Build a Physical Authentication System for your Home

You will need:

- A Raspberry Pi 2B – with a Wi-Fi module and an SD card (at least 8GB)
- Adafruit VL53L0X Time of Flight Distance Sensor
- A USB Webcam
- An Amazon AWS account
- A phone capable of running Android
- A monitor, HDMI cable, a USB keyboard and mouse (optional)

Introduction:

We equip the Raspberry Pi with a USB camera and a distance sensor. We maintain a database of whitelisted people. The entry fields in the database are the person's face and a unique distance as measured by the distance sensor. When the distance measured by the sensor matches the entry in the database, the camera captures a picture of the person standing before it and verifies it with the corresponding database entry using a Face Recognition software. If there is a match, the administrator is notified on his Android phone via an app. The administrator can then either grant access to the person or deny it using a simple Yes-or-No GUI on the app. The result is then relayed back to the person requesting access. All communication between the Raspberry Pi and the administrator's Android phone is done using a sever hosted in Amazon AWS cloud.

What You Will Learn:

- Configuring your Raspberry Pi to capture images from a USB webcam connected to it.
- Working with Adafruit VL53L0X Time of Flight Distance Sensor.
- Working with Amazon AWS services like Amazon S3, Amazon Rekognition, Amazon EC2 and Amazon Polly.
- Basic server-side scripting and Android Studio programming.
- System and Network related issues to decrease latency and get the best out of your setup.

Let's Build:

1) Configuring your RaspberryPi 2B:

- Load Raspbian operating system onto you SD card from your computer by following the instructions on this page: https://www.raspberrypi.org/documentation/installation/installing-images/

- If you would like to remotely access the Raspberry Pi, then you could use VNC by following this guide: https://www.raspberrypi.org/documentation/remote-access/vnc/
- Insert the SD card in your Raspberry Pi. Connect all the peripherals (the USB webcam, HDMI cable, Wi-Fi module, USB for keyboard and mouse). Boot it up. Select Raspbian by NOOBS. The operating system will be installed and the Raspberry Pi will reboot automatically.
- Install the following libraries:
  - boto3 - Used to automate AWS functionalities like EC2, S3, Rekognition and Polly
  - pygame - Used to play .mp3 files from raspberry Pi
  - os
  - requests
  - time
  - v4l2capture - Used to control webcam connected to raspberry Pi (link for installation instructions: https://github.com/gebart/python-v4l2capture)

  - VL53L0X - Used to obtain data from the distance sensor (link for installation instructions: https://github.com/johnbryanmoore/VL53L0X_rasp_python)
- Connect the distance sensor to the Raspberry Pi as shown in the link. You would have to enable I2C in your Raspberry Pi preferences to get it work: https://raw.githubusercontent.com/johnbryanmoore/VL53L0X_rasp_python/master/VL53L0X_Mutli_Rpi3_bb.jpg
- All subsequent programming must be done in the "python" subdirectory of the directory in which the VL53L0X has been installed.
- Create your database using the programs in the "Database" folder and customizing them to your needs.

2) Setting up AWS:

- Create your AWS account.
- You would need to set up AWS CLI on your Raspberry Pi to use Amazon AWS services directly from the device without hard-coding your credentials into your program. Follow the instructions in this video: https://www.youtube.com/watch?v=FwbavIglhis
- Using Amazon S3 – Amazon S3 is a Simple Storage Service which we use to store the and retrieve the images captured from the Raspberry Pi. Amazon Rekognition uses a request field requiring the file's S3 bucket key. We just create a bucket to which we can upload our standard image which will be used by Amazon Rekognition to compare faces with.
- Using Amazon EC2 – Amazon EC2 is an Elastic Compute Cloud where we run our server. You can choose any Linux-based cloud for your instance. I chose the Ubuntu version. Create a key pair and store this .pem file in a safe directory on your

Raspberry Pi. This file is essential for all communication between your Amazon EC2 instance and your device. Change the file's permission to read only owner by using the command "chmod 400". Configure the Security Group of your Amazon EC2 instance to allow all incoming TCP connection so that your Android phone and Raspberry Pi can connect to it from anywhere.

- Using Amazon Rekognition – Amazon Rekognition provides various Face API which we can use to verify the identity of the person standing before the camera. We use the Face – Verify API to compare two different images and check if the person in these images are the same.
- Using Amazon Polly – Amazon Polly is a text-to-speech converter from which we can download the mp3 file of the text we enter in the comment box.
- All the above Amazon AWS services can be requested through boto3 API, the documentation for which is given in this link: https://boto3.readthedocs.io/en/latest/
- Our system is very light on memory and compute power so the free-tier which Amazon provides is sufficient for our needs.

3) Setting up your Server and Android App:

- SSH into your Linux Server.
  - For the polling version of the system, run the following command in your server terminal: "nohup python -m SimpleHTTPServer 8000", where 8000 is the port at which the server listens and nohup ensures that the program keeps running even after the terminal is closed.
  - For the non-polling version, run the program "longpollserver.py" in the server terminal.
- Make appropriate changes to the code, like changing the IP address of the Linux instance. The relevant files necessary to build the app have been posted in the repository.

How It Works:

- The Raspberry pi runs the VL53L0X distance sensor which starts collecting distance measurements.
- The webcam connected to the Raspberry Pi turns on and captures a picture when the unique distance is matched.
- The captured photo is verified against the standard using a face recognition technology. This can be achieved either by
  - Posting the captured image into an Amazon S3 bucket and using Amazon Rekognition to verify the face, or
  - Saving the image on the Raspberry Pi and running a face recognition software such as OpenCV or dlib on the Raspberry Pi itself.
- If the faces don't match, a "Sorry" status message is played and the program ends.
- If the faces match, the webcam starts capturing images and pushing them to an Amazon EC2 server.

- Images from the Amazon EC2 server are relayed to an Android app on the administrator's phone.
- The administrator can then decide whether to authenticate the person in front of the webcam using a simple Yes-or-No GUI.
- If the administrator grants access, a "yes" file would be relayed back to the server and then to the raspberry Pi.
- The authentication command can be delivered to the Raspberry Pi in two ways:
    - The polling approach, in which the Raspberry Pi keeps requesting the server for the result recursively, and
    - The non – polling approach in which the server “pushes” the authentication result to the Raspberry Pi when it is ready.
- If the raspberry Pi does find the result, it plays a "Welcome" message and quits the program. If it doesn't it plays a "Timeout" message and quits the program.

System Performance:

As evident from the section above, our system can be realised in many ways like for example: where the face recognition should be run and how the Raspberry Pi should receive information from the server. Not all versions offer the same latency. The main factors that affect the performance of the system are compute capacity and network connectivity. The best performance is observed when the face recognition is run on the server instead of the Raspberry Pi and when the server pushes the authentication result instead of the Raspberry Pi polling the server for the result.

Network overhead:

The type of protocol used to transmit data determines the network overhead incurred by the system. The header of the packet is the main source of this overhead. UDP header contains just the destination and source IP/PORT whereas TCP header contains many things like the sequence number of the packet to ensure ordered delivery, flags to ensure the packet is received and checksum of the data to make sure it didn't get corrupted and received correctly.

A 640x480 image frame is usually 20kB – 25kB and requires 14 – 18 packets (MTU = 1406B). The authentication result is in the form of a text file whose size is 3B and can be fit into a single packet. TCP/IP is the protocol I have used throughout the system. A TCP packet header is usually 40B. This means that the authentication result incurs 4300% TCP/IP overhead and each image frame incurs 2.80% - 2.88% TCP/IP overhead. Since we don’t require all functionalities that TCP offers for our system, using UDP greatly reduces the network overhead incurred, thus decreasing latency.

Running face recognition software on Raspberry Pi v/s using an API over the Internet:

Face recognition can either be run on Raspberry Pi or over the internet. On the Raspberry Pi, we can use software such as OpenCV and dlib. Amazon Rekognition constitutes to using an API over the internet. This presents a trade-off between processing power and network traffic. Using an API over the internet causes more network traffic whereas performing face recognition on the Raspberry Pi is heavy on its resources. I used dlib for face recognition. dlib takes around 10 seconds to return a result for a pair of images whereas Amazon Rekognition takes around 5 seconds for the same image pair. OpenCV and dlib take a lot of time to build and compile on the Raspberry Pi, literally pushing it to its limits. Also, they are open source libraries which makes them noticeably inferior to licensed software like Rekogntion. This is especially true with low-light images.

It is generally the case that the Amazon cloud has more compute capacity than the Raspberry Pi. So, it is wise to push facial recognition, a resource - intensive task, to the server rather than perform it on the Raspberry Pi.

<u>Image Frame Rate:</u>

Image frame rate is defined as the number of images refreshed in a second at the end user (in our case – the Android app). The fastest frame rate I could achieve was 5 frames per second but this was when I was displaying pre-recorded images. During a live-stream, the value was just 1 frame per second.

The factors that affect the image frame rate are the capture time of images, upstream bandwidth from Raspberry Pi to the server and downstream bandwidth from the server to the Android app. The main bottleneck in the transfer of images is the upload rate from the Raspberry Pi.

Each time, the Raspberry Pi should open the webcam, capture an image, save it and then upload it to the server. This process is the main source of delay in the streaming process. Decreasing the image resolution to decrease the size doesn’t help much: this suggests that more time is spent creating a secure connection between the Raspberry Pi and the server than transferring the file. Using UDP instead of TCP or SCP could speed up this process but at the cost of a less secure transmission.

<u>Polling v/s Non-polling:</u>

Polling is the process in which the Raspberry Pi iteratively enquires the server for the authentication result. I implemented Long-polling as an alternative to polling. In polling, the server replies instantly to the Raspberry Pi irrespective of it having the authentication result whereas in long-polling, if the server isn’t ready yet, it delays the response until it receives the authentication result.

Since the Raspberry Pi keeps waiting for response from the server in long-polling, it becomes incapacitated once it enters that thread. This makes live streaming impossible when implementing long-polling.

On the other hand, when the Raspberry Pi is iteratively querying the server for the authentication result, it is flooding the network with useless packets. This is because, the server must reply in negative to every query of the Raspberry Pi when it isn’t ready yet. The non – polling implementation is very “network – friendly” since communication between the server and the Raspberry Pi for the authentication result only occurs once.