IOT Report

What you will learn:

To build a Physical home authentication system

"A camera and distance sensor are connected to a raspberry pi which gives the live video stream to the server which in turn transmits to android using socket programming.

- How to read data from camera and distance sensor.

-How to detect faces using facial recognition program (opencv).

-How to send live video stream from Pi to server and display it on android.

- Also gesture recognition as extra password for authentication."

EQUIPMENT : Raspberry pi Kit , distance sensor, USB camera module, speaker.

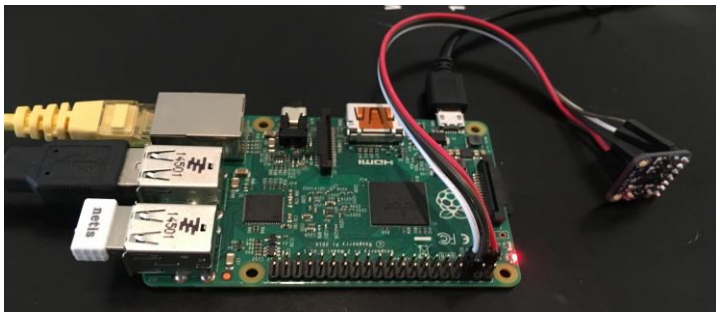
10 steps to build IOT system

1. Install operating system for the raspberry pi using the following links.

http://www.instructables.com/id/My-Raspberry-Pi-Setup/

2. Connect a distance sensor to Raspberry Pi. You can use the Python library available at

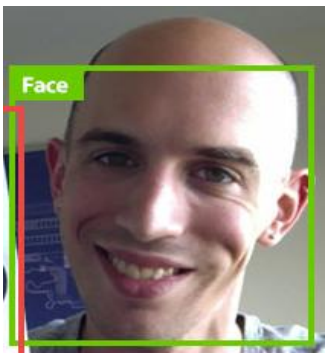
https://github.com/johnbryanmoore/VL53L0X_rasp_python to use the sensor.



3.Install opencv software on Raspberry pi for Face Recognition. (Takes almost 4 hours :P)
https://www.pyimagesearch.com/2015/07/27/installing-opencv-3-0-for-both-python-2-7-and-python-3-on-your-raspberry-pi-2/

4. Face Recognition

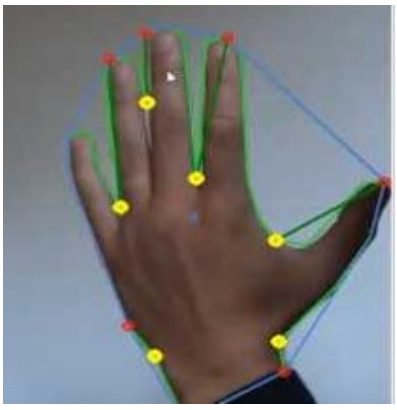
I. face_detection.py - Firstly we capture images of users to feed the training data.
II. Trainer.py - Train the algorithm with the image database of users
III. Face_recognition.py - This program captures a image, detects the face and classifies the name of user.



5. You may also implement Gesture recognition for authenticating the user along with face recognition

Gesture recognition

I. Gesure.py - Detects the hand from webcam feed and identifies the unique gesture and checks that with password set by the owner.



6. Now we need to set up server and obtain access keys for the server. You can set up ec2-server from here- https://aws.amazon.com/ec2/
7. Install Android studio With sdk version 22
8.w need to run client server program on server and raspberry pi.

I. Iot_final.py - this program runs on raspberry pi, it captures images of visitors of the house and gives the image input to facial recognition program which in turn

identifies the person's name. Then program verifies the user by checking in the database and sends stream of images to server.

II. finalserver.py -This program runs on server which takes input image stream from the pi and gives the input to Android.

III. Android app - It takes input from owner whether to authenticate the user or not and sends the authentication status to server which in turn sends it to pi.

9. Now, when server sends authentication status to Pi it should give voice message "welcome home Teja!" , for this we need to install amazon TTS

10. Install it from here- https://aws.amazon.com/polly/

SYSTEM PERFORMANCE

Video streaming is slow because pi processes each image and sends one image frame after another to the server.

Facial recognition system program requires huge training data in order to correctly classify the faces else it incorrectly classifies the identity.

On Android app, we can automate to give push notifications to user when ever a raspberry pi camera detects a person.

The Report

NETWORK MEASUREMENT INSIGHTS

1. Factors that determine network overhead are

In Tcp , we have 40 bytes of overhead along with that if packets are lost or duplicate packets can result in extra bytes sent to the server which results in overhead

For example our file size is x bytes

It is sent in x/1500 packets

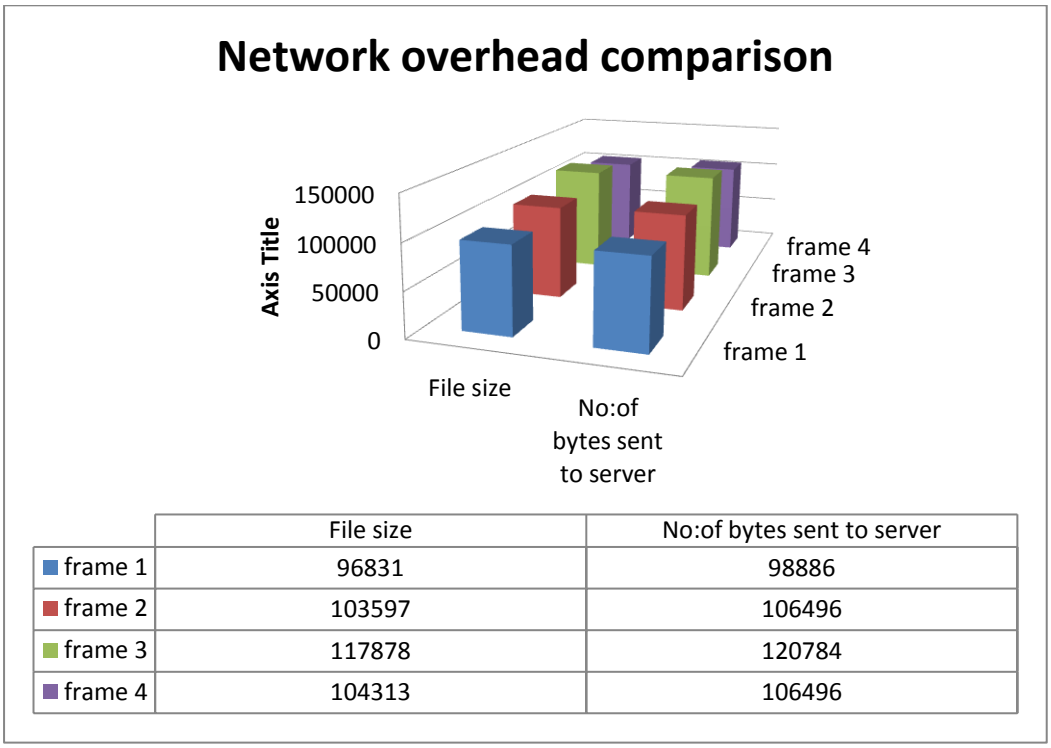
For each packet Tcp overhead = 40 bytes

For handshake syn, ack packets are also overhead ,assume this as s1 bytes

For lost packets or errors, extra bytes of data are sent , assume this as s2 bytes

so overhead = x- 40 - s1 - s2

```
4357 81.747447297 18.220.130.139 → 10.0.0.137   TCP 95 65533 → 44236 [PSH, ACK] Seq=1 Ack=40 Win=26880 Len=29 TSval=574193969 TSecr=282606
4358 81.747580109   10.0.0.137 → 18.220.130.139 TCP 54 44236 → 65533 [RST] Seq=40 Win=0 Len=0
4359 81.747917971 18.220.130.139 → 10.0.0.137   TCP 66 65533 → 44236 [FIN, ACK] Seq=30 Ack=40 Win=26880 Len=0 TSval=574193969 TSecr=282606
4360 81.747966876   10.0.0.137 → 18.220.130.139 TCP 54 44236 → 65533 [RST] Seq=40 Win=0 Len=0
4361 81.765151951 18.220.130.139 → 10.0.0.137   TCP 74 65533 → 44238 [SYN, ACK] Seq=0 Ack=1 Win=26847 Len=0 MSS=1460 SACK_PERM=1 TSval=574193973 TSecr=282608 WS=128
4362 81.765268877   10.0.0.137 → 18.220.130.139 TCP 66 44238 → 65533 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=282613 TSecr=574193973
4363 81.765398720   10.0.0.137 → 18.220.130.139 TCP 80 44238 → 65533 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=14 TSval=282613 TSecr=574193973
4364 81.765908299   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=15 Ack=1 Win=29312 Len=1448 TSval=282613 TSecr=574193973
4365 81.765976215   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=1463 Ack=1 Win=29312 Len=1448 TSval=282613 TSecr=574193973
4366 81.766107047   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=2911 Ack=1 Win=29312 Len=1448 TSval=282613 TSecr=574193973
4367 81.766142985   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=4359 Ack=1 Win=29312 Len=1448 TSval=282613 TSecr=574193973
4368 81.766224755   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=5807 Ack=1 Win=29312 Len=1448 TSval=282613 TSecr=574193973
4369 81.766291577   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=7255 Ack=1 Win=29312 Len=1448 TSval=282613 TSecr=574193973
4370 81.766361108   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=8703 Ack=1 Win=29312 Len=1448 TSval=282613 TSecr=574193973
4371 81.766422253   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=10151 Ack=1 Win=29312 Len=1448 TSval=282613 TSecr=574193973
4372 81.766512252   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=11599 Ack=1 Win=29312 Len=1448 TSval=282614 TSecr=574193973
4373 81.822576138 18.220.130.139 → 10.0.0.137   TCP 66 65533 → 44238 [ACK] Seq=1 Ack=15 Win=26880 Len=0 TSval=574193987 TSecr=282613
4374 81.822701710   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [PSH, ACK] Seq=13047 Ack=1 Win=29312 Len=1448 TSval=282619 TSecr=574193987
4375 81.822741709   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=14495 Ack=1 Win=29312 Len=1448 TSval=282619 TSecr=574193987
4376 81.826827719 18.220.130.139 → 10.0.0.137   TCP 66 65533 → 44238 [ACK] Seq=1 Ack=2911 Win=32640 Len=0 TSval=574193987 TSecr=282613
4377 81.826883656   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=15943 Ack=1 Win=29312 Len=1448 TSval=282620 TSecr=574193987
4378 81.826916521   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=17391 Ack=1 Win=29312 Len=1448 TSval=282620 TSecr=574193987
4379 81.827021051   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=18839 Ack=1 Win=29312 Len=1448 TSval=282620 TSecr=574193987
4380 81.827080738   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=20287 Ack=1 Win=29312 Len=1448 TSval=282620 TSecr=574193987
4381 81.827359382 18.220.130.139 → 10.0.0.137   TCP 66 65533 → 44238 [ACK] Seq=1 Ack=7255 Win=41344 Len=0 TSval=574193987 TSecr=282613
4382 81.827418652   10.0.0.137 → 18.220.130.139 TCP 1514 44238 → 65533 [ACK] Seq=21735 Ack=1 Win=29312 Len=1448 TSval=282620 TSecr=574193987
```



	File size	No:of bytes sent to server
frame 1	96831	98886
frame 2	103597	106496
frame 3	117878	120784
frame 4	104313	106496

2. Network tradeoff -face recognition software on Raspberry Pi vs using an API over the Internet.

I used Amazon face Rekognition for API and openCV on raspberry Pi

- Amazon face Rekognition API gives more accurate results than opencv face recognition system on the Pi.
- In case of using API, Images are sent to remote server and processed which might cause delay compared to processing the images on Raspberry Pi itself.
- Processing the images at raspberry Pi side software is faster than remote server.

3. The fastest image frame rate system is able to support is 5 frames

IMPLEMENTATION

I used socket programming for sending image frames from Pi to server. Pi captures an image, processes it and then sends to server and goes to wait mode until server asks to send image again. After receiving image from Pi, server sends it to android. On android side, program receives image frame from server and goes into wait mode until server sends asks to receive data.

BOTTLENECK

The Pi Program has to wait until server asks to send data, similarly android has to wait after receiving a file from server until server replies back to receive data. Series of socket communication takes place like this and delay is caused because of waiting time.

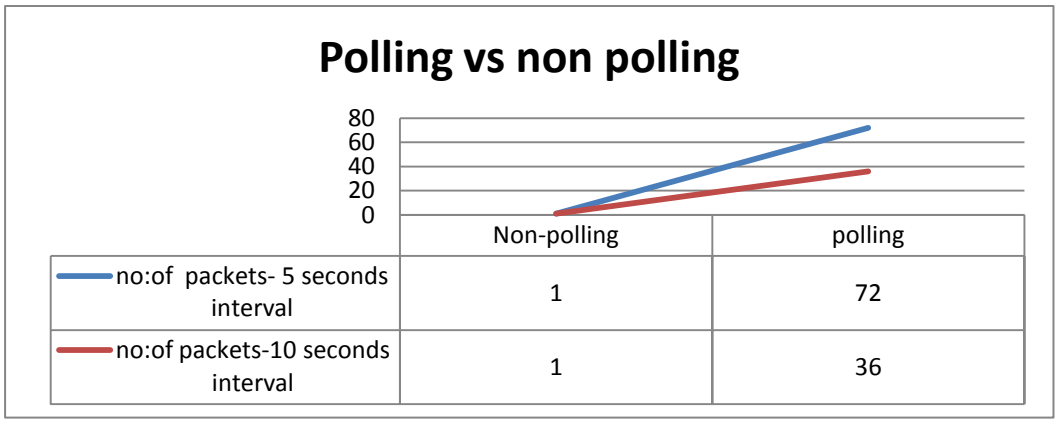
4. Non polling authentication

When raspberry Pi connects to server for sending image frames, instead of closing the connection after completion of sending image stream i maintained that connection without closing. Therefore when server receives authentication status from android it directly sends it to raspberry Pi with the previously established connection.

5. Positive and negative aspects of polling vs non-polling approach

- In Polling approach Pi has to continously poll the server to check whether it has the authentication status file which causes overhead on server. So for example 60 polling requests are made from Pi to server until it gets auth data, server has to reply to 60 requests which causes huge overhead
- Whereas in Non-Polling approach server sends authentication status only once directly to Raspberry pi when it receives status from the android which reduces the overhead on the server.
- The whole point of a non polling connection is that you don't ever have to ping the server for changes. Instead, the client just connects once and then the server can just directly send the client status whenever they are available.

Measurements



New Feature Implementation

Gesture Recognition

Raspberry Pi System authenticates the user with unique hand symbol. User has to show unique hand symbol to camera module which acts as a password for authentication.