Experiment No. 2
Line Follower Bot
EE 324: Control Systems Lab

Madan Y N 200070040
Hardik Panchal 200070054
M Shashank Balaji 200070043

Contents

1 Objective

1. Design and implement a PID controller for the Spark V robot to make it follow a continuous track, using the IR sensors provided on the robot for this purpose.
2. Ensure that the system meets the design constraint to trace the track within 30 seconds.

2 Control Algorithm

2.1 PID controller

A proportional–integral–derivative controller (PID controller) is a control loop mechanism employing feedback. A PID controller continuously calculates the error value e(t) i.e. difference between desired set point and measured variable, and applies a correction based on proportional(P), integral(I), and derivative(D) terms. Mathematically, the control function c(t) can be expressed as

$$c(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt}$$

Parameters K_p, K_i, K_d can be tuned to a particular system and particular design requirement

2.2 Main Idea

We were supposed to control the velocity of the robot using a PID controller and using velocity control, make the robot follow the given path. The path is recognized by IR sensors which give high value when they are right above path and low reading otherwise. We tuned the maximum and minimum of the IR sensor output value using potentiometer and normalized the potentiometer readings to $[0, 1]$. We then assigned weights $0, 10, 20$ to left, center, right potentiometers respectively. This weighted sum is a measure of robot's orientation with respect to the path to be followed. We use the PID to make sure that the weighted sum is always close to 10 which ensures that the robot is on the right path

2.3 Main Code

```
int main(void)
{
    init_devices();
    port_init();
    lcd_set_4bit();
    lcd_init();

    max_l = 110;
    max_r = 76;
    max_c = 76;
    min_l = 6;
    min_r = 6;
    min_c = 6;
    const unsigned basespeeda = 70;
    const unsigned basespeedb = 70;
    Kp = 12;
    Ki = 0;
    Kd = 20;

    while(1)
    {
            forward();
            l=ADC_Conversion(3);
            c=ADC_Conversion(4);
            r=ADC_Conversion(5);
            if(l<10 && r<10 && c<10){
                        velocity(100,100);
                        back();
                        _delay_ms(150);
                }
        l = (l-min_l)/(max_l - min_l);
        r = (r-min_r)/(max_r - min_r);
        c = (c-min_c)/(max_c - min_c);
        position = (l*0 + c*10 + r*20)/(l+c+r);
        // left -> 0
        // centre -> 10
        // right -> 20 --- values of position
        error = 10 - position; //10 is the ideal position (the centre)
        P = error;
        I = I + error;
        D = error - lastError;
        lastError = error;
        int motorspeed = P*Kp + I*Ki + D*Kd;
        //calculate the correction needed to be
        applied to the speed

        int motorspeeda = basespeeda + motorspeed;
        int motorspeedb = basespeedb - motorspeed;
                if (motorspeeda < 0) {
                        motorspeeda=0;
        }
        if (motorspeedb < 0) {
                    motorspeedb=0;
                    }
        forward();
                velocity(motorspeedb,motorspeeda);
                _delay_ms(1);
    }
```

2.4 Parameter tuning

There are some constraints given, the PID controller had to be appropriately tuned so as to ensure that performance of the system is within the specified constraints. The system has 3 tunable parameters, namely Kp, Kd and Ki, each of which affects a different part of the system's response. Our approach to stabilize the system in three steps is as follows,

1. Initially we kept K_i & K_d as 0 and increased just K_p value. We know that K_p determines how quick the system's response is thus it determines the rise-time. We increased K_p so as to bring t_r within 0.5s. But we found that the rise time wasn't getting affected after a certain threshold. Our rise time was staying around 0.7 sec no matter the increase we made to K_p. So, we had to stay with this rise time. This maybe due to the plant(motor) transfer function. However, there was slight increase in overshoot time as we increased K_p. So, we stopped at optimal value and went on to tune K_d and K_i to meet other requirements.

2. Once K_p value was set, we increased K_d by keeping other gains constant in order to make the turns smooth as using only K_p, the system was like giving aggressive response to the obtained error so to add some smoothing we need some non-zero K_d. After a certain value of K_d the turns were becoming very slow and ultra cautious which was not needed as we need to meet the 30 seconds criteria. So, we finally settled to the value if K_d as 20. then we finally went onto tune K_i to get acceptable steady state error.

3. Finally, for K_i we tried to tune it by increasing it slowly keeping the other gains fixed so as to achieve a reasonable value of the steady state error, But we were already getting the very good performance without using K_i, so we kept the value of K_i at 0. we tried to increase it slightly but it increased the error instead of reducing it, so we stopped there and noted all three parameters.

3 Challenges Faced and their Solution

The main challenge was to remove the drift which bots had as one wheel was running fast and one was slow at forward command so it was drifting towards one direction due to that. We tried many bots and luckily got one bot which had no issue of that kind. The other issue was to tune the pots to make the readings of sensors appropriate, but most of the bots were giving minimum reading of 6 to 8 which was fine but the maximum reading in the black region of the track was maximum 20 to 30 at some parts and around 60 to 70 in some parts so tuning them was painful task but we got some stable values with the bot which we used. Mostly finding a good bot and tuning it appropriately was the biggest challenge. Writing the code after getting the basic idea was not much difficult.

4 Results

The values obtained after tuning the values of three parameters are given below. We also have the final tuned K_p, K_d and K_i values in table below.

4.1 Tuned gains

Tuned gain	**value**
K_p	12
K_d	20
K_i	0

4.2 Observed parameters

Parameter	**Target**	**Value obtained**
Track tracing time(t)	30 s	30.67 s

5 Observations and Inference

This experiment involved a very good practical experience of applying the concepts of PID controller learned in class. The integration of line follower bot control was quite a nice task with a particular set of design specifications. The designed controller was responsible to keep the bot on the track on all possible types of paths like turns or straight etc. and this was performed through a simple feedback system and parameter tuning via repeated output observation. The final parameters obtained satisfied all the specified design constraints and bot was able to follow the black path in entire track very faithfully.