GrovePi Security System

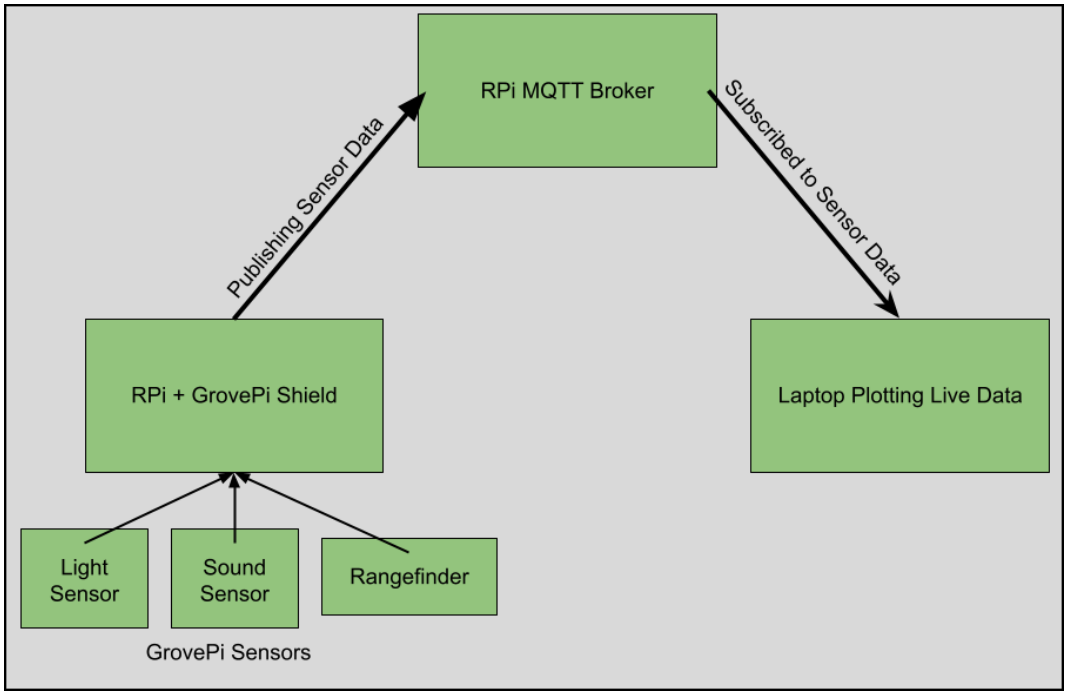
Jerry Appelhans, Riad Alasadi

Description:

Our project is a security system that uses light, sound, and rangefinder measurements to protect a valuable object. Our device could be placed inside a vault which would be dark and quiet under normal conditions. If someone tried to break into the vault, the sound sensor would detect any loud noises. If someone were able to open the door, the light sensor would detect light from outside, and the rangefinder would detect that the door has moved. If any of these sensors receive abnormal measurements, an alarm is triggered.

In addition to these local capabilities, the system publishes the sensor data to a broker using MQTT. A laptop can subscribe to this MQTT topic and receive the data to plot in real time. This live plot serves as a remote monitoring system for the vault.

Block Diagram:



Rpi.py:

The Rpi.py file is run on the RPi. When the code is run, it first creates a new thread to connect to the MQTT broker. The code then continuously reads data from each of the light, sound, and rangefinder sensors to provide measurements to compare with the thresholds. If any of the measurements exceed their corresponding thresholds, the buzzer is triggered, and

therefore a loud sound is heard on the buzzer until the rest button is pressed. During this time, Rpi.py is publishing these measurements to the "security/sensor_data" topic.

This part of the project was straightforward, as all that was needed to be done was to setup the publishing of the Rpi.py to the broker and making sure that each component reacted accordingly when the readings exceeded the thresholds

laptop.py:

The laptop.py file is run on the laptop. When the code is run, it first creates a new thread to connect to the MQTT broker. When the RPi publishes messages to the "security/sensor_data" topic, the laptop receives these messages from the broker. In the callback, the data is appended to a message buffer to temporarily store it for plotting.

For the live data plotting, we used the "matplotlib.animation" Python library. This allows for new frames to be added to an existing plot. The plot of the sensor data updates every 500ms using the last message in the message buffer mentioned above. The first three plots show the incoming sensor data, and the fourth plot displays the current status of the alarm (0 for off and 1 for on). The threshold values for each sensor are plotted in red.

One major challenge with writing this code was managing the MQTT subscription and data plotting concurrently. When the MQTT client connects to the broker, it enters an infinite loop that waits for new messages. This made it impossible to also update the plot. To fix this issue, we created a separate thread to handle the MQTT callbacks.

Another challenge was synchronizing the data messages with updating the plot. To fix this issue, we just publish from the RPi more often than updating the plot. The RPi sends new measurements every 100ms which are added to the message buffer. The plot is only updated every 500ms using the latest available data.

Reflection:

Overall, we found this project to be a success, not only because we were able to produce a security system that functioned correctly, but also because we were able to use our knowledge gained from this course, specifically from the MQTT, GrovePi, and data processing labs. With this success comes some limitations; for example, the limited capabilities of the available hardware would prevent this project from being used for commercial use. An example of these hardware limitations was experienced by us when using a defective sound sensor that needed to be replaced. However, the overall idea of this project can be adopted using more advanced technologies when designing a security system.