**# Raspberry Pi 3B+ Facial Recognition and QR code scan embedded with larduino which is used for line following.**

Hardware and software tools

**4.1 Design Tools**

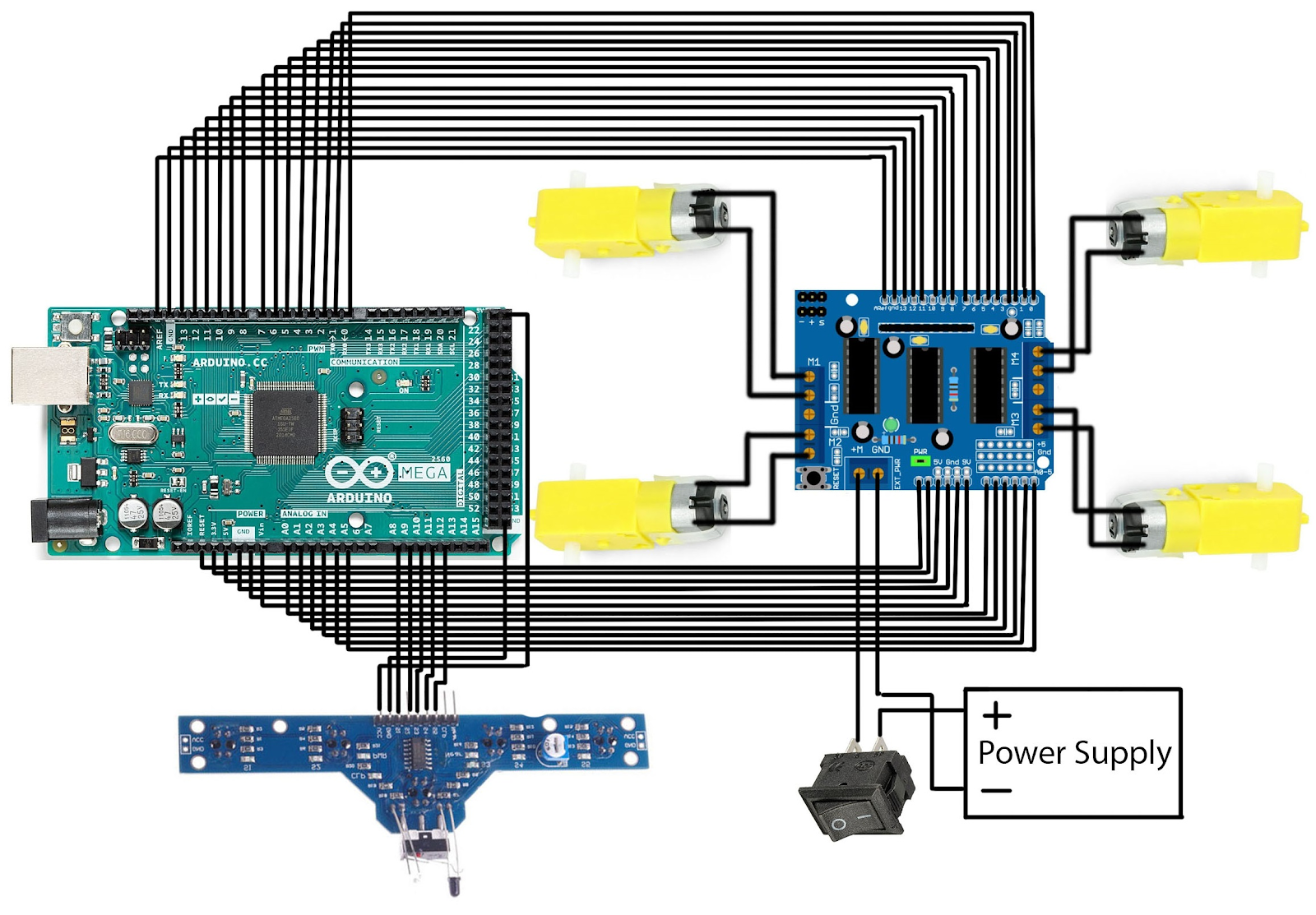
To implement the project, we have used the following design tools. The design tools include the following list.

* **Arduino Mega 2560 Microcontroller**
* **L293D Motor Driver Shield**
* **5 array infrared sensor (IR)**
* **Raspberry PI 3 B+**
* **PI camera:**
* **SG-90 servo motor**
* **DC-DC buck converter (LM2596 with display**
* **Power supply**
* **Arduino IDE**
* **Thony, Python IDE**
* **Linux Terminal in Raspberry PI**

Initially, we solder the wires to four dc motors and then connect them to M1, M2, M3 and M4 pins of the L293D motor driver board. Then, place the L293D motor driver over the Arduino mega 2560 board. Then, the connection will be as below automatically –

|  |  |
| --- | --- |
| Arduino pin | L293D pin |
| AREF | AREF |
| GND | GND |
| 13-0 | 13-0 |
| RESET | RESET |
| 3.3v | 3.3v |
| 5v | 5v |
| GND | GND |
| Vin | Vin |
| A0-A7 | A0-A7 |

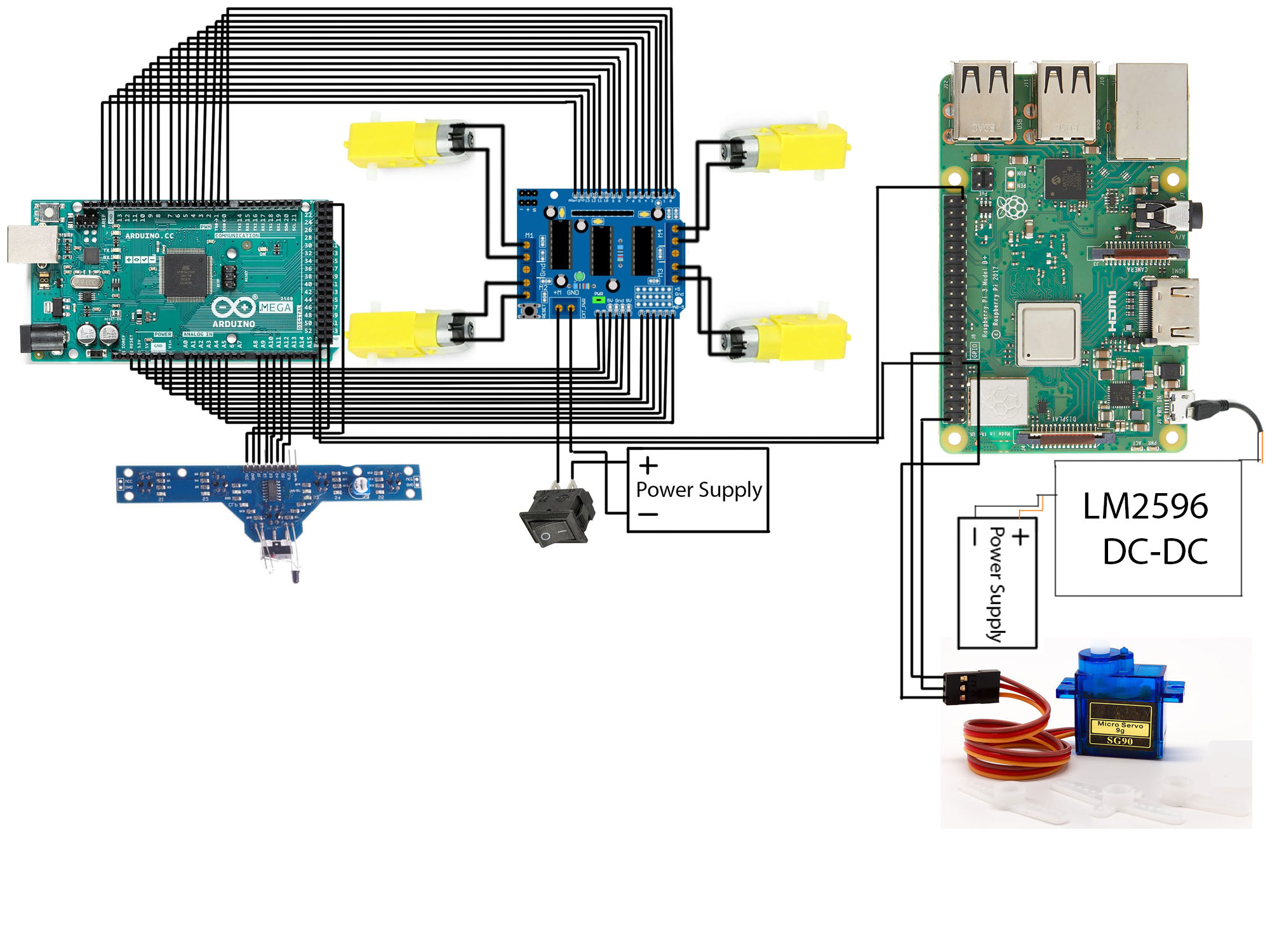
|  |  |
| --- | --- |
| Arduino Pin | 5 Array IR sensor |
| A8 | S1 |
| A9 | S2 |
| A10 | S3 |
| A11 | S4 |
| A12 | S5 |
| 5v | Vcc |
| GND | GND |



**Fig: Full line following robot setup**

**4.3.2 Assembling procedure for Raspberry PI with a camera and object delivery box**

In this assembling procedure, we attach the camera’s flat cable to the raspberry PI first. Secondly, we connected the servo motor to the raspberry PI and put this servo motor inside the box. After that, this servo motor is responsible for opening the gate of the box.

  
  
 **Fig: Object delivery robot with all components**

Implementation and Performance analysis

**5.1 Experimental setup**

We did our experimental setup in several ways. Firstly, we have done our line following part. Secondly, we set up our raspberry pi camera with a raspberry PI and it can detect a face. Here, we took a minimum of 10 pictures of each person for better accuracy and train those images by using the OpenCV built-in function. Thirdly, we scan different QR codes and analyze the timing response to scan them. The next part is, to set up the servo motor and make sure there’s no jitter issue happening. We set up the minimum pulse width, maximum pulse width values in servo motor configuration. And in the final step, we combine Arduino mega board with raspberry PI by connecting PI’s ground pin to Arduino ground pin and Arduino’s analog pin to GPIO18 (12th pin)

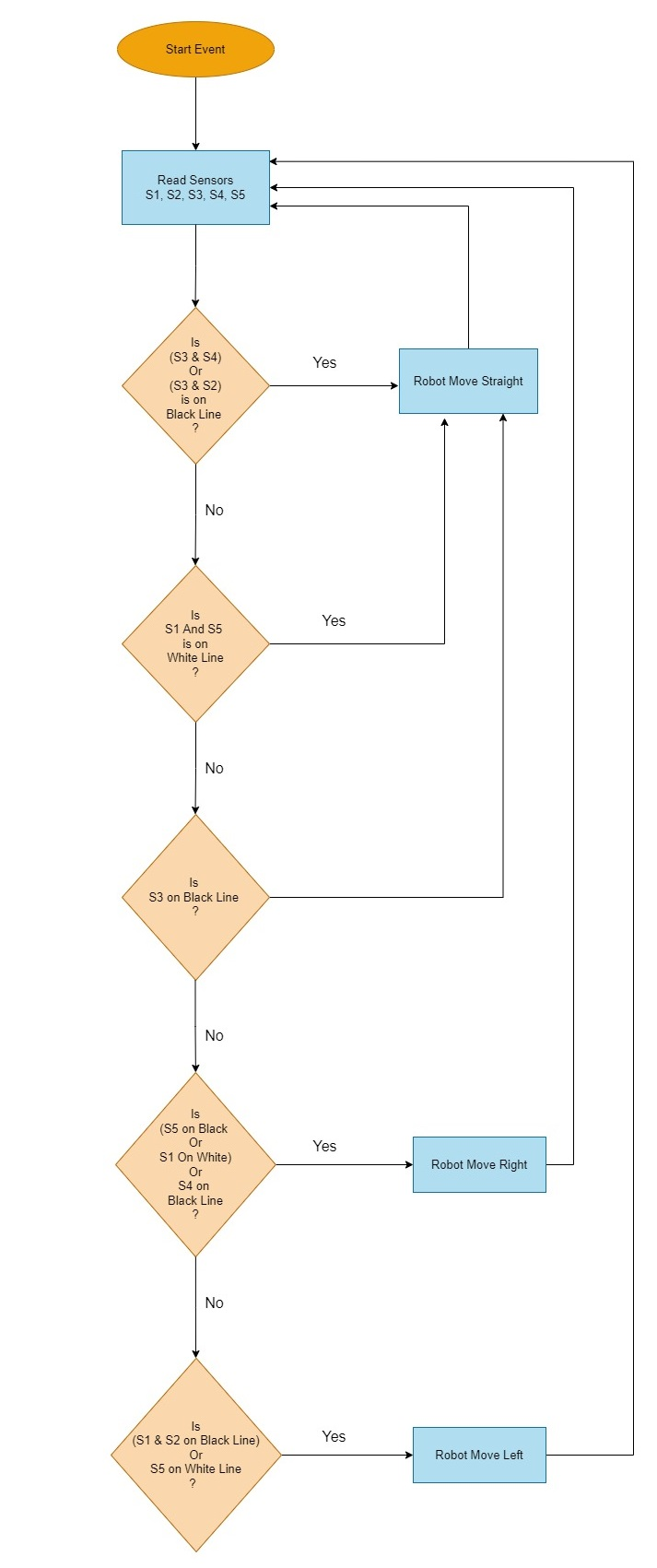
**5.2.1 Line following Part:**

We have installed the “AFmotor.h” library for the L293D motor driver shield. We call the constructor “AF\_DCMotor” so that we can select front, back, left, right motors and customize as needed. AF\_DCMotor also has a setSpeed() function. We set the speed of the motor 90, so that LFR can move slowly and detect authenticated person’s face. We put the value of the serial monitor to 9600 baud rate.

In the IR sensor portion, we took the analog value by placing the sensors on a white and black line. Then, we analyzed those values and wrote the code by following the flowchart.

**5.2.2 Flowchart for line following**:

Initially, 5 array IR sensors will take the reading of all sensors. If the sensor analog value is less than 500, it will count as a black line. If sensor 3 and sensor 4 read value less than 500; or sensor 2 and sensor 3 read value less than 500, then, it will go straight. If sensor 1 and senor 5 read value greater than 500, LFR goes to the straight line. If the value of the middle sensor means sensor 3’s reading value is less than 500, it will go straight. If the right side sensors see a black line, then LFR will go right. And finally, if the left sensors see the black line and the right sensors see the white line, LFR will go left.



**Fig: Line following robot flowchart**

**5.3 Face recognition with Raspberry PI**

In this section, we will be able to configure a face recognition system and recognize an authenticated person’s face. That’s why we need to configure OpenCV.

**5.3.1 OpenCV configuration**

**1**. Initially, we updated, upgraded our PI OS by putting the following commands

**sudo apt-get update**

**sudo apt-get upgrade**

**2**. Then, we have to install picamera with array so that it pulls numpy as a dependency.

**pip install picamera[array]**

**3**. This is a script to make putting together all the build flags when compiling/linking a lot easier

**sudo apt install cmake build-essential pkg-config git**

**4.** There are several packages. All of those are computer imaging applications with different extensions

**sudo apt install libjpeg-dev libtiff-dev libjasper-dev libpng-dev libwebp-dev libopenexr-dev**

**sudo apt install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev libxvidcore-dev libx264-dev libdc1394-22-dev libgstreamer-plugins-base1.0-dev libgstreamer1.0-dev**

**sudo apt install libgtk-3-dev libqtgui4 libqtwebkit4 libqt4-test python3-pyqt5**

**sudo apt install libatlas-base-dev liblapacke-dev gfortran**

**sudo apt install libhdf5-dev libhdf5-103**

**5.**

It will install pip and numpy of python3.

**sudo apt install python3-dev python3-pip python3-numpy**

**6.** Expand the swap file before running the next commands. They need a larger swap file system to execute. We set the value from 100 to 2048.

**sudo nano /etc/dphys-swapfile**

**7. sudo systemctl restart dphys-swapfile**

After changing the value, we must have to restart it to change the effect to the terminal.

**8.** OpenCV is an open-source computer vision library that can be useful for object detection and analyzing. We have cloned this to the “pi” folder

**git clone https://github.com/opencv/opencv.git**

**9.** Cloning the OPenCV contrib repository.

**git clone https://github.com/opencv/opencv\_contrib.git**

**10.** Created a directory named opencv in “pi” folder

**mkdir ~/opencv/build**

**11.** Change the directory to build folder

**cd ~/opencv/build**

**12.** OpenCV uses CMake build configuration tool. Specifies the build type on single-configuration generators. Set up the OpenCV build with CMake.

**cmake -D CMAKE\_BUILD\_TYPE=RELEASE \**

**-D CMAKE\_INSTALL\_PREFIX=/usr/local \**

**-D OPENCV\_EXTRA\_MODULES\_PATH=~/opencv\_contrib/modules \**

**-D ENABLE\_NEON=ON \**

**-D ENABLE\_VFPV3=ON \**

**-D BUILD\_TESTS=OFF \**

**-D INSTALL\_PYTHON\_EXAMPLES=OFF \**

**-D OPENCV\_ENABLE\_NONFREE=ON \**

**-D CMAKE\_SHARED\_LINKER\_FLAGS=-latomic \**

**-D BUILD\_EXAMPLES=OFF ..**

**13.** Start the compilation process by putting this command. It took more than 10 hours to compile all of those necessary packages and build tools.

**make -j$(nproc)**

**14.** This command will install all those packages.

**sudo make install**

**15**. ldconfig creates links and cache to the recent shared libraries found in the directories specified on the command line. It also checks the header and filenames of the libraries it encounters when determining which versions should have their links updated

**sudo ldconfig**

**16.** pip is a package management system that is written in python and used to install and manage software packages. The cache is useless here. So we are disabling it. We are installing face recognition and it recognizes and manipulate faces from python or from the command line with the world’s simplest face recognition library

**pip install face-recognition --no-cache-dir**

**17.** There has some functions in imutils. And we use these functions to make basic image processing functions. For example, translation, rotation, reshaping, skeletonization and showing Matplotlib images easier with python 2 and 3 series. We have both python 2.7 and 3.7 here.

**pip install imutils**

**18.** We changed the value of the swap file before. Now, we set it to 100 as it was initially.

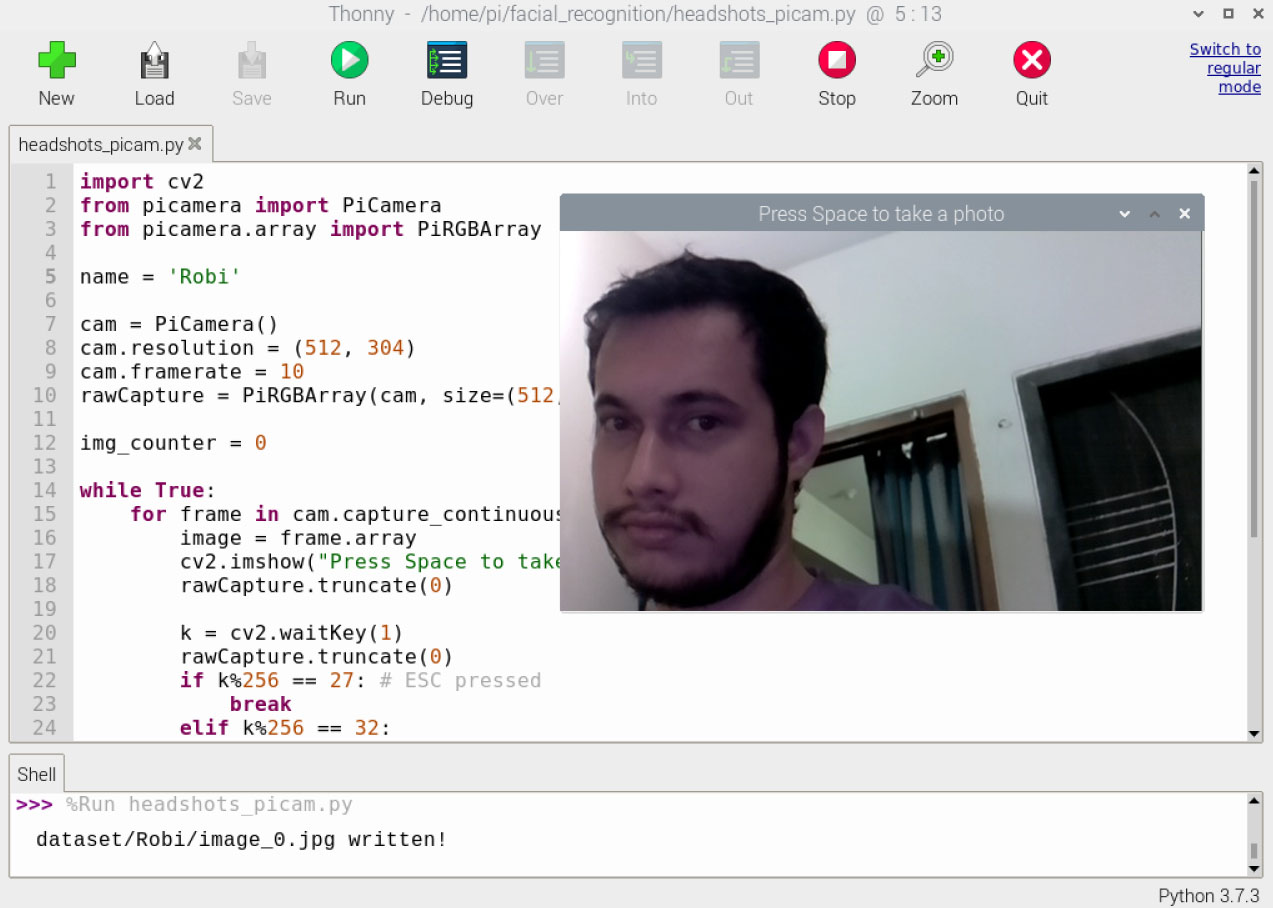
**sudo nano /etc/dphys-swapfile**

**19.** Create facial\_recognition folder and create dataset folder inside this and create other necessary files.

**mkdir facial\_recognition**

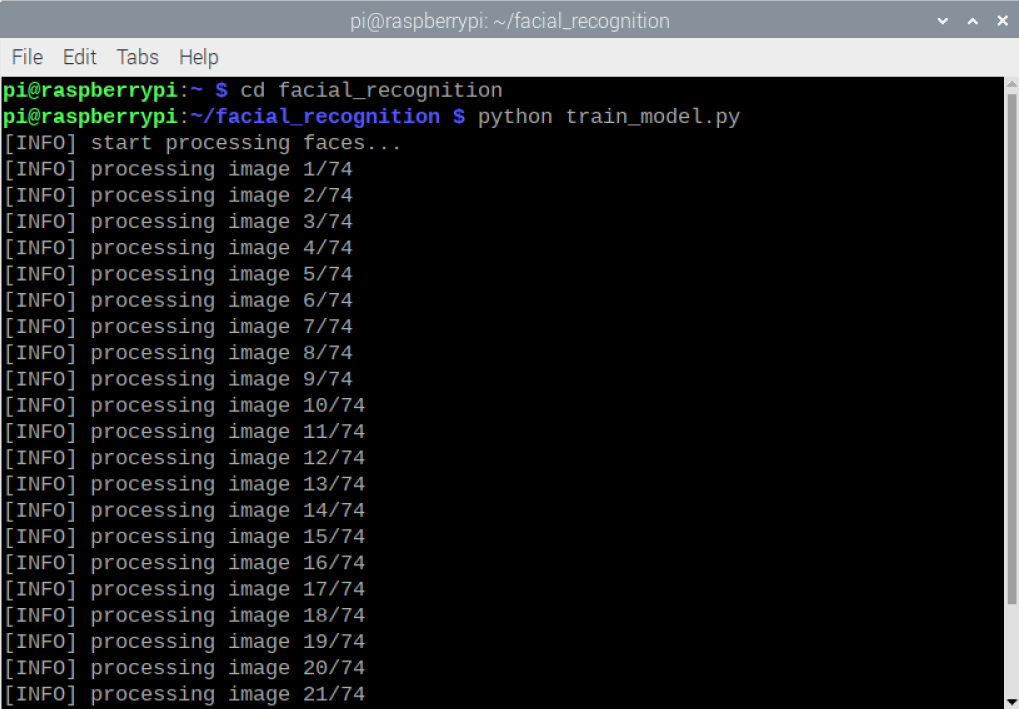
**5.3.2 Take image by PIcamera**

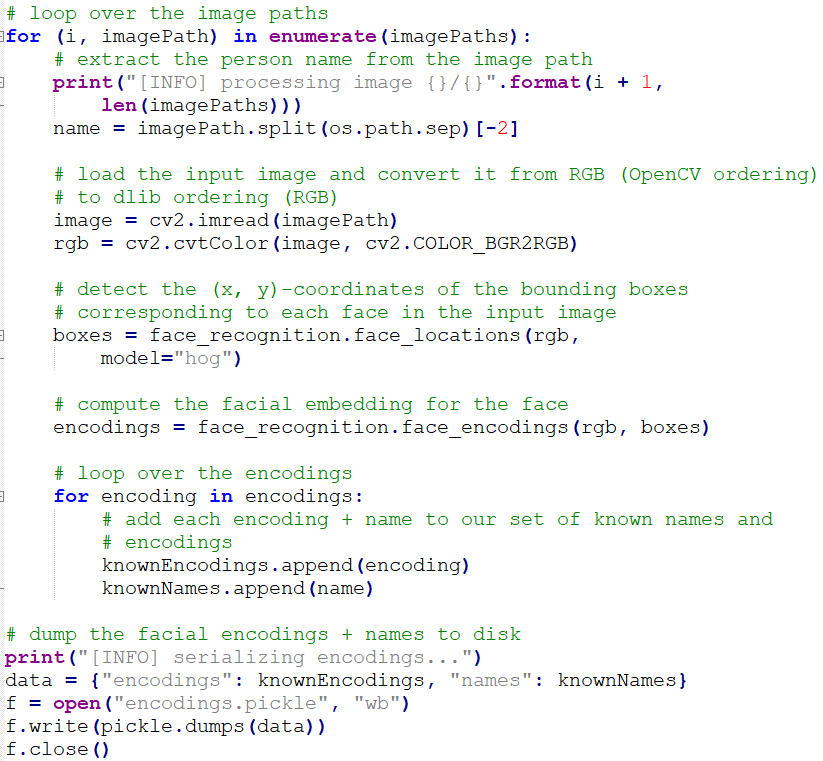
We have created a file named “headshots\_picam.py” inside the facial\_recognition folder. We wrote the code for taking pictures by RPI camera as long as we want. Have to define what the person’s name is. According to this, we created a folder named “dataset” and inside this, there will be a folder according to the person’s name where the images are saved**.**



**5.3.3 Train Images**

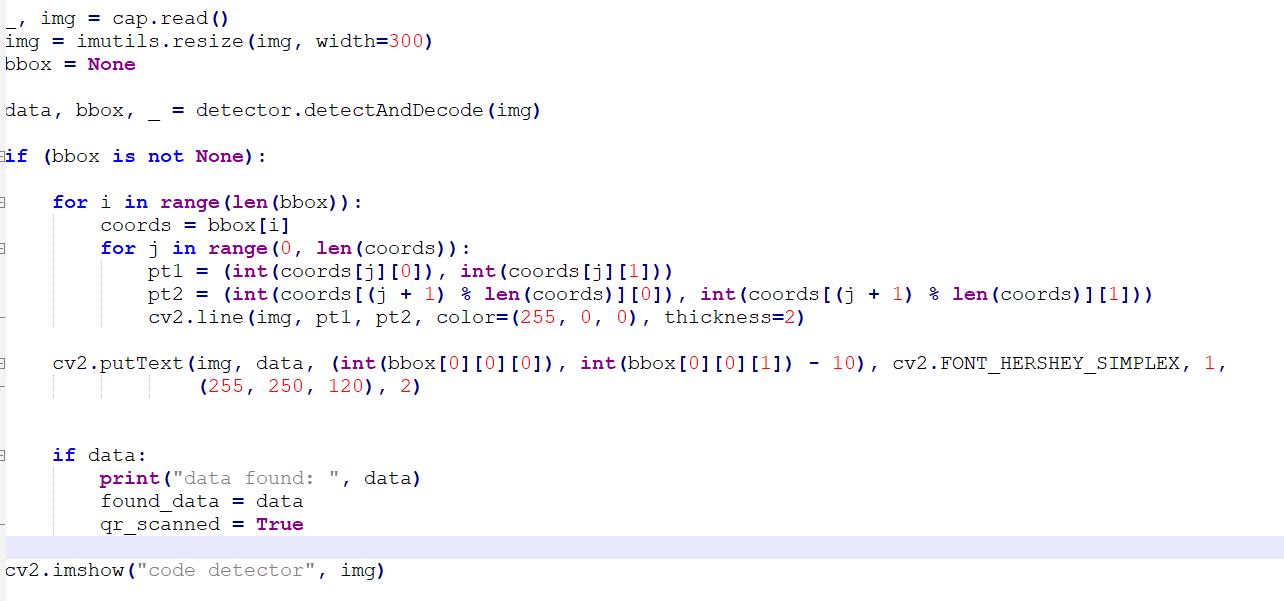
After taking images, we trained these images. After training these images, some information will be generated and overwritten (if necessary) in “encodings.pickle” file.





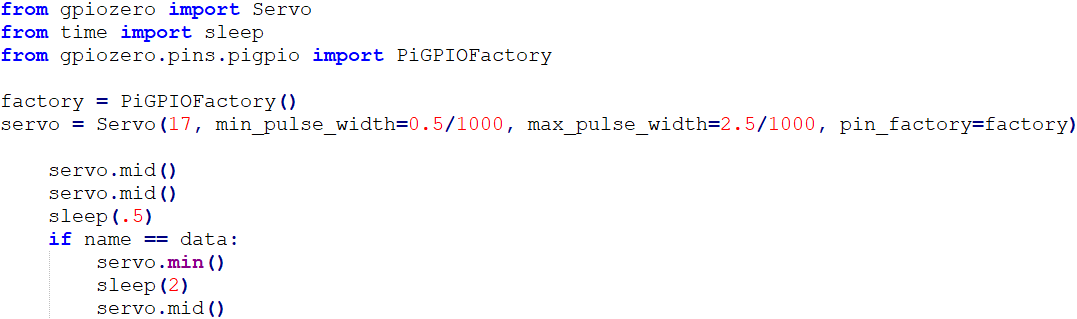
* 1. **QR code scan with PI camera**

Necessary packages are already installed. In the “facial\_recognition” folder, we have the project.py file. After detecting the face, picam wants to scan the QR code. Initially, the camera scans for the image of the QR code. Then, read the QR code by detecting the bounding box coordinates and decoding the hidden QR code. There has a blue box around the data. This draws one and then, writes the data along with the top. If we want to stop detecting, need to press ‘q’ only.



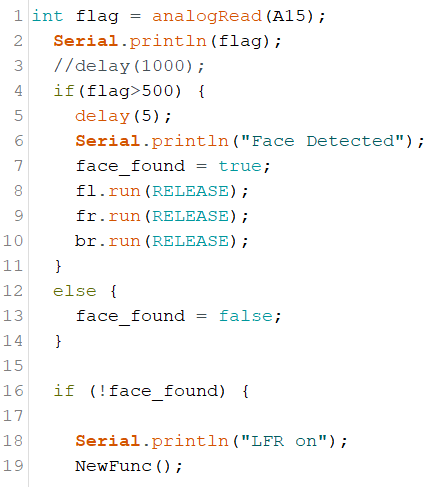
* 1. **Servo motor configuration in Raspberry PI**

There we have gpiozero library that we used for the servo motor. We also set the minimum pulse to 0.5 milliseconds and the maximum pulse to 2.5 milliseconds. Factory is the other way of controlling the pins hardware control, not the software. We used these calculations for smooth movement of servo motor so that there won’t happen jitter issue.

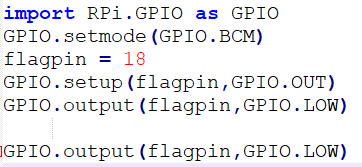


* 1. **Arduino Mega and Raspberry PI connection as Serial via GPIOs**

We have connected a wire from A15 to RPI’s pin 18[]. In the LFR code, we declared a variable for that. We have manually tested the value of that pin after detecting the face and the command here is to stop the LFR after detecting a face. If the camera of RPI can’t detect the face, LFR will run continuously.



Here, the GPIO library is already imported. It can transmit data through pin 18. LFR will continue when this pin sends “LOW”.BCM follows the lower-level numbering system defined by the RPI’s Broadcom –chip brain. GPIO. BCM refers to the pins by the “Broadcom SOC channel” number.

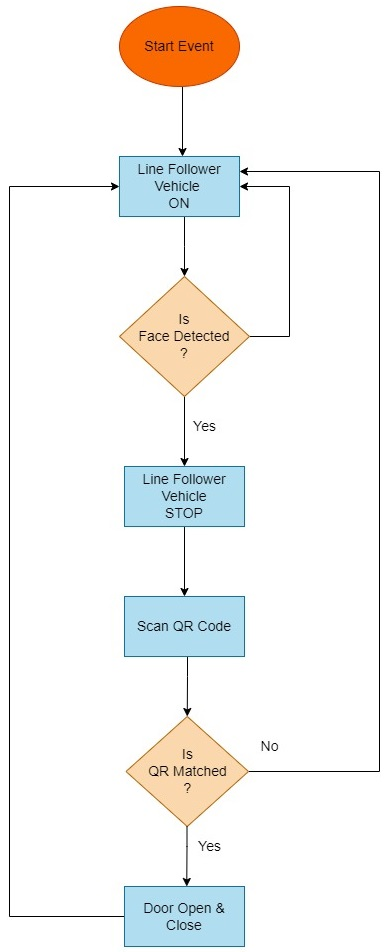


* 1. **Ready Object delivery system**

In this section, we described the flowchart of the Object delivery system, how to execute the system properly.

* + 1. **Flowchart of Object Delivery System**

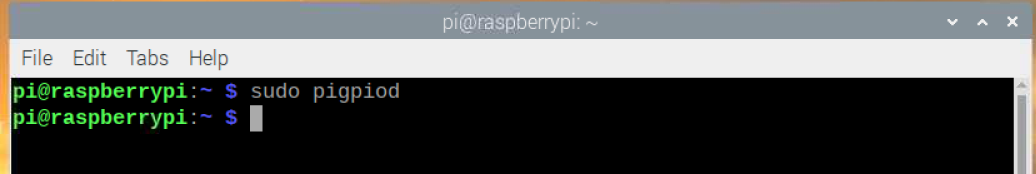
Initially, we enable the switch of the line follower robot and it will run by following the line. At the same time, we run our “Project.py” file and it will try to detect a known face. If the camera detects a known face, LFR will stop following line. Then, the camera will be ready to scan the QR code. If the QR code matches the specific known face, then, the servo motor will rotate back to degrees for 2 seconds to get the object. After that, the servo motor goes to its previous position. If the wrong QR code is scanned, the servo motor will not rotate. For both cases, after scanning, LFR will start to follow lines.



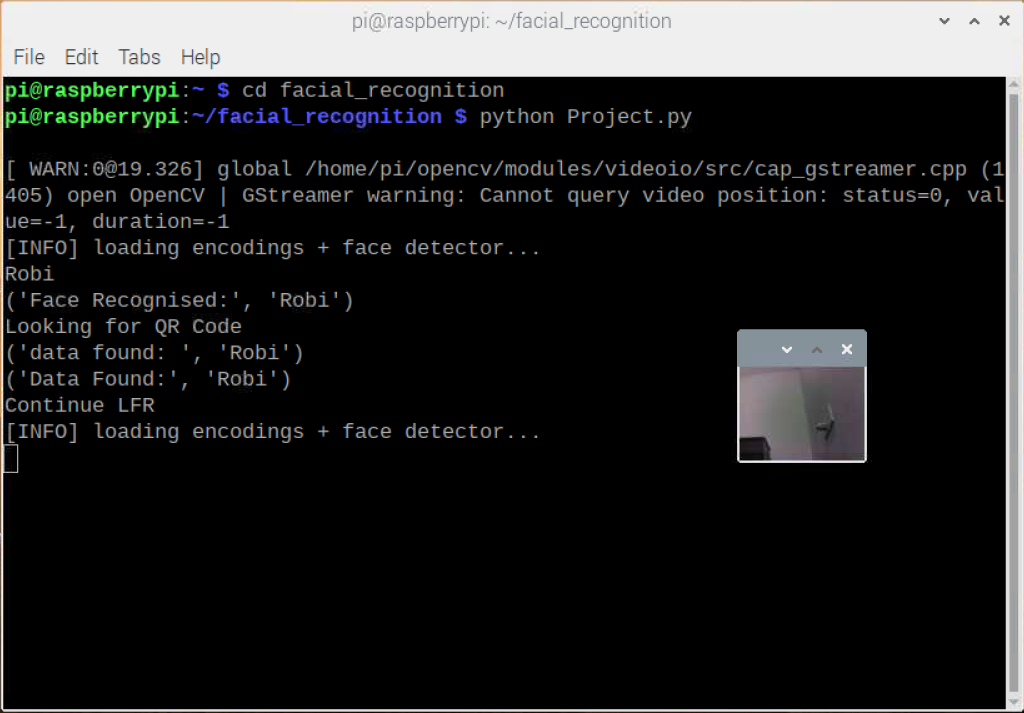
**Fig: Flowchart of Object Delivery System**

**5.7.2 Execute the whole system**

Initially, we need to run “pigpiod” and this utility runs in the background. This library controls the GPIO. Once “pigpiod” is launched, it accepts the commands from the pipe and socket interfaces.



Secondly, we moved to the facial\_recognition folder from pi folder. We have a “Project.py” file. It includes the code for face recognition, QR code scan, servo motor’s operating code, RPI and Arduino combined working code.



After executing this file from the terminal, one separate window will appear and ask for the authenticated face. After detecting the face, it’s ready to scan the QR code. Remember, there is a unique QR code for each person. If the wrong QR is scanned, the door will not open and the line following will be continued.

* 1. **Performance analysis**

We changed some configurations in the c code which we used in Arduino mega board and raspberry PI. After changing the configuration, we saw some changes in performance. In this section, these things will be discussed.

* + 1. **Performance analysis of line follower robot**

Initially, we set the motor speed at 90. Though, the motor speed maximum can be 255. But if we slow the motor to 60 or 70, it can’t run because of the weight of the whole system. This object delivery system weighs 1.3 kilograms. If we set the motor speed to 100 or 150, the IR sensor could not read the sensor value so quickly, which did not work too.

* + 1. **Performance analysis of PI camera**

PIcamera has some lag issues, if we increase the resolution of the video output window, the response time of the camera output is too slow.

* + 1. **Performance analysis of servo motor**

The servo motor can run well if it uses max, min pulse time. Otherwise, there has a jitter issue.