EE250 Final Project - Home Security

Renzhi Li, Anchal Srivastava

Project Description:

Our project is a model of a home security system, consisting of a fire/burglary detection and alerts. It consists of 3 sensors [GrovePi ultrasonic sensor, GrovePi temperature & humidity sensor (DHT), and webcam from laptop], three nodes [Laptop, Azure Virtual Machine (cloud), and Raspberry Pi (Rpi)], a web page [show alarm status and provides button to sound/reset alarm], and a buzzer [physically sound alarm].

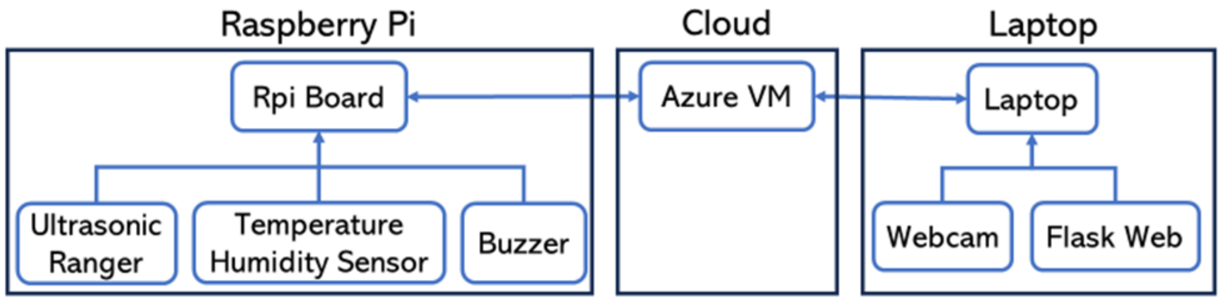

Figure 1. System Block Diagram

Components:

Burglary Alarm: The ultrasonic ranger detects if an object is close [someone passing through the door], and then triggers the laptop webcam to capture an image of the people/object entering. The captured image is sent to the cloud, where facial recognition processing is performed. If no person is detected in the image or if the captured person is the house owner, no action is taken. However, if the person is not recognized, an alarm is sent through MQTT. Then, the buzzer on Rpi is sound and the Flask webpage displays "burglary alarm". The alarm stays active until the alarm is resetted.

Fire Alarm: The temperature and humidity sensor detects local temperature and humidity. If the temperature goes above 40 degree Celsius, the Fire Alarm is sound. The Alarm then sets off the buzzer, and displays the Fire Alert on the Flask webpage. The alert stays active on the webpage and the buzzer stays on until the alarm is resetted.

Visualization: The Flask webpage has a "Reset Alarm" button and a display section for showing ongoing alarms. A fire alert will show "Fire Alert," a burglary alert will show "Burglary Alert," and if there's no active alarm currently the web will simply show "No Alert." Clicking on the reset button publishes a no alert message to the cloud and raspberry pi which sets the buzzer output to 0, hence stopping the alarm.

Protocols:

The primary form of communication between the nodes is MQTT. The cloud is subscribed to the "/ultrasonic", "/dht", and "/camera"; Laptop VM is subscribed to the camera trigger; and the rpi is subscribed to alarm. The rpi reads from ultrasonic ranger and DHT, and then publishes the data read with the local time of the measurement (json) to "/ultrasonic" and

DHT topics separately. The cloud then processes the data received through MQTT. If the temperature is above 40 Celsius, it will publish a “F” (fire) to “/alarm”. If something is detected to be less than 100 cm, then the cloud publishes to the “cameraTrigger” with the message “C”, which tells the camera to capture an image. If the face on the picture captured is not identified, the cloud publishes “B” (burglary) to “/alarm”. When receive “F” or “B” message on “/alarm”, the webpage will display alarm status and the buzzer connected to rpi will sound. Once the reset button on the webpage is clicked, the webpage publishes “S" to “/alarm”. Upon receiving the message, the alarm status on flask web will display “no alarm” and the buzzer will stop sounding.

Processing Techniques:

Facial Recognition: The main processing technique used is facial recognition for burglary detection. The facial recognition software uses the pictures organized by the name of the person in the image in the training folder to train the data. The valid users can be authenticated by training the model using these saved pictures. We use the hog (histogram of oriented gradients) model, which is a common model for object detection. We used python’s library, face_recognition, to detect the face in each image and get its encoding. The encoding is saved using the pickle library. Once the training images are encoded, the image captured from the webcam is processed with the recognize face function. The image processing is run on the azure cloud vm.

Other Implementations:

OpenCV Webcam Implementation: To get the webcam triggered and use it to capture the image, we used python’s OpenCV library to call on VideoCapture.

Challenges:

One of the drawbacks of this project was that our temperature and humidity sensors have a very limited range of readings. So the threshold we set for our detection is probably not aligned with what the temperatures for an actual fire would look like. Another particular challenge we faced was figuring out how to send the image from the laptop to the cloud for image processing, since we couldn’t figure out how to send images over MQTT. We fixed this problem by converting the image into png first and then a byte array so it can be sent through MQTT to be interpreted and opened.