Smart Lab
CS321 Project

By:
Aadi Aarya Chandra – 200101002
Vishal Bulchandani – 200101108
Ravi Kumar - 200101089
Pranav Nair - 200101082

Project Report

Our project consists of the following implemented sub-systems –

1. Flutter Application
2. Face Recognition
3. Intruder Detection
4. Environment Sensing

We describe these sections one by one below.

A common Gmail account is made for the whole project. This account can be used (in future) for admin controls, as well as email notifications for intruder, environment sensing data, entry logs, etc. The credentials for the account are –
Code repository: https://github.com/vi-bulchandani/smart-lab
Email – smartlab.roboticsiitg@gmail.com
Password – varp2023

Flutter Application

Directory structure

1. src/face_recognition_app – This contains the source code of the application, can be seen in VSCode or Android Studio and changed as you need. Make sure to run install.bat/install.sh file before making any changes or doing a debug run and making a new apk.
2. app-release-facenet.apk – Deployed version (**only for Android**) of the app, using the Facenet TFLite model for face detection and recognition
3. app-release-facenet512.apk – Deployed version (**only for Android**) of the app, using the Facenet512 TFLite model for face detection and recognition

Please go through the README file in the src directory to learn more about the functions of the various files and make changes as you wish.

Three models are supported in the code for face recognition – Facenet, Facenet512 and MobileFacenet. To switch to one of these, you may make changes to the code by just commenting out the part for the other models in the src/face_recognition.dart file. Note that you will have to make changes to the face recognition python script (described in next section) as well. We have already provided apk's for Facenet and Facenet512, so no changes in code on the app side are required for them.

Features

1. Google account authentication – Google account based authentication is allowed.
2. Linked to Firebase – A firebase account is created for handling the backend of the app. This is created with the Gmail account mentioned earlier. The firebase account has two services attached to it, viz. Firebase Auth (for authentication) and Cloud Firestore (for storing face embedding and other user data). To handle face recognition later, one needs to have a service account created for the app. The private key of this account is provided as a json file (see face recognition section).

3. Upon authentication/registration, the app checks if you are an authorized user. The list of authorized users is manually maintained in the Cloud Firestore. To modify this list, go to the authorisedUsers collection in Firestore and change. Please see the documentation of Firestore to learn more.
4. If you are an authorized user, the app first asks for your name, contact no. and one photo for face recognition, which can be taken from your phone.
5. The home screen of the app has 3 tabs –
    a. Update photo – to update your profile pic used for face recognition. Face embeddings are stored in Firestore in the metadata collection in the faces document, along with email ID used for unique identification.
    b. Environment Sensing data – Live data from the Thingspeak server is shown here. Also, the controls for maximum and minimum AC temperature threshold are given, using which the AC Flap can be controlled.
    c. Entry logs – This shows all the entries made into the lab with their timestamps, including any intruder, if any. This list is stored in Firestore in the metadata collection in the environment document. This tab also shows the total person count in the lab, which is also taken from Firestore database where it is updated regularly.

Face recognition

Note: Face recognition works in synergy with intruder detection, so some of the hardware for the two sub-systems is common.

Directory Structure

1. run_model.py – contains the script to run on the laptop for face recognition
2. Other files are various tflite models which can be used for face recognition. To switch between models, change one line in the script (see the comments in the script for this)

Hardware required

1. USB camera
2. Arduino
3. Green LED (to indicate entry allowed), red LED (to indicate entry not allowed)
4. LED Bar (to act as camera flash in dark environment) **(not currently in use)**
5. Push button (to kickstart face recognition) **(not currently in use)**
6. Line sensors, buzzer (from intruder detection)

Features

1. First the Arduino along with the other hardware interfaced to it, is connected to the laptop. The code (see next section) is uploaded on it if not already done. The pin numbers for the various components are mentioned at the top of the Arduino code.
2. Then the python script run_model.py is run. It downloads the latest face embeddings from Firestore first (currently only at the start of the script, this can be made to run every few minutes).

NOTE: A service account has to be created on Google Cloud Console from the smartlab Gmail account, and its private key has to be provided as a json file within this script. One has been already created by us and its json file has been shared. This json file has to be in the 'Face Recognition' folder, in the same directory as the run_model.py script.

3. Then, the camera feed starts. Upon pushing the button (currently the ‘a’ key on the keyboard of the laptop is used as a substitute), if there is any face in front of the camera, it is captured and after a few seconds, it is detected as either Unknown, or one of the existing users. If unknown, the red LED glows, else the glows. If one of the existing users is detected, their email is shown in the terminal and the green LED glows.
   The buzzer is supposed to be turned off at this point, and the entry logs have to be sent to the Firestore database along with the updated person count. These two things are not yet implemented.
4. The LED bar can be turned on when ambient light is low. This is also not yet implemented, but simple changes can be made to the Arduino code by checking code of Grove LED Bar.

Intruder Detection

Hardware required

1. Line sensors – 4 in total - Two are installed right outside the door, two inside. They are installed right above the door and the upper surface of the door is painted white/white paper is pasted on it for better detection by sensors.
2. Arduino – This is the same Arduino used in the last section. The code for it is provided in intruder.ino file.
3. Buzzer, push button, LED Bar, Green and Red LEDs

Features

1. It regularly gets updates from the laptop/raspberry pi where the face recognition model is running about whether a recognized face is detected or not using a variable 'recognised'.
2. Line sensor data is regularly read. Since the door in the robotics lab is of push type, when there is an exit, the line sensors installed outside would signal first. This would only decrement the person count. On entry, the sensors inside would signal first. This increments the person count and if the person is recognized, the buzzer doesn't buzz, else it buzzes. (Currently the buzzer always buzzes on entry, this bug is not removed yet).
3. At this point, entry logs and update person count are supposed to be forwarded to raspberry pi/laptop, which would send it to Firestore. This part is not implemented yet.

Environment Sensing

Hardware required

1. NodeMCU
2. Temperature-Humidity Sensor (DHT11)
3. Carbon Monoxide Sensor (MQ7)
4. LED display
5. Stepper Motor with driver
6. Two circular gears – one attached to motor, one is a free wheel.
7. Two linear gears pasted end to end on a rectangular piece of cardboard of suitable size to act as AC vent flap.

The connections are described in the env.ino file.

The AC flap is designed to be a rectangular cardboard piece, with two linear gears connected end to end to provide an assembly that can translate horizontally when a circular gear moves below the linear gears. The circular gear attached to motor rotates with the motor, causing the linear gear to translate, thus moving the AC flap.

Note: A strong WiFi connection is also required for NodeMCU, and its ssid and password have to be mentioned in the NodeMCU code.
WiFi OTA (Over-The-Air) capability is also implemented in the code, which allows you to update the uploaded code on the NodeMCU wirelessly (except the first time uploading, which has to be done via Serial). For this, the machine uploading the code and NodeMCU have to be on the same WiFi connection.

Features

1. First the NodeMCU connects to the Internet using the WiFi credentials provided. Then it calibrates the sensors and the motor.
   Issue: The NodeMCU is probably not powerful enough to handle both the WiFi connection and the motor running, hence this section may have to be

transferred to a Raspberry Pi in the future or some other means of reducing required motor power may be explored.

2. It regularly reads (every minute) the sensors data and uploads them to the ThingSpeak channel. The read and write API keys and channel ID for Thingspeak channel that we have created are –

   Write API Key - PX70HD36BJ7MACXK

   Read API Key - 17J3EX7IDD6YMO9Y

   Write channel ID (also read channel ID) - 2098172.

   If you create your own channel, make sure to change these.

   These have to be written in the code in the first few lines before running the code. In addition, the rotation of the motor can be adjusted as per preference by updating the 'adjustment' variable.

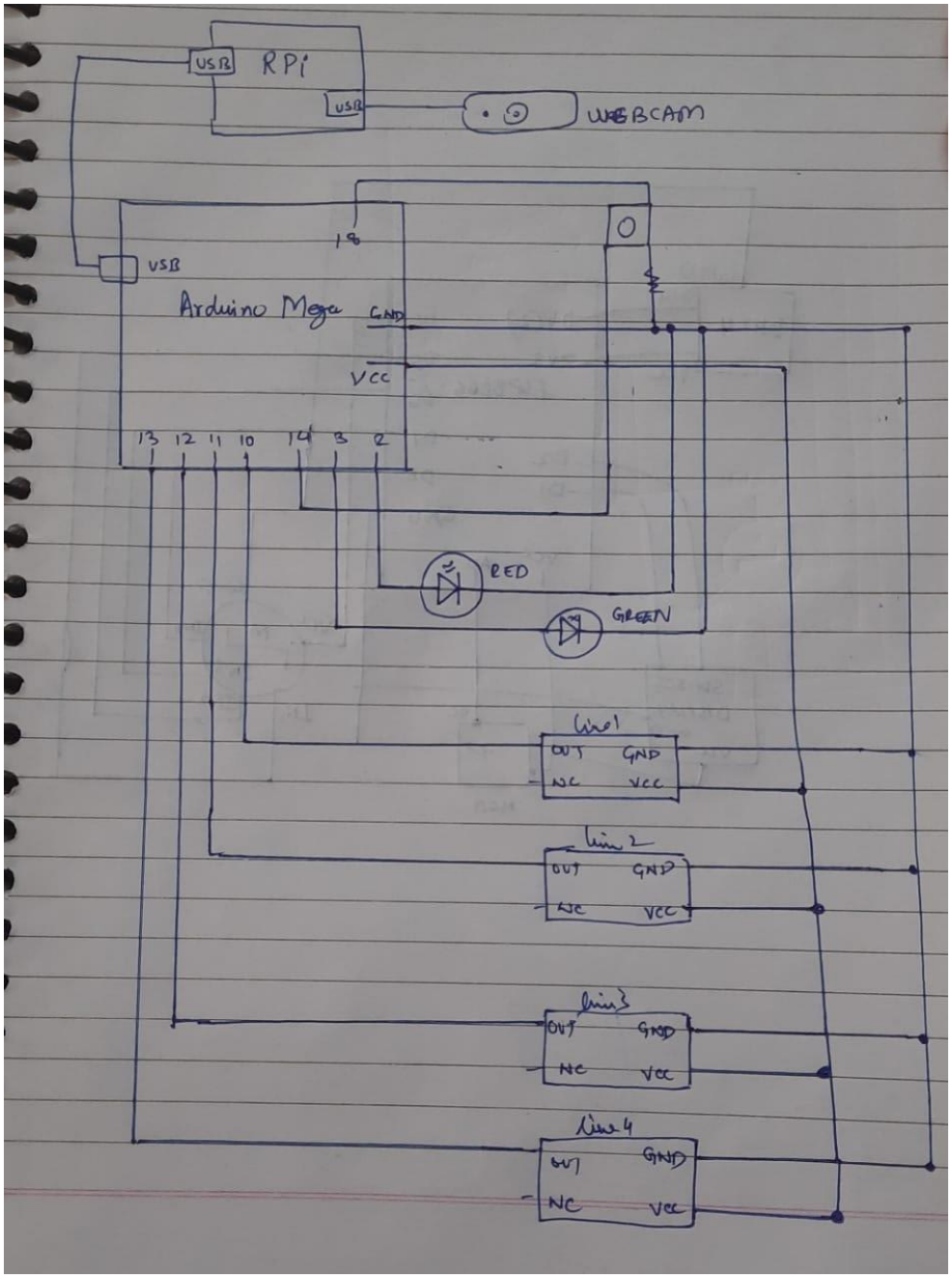
3. The set maximum and minimum temperature in the app, which were sent to this Thingspeak Channel, are also read every minute. When the ambient temperature drops below the set minimum temperature, the AC flap closes. When it goes above the set maximum temperature, it opens.

   Note: The initial position of the flap has to be set properly. We recommend using the motor-motion-test.ino file to check how the motor moves the flap and then set the required parameters in the env.ino file accordingly.

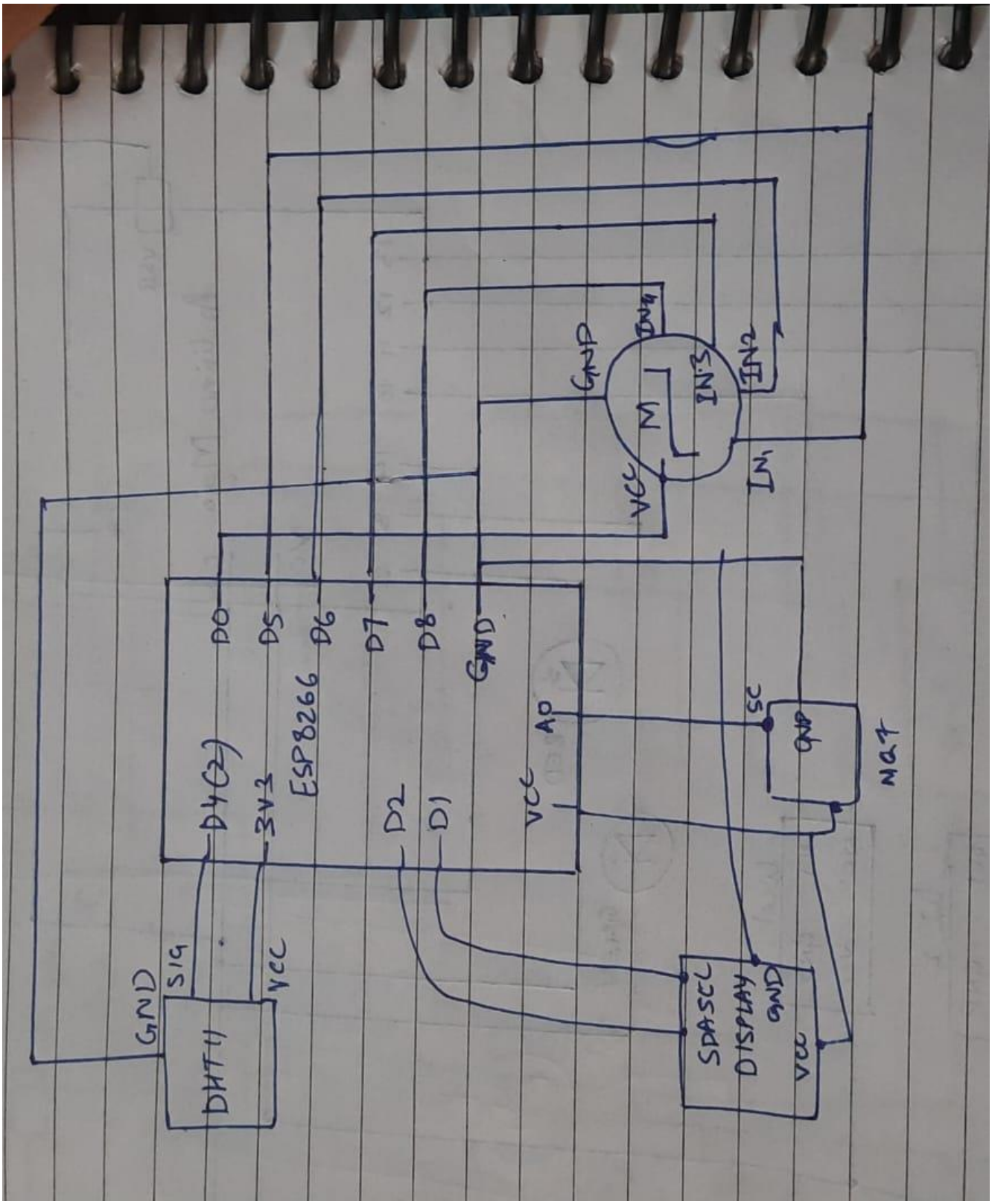
4. The flap state and sensor data are written to the Thingspeak channel.

Circuit Diagrams

For Arduino/Rpi setup



For NodeMCU setup:



Challenges face…

and future improvements

1. The app may collect user employee ID/roll number and other data and have profile-based system.
2. Special admin controls may be implemented for a designated admin.
3. Email notifications for alerts may be sent.
4. More photos may be taken for better face recognition.
5. Currently running the TFLite models on the Pi is difficult and didn't work for us. The Pi 2 doesn't have enough power to run the Arduino used for face recognition. Pi 3/4 may be able to run the models and work well.
6. Interfacing the face recognition Arduino with pi has quite a lot of bugs, eg the 'recognised' variable doesn't get updated properly, Arduino is not able to send entry/exit data to pi, person count is not updated properly, etc. These are essential improvements required for the project to work; using proper interrupts may help solve this problem.
7. Connecting all components properly was a challenge as even one connection getting loose is difficult to find and debug. A permanent solution is soldering.
8. The face recognition models do not give correct prediction of person – this may have to do with camera quality and evaluation metrics used in our code. The face_recognition library of python solves this problem, but it creates an encodings.pickle file which has to be somehow stored and updated frequently on the server. This would work but is time-consuming.
9. A digital light sensor may be connected to this Arduino to check ambient light and turn the LED bar on/off. The LED bar could also act like a flash for the camera.
10. The NodeMCU in environment sensing may have to be replaced by a Pi 3 due to power issues.
11. The AC vent flap gets stuck sometimes during motor rotation – this is mainly due to the motor not being attached firmly to the wall and to the gear. A permanent solution is nailing the motor and free wheel to the wall.

12. A simple push button/sensor may be put in place at the boundaries of flap movements to act as a check for the actual flap state (open/closed). This is needed to make the system a closed loop one.