Control Systems Lab - Experiment 2
Line Follower Bot

Pulkit Paliwal 20D100021
Rohan Rajesh Kalbag 20D170033
Yuvraj Singh Tanwar 20D100030

October 2022

1 Aim

Design and implement a PID controller for the Spark V robot to make it follow a continuous black track, using the IR sensors provided on the robot for this purpose. Additionally, to tune the parameters such that the track is traced within 30s.

2 Introduction

Spark V is a low cost robot designed for robotics hobbyists and enthusiasts.

Spark V robot is based on ATMEGA16 microcontroller. It is programmed using AVR Programmer tool in Embedded C/C++.

It allows movement in 8 possible ways which can be adjusted by changing the value of the registers. These 8 are FWD, REV, RIGHT, LEFT, SOFT RIGHT, SOFT LEFT, SOFT RIGHT 2, SOFT LEFT 2.

Additionally the Spark V allows speed control using PWM. It has 3 IR sensors under its panel, each of which give reading from 0-255 based on the reflectivity of the surface it is held against.

3 Procedure

- To get an understanding of the hardware provided, first we went through the Hardware and Software manuals of Spark V provided with the handout.
- To get an idea of the values that the robot sensors produce, a test code just printing the values of the sensors was created. We noticed that the sensor values increase when placed in front of a non reflecting (black) surface. We also saw that the values of the sensors increased on rotating the potentiometer rightwards for all of the sensors.

- Our first iteration included making the robot turn right/left if the value of (left sensor - right sensor) is greater/lesser a given threshold and was implemented through a long decision tree of if-else-elseif blocks. The bot completed the track within 30s, the assigned Teaching Assistant indicated that the turns were very sharp and jerky since there was no usage of PID and suggested the need for a PID based turning algorithm.

4 Devised PID Algorithm and Relevant Code Snippets

The basic code template provided including `main.cpp`, `led.c` were used and few more functions were created to add the functionality we needed for the PID controller.

We make use of the custom function `speed_control()`, to update the motion as well as PWM of the motors on the basis of two control signals provided for each the left and right motor. The PORTB register is accordingly varied to adjust the motion.

```
void speed_control(int left_motor, int right_motor){
        unsigned char l_val, r_val;
        unsigned char newPORTB = PORTB;
        newPORTB &= 0xF0;

        if(left_motor < 0){
                //take absolute value
                l_val = -left_motor;
                //perform soft left 2
                newPORTB |= 0x01;
        }
        else{
                l_val = left_motor;
                //perform soft right
                newPORTB |= 0x02;
        }

        if(right_motor < 0){
                //take absolute value
                r_val = -right_motor;
                //perform soft right 2
                newPORTB |= 0x08;
        }
        else{
                r_val = right_motor;
                //perform soft left
                newPORTB |= 0x04;
        }


        l_val = l_val > 255 ? 255 : l_val;
        r_val = r_val > 255 ? 255 : r_val;
        PORTB = newPORTB;
        velocity(l_val, r_val);

}
```

For the above block, if the value of right/left motor control signal < 0 we perform SOFT RIGHT 2 and SOFT LEFT 2 accordingly, else we perform SOFT RIGHT and SOFT LEFT accordingly. We also change the PWM using the custom `velocity()` function to adjust the speed of left and right motors using PWM. By setting values ranging from 0 to 255 to the registers OCR1AL and OCR1BL.

```
void velocity(unsigned char left_motor, unsigned char right_motor){
       //PWM control
       OCR1AL = left_motor;
       OCR1BL = right_motor;
}
```

To decide with what speed it must move forward, we monitor the average of the three sensor readings, and set both left and right motor PWM to a function of these, and bound it between 0-255. Which was done by the subsequent code snippet.

```
cl=ADC_Conversion(3);
cc=ADC_Conversion(4);
cr=ADC_Conversion(5);
double average = (cl + cc + cr)/3;
//set the translate
double translate = 45 + average*0.9; //func for forward PWM
//check if its in range of 255
translate = (translate > 255) ? 255 : translate;
```

To make use of a PID based logic for the left and right turning, we initialize controller constants (k_p, k_i, k_d) values to arbitrary values, we keep track of the following signals `err`, `control_sig`, `prev_error`. The `err` signal we use will house the difference between the left and right sensor values.

```
double err = (cr - cl);
double control_sig = kp * err + ki * integral + kd * (err - prev_err)/dt;

integral += err * dt;
prev_err = err;
```

To decide on how to adjust the left and right sensor values, we create a signal called `rotate` which will depend on `control_sig`, which we keep updating every iteration from its prev value. We will bound this in between -1 to 1 for the sake of simplicity later. Division by the cc + offset is used to get an appropriately scaled version of control signal as `control_sig` on itself can take a very large range of values.

```
//set the value of rotate
double rotate = prev_rotate + (control_sig/(cc + offset));

//bound in range of -1 to 1
rotate = (rotate < -1) ? -1 : rotate;
rotate = (rotate > 1) ? 1 : rotate;
```

Now using the `rotate` and `translate` signals produced we need to correct the PWM values to the motors accordingly. For this we make use of the custom function `correction()`. We now check the value of the control signal `rotate`. If it is negative we increase the left control signal to be more than right control signal by a value of `2*translate*rotate`, if it is positive we make right control signal less than left by a value of `2*translate*rotate` and accordingly adjust the motion of bot.

```
void correction(double translate, double rotate){
	int l_val, r_val;

	if(rotate < 0){
		l_val = translate + 2*translate*rotate;
		r_val = translate;
	}
	else{
		l_val = translate;
		r_val = translate - 2*translate*rotate;
	}
	speed_control(l_val, r_val);
}
```

4.1 Two Discontinuous Black Squares on the Path

The **condition for the discontinuity black squares** on the track, we unconditionally move for 100ms ahead with a slower speed without worrying about the rotate signal using following snippet.

```
if((cl > 45 && cr > 45) && cc < 130) {
    // if we encounter the standalone black square on the track, which occurs twice
    // ignore left and right sensor values
    rotate = 0;
    // reduce the forward speed
    translate /= 5;
    correction(translate, rotate);
    // sleep for 100 ms to escape the square
    _delay_ms(100);
}
```

5 Main Iterative Block of Code

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

	// control value constants
	double kp = 1.05;
	double ki = 0.00000028;
	double kd = 0.49;
	double dt = 1;

	double prev_err = 0;
	double prev_rotate = 0;
	double integral = 0;

	while(1){
		double cl, cc, cr;
		double offset = 0.5;

		cl=ADC_Conversion(3);
		cc=ADC_Conversion(4);
		cr=ADC_Conversion(5);

		double average = (cl + cc + cr)/3;
		double err = (cr - cl);
		double control_sig = kp * err + ki * integral + kd * (err - prev_err)/dt;

		integral += err * dt;
		prev_err = err;

		//set the translate
		double translate = 45 + average*0.9;
		//check if its in range of 255
		translate = (translate > 255) ? 255 : translate;

		//set the value of rotate
		double rotate = prev_rotate + (control_sig/(cc + offset));
		//check if its in range of -1 to 1
		rotate = (rotate < -1) ? -1 : rotate;
		rotate = (rotate > 1) ? 1 : rotate;

		if((cl > 45 && cr > 45) && cc < 130) {
			rotate = 0;
			translate /= 5;
			correction(translate, rotate);
			_delay_ms(100);
		}

		correction(translate, rotate);
		prev_rotate = 0.2*rotate;

	}
}
```

6 Result

The track was completed by the bot well under 30s (in times of the order of 26s), and all the turns were performed smoothly after adding the PID, unlike the initial thresholding based approach which was giving sharp and jerky turns. The PID controller performed the track completion for the values of controller constants

$$k_p = 1.05$$

$$k_i = 2.8 \times 10^{-7}$$

$$k_d = 0.49$$

7 Problems Faced

- The code initially did not work for the values of initialized values of k_p, k_i, k_d. After multiple iterations of tuning the values, we were finally able to get smooth turning at curves on the path.
- We hadn't bounded the control signals initially that led to observing no variation on varying the parameters. We identified that on bounding the rotate signal in -1 to 1 and also adding the > 255 checks for PWM helped in solving this.
- The block for the discontinuous squares in the track didn't work, we had to vary the thresholds (45, 130) etc in multiple iterations, we also had to accordingly change the sensitivity of the sensors using the potentiometers for each sensor.
- We had initially used different constants in various control signal expressions, which led to difficulty in tuning by just varying controller parameters. On speaking with the Teaching Assistant we were advised to remove such scaling constants and instead vary the value of k_i, k_d, k_p alone appropriately.