Project 19: eyeCU

Project Advisor: Carlos Rojas

Computer Engineering Department

Hoang, John (BS Software Engineering)
Leung, Justin (BS Software Engineering)
Nguyen, Thi Tuong Vi (BS Software Engineering)
Bahramiandehkordi, Arash (BS Software Engineering)

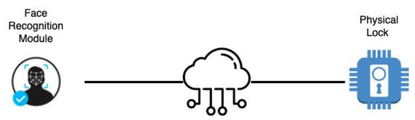
Introduction

The main problem with current cloud-based camera security systems, especially the ones with wireless cameras, is the lack of options for users to react to video feed. The Ring doorbell can take more than thirty seconds to notify the registered users of occurred events and at most gives users the option to communicate through the speakers. 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 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.

Methodology

Our approach for the underlying problem centers around two separate components: face recognition module and the physical programmable lock. In the production setting, the machine learning module would be deployed on the cloud while the controller unit is idly packed into a compact printed circuit board (PCB). However, for demonstration purposes, our prototype implements the controlling logics on a Raspberry Pi model 3B+ interacting with a development server that hosts the face recognition unit.

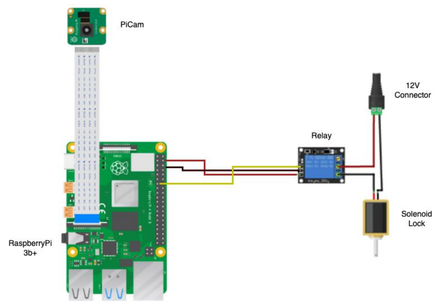


Face Recognition Module

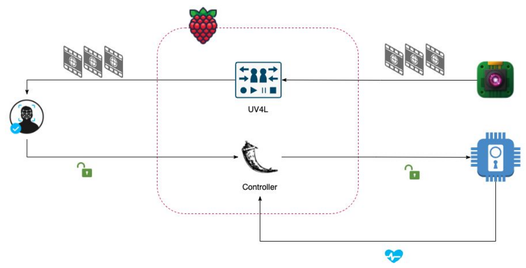
Built on top of a Flask server, the face recognition module takes care of authorizing users based on the live video feed received from the Raspberry Pi. When the raw data is sent to the server, the image processor breaks them down into usable feature vectors, which are input into the face detector to extract faces in the frames. Then, the face classifier processes the acquired faces deciding if the user is authorized for entry or not. The result can be conveniently monitored via our React dashboard.

Lock Controller

The lock package consists of a Raspberry Pi (model 3b+), a PiCam, and a power source. The PiCam is connected to the Pi via the standard camera port. Because the unit doesn't have enough power to operate the lock by itself, an external power source is used while the main signal propagates through a relay via pin 11 on the Pi.



Internally, a lightweight UV4L streaming server periodically transmits the live feed from the PiCam to the subscribed clients on demand. Even though the camera module can support up to 1080p, the streaming server is configured to limit to 1 frame per second to reduce the power consumption as well as network traffic. The Pi also houses a flask server that constantly listens to any incoming request from the recognition module and moves the lock accordingly. The status of the lock, including states, battery percentage, signal strength, etc., would be regularly reported back to the backend server for health checking purposes.

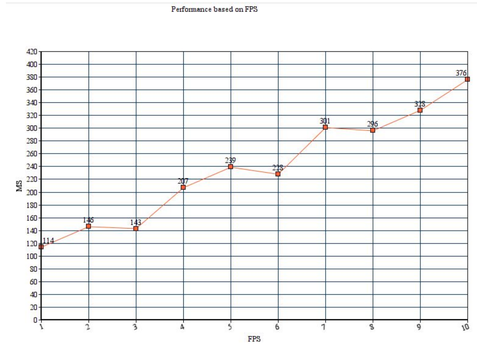


Since the heavy lifting tasks are performed on the cloud, the Pi consumes significantly less power, but it heavily depends on network connectivity. This prototype is strictly built for functionality, so there is no security measure implemented. Hence, the actual performance in a production setup might be different from record in the development phase.

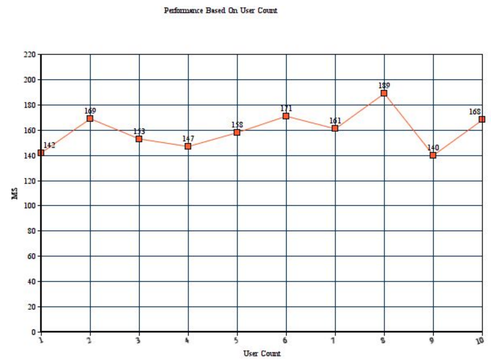
Analysis and Results

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, 8gb RAM, and integrated Intel graphics.

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 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.

Actual

Predict		Negative	Positive
	Negative	27	8
	Positive	2	23

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))

Summary/Conclusions

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.

This project left the team with a lot of exposure and experience to 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.

Key References

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

[2] https://flask-socketio.readthedocs.io/en/latest/

[3] https://flask.palletsprojects.com/en/2.1.x/

[4] https://www.linux-projects.org/documentation/uv4l-raspicam/

Acknowledgements

The team would like thank Professor Carlos Rojas for being our team advisor and offering his time for bi weekly meetings, paper revisions, insightful advices, and motivation.

Shoutout to all the team members who grinded this wonderful project.

And shoutout to Stack Overflow for helping us debug our code