ROBOT RESEARCH LAB

PID Line Follower Tuning

By gberl001

February 16, 2019

This article is a continuation of my previous article on How to Program a Line Following Robot (http://wp.me/p8376Q-8Y) and will cover **PID line follower tuning**. This article assumes you understand what PID is and how to implement a PID or PD control loop. There are some references (such as the goal and setpoint) which are specific to values that you get from using the QTRSensors library from Pololu, you don't have to use their sensors, just their library for some of the references made in this article. Let's learn how to tune a PID line following robot!

Setting up

Work with P only

To tune a PID or any combination of PI and D, I always start with tuning P by itself. The proportional value is going to contribute the most to properly following the line so it is important to get this value correct. If you have I or D included, you don't know if your P is causing the problem or something else so eliminate the other two and stick with just your proportional or KP value.

Speed

One of the biggest mistakes I see is people trying to tune at their full intended speed. Tuning a PID is very difficult at full speed. I always recommend dropping the speed down to at least 50% of your intended full speed unless your full speed is very slow already. You may need to make adjustments to your PID values as you increase the speed but tuning at full speed with already decent values is much easier than tuning at full speed with already decent values. In addition, faster speeds introduce other issues which may make tuning difficult, one issue being traction. If you can't be sure you're getting full traction then you can't rule your KP value as the full contributing factor in your tuning issues. I'm tuning at full speed for a few reasons, effects of PID values show up more with faster speeds (this is why I recommend going slower) so this makes it easier to show on video. Another reason is because my robot is not very fast to begin with.

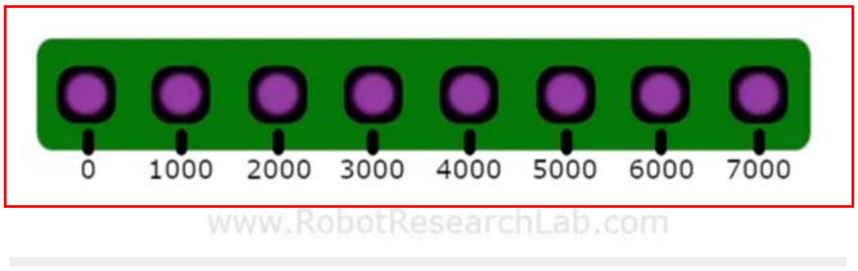
He used 100% for different reasons. But suggests starting at 50% speed

Another note I want to make here is, my speed is 0 to 100 because I'm using a library I wrote (https://github.com/mcc-robotics/Dynamic_Motor_Driver) which works with a few common motor drivers and uses a "power" range of 0 to 100. If I wasn't using a library, I would likely have a range of 0 to 255 which is the full PWM range.

My Initial Speed is $\boxed{100\%}$ (100)

Determining a starting point

UNDERSTAND THE SENSOR RANGE



So, the biggest question I get is, how do I pick a starting value for KP? My method is pretty straight forward and involves just some basic math. The readLine() function from the QTRSensors library returns a value between 0 and 1000*(NUM_SENSORS – 1) okay? So, with my robot I have eight sensors and therefore my readLine() result will be somewhere between 0 and 1000*(8-1) or 7000. Take a look at the image for an example, 0 means the line is under the left most sensor, 1000 means the line is under the second sensor and so on until the eighth sensor which is a value of 7000. The beauty of the readLine() function is it offers an “analog” range, it can have a value of 1200 which means it’s under the second sensor but slightly toward the third. It could have a value of 3538 which means it’s roughly right in the middle of the fourth and fifth sensors.

My Range is 0 to 7000

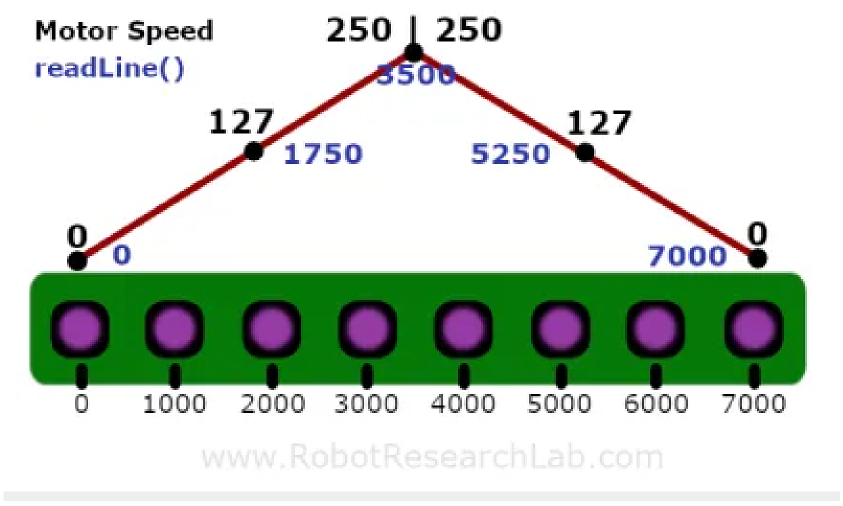
DETERMINE A GOAL

Now that you understand the values coming from readLine(), if I want to follow on center then my goal will be MAX_VALUE/2 or 7000/2 or 3500. That should make sense, since I have eight sensors I’d want to follow between the fourth and fifth sensors for following on center and the value from readLine() between the fourth and fifth sensors will be 3500. Just a note here, I don’t have to follow on center, if there was an obstacle next to the line I could follow to the left or right of center to get past this obstacle, that’s another beauty of PID, it’s easy to change your goal.

My Goal is 3500

DETERMINE YOUR INITIAL KP

Finally, we can figure out a good starting point for our KP value. What I do is I calculate the highest possible error I could get and find the coefficient that converts that max error to a value equal to my speed. To clarify on that a little, here comes some more basic math. Say I drift off the line to the right, the farthest I could get is to the point where readLine() returns a 0 as this is when the left most sensor will see the line. So, my goal (3500) minus this max error is 3500 – 0 or 3500. Same thing going the other direction, if I drift to the left the farthest I could get is to the point where readLine() returns a 7000 as this is when the right most sensor will see the line. So, my goal minus this max is 3500 – 7000 or -3500. In either case my max error is a value of 3500 (disregarding the negative).



Now, if my motor speed is 100 (as in 100%) I come up with a value to convert 3500 to 100. So that you understand why, take a look at the image as you read the next sentence. When the left most sensor or the right most sensor sees the line my adjustment (KP*error) will come out to 100, therefore shutting off the motor if the robot is that far off as we will be subtracting that adjustment from one motor and adding it to the other. To determine a starting KP value, my basic math is as follows.

1. MAX_SPEED = MAX_ERROR * KP
2. Fill in our known values
3. 100 = 3500 * KP
4. To solve for KP, we move the 3500 to the other side and get
5. 100 / 3500 = KP

My KP is 0.028

Just a final note here, I generally only go to the thousandths place, sometimes only to the hundredths but in this case I want the granularity so I'll use the full **0.028**

Let's Get Tuning!

Proceeding into this section you should know the following information

- **Your Sensor Range**
- **Your Goal Position (setpoint)**
- **Your Motor Speed at 50% or less**
- **Your KP computed from above**

After putting my information into the code (an example of which you can find in this article (http://wp.me/p8376Q-8Y)) let's see how my first run goes. Keep in mind we'll need at least two trial runs before we can start making any decisions on how to adjust KP.

Trial 1

Trial one will have our initial KP value of 0.028

KP	0.028
KD	0.0
MAX_SPEED	100
MIN_SPEED	0
SET_SPEED	100

ANALYSIS

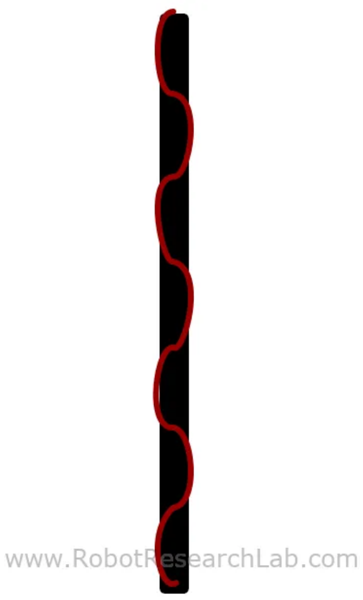
I was very surprised about how accurately my robot followed the line with just an initial KP value. Most likely since I'm used to working with heavier bots so they aren't as responsive as this light little bot. At this point we have nothing to judge by though so I like to go lower and higher to determine which final direction the KP value should go. Our goal KP will give us a line follow pattern that looks like the image you see here. The path should maintain course on the line with a tight pattern and stay consistent.

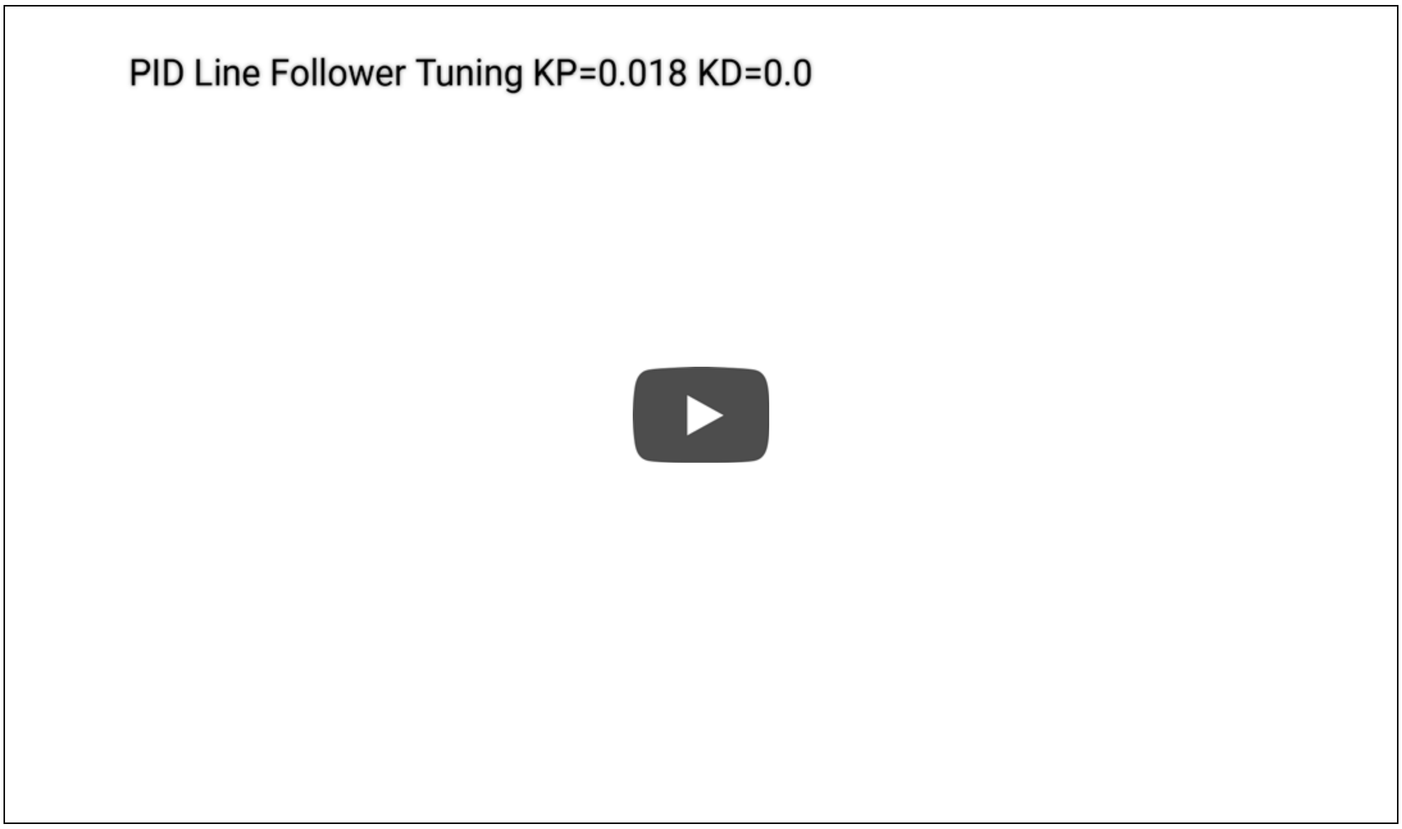
On to trial 2…

Trial 2

First I like to go down in value just to see what would happen. By lowering the KP to 0.018 the motors won't shut off until we reached an error of ~5500 but since our max error is 3500 that means we'll never reach the point of a motor turning off. The max adjustment we would make is 3500*0.018=**63** so our motors will never go below a value of 47 with this KP.

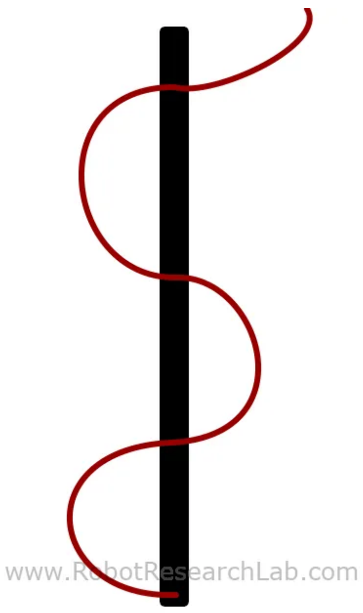
KP	0.018
KD	0.0
MAX_SPEED	100
MIN_SPEED	0
SET_SPEED	100





ANALYSIS

As you can see from the video, the lower KP has increased the amount of sway in the line follow which isn't good, we want something that's going to line follow a little more aggressive than our initial value. Another thing to note, I had to add another piece of "track" since the lower KP allowed my robot to leave the line so much that it would run off the track if I hadn't. A low PD value will give you a pattern like you see in the image, it may or may not be consistent but more importantly, it doesn't follow the line very well. Once you reach a turn, a robot with too low of a KP might completely miss the turn and never make it back to the line.



Trial 3

Now I'll go the same distance from my initial value in the other direction. By looking at trial 1 and trial 2, I should expect that this KP will follow tighter on the line, but will it be too high of a KP?

KP	0.038
KD	0.0
MAX_SPEED	100
MIN_SPEED	0
SET_SPEED	100

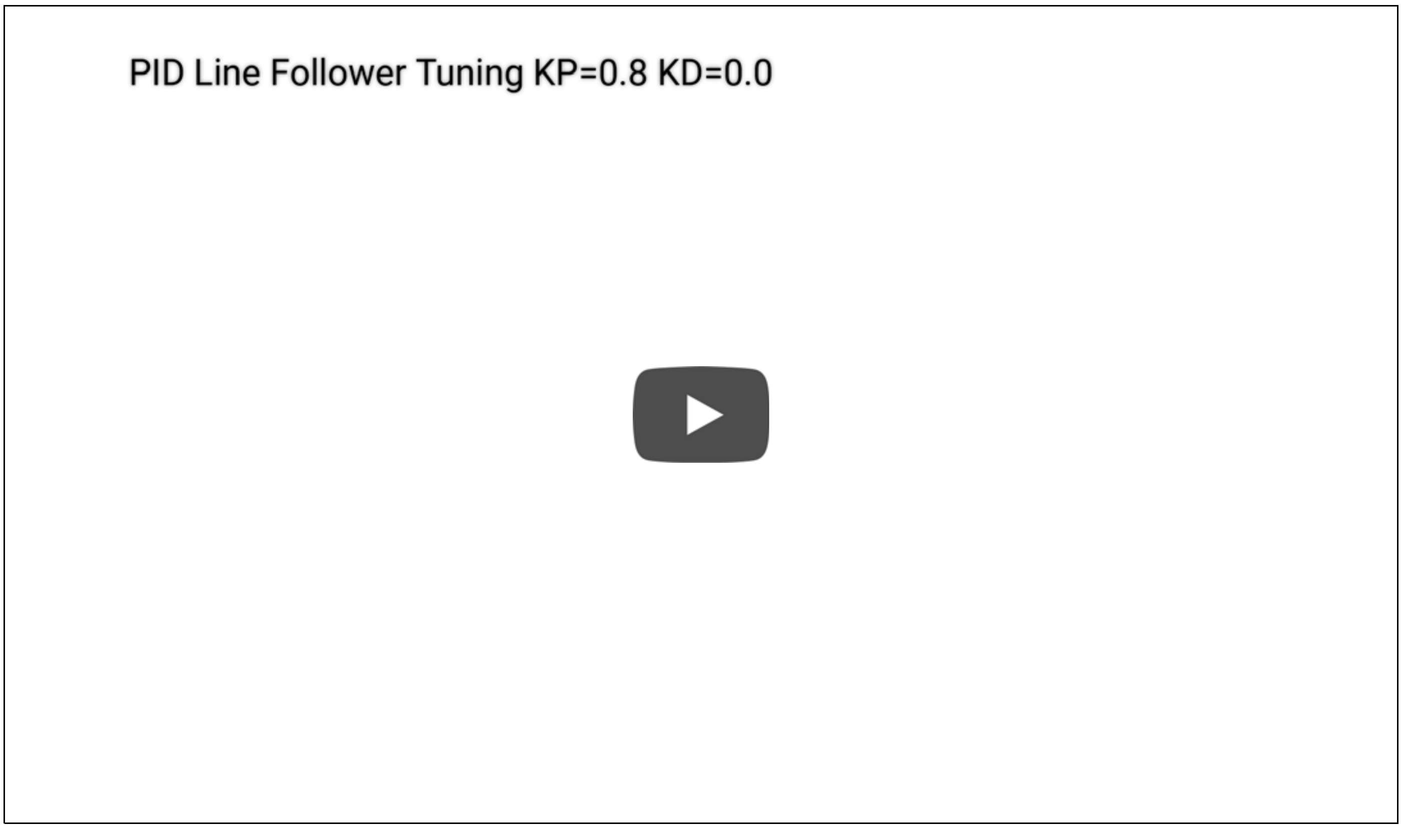
ANALYSIS

Though it may not be as noticeable in the video, this is more aggressive than my initial value. This doesn't overshoot the line which is good so I'm thinking this is the value I'm going to use. If I were just using KP I might stick with my initial 0.028 value but since I'm going to be adding a KD I want it to be a little more aggressive since the KD value will mellow that out a bit.

Trial 4

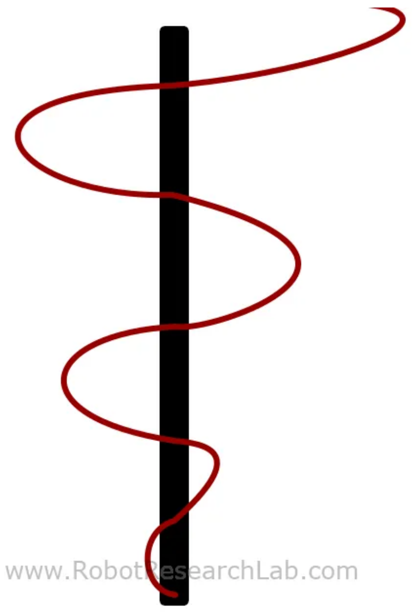
A how-to article wouldn't be complete without including an example of what you'd see with a value that's too high.

KP	0.8
KD	0.0
MAX_SPEED	100
MIN_SPEED	0
SET_SPEED	100



ANALYSIS

As you can see, the robot is overshooting the line left and right (no pun intended). I couldn't get a KP high enough to show you with this robot but with heavier robots it's very common for a KP that is too high to follow for a short period and eventually get so erratic that it shoots off the line completely. This is one way to know your KP is too high, take a look at the image, it shows a possible pattern you'd see with a KP too high.



Trial 5

Generally a good starting point for a KD value is 10-20 times your KP value. So, I've started with a KD of ten times my KP value.

KP	0.038
KD	0.38
MAX_SPEED	100
MIN_SPEED	0
SET_SPEED	100

PID Line Follower Tuning KP=0.038 KD=0.38

ANALYSIS

This is a pretty good value for KD, I might even be inclined to stop here but I do notice a little bit of a delayed response when the robot hits the turns and I know increasing my KD will help with that.

Trial 6

KP	0.038
KD	0.76
MAX_SPEED	100
MIN_SPEED	0
SET_SPEED	100

The Kd has been increased by a factor of 2

ANALYSIS

I don't think I'm going to get much better than this and for the purposes of this robot I don't need to. You may want to spend more time fine tuning your robot, a lot of it is going to come down to how fast you're traveling and what your track looks like. For a track with sharp turns you'll need to tune a lot more than with a track with smooth turns.

Trial 7

As with a high KP, a how-to article wouldn't be complete without an example of a KD that is too high.

KP	0.038
KD	2.76
MAX_SPEED	100
MIN_SPEED	0
SET_SPEED	100

ANALYSIS

It may not be totally obvious from the video but the robot has begun to oscillate again like it did with the high KP, so much so that it increased the lap time by a half a second. That not seem like much but if you're racing your robot that could be the difference between first and second.

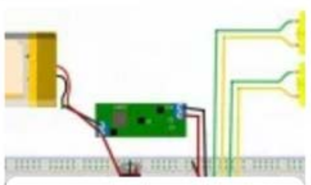
This is one example that goes to show, if you tune both numbers at the same time, it's hard to know which one is the bad one. Since I've tuned my KP separately, I know the culprit is my KD value.

Final Notes

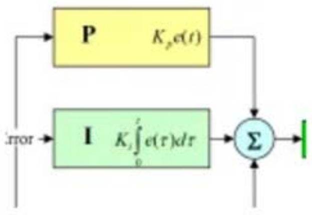
There are some final things I wanted to cover as it's important to know the most common aspects that affect a PID algorithm. Keep an eye out for the following.

- Traction – If you don't have good traction your KP will never allow you to follow the line.
- Speed – If you are going too fast, especially with turns on your track, it will be difficult to tune initial values, this couples with processor speed below
- Sensor position – If the sensor is farther from the wheels you may need a higher KP because the distance from the point of change (fulcrum) is farther so it makes more of an impact.
- Processor speed – yes, this can make a huge difference. If you have a slow processing speed and a fast robot, your sample rate will be very low. For instance, if you travel at 3 ft/s and are able to make adjustments 50 times a second then you are reading the line every 3/4" which isn't too bad. But if you have a slow processor or you have a lot of other things going on in your program and you are only updating 10 times a second, you are reading the line every 3.6", this means you could be 3.6 inches off the line before realizing it. When tuning PID try to keep your code limited to only PID code, avoid print statements as well.

Related Posts:



How to program a motor circuit with the TB67H420FTG



How to Program a Line Following Robot

Multiple Line Following Modes

Build A Custom PID Line Following Robot From Scratch

4 Comments

Bilişim Hocası 7 months ago

Hi, Nice tutorial. You know auto tune pid?

gberl001 7 months ago

Thanks, I know the theory behind auto tune PID but I've never actually implemented one. It's on my list of things to do though. I know that there is a library called PIDAutoTuneLibrary by the same guy who created the popular Arduino PID library. I'm not sure how well that library would work with a line following robot though, seems like it's more for stationary projects like temperature tuning.

Hi sir. I sent you message on Facebook. I don't know if you are using Facebook. Can you check it please?
Thanks…

gberl001 6 months ago

Halil, sorry, I don't really use Facebook. It's there for people I know to get ahold of me but I only check it once every few months. If you want, you can use my email on this site it's my first name at this site (@robotresearchlab.com). You can also use this contact page I just created ha ha http://robotresearchlab.com/contact-me/ (http://robotresearchlab.com/contact-me/) either method will go to the same place. Thanks for visiting.