SAMVRIT BLOGS

ROBOTICS AND MORE!

HOME

TUTORIALS

MY PROJECTS

ABOUT ME

You are here: Home » Tutorials » PID Control – Arduino Line Follower Robot

Search …

PID CONTROL – ARDUINO LINE FOLLOWER ROBOT

What Is PID?

PID is a simple, yet very effective control method, in which a particular physical/electrical quantity is controlled, and made equal to a set value called the 'set-point'. In this case, we will be controlling the speed of the right and left motors of the robot, so that the robot follows the center of a black line using Arduino. This is done by calculating the amount of deviation of the robot from the center of the line using IR Sensors. If you wish to know how exactly the PID Control works, click here.

Things Required

- 5 Digital IR Sensors
- Arduino Board (Uno or a custom board)
- 2 DC motors
- Motor Diver (if your custom board does not have it already)
- Jumper Wires

Connections

RECENT POSTS

FEST REVOLUTION – ONLINE TICKETING SYSTEM

HOW I GOT INTO ROBOTICS

CATEGORIES

Select Category

CONNECT WITH ME

Facebook
LinkedIn

- Connect the 5 digital IR sensors to the analog inputs (you can still read digital values from analog inputs).
- Connect the Left motor pins to digital pins 4 and 5
- Connect the Right motor pins to digital pins 6 and 7
- Connect the PWM pins to pins 9 and 10

Building the Program

I am going to assume that you have already built your robot. Remember, your actual code should be tweaked and modified from this to work. This exact code might not work exactly how you expect unless you make changes depending on your robot's design and connections. Now let's get coding!

Since we are using digital IR sensors, the output of the sensors will be 0 if it detects white and 1 if it detects black. So, remember these:

Sensor Value	Position
0 0 1 0 0	Center of the line
1 0 0 0 0	Right of the line
0 0 0 0 1	Left of the line

Initialisation

Let's initialize all variables to 0 in the beginning, and let the initial motor speed be 100. Also initialize the ports. You can use the serial monitor to view the instantaneous values of different variables. This will be helpful for tuning.

```
float Kp=0,Ki=0,Kd=0;
float error=0, P=0, I=0, D=0, PID_value=0;
float previous_error=0, previous_I=0;
int sensor[5]={0, 0, 0, 0, 0};
int initial_motor_speed=100;

void read_sensor_values(void);
void calculate_pid(void);
void motor_control(void);

void setup()
{
 pinMode(9,OUTPUT); //PWM Pin 1
 pinMode(10,OUTPUT); //PWM Pin 2
 pinMode(4,OUTPUT); //Left Motor Pin 1
 pinMode(5,OUTPUT); //Left Motor Pin 2
 pinMode(6,OUTPUT); //Right Motor Pin 1
 pinMode(7,OUTPUT);  //Right Motor Pin 2
 Serial.begin(9600); //Enable Serial Communications
}
```

Error Calculation:

We will be using the weighted values method, i.e, we will be assigning a different value for different combinations of sensor values and using it to calculate the deviation from the center. use simple switch condition for

Sensor Array Values	Error Value
0 0 0 0 1	4
0 0 0 1 1	3
0 0 0 1 0	2
0 0 1 1 0	1
0 0 1 0 0	0
0 1 1 0 0	-1
0 1 0 0 0	-2
1 1 0 0 0	-3
1 0 0 0 0	-4
0 0 0 0 0	-5 or 5 (depending on the previous value)

Code for this:

```
void read_sensor_values()
{
  sensor[0]=digitalRead(A0);
  sensor[1]=digitalRead(A1);
  sensor[2]=digitalRead(A2);
  sensor[3]=digitalRead(A3);
  sensor[4]=digitalRead(A4);

  if((sensor[0]==0)&&(sensor[1]==0)&&(sensor[2]==0)&&(ser
  error=4;
  else if((sensor[0]==0)&&(sensor[1]==0)&&(sensor[2]==0)&
  error=3;
  else if((sensor[0]==0)&&(sensor[1]==0)&&(sensor[2]==0)&
  error=2;
  else if((sensor[0]==0)&&(sensor[1]==0)&&(sensor[2]==1)&
  error=1;
  else if((sensor[0]==0)&&(sensor[1]==0)&&(sensor[2]==1)&
  error=0;
  else if((sensor[0]==0)&&(sensor[1]==1)&&(sensor[2]==1)&
  error=-1;
  else if((sensor[0]==0)&&(sensor[1]==1)&&(sensor[2]==0)&
  error=-2;
  else if((sensor[0]==1)&&(sensor[1]==1)&&(sensor[2]==0)&
  error=-3;
  else if((sensor[0]==1)&&(sensor[1]==0)&&(sensor[2]==0)&
  error=-4;
  else if((sensor[0]==0)&&(sensor[1]==0)&&(sensor[2]==0)&
    if(error==-4) error=-5;
    else error=5;

}
```

You can make your own algorithm to make this part of

the code simpler.

Calculating The PID Value

Now this error variable should be made use of by the calculate_pid function, which will calculate the PID value and output it to the motor. The error keeps adding itself to the Integral term during every iteration. Also, the current “error” value should become the “previous_error” value for the next iteration.

Code for this:

```
void calculate_pid()
{
    P = error;
    I = I + error;
    D = error – previous_error;

    PID_value = (Kp*P) + (Ki*I) + (Kd*D);

    previous_error=error;
}
```

Motor Control

The PID_value may be a positive or a negative value. So if it is negative, the left motor speed increases and the right motor speed decreases, and vice versa if the PID_value is positive. The code for this is:

```
void motor_control()
{
    // Calculating the effective motor speed:
    int left_motor_speed = initial_motor_speed-PID_value
    int right_motor_speed = initial_motor_speed+PID_valu

    // The motor speed should not exceed the max PWM valu
    constrain(left_motor_speed,0,255);
    constrain(right_motor_speed,0,255);

    analogWrite(9,left_motor_speed);   //Left Motor Spee
    analogWrite(10,right_motor_speed);  //Right Motor Sp
    //following lines of code are to make the bot move f
    /*The pin numbers and high, low values might be diff
    depending on your connections */
    digitalWrite(4,HIGH);
    digitalWrite(5,LOW);
    digitalWrite(6,LOW);
    digitalWrite(7,HIGH);
}
```

Putting It All Together

Let's put all the functions together in the main loop funtion. Remember, the order in which you put these functions in the loop function is very important. Hence, the code becomes:

```
void loop()
{
    read_sensor_values();
    calculate_pid();
    motor_control();
```

Copy the complete Arduino code.

Tuning the Kp, Ki, Kd Values

This is THE most important part of your program. The PID constants, ie., Kp, Ki and Kd values are tuned only by trial and error method. These values will be different for every robot and for every configuration. Try this method while tuning:

- Start with Kp, Ki and Kd equalling 0 and work with Kp first. Try setting Kp to a value of 1 and observe the robot. The goal is to get the robot to follow the line even if it is very wobbly. If the robot overshoots and loses the line, reduce the Kp value. If the robot cannot navigate a turn or seems sluggish, increase the Kp value.
- Once the robot is able to somewhat follow the line, assign a value of 1 to Kd (skip Ki for the moment). Try increasing this value until you see lesser amount of wobbling.
- Once the robot is fairly stable at following the line, assign a value of 0.5 to 1.0 to Ki. If the Ki value is too high, the robot will jerk left and right quickly. If it is too low, you won't see any perceivable difference. Since Integral is **cumulative, the Ki value has** a significant impact. You may end up adjusting it by .01 increments.

- Once the robot is following the line with good accuracy, you can increase the speed and see if it still is able to follow the line. Speed affects the PID controller and will require retuning as the speed changes.

Lastly, PID doesn't guarantee effective results just by simple implementation of a code, it requires constant tweaking based on the circumstances, once correctly tweaked it yields exceptional results. The PID implementation also involves a settling time, hence effective results can be seen only after a certain time from the start of the run of the robot. Also to obtain a fairly accurate output it is not always necessary to implement all the three expressions of PID. If implementing just PI results yields a good result we can skip the derivative part.

GOOD LUCK!!

64 THOUGHTS ON "PID CONTROL – ARDUINO LINE FOLLOWER ROBOT"

ADITHYA
can use IR LED instead of digital IR sensors