Facial Recognition Rack Mount System

CPE 496 Final Report

Jared Nixon – Team Lead

Garrett Eledui – Software Integration

Jason Parker – Hardware Integration

Daniel Hasty – Software Support

# Sponsored by

Thom Rigsby of

# Emerson Network & Power

### Spring 2017

**Project Summary**

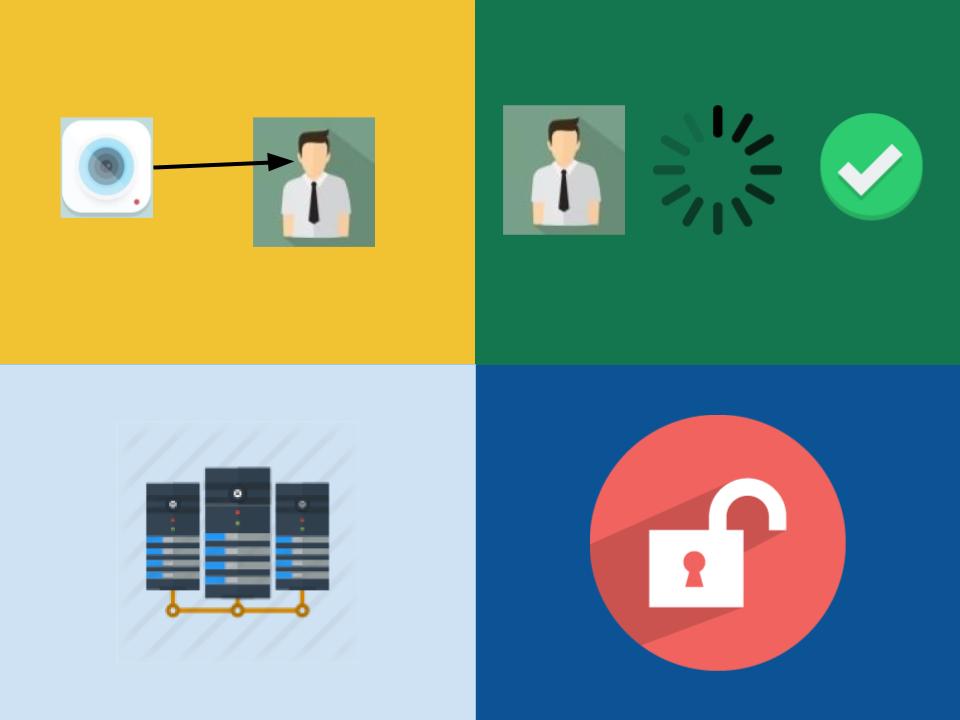
The Facial Recognition Rack Mount project is a system-level hardware/software design with a primary intention of keeping server rooms safe and secure. With physical security of modern datacenters being often overlooked, the goal of this project is to provide the means of increasing that security.

**Project Photo:**



**Concept Overview:**

* Take a photograph
* Verify If the image contains a face
* If yes, let the lock change state to unlocked.
* If no, let the lock remain at its state of locked.



**Team Description**

Jared Nixon – **Team Lead**

Responsibilities: Ensure deadlines are made, keep the team moving in the right direction, and support software development.

Garrett Eledui – Software Lead

Responsibilities: Develop / Integrate facial recognition software.

Jason Parker – Hardware Lead

Responsibilities: Develop communication between system components.

Daniel Hasty – Software Support

Responsibilities: Develop software where help is needed to maintain deadlines.

**Introduction**

Facial Recognition Rack Mount – Sponsored by **Emerson Network Power**

**Marketing Requirements:**

* The system should have a camera, processing unit, and means of turning a lock on a server rack.
* The system should be easy to use with minimal experience
* The system should have the ability for the user to request entry and for the user to know whether they were authenticated or not.

**Engineering Requirements:**

* Performance – The system should unlock the rack door in no more than ten seconds from the point when access is requested.
* Functionality – The system will process the photo and relay an access granted or access denied signal to the actuator.
* Functionality – The system will auto lock after 60 seconds regardless of if the user is finished with the rack. Upon finishing with the rack, the user may have to press the button again to unlock the rack (if the timeout period has already passed), then close the door.
* Usability – Users of the system should be able to unlock with the push of a button and have user feedback.

**Background**

**Competition:**

* AMG Product: contains facial and fingerprint recognition, and has a database.
  + Pros: Product is pre built with very impressive features.
  + Cons: Product costs $499.00

**Projects:**

* Windows Team IoT: Facial Recognition Door.
  + Team development project that uses facial recognition to unlock a door.
  + This is accomplished using Microsoft API: Project Oxford.
  + API seems to be open to use.

**Tradeoff Analysis of Design Alternatives**

*Rasp Pi vs Microcontroller* – The Raspberry Pi 3 is the processing unit of choice because it already has facial recognition software (OpenCV) readily available for use with it. It also has more processing power with a clock speed of 1.2 GHz and 1 GB of RAM.

*Camera Module vs. Standard Camera* – The Raspberry Pi Camera Module is the camera of choice because it plugs directly into the Raspberry Pi and is designed for specific use with it. It is capable of capturing static images at 3280x2464, which is more than enough for the camera need of this project, and the camera software is supported with the latest version of the Raspbian Operating System.

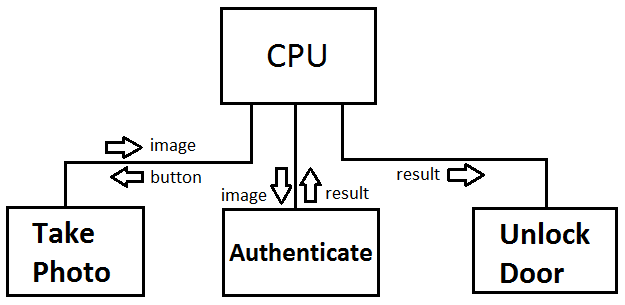
*OpenCV vs Windows API* – OpenCV is the approach decided on for the facial recognition software. Though the Windows 10 API has already been proven to work, OpenCV is already directly compatible with the Raspberry Pi 3.

*Push Button vs Automatic Photograph* – Push Button is the more viable approach here because it allows the user to notify the system when they are ready to have a photo taken. Using automatic photographing can be very CPU inefficient depending on the polling time, whereas a push button will allow one photograph, and involve significantly less code.

*C++ vs Python* – Because of the integration of the OpenCV library, the team decided to go with Python as the programming language of choice. This makes the facial recognition software much easier, as well as interfacing with the raspberry pi’s GPIO ports, which contribute to an overall better solution.

*Push vs Pull Actuator* – For the sake of power consumption, an actuator that stays outward and retracts when current is passed through would be the most efficient actuator to use. This will ensure that power does not need to be applied 99% of the time.

**Functional Decomposition:**



**Facial Recognition: OpenCV Haar Cascade**

Facial recognition really just comes down to detecting combinations of pixels in an image that are aligned in a particular pattern. If we remove color from an image, we are left with white pixels, black pixels, and then variations of gray pixels that have different levels of light and dark. The software looks for sections in the image that have both dark pixels and light pixels. This allows us to detect edges. There are several main types of edges that the software looks for. Edge features and Line features are the ones we are most interested int. By simple detecting edges in a black and white image, we can tell the software what patterns of edges a face should look like. e.g. if we have a darker edge with white in the middle is very commonly a nose due to the brighter lighting on it. Whereas eyes can be found by finding a brighter area of pixels above a darker area of pixels, and similarly with a mouth. If we find a horizontal brighter over dark area that is right above a vertical brighter area with two darks on either side, we know that these two main areas are most likely the making of a set of eyes and nose.

The software we used is open source software called OpenCV and the particular library that we are used is called Haar Cascade. This software uses a machine learning algorithm. This means that you can teach the software to look for certain patterns of pixels in an image, even though it is pretrained to identify faces. You can give the software two sets of images; good images which contain the object that you are training the software to detect, in our case a face, and bad images which don’t contain a face. You can tell the software (if you train it) that you only want it to look for commonalities in the good images, which should then find the face pattern over and over again, then you tell it that these other images you want to ignore. We could use this concept to detect any kind of object in an image as long as it had a somewhat consistent form, as it is not restricted to just faces. With the machine learning behind talked about above, the system designed in this project does not use machine learning, as our system is already efficient at finding faces and does not need to get significantly better (though it could).

**Global/Societal Impact of Project**

**Impact of the datacenter environment:**

* The project will have an impact on data centers from a space management standpoint. Most server racks are filled to the rim to get the most efficiency out of them, meaning there is not much room to mount a raspberry pi. Our design will be placed in what is called the ZERO U space of the rack (i.e, the left or right side where space is not allocated for servers / power management units), so server space will not have to be sacrificed.

**Impact of modern securities:**

* The goal of this project is to increase security on rack mounts. Bearing in mind this is a basic prototype, there still will be vulnerabilities such as the door behind the rack not being locked, authorized / non-authorized faces not being distinguished, and brute force break-ins. This project is meant to begin the R&D process of an eventual secure system.

**Test Plan**

*Unit Testing*:

* Can the button be pushed and acknowledged?
* Can a picture be taken and sent to the proper facial function?
* Can the software verify that a picture was taken?
* Can the facial function verify a face is present?
* Can the CPU send voltage command to actuators?
* Can the CPU send voltage command to LED?

*Integration Testing*:

* Can the button be pushed and trigger a picture to be taken?
* Can the software receive the photo and verify a face is present / not present, and send voltage out to appropriate peripheral?
* Can the actuators / LED recieve voltage and take appropriate action?

*Regression Testing*:

* No specific tests can be listed for regression testing because the team does not know problems that will happen during the development process. The point is to ensure fixing a problem does not introduce other problems.

*Acceptance Testing*:

* See marketing requirements and compare to actual functionality. The marketing requirements are the only guidelines for this testing process.

**Risks**

The main (and only) risk that arose while developing our project was that the solenoids initially purchased for use may not be strong enough to keep anyone out. As Jared said in presentations, they may not even keep a toddler our, as they are only the size of a quarter. After further testing of our risk throughout the semester, and some reinforcement from Home Depot, these solenoids turned out to be extremely strong and powerful. This was a pleasant risk assessment because that meant the team could power them from the 5 V power supply on the raspberry pi, and not have to consider external power as a solution.

**Work Breakdown Structure and Project Schedule**

*Milestone One* – Deadline of November 4th

* Research hardware/software integration and compatibility.
* Research facial recognition algorithms.
* Research actuators that are likely to be successful.
* **DELIVERABLE: Project Proposal Presentation (11/2)**

*Milestone Two* – Deadline of November 11th

* Submit hardware components needed to Thom based upon research findings.
* Decide on appropriate coding language (C++, Python, etc.)

*Milestone Three* – Deadline of February 10th

* Facial Recognition lock in the most basic form (will open for anyone).
* Ensure the lock will NOT open for objects, and strictly for faces of people.
* Validate the camera takes successful samples of person accessing rack.
* Validate current flows only one way to the actuator before full hookup (use a diode).
* **DELIVERABLE: Preliminary Design Review (2/23)**

*Milestone Four* – Deadline of April 14th

* Test the final prototype of the project.
* Verification / Validation processes

*Milestone Seven* – Complete Final Design

* **DELIVERABLE: Project Video (4/17)**
* **DELIVERABLE: Project Poster (4/20)**
* **DELIVERABLE: Project Demonstration (4/20)**
* **DELIVERABLE: Final Design Review (4/20)**
* **DELIVERABLE: Project Notebook (4/27)**
* **DELIVERABLE: Project Final Report (4/27)**

**Course Team-Specific Deliverables**

* Topical Seminar – February 10, 2017
* Preliminary Design Review – February 23, 2017
* Bulletin Board Update – Six Times Through Semester
* Project Briefings – March 30, 2017 & April 13, 2017
* Project Video – April 17, 2017
* Project Poster – April 20, 2017
* Project Demonstration – April 20, 2017
* Final Design Presentation – April 20, 2017
* Project Notebook – April 27, 2017
* Final Report – April 27, 2017

**Project Cost Evaluation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Quantity** | **Projected** | **Actual** |
| **Raspberry Pi** | 1 | $39.95 | $39.95 |
| **Camera Module** | 1 | $18.50 | $29.95 |
| **Case** | 1 | $5.00 | $8.00 |
| **Solenoids** | 2 | $18.00 | $9.90 |
| **RGB LED (25 pack)** | 1 | $10.00 | $12.50 |
| **Push Button** | 2 | $4.00 | $3.00 |
| **SD Card (w/ OS)** | 1 | $11.95 | $11.95 |
| **Transistors (3 pack)** | 1 | N/A | $2.50 |
| **Wires** | 1 | N/A | $10.00 |
| **Mock Door** | 1 | N/A | $40.00 |
| **Pi Power** | 1 | N/A | $9.00 |
|  |  |  |  |
| **Shipping Total:** | $18.08 |  |  |
|  |  |  |  |
| **Totals:** |  | $107.40 | $176.75 |

**Project Achievements:**

* Successful prototype of a single facial recognition rack mount system.
* Successful understanding of the emphasis of the project (actuator control).
* Successful integration of facial recognition software via Python.
* The team met all marking and engineering requirements given by the sponsor.

**Lessons Learned:**

* Recall the basics on complex problems.
  + Basics are a subset of complex engineering problems, such as V = IR.
* How to stay on top of deadlines?
  + Communication is key.
  + Don’t let deadlines sneak up, always be looking ahead.
* What makes a system open for expansion?
  + Starting simple and adding features later.
  + Don’t start too large, it can make expansion very difficult.

**Future Project Possibilities:**

Due to this project being a very successful R&D project at its core, the project is very open for future enhancements. With the base prototype done, future projects could include things such as detecting specific faces as authorized or not authorized, having multiple rack mounts communicating with each other, or having a large database that can log successful / unsuccessful attempts to access the system (or other useful information).

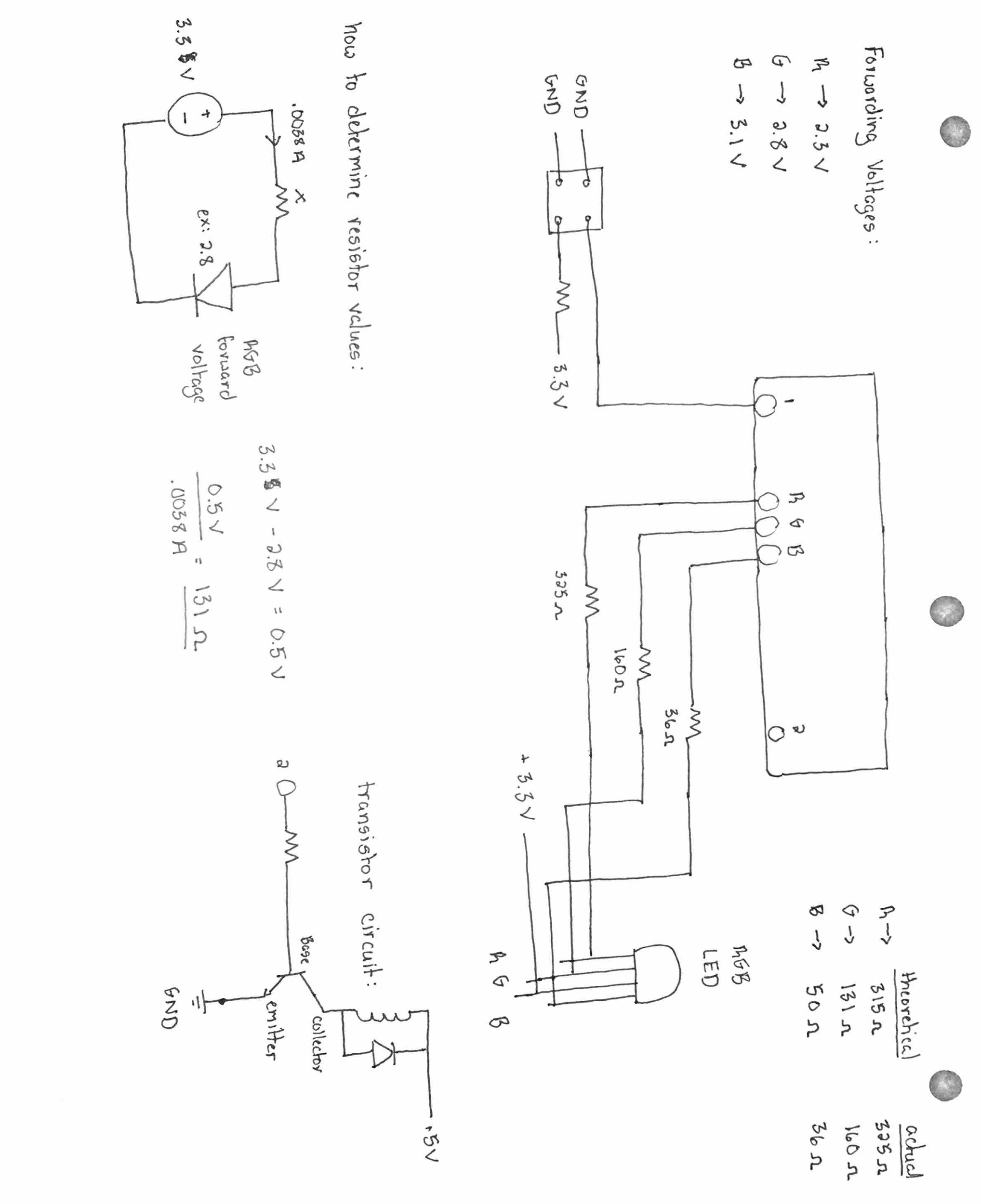
**Note About System Use:**

In order to use the system, power on the raspberry pi. When it is fully booted, the light on the door should turn blue. This is indicating that the system is ready for a photo. Push the button one time with a face verified, and upon the timer going off to relock the door, the solenoids will engage and make the system secure. In other words, a verified face needs to be ran through first in order to engage the secure system with the solenoids.

**Conclusion:**

In conclusion, Team Face Off successfully gained valuable experience of working in a team environment, working with an engineering customer, and defining marketing and engineering requirements with the customer. This two semester project was the closing to the University of Alabama in Huntsville Computer Engineering program and a very valuable simulation of a real world experience. The project was successful as the team met all engineering and marketing requirements, and learned very valuable skills in regards to team work, communication, and ethical work.

**Appendix A – Design Schematic:**



**Appendix B – Circuit Photo:**

****

**Appendix C – Source Code:**

from time import sleep

import picamera

import RPi.GPIO as GPIO

import cv2

import time

import signal

import sys

from threading import Timer

### LOCKS THE DOOR ###

def lockSolenoid():

GPIO.output(solenoidPIN1,GPIO.HIGH)

GPIO.output(solenoidPIN2, GPIO.HIGH)

### UNLOCKS THE DOOR ###

def unlockSolenoid():

GPIO.output(solenoidPIN1,GPIO.LOW)

GPIO.output(solenoidPIN2, GPIO.LOW)

### POWER OFF SOLENOID AFTER PROGRAM TERMINATION ###

def poweroff(signum, frame):

unlockSolenoid()

sys.exit(0)

### TURNS OFF AN LED PIN ###

def turnOff(pin):

GPIO.output(pin,GPIO.HIGH)

### DRIVES RED LED ###

def redON():

GPIO.output(redPIN,GPIO.LOW)

GPIO.output(greenPIN,GPIO.HIGH)

GPIO.output(bluePIN,GPIO.HIGH)

### DRIVES GREEN LED ###

def greenON():

GPIO.output(redPIN,GPIO.HIGH)

GPIO.output(greenPIN,GPIO.LOW)

GPIO.output(bluePIN,GPIO.HIGH)

### DRIVES BLUE LED ###

def blueON():

GPIO.output(redPIN,GPIO.HIGH)

GPIO.output(greenPIN,GPIO.HIGH)

GPIO.output(bluePIN,GPIO.LOW)

### FUNCTION TO LOCK THE DOOR BACK ###

def lockDoor():

print("Time Up. Locking door")

blueON()

lockSolenoid()

### FUNCTION TO RESET STATE TO READY AFTER DENIAL ###

def denied():

blueON()

### FACIAL RECOGNITION FUNCTION ###

def faceFind():

imagePath = 'employeePhoto.jpg'

cascPath = '/usr/share/opencv/haarcascades/haarcascade\_frontalface\_default.xml'

faceCascade = cv2.CascadeClassifier(cascPath)

image = cv2.imread(imagePath)

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

faces = faceCascade.detectMultiScale(

gray,

scaleFactor = 1.1,

minNeighbors = 4,

minSize = (160,160),

flags = cv2.cv.CV\_HAAR\_SCALE\_IMAGE

)

if (len(faces) == 1):

print("numer of faces is:", len(faces))

print('Access Granted!')

greenON()

unlockSolenoid()

t = Timer(10.0,lockDoor)

t.start()

elif (len(faces) > 1):

print("numer of faces is:", len(faces))

print('Access Denied, 1 face at a time, please')

redON()

t = Timer(6.0,denied)

t.start()

else:

print('Access Denied, no faces found')

redON()

t = Timer(6.0,denied)

t.start()

print "Found {0} faces!".format(len(faces))

for(x, y, w, h) in faces:

cv2.rectangle(image, (x,y), (x+w, y+h), (0, 255, 0), 2)

#cv2.imshow("Faces found", image)

#cv2.waitKey(10000)

#cv2.destroyAllWindows()

#cv2.waitKey(1)

return

### MAIN FUNCTION HERE ###

bluePIN = 5

greenPIN = 6

redPIN = 13

solenoidPIN1 = 26

solenoidPIN2 = 19

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM)

GPIO.setup(redPIN,GPIO.OUT)

GPIO.setup(greenPIN,GPIO.OUT)

GPIO.setup(bluePIN,GPIO.OUT)

GPIO.setup(solenoidPIN1,GPIO.OUT)

GPIO.setup(solenoidPIN2, GPIO.OUT)

GPIO.setup(18, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

turnOff(redPIN)

turnOff(greenPIN)

turnOff(bluePIN)

blueON()

unlockSolenoid()

camera = picamera.PiCamera()

print('Press the button for rack access.')

signal.signal((signal.SIGINT | signal.SIGTERM), poweroff)

#signal.pause()

while True:

input\_state = GPIO.input(18)

if input\_state == False:

print('Photo taken... Please Wait.')

camera.capture('employeePhoto.jpg')

# Call function to detect faces.

faceFind()

sleep(0.2)

print('Press the button for rack access.')