Project Overview

Motivation:
This project is to implement a locking mechanism that can be activated only by an authorized user, which is determined using facial recognition. This device would be useful in a situation where someone has forgotten their key or the code to their lock. Also, part of the project involves storing information about when the lock was unlocked and by whom. This could be a great security feature and could be expanded from the implementation in this project, which involves keeping a text log, to storing footage of individuals who access the lock or unauthorized individuals who attempt access.

Technical Approach:
I created a simple user interface, inside of an Electron [1] application, for the user to lock and unlock the lock. This was implemented using a single button that is active when an authorized user is present and inactive when one is not.

Figure 1 shows a diagram of the configuration for the components used for this project and how they are connected. For this project, my goal was to show the programming implementation of the lock, so I used the servo from the SunFounder PiCar-4WD to represent a locking mechanism. The servo at 0-degrees signifies that the “lock” is locked; the servo at 90-degrees signifies that the “lock” is unlocked. A Raspberry Pi 4 was used to control the servo and a Raspberry Pi Camera V2 was used to capture images for facial recognition. The following is a full parts list.

Parts List

- Servo, HAT module, battery holder : https://www.sunfounder.com/products/raspberry-pi-car-robot-kit-4wd
- Raspberry Pi 4: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/
- Raspberry Pi Camera Module V2: https://www.raspberrypi.org/products/camera-module-v2/
- 18650 Batteries x2

A client server model is used for communication between the user and the Raspberry Pi, where the client is a computer running the Electron app and the server is the Raspberry Pi. The Raspberry Pi is responsible for performing facial recognition and controlling the lock. The Electron app tells the user the status of the lock, i.e., locked or unlocked. The app also has a button which an authorized user can click to lock or unlock the lock.

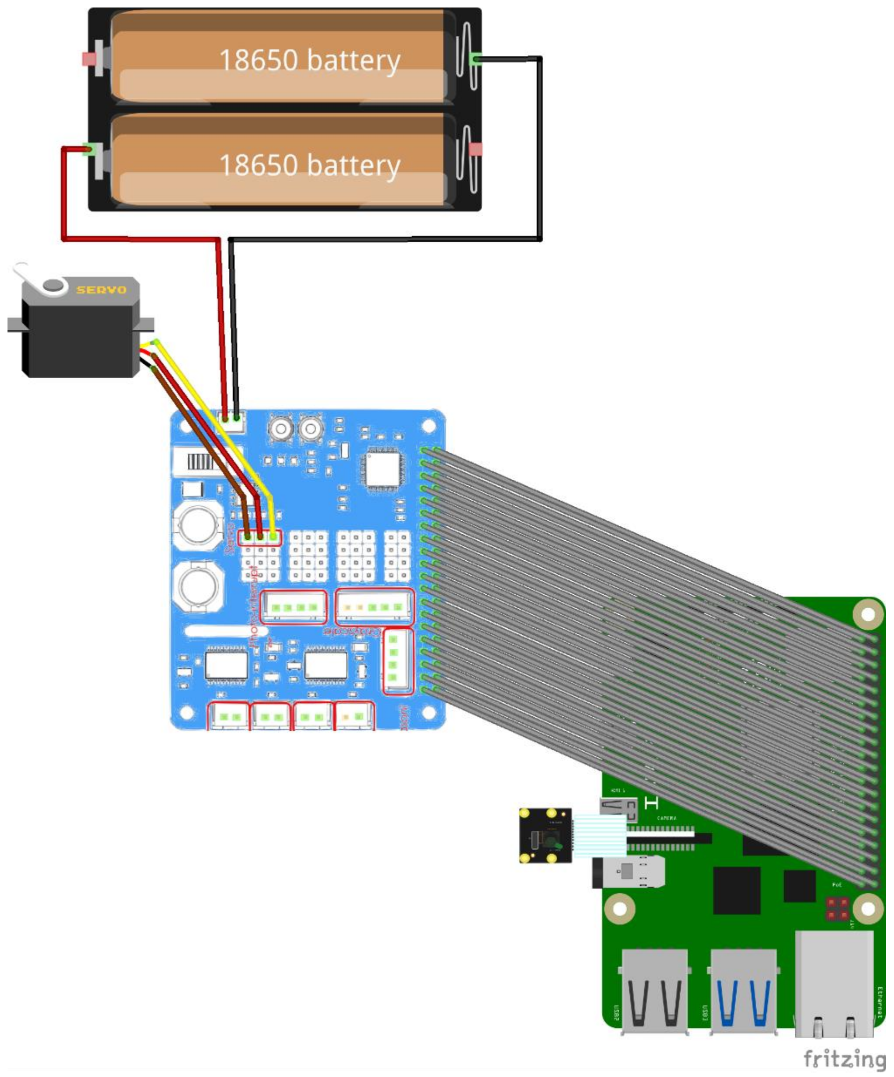


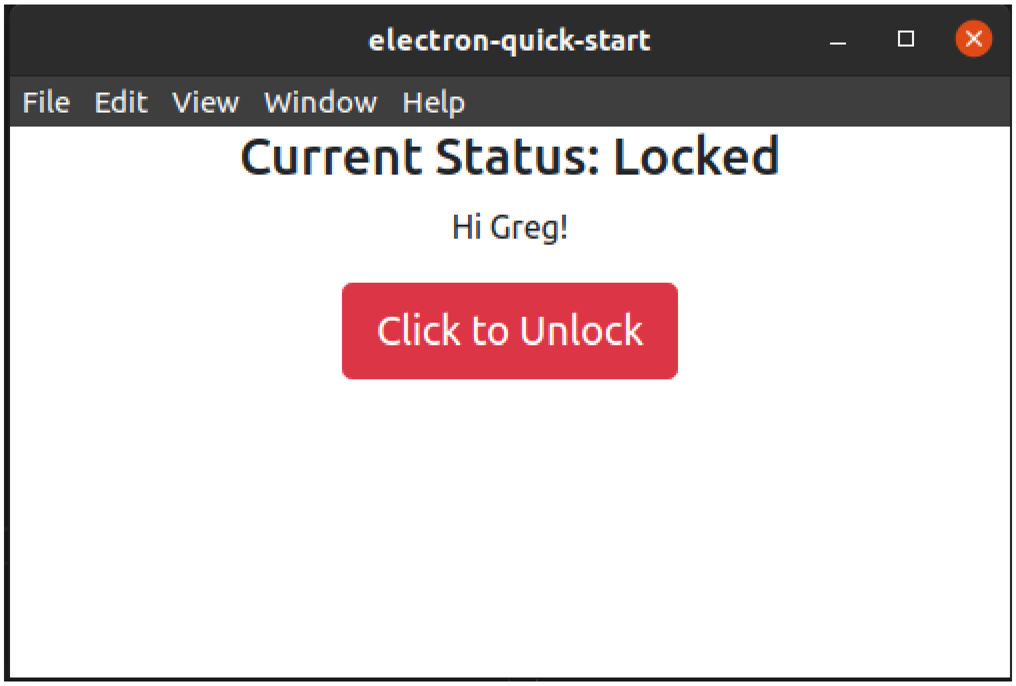
Figure 1. Diagram showing the configuration for components used for project.

Implementation Details:

The face_recognition Python library [2] was utilized to implement facial recognition. As part of the implementation, a directory was created containing images of authorized users. For each image in the folder, the face_recognition package was used to isolate the face of the authorized user in the image and create an encoded representation of that face.

A Raspberry Pi camera V2 is used to capture video. Each frame of the video is recorded in JPEG format using an aspect ratio of 640-pixels by 480-pixels. For each frame, the face_recognition package is used to isolate any faces in the frame and create an encoded representation of those faces. These encodings are then compared to each of the authorized user face encodings and scored by how similar they are, with a lower value meaning that they are more similar. Through experimentation, a similarity score of 0.6 was found to be a good threshold for determining if the faces are a match; therefore, if the similarity score between an authorized user's facial encoding and the encoding of a face from the frame of the video stream is found to be less than 0.6, those faces are considered the same.

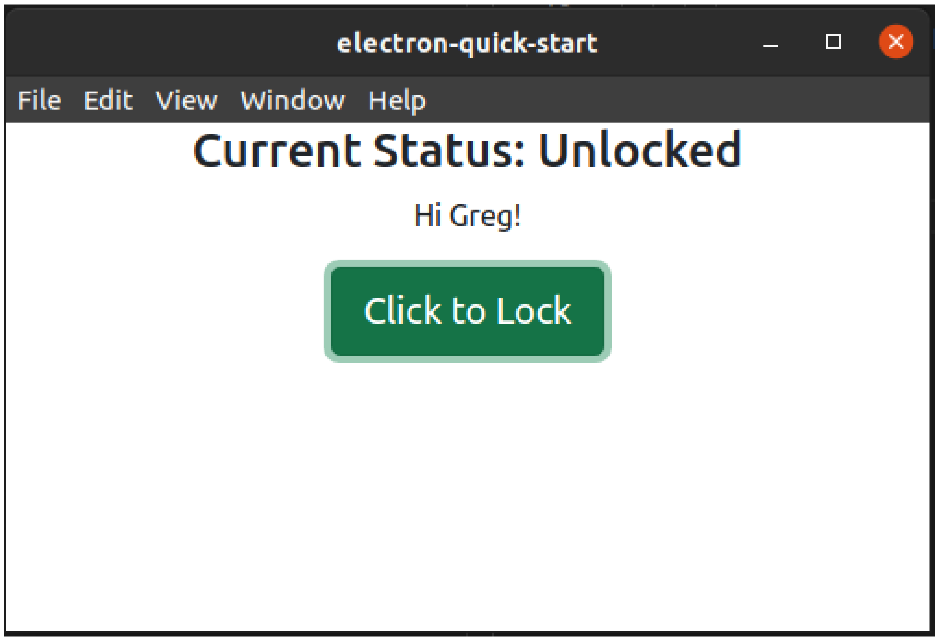
If a match is found, the name of that individual whose face is recognized is sent to the client, i.e., the computer running the Electron app. A greeting to that user is then displayed, as shown in Figure 2. The button, which has up until this time been disabled is now enabled and the user is prompted to click the button to unlock the lock. When the user clicks the button, the status of the lock is switched from locked to unlocked, as is shown in Figure 3. This status is sent back to the server (Raspeberry Pi) and the Raspberry Pi rotates the servo 90 degrees, signifying that the lock is unlocked. Additionally, when the user unlocks the lock, their name and the time in which they unlocked the lock are saved to a text file on the server (Raspberry Pi).



Current Status: Locked

Hi Greg!

Figure 2. Electron app showing lock in locked state but able to be unlocked because the authorized user, Greg, has been recognized by the camera.

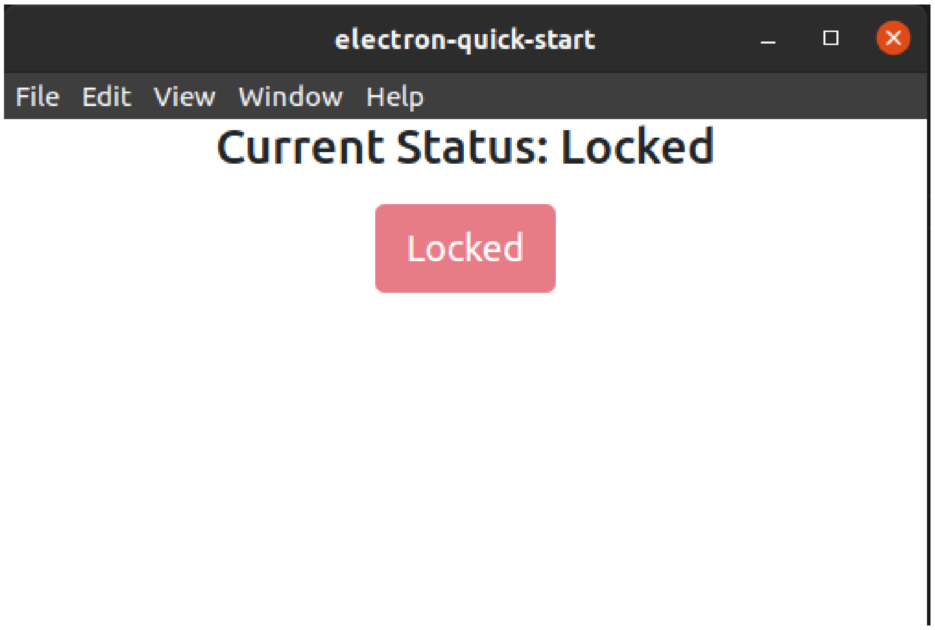


Current Status: Unlocked

Hi Greg!

Figure 3. Electron app showing the lock in the unlocked state. This happened after the authorized user, Greg, clicked the button in Figure 1.

An assumption is made that the user will only be temporarily unlocking the lock; therefore, no authorization is required to lock the lock after it has been unlocked. Whenever the user decides to lock the lock, they can click the button again. At this point the client sends a signal to the server to rotate the servo back 90-degrees to the locked position, and the button is disabled until an authorized user's face is recognized again. This can be seen in Figure 4.



Current Status: Locked

Figure 4. Electron app showing lock in locked status.

Results/Conclusion:
This project successfully implemented a version of this lock which will recognize a face and if that face is the face of someone that is in a list of authorized users, they will be allowed to lock and unlock the lock. This was done using a servo motor to demonstrate that this can be performed. If this were to be implemented using an actual lock, a product like the Adafruit 12VDC solenoid lock (https://www.adafruit.com/product/1512) could be used.

An initial issue that I encountered was that when I gave commands in the Electron App, it was taking a significant amount of time for the Raspberry Pi to respond. My solution for this was to change the speed at which I was sending data about what the status of the lock should be to the Raspberry Pi.

The way that the program is implemented is that the Raspberry Pi processes a frame, and then reads data from the client. The data from the client tells the Raspberry Pi if it should lock or unlock the lock. It turns out the Raspberry Pi was only able to process a frame every 1300 – 1700-milliseconds. If the client were sending data at a faster rate than this, a queue of unread data would build up. This would mean that the command from the client to lock or unlock the lock would be deeper in the queue and need to wait until all the data ahead of it was read, which could be many frames later. Sending data to the server at a slower rate than the server can process frames, causes the processing loop in the server to sync to the client, because the server will pause execution after it processes a frame until it receives the data from the client. It turned out that setting this to 1800-milliseconds worked out well. This amount of time seems reasonable since a fast frame rate is not required for an application like this.

The biggest downside to this project implementation is that the facial recognition can be tricked using a photo of an authorized user. This is a problem that many facial image recognition systems on older (and some current) generations of smartphones have. One way this problem was solved by Samsung was to perform a facial matching, like was performed in this project, along with iris matching. The iris matching was done by scanning a user's iris with infrared. The phone will unlock if both the face and iris match those of the authorized user [3]. Apple solved the problem on their iPhones by storing an infrared image and a depth map of the user's face, which is created by projecting 30,000 infrared points on the user's face. These scans are then performed on anyone trying to access the phone and compared to the scan of the authorized user [4]. It would be great to expand the project to implement one of these technologies, however if this became a product, it would likely be more convenient to create an application for a device, like an Apple or Android phone, that has the necessary hardware built into it. The user could then confirm their identity using the more advanced facial recognition technology already built into their smartphone.

References:
[1] https://www.electronjs.org/

[2] https://github.com/ageitgey/face_recognition

[3] https://zbigatron.com/apples-and-samsungs-face-unlocking-technologies/

[4] https://support.apple.com/en-us/HT208108#:~:text=The%20TrueDepth%20camera%20captures%20accurate,infrared%20image%20of%20your%20face.&text=Face%20ID%20automatically%20adapts%20to,makeup%20or%20growing%20facial%20hair