TechnoVision

Team:
Tezan Sahu (Team Leader)
Shubham Varnkar
Tushar Agrawal
Shaurya Sarna

ATTENDANCE SYSTEM USING FACE RECOGNITION

Github Repository: https://github.com/tezansahu/ITSP-TechnoVision

Video Link for our Project: https://youtu.be/fqAU3uDpXJg

Project Overview:

For the Institute Technical Summer Project-2018, our team intended to build an Attendance Recording Bot that could use face recognition to mark the attendance of students while they are sitting for the lecture. We created an app for the students so that they get to check their attendance, which would be updated after the bot records their attendance. Another app, for the professor, to control the bot has also been developed by us.

Currently, this bot can work in classrooms where there are a small number of students.

Outline of the Working:

1. Students will have an app wherein they will have to login. We will show the attendance of the student also in that app for all courses that he/she has registered.
2. The bot will be controlled by the professor using an app.

3. The bot will perform regular movements with differential turning mechanism. We will also attach servos to the camera stand on the bot to facilitate turning of the camera using the Bot Controller app.
4. The mobile phone on the bot will send a live streaming of the class to a server using IP Webcam.
5. RPi will collect the streaming video from the server of IP Camera (the app used by us).
6. Using the previous training about the faces of registered students, the bot will recognize the students' faces in the live video. We are implementing this by using either "face_recognition" package, and OpenCV.
7. As soon as a student is recognized, his attendance will be marked in a backend database maintained by us.
    - Data would flow in batches decided upon the number of frames captured, instead of one frame at a time
8. As soon as the attendance is marked, it would reflect in the attendance sheet that the student views in his account in the app.

Materials Used:

- **Raspberry Pi with Case (Rs. 3266):** As a processor for the execution of all the python scripts (for face recognition and bot control). It is using a **16GB SD (Rs. 650)** card to run the Raspbian Stretch OS. Used an **HDMI Cable (Rs. 190)** to boot it for the 1st time.
- **Phone Camera:** To be used as the camera for live streaming of video using IP Webcam application.
- **Chassis, Platform & Phone Stand (Rs. 100):** All made from plywood/laminated plywood.
- **PVC Pipe (Rs. 30):** Approx. 1m long hollow pipe fixed using **L-bends (Rs. 20)** to the chassis to serve as the neck of the bot. At the top of it, the platform for the phone is attached using a **Ball Bearing (Rs. 295)** for smooth rotation.
- **Motor & Wheels (Rs. 900):** 4 100 RPM motors and wheels of 10 cm diameter (2 cm thick)
- **L293D Motor Driver Module (Rs. 164):** Acts as an interface between the Raspberry Pi and the motors, sending signals about the movement to be performed by the bot.
- **Servo Motor (Rs. 450):** Fixed at the top of the PVC pipe, connected to the platform in order to rotate the phone camera at the controller's will.
- **Power Bank (Rs. 1500):** 16,000mAh rated power bank to supply 5V, 2A to the Raspberry Pi for powering it up.

- **Lead Acid Battery (Rs. 530):** 12V lead-acid battery to power the motors/servo motor for wheel and platform movement.
- **Nuts**, **Bolts**, **Jumper Wires**, **M-Seal**, etc. **(Rs. 300)**

The Task of “Face Recognition”:

Face recognition based on the geometric features of a face is probably the most intuitive approach to face recognition. Marker points such position of eyes, ears, nose, etc. are used to build a feature vector. The recognition is performed by calculating the euclidean distance between feature vectors of a probe and reference image. Such a method is robust against changes in illumination by its nature, but has a huge drawback the accurate registration of the marker points is complicated, even with state of the art algorithms.

The problem with the image representation we are given is its high dimensionality. Two-dimensional p x q grayscale images span a m = pq-dimensional vector space, so an image with 100 x 100 pixels lies in a 10,000-dimensional image space already. We can only make a decision if there’s any variance in data, so what we are looking for are the components that account for most of the information. The Principal Component Analysis (PCA) was proposed to turn a set of possibly correlated variables into a smaller set of uncorrelated variables. The idea is, that a high-dimensional dataset is often described by correlated variables and therefore only a few meaningful dimensions account for most of the information.

The PCA method finds the directions with the greatest variance in the data, called principal components (along the “eigenvectors” of the covariance matrix of the dataset). This method is used in the “Eigenfaces Algorithm” for face recognition.

What We Used:

For the task of face recognition in our project, we used the recently developed python module called “**face_recognition**”, using dlib’s state-of-the-art face recognition built with deep learning.

It works by encoding the “known” faces and tries to match them to the encodings of the faces received from the live video stream. It works using two frameworks:

1. HOG (Histogram of Oriented Gradients)
2. CNN (Convolutional Neural Network)

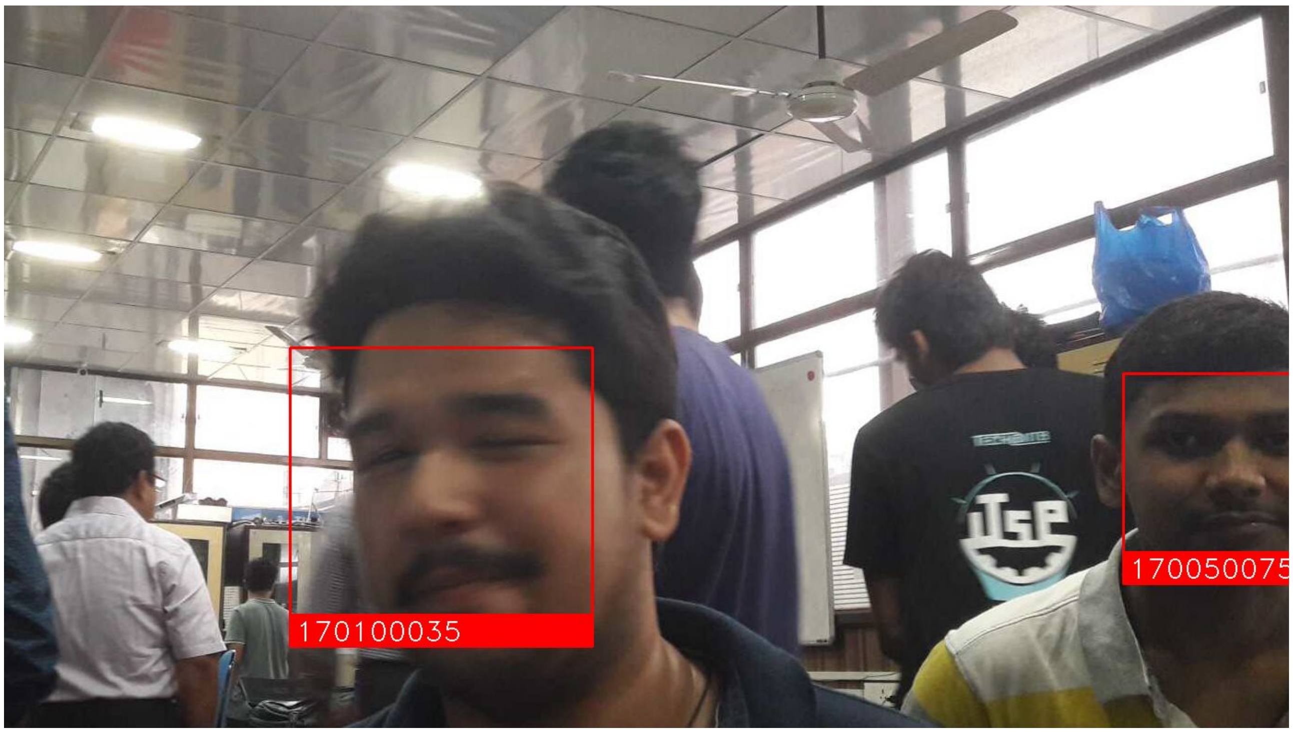
For our project, we have used the CNN implementation with an optimum tolerance value that we got by repeated testing. Although this implementation is a bit slow, the accuracy is ~95%, which is way better than HOG (a meagre ~80%).

Some Snapshots During Testing:

The faces of some students that are stored in our database are shown below. These are encoded and compared to the frames of the video at runtime.

(170100035) (170050075) (170100027) (170100040)

Recognition of faces in a frame of a live video using our face_recognition and openCV packages:



The Backend for Our Attendance System:

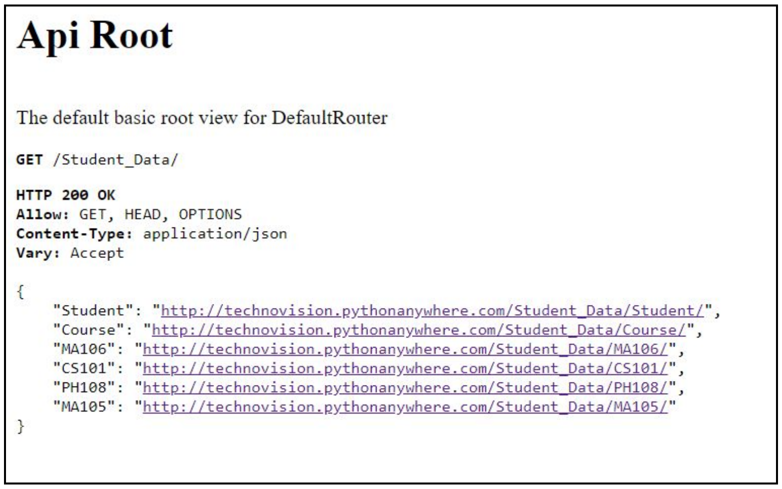
Out of the several options available for backend development, we chose to use “**Django**” for the development and deployed it on “**PythonAnywhere**”. Since we wanted to retrieve and modify data in the backend frequently, we used the RESTful API of Django called “**django-rest-framework**”.

The database consists of the following tables:

- **Students** (Roll No, Name, Branch, Image, Registered Courses)
- **Courses** (Course Code, Course Name, Credits, Number of Lectures, Professor teaching the Course)
- **Individual Courses** (Currently: MA105, MA106, CS101 & PH108)
  - Each course has with it associated all the students who have registered for it along with their attendance details by date.
- **Professors** (ID, Name, Password, Course Taught)

The data is being retrieved using “**urllib**” and “**json**” python libraries in JSON format. After recording of attendance, the attendance for each student is modified using “**requests**” library through PATCH requests.

Some Screenshots of the Deployed Backend:



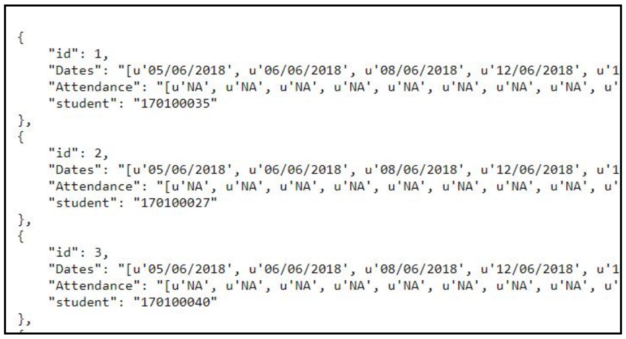
Api Root

The default basic root view for DefaultRouter

```
GET /Student_Data/

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "Student": "http://technovision.pythonanywhere.com/Student_Data/Student/",
    "Course": "http://technovision.pythonanywhere.com/Student_Data/Course/",
    "MA106": "http://technovision.pythonanywhere.com/Student_Data/MA106/",
    "CS101": "http://technovision.pythonanywhere.com/Student_Data/CS101/",
    "PH108": "http://technovision.pythonanywhere.com/Student_Data/PH108/",
    "MA105": "http://technovision.pythonanywhere.com/Student_Data/MA105/"
}
```



```
{
    "id": 1,
    "Dates": "[u'05/06/2018', u'06/06/2018', u'08/06/2018', u'12/06/2018', u'1
    "Attendance": "[u'NA', u'NA', u'NA', u'NA', u'NA', u'NA', u'NA', u'NA', u'
    "student": "170100035"
},
{
    "id": 2,
    "Dates": "[u'05/06/2018', u'06/06/2018', u'08/06/2018', u'12/06/2018', u'1
    "Attendance": "[u'NA', u'NA', u'NA', u'NA', u'NA', u'NA', u'NA', u'NA', u'
    "student": "170100027"
},
{
    "id": 3,
    "Dates": "[u'05/06/2018', u'06/06/2018', u'08/06/2018', u'12/06/2018', u'1
    "Attendance": "[u'NA', u'NA', u'NA', u'NA', u'NA', u'NA', u'NA', u'NA', u'
    "student": "170100040"
},
```

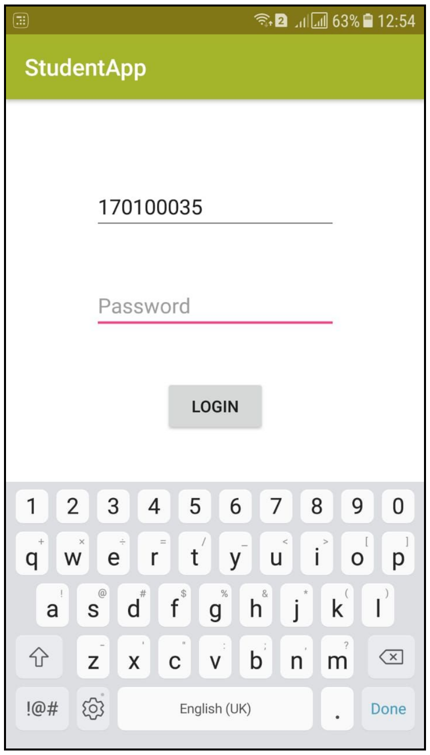
The Android App for Students:

This is the app that the students would use to check their attendance for registered courses. It comprises of the Activities:

- **Login (Main) Activity** (for students to login and see the courses they have registered for)
- **Course Activity** (displays details of the student and a list of courses that the student has registered for)
- **Attendance Activity** (shows the attendance of the student by date for the course that the student selected in the previous activity)

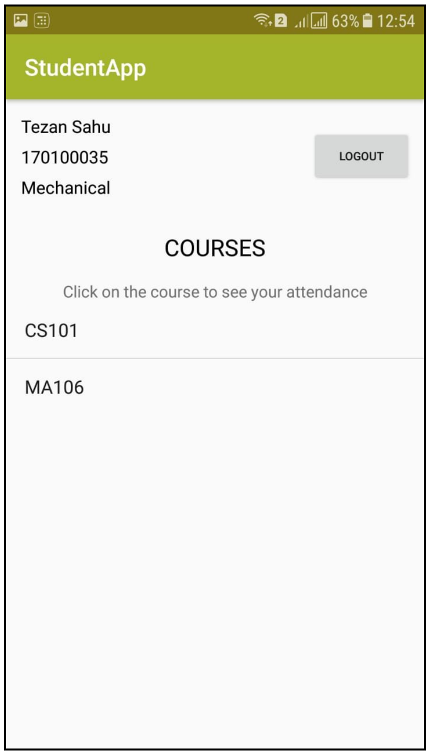
The app connect to the database to fetch required results using the "**Ion**" library in Android Studio and makes use of JSON objects and arrays to efficiently read and display the data from the API.

Some Screenshots of the App for Students:



170100035

Password



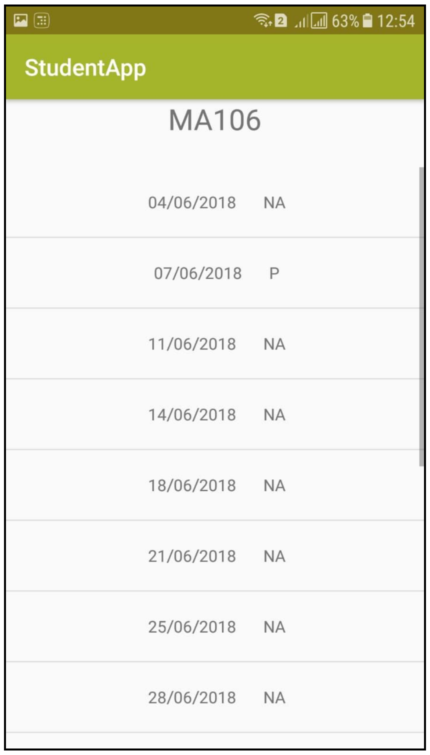
Tezan Sahu
170100035
Mechanical

COURSES

Click on the course to see your attendance

CS101

MA106



MA106

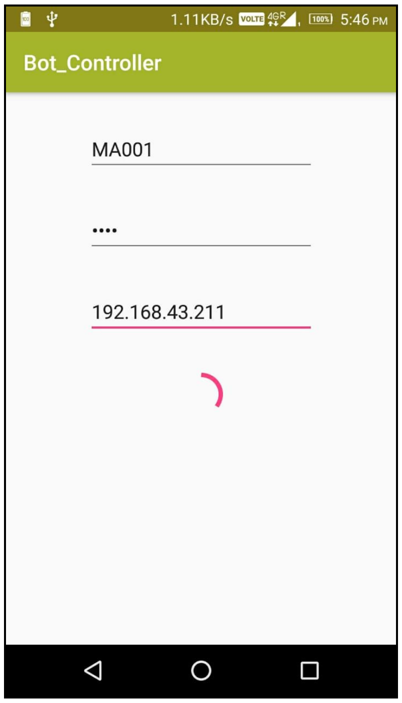
04/06/2018	NA
07/06/2018	P
11/06/2018	NA
14/06/2018	NA
18/06/2018	NA
21/06/2018	NA
25/06/2018	NA
28/06/2018	NA

The Android App for Bot Control by Professor:

This app would be used by the professor to control the bot (via bluetooth) as it moves around the class, recording live video and recognizes the faces of registered students. It comprises of the following activities:

- **Login Activity** (For the professor to log in to control the bot. The IP address entered is the same as that shown on the IP webcam app used on the bot during video recording. This is sent to the Raspberry Pi via bluetooth to fetch video from the IP Webcam Server.)
- **Connect Activity** (Shows a list of paired bluetooth devices to connect. The Raspberry Pi must be paired with this device for it to receive data. This contains 3 threads: *Connect*, *Connected* and *Accept* threads. It also contains a “handler” to communicate between the threads.)
- **Remote Control Activity** (Contains to control the motion of the bot and the camera)

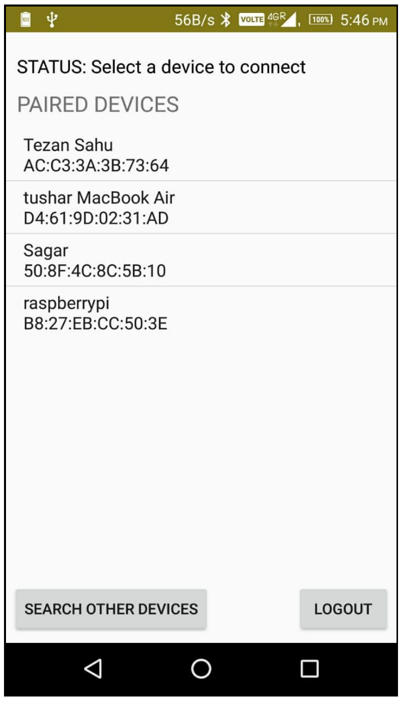
Some Screenshots of the App for Bot Control:



MA001

••••

192.168.43.211



STATUS: Select a device to connect

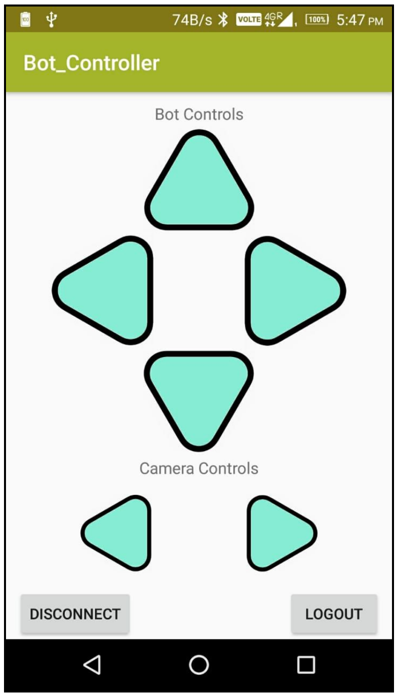
PAIRED DEVICES

Tezan Sahu
AC:C3:3A:3B:73:64

tushar MacBook Air
D4:61:9D:02:31:AD

Sagar
50:8F:4C:8C:5B:10

raspberrypi
B8:27:EB:CC:50:3E



The Python Codes:

The python codes that implement all the face recognition and bluetooth client receiving logics are being run on the Raspberry Pi. We have split these into two separate scripts that would run parallely on the boot up of the Raspberry Pi.

Code for Face Recognition and Attendance Marking:

This piece of code uses a “**technovision_utils.py**” script (created by us) that contains functions used to:

- Check if the file “ip_address.txt” has the IP Address & Course; if it does, start executing further code.
- Fetch attendance data and images of the students for the allotted course from the backend.
- Connect to the IP Webcam server that is streaming live video of the class
- Check the video frame-by frame for available faces and mark the attendance of students.
- After every 30 frames, mark the attendance of students found till that frame “Present (P)” (using “PATCH” requests to the backend)
- On logout of the professor, exit condition written into “exit_check.txt” file; after checking its contents, mark the attendance of all remaining students as “Absent(A)” and then end the script.

This functions library uses the following python modules:

- face_recognition
- openCV
- scikit-image
- json
- urllib
- requests

Code for Bot Control using Bluetooth:

This piece of code is used to receive data and commands from the Bot Controller App via bluetooth through an RFCOMM channel and trigger signals to the corresponding GPIO pins on the Raspberry Pi to perform forward, backward, turn-left and turn-right for the bot as well as servo-left and servo-right movements for the phone camera. This code uses the following modules:

- RPi.GPIO
- Bluetooth

Interaction between the two main scripts is ensured using text files “**ip_address.txt**” and “**exit_check.txt**” which are used to trigger the running of “test_recog.py” and stop it respectively.

Project Timeline:

Week 1 (28th May - 3rd June):

- Bot design finalized with all minor specifications in mind
- Procured all the materials to be used for the project
- Raspberry Pi booted on the laptop using SSH
- All necessary python packages and other dependencies installed on RPi
- Drilling and laser cutting on the chassis and platform complete

Week 2 (4th June - 10th June):

- Proceeded with the remaining mechanical development of the bot
- Django Backend development
- Framework of RESTful API ready
- Data for testing purposes entered
- Final Brainstorming on the data flow and integration of the bot, app and backend
- Started building the App for Students to use (Layouts of all activities ready)
- Basic python script to recognize faces from a live streaming video ready.

Week 3 (11th June - 17th June):

- Started working on the Electrical aspects of the bot
- Django backend development completed
- Deployed our Django backend on “pythonanywhere.com”
- Development of the Android App for Students completed
- Layouts ready for App for Bot Control

During the 3rd Week, our SD Card got corrupted. The new SD card arrived only on Monday (18th June). Hence, we had to reinstall all the stuff and start over again.

Week 4 (18th June - 24th June)

- Installed all the necessary Python packages onto the SD Card again.
- Worked on the bluetooth connectivity and testing of the App for Bot Control.
- Python codes for retrieving, manipulating and sending “patch” requests to the backend ready.

During the 4th Week, another SD Card got corrupted. Luckily, we had a backup SD Card and hence did not lose much time in getting a new SD Card up and running.

Week 5 (25th June - 1st July)

- Designed the stands for placing the mobile phone on the platform.
- Tweaking of the backend to update some fields and add super-users.
- Testing and debugging of the Apps for Bot Control and Attendance Viewing.
- Python script to control the bot using bluetooth signals from the Android App almost ready.
- Almost completed the electrical circuitry of the bot.

Week 6 (2nd July - 8th July)

- Final tweaking of codes for IP address receiving and servo motor controls
- Running both scripts simultaneously on boot of RPi and integrate between them
- Final testing and debugging of the entire system

Challenges Faced:

- The PVC pipe would shake a lot on slight perturbation
  - **Solution:** We drilled 2 holes towards the top of the pipe and connected 2 plastic ropes through it to the corners of the bot as suspensions for additional stability.
- SD card corrupted twice during testing of codes on the bot
  - **Solution:** After searching hard, we found that the actual issue was of undervoltage. For the later part of the testing, we powered the RPi using our laptop (did not cause undervoltage).

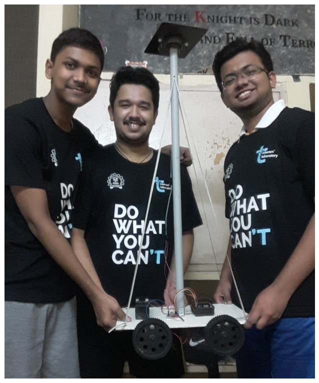
Possible Limitations:

Some of the possible limitations of our project include:

- The **time lag** during recognition tasks is a serious issue. The current time lag is due to extra time required to import packages, fetch data from the internet, process frames using a Neural Network and also the constraint of processor capacity.
- This time lag will increase with increase in number of students, so **scalability** is an issue
- With around 95% accuracy, the bot may **recognize some people incorrectly** or fail to recognize people at some points of time.
- The bot can be **tricked by photographs of people**. So, we may implement “liveness detection” in future.
- The **maximum distance** up to which faces can be recognized is around 2-3 m (**quite less**). So, the bot needs to move more in order to capture the faces of everyone in class.

- The passwords stored in our database are not encrypted. So **security** is an issue.
- Our bot can move only in classrooms with **flat floors**, as opposed to the usual stepped seating arrangement.

Some Snapshots During the Project:



Reference Links:

1. To use IP Webcam for streaming live video from mobile phone to Server and retrieve it using openCV on RPi:
    - https://thecodacus.com/ip-webcam-opencv-wireless-camera/#.WxWcYUiFNPY

2. Documentation of "face_recognition" python package and its usage:
    - https://pypi.org/project/face_recognition/
    - http://face-recognition.readthedocs.io/en/latest/face_recognition.html

3. Django Tutorials for RESTful API creation:
    - https://www.youtube.com/playlist?list=PLXmMXHVSvS-DQfOsQdXkzEZyD0Vei7PKf
    - http://django-mysql.readthedocs.io/en/latest/model_fields/list_fields.html

4. Android App Development course that we followed:
    - https://www.youtube.com/watch?v=v_OQHLjmfEM&list=PLYKXDWkoIMUH088iBEr_B2WPfbPwtaG-V

5. Servo Motors Controlling through python codes on Raspberry Pi:
    - https://rpi.science.uoit.ca/lab/servo/

6. Bluetooth Connection and Data Transfer between Android App and Raspberry Pi:
    - https://blog.iamlevi.net/2017/05/control-raspberry-pi-android-bluetooth/
    - https://github.com/elec-club-iitb/xlr8-remote-control

7. Running python scripts as services on the boot of Raspberry Pi:
    - https://www.raspberrypi-spy.co.uk/2015/10/how-to-autorun-a-python-script-on-boot-using-systemd/