**FINAL PROJECT REPORT**

LINE FOLLOWING ROBOT USING OpenCV and an Arduino Uno

By :

Name : Saumya Ranjan

Roll No. : BTECH/10138/20

Embedded Team

Mentor : Shivam Shandilya

Prerequisites/Components Used

* **Physical Components** : One Arduino Uno board, L298N motor Driver, A mobile Phone, DroidCam App, Jumper Cables, 12v 100 RPM DC motors x2, Metal Chassis, Castor wheel, 11.1 Volt Li-Po Battery, Laptop to power the Arduino and for image processing
* Knowledge of Image Processing through OpenCV in Python
* Knowledge of Arduino Uno
* Knowledge of Arduino IDE
* Knowledge of different aspects of PID control system for the PID tuning of the system
* Knowledge of L298N Motor Driver

Introduction

A Line Following robot which with the help of image processing using OpenCV follows a black line/path in a white background autonomously.

Working

Phone Camera captures the present frame -> sends it to the laptop for image processing -> OpenCV determines whether the black line is in the centre or on the left or right -> Sends this information to the Arduino IDE -> The Arduino IDE with the help of PID control changes the speed of the motors based on the corrections required -> If the black line is to the right of the screen, instructions are given to turn to left and vice versa.

* To control the direction the robot turns to, change to the speed of the motors is introduced.
* The robot turns right if the right wheel’s speed is less than the left wheel and vice versa.
* The robot moves forward if both the motors have the same speed.
* The L298N is the motor driver used to drive the motors.

IMAGE PROCESSING USING OPENCV

General Overview

(The direction of the robot is from right of the screen to the left of the screen.

The up direction in the image corresponds to the right of the robot and vice versa.)

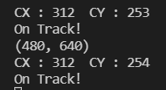
1. Straight movement
2. The original frame captured by the camera. The white dot is used to determine the approximate centre of the contour of the black line in the frame.

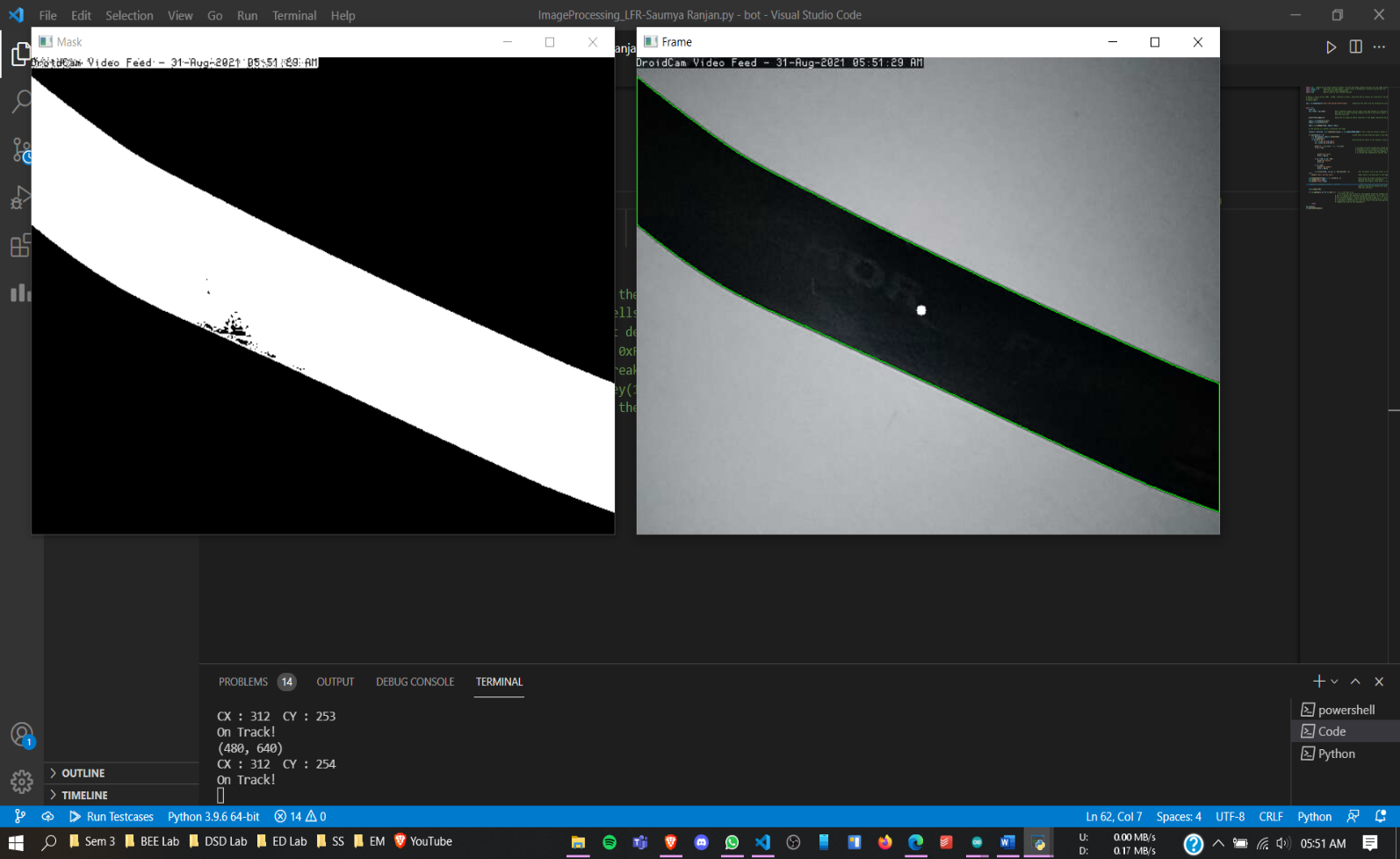


1. The same frame with image masking applied on it, it shows what the robot sees when looking at the frame.

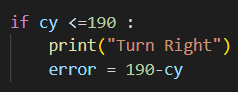


1. Since the black line is approximately in the middle of the screen , the bot has instructions to go straight.



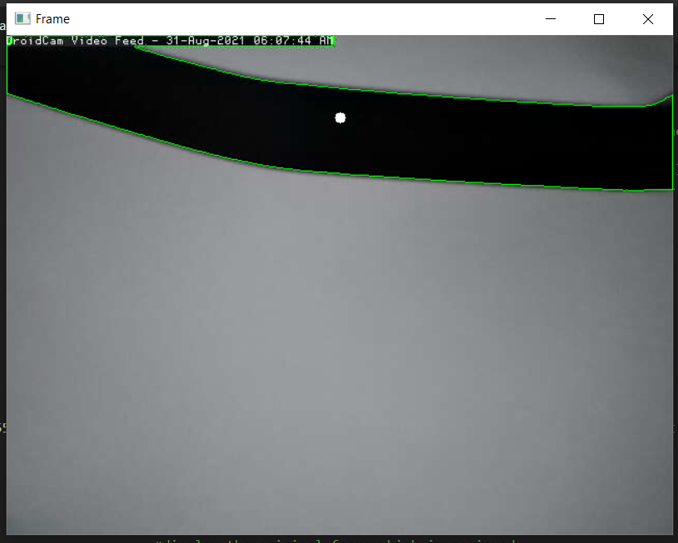


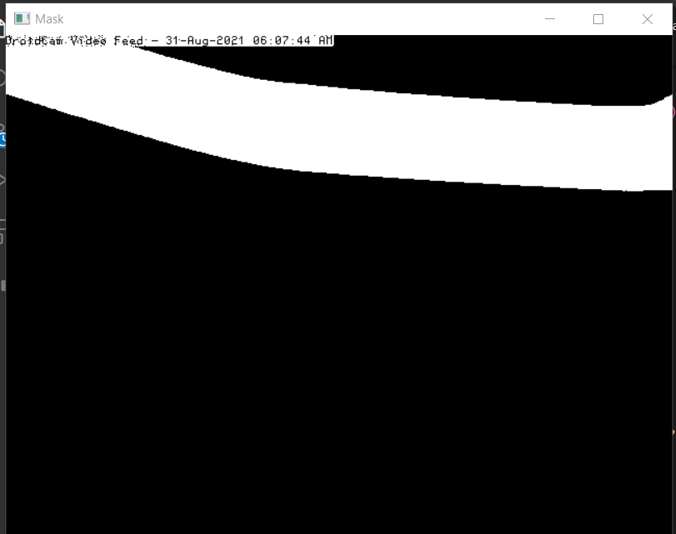
1. Right Movement

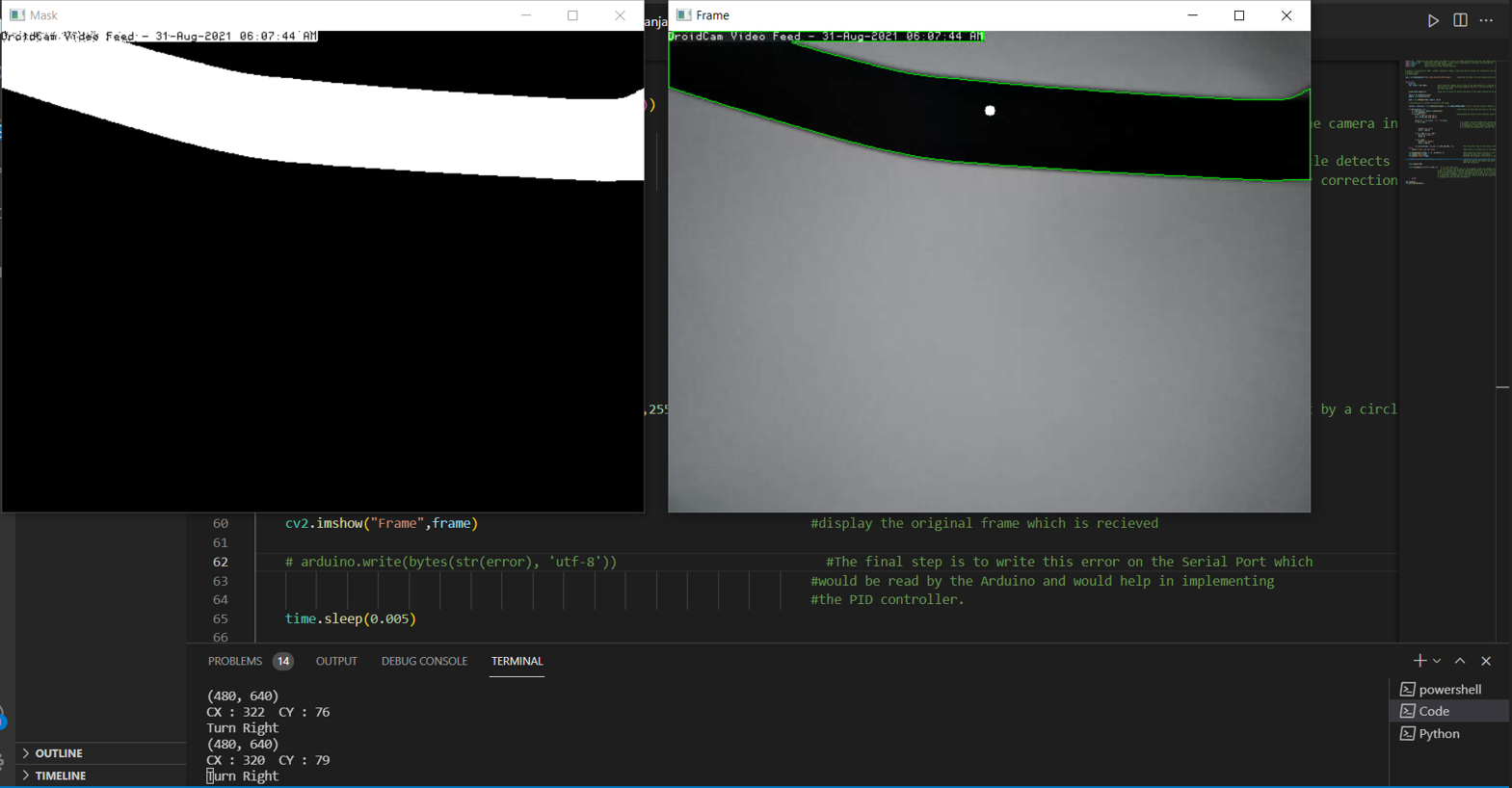


* cy is the variable which is the height of the central dot from the top of the screen.
* Since the camera is recording in portrait mode and the image processing is done on the landscape mode.

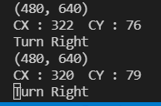
The (up) direction in the images attached correspond to the (right) of the screen.

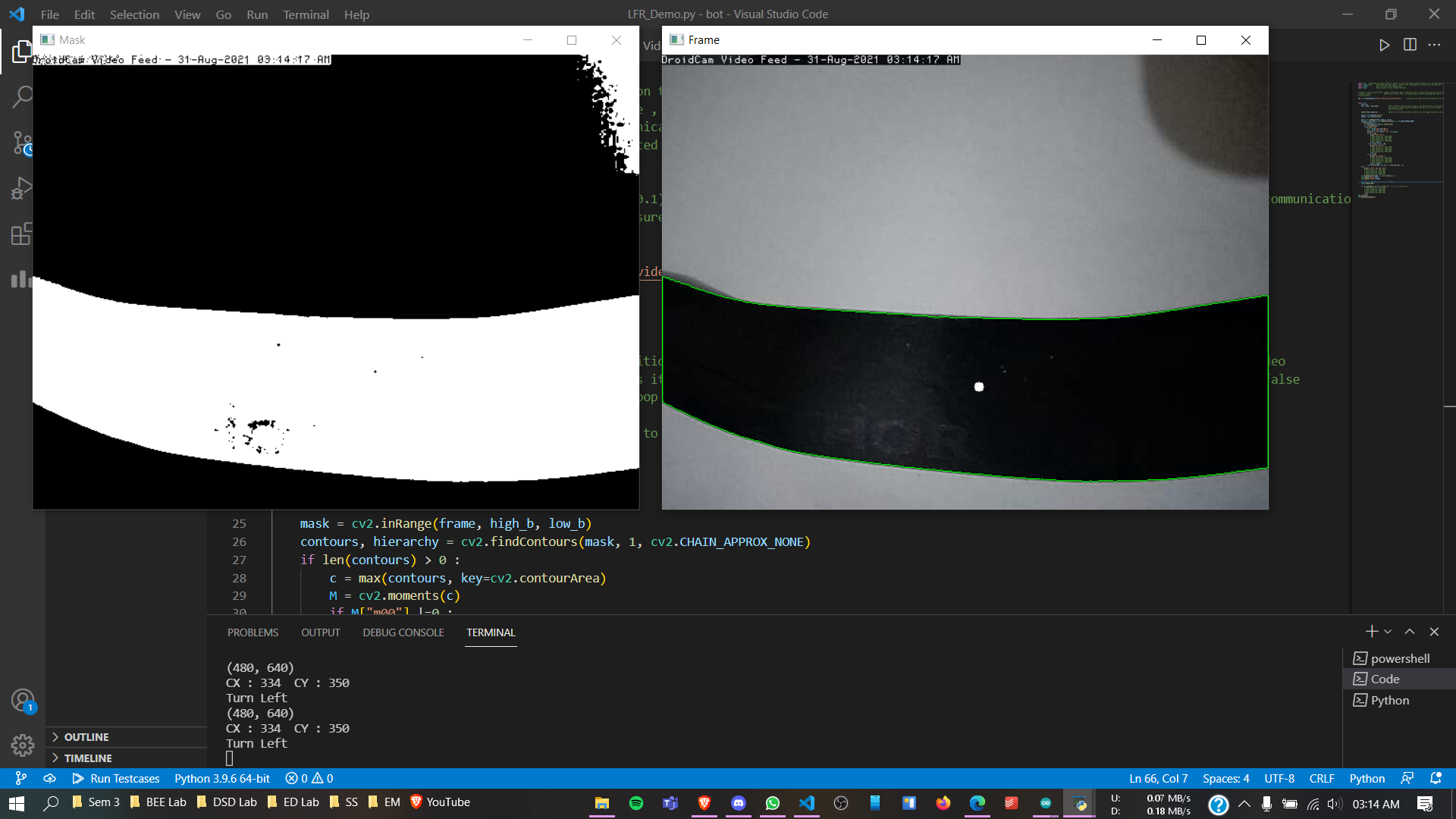
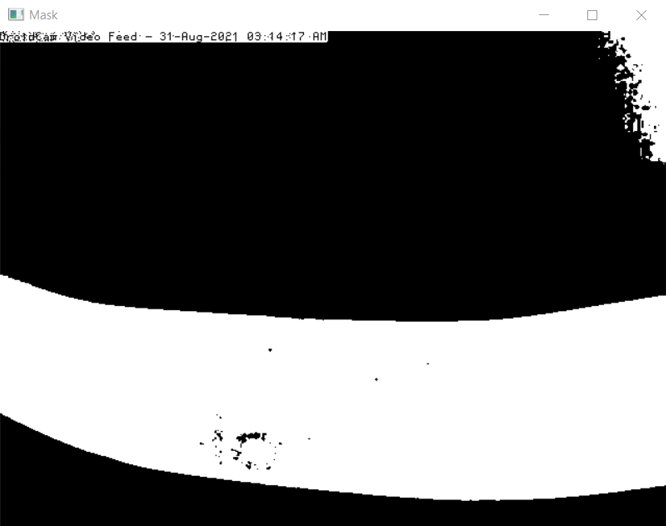
* So if the black line is to the up (right) of the frame, it needs to go to where the black line is, i.e. to the right , the bot gets instructions to turn to right.



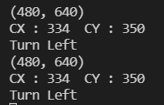


Since the location of the centre of the dot is 76 pixels from the top of the screen (which corresponds to 76 pixels from the right of the camera frame) , the instruction should be to turn right so the black line is maintained in the middle of the camera frame.



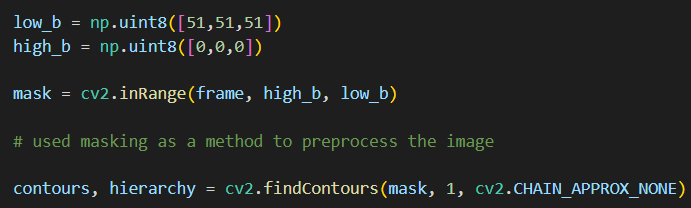
A similar processing is done for the leftwards movement. 





Since the dot is at the bottom of the screen, the instruction is to go leftwards.

* Black Line Detection Using Masks and Contours



Using contours to find the outline of the black regions in the mask frame created from the original frame.



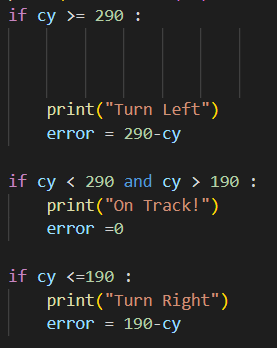
If there are some black regions found, find the contour with the most area. ( This is done to avoid focusing on other dark objects in the vicinity of the path )



Calculating the centre of the largest detected black region using the moments method. (the white circle)



* Conditions for Turning



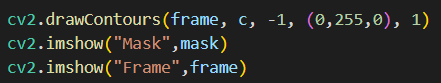
The “centre” is defined as being in the range of 190 to 290 pixels.

If the y coordinate of the dot is more than 290, i.e. 290 pixels from the top of the screen, the error variable will contain the necessary correction to turn the bot leftwards.

If the y coordinate of the dot is less than 190 pixels from the top, the bot turns right, the error variable will contain the necessary correction to turn the bot rightwards.

If the y coordinate of the bot is between 190 to 290 pixels , the bot continues to go straight, and the error variable contains the value 0, i.e. no correction required.

* Verification



Finally to view whether the camera is detecting properly, we print out the contour lines along with the frame.  
We also print the mask, and the original frame along with a detecting circle to keep track of the centre of the detected contour and to give us a measure of adjustment required.

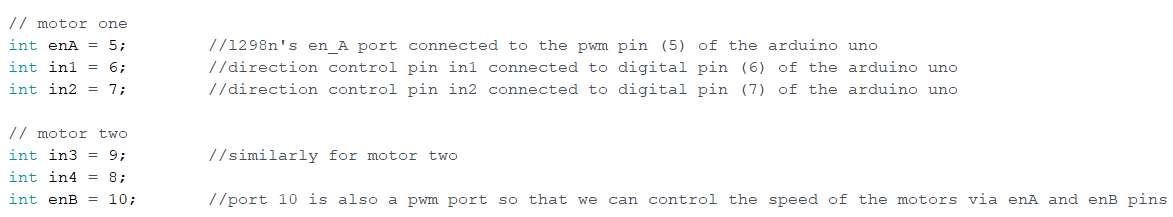


This is used to draw a white filled circle of radius 5 pixels at the centre of the contour for a better view of the detection.



The final step is to send this error data to the Arduino IDE for PID control.

ARDUINO CODE AND PID TUNING

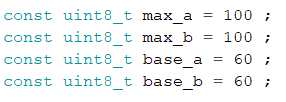


These are the connections which need to be communicated to the Arduino Uno board.

The speed control pins viz. ENA and ENB are used to turn the motors ON, OFF and control its speed.

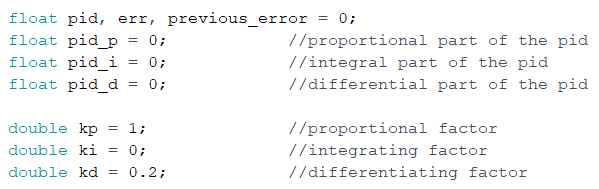
Pulling these pins HIGH will make the motors spin, pulling it LOW will make them stop. But, with Pulse Width Modulation (PWM), we can actually control the speed of the motors.

That is why the enA and enB are connected to PWM pins to regulated the speed at which the motors rotate.



Maximum Speed of motor A and B is 100 (0-255).

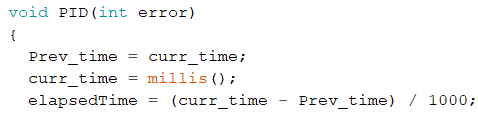
Base Speed of motors A and B is 60 (0-255).



Here we setup the initial errors.

The kp , ki and kd values I acquired from trial and error and can be different for different people.  
These were the values that my model worked best on at the base speed of 60.

The kp, ki and kd values are the factors with which the proportional , integral, and the differential errors influence the final error.



millis() returns the number of milliseconds passed since the Arduino board began running the current program.

elapsedTime is calculated by substrating the curr\_time and Prev\_time.



Proportional Error

this factor requires an error to be present to produce a correction, if there is 0 error then no correction will be produced

this produces a steady state error and alone can’t guarantee that the bot will follow the correct path , for that we require the other parts of the PID control too.



Integrating Error

This adds the small area below the error graph over the very small elapsed time to the previous total of the error.

The integrating part is used to reduce the steady state error by summing all the previous errors.

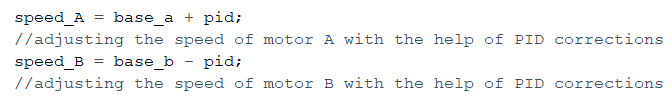


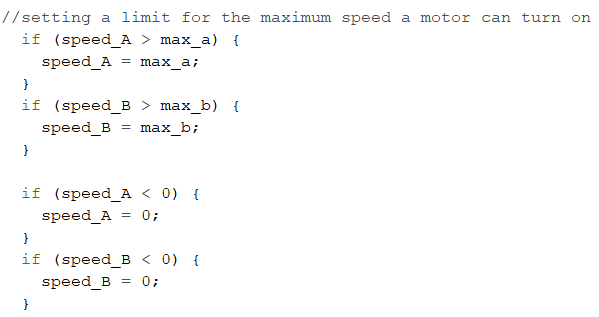
Derivative Error  
If the error is decreasing , its graph will have a negative slope and hence the derivative will be negative which when added to the total correction will mean that the corrections will also decrease.  
This helps to slow down the motors if the robot is approaching the centre point very fast.



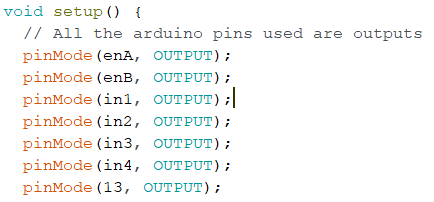
Then I calculate the final error by summing up all the errors.

Applying the final corrections to the motor speeds of the robot.





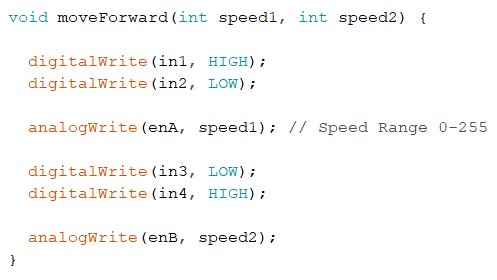
This ensures that the speed of the motors never go above the value of 100 which I set previously and limiting them to the maximum if they do.



All of pins on the Arduino are used as outputs where the Arduino will provide (0V or 5V) (off or on) in the in1, in2, in3, and in4 ports and for the enA and enB pins it will provide a pwm output which can be from 0-255 since these pins control the speed of the robot.



Setting the current time and beginning the communication with the Arduino uno board with a baud rate of 115200 bits/second.



Takes speed1 and speed2 as the inputs and sets enA and enB equal to them ( which are the speeds of motor 1 and motor 2 )

For motor 1,

If in1 is low and in2 is low, motor 1 will stop.

Since in1 is high and in2 is low, motor 1 spins forward with speed1

For motor 2,

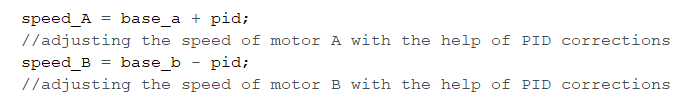
Since in3 is low and in4 is high, motor 2 spins forward with speed2

Loop()



errors variable stores the error sent through the serial port by OpenCV -> this part 

  
now this error is acted upon by the PID control function described before.



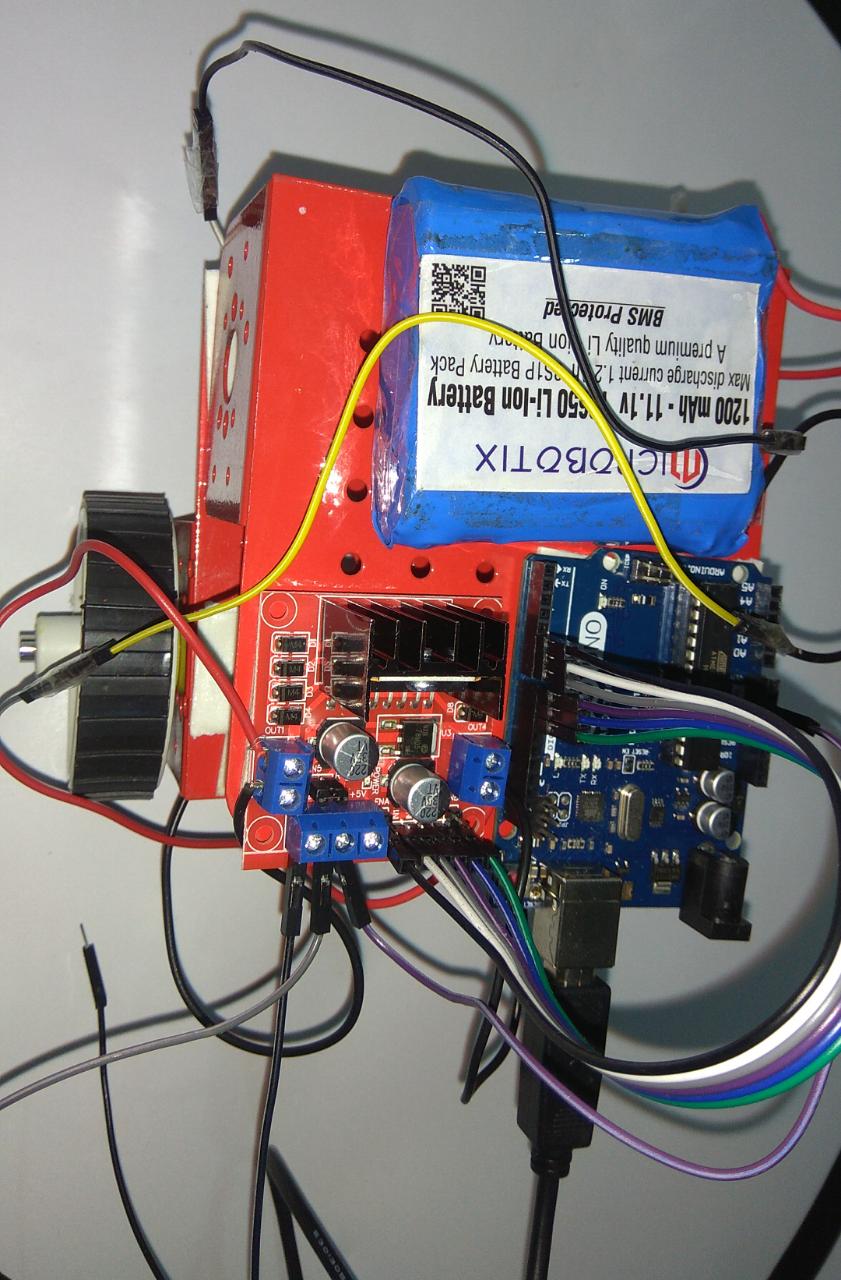
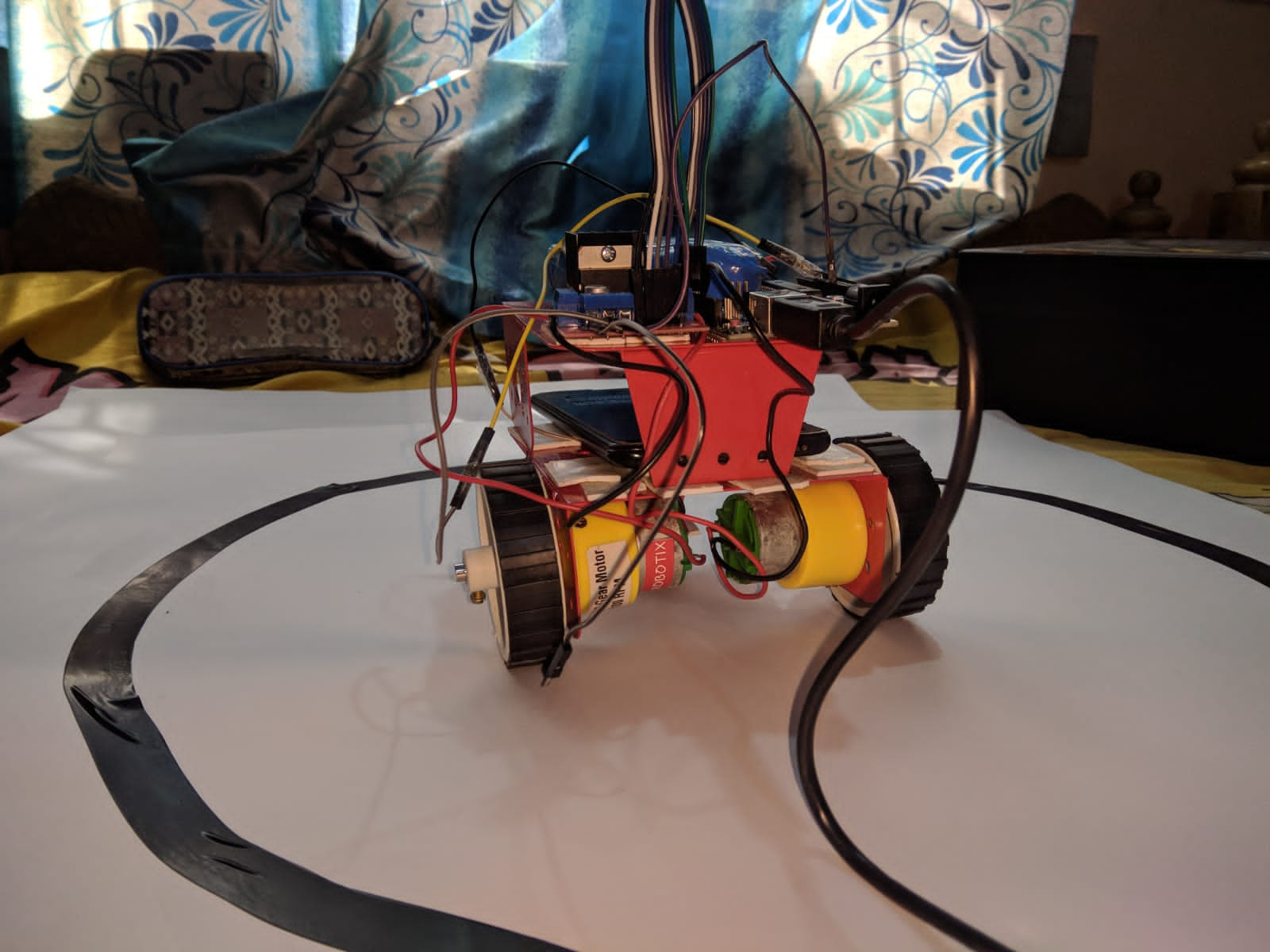
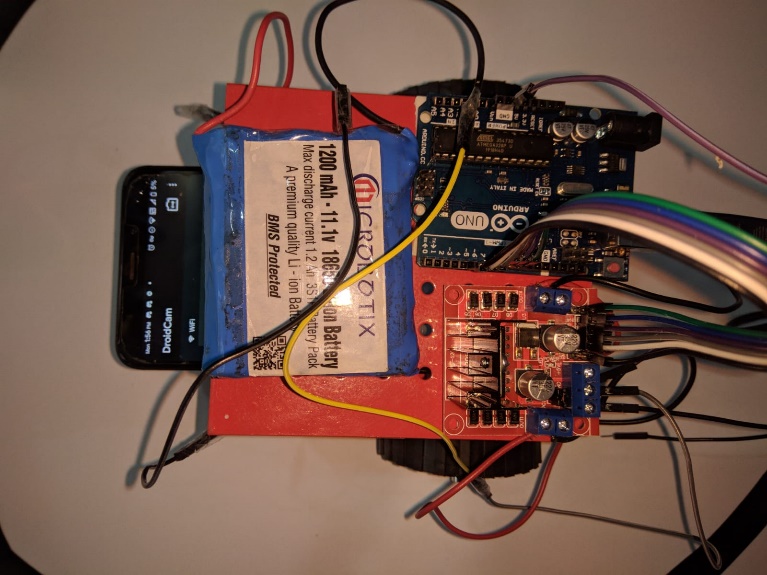
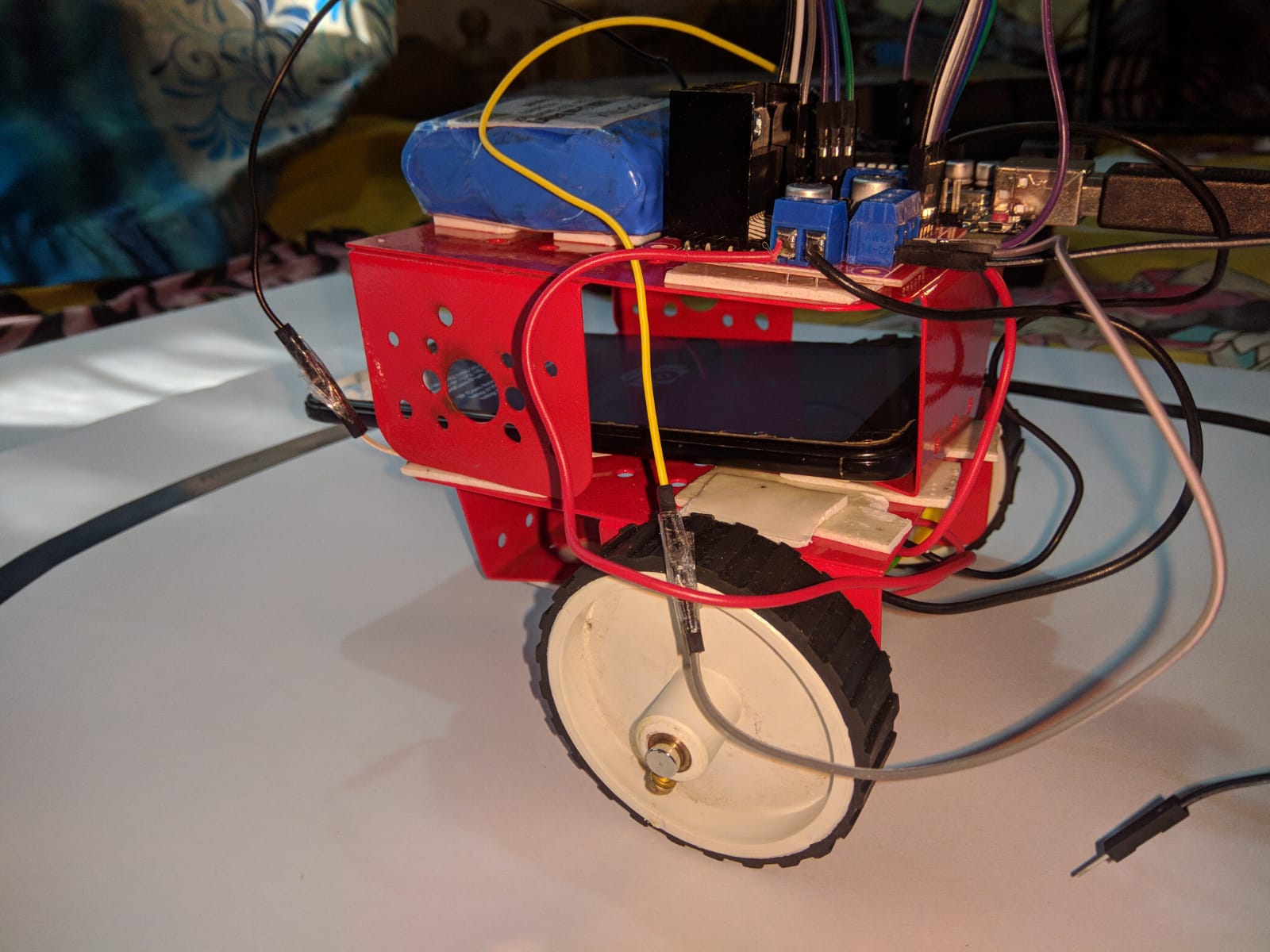
Which sets the speed of motors A and B.



And finally the L298N receives these values through the Arduino board to turn the motors.

This process repeats with feedback in the loop again and again until the Arduino is disconnected.

CONNECTIONS



The connections

Arduino Uno to L298N

pin 5 -> enA

pin 6 -> in1

pin 7 -> in2

pin 8 -> in3

pin 9 -> in4

pin 10 -> enB

GND Pin of Arduino to GND pin of L298N ( for common ground )

L298N to Li-Po Battery Pack

Positive terminal -> Vcc (+12 V pin)

Negative terminal -> GND (+5 V pin)

USB port of Laptop to the Arduino (to power the Arduino)

Port Used : COM4 port

Tasks still to be taken-up

* More PID Tuning to make the bot smoother
* Using an ESP 32 camera module instead of the Phone Camera to make the robot lighter

Technical Problems Faced

* During Image Processing , at high speeds sometimes the bot would veer off to one side even after detecting the black line due to a big difference in the maximum and base motor speed
* PID tuning

Project Repository Link

<https://github.com/saumyaranjan1111/Line-Following-Robot-Using-OpenCV-and-Arduino>

Video Link

<https://youtu.be/vleiL5a85_Q>

REFERENCES

<https://lastminuteengineers.com/l298n-dc-stepper-driver-arduino-tutorial/#:~:text=ENA%20pins%20are%20used%20to,the%20speed%20of%20Motor%20A>.

<https://en.wikipedia.org/wiki/PID_controller>

<https://www.youtube.com/watch?v=wkfEZmsQqiA>

<https://www.youtube.com/watch?v=UR0hOmjaHp0&t=5s>

<https://www.youtube.com/watch?v=tlkWX7R-NHE&ab_channel=EngineerKid>

<https://www.youtube.com/watch?v=oXlwWbU8l2o&t=6272s>

<https://www.youtube.com/watch?v=VQmxU7CV9bM>