Software Overview

Year: 2019 **Semester:** Spring **Team:** 2 **Project:** Guard DAWG System
Creation Date: January 21, 2019 **Last Modified:** January 31, 2019
Author: Ian Sibley **Email:** isibley@purdue.edu

1.0 Software Overview

For this product, there will be two main sections of software: the facial recognition system, and the system that controls the door lock based on inputs. The control system will be housed on the microcontroller, which interfaces with a keypad manual override and a bluetooth sensor to take in the respective "open" signals upon user verification and results in the actuation of the lock motor to allow verified users to enter. The facial recognition system is housed on a Raspberry Pi and is more complex, reading frames from a camera, sending them over wifi to a server for comparison against registered users, accepting the resulting confidence ratings, and sending a pass/fail signal over Bluetooth once the result is found.

The microcontroller will run its software as a state machine (Figure 4), with the main body consisting of the two activation processes, and a high/low power switch that's on the inside of the door, so users can decide when they want the system guarding their door. One of the activations, the keypad verification, will be run through a second state machine (Figure 3), allowing a series of key inputs to raise a positive "correct_passcode" flag if no mistakes were made in the preset password, and delay further input if too many false attempts are made at once.

This keypad verification flag will run the main state machine straight to the door actuation stage from any state except the low-power (or "off") mode. The main high-level stages will consist of an idle state, an off state, a "running" state, and an activation state. The idle state will wait for a button press and, when pressed, signal the Raspberry Pi to initiate the face recognition process. It will then wait in the "running" state until it gets a signal. If a positive signal is input to the system (keypad or otherwise for this state), it will go to the activation state to unlock the door. Otherwise, if the facial recognition system returns a false flag, the system will return to the idle state.

The activation state will unlock the door for several seconds. Afterward, it will look to lock the door; this will not happen unless a "closed" flag is positive, to allow for bolt-lock use. However, since the current design will use a push bar that can close if locked still, this variable will be hardcoded to a positive "closed" state but left in for flexibility in the design. Once the activation state successfully completes, it will return to the idle state, and wait for either a bypass, recognition activation signal, or for the low-power signal.

From the low-power state, no signals will change the state except for the power-on signal given from the switch inside the door again. As the low-power signal is activated, the current plan is to unlock the door in transition to that state, though some discussion should be had over whether this should be left to the user or not, as some might want to turn off a smart system such as this at

night to ensure no vagrant of devious intent has an opening to try and bypass the system while the users are sleeping.

The facial recognition process (Figure 1) on the Raspberry Pi is planned to be coded in Python, for its flexibility and the speed in the libraries numpy and OpenCV especially. Once the signal is passed to run the process, the camera will be run to retrieve frames from a video feed. There will be a need for cropping the face out of the image, and ideally this will occur on the Pi itself (for faster WiFi transmission, mentioned later), but there will need to be a verification that this doesn't slow down the overall process too much, as the overall goal is to minimize the time the user waits but maintain accuracy.

The largest face found in the image will be cropped, either way, either right before or right after a WiFi transmission to a server. If no face can be found and cropped, the result will become an immediate fail flag, and if this is discovered on the server's side will result in the response of the WiFi transmission to reflect so. If on the Pi, it will act as if it got a fail flag from the server (in record time), and proceed as specified below.

Once the faces discovered on the server are all run through FaceNet, and their resulting feature vectors compared to all the known users (Euclidean distance, with a threshold currently set for ~0.50, though empirical testing will be needed), the total amount of passed and failed comparisons will be averaged together for a confidence value for each person. These confidence values will be sent back to the Raspberry Pi and turned into an overall passed or failed status. If no faces are above the specified threshold or multiple people are above the threshold, the result is going to be a failed signal, as significant further testing and refinement will be needed to account for the cases of strong familial resemblance and twins, and should be worked on in further iterations of this product. If only one known user is above the threshold, the result will be a passed flag. Either way, this flag will be passed to the microcontroller over Bluetooth, and the Raspberry Pi will return to waiting for activation again.

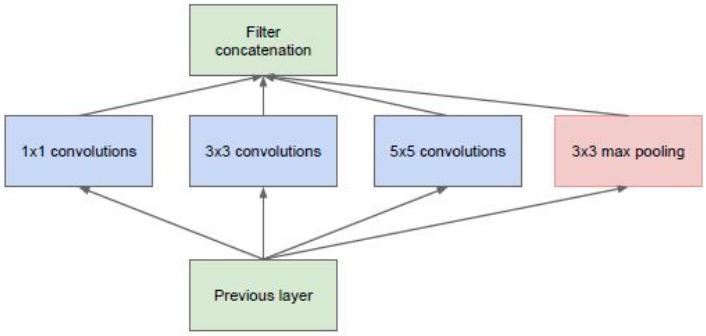
It should be noted that this second threshold is currently planned for a 75% match rate or higher, but the end threshold will be refined empirically for a more accurate boundary.

2.0 Description of Algorithms

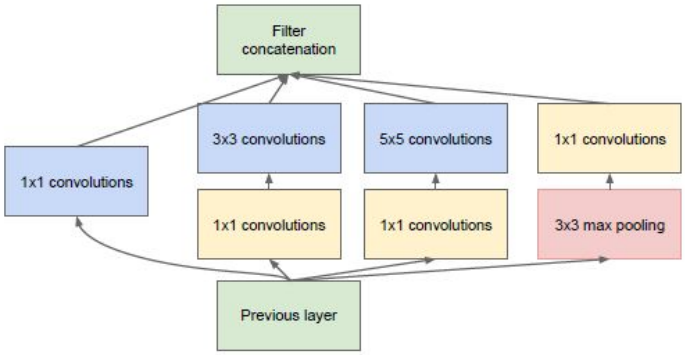
The main algorithms that are present in our product are the state machines, whose diagrams will be included below, and the FaceNet facial recognition algorithm, pre-trained to characterize faces and "describe" them with a resulting vector of 128 floating point ratings that can be linearly compared to other face results for similarity ratings via Euclidean distance.

The microcontroller will be running software that effectively implements a state machine (Figure 2) describe above in the software overview and depicted in Appendix 2. Initially the microcontroller is first turned on and must establish a bluetooth connection with the Raspberry Pi. From there, the inputs from various sensors like the keypad, bluetooth, and button presses dictate transitions into different verification states. All of the intensive computation with respect to the facial recognition software will be taken care of on the Raspberry Pi (Figure 1).

The FaceNet architecture itself is made up of many layers of convolution, pooling, and normalization, though a structure called an Inception Layer [1, 2]. This layer is a combination of different sized convolutions and max pooling all occurring in parallel and returning to one final point, so that the network can decide during training whether a dimension (1x1, 3x3, etc., depends on design) of convolution or a pooling process is the most relevant for that step, or a combination of multiple [4].



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

The documentation from the GitHub [3] isn't terribly clear what the structure used in the python module is, but it is design seems to be based on the more accurate of the Inception Layer designs, and runs a pre-trained model at 99.65% accuracy (+/- 0.252%) on the Labeled Faces in the Wild (LFW) dataset.

Retraining seems to be necessary for new faces added to the dataset, but this can be done with the pretrained model and its accuracy from more extensive training kept, retaining the extensive and expensive training processes already done for the model.

Alternatively, the face_recognition library in Python [6] is another 128 feature vector library, featuring ease of use and the use of OpenCV and DLIB's state-of-the-art facial recognition library. Due to the sheer documentation, and the similarity of the 128 feature vectors' linear comparison, face_recognition may become the network used for testing and staging purposes, in addition to being more immediately friendly to being merged into an application's internal code (seen in the relevant citation below). This currently compares to the documentation for FaceNet, of which, cited here, are documented in their self-made wike using only the console.

3.0 Description of Data Structures

The packet structure will use JavaScript Object Notation (JSON) for our WiFi transmission for the confidence rating responses from the server running the FaceNet algorithm, though the images themselves will be streamed up as files, ideally after cropping out the largest face in the given frame. This should be a reasonable achievement for the Pi, though speed is the main concern here, and investigations will be apart of our prototyping to see if the Raspberry Pi can

achieve reasonable speeds cropping the faces before WiFi transmission, versus cropping on the server with a wifi transmission of the full image.

Bluetooth supports serial communication [5], so messages between the microchip and the Pi will likely be short strings of initiation, likely words like "start", "low", "pass", and "fail", though they can be simplified down to single letters (s, l, p, f, etc.), if simplicity is desired on the microcontroller's state machine design, and whether there will need to be sub-states to pull in each character sent over serial, or if they can be pulled all at once.

4.0 Sources Cited:

[1] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[2] I. Aranguren and R. Rana, "FaceNet." .

[3] D. Sandberg, "davidsandberg/facenet," *GitHub*. [Online]. Available: https://github.com/davidsandberg/facenet/wiki/Variational-autoencoder#2-calculate-attribute-vectors. [Accessed: 25-Jan-2019].

[4] V. L. V. Ladenkov, "What is an inception layer?," *Data Science Stack Exchange*. [Online]. Available: https://datascience.stackexchange.com/questions/14984/what-is-an-inception-layer. [Accessed: 25-Jan-2019].

[5] "Raspberry Pi Resources," *Raspberry Pi Projects*. [Online]. Available: https://raspberry-projects.com/pi/pi-operating-systems/raspbian/bluetooth/serial-over-bluetooth. [Accessed: 25-Jan-2019].

[6] Ageitgey, "ageitgey/face_recognition," *GitHub*, 13-Nov-2018. [Online]. Available: https://github.com/ageitgey/face_recognition#python-module. [Accessed: 25-Jan-2019].

Appendix 1: Program Flowcharts

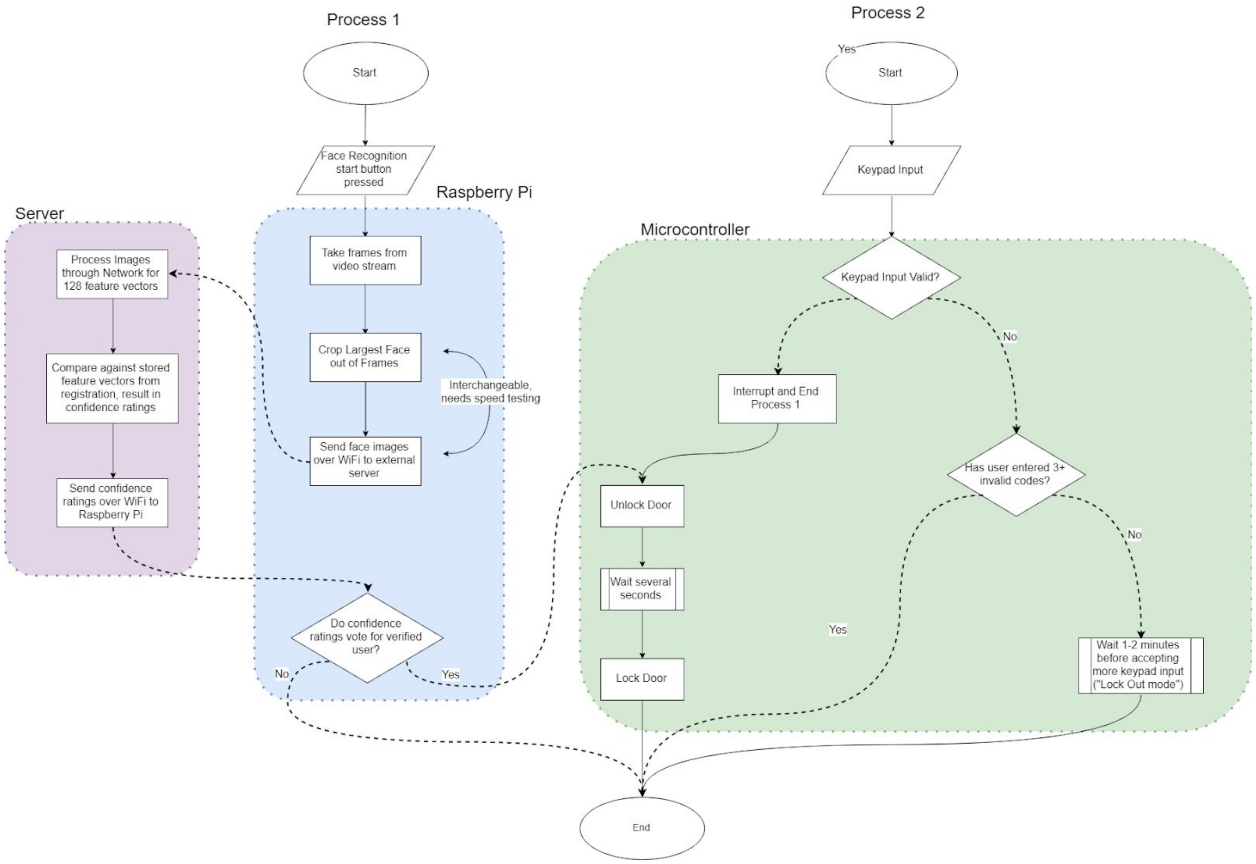


Figure 1: System Flowchart

Appendix 2: State Machine Diagrams

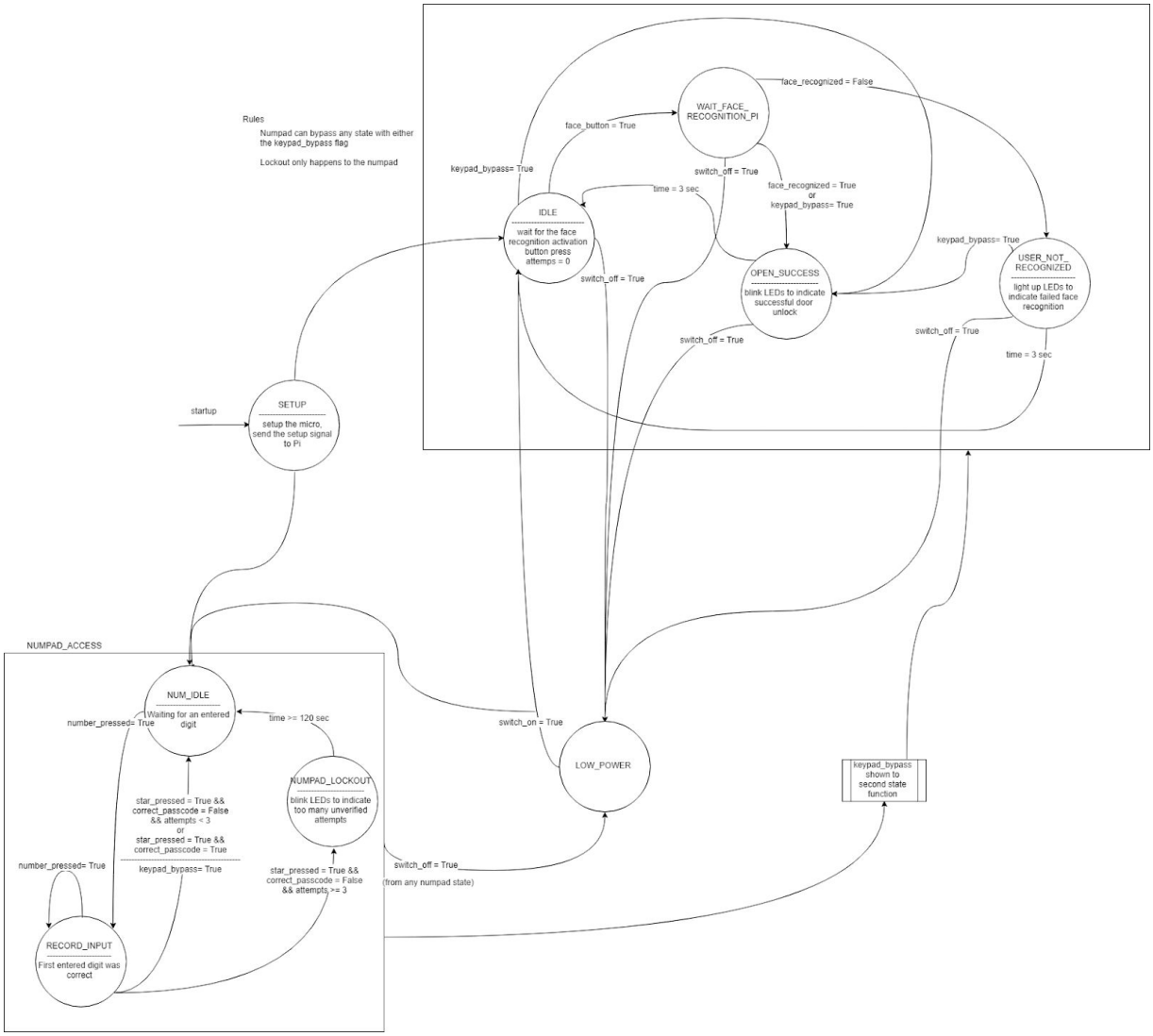


Figure 2: System State Machine

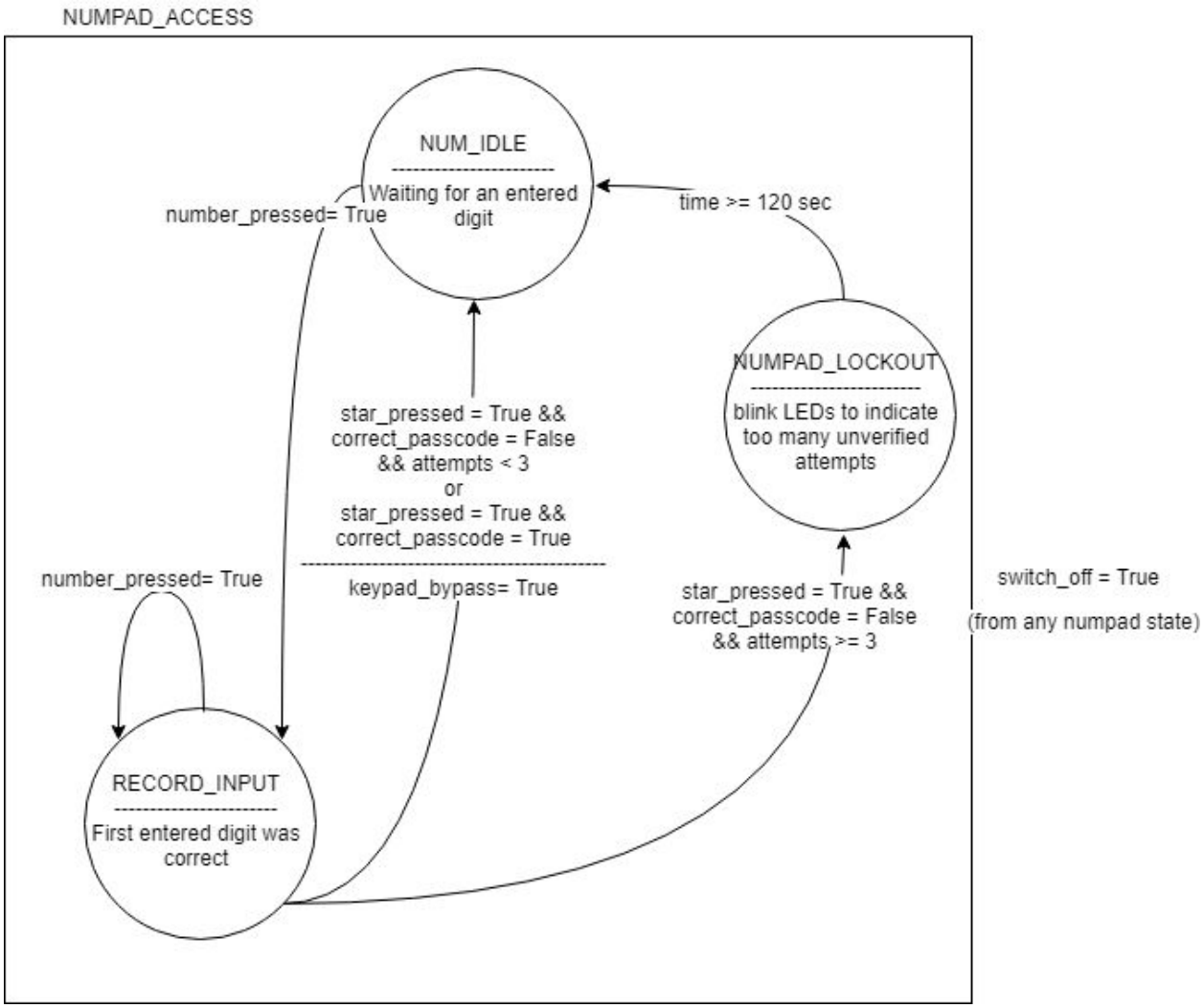


Figure 3: Numpad State Machine

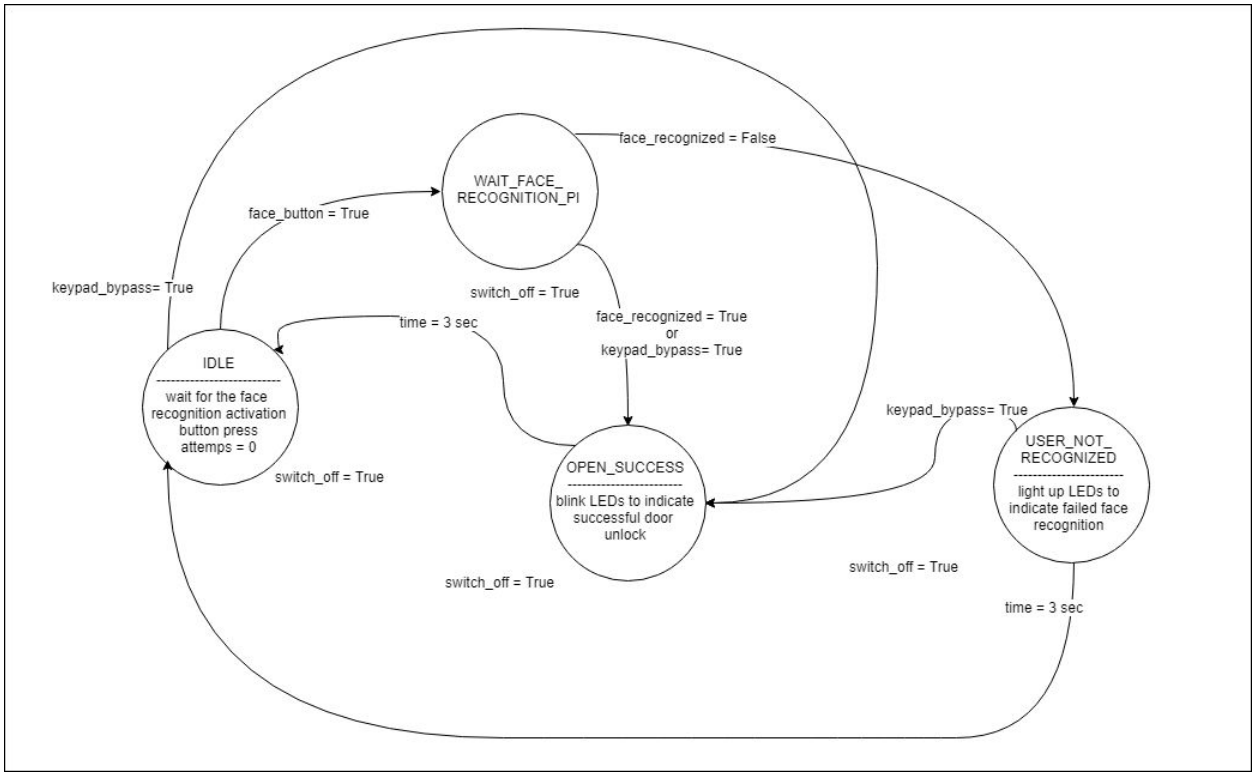


Figure 4: Facial Recognition State Machine