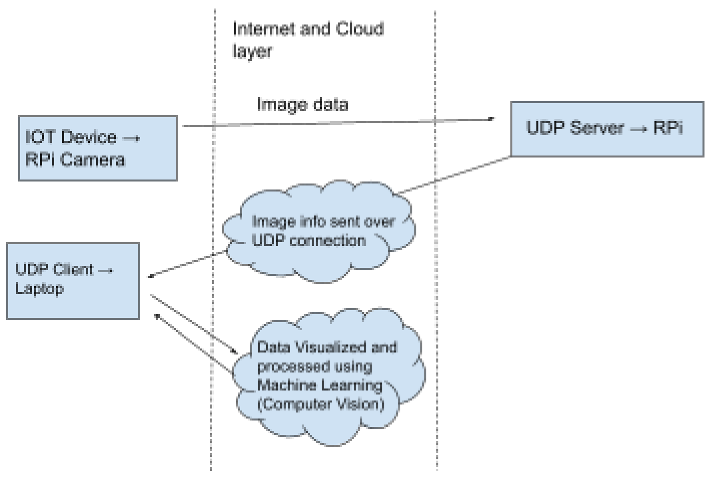
Kameran Mody & Greg Sarandi
4 May 2023
EE 250

IOT Ring Camera Final Project

Abstract:
Our final project is essentially a basic version of the popular Ring camera. Every 21 seconds (note: our video says 20 seconds however it is actually 21), the IoT device (RPi camera) takes an image which is then saved by the RPi (our server) and sent to our laptop (our client). The client-server structure is a UDP protocol. The laptop processes this image using computer vision facial recognition software, with python libraries provided by openCV. This is used to determine if there are faces in the image and if there is, the client outputs how many there are to the terminal. Additionally, the client places a green outline around the faces detected in the image and saves that to the client device. Likewise, it takes cropped images of the client faces and saves those to the client device. If no faces are detected, it only outputs the time taken of the photo as well as “no faces detected”. Thus, in the real world this could be used to detect who is entering one's home or space as a layer of privacy. If it is someone the homeowner does not recognize, they will be able to find out in seconds since the image of the entered person’s face will be sent to the owner’s device.

Block Diagram:


Internet and Cloud layer
Image data
IOT Device → RPi Camera
UDP Server → RPi
Image info sent over UDP connection
UDP Client → Laptop
Data Visualized and processed using Machine Learning (Computer Vision)

Components:
This system begins with the Raspberry Pi Camera 2 NoIR. This device is a raspberry pi extension that plugs into the camera port of the Raspberry Pi. It is capable of taking HD photo and video, and the NoIR feature allows it to take photos at night with the assistance of infrared lighting. We used a Raspberry Pi Module 4 as our specific raspberry pi module, which used the Raspbian Bullseye OS. This specific OS was used because it allows the Pi Camera 2 to work more seamlessly; rather than use the deprecated raspistill feature for photo taking, the libcamera-still feature can be used to take a photo with the pi camera thanks to the Bullseye OS. In our system implementation, we have the camera set to take a photo every 21 seconds. In theory, we would like to simply stream video from the camera to the Rpi server and its clients, however due to the low-processing capabilities of the Rpi this was very difficult to achieve. Instead we settled for a system that essentially takes a non-continuous video feed at 1/21 = 0.04 frames per second (talk about luxury!). Once a photo has been captured by the picamera, it is sent to the

raspberry pi. The raspberry pi then outputs a message saying “image captured at” followed by the time the image was received by the raspberry pi. The raspberry pi then acts as the server in a UDP server-client protocol, with the client being Greg’s laptop. UDP was chosen over TCP since data was being sent every 21 seconds continuously, and timing was preferred rather than pure message accuracy. The raspberry pi sends the image over the UDP protocol to the client by dividing it into packets of 1024 bytes each. Once this process begins, it outputs the message“sending image info”. It then sends each packet to the UDP client. Once this process is done, the raspberry pi outputs the message “finished sending image info”. At the UDP client, the 1024 byte packets are used to reassemble the original image. Once reassembled, the client outputs the message “image received at” following the time the image was received and reassembled. The image is then processed using OpenCV libraries to perform facial detection. The facial detection script uses the HaarCascade weights provided by OpenCV to detect if there is a face present in the image. Once this process begins, the client outputs the message “Analyzing image” to the terminal. If one or more faces are detected, the client outputs the message: “[INFO] xx people have been detected at the door”, with the number of faces replacing xx. If no faces are detected, the client outputs the message: “No faces detected”. Next, if there are faces detected in the image, the openCV libraries are again used to place a green square/rectangle around each face in the image. This image is then saved to the client as a new .jpeg file. Additionally, the openCV libraries are used to crop the faces from each image, and save the individual faces as a new .jpeg file. The end-user can then use these .jpeg files to see who is at the door. All images are saved by the UDP client, regardless of if a face was or was not detected.

Reflection:
Overall, all things considered, this final project went very well. Our device worked very smoothly and we got it to work in a decently quick amount of time. However, it does have its limitations mainly because of the time constraints and materials we had access to. Due to using the Bullseye OS on the Raspberry Pi, we were not able to implement any Grovepi components into the system since they require the legacy Buster OS. We would have liked to use the Grovepi Ultrasonic ranger in our system; with this, we could have the camera only take pictures when an object is detected within 3 feet (about 91.5 cm) and thus be more efficient. Also, we could have used better face detection weights. We used the standard HaarCascade weights provided by OpenCV, however with more time and money we could have used the more accurate Caffe weights, allowing for fewer false positives and false negatives, and thus increased precision, specificity, and recall. Also, we would have liked to use an MQTT pub-sub architecture rather than UDP, however we had issues setting up a broker and thus had to resort to UDP. With more time, we could resolve these issues and implement the desired MQTT pub-sub architecture.

Due to time, we were not able to add a speaker to the device that ideally would make a doorbell sound as a face is detected to perfectly simulate a Ring camera doorbell. Ideally we would have a cover and case for our circuit. This would help reduce wear and tear from elemental factors as well as reduce the possibility of damaged components. Due to low resources and time, we weren’t about to achieve this. Because of this, we would not be able to place our device outside which is a significant limiting factor. Thus, this plays into our lessons learned about the project. We ideally should have started a little earlier so we could order parts to better secure our project and so we could implement all the pieces we wanted to. In addition, we learned that there are infinite applications to IOT projects as you can essentially do any task you put your mind to given how many tools and expansion ideas there are. The only limiting factor to the process is your imagination which makes projects such as these so exciting.