main difference, however, is that NFC is limited to communication no greater than 10 centimeters [12]. The capability of HF RFID to be used alongside NFC is what guided the decision to use HF over LF for this project.

The PN532 chip-set in the Adafruit shield works with both HF RFID tags and NFC specific tags, allowing it to read and write both types of tags. Finally, through previous testing of the Adafruit shield, the typical read range of this unit was found to be around 80mm with the Mifare Classic 1k tags described below.

RFID Tags: The tags used in this project are Mifare Classic 1k tags, operating on the HF range. These tags are manufactured by NXP Semiconductors and are designed according to the ISO/IEC 14443 Type A standard – a standard for contactless proximity cards and their transmission protocols [13]. The Mifare tags are capable of holding up to 1kb of data in their Electrically Erasable Programmable Read-Only Memory (EEPROM), which has an endurance rating of 100,000 write cycles [13]. This amount of storage has not been an issue for this project, as the most data being written to a card is a name and address information. However, should there be an issue with storage, there are Mifare Classic 4k tags available with 4kb of EEPROM storage.

A variety of tags were tested throughout the research, including coin, key fob, credit card, and sticker style tags as shown in Figure 3. The sticker style tags were used to store the digital address information as they attached easily and securely to the front of the mail box with their adhesive backing. The card and key fob style tags were selected for homeowner and postal worker use as these provided the best read ranges and are the most convenient style to use in daily life.

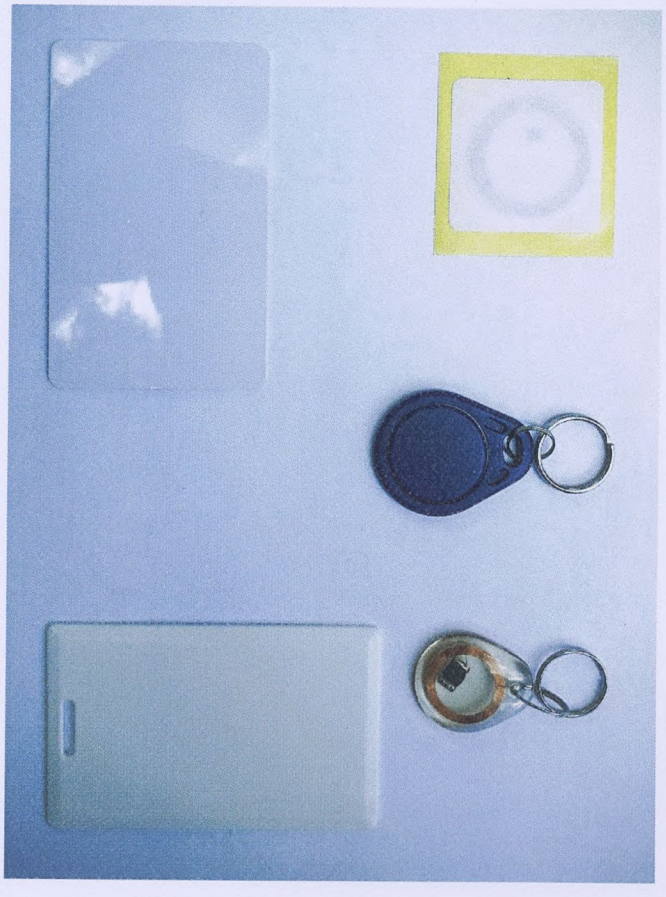

Figure 3: RFID tags.

Sensors: The sensor being used for motion detection is a Passive Infrared (PIR) sensor, as shown in Figure 2, Item C. PIR sensors, like the one in this project, work

by incorporating two infrared sensors to detect a change in infrared radiation – a type of electromagnetic radiation. Infrared radiation is one way that heat can be transferred, and it is important to note that nearly everything emits infrared radiation; anything with a temperature greater than negative 450 degrees Fahrenheit emits infrared radiation [14].

As mentioned, two of these sensors are actually used behind the lens to accurately detect movement and prevent false positives from changes in temperature or sunlight. For example, if there is a change in sunlight hitting the two sensors, both sensors will be affected at the same time – which will result in no output from the PIR sensor as each smaller sensor will cancel the other one out. However, if a person were to walk across the path of the PIR sensor, the two individual infrared sensors will be affected in a sequence of: first, both, then second. A sequence like this produces a positive, neutral, then negative output (or vice versa, depending on which way the subject is moving) [15]. A sensor designed like this is desirable, as it should cut down on false positives.

Finally, the PIR sensor is outfitted with a Fresnel lens to extend the sensor's range and detection field. The literature for the PIR sensor claims an effective range of 21 feet with a detection field of 120 degrees. Tests of these claims were conducted and are presented in chapter five. The PIR sensor will be connected to the Arduino, and the Arduino programmed to begin taking photographs with a camera mounted inside the mailbox once the sensor has been triggered. The

camera will continue taking photographs as long as motion is detected, capturing the person or vehicle that triggered the sensor.

Camera: Two cameras were tested for use with the smart mailbox. The first camera tested was a Logitech c270 (Figure 2, Item D) – a small, inexpensive USB webcam. Logitech's camera is capable of capturing pictures in a resolution up to 1280x720 pixels, or 720p, at a size of up to 3.0 megapixels. More importantly, however, is the fact that this webcam is compatible with the USB Video Class (UVC) protocol. UVC is a standardized video driver that dictates how video is moved from a USB camera to another device, such as an Arduino [16]. Because the webcam is compatible with UVC, a downloadable utility known as fswebcam will be installed on the Linux side of the Arduino for use in controlling the webcam. The fswebcam utility provides a variety of options to set for the camera, which will be covered with more detail in chapter four.

The second camera tested was from a company called ELP and was purchased through Amazon. The ELP camera is capable of capturing pictures with 720p resolution at a size of up to 1.0 megapixels and is also compatible with the UVC protocol. The main difference between this camera and the Logitech, however, is the addition of a circular Infrared, or IR, LED array around the lens of the camera. The camera has a light sensor that turns on the IR LEDs once it is dark, allowing the IR lights to help illuminate the camera's field of view for better low light pictures. Chapter four will outline how the two cameras were tested, and which one ended up being used in the smart mailbox.

The motion sensor and camera will be used together to monitor the entrance to the home, and alert the owner of any activity. The camera will begin taking photographs when the motion sensor is tripped. The Arduino will then save the photographs to the onboard Micro SD card. After storing the photograph on physical media, a copy will be uploaded to a Dropbox account and then emailed to the homeowners email, allowing them to receive a copy of the picture on the smart phone. Not only does this give the homeowner multiple ways to view the photographs, it also provides backup copies of the media – three different places in total.

Solenoid: The solenoid used for this project has an armature with a slanted cut, as seen in Figure 2, Item E. When the solenoid is energized, the armature is pulled into its housing, allowing the door to open freely. After a set time, the energy to the solenoid is stopped, and the armature returns to its resting state outside of the housing. The solenoid is controlled by the RFID reader inside the mailbox. When a pre-approved tag is read, the Arduino energizes the solenoid, unlocking the door. Tag IDs for the owner(s) of the mailbox are stored onboard the Arduino, along with a universal ID for postal worker's tags. Additionally, whenever the postal worker's tag is used to open the mailbox, the owner will be alerted that they have received mail.

Chapter 4: Creating a Smart Mailbox

The bulk of the time spent on this project has been in creating the physical smart mailbox - the prototype. Once the research into the various pieces of hardware that would be needed to assemble the prototype was complete, and all of the various bits and pieces had been acquired, the building of the smart mailbox began in a piecemeal fashion. Section 4.1 will highlight this process, demonstrating how the main components are assembled separately of each other and tested for proper functionality. Following that, section 4.2 will outline how everything is put together to create the fully functional prototype. Finally, section 4.3 will go over how all of the hardware was installed in the mailbox.

4.1 Setting up the Individual Components

To begin building the prototype, all of the different components that would comprise the smart mailbox first needed to be setup and tested, making sure they worked fine on their own before adding them to the larger picture. This section will outline how each piece of hardware is connected and set up, along with how its functionality was tested.

4.1.1 Arduino Yun Setup/Configuration and USB Webcam

The first step in creating the prototype smart mailbox was to prepare the Yun for its role as the brains of the mailbox. To do this, a computer with WiFi capabilities is used to connect to the Yun as an access point, then a web browser is used to open the Yun's configuration panel. After connecting to the Yun, it is given a name and then a password is created to secure the board from un-

authorized access over WiFi. After this, the board's wireless parameters were configured by setting the name, security type, and password for the personal wireless network [28] it would be connecting to – this allows for the Yun to connect automatically to the stored WiFi network.

After successfully connecting the Yun to the network, it was time to update and install some packages, drivers, and utilities on the Linux side of the board. In order to accomplish this, a secure shell, or SSH, must be used to connect to the Linux side of the board and gain access to the command prompt. On the Windows based machine being used for this project, the software PuTTY was used to establish a SSH connection [27] [29]. After successfully connecting, root access to the Linux operating system is established by using the password set up in the configuration panel earlier [29]. Using the command prompt, the following actions were completed:

- Updated the package manager
- Installed the USB Video Class, or UVC, drivers
- Installed the fswebcam utility

The UVC drivers and the fswebcam utility are important for the proper functioning of the USB webcam [22].

After finishing these updates and installations, it was a good time to test the cameras to make sure they were going to work properly. In order to do this, each camera was connected via USB and a Micro SD card inserted in the Yun. The

fswebcam utility has a simple command to snap a picture and store it to the Micro SD card onboard the Yun [37]. This command was used to take a few pictures, then the Micro SD card was removed and a card reader was used to check that the camera was operating correctly and the images had been saved. Several test runs were required so that the proper settings could be adjusted following the documentation at [37]; ultimately the resolution was set to 1280x720 using *–r 1280x720* and the banner that displayed the time stamp had to be turned off using *--no-banner*. With these small adjustments, the camera was working great.

4.1.2 Connecting the PIR Sensor

The next step in creating the prototype was to wire up the PIR sensor using a simple circuit on a breadboard to test that it was working correctly. In order to do this, a red LED light was connected through the digital out of the PIR sensor so that it would light up when the sensor detected motion [23]. The Yun was used to control these components by uploading some basic code using the Arduino IDE:

```
/*
  Author: Ross Tew
  File: PIR_Test.ino
  About: Simple test code for the PIR Sensor that lights up a
  LED when motion is detected, and turns it off afterwards.
  Based on sample code from Adafruit's PIR Sensor guide @ [23]
 */

int ledPin = 13; // pin for the LED
int inputPin = 8; // input pin for PIR sensor
int pirState = LOW; // start state, assuming no motion detected
int val = LOW; // variable for reading the pin status

void setup() {
  pinMode(ledPin, OUTPUT); // declare LED as output
  pinMode(inputPin, INPUT); // declare sensor as input

  delay(60000); // Delay to allow PIR sensor to initialize
  // Datasheet suggests 60 seconds
}
```

```
void loop(){
  val = digitalRead(inputPin); // read input value
  if (val == HIGH) { // check if the input is HIGH
    digitalWrite(ledPin, HIGH);  // turn LED ON
    // Check the pirState variable to see it if is still LOW,
    // which indicates motion was JUST detected
    if (pirState == LOW) {
      // Change the state to HIGH:
      pirState = HIGH;
    }
  } else { // Digital read is no longer HIGH
    if (pirState == HIGH){
      digitalWrite(ledPin, LOW); // turn LED OFF
      pirState = LOW; // Reseat the state back to LOW
    }
   }
}
```

The code in this sketch checks for motion from the PIR sensor using the *digitalRead()* function: when this value is *HIGH*, that means the sensor has detected motion and the voltage output of the sensor has been increased to *HIGH*. The *pir_state* variable acts as a control variable since the output voltage from the sensor will stay *HIGH* for several seconds, but in the meantime the main *loop()* function is still repeating. The *pir_state* variable prevents the code that should be executed once when motion is detected from being repeated many times as the main *loop()* function repeats by being switched to *HIGH* at the end of that section of code. Now, even though the main *loop()* function is repeating itself and the PIR sensor is still reporting a voltage of *HIGH* from the *digitalRead()* function, the nested *if* statement that checks the *pir_state* variable prevents the code inside this statement from executing again until the PIR sensor's voltage drops, and then is raised again due to a new occurrence of motion. In this specific sketch, the only action taken is to light up the LED when motion was detected, then turn it off after the sensor's voltage dropped back to *LOW*.

A circuit prototyping software known as fritzing [30] was used to design a layout digitally before hooking anything up. This fritzing diagram of the simple circuit can be seen in Figure 4 below.

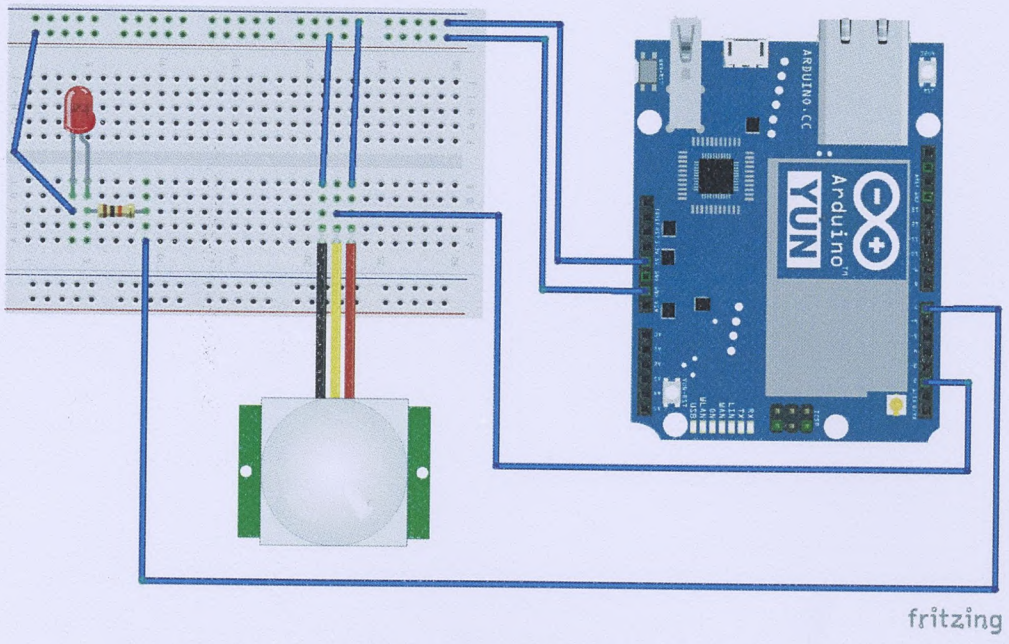

Figure 4: fritzing diagram of PIR sensor testing circuit.

Using the LED light as an indication of the sensor detecting motion confirmed the PIR sensor was working properly. Next, the PIR sensor and camera needed to be tested to make sure they worked well with each other. To do this, the sensor was connected straight to the Yun, the webcam connected via USB, and a Micro SD card inserted in the board. A simple test sketch was created in the Arduino IDE and then uploaded to the board:

```
/*
  Author: Ross Tew
  File: Cam_and_PIR_Test.ino
  About: Code to test the PIR Sensor with the camera. Whenever
  motion is detected, the camera takes a picture and stores it to
  MicroSD card on board the Yun.
  Based on sample code from Adafruit's PIR Sensor guide @ [23]
  and their security camera project @ [22]
```

```
 */
#include <Bridge.h>
#include <Process.h>

Process picture;

String filename; // Variable for picture file name
int pir_pin = 9; // pin for PIR sensor input
int led_pin = 13; // LED pin
int pir_state = LOW; // Control variable for PIR's state
int val = LOW; // Variable for storing the digitalRead value of PIR Sensor
String path = "/mnt/sda1/"; // MiniSD card storage location

void setup() {
  // Function to start the Bridge - allows communication between the two chips:
  Bridge.begin();

  pinMode(pir_pin, INPUT); // Declare sensor as input
  pinMode(led_pin, OUTPUT); // Declare LED as output

  delay(60000); // Delay to allow PIR sensor to initialize
  // Datasheet suggests 60 seconds
}

void loop() {
  // Read the digital value from the PIR Sensor to see if there is motion:
  val = digitalRead(pir_pin);
  if (val == HIGH) { // A value of HIGH means there was motion
    digitalWrite(led_pin, HIGH);   // turn the LED on
    // If control variable is LOW, motion was JUST detected:
    if (pir_state == LOW) {
      filename = ""; // Blank filename

      // Run a shell command to get the date/time for building unique filename:
      picture.runShellCommand("date +%s");
      while(picture.running());

      // Read the results from above and convert the chars:
      while(picture.available()>0) {
        char c = picture.read();
        filename += c;
      }

      filename.trim();
      filename += ".png";

      // Run the fswebcam command to take a picture, giving it the unique
      // filename from above and saving it to MiniSD card:
      picture.runShellCommand("fswebcam -r 1280x720 --no-banner " + path +
filename);
      while(picture.running());

      // Change the PIR state to HIGH:
      pir_state = HIGH;
    }

  } else { // Digital read is no longer HIGH
      if (pir_state == HIGH){
        digitalWrite(led_pin, LOW); // turn LED OFF
        pir_state = LOW; // Reseat the state back to LOW
```

```
        }
    }
}
```

The code in this sketch builds on the previous sketch's code, adding the required commands to take a picture and store it on the Micro SD card. The function *runShellCommand()* is used to execute the fswebcam utility on the Linux side of the Arduino. Note that a *Process* must be defined, in this case named *picture*, to enable the use of the *runShellCommand()* function. *Process* is a class in the Arduino *Bridge* library [31] – the library that enables communication between the Arduino based chip and the Linux based chip on the Yun's board [31]. Using this setup, the sensor and camera worked together without issue.

At this point, it was clear the sensor and camera components were working correctly, so it was time to test the two cameras and see which one would be the best fit for the project. The same sketch from above was used and the system was left to run for forty-eight hours in a high traffic area indoors. The same test was conducted for both cameras and the results are displayed in Table 1. The Logitech webcam took a total of 2632 pictures over the forty-eight hour period and only five of these picture were unusable. For this test, an unusable picture was one that was either blank or had enough artifacts to make it impossible to tell what the picture taken was of. Section 5.2 will go into more detail about artifacts, including compression and blocking artifacts. With only five unusable pictures, the Logitech webcam had a success rate of 99.81%.

Table 1: Results from testing the reliability of two different USB cameras.

Camera Testing				
Camera	**Usable Pictures**	**Unusable Pictures**	**Total Pictures Taken**	**Success Rate**
Logitech C270 USB Camera	2627	5	2632	99.81%
ELP USB Camera w/ IR Array	1935	541	2476	78.15%

The ELP camera took a total of 2476 pictures over the forty-eight hour period and 541 of these were unusable. The success rate of the ELP camera was considerably lower than the Logitech's at 78.15%. Preliminary testing of this camera revealed some issues, so it was not too surprising to see these results. The ELP camera was not as reliable as the Logitech – it was unable to produce consistent results. As a result of this testing, the camera of choice was the Logitech C270 webcam.

4.1.3 Connecting the RFID Shield

The next step in the prototyping process was to connect the RFID reader to the Yun board so that the reader's functionality could be tested. The RFID unit being used in the smart mailbox is classified as a shield, as opposed to a breakout board. The main difference between the two is their form factor – the shield can be attached to the Yun with stacking headers, while the breakout board needs a circuit with a level shifter chip before it can be wired to the Yun [32]. Having the shield allowed for almost plug-and-play ease of use, but also meant there would be some

extra effort involved when it came time to mount the shield in the door of the mailbox, away from the Yun. To begin using the shield, headers needed to be soldered to the shield's board so that it could be connected to the Yun. After this, the male and female headers on the RFID shield and the Yun are lined up and then pushed into place, connecting the two devices.

The RFID shield came from Adafruit [33], who has created their own Arduino library for the unit [34], including example sketches. These files were downloaded and the new library was added to the Arduino *libraries* folder, then testing began by using some of the example sketches provided by Adafruit. By using these sketches, it was possible to see that the shield was working properly – recognized by the Yun – and that it was able to write and read to and from the RFID tags without issue.

4.1.4 Wiring the Solenoid to the Yun

The last component to test was the locking solenoid. The solenoid uses a good bit of power very quickly to engage the electromagnet that retracts the metal armature into the body. Because of this, it was a little more involved than the other components as it needed a circuit that provided the solenoid with its own power, and provided protection to the Yun's board so that it would not be damaged.

The circuit design for this step of the build process was created using a breadboard as a temporary means of testing the solenoid before soldering a dedicated circuit on perfboard. The fritzing circuit prototyping software was used again to design the initial breadboard connections – the diagram for this design

can be seen in Figure 5. The circuit uses a TIP120 transistor, 1k Ohm resistor, 1N4004 diode, and female barrel jack.

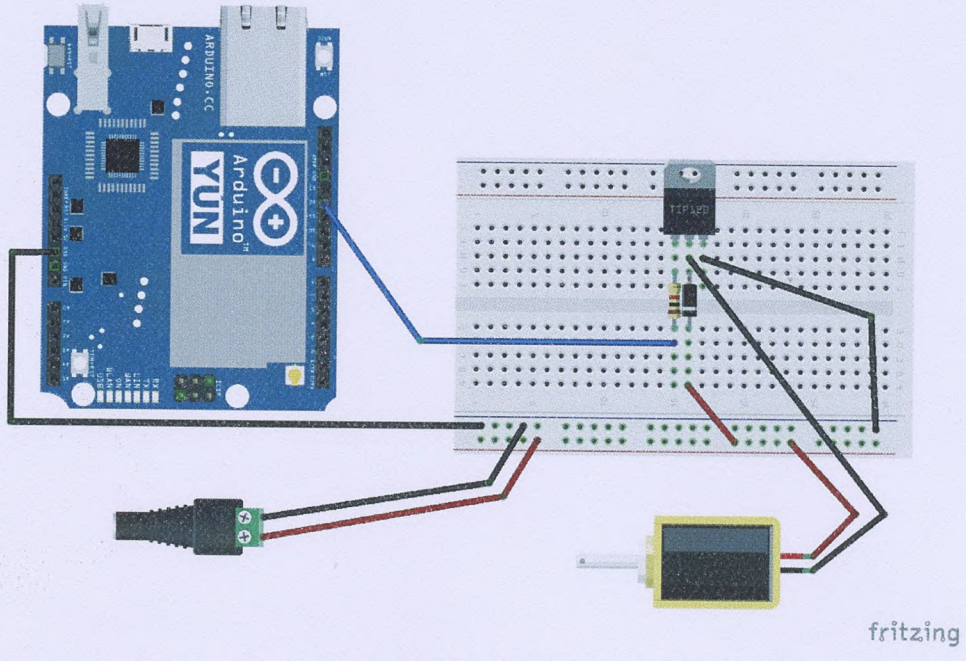

Figure 5: fritzing diagram of circuit for testing the solenoid.

The circuit was wired up, then a test sketch was created in the Arduino IDE to test the solenoid's functionality by powering the solenoid every time the "Enter" key was pressed on the keyboard of the computer the Yun was connected to. When the solenoid is powered, the electromagnet retracts the metal armature into the body of the solenoid, which then allows the door to be opened.

After successfully testing the solenoid, the RFID reader needed to be paired with it to test their functionality together and make sure the reader could properly control the solenoid when an authorized tag was read. In order to do this, the reader was hooked up to the Yun, then another sketch was created in the Arduino software to power the solenoid when an approved RFID tag was read:

```
/*
  Author: Ross Tew
  File: RFID_and_Solenoid_Test.ino
  About: Used to test the RFID reader and locking solenoid work properly.
  RFID reader reads a tag and checks to see if it is approved - if so the
  solenoid is unlocked, if not, a message is printed to the console.
  Based on example sketches included in Adadfruit's library for their
  PN532 shield found @ [34]
*/
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_PN532.h>


// When using the shield with I2C, define just the pins connected
// to the IRQ and reset lines:
#define PN532_IRQ   (6)
#define PN532_RESET (3)  // Not connected by default on the NFC Shield

// This line is used for a shield with an I2C connection:
Adafruit_PN532 nfc(PN532_IRQ, PN532_RESET);

void setup(void) {
  Serial.begin(115200);
  while(!Serial)
    ;

  pinMode(12, OUTPUT);

  Serial.println("Hello!");

  nfc.begin();
  uint32_t versiondata = nfc.getFirmwareVersion();
  if (! versiondata) {
    Serial.print("Didn't find PN53x board");
    while (1); // halt
  }
  // Got ok data, print it out!
  Serial.print("Found chip PN5"); Serial.println((versiondata>>24) & 0xFF,
HEX);
  Serial.print("Firmware ver. "); Serial.print((versiondata>>16) & 0xFF, DEC);
  Serial.print('.'); Serial.println((versiondata>>8) & 0xFF, DEC);

  // configure board to read RFID tags
  nfc.SAMConfig();

  Serial.println("Waiting for an ISO14443A Card ...");
}


void loop(void) {
  uint8_t success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 }; // Buffer to store the returned UID
  uint8_t uidLength; // Length of the UID (4 or 7 bytes depending on ISO14443A
card type)
  uint32_t cardID = 0;

  // Wait for an ISO14443A type cards (Mifare, etc.):
  success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength);
```

```
  if (success) {
    Serial.println("Found an ISO14443A card");

    // turn the four byte UID of a mifare classic into a single variable #
    cardID = uid[3];
    cardID <<= 8; cardID |= uid[2];
    cardID <<= 8; cardID |= uid[1];
    cardID <<= 8; cardID |= uid[0];
  }
  else
  {
    // PN532 probably timed out waiting for a card
    Serial.println("Timed out waiting for a card");
  }

  if(cardID == 1072309060 || cardID == 3492602243){
    Serial.println("Access granted!");
    digitalWrite(12, HIGH);   // turn the LED on (HIGH is the voltage level)
    delay(3000);              // wait for 3 seconds
    digitalWrite(12, LOW);    // turn the LED off by making the voltage LOW
  }
  else{
    Serial.println("Access DENIED!");
  }

  Serial.println("");
      // Wait 1 second before continuing
      delay(1000);

}
```

The sketch for this test constantly checks for a RFID tag and the main *loop()* function is actually held up until one is found. The function that does this is *readPassiveTargetID()* which controls the RFID reader and has it constantly searching for a tag within its read range. Once a tag is found, the tag number is converted from hexadecimal to an unsigned integer and stored in the *cardID* variable. The *cardID* is checked to see if it matches any approved identification numbers, and if so the solenoid is powered on for three seconds opening the latch. Once this was confirmed to be working properly, this phase of the project was complete.

4.2 Combining the Components into One System

The next step of the project was to assemble the prototype by combining all of the components into one system where they all worked alongside each other. This would involve some more work with the hardware, such as modifying some of the circuits, but would also include a lot more work on the software side of the project. The final design of the system that dictated how these different pieces worked together both physically and digitally can be seen in the flowchart in Figure 6. The flowchart represents the goal of this phase of the project. This section will present an overview of how everything came together into one unit.

4.2.1 The Wiring and Circuits

The first issue that needed to be addressed was the need for the RFID shield to be mounted in the door of the mailbox separate of the Yun. The RFID shield was intended to be stacked on the Yun through the headers soldered to the board, which meant wires would need to run from the male header pins on the shield to the female header sockets on the Yun's board. As this is a prototype, the ability to disconnect the shield if needed was important, so jumper wires were used to plug into the header pins on the RFID shield. However, jumper wires were not long enough to run the length of the mailbox, so solid copper hookup wire was soldered to the ends of the jumper wires so they could reach and be inserted into the female headers on the Yun. This setup allowed the RFID reader to be connected and disconnected as needed.

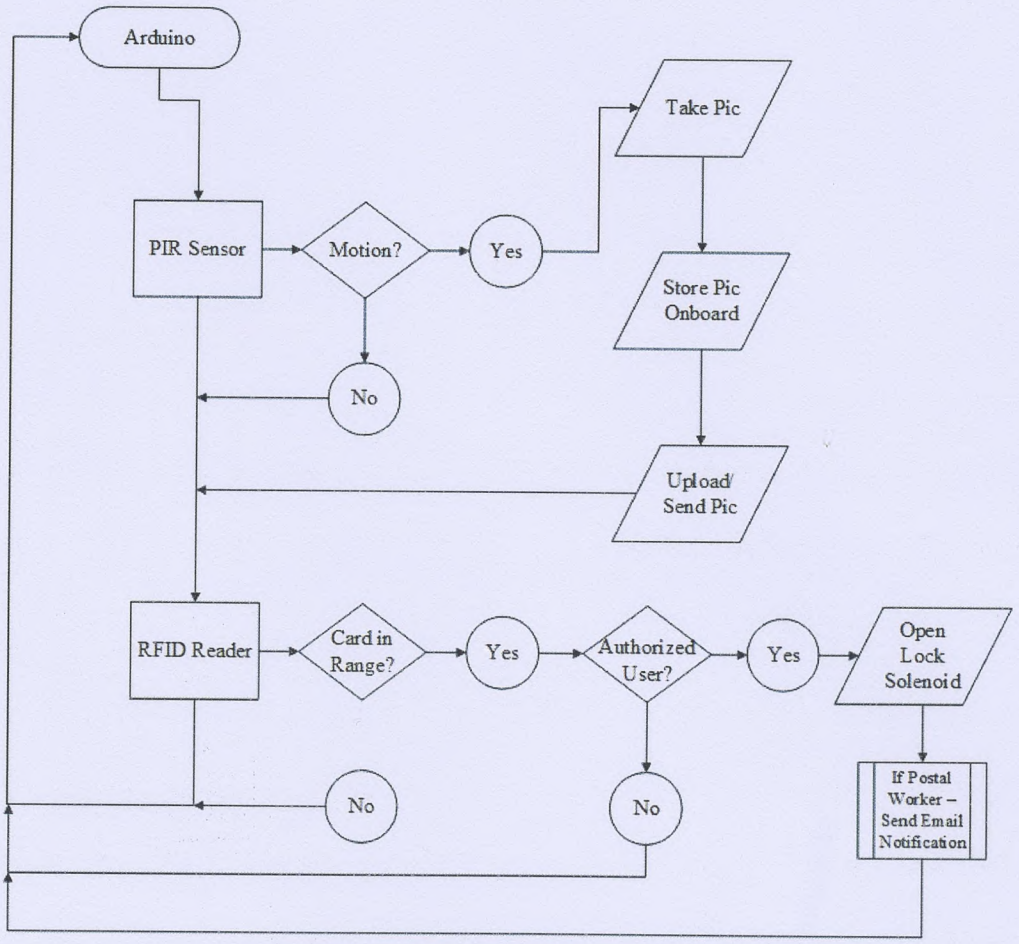

Figure 6: Flowchart for smart mailbox.

The next issue that needed addressing was the circuit for the solenoid and the need to share power and ground connections from the Arduino board. The solution to these two issues was to use two simple circuits on one small piece of perfboard. The circuit designed for the solenoid on the breadboard from section 4.1.4 was transferred to the perfboard, then another area of the perfboard was used to share 5V power and ground from the Arduino for the RFID shield and PIR sensor. A fritzing diagram for the perfboard can be seen in Figure 7. With these

issues resolved, only a few simple connections remained and all of the hardware was complete and ready to go.

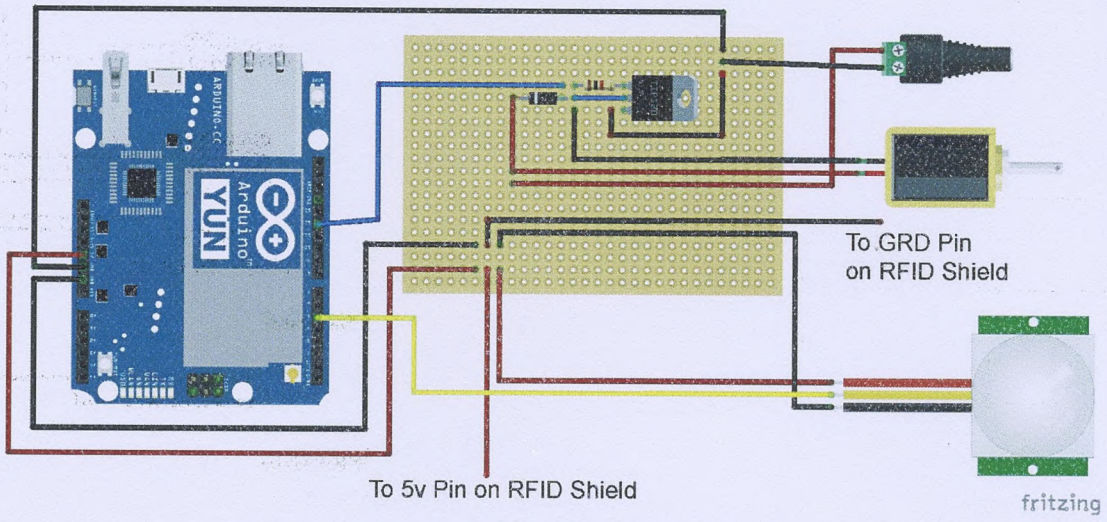

Figure 7: fritzing diagram of completed perfboard circuit.

4.2.2 The Software Side

To outline the software side of the project, section 4.2.2.1 will first go over the internet-based applications, Gmail and Dropbox, being used in this project and how they are controlled by the smart mailbox. The section following will provide an overview of the Arduino and Python code used to control the system.

4.2.2.1 Accounts, Authorizations, and Temboo

The two internet based utilities that were needed for this project were email and cloud storage. Google's Gmail is used for email, while Dropbox is being used for cloud storage. To help simplify the use of these online applications, the "cloud-based code generation platform" known as Temboo [17] would be used: Temboo worked as a middleman between the smart mailbox's code and the Gmail and

Dropbox APIs. This allowed the use of Temboo's Python Software Development Kit, or SDK, to write short, simple code that could send an email, send an email with a picture attached, and upload a picture to Dropbox.

The first step of this process was creating accounts for Temboo, Gmail, and Dropbox. After this, the developer options for Google and Dropbox were used to create "Apps" that Temboo would have access to [22]. The trickiest part of this process was setting up the correct authorization and keys to be used between the different apps and Temboo; fortunately the Temboo website shows users how to do this [22]. After all of this was successfully set up, it was time to finalize the code for the project.

4.2.2.2 Arduino and Python Code

The final code for the smart mailbox consisted of one Arduino sketch and two Python scripts. To create the final Arduino sketch, the test sketches used in section 4.1 were assembled into one main sketch, modifying them as needed so everything would work properly together. The structure of the Arduino sketch consists of one main loop in the code, as is typical in these sketches, with two main if statements to control the flow of the program:

```
/*
  Author: Ross Tew
  File: YunFinalCodeV3.1.ino
  About: Final code for smart mailbox - this controls the PIR
  sensor, camera, RFID reader, and locking solenoid. Uses two Python
  scripts - upload_send_picture.py and mailbox_opened.py - that
  work with Temboo to send emails and upload files to Dropbox.
  Parts of code based on the Adafruit project @ [22]
  along with example sketeches included in their library for the PN532
  shield found @ [34]
*/
#include <Bridge.h>
```

```
#include <Process.h>
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_PN532.h>

// When using the shield with I2C, define just the pins connected
// to the IRQ and reset lines:
#define PN532_IRQ   (6)
#define PN532_RESET (3)  // Not connected by default on the NFC Shield

// This line is used for a shield with an I2C connection:
Adafruit_PN532 nfc(PN532_IRQ, PN532_RESET);

// Define a process to use for running linux processes on the AR9331 chip:
Process linuxProcess;

// Setting up variables:
String filename;
int pir_pin = 8; // PIR Sensor's output is connected pin 8
int pir_state = LOW; // Control variable for PIR's state
int pir_val = LOW; // Variable for storing the digitalRead value of PIR Sensor
int sol_pin = 12; // Solenoid controled by pin 12
int led_pin = 13; // Red LED light on Yun board
String path = "/mnt/sda1/"; // MiniSD card storage location

void setup() {
  // Function to start the Bridge - allows communication between the two chips:
  Bridge.begin();

  // Set the pin modes for the PIR, solenoid, and LED:
  pinMode(pir_pin, INPUT);
  pinMode(sol_pin, OUTPUT);
  pinMode(led_pin, OUTPUT);

  // Start the RFID/NFC shield:
  nfc.begin();
  uint32_t versiondata = nfc.getFirmwareVersion();
  if (! versiondata) {
    while (1); // halt
  }

  // Set the max number of retry attempts to read from a card
  // This prevents waiting forever for a card, which is
  // the default behaviour of the PN532.
  nfc.setPassiveActivationRetries(1);

  // configure board to read RFID/NFC tags
  nfc.SAMConfig();
}

// The main logic for that controls the mailbox, constantly repeating:
void loop() {
  // Read the digital value from the PIR Sensor to see if there is motion:
  pir_val = digitalRead(pir_pin);
  if (pir_val == HIGH) {   // A value of HIGH means there was motion
    digitalWrite(led_pin, HIGH); // Turn on red LED
    // If control variable is LOW, motion was JUST detected:
    if (pir_state == LOW) {
      filename = ""; // Blank filename
```

```
      // Run a shell command to get the date/time for building unique filename:
      linuxProcess.runShellCommand("date +\"%G%m%d%H%M%S\"");
      while(linuxProcess.running());

      // Read the results from above and convert the chars:
      while(linuxProcess.available()>0) {
        char c = linuxProcess.read();
        filename += c;
      }

      filename.trim();
      filename += ".png";

      // Run the fswebcam command to take a picture, giving it the unique
filename
      // from above and saving it to MiniSD card:
      //linuxProcess.runShellCommand("fswebcam -r 1280x720 --no-banner " + path
+ "`date +\"%G%m%d%H%M%S\"`.png");
      linuxProcess.runShellCommand("fswebcam -r 1280x720 --no-banner " + path +
filename);
      while(linuxProcess.running());

      // Run the python script that uploads/emails the picture:
      linuxProcess.runShellCommand("python " + path + "upload_send_picture.py "
+ path + filename);
      while(linuxProcess.running());

      pir_state = HIGH;
     }
   } else {
     if (pir_state == HIGH) {
       digitalWrite(led_pin, LOW); // Turn LED off
       pir_state = LOW; // Reset control variable back to LOW
     }
   }

  // Set up variables to use with RFID reader:
  uint8_t success;
  uint8_t uid[] = { 0, 0, 0, 0, 0, 0, 0 };  // Buffer to store the returned UID
  uint8_t uidLength;                        // Length of the UID (4 or 7 bytes
depending on ISO14443A card type)
  uint32_t cardID = 0;

  // Check twice (first time, then one retry as set in setup()) for an
ISO14443A
  // type card (Mifare, etc.).  If one is found, the 'uid'
  // will be populated with the UID, and uidLength will indicate
  // if the uid is 4 bytes (Mifare Classic) or 7 bytes (Mifare Ultralight)
  success = nfc.readPassiveTargetID(PN532_MIFARE_ISO14443A, uid, &uidLength);

  if (success) {
    // turn the four byte UID of a mifare classic into a single variable #
    cardID = uid[3];
    cardID <<= 8; cardID |= uid[2];
    cardID <<= 8; cardID |= uid[1];
    cardID <<= 8; cardID |= uid[0];
  }

  // Check if card is homeowner's, if so, open the lock:
  if(cardID == 1072309060){
```

```
    digitalWrite(12, HIGH);   // Energize the solenoid to retract the lock
(HIGH is the voltage level)
    delay(5000);              // wait for 5 seconds
    digitalWrite(12, LOW);    // Cut the power to the solenoid off by making
the voltage LOW, lock pops back up
  }

  // Check if card is Postal Worker's, if so open lock and send message:
  if(cardID == 3492602243){
    digitalWrite(12, HIGH);   // Energize the solenoid to retract the lock
(HIGH is the voltage level)
    // Run python script to send email alert:
    linuxProcess.runShellCommand("python " + path + "mailbox_opened.py");
    while(linuxProcess.running());

    delay(3000);              // wait for 3 seconds
    digitalWrite(12, LOW);    // Cut the power to the solenoid off by making
the voltage LOW, lock pops back up
  }
}// End of main loop
```

One issue that arose when the various bits of code were combined had to do with the default settings for the PN532 chip in the RFID reader. Preliminary testing revealed the program was getting stuck when the RFID reader was checking to see if a RFID tag was present with the *readPassiveTargetID()* function. This halt in the program's execution was not allowing the PIR sensor to check for motion until after a RFID tag was presented, and even then the program's execution would halt again as soon as the main *loop()* function came back around to *readPassiveTargetID()*. The Adafruit library for the RFID reader provided some insight into the issue: the default set up for the reader was to wait indefinitely for a RFID tag, continuously scanning until one was found. Fortunately, the library included a setup parameter *setPassiveActivationRetries()* that could be included in the Arduino's *setup()* function that allows the user to specify the number of retries the reader should attempt before moving on. Adjusting this setting fixed the issue and the mailbox was working as desired.

As mentioned, there are two Python scripts which are stored onboard the Micro SD card and used by the Linux side of the Yun. The first script, *upload_send_picture.py*, is used when a picture is taken with the USB camera:

```
# Author: Ross Tew
# File: upload_send_picture.py
# About: Script used to execute two Temboo Choreos, one
# to send an email with an emage attachment, and another
# to upload an image to Dropbox. Adapted from Adafruit's
# guide @ [22]

import base64
import sys
# Import the needed libraries from the Temboo Python SDK
from temboo.core.session import TembooSession
from temboo.Library.Dropbox.FilesAndMetadata import UploadFile
from temboo.Library.Google.Gmailv2.Messages import SendMessage

print str(sys.argv[1])

# Image needs to be encoded into base64 for uploading:
with open(str(sys.argv[1]), "rb") as image_file:
    encoded_string = base64.b64encode(image_file.read())

# Create a "session" with Temboo credintials:
session = TembooSession('jrtew', 'myFirstApp', '**********')

# Create the first "Choreo" for uploading a file:
uploadFileChoreo = UploadFile(session)
uploadFileInputs = uploadFileChoreo.new_input_set()

# Set the parameters for Temboo to use for the upload file Choreo:
uploadFileInputs.set_AppSecret("**********")
uploadFileInputs.set_AccessToken("**********")
uploadFileInputs.set_FileName(str(sys.argv[1]))
uploadFileInputs.set_AccessTokenSecret("**********")
uploadFileInputs.set_AppKey("**********")
uploadFileInputs.set_FileContents(encoded_string)
uploadFileInputs.set_Root("sandbox")

# Execute the Choreo:
uploadFileResults = uploadFileChoreo.execute_with_results(uploadFileInputs)


# Repeat the same as above but for the send message Choreo:
sendMessageChoreo = SendMessage(session)
sendMessageInputs = sendMessageChoreo.new_input_set()

# Set paratmeters:
sendMessageInputs.set_AttachmentContentType("image/png")
sendMessageInputs.set_AttachmentFileContent(encoded_string)
sendMessageInputs.set_AttachmentFileName(str(sys.argv[1]))
sendMessageInputs.set_ClientID("552863990646-
9urgopavnoumrc29opj0smjld7fiqfpc.apps.googleusercontent.com")
sendMessageInputs.set_ClientSecret("**********")
sendMessageInputs.set_From("tew_jonathan@columbusstate.edu")
```

```
sendMessageInputs.set_MessageBody("Motion detected, picture taken.")
sendMessageInputs.set_RefreshToken("**********")
sendMessageInputs.set_Subject("Motion Alarm Activated")
sendMessageInputs.set_To("tew_jonathan@columbusstate.edu")

# Execute Choreo:
sendMessageResults = sendMessageChoreo.execute_with_results(sendMessageInputs)
```

After the picture is stored onboard, the script runs, passing the image data along with the account information through Temboo, which then sends an email with the picture as an attachment and uploads it to Dropbox.

The second script, *mailbox_opened.py*, runs after a Postal worker is positively identified as opening the mailbox with their RFID tag:

```
# Author: Ross Tew
# File: mailbox_opened.py
# About: Script used to execute a Temboo Choreo for
# sending an email notification that the mailbox has
# been opened by a Postal Worker.

import sys
# Import the needed libraries from the Temboo Python SDK
from temboo.core.session import TembooSession
from temboo.Library.Google.Gmailv2.Messages import SendMessage

# Create a "session" with Temboo credintials:
session = TembooSession('jrtew', 'myFirstApp', '**********')

# Create a "Choreo" for sending a message through Temboo:
sendMessageChoreo = SendMessage(session)
sendMessageInputs = sendMessageChoreo.new_input_set()

# Set the parameters for Temboo to use for the send message Choreo
sendMessageInputs.set_ClientID("552863990646-
9urgopavnoumrc29opj0smjld7fiqfpc.apps.googleusercontent.com")
sendMessageInputs.set_ClientSecret("**********")
sendMessageInputs.set_From("jrtew2@gmail.com")
sendMessageInputs.set_MessageBody("A U.S. Postal Service employee has opened
your mailbox.")
sendMessageInputs.set_RefreshToken("**********")
sendMessageInputs.set_Subject("You've got mail!")
sendMessageInputs.set_To("tew_jonathan@columbusstate.edu")

# Execute the Choreo:
sendMessageResults = sendMessageChoreo.execute_with_results(sendMessageInputs)
```

This script passes the Temboo account information along with the desired Gmail information to Temboo, which then triggers an email to be sent that will notify

the user a Postal worker has opened their mailbox. This script is very similar to the second half of the *upload_send_picture.py* script, using Temboo to send a simple Gmail message.

4.3 Installing the Hardware

The last step for creating the prototype was to install all of the hardware inside the mailbox. One important consideration when choosing the mailbox to use for the prototype was the material it was made from. The RFID reader was to be mounted inside the door of the mailbox, so it was important the material of the door allowed for reading RFID tags through it. The majority of mailboxes are either metal or plastic, so a plastic mailbox was selected from a local home improvement store. The metal mailboxes would have interfered with the RFID readers signal, potentially preventing it from reading any tags through the door.

A product called 3M Dual Lock [38] was used to attach the RFID reader and the solenoid to the inside of the mailbox door, as shown by Figure 8. The Dual Lock fastens the reader and the solenoid securely to the door, but also allows them to be removed and re-attached easily – especially useful for the solenoid so that it could be removed for testing.

Two holes were created in the side of the mailbox to allow the camera and the PIR sensor to "look" out from inside. Figure 9 shows how the mailbox looks with the holes on the side. The bigger hole for the camera has a sheet of Plexiglas behind it to protect the camera. Additionally, two small holes were created in the back left corner of the mailbox to allow power cables to pass through for the

Arduino and the solenoid. The Arduino and the perfboard circuit were placed in the back left corner of the mailbox.

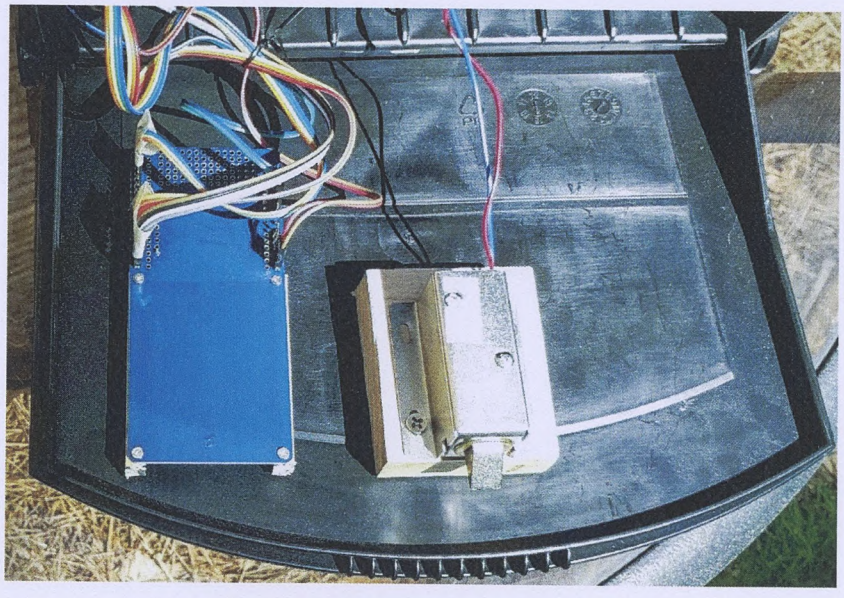

Figure 8: RFID reader and solenoid mounted inside mailbox door.

Figure 9: PIR sensor (left) and camera (right).

Finally, an aluminum bracket in the shape of an "L" was created and mounted to the roof of the mailbox, which can be seen in Figure 10. This bracket catches the locking armature part of the solenoid and keeps the door from being opened unless the solenoid retracts its armature.

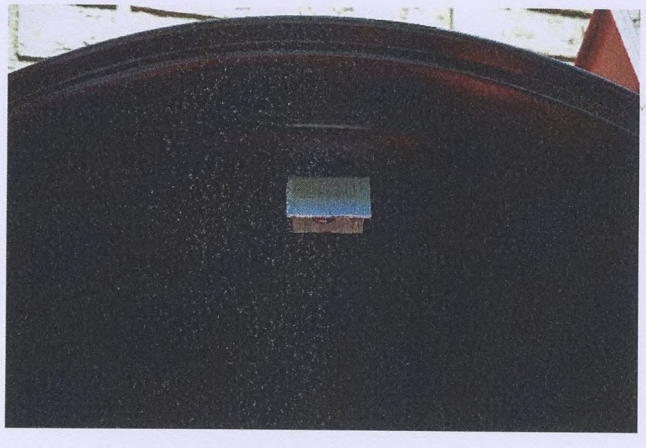

Figure 10: Aluminum bracket for locking the door.

With all of the hardware and software squared away at this point, the smart mailbox prototype was up and running. The next step in the research would be to begin testing and evaluating the prototype to make sure that it functioned properly and to work out any bugs that may arise through these tests and evaluations.

Chapter 5: Testing and Evaluation

Multiple tests were performed on the prototype to determine how well it worked. These tests focused on the following areas:

- Effectiveness of the PIR sensor
- Success rate of the picture upload/email and the mailbox opened notification systems
- The average time it takes to:
    - Upload a picture and notify the homeowner of visitors/intruders
    - Notify the homeowner of a mailbox opening.

5.1 PIR Sensor Effectiveness

The effectiveness of the PIR sensor was determined by testing both the maximum distance it could detect a human, along with the maximum angle it could detect a human. These two tests provided a way to compare how close the PIR sensor performs to the manufacturer's claims in a real-world setting.

Figure 11, Diagram A depicts how the maximum detection distance was tested. The sensor was set up outdoors with an LED light attached to it that lit up when motion was detected. The manufacture states a maximum detection distance of 21 feet, so the test subject started off at a distance of 30 feet and began walking towards the sensor. Once the LED lit up, indicating the PIR sensor had detected the subject, the distance was marked, then measured from the sensor. This test was repeated ten times, at the end of which the mean of the ten distances was taken. Table 2 shows the results from this testing. The resulting mean from these

tests was 152.9 inches, or twelve feet and 8.9 inches. The datasheet for the sensor claims a maximum detection distance of twenty-one feet [24], so the results here show a shorter actual detection distance. One thing to note: the PIR sensor's sensitivity potentiometer was turned down some in an effort to help decrease false positive detections while using the sensor outdoors – this likely decreased the maximum detection distance somewhat.

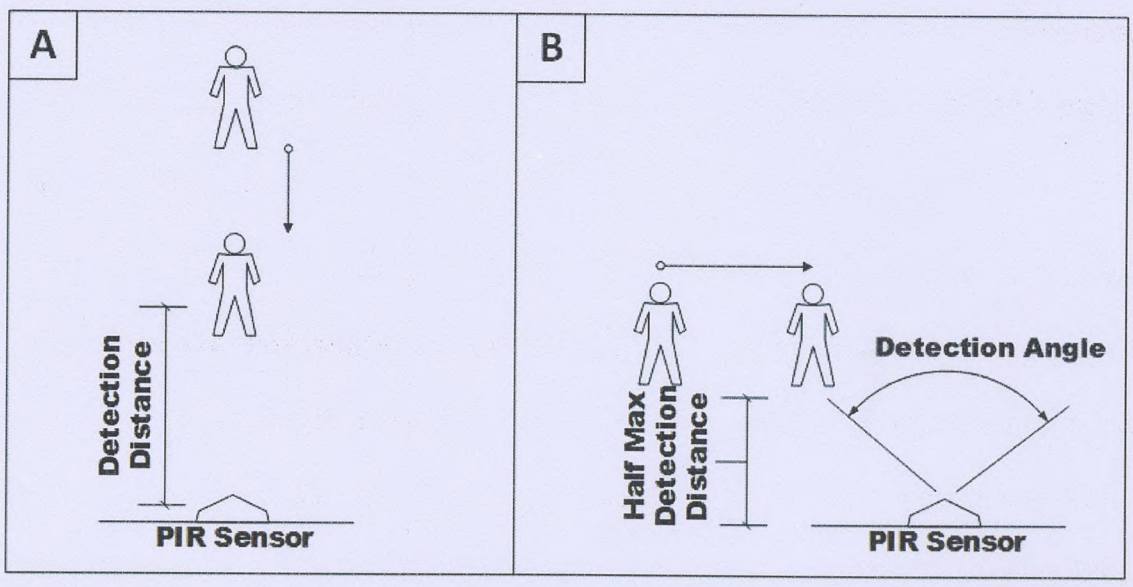

Figure 11: Diagram A – Testing the maximum detection distance of the PIR sensor. Diagram B – testing the maximum detection angle of the PIR sensor.

To determine the maximum detection angle, a similar test was performed as above and is depicted in Figure 11, Diagram B. The sensor was again set up outdoors, with an LED attached to notify when it had detected movement. The PIR manufacturer claims a detection angle of 120 degrees, so the test subject started at a distance equal to half the maximum detection distance from the previous test and an angle of 130 degrees. The subject walked parallel to the sensor until the

LED lit up, after which the spot where the subject was standing was marked and the angle of detection was measured. The test was repeated ten times, and then the mean of all ten results was taken. The results from this testing are shown in Table 3. The mean of the angles measured was 94.6 degrees, while the datasheet states a maximum detection angle of 120 degrees [24].

Table 2: Results from maximum distance testing of PIR sensor.

PIR Sensor Maximum Detection Distance Testing	
Test #	Detection Distance
1	147"
2	107"
3	144"
4	167"
5	152"
6	165"
7	163"
8	148"
9	193"
10	143"
Mean:	152.9" or 12' 8.9"

Table 3: Results from maximum detection angle of PIR sensor.

PIR Sensor Maximum Detection Angle Testing	
Test #	Detection Angle
1	94°
2	100°
3	96°
4	96°
5	92°
6	90°
7	94°
8	96°
9	90°
10	98°
Mean:	94.6°

5.2 Success Rate Evaluations

The focus of these tests will be to determine how reliable the system is in terms of the internet connected applications used in the smart mailbox. The system needs to be reliable whenever a picture is taken, then uploaded to Dropbox and

emailed as an attachment using Gmail, and when an email notification is sent through Gmail that a postal worker has opened the mailbox. The testing in this section will focus on the success rate of the system by determining how often these tasks fail.

To test the picture upload/email success rate, a test subject triggered the system to upload/email a photograph of them twenty different times. Throughout this test, two issues were monitored: how many times, if any, the picture failed to upload/email and how many times, if any, the picture was uploaded/emailed with any compression artifacts. Compression artifacts would indicate a loss of data [35] when the image is compressed to store on the Micro SD card. The most common of these compression artifacts is the blocking artifact, which is also the most noticeable [35]. A great amount of these artifacts may render the image unusable – something that must be avoided in this type of application. If there were enough compression artifacts to render the image unusable, it was marked as a failure.

Testing the email notification from the RFID system was done in a similar way: this feature was invoked using an RFID tag with the postal worker ID twenty different times. Any notifications that failed to work correctly were kept track of.

After all these tests were complete, the success rate for each test was calculated by dividing the number of successful uploads or notifications by the number of attempts. Table 4 and Table 5 show the results from both tests. Both sets of tests revealed a one hundred percent success rate, proving the reliability of these systems in the smart mailbox.

Table 4: Success rate results of picture upload/email system.

Picture Upload/Email Success Rate	
Test #	Successful (Yes/No)
1	Yes
2	Yes
3	Yes
4	Yes
5	Yes
6	Yes
7	Yes
8	Yes
9	Yes
10	Yes
11	Yes
12	Yes
13	Yes
14	Yes
15	Yes
16	Yes
17	Yes
18	Yes
19	Yes
20	Yes
Success Rate:	100%

Table 5: Success rate results of email notification for mailbox opened system.

Email Notification Success Rate	
Test #	Successful (Yes/No)
1	Yes
2	Yes
3	Yes
4	Yes
5	Yes
6	Yes
7	Yes
8	Yes
9	Yes
10	Yes
11	Yes
12	Yes
13	Yes
14	Yes
15	Yes
16	Yes
17	Yes
18	Yes
19	Yes
20	Yes
Success Rate:	100%

5.3 Performance Time Evaluations

The testing in this category will be directed at how long it takes the system to perform two of its main tasks: capturing/saving/uploading/emailing a photograph

and sending an email notification that the mailbox has been opened by a postal worker.

The process for testing these two tasks was straight forward: the 40 tests from the previous sub-section (20 tests for the picture system, 20 tests for the RFID email notification) were timed, averaging the time it took for each of the tasks. In order to obtain consistent start and stop times, a red LED on the Arduino board that is programmed to light up when the PIR sensor detects motion was used for the start of the timer. The timer was stopped once the email was received and the picture appeared in Dropbox. For the RFID email notification, the click of the solenoid retracting was used to start the timer, and the timer was stopped once the email was received. A laptop running Windows 8.1 and connected to the internet over WiFi was used as the device to check the emails and Dropbox.

The results for the picture processing test are shown in Table 6. The time it took for the picture to be emailed and uploaded ranged from just under twelve seconds to just over twenty-two seconds, with most results being on the lower end of that range. The mean of all twenty tests worked out to be exactly thirteen seconds. The results for the email notification test are shown in Table 7. These times were quicker, ranging from around two seconds up to around twelve seconds. The mean of these twenty tests worked out to be 6.20 seconds.

These tests may not represent how every user would experience the system, as there are a wide variety of factors that play a part in its speed: home internet connection speed, modem, router, and hardware on the device receiving

the notifications (cell phone, tablet, PC, etcetera) just to name a few. However, these tests give a solid indication of the typical time it will take for these different tasks to be accomplished by the system.

Table 6: Results from time evaluations for picture upload/email system.

Time Taken to Upload/Email Picture	
Test #	Time in Seconds
1	12.72
2	14.19
3	12.61
4	13.95
5	12.52
6	11.91
7	12.49
8	12.46
9	11.83
10	11.85
11	12.27
12	12.21
13	11.92
14	11.89
15	12.02
16	12.35
17	22.69
18	11.84
19	13.63
20	12.72
Mean:	13.00 Seconds

Table 7: Results from time evaluations for mailbox opened email notification system.

Time Taken to Email Mailbox Opened Notification	
Test #	Time in Seconds
1	2.00
2	4.86
3	3.60
4	1.90
5	11.67
6	1.87
7	1.90
8	10.31
9	1.98
10	2.92
11	10.98
12	2.26
13	12.70
14	2.24
15	12.34
16	2.22
17	12.11
18	12.01
19	1.95
20	12.08
Mean:	6.20 Seconds

Testing the time it takes for the system to perform these tasks was important, so that the user would be able to receive the different notifications as quickly as possible. The results from this testing proved that there were no serious delays from the use of any of the technologies being used. The resulting times are believed to be adequately fast for the two tasks.

5.4 False Positive Detection Testing

The focus of these tests will be to determine how reliable the system is in terms of false positive motion detection. As discussed under *Sensors* in section 3.2, the PIR sensor is designed in a way to cut down on false positives due to issues such as temperature swings and changes in sunlight. However, this does not mean the sensor is perfect. The tests in this section will show whether or not the PIR sensor is well suited for the task of detecting people passing by the mailbox so the camera can take a picture of any visitors or intruders.

In order to test the PIR sensor's ability to perform in real-world conditions, the system was set up in an outdoor environment that provided plenty of different conditions: temperature changes, sunlight changes, movement from small wildlife (e.g., birds), and windy conditions that may disturb surrounding foliage. These conditions provided plenty of opportunities for a false positive from the sensor. The mailbox was placed at an appropriate height of forty-three inches, which falls between the United States Postal Service's recommended guidelines for mailbox installations that dictate a height of forty-one to forty-five inches from the ground [36]. The plan for keeping track of the sensor's performance was to leave the

system in place for forty-eight hours, recording any motion capture the system picks up through the pictures stored on the Micro SD card. The picture file names are the date and time to the second for when motion was detected, which allows the ability to keep track of when a specific picture was taken just by looking at the file name. After the forty-eight hours was up, the plan was to study the pictures for any false positives – a picture with no one in the frame to trigger the motion detection system. A record was kept for the number of false positives, actual human detections, and total number of detections.

After the first thirty-three hours of the test, the Micro SD card was checked to see how the system was performing. Unfortunately, it was decided to cut the test short as a result of this check. The results are displayed in Table 8. Based on these results, it was clear the PIR sensor was not suited for use outside. The conditions during the testing period were a mix of clear skies and light cloud coverage, along with consistent, moderate wind which was most likely the main factor for the large number of false positives. The results from this testing were disappointing.

To confirm that the poor results of this testing were indeed caused by the outdoor setting, the mailbox was set up indoors in a low traffic area for forty-eight hours. The mailbox was checked on several times throughout this test. After reviewing the pictures stored on the Micro SD card, it was clear the only times the PIR sensor was triggered was during these checks. The results from the indoor test are shown in Table 9. The indoor testing proves the PIR sensor being used in

the mailbox is indeed unfit for outdoor applications. This is disappointing, as the sensors are not marked as “Indoor Only”, or make any mention of that. Upon closer inspection of the datasheet for the sensor, it does, however, mention that wind and light may cause problems. A sensor designed to work outdoors will be a necessity for the mailbox.

Table 8: Outdoor testing results for false positive motion detection.

False Positive Testing of Motion Detection System - Outdoors		
False Positive Detections	**Human Detections**	**Total Detections**
931	6	937

Table 9: Indoor testing results for false positive motion detection.

False Positive Testing of Motion Detection System - Indoors		
False Positive Detections	**Human Detections**	**Total Detections**
0	16	16

Chapter 6: Conclusions and Directions for Future Work

The goal for this project was to create a prototype for what is considered a smart mailbox, while also incorporating the idea of a digital address. This goal has been reached, as outlined by this paper which demonstrates that these different components and systems can be combined together to create one cohesive system that may be used for address digitization, mail notification, and home surveillance.

However, it must be noted that there is one caveat to this: a motion sensor *designed* to be used outdoors *must* be found and substituted for the model used in this project. Due to time and financial constraints, a suitable sensor was not able to be found before the close of this project. Sensors designed to work in outdoor motion sensing flood lights or drive way alarms may prove to be the right match.

If this project were to be continued, the next phase would be to focus on the sensor as the number one priority. The second priority would likely be Temboo. Temboo has worked well for this project so far, as it helped to simplify and streamline some of the work in regards to working with the Gmail and Dropbox APIs. However, this is an area that could be improved by working with those APIs directly and cutting out the middleman – Temboo. Another reason for this is the free version of Temboo only allows 250 choreo runs per month, where a choreo run is considered a request to their severs to perform a task such as sending an email or uploading an item to Dropbox. Every time the smart mailbox takes a picture, two of these choreo runs are used, which adds up fast.

Finally, the physical appearance of the mailbox could use some refinement – especially the interior. The wiring from the RFID reader and the solenoid could be run under a false floor to the back of the mailbox. An interior wall could be built to house the Yun, camera, and sensor out of view when the door was opened. These changes would result in a more aesthetically pleasing design, and, more importantly, an even better functioning smart mailbox.

Bibliography:

[1] A. Ustundag, E. Cevikcan, "Vehicle route optimization for RFID integrated waste collection system, International Journal of Information Technology & Decision Making", vol 7-4, 611-625, Dec. 2008.

[2] C. Albanesius, "Amazon testing drone delivery system", PC Magazine, 9-11, Jan. 2014.

[3] E. Fish, "Do it yourself with arduino: what it is, how to get started", PC World, vol 29-11, 20-21, Nov. 2011.

[4] "Mr. Postman", Simple Elements, accessed June 9, 2016, http://simpleelements.us/2.0/mr-postman/.

[5] "Method and system for tracking and processing items in personal mailbox", Patents, accessed June 10, 2016, https://www.google.com/patents/US8238602.

[6] J. Simpson, "Passive infrared technology", EC&M Electrical Construction & Maintenance, vol 113-9, C12-C14, Sept. 2014.

[7] L.E. Ferreras, "The driverless city", Civil Engineering, vol 84-3, 52-55, Mar. 2014.

[8] S. Cendrowski, "Alibaba to test drone delivery in three cities", Fortune.com, Feb. 5, 2015.

[9] "Drones could soon be delivering mail in the land down under", Fortune, last modified April 15, 2016, http://fortune.com/2016/04/15/australia-post-drone-delivery/.

[10] "NXP Tools", NXP Semiconductors, accessed June 9, 2016, https://nxp-rfid.com/tools/.

[11] "Two in three phones to come with NFC in 2018", NFC World, last modified February 12, 2014, http://www.nfcworld.com/2014/02/12/327790/two-three-phones-come-nfc-2018/.

[12] Haselsteiner, Ernst, and Klemens Breitfuß, "Security in near field communication (NFC)", in Workshop on RFID Security, 2006, pp. 12-14.

[13] "MIFARE—Standard Card IC—MF1 IC S50 Functional Specification", Philips Semiconductors—Product Specification—Revision 5.1, 2001, pp. 1-19.

[14] J. Lucas, "What is infrared?", Live Science, last modified March 26, 2015, http://www.livescience.com/50260-infrared-radiation.html.

[15] "How infrared motion detector components work", Glolab Corporation, accessed June 9, 2016, http://www.glolab.com/pirparts/infrared.html.

[16] E. Huang, "An explanation of the USB Video Class (UVC)", Synopsys, last modified December 6, 2012, https://blogs.synopsys.com/tousbornottousb/2012/12/06/an-explanation-of-the-usb-video-class-uvc/.

[17] Temboo, accessed June 9, 2016, https://temboo.com/.

[18] "Adafruit CC3000 WiFi Shield", Adafruit, accessed July 1, 2016, https://www.adafruit.com/products/1534.

[19] "Weatherproof TTL serial JPEG camera", Adafruit, accessed July 1, 2016, https://www.adafruit.com/products/613.

[20] "RS-232 vs. TTL serial communication", SparkFun Electronics, last modified November 23, 2010, https://www.sparkfun.com/tutorials/215.

[21] "Mr. Postman – the smart, secure, wi-fi enabled mailbox", Kickstarter, accessed July 8, 2016, https://www.kickstarter.com/projects/1033096239/mr-postman-the-smart-secure-wi-fi-enabled-mailbox?ref=search.

[22] M. Schwartz, "Wireless secuirty camera with the Arduino Yun", Adafruit, accessed July 12, 2016, https://learn.adafruit.com/wireless-security-camera-arduino-yun?view=all.

[23] L. Ada, "PIR motion sensor", Adafruit, accessed July 12, 2016, https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor?view=all.

[24] "HC-SR501 PIR motion detector", Datasheet, pp.1-3.

[25] "RFIDuino lock box getting started guide", Robot Geek, accessed July 8, 2016, http://learn.robotgeek.com/getting-started/41-rfiduino/178-rfiduino-lock-box-getting-started-guide.html.

[26] "Arduino Yún", Arduino, accessed July 1, 2016, https://www.arduino.cc/en/Main/ArduinoBoardYun.

[27] S. Tatham, "PuTTY: a free SSH and telnet client", last modified March 10, 2016, http://www.chiark.greenend.org.uk/~sgtatham/putty/.

[28] "Getting started with the Arduino Yun", Arduino, accessed July 1, 2016, https://www.arduino.cc/en/Guide/ArduinoYun.

[29] S. Nasir, "Access Linus server of Arduino Yun with PuTTY", last modified March 17, 2015, http://www.theengineeringprojects.com/2015/03/access-linux-server-arduino-yun-putty.html.

[30] fritzing, accessed July 12, 2016, http://fritzing.org/home/.

[31] "Bridge library for Yun devices", Arduino, accessed July 12, 2016, https://www.arduino.cc/en/Reference/YunBridgeLibrary.

[32] "PN532 NFC/RFID controller breakout board – v1.6", Adafruit, accessed July 12, 2016, https://www.adafruit.com/products/364.

[33] "Adafruit PN532 NFC/RFID controller shield for Arduino + extras", Adafruit, accessed July 12, 2016, https://www.adafruit.com/products/789.

[34] "Arduino library for SPI and I2C access to the PN532 RFID/near field communication chip", GitHub, last modified May 27, 2016, https://github.com/adafruit/Adafruit-PN532.

[35] M. Shen, C. Kuo. "Review of postprocessing techniques for compression artifact removal." Journal of Visual Communication and Image Representation 9, no. 1, 1998, pp. 2-14.

[36] "Mailbox guidelines", United States Postal Service, accessed July 12, 2016, https://www.usps.com/manage/mailboxes.htm.

[37] P. Heron, "Ubuntu manuals – fswebcam", Ubuntu Manpage Repository, accessed July 1, 2016, http://manpages.ubuntu.com/manpages/xenial/en/man1/fswebcam.1.html.

[38] "3M Dual Lock reclosable fasteners", 3M, accessed July 12, 2016, http://solutions.3m.com/wps/portal/3M/en_US/Adhesives/Tapes/Brands/Dual-Lock-Reclosable-Fasteners/.