**Ed’s Robot Project - Week 4**



The robot wars continue. This week the robot won the battle!

I finished the build ok. And built a new test script utilizing the PID formula.

One reference suggested initializing ‘I’ and ‘D’ parameters, the integral and derivative variables, to zero and nail down the ‘P’, proportional, value first.

A few “ideal” values emerged to follow a straight line - depending on the speed of travel. This ‘P’ constant tells the right and left motors how much to adjust their speed to get back to center, where only the center sensor “sees” the black line. The sensors are spaced so that only the center sensor sees the tape when properly aligned. Drifting a little to the left or right will activate two sensors, the center and the adjacent sensor.

**Here are my sensor combinations as the robot tracks from right to left:**

**1 . 0 . 0 . 0 . 0 - the first sensor sees the line**

**1 . 1 . 0 . 0 . 0 – now the first and second sensors see it…**

**0 . 1 . 0 . 0 . 0**

**0 . 1 . 1 . 0 . 0**

**0 . 0 . 1 . 0 . 0**

**0 . 0 . 1 . 1 . 0**

**0 . 0 . 0 . 1 . 0**

**0 . 0 . 0 . 1 . 1**

**0 . 0 . 0 . 0 . 1**

**0 . 0 . 0 . 0 . 0 will tell the robot to reverse**

**1 . 1 . 1 . 1. 1 tells the robot to STOP**

**In the code, the sensor combinations are assigned correction values and a direction:**

**(4, 3, 2, 1, 0, -1, -2, -3, -4) .**

The sensor is on the far right and assigned -4. This value is used as a multiplier for the ‘P’ constant and indicates that we have moved too far left and need to increase power to the left wheels by this amount –- and decrease power to the right wheels to facilitate moving back to center. and used to + 4, meaning. With a good ‘P’ value the robot will pretty much track straight if starting out aligned correctly. If starting out on one of the outside sensors, one or five, the robot will slowly move toward center, overshoot a little one way, then the other and settle in on center. This proportional param indicates how far off track we are and in which direction.

Next is the derivative, ‘D’, constant. This “rate of change” param adjusts the motors further by multiplying this constant by the difference of the current weight and the previous rate. If we were at position ‘-2’, or ‘0.0.0.1.0’, and are now at position ‘-3’, ’0.0.0.1.1’ , we have a change of -1. If we were at -3 and are now at position ‘2’, ‘0.1.0.0.0’, the delta is +5. This param tells us our rate of change and a good constant here will eliminate overshooting our center position and overcorrecting. The effect of good ‘P’ and ‘D’ values is the robot sticks to the line like glue. The PID formula resulted in a great improvement over my original code – with a lot less complexity.

The integral, ‘I’, parameter is the integral portion of the formula. I’ve ignored this param so far. Most texts say to ignore it for line follower applications. I display the value for my own reference, but I have left this constant set at zero for now. The chatter I’ve reviewed online all state that you can leave this out of the equation, for a PD formula rather than PID, but the vales being returned suggest to me that this may be useful. This value represents the constant error of the effectiveness of your PID equation by comparing your previous composite PID values with your resulting position. The last calculation was the attempt to power the motors sufficiently to return the robot to center, so how wrong, how much over or under, was that number – each time so far? This number attempts to correct the formula according by your actual experience. This is the element that gets your elevator to within 3cm of the floor your stopping at regardless of how many people are in it.

While I’ve had a few “Eureka!” realizations that have resulted in positive advances, the performance of my code is still far from satisfactory. The sensors ARE seeing the line moving from position to position – I was afraid that the change was occurring too quickly for it to track even at the slowest speed. AND the robot is able to turn 360 degrees in place – so turning far enuf shouldn’t be the issue.

So… the robot tracks a straight line like a hero but the best constant settings for a straight line do not work for following a turn. Divining new constants that would track a straight line AND make a right turn took forever… and depend on the speed of the robot… and the speed of the robot depends on the battery level. Even worse, the constants required for a left turn are very different than those to make a right turn. Not very constant constants!

These values are way to temperamental for reliable operation and the constants for my specific turns will not likely work for any other radius turn.

Right Turn:

https://youtu.be/nGPhYKB5oww

Left Turn:

https://youtu.be/hNElfGFA8WE

**So, what’s the problem?**

First, I only have five sensors. Although some implementations use five sensors, most use eight. Those that use five increase the sensor resolution further by using the sensor’s analog data to discern the degree of light reflection – how centered is the sensor over the tape? I am digitizing my analog inputs, so I checked this out. My results were not satisfactory. Unfortunately, my analog values are not in the same range as those documents, did not differ significantly positionally, nor were they consistent. My sensor pins are labelled “digital out”. I thought maybe I need actual analog sensors to pull this off. So, I checked (while writing this) and found this on my first “Google”:

Waveshare ITR20001/T Tracker Sensor Module

**$6.86** 

**Waveshare ITR20001/T Tracker Sensor Module Infrared Detector**  
  
**Main Features:**  
● 5-channel ITR20001/T infrared detector, high sensitivity, anti-jamming, stable performance  
● **High-precision analog output, combined with PID control algorithm, allows the robot tracks lines more stable  
● Great for robot line tracking**

**Are you kidding me? Stay tuned!**