CLUSTER INNOVATION CENTRE, DELHI UNIVERSITY

Calculus and Linear Algebra Project Report on

Construction of Line Following Robot Using PID

BY - Karteek Reddy | Shreyas Sachan | Sudhanjali Sethi | Vaibhav Jain
(B.Tech – 1st Year)

Cluster Innovation Centre, Delhi University

ABSTRACT

This report aims to guide any individual having an interest in making a Line Following Robot (LFR). This report is especially useful for students beginning their study in the field of robotics. It also briefly covers some basic concepts on Proportional–Integral–Derivative (PID) mechanism commonly used in control systems.

We tried to make this report in form of a manual which can be used in future as a DIY guide. Step-by-step instructions which are spread throughout the text in different sections help to understand the complex process of making a LFR easier. Devices used are elaborately explained so that reader will be able to use them in other projects. Reader is expected to have basic knowledge of calculus and programming in C or C++ or embedded C.

After reading this text, reader will be exposed to the world of robotics and electronics. Reader can choose to work on further improvements on LFR or explore many other projects in Robotics.

Acknowledgement

We would like to express our special thanks of gratitude to our mentors Dr. Harendra Pal Singh and Dr. Nirmal Yadav who gave us the opportunity to work on this project, which also helped us in learning a lot of things. We are really thankful to them. We are also thankful to all the seniors, who have always been a source of motivation and knowledge.

This project consumed huge amount of work and dedication.

Still, it would not have been possible if we did not have a support of many individuals and internet. Therefore we would like to extend our sincere gratitude to all of them.

CONTENTS

1. INTRODUCTION

A Line Following Robot (LFR) is an autonomous robot which is capable of following a black line on a white surface. It is a very basic robot and popularly the first step of students entering into the field of robotics.

This is inspired us to make this manual for making a LFR. The manual provides a step by step instruction along with explaining the theory behind it and a brief introduction to Arduino IDE and basic Electronics. First we will collect all the tools required and learn to use them in *Instruments*. Then we will familiarise ourselves with the theory and algorithms essentials to make the robot move in *Theory*. After that we will be ready to start to construct and code our robot in *Procedure*. In the end, we will end with some analysis in *Discussion*.

2. INSTRUMENTS

2.1 List of items required

Given the following is a list of the instruments we are required to make our robot:

1. **Arduino Uno/Mega/Nano** x 1

   These are microcontroller boards used for controlling robot and processing information we get from sensors. We recommend using Arduino Uno for beginners since it ample amounts of pins yet user friendly in size.

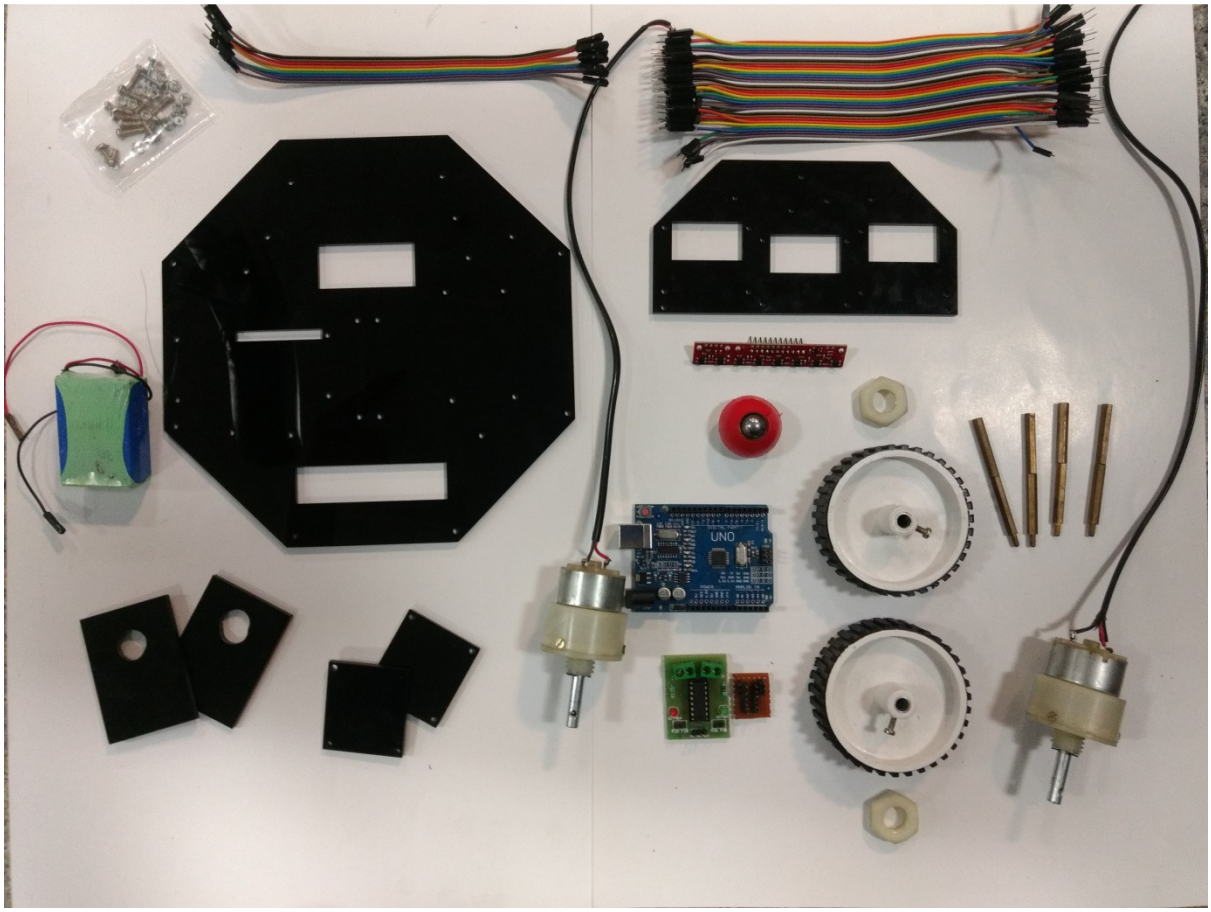
2. **Pololu QTR-8RC sensors** x 1

   These sensors detect black line on white board thus acts as the eyes for the robot.

3. **L298 Dual Motor Controller module** x 1

   This is used to control motors' speed and direction.

4. **Standard 12V DC motors** x 2
5. **Caster wheel** x 1
6. **LFR Chassis (**upper and lower plates**)**
7. **Plastic Wheels** x 2
8. **Puff-board**
9. **Rod screw** x 4
10. **Screw and Bolts**
11. **Jumper Wires**

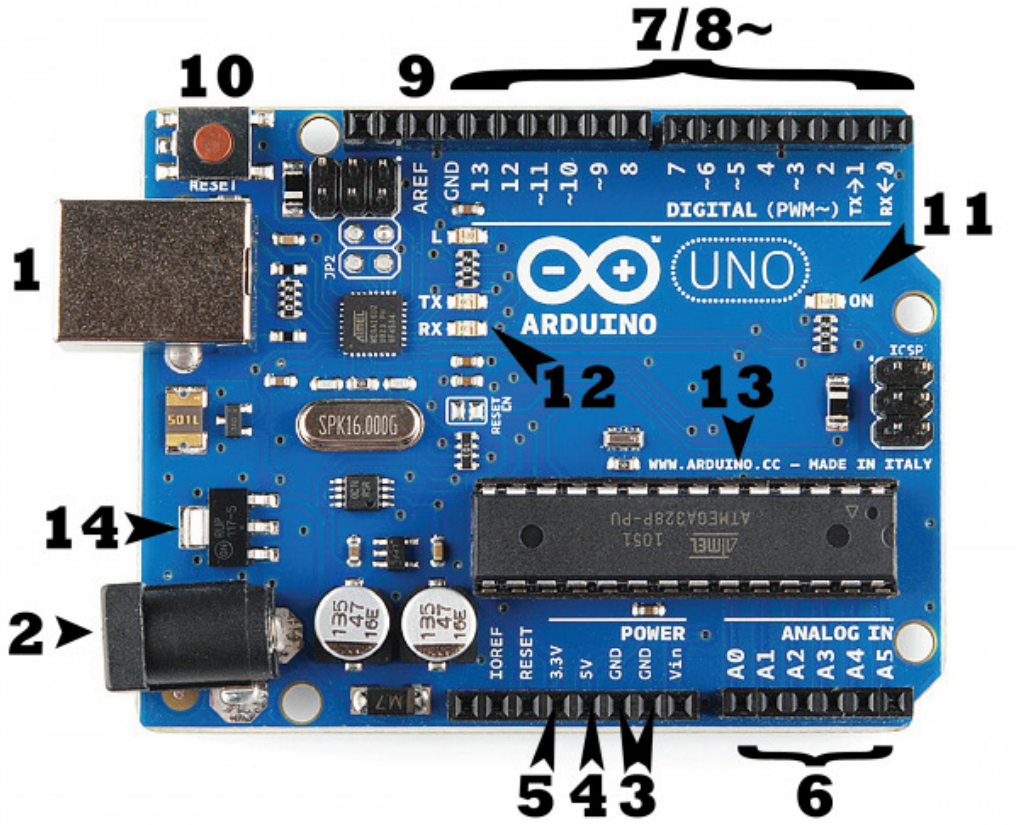


12. **Standard 12V battery** x 1
13. **Wire cutter**
14. **Glue** or **Double Sided Tape**
15. **Soldering Iron**
16. **Soldering Wire**

2.2 Using Arduino IDE

Arduino is an open-source platform used in electronics projects. It has two main parts, a programmable circuit-board or the "micro-controller" and the Integrated Development Environment or the IDE Software. It can thus be used to write the computer code and then upload it to the circuit board.

Arduino IDE is an important open-source platform because it doesn't need a separate programmer to upload the code to the circuit board, all you need is a USB cable. Next, it uses a simplified version of C++, which makes it user-friendly. The hardware parts of Arduino are quite inexpensive which makes it the optimum choice. And the best part is that it is free!



(1) - The USB Connection

(2) - The Barrel Jack

(3)- The GND or Ground Pins

(4)- The 5 Volts Pin

(5)- The 3.3 Volts Pin

(6)- The Analog Pins

(7)- The Digital Pins

(8)- The PWM or Pulse Width Modulation Pins

(9)- The AREF or Analog Reference Pin

(10)- The Reset Button

(11)- The Power LED Indicator

(12)- The TX RX LEDs

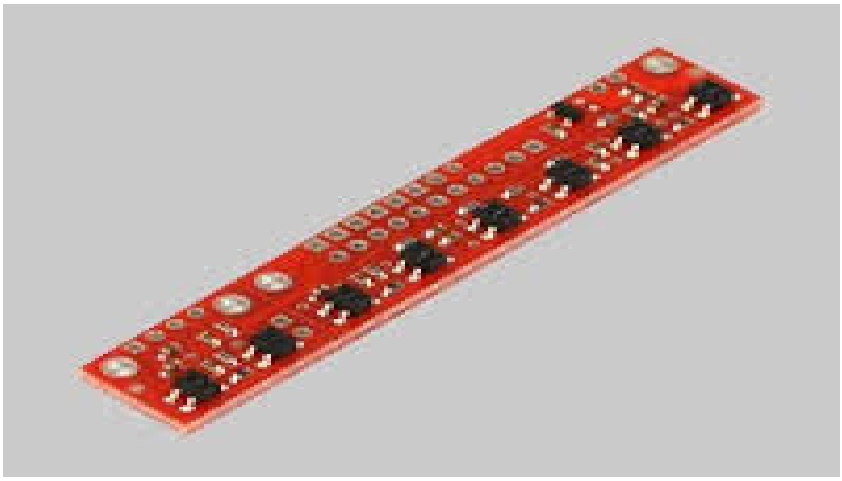
(13)- The IC or Integrated Circuit

(14)- The Voltage Regulator

The Arduino UNO is powered using the USB cable (1) which is terminated into the barrel jack (2). The GND or Ground pins (3) are used to ground the circuits. The 5V pin (4) and 3.3V pin (5) are used to supply 5 volts and 3.3 volts power respectively. The Analog In pins (6) are used to convert analog signals into digital values. In Arduino UNO A0 to A5 are the Analog In pins. The Digital pins (7) are used for both digital input as well as digital output. In Arduino UNO 0 to 13 are digital pins. There is a tilde (~) in some of the digital pins (8), these pins can act as digital pins and can also be used for Pulse Width Modulation. PWM or Pulse Width Modulation is a method for obtaining analog results by digital means. The AREF or Analog Reference pin (9) is used to set a certain voltage between 0 and 5 Volts as the upper limit for analog pins. The Reset button (10) is used to temporarily connect the reset pin to the ground pin, so as to repeat the

code loaded on the Arduino. There is a tiny LED (11) which illuminates when the Arduino is connected to a power source. The TX and RX LEDs (12) are used for serial communication. TX stands for transmit and RX stands for receive. The IC or Integrated Circuit (13) is the brain of the Arduino. All the other circuits are connected to the IC and it thus controls the functioning of the Arduino. The Voltage Regulator (14) controls the amount of the voltage that enters the Arduino board.

2.3 Using QTR sensors



Dimensions - 2.95" x 0.5"

Operating Voltage - 3.3-5.0 Volts

Supply Current - 100 mA

Output format for QTR-8RC - 8 digital I/O-compatible signals that can be read as a timed high pulse

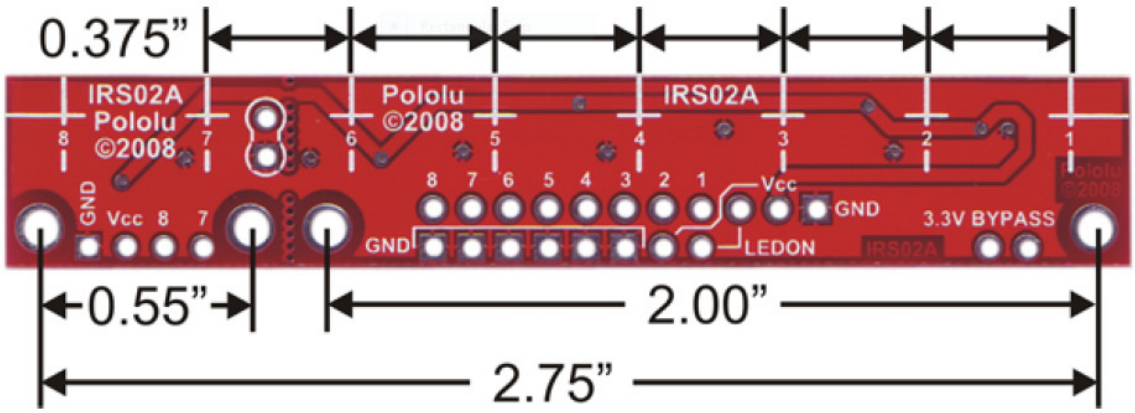
Optimal Sensing Distance - 0.125"

Maximum Recommended Sensing Distance - 0.375"

Weight without header pins - 3.1 gm

MODULE CONNECTIONS

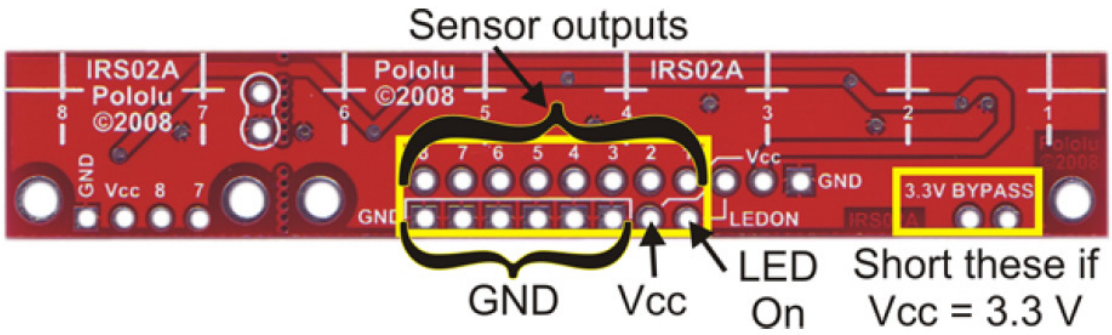
The QTR-8RC sensors provide some connection flexibility. The pins are standard 0.1" spacing, and are arranged to support connection using either an 11×1 header strip or an 8×2 header strip. A 25-pin 0.1" header strip is also included. The strips can be broken into pieces and then soldered as desired.



QTR-8A and QTR-8RC reflectance sensor array dimensions.

Vcc, GND and 3.3 BYPASS

The QTR-8RC sensor array has eight distinct sensor outputs, one for each LED. The sensor receives power at the Vcc and GND pins. The sensor's outputs are relative to the ground.



QTR-8x reflectance sensor array with 8×2 connection pins labeled.

The optimum Vcc should be as high as 5V. Using a lower Vcc would decrease the brightness of the LED and could even cause them to turn off immediately. This problem is solved by shorting the two 3.3V pins together, which bypasses one stage of the LED current-limiting resistors together and also increases the LED brightness.

LEDON

This pin is linked to the MOSFET which delivers power to the IR LEDs, thus it determines whether the LEDs are on or off. When this pin is driven high, the LEDs are on. And when this pin is driven low, the LEDs are off. This pin can even be connected to a high-frequency PWM which can control LED brightness and decrease power consumption.

QTR-8RC Sensor Outputs

The QTR-8RC reflectance sensor array also has eight distinct sensor outputs, one from each LED/phototransistor pair. In the 8RC sensor model, each phototransistor uses a capacitor discharge circuit that allows a digital I/O line on a microcontroller to take an analog reflectance reading by timing how long it takes the output voltage to decay due to the phototransistor.

When you have a microcontroller digital I/O connected to a sensor output, the typical sequence for reading that sensor is:

1. Turn on IR LEDs.

2. Set the I/O line to an output and drive it high.

3. Allow at least 10 micro-seconds for the sensor output to rise.

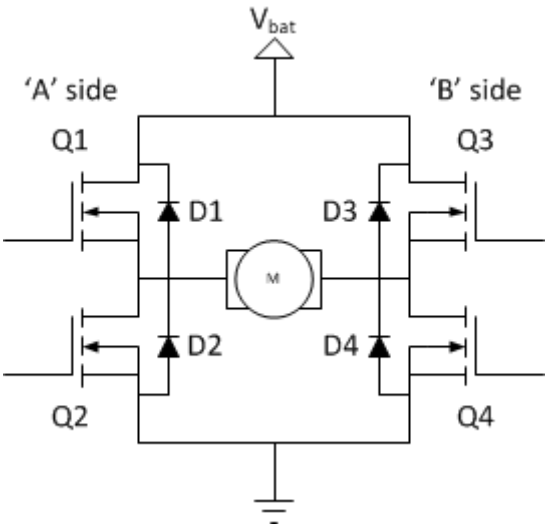
4. Make the I/O line an input (high impedance).

5. Measure the time for the voltage to decay by waiting for the I/O line to go low.

6. Turn off IR LEDs.

2.4 Using H-Bridge

An H-bridge is a circuit consisting of four switching elements and a load placed in the center, arranged in a way so as to form H-like shape.

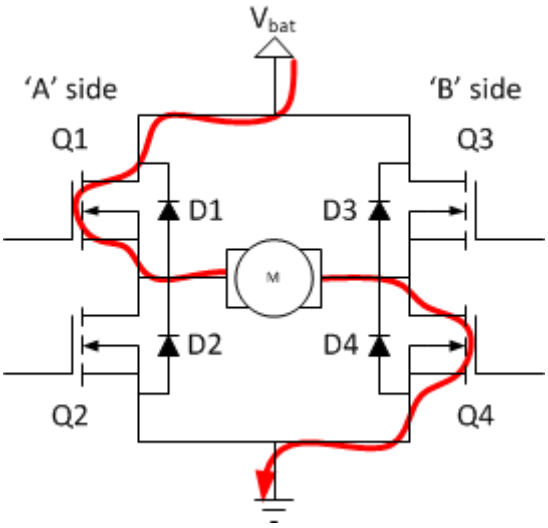


The H-bridge is connected to the battery on one end and is grounded at the other end. The four switched can be turned on or off independently.

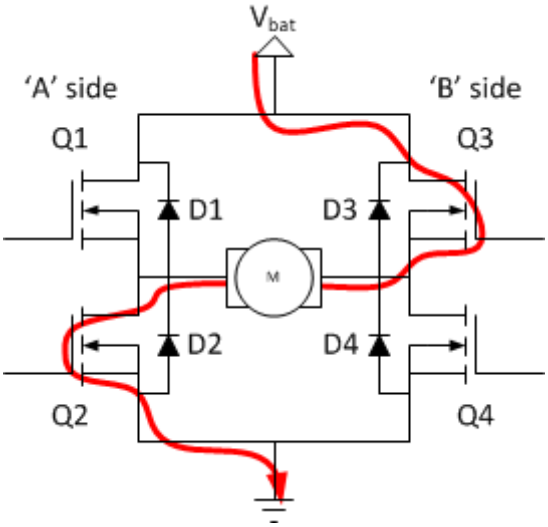
OPERATING THE H-BRIDGE

The H-bridge can be operated in multiple ways.

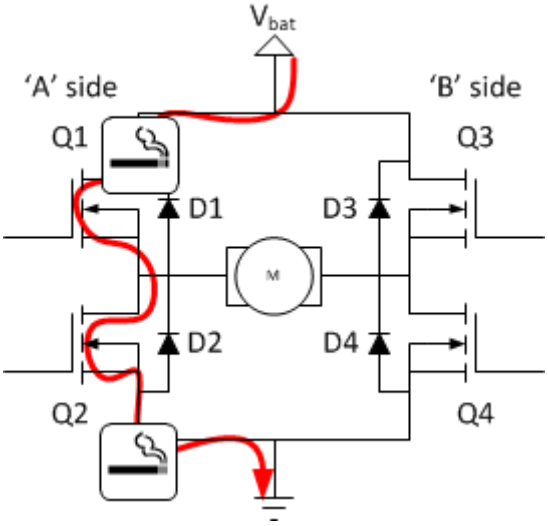
Case-1 Assuming that a DC motor is used as the load at the centre, if Q1 and Q4 are switched on, the left lead of the motor would be connected to the battery and the right lead would be connected to the ground. Current would start flowing, and the motor shaft would spin.



Case-2 Now if you switch Q2 and Q3 on, the right lead of the motor would be connected to the battery and the right lead would be connected to the ground. Current would start flowing in the opposite direction this time, and the motor shaft would spin backwards.



NOTE- Do not connect both Q1 and Q2, or Q3 and Q4 simultaneously. This could lead to the formation of a low-resistance path between the battery and ground, and could lead to short-circuiting.



Thus from the A-side out of the four possible connections only three would be successful.

Q1	Q2
On	On
Off	On
On	Off

Similarly for side-B:

Q3	Q4
On	On
Off	On
On	Off

Altogether this allows for 9 different states for the full bridge to be in:

Q1	Q2	Q3	Q4
Off	On	On	On
Off	On	On	Off
Off	On	off	On
On	Off	On	On
On	Off	On	Off
On	Off	off	On
On	On	On	On
On	On	On	Off
On	On	off	On

Advantages of using H-bridge

Arduino provides an output voltage of 5 Volts only. So you can't control motors using Arduino only. To solve this problem H-Bridge is used. Using an external power source you provide a voltage of 12 Volts. And input-output logics are provided using Arduino. H-Bridge thus acts as a voltage regulator and provides voltage to the input and output according to requirements.

3. THEORY

PID CONTROLLER

The basic principle of a Line Following Robot is that you're given a robot, and a line, and you want your robot to move on that line.

Target - the line on which the robot is supposed to move.

Error - the difference between the current position of the robot and the target.

$u(t) = k_p e(t) + k_d \, d(e(t))/dt + k_i \int_0^t e(\tau) d\tau$

where , u(t) = control variable

e(t) = Error as a function of time

k_p = Proportionality constant

k_i = Integral constant

k_d = Derivative constant

τ = t variable (takes value between 0 to t)

t = instantaneous time

LINEAR ALGEBRA BEHIND PID

In PID, we deal with some constants (kp, kd and ki) which are used in the equation ….. These constants generally modify the errors to find the power difference required to turn the robot. We basically apply some matrix transformation to the different errors (proportional error, derivative error and integral error) which is given as-

$$\begin{bmatrix} new\ proportional \\ new\ derivative \\ new\ integral \end{bmatrix} = \begin{bmatrix} kp & 0 & 0 \\ 0 & kd & 0 \\ 0 & 0 & ki \end{bmatrix} \begin{bmatrix} proportional \\ derivative \\ integral \end{bmatrix}$$

The above transformation transforms the proportion, derivative and integral error into new proportional, new derivative and new integral errors whose sum gives the power difference which is required between the two wheels of the robot in order to move it left or right.

P -CONTROLLER

When your robot is not "on" the line, and there is some error, and you want your robot to move so that it approaches the line, you need to steer in proportion with the error. For this the Proportional Controller or the P-Controller is used. The constant of proportionality is Kp.

D-CONTROLLER

Due to the P-Controller, the robot would approach the line, but due to the initial direction of velocity, it wouldn't stop as soon as it reaches the line, but would slightly overshoot. The robot then stops and comes back towards the line. Again due to the direction of velocity, the robot would again overshoot. The process goes on and on, and the robot would never move in the straight line. To solve this problem, the Differential Controller or the D-Controller is used. The constant of proportionality is Kd.

I-CONTROLLER

Sometimes, due to in-built errors the robot doesn't move properly. For rectifying these errors the Integral Controller or the I-Controller is used. The constant of proportionality is Ki.

The signal (u) just past the controller is now equal to the proportional gain (Kp) times the magnitude of the error plus the integral gain (Ki) times the integral of the error plus the derivative gain (Kd) times the derivative of the error.

The constant Kp would decrease the steady-state error but would never be able to eliminate it. The constant Ki would eliminate the steady-state error but it may worsen the transient response. The constant Kd would increase the stability of the system, reduce the overshoot and would improve the transient response.

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR

Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	Small Change

4. PROCEDURE

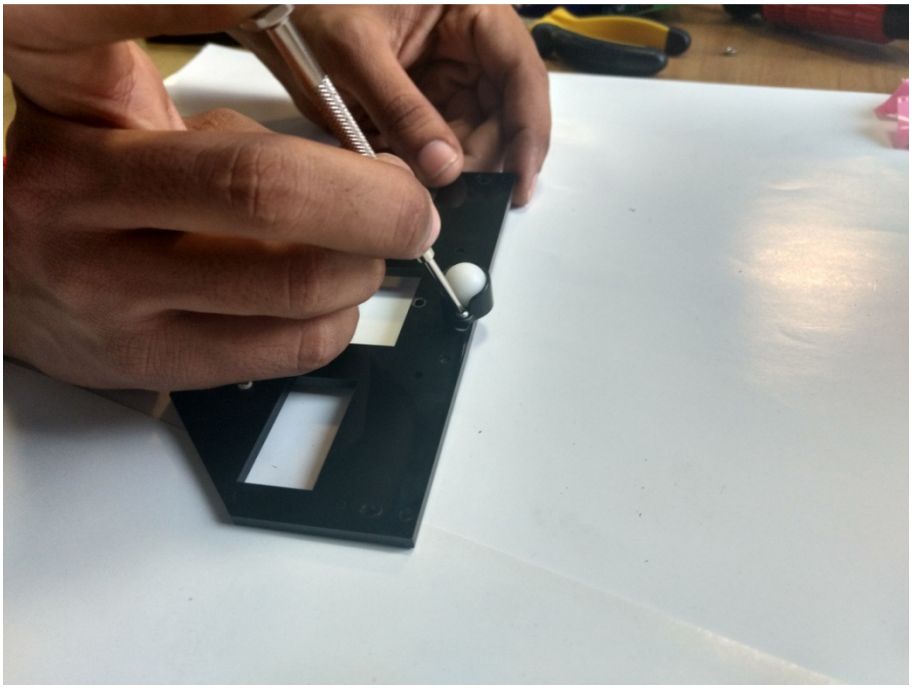
We have now understood the underlying concepts of science and electronics required to make this LFR. Therefore we are now ready to start constructing the LFR.

4.1 Construction

Part-1: Assembling the Robot

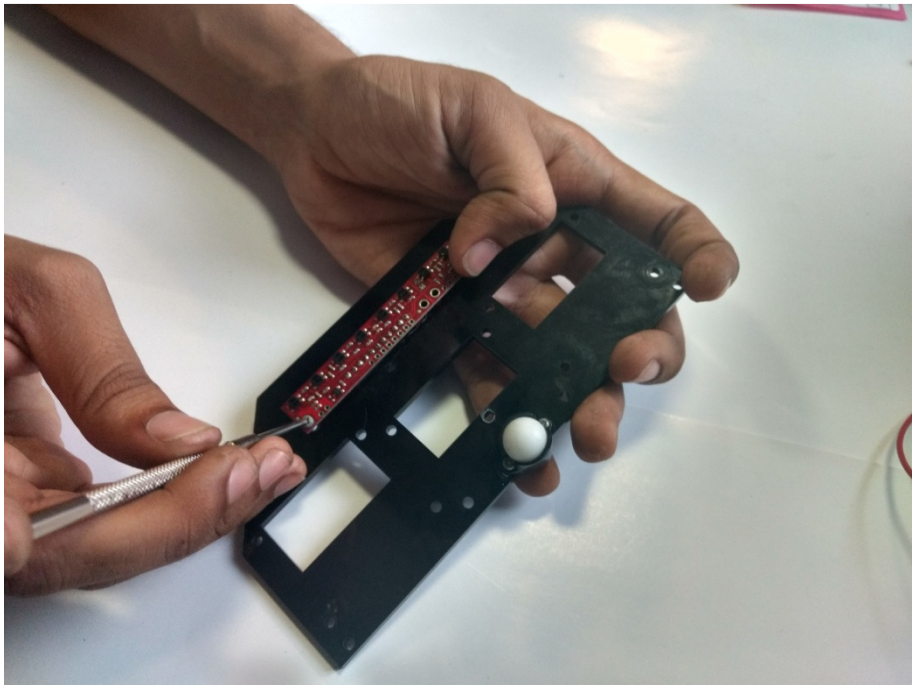
- ✓ **Step – 1**

Attach a caster wheel to the lower chassis plate.



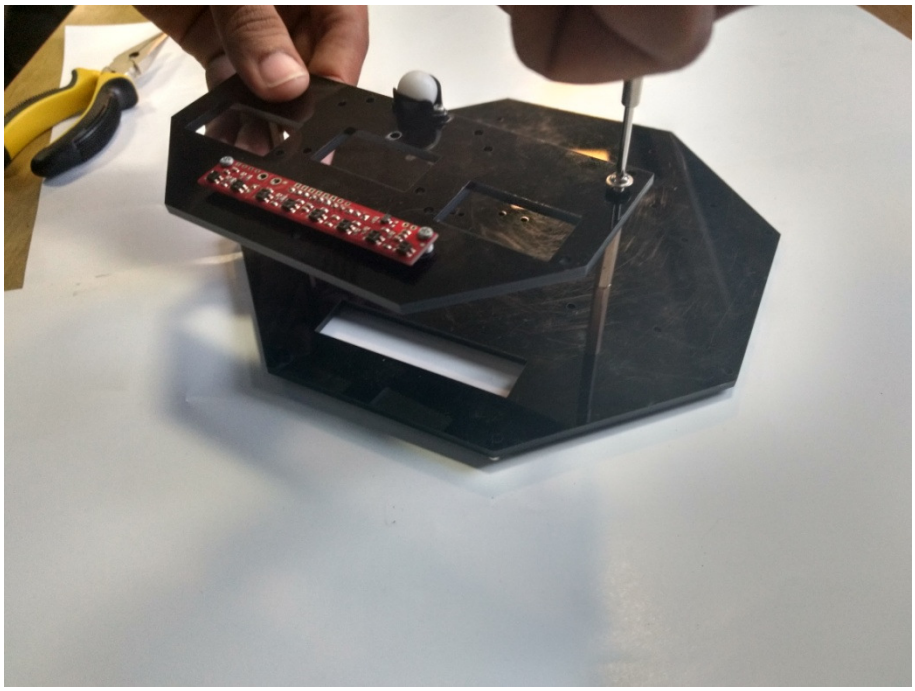
✓ **Step – 2**

Fix the QTR sensor to the lower chassis plate so that the IR emitters face downwards



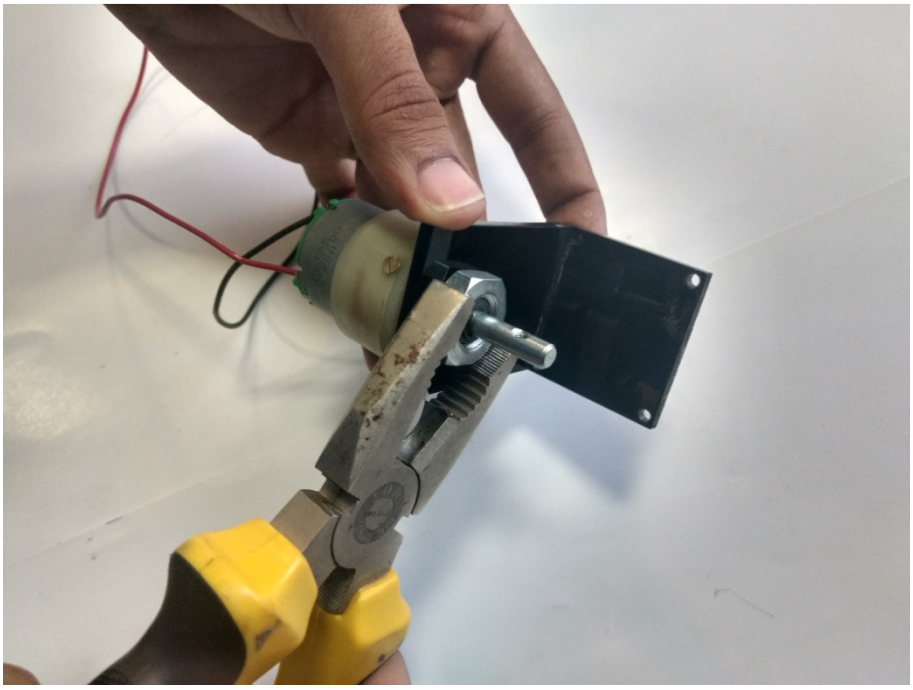
✓ **Step – 3**

Join the upper plate of chassis to the lower plate by 2 rod screw. Make sure that both QTR sensors and caster wheel face at bottom of robot.



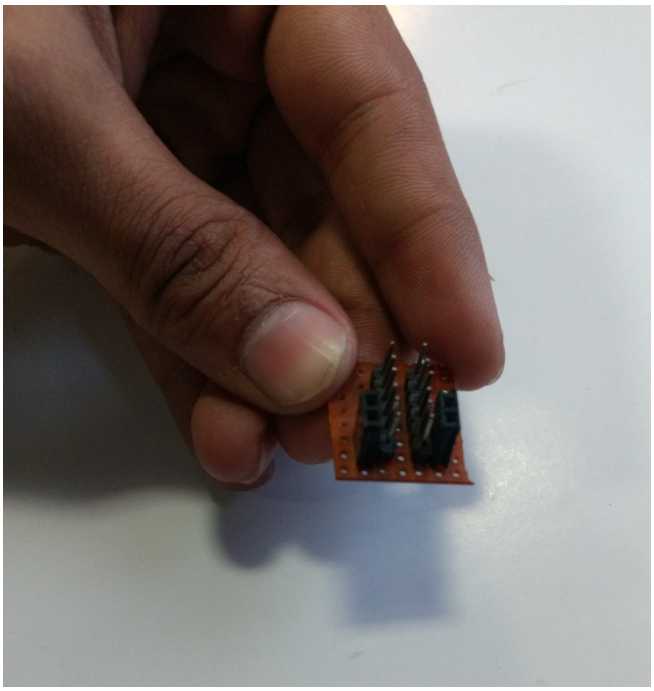
✓ **Step – 4**

Attach wheels to its holder then fix them under the upper chassis plate by screws.



✓ **Step – 5**

Break off two set of one 6-head pins and one 2-slot pins each. They will be our common ground and 5V.

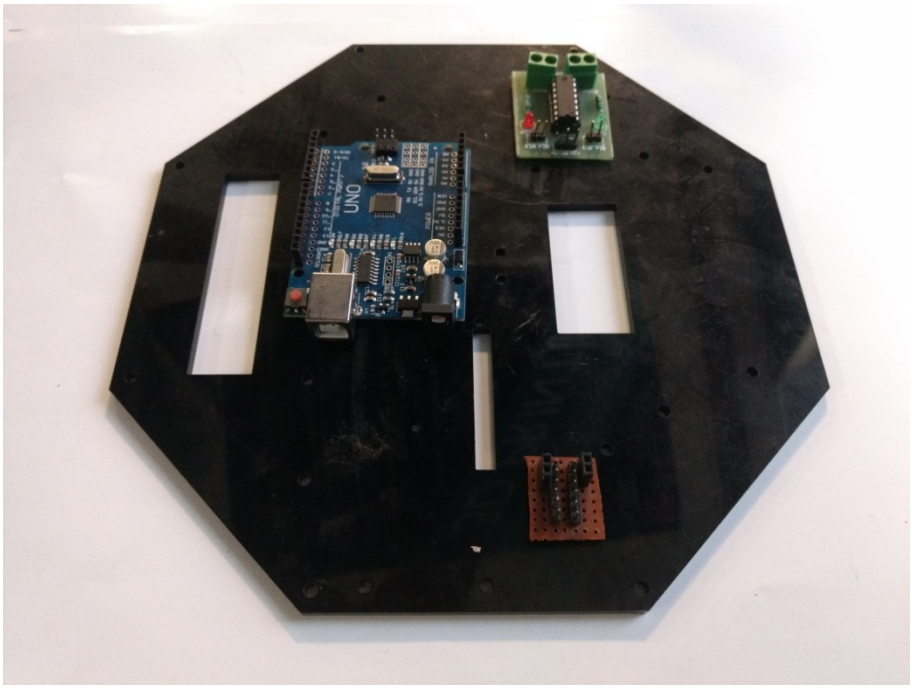


✓ **Step – 6**

Affix both the sets on a puff-board and solder them. Make sure that the two sets don't connect in themselves. For future convenience, mark the sets as Ground (β) and 5V (α).

✓ **Step – 7**

Affix the Arduino, H-Bridge and Puff-board on top of the robot using nut bolts or double sided tape.



Part-2: Wiring the Robot

✓ **Step – 1**

Connect the QTR sensor to the Arduino using jumper wires. Connect the wires by referring to the following table:

Gnd	V_{cc}	V_{cc}	S1	S2	S3	S4	S5	S6	S7	S8
β	5V	none	A5	A4	A3	A2	A1	A0	D2	D4

where Ai or Di are analog and digital pins on the Arduino board.

- ✓ **Step – 2**

Connect the H-Bridge to the Arduino using jumper wires. Connect the wires by referring to the following table:

M1.a	M1.b	V_{cc} 12	Gnd	V_{cc} 5	M2.a	M2.b
D5	D6	α	β	5V	D10	D11

- ✓ **Step – 3**

Connect the V_{IN} and Gnd of Arduino board to α and β of puff-board.

- ✓ **Step – 4**

Connect the + and – terminals of battery to the α and β respectively.

Your LFR is ready to be Coded upon.

4.2 Coding

Since we are already familiarised with the Arduino IDE, we will directly come to the coding part.

- ✓ **Step – 1**

Open the Arduino IDE.

- ✓ **Step – 2**

Paste the following code in a blank sketch. It is recommended that you go through the code once so that you can understand what is happening. All concepts used in the code are covered in *THEORY* section (though you will need basic prior knowledge of any of the following language: C, C++ or embedded C).

```
/* THIS IS THE CODE FOR A LINE FOLLOWING ROBOT USING THE PID CONTROL LOOP */

#include <QTRSensors.h>          //library for QTR sensors

//defining pins numbers as keywords
#define M1_a 7
#define M1_b 6
#define M2_a 10
#define M2_b 9

float param[3]={0.7,0.1,0};   // param vector stores the default PID constants
int current_time=0;
int last_time=0;

QTRSensorsRC qtr((unsigned char[]) {A5, A4, A3, A2, A1, A0, 2, 4}, 8, 2500);

//function 'setup' runs only once
void setup()
{
  pinMode(M1_a, OUTPUT);
  pinMode(M1_b, OUTPUT);
  pinMode(M2_a, OUTPUT);
  pinMode(M2_b, OUTPUT);
```

```
  Serial.begin(9600);

  for (int i = 0; i < 100; i++)  // make the calibration take about 5 seconds
  {
    qtr.calibrate();
    delay(20);
  }
}

int error=0;
int lasterror=0;
int derivative=0;
int integral=0;
int maximum=150;
int diff_time;

//function 'loop' runs infinitely until you stop the robot
void loop()
{

   unsigned int sensors[8] ;
   int position = qtr.readLine(sensors);

   //implementation of PID
   error = position - 3500;    // calculating the Kp
   current_time = millis();    // millis function gives the current time
   diff_time = current_time - last_time;
   derivative = ( error - lasterror ) / diff_time;   // finding the rate of change of error
   last_time = current_time;
   integral += error;     // calculating the summation of all the errors occurred during the run ( Ki )

   //PID equation
   int power_difference = param[0]*error + param[1]*derivative + param[2]*integral;


   const int maximum = 100;


   if (power_difference > maximum)
     power_difference = maximum;
   if (power_difference < -maximum)
     power_difference = -maximum;
```

```
  //for movement of motors

  if (power_difference < 0)      // (power_difference < 0) means go left
  {
    analogWrite(M1_a, maximum + power_difference);
    analogWrite(M1_b, 0);
    analogWrite(M2_a, maximum - power_difference);
    analogWrite(M2_b, 0);
  }
  else                        // else go right
  {
    analogWrite(M1_a, maximum + power_difference);
    analogWrite(M1_b, 0);
    analogWrite(M2_a, maximum - power_difference);
    analogWrite(M2_b, 0);
  }

}
```

- ✓ **Step – 3**

Add QTRSensors.h library. To use QTR sensors we call functions defined in QTRSensors.h. To add a library in Arduino IDE, just download any library from internet and copy them to library folder where your Arduino IDE is installed.

- ✓ **Step – 4**

Connect Arduino to your system by an USB cable. Make sure you select correct Port in TOOLS on the menu bar of IDE.

- ✓ **Step – 5**

Build and Upload your code to your Robot.

Your LFR is ready to be Tested for run.

4.3 Test Run

- ✓ **Step – 1**

Switch on your Robot. As soon as you do that Robot starts to calibrate for 5 seconds.

- ✓ **Step – 2**

Calibrate your LFR. Put your robot on the surface you want to test then move it manually such that every sensor on QTR reads white line at least thrice. Make sure your Robot does not lift from surface as it may disturb calibration process.

- ✓ **Step – 3**

Set your PID constant. It is unlikely that the default constants given in the code are going to be correct. You have to set them on experimental basis using the concepts you learned the *THOERY* section.

Once PID constants are set correctly, sit back and watch your LFR run on a black line on white surface.

5. REFERENCES

Ref. 1 – Arduino Projects; by J. Robert (www.instructables.com/id/Arduino-Projects/)

Ref. 2 – Introduction to Autonomous Mobile Robots; by R. Siegwart

Ref. 3 – Linear Algebra and its Applications; by David C. Lay

Ref. 4 – Advanced Line Following Robot
*(*http://www.societyofrobots.com/member_tutorials/book/export/html/350 *)*

Ref. 5 – Thomas' Calculus (12th edition)