EEE 174 / CpE 185 Final Project Report

Security System

WholeRoom Security

Cole Croteau

[redacted]

[redacted]

Team members and responsibilities

- Cole Croteau:
  - Collective Responsibility: Data Logging, LED System, Buzzer System
  - Main Responsibility: Keypad System

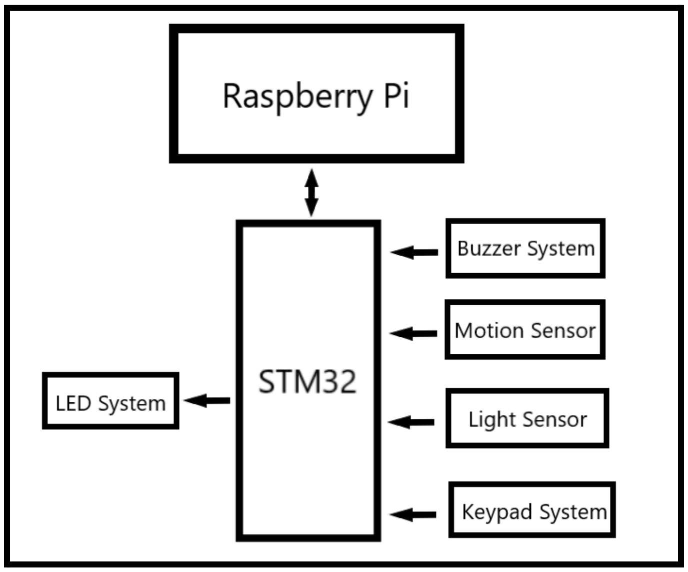
- [redacted]:
  - Collective Responsibility: Data Logging, LED System, Buzzer System
  - Main Responsibility: Motion Sensor

- [redacted]:
  - Collective Responsibility: Data Logging, LED System, Buzzer System
  - Main Responsibility: Light Sensor

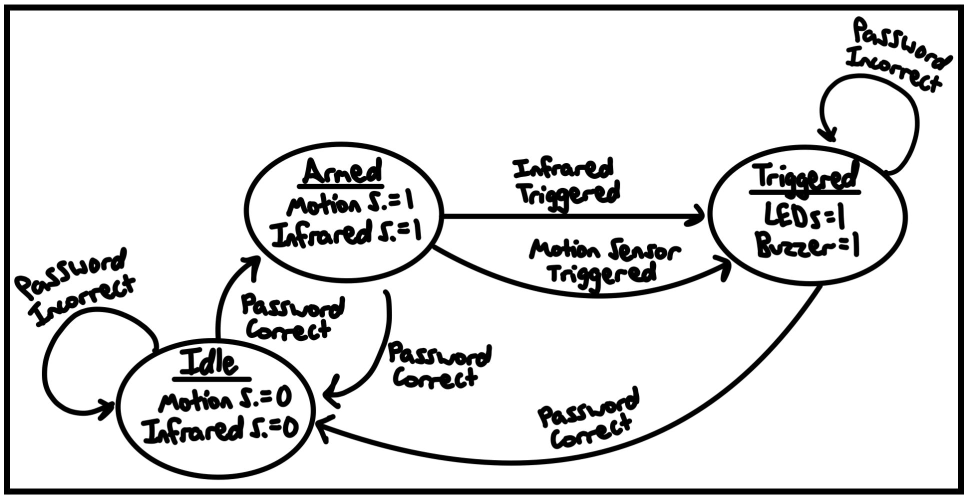
Project Description

This project aims to create a security system using the STM32 microcontroller. It will use a variety of components including LEDs, push buttons, a motion sensor, and an infrared sensor. It will have features like data logging (time alarm triggered, which sensor triggered the alarm, etc.), mode swapping (armed, disarmed, etc.), and a password system to disable the alarm after it has been triggered.

Block Diagram

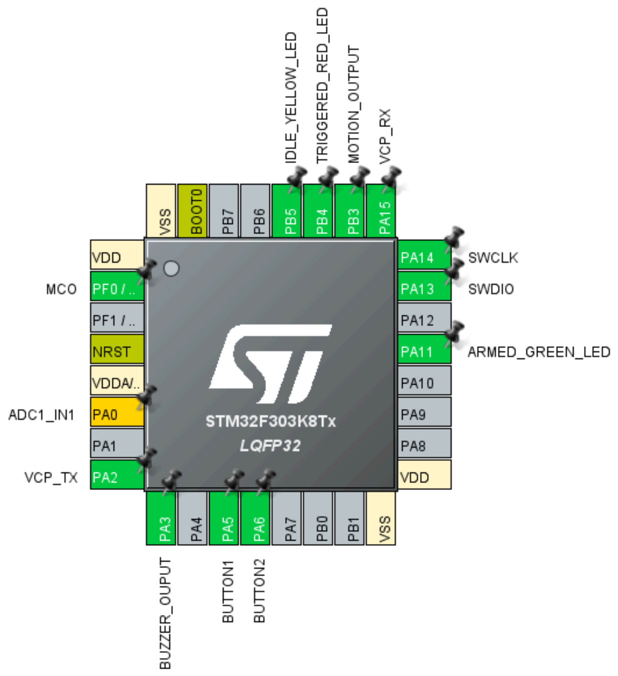


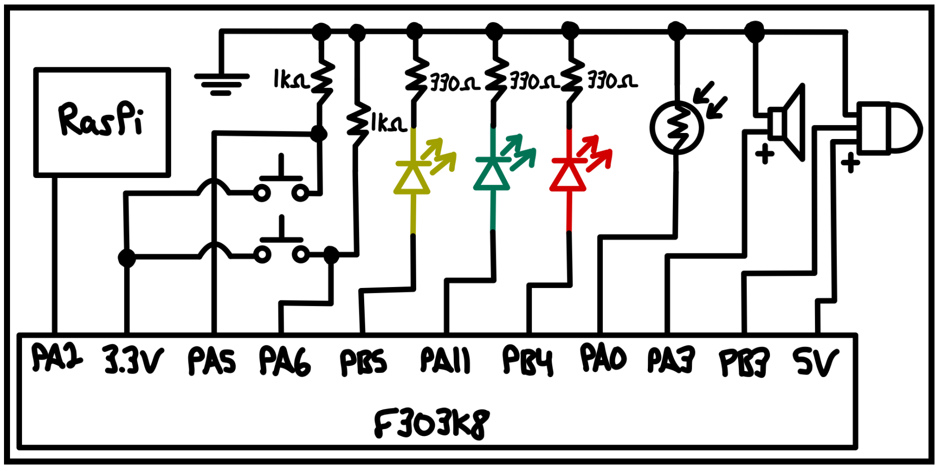
FSM Diagram



Background

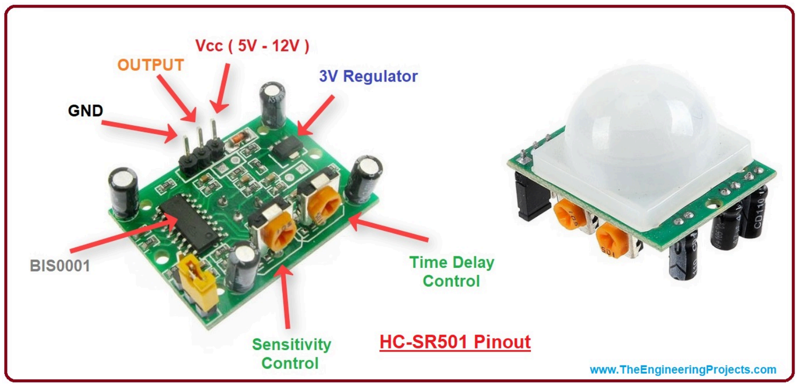
As stated previously, this project will use a combination of components including LEDs, push buttons, a motion sensor, a photoresistor light sensor, a buzzer, the STM32 Nucleo board, and a Raspberry Pi. To go into the specifics of each of the sensors and devices that will be incorporated with the STM32 board, the motion sensor will be used to detect any movement and if so, the security system will trigger. On a similar note, the light sensor will be used to detect any changes in light levels, like a flashlight shining over the system, in which the security system will also trigger. When in the Triggered State, the buzzer will be incorporated to buzz until the user disarms the security system. As for the Raspberry Pi, it is used to keep track of the status of the security system and data log whenever there is a change in the system. For instance, it will keep track of when the system was last armed, when it was triggered, and so on.

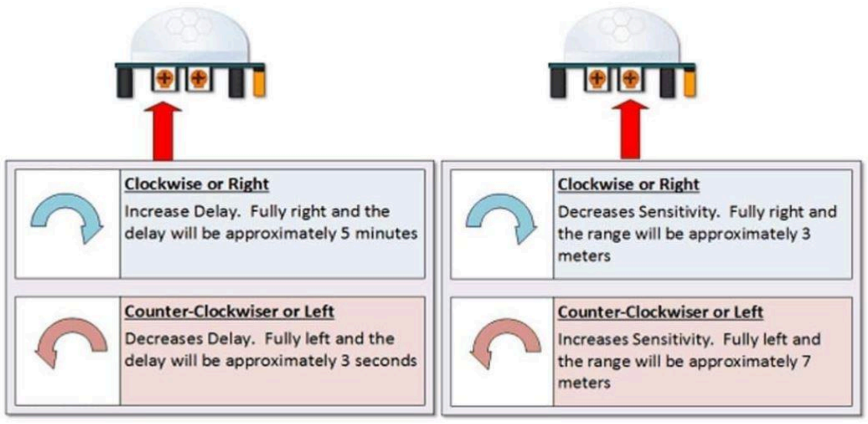




[Topic 1 - Motion Sensor]

The motion sensor, or specifically, the PIR Infrared Sensor, can detect movement through the use of a spherical dome, as shown in the image. This spherical, white dome acts as a sort of lens that focuses infrared light on the sensor within. With infrared light, the lens can detect temperature changes, which correspond to the heat emitted from our bodies. So, when a hand, for example, gets near the dome, the infrared sensor can read the heat from the hand, which, in turn, signals to the motion sensor to trigger. With this specific PIR Infrared Motion Sensor, three pinouts represent the ground, output, and Vcc. Two other settings on the board can either change how long it takes for the motion sensor to send a signal after it is triggered, and how sensitive the sensor is. The way the motion sensor will be connected is by using a wire to connect the output and ground to the STM32 board and breadboard.

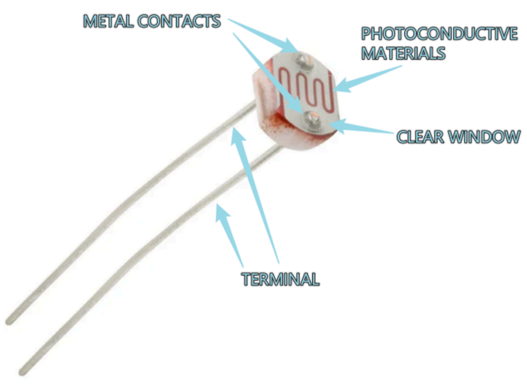




Clockwise or Right Increase Delay. Fully right and the delay will be approximately 5 minutes	**Clockwise or Right** Decreases Sensitivity. Fully right and the range will be approximately 3 meters
Counter-Clockwiser or Left Decreases Delay. Fully left and the delay will be approximately 3 seconds	**Counter-Clockwiser or Left** Increases Sensitivity. Fully left and the range will be approximately 7 meters

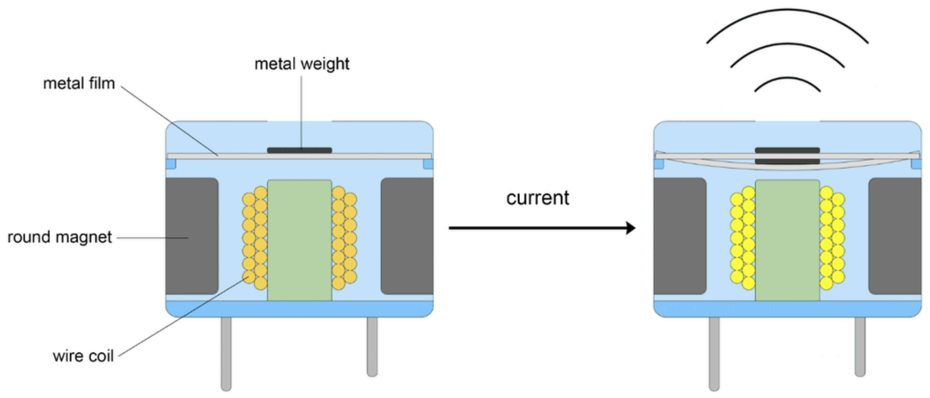
[Topic 2 - Light Sensor]

The light sensor that we will be using is a photoresistor which is a passive component that decreases in resistance as a result of increasing light on its surface. With this component, we want to detect when there is a significant increase in light, such as a ceiling light being turned on.



[Topic 3 - Buzzer Device]

The buzzer that we will be using is an active buzzer that contains a built-in oscillating source. With this source type, the active buzzer will activate when a signal is sent from the STM32 board. To go into the structure of the buzzer, it is mainly comprised of the oscillator, solenoid coil, vibration diaphragm, and magnet. When the current passes through the solenoid coil, it generates a magnetic field in which the vibration diaphragm periodically vibrates to emit a frequency that can be heard.



Results

Test Cases

<u>LED System</u>

- Testing if the Red LED will activate when entering the Triggered State
  - After coding the Triggered Handler, when the motion sensor or light sensors send the program to enter the Triggered State, the Red LED does indeed light up while all the other LEDs are off.

- Testing if the Red LED will activate when entering an incorrect passcode
  - After coding the Passcode Handler, when the handler detects a missed input on either button, the Red LED activates and flashes.

- Testing if the Yellow LED will activate when entering the Idle State
  - After coding the Passcode Handler, Idle Handler, and Triggered Handler, when the passcode handler sends the program to enter the Idle State from the Triggered State, the Yellow LED does indeed light up while all the other LEDs are off.

- Testing if the Green LED will activate when entering the Armed State
  - After coding the Armed Handler, when the passcode handler sends the program to enter the Armed State from the Idle State, the Green LED does indeed light up while all the other LEDs are off.

- Testing if the Green LED will activate when entering the correct passcode
  - After coding the Passcode Handler, when the handler detects a correct input from both buttons, the Green LED activates and flashes.

<u>Buzzer System</u>

- Testing if the Buzzer will activate when inputting the wrong passcode to the keypad
  - After coding the Triggered Handler, when the motion sensor or light sensors send the program to enter the Triggered State, the buzzer does indeed power on and buzzes until the user disarms the security system

<u>Infrared Motion Sensor</u>

- Testing how responsive the motion sensor is:
  - I noticed there was a delay in the responsiveness of the motion sensor. However, since the motion sensor has built-in potentiometers that control the sensitivity and timing of the signal, tuning the system to avoid delay was fairly easy. I also tuned back the sensitivity of the motion sensor to give more leeway when the system is in its Armed State. Before, when the motion sensor was being used, it noticed there were times when the sensor would detect motion, even though there was not. So, to counteract this, I turned back the sensitivity of the motion sensor, which solved the issue.

- Testing if the motion handler will correctly trigger the security system
  - After writing the code for the motion sensor to be incorporated into the various handlers and tuning the timing and sensitivity of the sensor itself, I tested whether the motion sensor would cause the security system to go into its Triggered State. When the system was done arming, I tested whether the motion sensor would detect my moving hand and it did. Once I moved my hand in front of the sensor, the security system immediately went into its Triggered State, as indicated by the red LED and buzzer noise.

<u>Keypad System</u>

- Testing if a specific passcode will disarm the security system
  - Programming the passcode to be button 1, button 2, button 1, button 2, and pushing the buttons in that order changes the state from triggered to idle as intended.

- Testing if a specific passcode will arm the security system
  - Using the same passcode as before, inputting the passcode while in the idle state does arm the alarm.

- Testing if holding down the buttons between inputs breaks the logic flow
  - Holding each button down for at least a second before releasing the button at each input does not break the logic flow of the passcode handler and acts as if you just tapped the button. It's working as intended.

- Testing if pressing the buttons at different time intervals breaks the logic flow

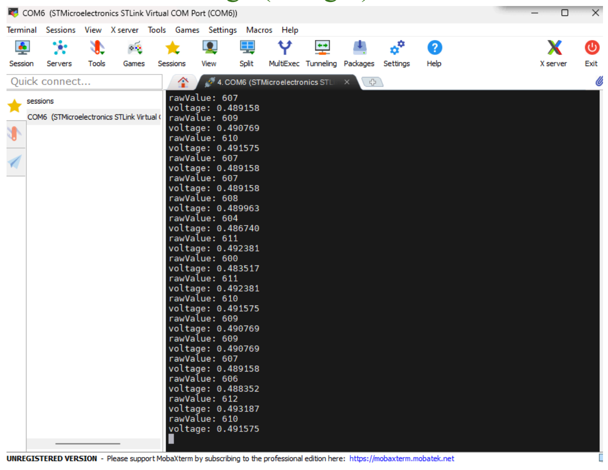
    - Whether you push the buttons in quick succession, 1 second between each press, or at varying intervals, the logic flow does not break and works as intended.

Photoresistor Sensor

Through various tests between members, it has been discovered that each of our photoresistors performed differently from each other. For instance, one photoresistor would be very sensitive to light, while another photoresistor would be resilient to light. So, it must be said that for each member, various changes to the voltage value had to be used as a basis for the light handler for the photoresistor to work amongst our own builds.

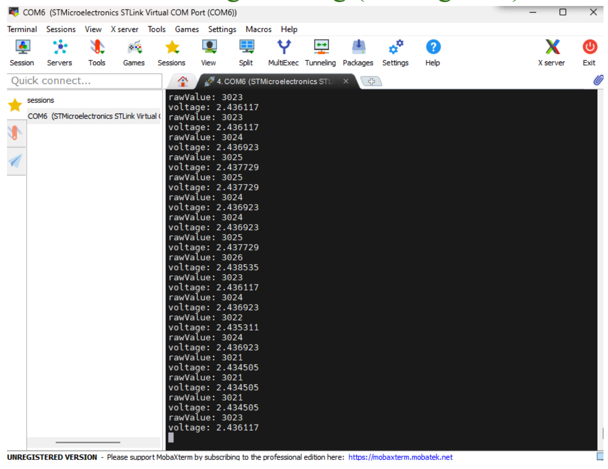
- Testing how responsive the light sensor
    - Since the photoresistor acts as a resistor based on the light intensity over its surface, we wanted to ensure that our light sensor only responds to significant changes in light intensity. For example, we may not care for it to detect changes in natural light but we don't want it to detect when a light switch may have been turned on in a room while the security system is armed. To do this, I used ADC conversion to look at the voltages at the phototransistor and found that in a room with only a natural light source, it had a significantly low voltage of ~0.4V and when a flashlight is turned on near the photoresistor, the voltage level was approximately 2.4V depending on the distance between the flashlight and the photoresistor. For our prototype, I based our threshold on these results but it can easily be changed to any desired sensitivity level based on your environment.
    - These results were obtained using a 4.7k resistor connected in series to ground.

Photoresistor Voltage (no light)


COM6 (STMicroelectronics STLink Virtual COM Port (COM6))
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help X server Exit
Quick connect...
4. COM6 (STMicroelectronics STL
sessions
COM6 (STMicroelectronics STLink Virtual (
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

```
rawValue: 607
voltage: 0.489158
rawValue: 609
voltage: 0.490769
rawValue: 610
voltage: 0.491575
rawValue: 607
voltage: 0.489158
rawValue: 607
voltage: 0.489158
rawValue: 608
voltage: 0.489963
rawValue: 604
voltage: 0.486740
rawValue: 611
voltage: 0.492381
rawValue: 600
voltage: 0.483517
rawValue: 611
voltage: 0.492381
rawValue: 610
voltage: 0.491575
rawValue: 609
voltage: 0.490769
rawValue: 609
voltage: 0.490769
rawValue: 607
voltage: 0.489158
rawValue: 606
voltage: 0.488352
rawValue: 612
voltage: 0.493187
rawValue: 610
voltage: 0.491575
```

Photoresistor Voltage Testing (flashlight on)


COM6 (STMicroelectronics STLink Virtual COM Port (COM6))
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help X server Exit
Quick connect...
4. COM6 (STMicroelectronics STL
sessions
COM6 (STMicroelectronics STLink Virtual (
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net

```
rawValue: 3023
voltage: 2.436117
rawValue: 3023
voltage: 2.436117
rawValue: 3024
voltage: 2.436923
rawValue: 3025
voltage: 2.437729
rawValue: 3025
voltage: 2.437729
rawValue: 3024
voltage: 2.436923
rawValue: 3024
voltage: 2.436923
rawValue: 3025
voltage: 2.437729
rawValue: 3026
voltage: 2.438535
rawValue: 3023
voltage: 2.436117
rawValue: 3024
voltage: 2.436923
rawValue: 3022
voltage: 2.435311
rawValue: 3024
voltage: 2.436923
rawValue: 3021
voltage: 2.434505
rawValue: 3021
voltage: 2.434505
rawValue: 3021
voltage: 2.434505
rawValue: 3023
voltage: 2.436117
```

- Testing if the light handler will correctly trigger the security system
    - After writing the code and ensuring that the light sensor is functioning correctly on its own, it was implemented into our full project code and worked as expected. When the system was successfully armed, I placed a flashlight above the photosensor and turned it on. This successfully

triggered the alarm system and went into the Triggered State, which is indicated by setting our Red LED and buzzer.

Data Logging

It must be stated that the code written for this portion of the project could have worked better than we intended. The way we planned to implement the data logging was by communicating between the STM32 board and the Raspberry Pi through the use of a UART connection. However, when testing, the data that was being transmitted was inconsistent. Sometimes the shell on the Raspberry Pi was able to display the printed messages, but other times nothing would display at all. Although countless benchmarking tests were done to try and transmit readable prints, we were still not able to create a reliable data logging system.

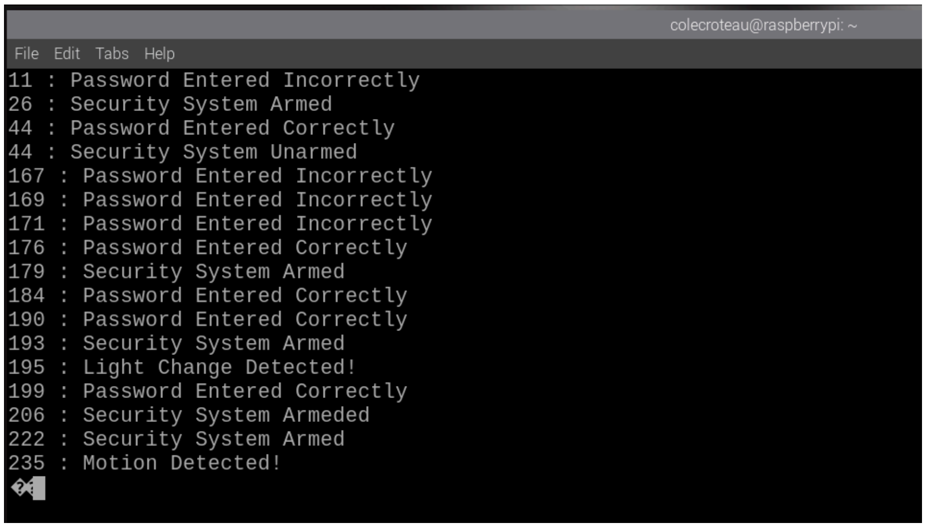
- Logging password attempts on the keypad
    - Inputting a correct password correctly logs it to the RasPi, same goes for an incorrect password

- Logging when the security system was last armed
    - The RasPi correctly displays when the system goes to an armed state

- Logging when the security system was last unarmed
    - The RasPi correctly displays when the system goes to an unarmed state

- Logging when the security system was last set off
    - The Raspi correctly displays when the system goes to an armed state.

- Logging how the security system was set off
    - The Raspi correctly displays when the system gets triggered by either the motion sensor or photoresistor, and displays which caused it accordingly.



```
11 : Password Entered Incorrectly
26 : Security System Armed
44 : Password Entered Correctly
44 : Security System Unarmed
167 : Password Entered Incorrectly
169 : Password Entered Incorrectly
171 : Password Entered Incorrectly
176 : Password Entered Correctly
179 : Security System Armed
184 : Password Entered Correctly
190 : Password Entered Correctly
193 : Security System Armed
195 : Light Change Detected!
199 : Password Entered Correctly
206 : Security System Armeded
222 : Security System Armed
235 : Motion Detected!
�
```

Conclusion

This project was able to do and achieve what it was intended to do, act as a security system. The project was able to switch between the three different states of Idle, Armed, and Triggered, where each of the states is determined by an LED system. Not only that, the project was able to be used in combination with a PIR motion sensor and a photoresistor to detect changes in movement and light, which would, in turn, trigger the system and sound the buzzer. Furthermore, we were also able to incorporate a passcode system to be able to enter code to either arm or disarm the security system.

Although the project performed at semi-acceptable levels, many areas of improvement could have been made. For instance, the keypad passcode system, although functional, can be quite finicky and unpredictable. So, better implementation of that system, where the passcode system is snappy and responsive, could benefit the security system as a whole. Another area of improvement that was unable to be implemented was a Data Logging system. Although the security system was able to keep track of the photoresistor voltage using ADC Conversion, we were not able to implement a reliable data logging system using the Raspberry Pi. Each attempt with trying to implement the Raspberry Pi, it just resulted in further problems that we were, unfortunately, unable to solve in time. In some cases, the Raspberry Pi was able to print out the needed messages through UART. However, in other cases, the shell would display nothing at all. So, if more time was provided to implement a functional data logging system, the security system, as a whole, could definitely benefit.

Even though the security system was not able to implement all the features that we, as a team, sought out for this project as a prototype, it truly performed well at what it was intended to do. The project was able to inherit a finite state machine that simulated the three different states that the security system has and was able to use various sensors to detect changes in the environment or alarm when the system was triggered. Furthermore, the project was able to use LEDs to signify the state of the system and a passcode system to either arm or disarm the system. Overall, this project was able to do and achieve what it was intended to do, act as a security system.

References

Any outside references or resources used for the project.
Web links to reference sites used, (data sheets, sensors, open software sites, etc.)

Pinouts for the STM32(F30K8) Board:

- https://os.mbed.com/platforms/ST-Nucleo-F303K8/

For the Photoresistor:

- https://www.micropeta.com/video18#google_vignette

For the Motion Sensor (includes pinouts for the board):

- https://www.amazon.com/HiLetgo-HC-SR501-Infrared-Sensor-Arduino/dp/B07KZW86YR/ref=sr_1_3?crid=2U6PQH3CEMAUJ&dib=eyJ2IjoiMSJ9.zxcyRjj1a2cgLHKDl6EoPPrTi8EKo_te6iPskZC2ypzKZu-n0ZLX3Fe6cvrJM1paPv0HBhkMez94OE_0S7MP6IgHvlVgOuy6LaKcWqWTnK8Ui4-RZM5wFMrrunmtoxmW9I7LyztY3t3BuqYYjCT1ZMq7YVlINUjBzQvhIup9Te0fldymjkkj92C9i5VcyVUvXZ_NU3fGcwrjMPb38Jfv6SrhwOZrrAuHhQGDrbnAbpg.k2cSjHfOVs-OF9__Q6M4h3em6oNdGV83crR5ylUXtZw&dib_tag=se&keywords=pir+motion+sensor+raspberry+pi&qid=1709764389&sprefix=pir+motion+sensor+rasberry+p%2Caps%2C139&sr=8-3

For the Buzzer Component:

- https://components101.com/misc/buzzer-pinout-working-datasheet

Pinouts for the Raspberry Pi:

- https://iosoft.blog/2019/01/28/raspberry-pi-openocd/