# PID Tutorials for Line Following

sensor, Arduino, pololu, tutorial, line, array, pid, QTR-8RC, Kp, Kd, Tuning
Blog entry
Enigmerald's blog by Enigmerald Collected by 6 users
By Enigmerald
January 6, 2014

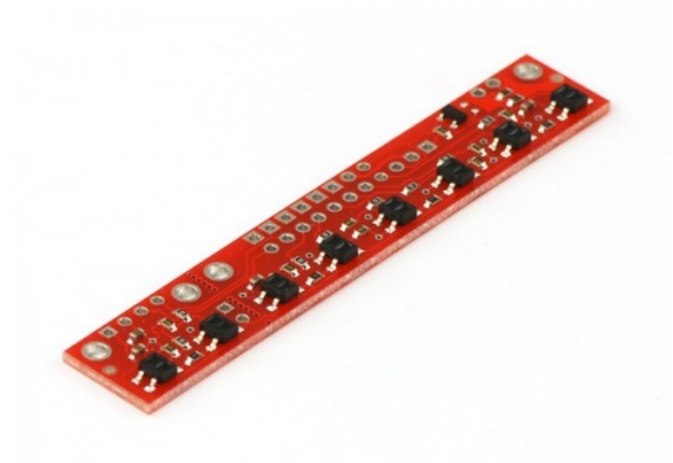
*Hey there guys. This post is outdated. Visit http://letsmakerobots.com/node/39972 for the updated tutorial.*

Hello LMRians,

In this writeup, I will try to create a tutorial for tuning your robot using PID to follow a line. This tutorial won't go deep down into the details of PID and its applications but will just attempt to show how a robot can be tuned with the PID parameters to follow a line.

In my PID based line follower, to remove any sort of difficulties and complexities, I chose the Pololu QTR-8RC array sensor. In this tutorial as well, I will hover around this specific line sensor and how it's readings can be integrated into your own PID line following code.

The Line Following Sensor : Pololu QTR-8RC Array Sensor



The Pololu QTR-8RC sensor trims the fat when it comes to PID based line following. It not only signals you as to where the lines are located, but in turn, it also outputs the position of the robot while following a track.

**KEY TERMS IN PID BASED LINE FOLLOWING:**

PID stands for Proportional, Integral and Derivative. In this tutorial, I am taking a few peeks at Pololu's documents. As stated there :

- The **proportional** value is approximately proportional to your robot's position with respect to the line. That is, if your robot is precisely centered on the line, we expect a proportional value of exactly 0.

- The **integral** value records the history of your robot's motion: it is a sum of all of the values of the proportional term that were recorded since the robot started running.

The **derivative** is the rate of change of the proportional value.

In this tutorial, we will talk about just the Kp term and the Kd term, however, results can be accomplished using the Ki term as well. Perfectly tuning the Kp and Kd terms should be just enough though.

There are two important sensor readings to look into, when implementing PID. However, these readings are not simply the classic "analogRead(leftFarSensor)" type of readings. These readings are not only the analog readings, but also the positional readings of the robot. ( Let's not get into the technicalities of the reading methods of the Pololu Array Sensor. If you are interested, it is well documented on the Pololu website. The basic thing is : **This sensor provides values from 0 to 2500 ranging from maximum reflectance to minimum reflectance, but, at the same time, also provides information on how far the robot has stranded from the line.** )

**Setpoint Value:**

This is our goal. It is what we are hunting for. The setpoint value is the reading that corresponds to the "perfect" placement of sensors on top of the lines. By perfect, I am referring to the moment when the line is exactly at the center of our sensor. I will add on this later. If the robot is perfectly tuned, then it is capable of positioning itself throughout the course with the setpoint value.

**Current Value:**

The current value is obviously the instantaneous readings of the sensor. For eg : If you are using this array sensor and are making use of 6 sensors, you will receive a positional reading of 2500 if you are spot on, around 0 if you are far too left from the line and around 5000 if you are far too right.

**Error:**

This is the difference of the two values. Our goal is to make the error zero. Then only can the robot smoothly follow the line. Hopefully, this term will get more clear as we move forward.

**THE PID EQUATION:**

Wikipedia provides a number equations that all refer to the applications of PID. They might look complex and hard-to-understand. However, Line following is one of the simple applications of PID, thus, the equations used are not very daunting after all.

**1) The first task is to calculate the error.**

**Error = Setpoint Value - Current Value = 2500 - position**

As you can see, I have made use of 6 sensors. Our goal is to achieve a state of zero-error in the PID equation. Just imagine, as I stated earlier, the sensor gives a positional reading of 2500 when the robot is perfectly placed. Substituting 2500 as the position in the above equation we get,

**Error = 2500 - position = 2500 - 2500 = 0**

Clearly, 2500 is the position that zeros out the equation and is thus, our **Setpoint Value.**

(Note : You may notice, what if the sensor is to the far left? The positional reading would then be 0. Would that result in an error of 2500? Correct. And that error of 2500 is taken into account, and necessary changes are made in the next equation )

**2) The second task is to determine the adjusted speeds of the motors.**

**MotorSpeed = Kp * Error + Kd * ( Error - LastError );**

**LastError = Error;**

**RightMotorSpeed = RightBaseSpeed + MotorSpeed;**

**LeftMotorSpeed = LeftBaseSpeed - MotorSpeed;**

Now that we have calculated our error, the margin by which our robot drifts across the track, it is time for us to scrutinize the error and adjust the motor speeds accordingly. Logically speaking, an error of 0 means our robot is out to the left, which means that our robot needs to go a bit right, which inturn means, the right motor needs to slow down and the left motor needs to speed up. **THIS IS PID!** It is basically just an implementation of the general rules of line following in real-time, in high-definition, and in split-seconds!

The **MotorSpeed** value is determined from the equation itself. **RightBaseSpeed** and **LeftBaseSpeed** are the speedsat which the robot runs at when the error is zero. Any PWM value from 0 -255 should do. ( If you are using an 8-bit PWM microcontroller that is :P )

So this is what we have so far :

```
1  int error = position - 2500;
2
3  int motorSpeed = Kp * error + Kd * (error - lastError);
4  lastError = error;
5
6  int rightMotorSpeed = rightBaseSpeed + motorSpeed;
7  int leftMotorSpeed = leftBaseSpeed - motorSpeed;
```

( I have become a fan of this beautifier. Thank you Ladvien Sir! )

**Troubles you may run into:**


- LMR on Facebook
- LMR on Twitter
- Creative Commons License Idea
- LMR Scrapbook
- User list
- Unread posts
- RobotShop Rover Development
- RSS feeds
- Spam Control

As I stated earlier, PID is a super hero and can be a mega-villain as well. It can deceive you with the erroneous responses but at the same time, it's accurate results will WOW you, if it gets implemented correctly. These are a few troubles I personally ran into while building my line follower.

**1) Errors in Sign :**

This is the most dangerous of all the errors you may encounter. A small sign error in any part of the PID equation could spell trouble. In this example, I have used the equation **error = 2500 - position** . This may not work for you. This works for a line follower following a white line on a black background. If you have a black line on a white background, you might need to set it to **error = position - 2500** and might also need to make other changes in the signs in the PID equations.

If you are finding trouble to adjust your robot, an another option may include carrying your robot above a certain height, just to determine if the wheels are spinning in the correct directions. For eg : If the robot is at a position of 5000, the robot is to the far right. However, if the wheels are still spinning in such a way that the robot is still tending to move to the right, there is a sign error in your second equation.

**RightMotorSpeed = RightBaseSpeed + MotorSpeed; ( + might need to be changed to - )**

**LeftMotorSpeed = LeftBaseSpeed - MotorSpeed; ( - might need to be changed to + )**

**2) Troubles with finding the PID parameter values :**

I have read many comments on many PID line following projects and the most popular question that stands out is : **"Please provide me the Kp, Ki and Kd values!!!"**

Really? It's a great experience, if not the best, in finding them. You falter and falter and falter, and finally, you will have something running at the end. So give it a try yourself!

Whatever you do and however you forge ahead with tuning the PID parameters, I suggest to start small. Maybe a PWM value of around 100 should do for the base motor speeds. Then plug in the Kp and Kd terms. As I mentioned way above, the P term is the Proportional and the D term is the Derivative. The derivative of an error is smaller than the error itslef. Thus, to bring about meaningful corrections, it needs to multiplied by a bigger constant and thus, the Kp term is very small in comparison to the Kd term. You could start just with the Kp term, and then when you have something fine, you can add the Kd term and experiment as you move towards your goal of a smooth line follower.

Thanks for reading this tutorial. It's not that well managed and systematic, I guess. I may have made a few errors along the way, so constructive suggestions are highly appreciated. I, myself, am still learning to make my programs look more matured. I will try to make some more updates and hopefully add some more ideas, if I learn them in the days to come. Here is the full PID Line Following code for now : ( Still, there are no values for the constants :P . I'll give you a hint though. Kd is at least 20 times bigger than Kp in my case )

```
#include <QTRSensors.h>

#define Kp 0 // experiment to determine this, start by something small that just makes your bot follow the line at a slow speed
#define Kd 0 // experiment to determine this, slowly increase the speeds and adjust this value. ( Note: Kp < Kd)
#define rightMaxSpeed 200 // max speed of the robot
#define leftMaxSpeed 200 // max speed of the robot
#define rightBaseSpeed 150 // this is the speed at which the motors should spin when the robot is perfectly on the line
#define leftBaseSpeed 150  // this is the speed at which the motors should spin when the robot is perfectly on the line
#define NUM_SENSORS  6     // number of sensors used
#define TIMEOUT       2500  // waits for 2500 us for sensor outputs to go low
#define EMITTER_PIN   2     // emitter is controlled by digital pin 2

#define rightMotor1 3
#define rightMotor2 4
#define rightMotorPWM 5
#define leftMotor1 12
#define leftMotor2 13
#define leftMotorPWM 11
#define motorPower 8

QTRSensorsRC qtrrc((unsigned char[]) {  14, 15, 16, 17, 18, 19} ,NUM_SENSORS, TIMEOUT, EMITTER_PIN); // sensor connected through analog pins A

unsigned int sensorValues[NUM_SENSORS];

void setup()
{
  pinMode(rightMotor1, OUTPUT);
  pinMode(rightMotor2, OUTPUT);
  pinMode(rightMotorPWM, OUTPUT);
  pinMode(leftMotor1, OUTPUT);
  pinMode(leftMotor2, OUTPUT);
  pinMode(leftMotorPWM, OUTPUT);
  pinMode(motorPower, OUTPUT);

  int i;
for (int i = 0; i < 100; i++) // calibrate for sometime by sliding the sensors across the line, or you may use auto-calibration instead

  /* comment this part out for automatic calibration
  if ( i  < 25 || i >= 75 ) // turn to the left and right to expose the sensors to the brightest and darkest readings that may be encountered
     turn_right();
   else
     turn_left(); */
   qtrrc.calibrate();
   delay(20);
wait();
delay(2000); // wait for 2s to position the bot before entering the main loop

    /* comment out for serial printing
```

```
49
50    Serial.begin(9600);
51    for (int i = 0; i < NUM_SENSORS; i++)
52    {
53      Serial.print(qtrrc.calibratedMinimumOn[i]);
54      Serial.print(' ');
55    }
56    Serial.println();
57
58    for (int i = 0; i < NUM_SENSORS; i++)
59    {
60      Serial.print(qtrrc.calibratedMaximumOn[i]);
61      Serial.print(' ');
62    }
63    Serial.println();
64    Serial.println();
65    */
66  }
67
68int lastError = 0;
69
70void loop()
71{
72  unsigned int sensors[6];
73  int position = qtrrc.readLine(sensors); // get calibrated readings along with the line position, refer to the QTR Sensors Arduino Library fo
74  int error = position - 2500;
75
76  int motorSpeed = Kp * error + Kd * (error - lastError);
77  lastError = error;
78
79  int rightMotorSpeed = rightBaseSpeed + motorSpeed;
80  int leftMotorSpeed = leftBaseSpeed - motorSpeed;
81
82    if (rightMotorSpeed > rightMaxSpeed ) rightMotorSpeed = rightMaxSpeed; // prevent the motor from going beyond max speed
83  if (leftMotorSpeed > leftMaxSpeed ) leftMotorSpeed = leftMaxSpeed; // prevent the motor from going beyond max speed
84  if (rightMotorSpeed < 0) rightMotorSpeed = 0; // keep the motor speed positive
85  if (leftMotorSpeed < 0) leftMotorSpeed = 0; // keep the motor speed positive
86
87   {
88  digitalWrite(motorPower, HIGH); // move forward with appropriate speeds
89  digitalWrite(rightMotor1, HIGH);
90  digitalWrite(rightMotor2, LOW);
91  analogWrite(rightMotorPWM, rightMotorSpeed);
92  digitalWrite(motorPower, HIGH);
93  digitalWrite(leftMotor1, HIGH);
94  digitalWrite(leftMotor2, LOW);
95  analogWrite(leftMotorPWM, leftMotorSpeed);
96}
97}
98
99void wait(){
100    digitalWrite(motorPower, LOW);
101}
```

Happy Tuning!

Ashim

### Comment viewing options

Threaded list - expanded ▼ | Date - oldest first ▼ | 10 comments per page ▼ | Save settings

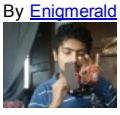
Select your preferred way to display the comments and click "Save settings" to activate your changes.

By lumi @ Tue, 2014-01-07 06:15

**Pretty good tutorial. That**

Pretty good tutorial. That reminds me that I need to dig into the PID a bit more to get some applications running.

By Enigmerald @ Wed, 2014-01-08 07:40



**Thanks Lumi. Can't wait to**

Thanks Lumi. Can't wait to see what you come up with!

By lumi @ Wed, 2014-01-08 09:21

**LOL, it will perhaps happen**

LOL, it will perhaps happen this year :-) So many plans and so little time.

By indrekm @ Mon, 2014-10-20 05:29

**Really good tutorial!**

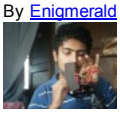
Hi

How behaves your robot in case of  90 degree angle? Radius of angle is 0.

Is the robot able to follow the line.

Best regards.

By Enigmerald @ Fri, 2015-02-20 12:04



**Hello indrekm.A 90 degree**

Hello indrekm.

A 90 degree turn is an extremely sharp turn. The PID parameters need to be tuned to near perfection in order to make a correct turn. But that is rather difficult. So, I suggest you set a 90 degree turn as a "special" case and work out a separate algorithm that varies from the PID routine. A simple "scan" left/right, "turn" left/right should do.

By benbruder @ Mon, 2015-04-13 23:02

**Kp and Kd constants**

Kp = 0.2 and Kd = 5 worked for me as constants!

Let's make robots!