eyeCU

A Project Report
Presented to
The Faculty of the Computer Engineering Department
San Jose State University
In Partial Fulfillment
Of the Requirements for the Degree
Bachelor of Science in Software Engineering

By
Arash Bahramiandehkordi
John Hoang
Justin Leung
Thi Tuong Vi Nguyen

05/2022

APPROVED FOR THE COLLEGE OF ENGINEERING

DocuSigned by:
Carlos Rojas
56CF228C1CA848C...

5/8/2022

Carlos Rojas, Project Advisor

Dr. Wencen Wu, Instructor

Dr. Rod Fatoohi, Computer Engineering Department Chair

ABSTRACT

EyeCU

By Arash Bahramiandehkordi, John Hoang, Justin Leung, Thi Tuong Vi Nguyen

Having unwanted bypassers approach one's home may warrant unease for some homeowners. California has an average of 3.7 million homes that are broken into each year. If homeowners are absent from their homes, this leaves their houses potentially vulnerable. A system that can identify familiar faces and identify who comes by the house can increase home security and ease the stress caused by trespassers. Not only will homeowners be able to know who comes by, but they will also be able to give access to registered individuals. This would also be convenient for those renting out their homes.

The main problem with the current security cameras on the market is that their functionalities are limited to recording and relaying real-time footage, unable to react to user interactions. These cameras track and record but they cannot differentiate individuals. How will the homeowner know if it's an expected guest or a total stranger? Hence, the ability to handle different types of authorized or unauthorized access is suitable for dealing with more creative types of intrusion.

This project implements a camera-based face recognition system that can be connected to the smart system of a house to perform adequate action depending on the situation. The system would collect the facial features of each authorized user and register them in the system with the allowed number of access. The events and system actions would be logged in separate files for examination purposes if applicable. On top of that, a list of handlers would be configured to deal with different types of access. The house owner would receive text notifications when unexpected events occur. The system can also be configured to use smart devices, such as a smart doorbell or smart floodlight, to scare away thefts and home invaders. The additional functionalities would create an impact on the traditional security cameras in the Internet of Things era.

Chapter 1. Introduction

1.1 Project Goals and Objectives

Security camera systems have gained in popularity, especially now that they are affordable for everyday use. However, the traditional security camera, which is meant for monitoring, is not sufficient to protect homes. Hence, this project aims to explore the idea of combining traditional security cameras, modern AI technology, and entry-granting systems. The project will be built and tested on a smaller scale for demo purposes, including a raspberry pi, a solenoid door lock, and an accompanied locally hosted web application that handles processing and accessing. The system will be tested with both a wired power source and a backup power source.

1.2 Problem and Motivation

The main problem with the current cloud-based camera security systems, especially the ones with wireless cameras, is the lack of options to react to video feed. For example the Ring doorbell can take more than thirty seconds to notify the registered users of occurred events. Even though the speed can normally be forgiven, it might be critical when a time-sensitive event, such as a break-in, happens. With the approach our project tries to accomplish, we tackle the weakest link in the system, which is the owner interaction. Using AI technology, the new system automatically handles entry authorization and unexpected events without the owner's instructions.

We've addressed this issue by implementing user control over the door's locking mechanism. When a face that has been registered into the system is detected, the user can visit the web application and through their linked account, control the solenoid door lock.

1.3 Project Application and Impact

Plenty of similar products, such as keyless entrances or digital doorbells, are floating around on the market, but none of them seriously tackle the AI potential of these devices. For example, smart doorbells such as Google Nest Hello or Arlo have built-in cameras, but the camera is sorely for monitoring purposes. If AI technology properly integrates into these products, it can recognize authorized and unauthorized individuals. The system can identify the unexpected guests that come to your house often when you are not home. It can help prevent some known attacks on FaceID such as x-glasses or liveness-relay. The system can always check if it is a real person by interacting with that person. "Please smile" or "can you turn

around?" can serve as perfect CAPTCHAs. Many functionalities come with more energy consumption, so the manufacturers would be motivated to develop better specific purpose cameras and hardware that are more energy-saving and IoT-friendly.

1.4 Project Results and Deliverables

Actual Project Results:

The final product of this project consists of three different components: a Raspberry Pi (rPi), a back-end server, and a front-end server. The raspberry pi has a pi camera and an electrical lock installed via a relay with an external battery. Internally, the device hosts a Flask server, which controls the solenoid lock, and a streaming server that takes care of sending video feed from the camera. The backend server houses the machine learning model for facial recognition and the main logic for granting access to the user. The backend communicates with the lock via rest APIs hosted on the rPi. On the other hand, the front end is used as the administration dashboard. It provides a user management interface where users' pictures and information can be added. There is also a monitoring mode where the camera feed with recognition data is rendered in real-time with SocketIO.

Project Deliverables:

The first milestone is the machine learning component. The model should be able to read the facial / expression data, train the current model as well as perform recognition tasks. The second stage is the mechanical component. A Raspberry Pi will control the movement of the lock arm based on some external signal from the cloud-based server. The next milestone is a cloud-based server deployment, which manages the entrance data as well as hosts a machine learning model. The final deliverable would focus on the final integration between the mentioned components and testing.

This project's product is a combination of both hardware and software components. On the hardware side, we have a Raspberry Pi controlling all the back-end logic. This includes hosting the facial recognition software, our web application back-end, and our solenoid door lock instance. The front end is built with React and the back end is a Python Flask web server.

We have 2 Flask instances running both on the Raspberry Pi. The first instance contains the web application back-end and facial recognition processing. Our second instance is only supporting the solenoid door lock which communicates with the back-end through endpoints and webhooks.

Front-end receives data from all three layers. The front-end communicates with the backend through REST API endpoints, receives both live video feed from the camera, and gets updates about the solenoid lock's status through SocketIO endpoints.

The facial recognition processing is running asynchronously. The breakdown is the camera captures a live feed, goes through the back-end for facial processing, and gets emitted into a SocketIO endpoint. The latency in having the front-end subscribing to the camera with this method is minimal and causes no impact.

In summary, the prototype is deployed as a backend and a frontend, both of which are deployed on the cloud, and a raspberry pi which is exposed to the internet via ngrok tunnel. The specifications, designs, and other technical details are included in the formal report.

1.5 Project Report Structure

Background and related work give insight into the available technologies that can be utilized in efforts of bringing this project together.

Project Requirements list the system's functional and non-functional requirements. These are the features and behaviors we aim to implement within our project. We have visual documentation of some of the main features as UML diagrams and state diagrams to explain the flow of logic. This section also documents the technologies we actually used to create the system and what is required to support our system during the implementation process.

System Design documents how we planned to build the system. Here, we discussed our project's architecture and design challenges and solutions that we ran into throughout the project's duration.

System Implementation documents our implementation phase. This section discusses what libraries we used, why we used them, and how it benefits us. It also explains our challenges when we started implementing our system and what we did to overcome them.

Chapter 2. Background and Related Work

2.1 Background and Technologies

In order to successfully execute this project, the team will use a variety of libraries that can assist in real-time facial recognition. Understanding tools such as OpenCV will assist in ML by using preexisting facial recognition libraries, which will help streamline the project.

We will be using HTML, CSS, and React for the front end of our application. The front end will be for users to interact with our system. For our backend, we will be using Python, Flask, and SQLAlchemy to handle the business logic of the application. SQLAlchemy will be used to store user data for their personal accounts and to register pictures of their faces for our facial recognition software to detect their faces. SQLAlchemy offers ORM which can streamline queries and manipulations to our database as we insert or pull data.

One major requirement is the ability to connect the video captured by the camera and Raspberry Pi to the web application. This is done by using SocketIO to stream video asynchronously between the video processing and front-end and how we can communicate between the physical system and the web application.

The project's tech stack is broken down into three categories:

- Hardware
    - USB camera / Raspberry Pi camera
    - Raspberry Pi
    - Solenoid door lock
- Facial recognition software
    - Python
    - OpenCV
    - SocketIO
    - Ageitgey's face_recognition library
- Web application
    - Python
    - SQLAlchemy / SQLite
    - Flask
    - HTML, CSS, React

Our hardware technologies will utilize a Raspberry Pi camera to act like an amateur security camera. We will pair that with a raspberry pi to be able to port our facial recognition

software over. On the software side, the facial recognition feature will be created with Python and the following libraries: OpenCV, and Ageitgey's face_recognition library. For the web application component, where users can communicate with the camera, there will be a Python-Flask application utilizing SQLite as a portable, lightweight database to store user data locally and use SQLAlchemy to act as an ORM intermediary for our queries.

2.2 Literature Search

Facial recognition has increasingly become intertwined with society. We use it to unlock our phones, scan through security, and prove authentication or authorization. The three-step process in how facial recognition works can be broken down into detection, extraction, and recognition (Kortli et al, 2020). Compared to other biometric systems, facial recognition is, unfortunately, the least reliable and efficient due to the many constraints and challenges (Kortli et al, 2020). Facial recognition proceeds through three different methods to achieve computer vision: local approaches, holistic approaches, and hybrid approaches (Kortli et al, 2020). Local approaches divide sections of a face into smaller regions connected by key points to detect points of interest in the face (Kortli et al, 2020). The holistic approach processes the face entirely. This approach presents the image on a matrix and utilizes vectors to determine facial features (Kortli et al, 2020). Hybrid approaches as the name suggest combine local and holistic approaches (Kortli et al, 2020).

Security cameras have become more prevalent in today's society as a means to offer security and protection. Even though security cameras have been around since the 1970s, the topic of how impactful they are is still being questioned. To prove that our security camera system can create an impact on our user's life, we looked into a study conducted by Jansen and co. as they test the hypothesis that being in the presence of a camera influences the behavior of individuals. The controlled experiment had participants play a board game under the assumption they were either monitored by a camera, told they weren't monitored but actually are, or not even monitored at all. They explored theories from criminology as those who are being watched had the possibility of being 'punished' if they break game rules to simulate authorities (Jansen et al, 2018). Their findings supported the idea that people's decisions and behaviors are influenced by the notion of being watched by cameras and monitors and promoted lawful executions (Jansen et al, 2018).

Innovation leads to closing the gap between AI and human vision capabilities. Babanne and co. documents their process of creating a system that can detect fires and other abnormal activities with AI (Babanne et al, 2019). The footage recorded by these systems would be useful in robbery, crime, and environmental detection before it even happens to save loss and in order to do that, machine learning and AI training must be utilized (Babanne et al, 2019). By using videos of past events, they apply investigation processes and strategies as a form of

machine learning to help the AI identify and predict events (Babanne et al, 2019). These types of training methods could be applicable to training our AI in recognizing the faces of people. Furthermore, we could train the system to detect certain behaviors so delivery people could be recognized.

Homes may now be equipped with digital door locks instead of traditional key locks. The lock will automatically unlock itself by entering the correct PIN on the number pad (Yu, 2018). These smart locks are often a mixture of battery-operated and physical key-operated (Yu, 2018). The purpose of these locks is to make it easier for homeowners to keep their doors locked when they are away through some kind of app or to make it easier to interact (Yu, 2018). With our security system project, like the digital lock pad, simply having a registered face detected by our cameras would unlock the door. Similar to entering a PIN to be granted access, the authentication here is more biometrics.

References

1. Jansen, A. M., Giebels, E., van Rompay, T., & Junger, M. (2018). The Influence of the Presentation of Camera Surveillance on Cheating and Pro-Social Behavior. Frontiers in psychology, 9, 1937. https://doi.org/10.3389/fpsyg.2018.01937

2. Kortli, Y., Jridi, M., Falou, A. A., & Atri, M. (2020). Face Recognition Systems: A Survey. Sensors (Basel, Switzerland), 20(2), 342. https://doi.org/10.3390/s20020342

3. V. Babanne, N. S. Mahajan, R. L. Sharma and P. P. Gargate, "Machine learning based Smart Surveillance System," 2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2019, pp. 84-86, doi: 10.1109/I-SMAC47947.2019.9032428.

4. Y. Yu, "A practical digital door lock for smart home," 2018 IEEE International Conference on Consumer Electronics (ICCE), 2018, pp. 1-2, doi: 10.1109/ICCE.2018.8326305.

5. Salama AbdELminaam D, Almansori AM, Taha M, Badr E (2020) A deep facial recognition system using computational intelligent algorithms. PLoS ONE 15(12): e0242269. https://doi.org/10.1371/journal.pone.0242269

2.3 State-of-the-art Summary

The leading technology that drives facial recognition technology is machine learning (ML). Computer vision greatly benefits through the usage of machine learning to amplify the processing capabilities of facial recognition. Leading brands like Ring and ADT offer IoT security cameras and doorbells which only serve surveillance purposes. One common issue with these commercial cameras is the lack of increased security. The ability to record and see who comes by your home is an advantage of its own but that's the only extent of it. With our product, we can incorporate the two worlds to establish a more secure way of entering homes and detecting visitors. We can create a smart security system with a digital padlock by incorporating facial recognition software into a security camera. Digital padlocks offer the ability to unlock a door by entering a PIN instead of inserting a physical key (Yu, 2018). By combining the two, we could create a smart security system where the digital padlock would unlock itself based on the authorization level of the face recognized through the camera. In case of a power outage, the lock will have a backup mechanism that enables traditional key and lock utilization.

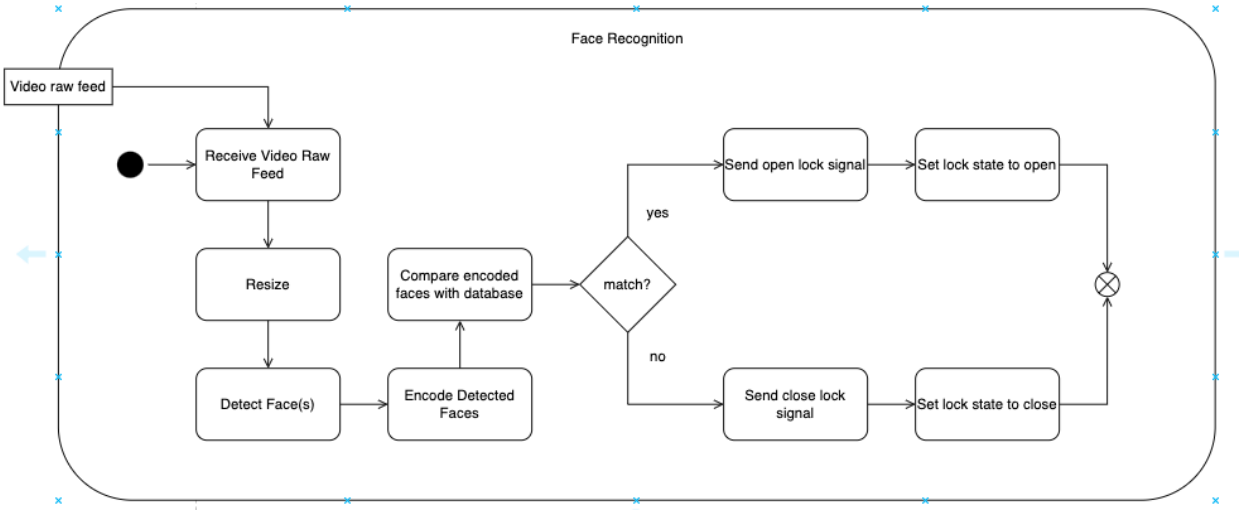
1. Y. Yu, "A practical digital door lock for smart home," 2018 IEEE International Conference on Consumer Electronics (ICCE), 2018, pp. 1-2, doi: 10.1109/ICCE.2018.8326305.

Chapter 3 Project Requirements

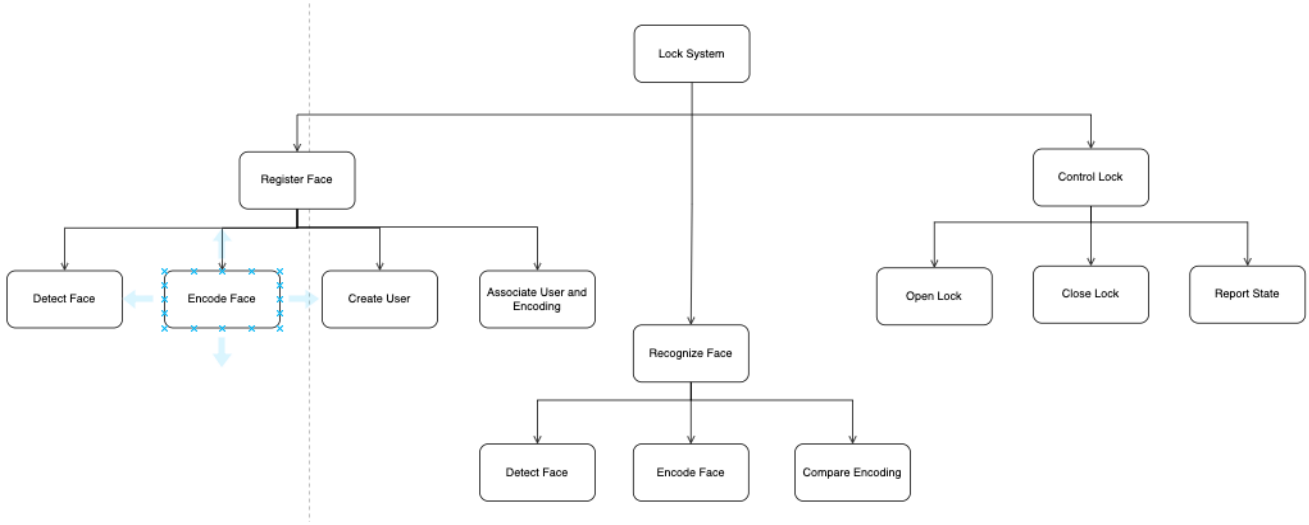
3.1 Domain and Business Requirements

UML 2 Activity diagram:

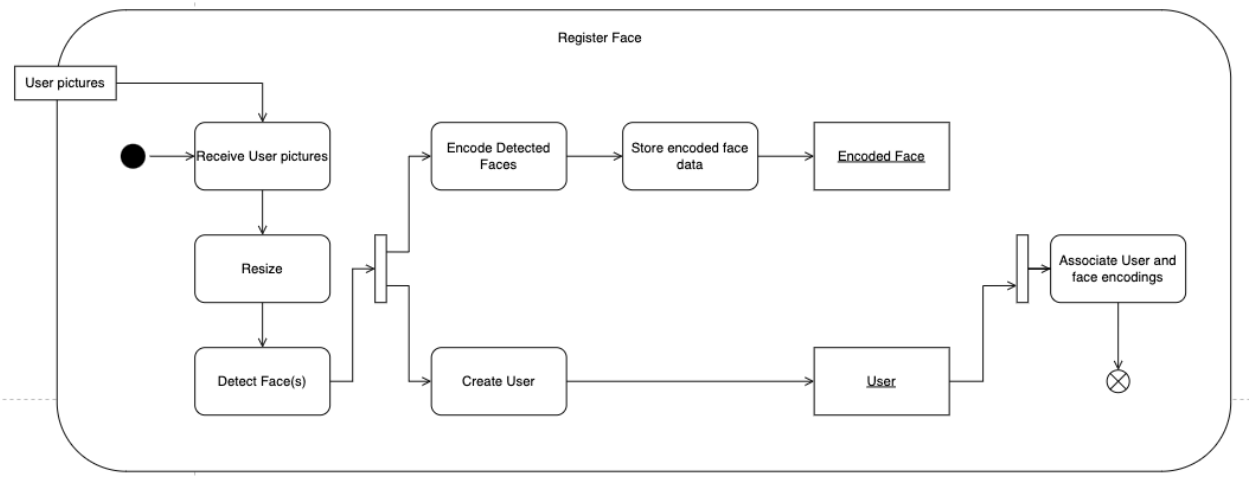
1. Face Recognition process:



2. Register Face process



3. Set of process decomposition diagram:



3.2 System (or Component) Functional Requirements

1. The camera shall be able to detect human faces.
2. Users shall be able to register for an account and upload a picture of their face
3. User faces shall be recognized by the camera through the facial recognition software
4. Users shall be able to unlock the associated deadbolt built into the system once their face is recognized by the system.
5. Users should be able to register more than one face for the camera to recognize.
6. The system should send a push notification when a detected face is in the frame.

3.3 Non-functional Requirements

1. The camera must detect the face in under 5 seconds.
2. The camera must match a detected face to a name within 10 seconds.
3. The camera should allow lock access within 10 seconds.
4. Face validation when uploading a picture of the user's face must finish within 5 seconds.

3.4 Context and Interface Requirements

The team is using Visual Studio Code to implement the entire software side of the project. The team is developing all aspects of the software on 3 different platforms: Windows 10, Linux, and macOS. Even though the operating systems are different, the project is accompanied by a virtual environment to hold all dependencies and libraries, allowing easy portability. We test our facial recognition feature and door locking mechanisms directly on our Raspberry Pi paired with the solenoid door lock.

3.4.1 Testing

Testing is done at three different levels: unit test, API test, and integration test. At the module level, each component such as face recognition service or entrance service is subjected to black-box testing in which a set of inputs and expected outputs are defined. The API testing focus on checking the core functionalities of the APIs making sure they perform the expected tasks. We use a combination of Postman and TestProject for testing the raw API and UI components for the front end. Finally, we make sure that each server sends and receives the requests to/from others with the right data format as a part of the integration test.

3.5 Technology and Resource Requirements

Requirements for back-end software are comprised of Python 3.8, Flask 1.1.4, Flask-SQLAlchemy 2.5.1, Flask-Cors 3.0.10, face_recognition 1.3.0, face-recognition_models 0.3.0.

Python is our main language, used to implement our back-end for our web application, the facial recognition feature, and controls the door lock.

Flask is our microframework that is used to support our web development. It's a lightweight Python framework that includes a multitude of libraries like WSGI and Werkzeug which acts as the webserver interface and request handling respectively.

SQLAlchemy is used on top of our SQLite database. SQLAlchemy is an Object Relational Mapper (ORM) that acts as an intermediary between our python application and our database. What SQLAlchemy offers is an abstract way of translating python classes into relational databases. We can utilize SQLAlchemy to map objects to respected database tables for queries.

Face_recognition API is a toolkit that makes use of dlib in a very easy way. Dlib, which is a c++ library with many advanced machine learning algorithms, becomes accessible through just a few python lines of python.

For the Node packages used, it was kept relatively simple. For styling, we're using React Bootstrap. For requests, we are using Axios and JWT decode for authentication for the backend.

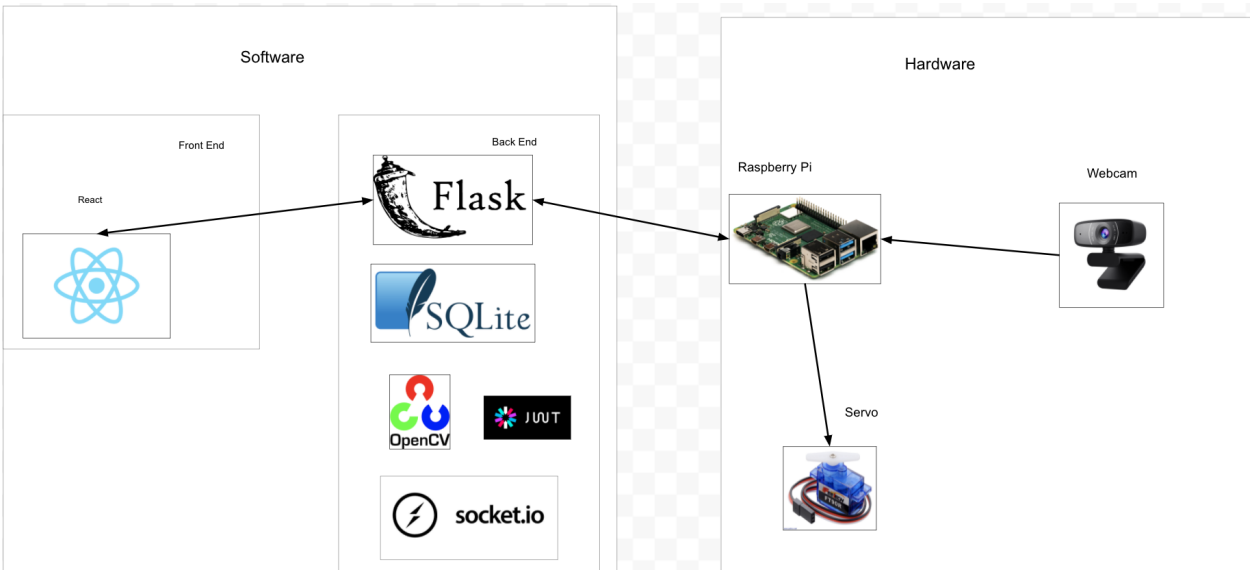
Fortunately, since the rPi has a Linux-based os on it, it makes porting our code onto it fairly easy. All we have to do is either copy our python code, or connect through a shell to our Github account, and we can run our project effortlessly.

Chapter 4. System Design

4.1 Architecture Design

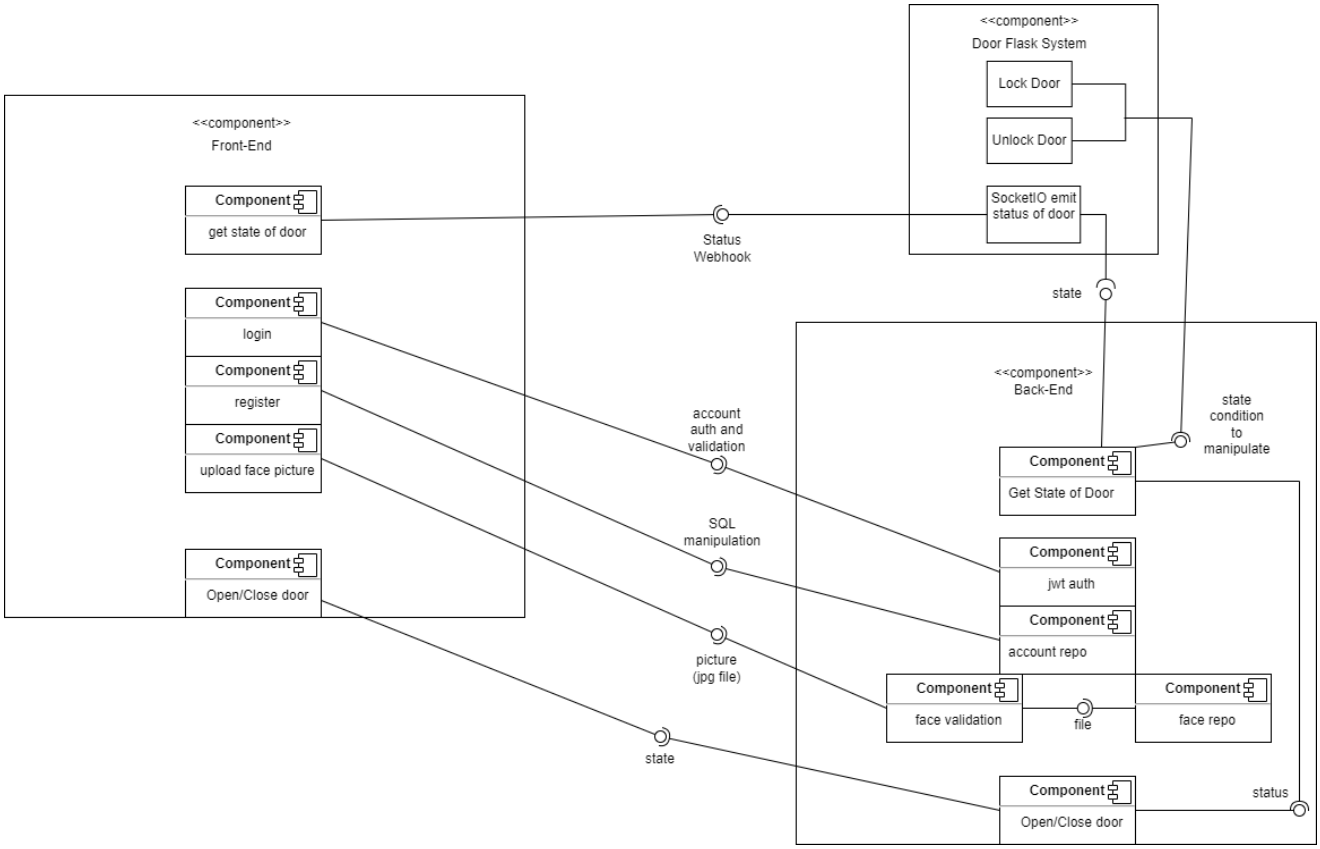
In terms of architecture, our design is relatively simple. Our hardware will be relying on a Raspberry Pi to communicate with our backend. The Raspberry Pi will be connected to a webcam that will constantly stream guests' videos to our backend database. It will also be controlling a servo that will lock the user's door. The backend, which is running a Python Flask instance, will take the video from the webcam and use facial recognition to identify guests. The backend will keep track of registered guests in an SQLite instance and communicate with the Raspberry Pi to open the door.

Our front end will be built using React. It will communicate with our backend to show users a live video feed and will allow users to manually open the door remotely. For capturing and processing image data, we'll be using OpenCV. Opencv is an open-source library used for real-time computer vision. We opted to use an open-source library, ageitgey's face recognition library, which gave us an advantage in spending more time integrating hardware and software components to form our projects rather than spending more time developing facial recognition software from scratch. This library renders frame by frame, comparing faces within the frame to locally stored photos of faces to detect and recognize a face.



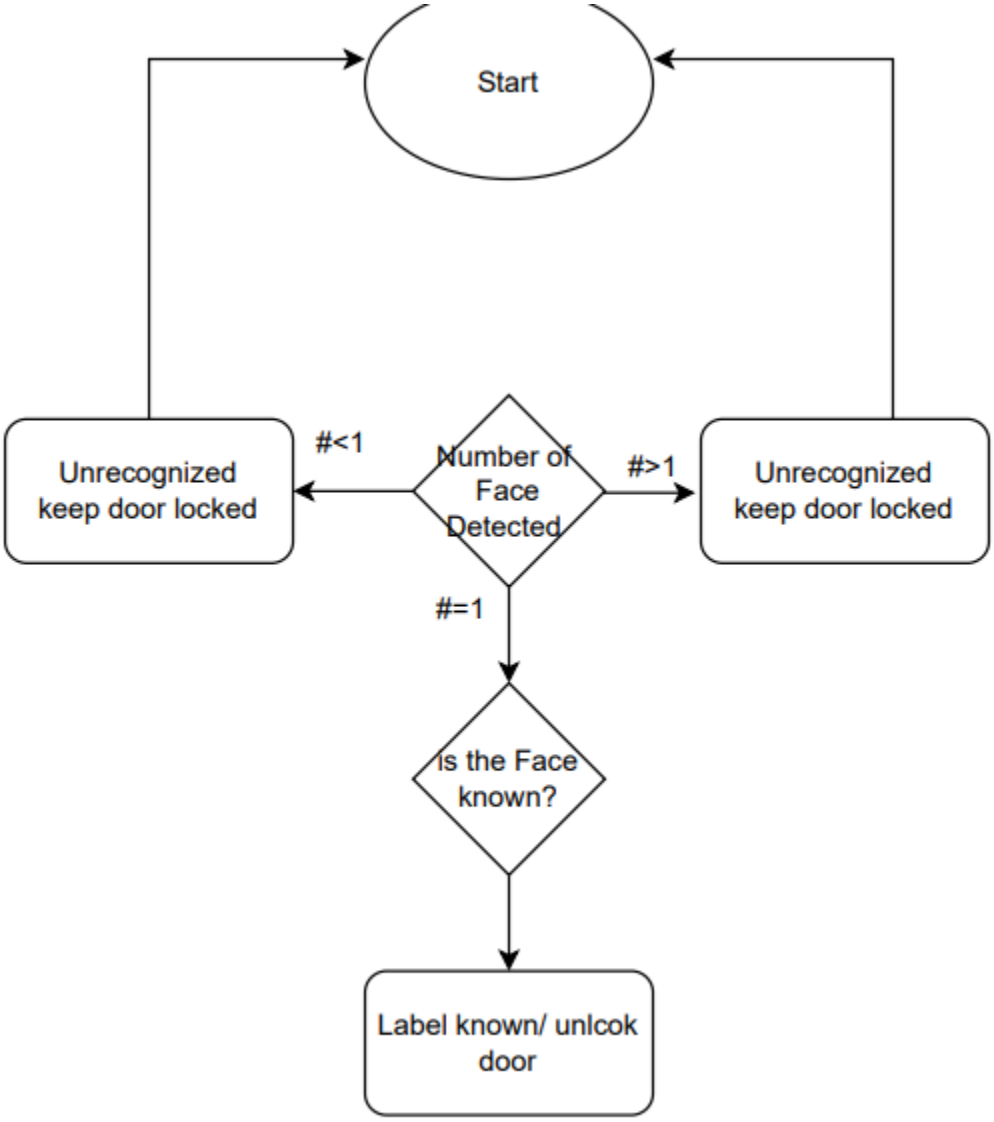
4.2 Interface and Component Design

Our system is comprised of three main components which are also comprised of several smaller components. Within our front-end, there are 5 smaller components that it handles. It requests an interface from our Door Flask system which returns the lock status. The login, register, and uploading facial picture components all send an interface into the back-end component in order to run the JWT authentication, access the account repository, and go through picture validation in order to access the face repository. The last component in the front end sends a status interface to the back end to update the Door Flask System component. But in order to do that, the back-end sends a series of requests to get the state of the door which responds with whether or not users will have the authorization to manipulate the lock.



4.3 Structure and Logic Design

A basic flowchart that shows our product's outcome, based on the number of faces our camera detects and recognizes.



4.4 Design Constraints, Problems, Trade-offs, and Solutions

4.4.1 Design Constraints and Challenges

As college students, we wanted to keep the physical components inexpensive. California is a two-party recording state which means that both parties must know they are being recorded otherwise the video recording will not be applicable in the court of law. With a majority of our group never working on machine learning or computer vision, we had to find specific libraries that would be easy to learn. One huge constraint that we are still working on is identifying possible test cases in terms of security in our application.

4.4.2 Design Solutions and Trade-offs

The project's total cost is relatively cheap because the only parts we had to purchase were the servos, Raspberry Pi, and webcam. We decided to use the cheapest webcams we could buy to make the project economically and resource-efficient. We plan on hanging a banner informing incoming guests that they are being recorded. The backend will not keep any footage, but we wanted to ensure that no laws are being violated for our project. We chose TensorFlow, OpenCV and SciKit learn because they have the most documentation for use cases like our application. In the event of a power outage, we will include a battery so the Raspberry Pi can lock the door. We'll be training our models on a decently powerful GPU so that the raspberry pi can just use a trained model for identifying individuals.

Chapter 5 System Implementation

5.1 Implementation Overview

Backend web application developed with Python Flask. Utilizing SQLAlchemy as our ORM and SQLite as our database storage engine. By using Flask we are able to implement our RESTful API in order to have communication with the front-end possible.

The face recognition module is developed around the face_recognition library with python. The unit is integrated as a service of the backend web application. It utilizes the video feed transferred from the PiCam installed on an external raspberry pi as well as communicates with the solenoid lock via the flask server hosted on the Pi. On the hardware side, the solenoid lock is connected to the Pi via a relay with an external power source. Even though the PiCam supports up to 60fps, the frame rate is adjusted to one frame per second to optimize the network throughput and save energy. The live feed is independently broadcasted on a different process on uv4l streaming service installed on the pi.

Capturing live video stream with the raspberry pi camera module. Using Raspbian as the debians-based operating system. Using pip and python 3 for the face_recognition API, which is responsible for recognizing homeowners' faces.

As covered previously, React is being used for the main front-end framework. Twitter’s bootstrap framework will be implemented as the main styling. Axios is used to communicate with the backend and Socketio will be used to get a live video feed from the backend and see the current state of the lock from the Flask instance running on it.

5.2 Implementation of Developed Solutions

The face_recognition API uses dlib, which is a c++ toolkit containing machine learning algorithms. The API was built to be very simple. There is a known folder that holds images of faces to be recognized. If the detected face matches with a picture in the known folder, then it will be labeled as recognized.

To control the lock, the flask server on the Pi listens to any incoming signal sent as HTTP requests. Depending on the type of request, a signal is sent to the GPIO interface (pin 17 in this prototype). Since a higher voltage is needed to control the electric lock, a relay with an external power source is utilized as a switch controlled with the weak signal from the GPIO pin.

5.3 Implementation Problems, Challenges, and Lessons Learned

One major challenge the team ran into was how are we going to pass the code that each teammate was responsible for. Using git as an industry standard, we still had an issue with dependencies and libraries that exist independently on each person's computer. We found that running applications inside virtual environments help contain library compatibility when working on different projects. This enabled us to share code without worrying about what that teammate had on their system. Using a script to download all required libraries, everyone could host code that others have done on their system regardless of how their environment may be initially set up.

When using the face_recognition API, we ran into an issue where people who were supposed to be recognized as unknown, got recognized as known. We're planning to create a harsher filter that consists of at least 60% of accuracy.

The solenoid lock gets hot really fast, so it is important to release it as soon as possible. The pi needs to be exposed to the internet so the webserver can communicate with it, so ngrok tunneling is implemented. Initially, the face recognition is designed to be placed on the pi, but, considering the performance of the device, it is moved to the backend server to maximize the processing as well as save the battery for the pi.

When we first started using Socket.io as our backend streaming package, we ran into issues with rendering HTML files with Flask. We realized that Socket.io was running synchronously so it was taking all the threads making the website unresponsive. After making Socket.io run asynchronously, our issues were fixed so we aim to introduce only asynchronous packages throughout our backend because it will be handling various calculations at once(updating SQL, identifying faces from a stream, and communicating with the front end and the locking mechanism).

Backend often ran into multithreading issues which caused a lot of segmentation faults. The main issue was the facial recognition was running on a sub-thread and ran into synchronization issues when the main thread was also calling the facial recognition API to perform validations. The resolution was to establish a signal from the main thread to indicate to the subthread when the front end was no longer showing the camera feed to kill that subthread, freeing up the API. The facial recognition thread is then recreated and reinitialized when the camera view was live again.

One major the team ran into during the initial implementation phase was following code patterns. Everyone has different backgrounds and habits in how they build and solve

problems. We had to establish code ownership and tried to match and follow the patterns each person had while establishing some overall general code patterns.

Chapter 6 Tools and Standards

6.1. Tools Used

There are several hardware options to pick including Servo, generic motor, and solenoid electric lock. We decided to use a solenoid together with a relay because the Raspberry Pi doesn't have enough power to operate the lock for long, especially in case of running on battery. Hence, having a separate relay with its own power source would solve the problem.

For the lock controller, there are multiple types of candidate devices including RaspberryPi, Adruino, and lightweight Teensy board. We end up using RaspberryPi because it facilitates a versatile platform for different frameworks like Python Flask and uv4l server. Even though Raspberry Pi consumes a bit more power while developing because it needs to render UIs, it should consume rather low energy while running on production mode with lite Raspian OS.

Because we only have two physical RaspberryPi on our team, the member without the device utilizes Azure RaspberryPi Emulator to develop their parts. However, the simulator couldn't fully support the newest OS, so there was a limitation on the configuration we could use, such as legacy piCam. In production, a new set of configurations is required to make the physical Pi work properly.

6.2. Standards

The product is built on top of golden standards for the Internet Of Things (IoT) specified by IEEE.

- Architectural framework: follow IEEE P2413-2019 to develop an architectural framework for the Internet of Things that promotes cross-domain interaction, and functional compatibility.
- Sensor Performance: follow IEEE 2700 to provide a common framework for sensor performance specification terminology, units, conditions, and limits.

More detail can be found here

Chapter 7 Testing and Experiment

7.1 Testing and Experiment Scope

Physical Device Test

1. Streaming Service Test

   When the service starts on the Pi, access localhost:8080 from the local browser or <Pi IP>:8080 from an external device in the same network to view the raw video feed. If the red light on piCam is on and the video is emitted properly, then the service is functional.

2. Lock Controller Test

   With the controller running, making a GET HTTP request to localhost:5001/open from the local browser or <Pi IP>:5001/open from an external device in the same network would open the lock. On success, the API would return a JSON with a success flag set to true, and the lock latch should be pulled in.

Backend Server Test

1. **Face Detection Test**

   Start the backend server when all services on the pi are running then open localhost:3000 in the local browser. A video feed of processed data should be displayed, and all faces are enclosed in red boxes.

2. **Face Recognition Test**

   Upload the users' images via the dashboard UI then follow the same steps in the section above. The name of recognized users, or "Unknown User" for unrecognized users, should be displayed under the bottom edge of the red boxes.

3. **Solenoid Functionality Test**

   Based on the status of the face in the camera, if the face is recognized by the processing software, users are allowed to click a button that triggers the lock to unlock. Users can manually lock the solenoid at any time but must be present in the camera to unlock it.

Integration Test

Make sure all services are running on both the backend and physical Pi. Submit the facial data of a user to the backend server via dashboard UI and point the piCam to that person's face. If the system works correctly, the machine learning model should recognize the user and send an HTTP request to the Pi to open the lock

7.2 Testing and Experiment Approach

We wanted to see how accurate the face recognition API was. The first step was to see if it can even detect faces. so we added a variety of pictures to analyze. We tried images containing only a person's face and ones that contain a group of faces within it. Then we ran our programs and camera to see if the person in live view would be detected. If a person were to be detected, a rectangular box would surround the person's face.

For our recognition test, we added images of the person that is soon to be recognized, and had multiple people including them, in view of the camera. If the recognition software were to work correctly, their name would be displayed underneath the detection box that was explained earlier.

Testing the integration of the rPI system with our web application took a few tries. The goal was to shoot live video from the rPI camera to the backend so it can perform the facial recognition processing. Once the processing is done, it shoots that video out to the front end for users to visit and see.

When a face is recognized through the system, users are allowed to trigger the solenoid lock to unlock. As of integration, this trigger is automatic as a way of testing if it can unlock when a face is detected.

7.3 Testing and Experiment Results and Analysis

To our surprise, all faces were detected pretty well on the first go. The rectangles that were supposed to surround individual faces had a high ratio of success. The only times we noticed that some faces weren't detected were because of blurry images or ones that had bad lighting.

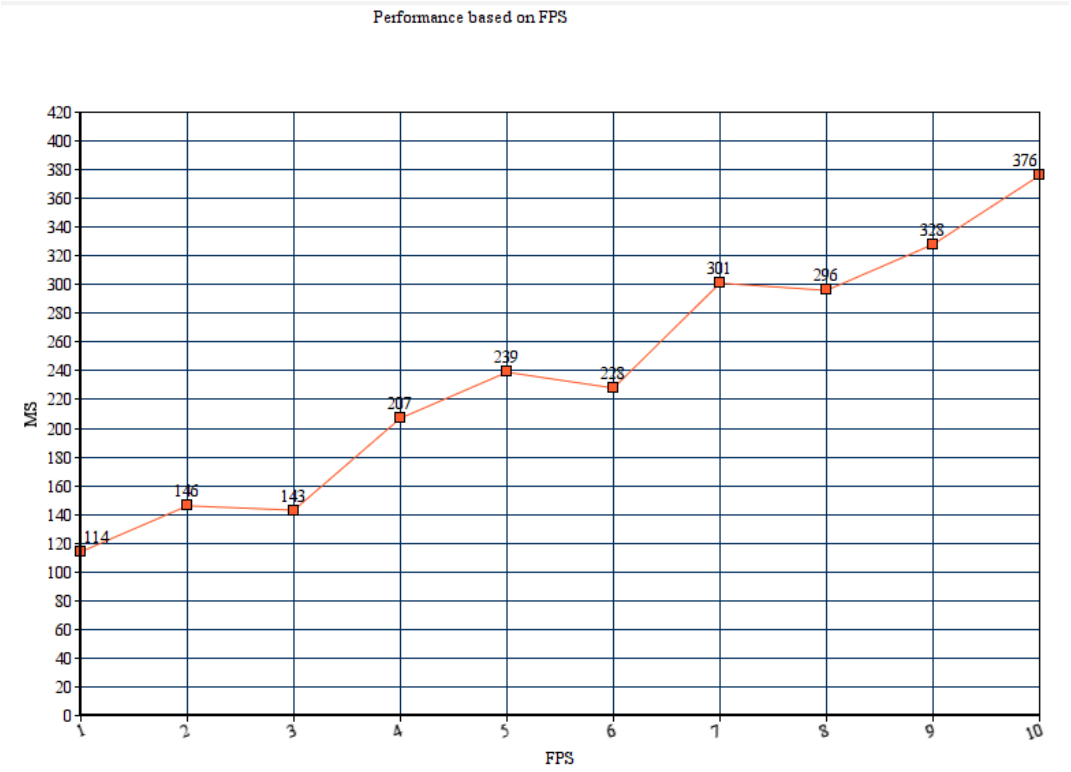
Recognition was a bit trickier. While we had successful outcomes, there were a few instances where multiple people were recognized as the same person. To counter this problem,

we increased the difficulty of the image comparison mechanism. Images are compared by being turned into tensors(matrices), then those matrices which contain numbers are compared to one another.

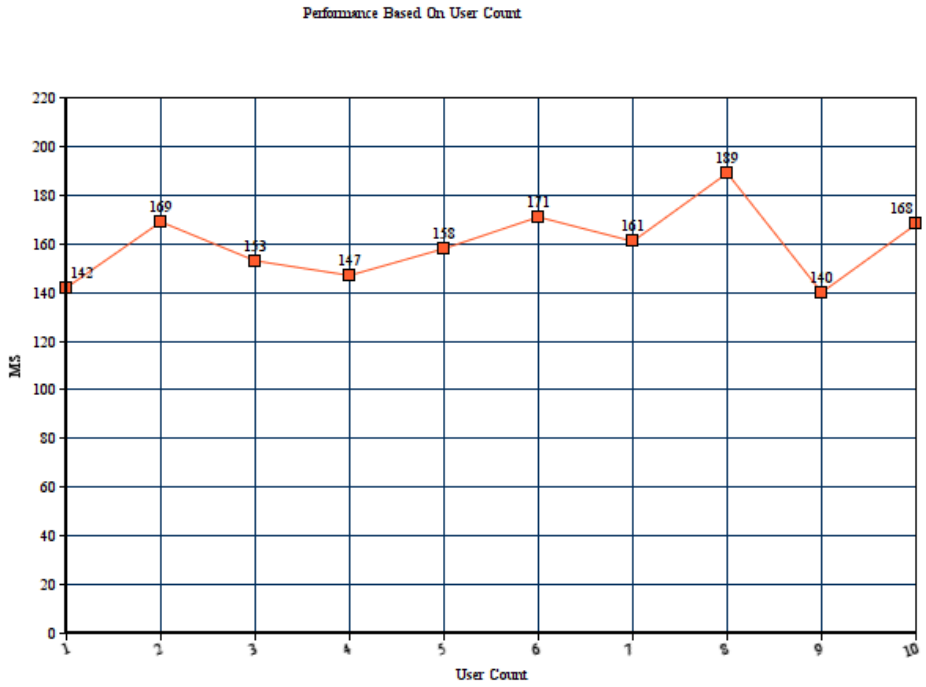
Integrating the rPI system and backend was a success. The video can be streamed to the front end with some slight issues. The video runs at 2 frames per second which cannot be dealt with. This was the unfortunate outcome none of us could expect. One other major issue is when the lock is triggered due to a face being recognized, we run into segmentation faults. This is due to the facial recognition library which utilizes C to perform processing. The HTTP requests to the lock messes with the API program counter and crashes it.

To evaluate the performance, we conduct several tests based on the number of data and fps. All the benchmarks are performed on a 2017 Macbook Air with dual cores and 8GB RAM.

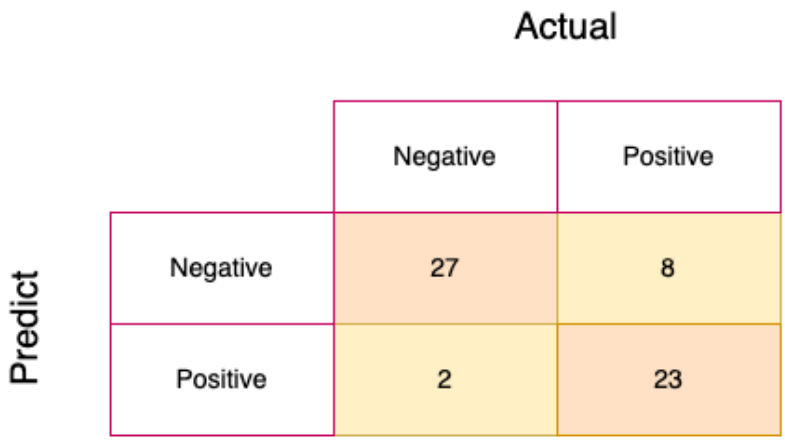
As the number of frames per second increased, the face recognition process got moderately slower. This might be a result of parallel requests reaching the server simultaneously. The problem could be solved by enforcing a lock on each device, but some of the frames might drop unprocessed.



Increasing the number of users, no significant bottleneck in performance is observed. There are spikes in the data, but it is most likely caused by a cold startup before running each iteration.



To test the accuracy of the model, we have several users, registered and unregistered, take turns to go in and out of the view of the camera for one minute and log the result. To simplify the test case, we limit one person per frame and configure the fps to one. The result shows the model has 83% accuracy and 92% precision.



Measure	Value	Derivations
Sensitivity	0.7419	TPR = TP / (TP + FN)
Specificity	0.9310	SPC = TN / (FP + TN)
Precision	0.9200	PPV = TP / (TP + FP)
Negative Predictive Value	0.7714	NPV = TN / (TN + FN)
False Positive Rate	0.0690	FPR = FP / (FP + TN)
False Discovery Rate	0.0800	FDR = FP / (FP + TP)
False Negative Rate	0.2581	FNR = FN / (FN + TP)
Accuracy	0.8333	ACC = (TP + TN) / (P + N)
F1 Score	0.8214	F1 = 2TP / (2TP + FP + FN)
Matthews Correlation Coefficient	0.6821	TP*TN - FP*FN / sqrt((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))

Chapter 8 Conclusion and Future Work

We couldn't implement everything that we wanted, however, all things considered, we managed to come through on our main objective. Users have their own account webpage, through which they can upload their images and view a live stream video of their front door. When they are recognized through the video feed, the door will be unlocked and allow them to enter.

Something to do in the future would be to improve the quality of our image recognition and our online technical stack. The use of the face recognition API made pipelining on the server much easier since it used python, however, we were having trouble with some of our own members being recognized as other members. Using UC Irvine's datasets to train and create our own models with TensorFlow, would probably yield higher accuracy.

There were some features that we would like to add to our project. Like keeping track of who interacted with the door via an activity log. There would be a sidebar next to the user's page and they would be able to know who was interacting in front of the camera and when.

Another aspect we intend to improve on is giving the user the ability to view the live stream from anywhere via an internet connection, and not just within a local network. This will enable the user to be able to check on their home and receive notifications of a possible break-in regardless of where they are. Due to the limitations of keeping this system running primarily on local endpoints, it became a struggle to find a way to push it all onto a cloud network. One main future work we would want to include is to establish a cloud network to allow any machine to connect to the endpoint to have a true remote connection. Currently, users are only able to access the web application through the local machine. We hope by pushing to the cloud, users can access the web application from anywhere on their mobile devices.

Hopefully, our improvements will allow us to scale to higher demand, make more accurate readings in face recognition, and make our users feel reassured and confident about feeling safer in their own homes.

This project left the team with a lot of exposure and experience with new technologies, ideologies, and practices. Hopefully, the knowledge gained from this project will help carry over into our future jobs within the industry and this project will become a stepping stone for us.