EE 324 : Lab 2 Report

Line Follower Robot

Bhavik Yadav(21d070090)
Sujeet Mehta(21d070076)
Jayveer Singh Sikarwar(21d070033)

October 4, 2023

Batch no. 12

1 Objective

- To design and implement a PID controller for the Spark V robot
- To make robot follow a continuous track, using the IR sensors provided on the robot for this purpose and can able to complete path within 30seconds without any deviation from target direction

2 Introduction

- The robot is built around the ATMEGA16 microcontroller. It is programmed using the AVR Programmer tool in Embedded C/C++.
- One of the standout features of Spark V is its ability to move in eight different directions, all of which can be easily adjusted by modifying the values of the relevant registers. These 8 are Forward, Reverse, Right, Left, Soft Right, Soft Left, Soft Right 2 and Soft Left 2.
- Furthermore, Spark V offers precise speed control using Pulse Width Modulation (PWM). This feature allows users to fine-tune the robot's speed, enabling it to move at different velocities for various tasks and applications.

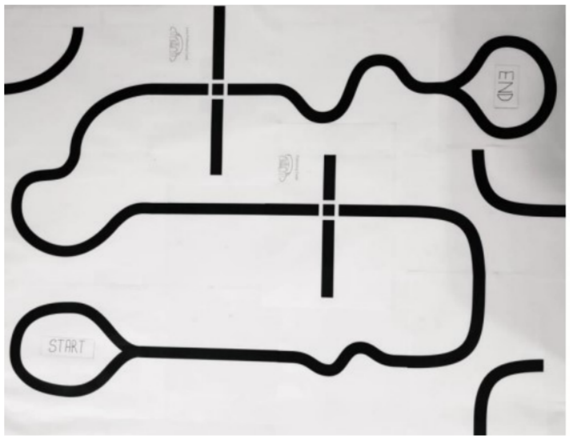

Figure 1: This is the track that bot had to follow

- In terms of sensing capabilities, Spark V is equipped with three Infrared (IR) sensors located under its panel. Each sensor provides readings ranging from 0 to 255, which correspond to the reflectivity of the surface it is detecting. These sensors enhance the robot's ability to navigate its environment by providing data on nearby objects and obstacles.
- Initially the bot needed to be tuned to read the correct values from the 3 Infrared sensors attached at the base corresponding to the left, right and center directions.
- After the bot read correct sensors values for the white and black regions on the board, these values were used to implement the PID algorithm for follwing the line.

3 Control Algorithm

- Initially the bot needed to be tuned to read the correct values from the 3 Infrared sensors attached at the base corresponding to the left, right and center directions. After the bot read correct sensors values for the white and black regions on the board, these values were used to implement the PID algorithm for follwing the line.
- The main function code is given here-

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "lcd.c"
```

```
#define RS 0
#define RW 1
#define EN 2
#define lcd_port PORTC
#define sbit(reg,bit)    reg |= (1<<bit)
#define cbit(reg,bit)    reg &= ~(1<<bit)
void motion_pin_config (void)
{
    DDRB = DDRB | 0x0F;   //set direction of the PORTB3 to PORTB0 pins as output
    PORTB = PORTB & 0xF0; // set initial value of the PORTB3 to PORTB0 pins to logic 0
    DDRD = DDRD | 0x30;   //Setting PD4 and PD5 pins as output for PWM generation
    PORTD = PORTD | 0x30; //PD4 and PD5 pins are for velocity control using PWM
}
int main()
{
    init_devices();
    lcd_init();
    lcd_set_4bit();
    unsigned char left_sensor   =0;
    unsigned char right_sensor  =0;
    unsigned char center_sensor =0;
    unsigned char old_left_center=0;
    unsigned char old_right_center=0;

    unsigned char l = 0;
    unsigned char c = 0;
    unsigned char r = 0;

    float Kp                    =19;
    int vel_thresh              =36;
    int pos_thresh              =8;
    int left_center             =0;
    int error                   =0;
    int right_center            =0;
    int Kd                      =8;
    int right_left              =0;
    int prev_error              =0;
    int ki                      =0;
    int integral                =0;
    while(1)
    {
        l=ADC_Conversion(3);
        c=ADC_Conversion(4);
        r=ADC_Conversion(5);
        lcd_print(1, 1, l, 3);
```

```
 lcd_print(1, 5, c, 3);
 lcd_print(1, 9, r, 3);


 left_center  = l - c;
 right_center = r - c;
 right_left   = r - l;

 /*if (left_center>right_center){
    error = left_center;
 }
 else error = -right_center;*/
 error=-right_left;
 if(c>45) error = 0;


 /*if ((right_center*left_center>=0)) {
     error=0;
     }*/
 float pid_drive = Kp*error+Kd*(error-prev_error)+ki*integral;

 if (error==0) {
     //OCR1AL= 100;
     //OCR1BL = 100;
     forward();
     }
/* else if (l < 7 && c < 7 && r < 7){
     back();
     _delay_ms(300);
     }*/
 else if (l > 6 && c > 6 && r > 6){
     forward();
     }
 else if ((pid_drive<pos_thresh)&&(pid_drive>0)) {
     OCR1AL=0;
     OCR1BL = min(pid_drive+vel_thresh,255);
     soft_left();
     }
 else if ((pid_drive>pos_thresh) &&(pid_drive>0) ) {
     OCR1AL = min(pid_drive+vel_thresh, 255);
     OCR1BL = min(pid_drive+vel_thresh, 255);
     left();
     }
 else if ((pid_drive>-pos_thresh)&&(pid_drive<0)) {
     OCR1AL= min(-pid_drive+vel_thresh,255);
     OCR1BL=0;
```

```
            soft_right();
            }
        else if ((pid_drive<-pos_thresh)&&(pid_drive<0)) {
            OCR1AL = min(-pid_drive+vel_thresh,255);
            OCR1BL = min(-pid_drive+vel_thresh,255);
            right();
            }
        prev_error=error;
        integral += error;
    }
}
```

The PID parameters are defined as follows:

- **Proportional Error:**, as the term itself suggests this response is proportional to the current error, i.e. it multiplies the error with a constant also known as the Proportional Gain (Kp). It is the proportional term which is responsible for providing an immediate response to the current error, which is the difference between the robot's desired position (on the line) and the current position (off the line). if the Kp gain is too high, it can lead to oscillations and instability as the robot might overshoot the line
- **Integral Error:**, the integral term focuses on the accumulation of past errors and aims to eliminate any steady-state errors that may persist. It basically adds up the past errors and multiplies them by a constant (Ki) to produce a correction term. This correction would help the robot deal with biases like uneven motor response and wheel imperfections, it basically would help us counter small and consistent errors.
- **Derivative Error:**, this type of response predicts how the error will change in the future, i.e. it is an estimate of the future trend of the error based on its current rate of change (Kd * rate of change of error). It acts as a dampening mechanism to reduce overshooting and oscillations caused by rapid changes in error. It basically prevents the robot from making sudden and excessive corrections.

4 Challenges faced and their Solutions

We faced various challenges throughout this experiment.

- We had to tune the potentiometers attached to our IR sensors because they were providing readings in a variety of ranges. To prevent favoring of one side and affecting the motion, it was essential to equalize the reading ranges of the right and left sensors. The aim was to ensure that the general ranges of readings from each sensor were similar. To overcome this difficulty, we continued to fine-tune our IR sensors using the potentiometer in order to achieve ideal sensing and proper PID controller operation

- The LCD on our robot was experiencing consistent issues with its functionality. When it was attached to the main cable, the LED was working fine. But as soon as we detach it from main cable then the LCD was not working. This posed significant challenges in the calibration process of the IR sensors. As a result, we had to change our bot twice.
- After dedicating three lab sessions to our project, our code was ultimately rejected because it did not adhere to the PID control methodology. This setback meant that all the work and progress achieved in those three labs went to waste, compelling us to start from scratch and develop a new codebase.
- We encountered significant challenges during the PID design process, primarily because our robot struggled to consistently follow the path, especially on crosswalks. Sometimes it was moving fine on the cross, but sometimes it was moving left or right randomly. We tried many things, but it was not working. Then we observed that the condition we were using for cross was slightly wrong as we kept lowest left sensor value to a little high. So we again checked the value of left sensor on black and white and changed the condition accordingly.

5 Results

- The final values of kp, ki and kd that we had were 19, 0 and 8 respectively.
- Our bot completed path in 24 seconds which was 6 secs less than target time without any deviation from decided path.

6 Observations and Inference

- Initial Behavior: In the initial tests, we observed how the robot behaves when following the path. Note any deviations,IR sensor values or steady-state errors.
- PID Parameter Effects: As we adjust the PID parameters (Kp, Ki, and Kd), monitor the bot's response. Noticed how changes in each parameter affect its behavior.
- Stability: Ensure that the robot's movement is stable and does not exhibit deviations . Adjust the parameters accordingly to achieve stability.

- Response Time: Measured the time it took for the robot to start following the path accurately. The goal was to achieve consistent path tracking within the 30-second time-frame.
- Environmental Factors: Be aware of how environmental factors such as lighting conditions, surface quality, and sensor calibration may affect the robot's performance.
- Battery Life: Keep an eye on the robot's battery life as it runs the PID control algorithm. Ensure that it can complete the 30-second task without draining the battery excessively.
- Fine-Tuning: Be prepared to fine-tune the PID parameters iteratively to achieve optimal performance.