Facial Recognition Safe

Edward Etheridge IV
Department of Computer Science
Binghamton University
Vestal, NY
eetheri1@binghamton.edu

Baptiste Saliba
Department of Computer Science
Binghamton University
Vestal, NY
bsaliba1@binghamton.edu

Richard Zheng
Department of Computer Science
Binghamton University
Vestal, NY
rzheng14@binghamton.edu

ABSTRACT

The Internet of Things, IoT for short, describes the wireless communication between multiple “things” in order to accomplish a common goal. For this project, we attempted to create a locking safe that could be opened via facial recognition. In order to demonstrate the project, we created a prototype safe out of Legos with an attached Logitech USB webcam. We used a 12V Solenoid Lock controlled via a Raspberry Pi 3 A+ to open and close the safe. Our facial recognition was done on a Heroku Server with Django and OpenCV. We created an iOS application to send an image to the web server and open the safe. While we were unsuccessful in establishing communication between the iOS application and the Heroku Server, we are able to take an image using the webcam attached to the Raspberry Pi, verify the identity of the individual in the photo, and open the lock.

KEYWORDS

Internet of Things, Raspberry Pi, Solenoid Lock, iPhone X, Facial Recognition

1 Introduction

The Internet of Things has recently seen a large increase in its popularity due to its wide range of applications in the real world. Most consumer products nowadays try to incorporate some sort of IoT in order to provide more value to the user than ever before; e.g. Smart Refrigerators, Alexa. This revolution, nicknamed the Internet of Things, describes the communication between wireless devices to fulfill a common goal.

2 Implementation

In this section, we discuss the hardware and software we used for the project, and how we combined them together to create our final product.

2.1 Hardware

In this subsection, we specifically discuss the hardware components of our project and show figures for visual aid.

2.1.1 Raspberry Pi 3 A+. The Raspberry Pi 3 A+ was the microcontroller we chose in order to handle automating the opening and closing of the lock. Our only requirement for the microcontroller was that it could connect to WIFI and allowed GPIO pins to be controlled, which most Raspberry Pi’s do, therefore, we did not need to be picky in choosing a microcontroller. Figure 1 below shows an image of the Raspberry Pi 3 A+.

Figure 1: Raspberry Pi 3 A+

2.1.2 Solenoid Lock. In order to keep the safe locked, we decided to attach a solenoid lock on the door. Figure 2 below shows an image of the solenoid lock we chose for this project.

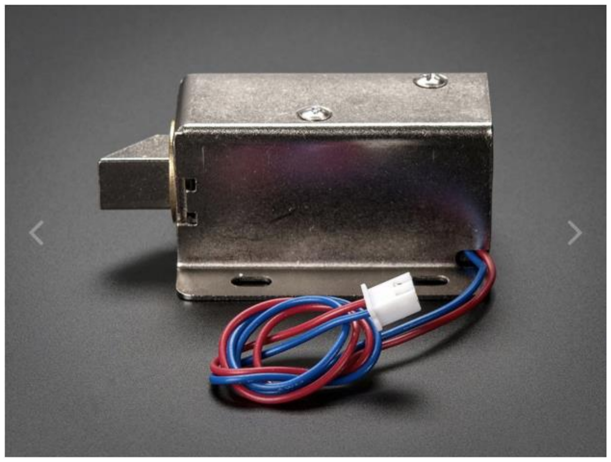

Figure 2: 12VDC Solenoid Lock

When 9-12 VDC is applied to the lock, the slug is pulled inwards, essentially allowing the lock to pass by the door. To use the lock in our project, we needed to rotate the slug 90° to align with the door of the safe.

2.1.3 Logitech USB Webcam. To gain access to the safe, our first method of entry is via a standard USB webcam attached to the safe.

Figure 3: Logitech C270 HD Webcam

Shown above in Figure 3 is the Logitech C270 HD webcam we used for this project. The webcam can capture images with a resolution of 1280x720 and videos at 30fps, although for this project we only cared about images. The webcam directly connected to the Raspberry Pi 3 A+ via USB.

2.1.4 Lego Safe. Originally, we were going to build a safe out of a more protective material but for the sake of time, we decided to make this project more of a proof of concept. Therefore, we constructed a simulation of a safe using Legos. On the right wall of the safe there is a slight hole which is where the solenoid lock slug sits inside when the safe is locked. The safe, shown in Figure 4 below, is 7.5"x5.5"x5".

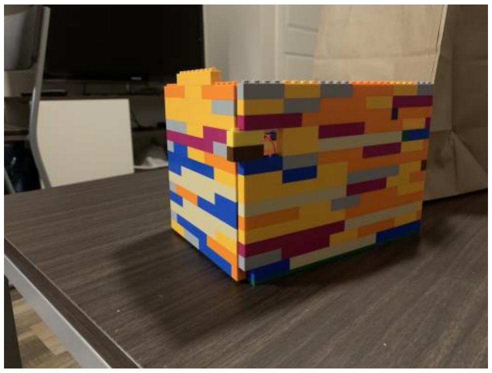

Figure 4: Lego Safe

2.1.5 iPhone X. The iPhone X was the first phone to contain Apple's TrueDepth Camera. The 7-megapixel front facing camera is further supported by an infrared camera and several other sensors. These sensors develop a system that allows the iPhone X to take videos at 1080p and provide wide color capture. This system is also used for Apple's own Face ID — their own facial recognition technology. These sensors and cameras, along with the phone being readily available to us, made it our best choice.

2.2 Software

In this subsection, we describe the software that made this project possible and successful. The software is split into 3 categories: Raspberry Pi code, web server code, and Swift code.

2.2.1 RPi.GPIO Library. To control the GPIO pins on the Raspberry Pi, we utilized the RPi.GPIO library [2]. The library, written in Python, allows for simple control of every GPIO pin. For this project, we controlled pin 4 (BCM Numbering) to change it from outputting 0v to 3.3v. The Raspberry Pi code can be viewed online for free by visiting [10].

2.2.2 fswebcam. To take images using a standard USB webcam on the Raspberry Pi, we utilized the fswebcam library [3]. This library allowed us to capture a 1280x720 image which we would eventually use to send to our web server for verification.

2.2.3 Django. This is the web python web framework that was used for the web application. Django allowed for the handling of HTTP requests and responses, as well as hosting the server on specified ports. By using Django, we were able to create a web application with a specific post route to upload images.

2.2.4 OpenCV. In order to identify the images of the user, we utilized the facial recognition facial recognition library OpenCV. We first used this library to train a facial recognition model by using the Recognizer class which OpenCV provided. With this trained model we can now identify faces from the images that are uploaded to the server.

2.2.5 Heroku. Heroku is the cloud platform we used to host our web application. One alternative platform could have been AWS, but as a personal preference, we decided to use Heroku. Heroku allows us to upload an image to our web server from anywhere by providing our web application a designated domain as well as server space for uploaded files.

2.2.6 Docker. In order to place the web application onto the Heroku server, it was necessary to use docker to virtualize an operating system and provide a docker container to the Heroku server. This docker container will specify the prerequisites that must be installed so that OpenCV will work. Additionally, it allows us to specify the database to use, which our case is a PostgreSQL database.

2.2.7 iPhone App/XCode. Shown below in Figure 5, our Facial Recognition App runs on an iPhone X. This app opens to a "snapchat"-esque interface. This app would then allow the user to take a picture of themselves. This app would then save the picture into the photo album. XCode provides easy ways to map buttons to functions. It also allows us to create the app with a preview of the interface before launching. This was supposed to be the second method to access the safe.

Figure 5: Safe Recognition App

2.2.8 AVFoundation. AVFoundation is one of Apple's featured frameworks for working with audiovisual media for their device operating systems. In our instance, we use this library to access the iPhone X's True Depth camera to capture images for facial recognition. This framework provides us easy access to the camera and the photo album that are already built in.

2.2 Circuit Design

Our circuit is based from the circuit shown here [1]. However, we needed to make modifications due to complications with our hardware. More specifically, the transistor that we purchased had a gate threshold voltage of 4V, meaning it required 4V to drive the transistor. Unfortunately, the GPIO pins on the Raspberry Pi only output 3.3V so alone, the GPIO pins would not be enough to drive the circuit. Therefore, we needed to take some of the voltage from our 12V power supply to assist the Raspberry Pi in driving the transistor. The full schematic for our circuit is shown below in figure 6.

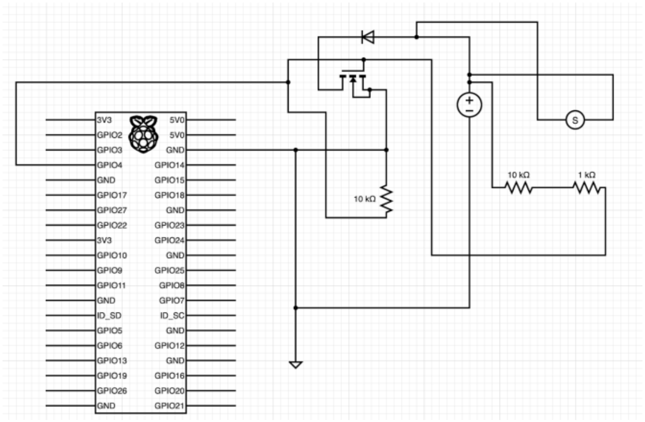

Figure 6: Circuit. The (s) in the circuit represents the solenoid lock.

The circuit is very similar to the reference [1], except for the addition of branching from the 12V power source. The circuit operates as follows: 12V from a DC power source feeds the solenoid lock and the source of the N-Channel Mosfet. This voltage goes across a diode rectifier to ensure that voltage does not travel in the opposite direction. The gate of the Mosfet is connected to GPIO 4 (BCM) of the Raspberry Pi along with an additional voltage from 12V. To be exact, measuring with a multimeter, we found that when the GPIO pin was set to input mode, the voltage from the gate to the drain of the Mosfet was 5.5V. When the GPIO pin was set to high, the voltage across the gate to drain was 3.3V. This was exactly what was needed as then we were able to control the transistor being open or closed via switching the GPIO pin to input or output. The drain of the Mosfet is connected to a 10k resistor to protect the Raspberry Pi. To get the correct voltage, we found that splitting some voltage from the 12VDC power source and running that in series with a 10k and 1k ohm resistor allowed for 3.3V to be supplied to the gate of transistor. Therefore, when the GPIO pin is set to output, the transistor is closed preventing the lock from being in a full circuit. When the GPIO pin is set to input, the transistor closes, and the lock receives the 12V it needs to open.

2.4 Overall System Design

The system operates as follows: a user runs the python script on the Raspberry Pi which sets the lock in place. The user then has the option to either press 'enter' to take a photo via the USB webcam or, use the iOS application to take a photo. In either case, the photo is then sent to the Heroku web server to be checked against the authorized faces. If the face is recognized, the web server sends a valid string to the Pi which then opens the lock for 7 seconds. If the user is not verified, the lock remains shut and they have 2 more attempts to verify their identity before the safe remains locked indefinitely. For safety and first-time setup, the user could just plug the safe in without running the python script, which would leave the lock open for the time being.

3. Evaluation

This project sought to create a functioning safe which utilized modern facial recognition algorithms as a means of identifying users and providing access to the safe's users. In order to accomplish achieve its intended purpose it must be able to run through the entire workflow without running into any errors. First, the safe must be able to take a picture of the user. Second, the Raspberry Pi must be able to send the image to the web server. Third, the web server must be able to run the image through a facial recognition algorithm and return, in an HTTP response body, whether the recognized face is a whitelisted user or not. Finally, the Raspberry Pi must be able to parse the HTTP response body and, based on the return value, unlock the solenoid lock. As we displayed with our in-class demo, we can accomplish this entire workflow, and can say that our project accomplishes its intended purpose. Despite the delayed response from the server to the Raspberry Pi, this design achieved our goals.

4. Conclusion

As a team we wanted to explore the various project possibilities which IoT offers. We thought it'd be interesting to explore: the various facial recognition libraries that are offered, how IoT can apply to personal security, and how sensors can interact with a web server to post images and data. We decided on this project in order to explore all these IoT functionalities. Despite the various difficulties that we encountered while creating this project we were able to create a fully functional safe which uses facial recognition while simultaneously exploring various fields which interested us.

ACKNOWLEDGMENTS

We would like to thank Dr. Mo Sha, Assistant Professor of Computer Science at Binghamton University, and our teaching assistants, Di Mu and Junyang Shi for their continuous support throughout the Fall 2019 semester.

REFERENCES

[1] Circuit Design https://www.circuito.io/app?components=9443,200000,842876
[2] RPi.GPIO Library - https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/
[3] Webcam - https://www.raspberrypi.org/documentation/usage/webcams/
[4] Docker Tutorial - https://www.youtube.com/watch?v=pGYAg7TMmp0
[5] Django Docker w/ OpenCV - https://www.youtube.com/watch?v=1pZbuvbvYY8
[6] OpenCV Basics - https://www.youtube.com/watch?v=-ZrDjwXZGxI&t=992s
[7] Using OpenCV Recognizer - https://www.youtube.com/watch?v=PmZ29Vta7Vc&t=1655s
[8] Swift camera: - https://medium.com/@rizwanm/https-medium-com-rizwanm-swift-camera-part-1-c38b8b773b2
[9] https://medium.com/@VojacekJan/deploying-swift-application-to-heroku-with-ease-2b81cdd07e6
[10] https://github.com/EetheridgeIV/FaceSafeRP
[11] https://developer.apple.com/av-foundation/
[12] https://github.com/rzheng14/FRSafe