# EE324: Control Systems Lab

## Experiment 3: Line Follower

**Batch: Tuesday - Table 7**

**Authors**

**Hardik Khariwal(22B3954)**

**Aditya Singh (22B3964)**

**Keshav Samdani(22B3952)**

October 24, 2024

## Objective

This experiment aims to design and implement a PID controller for the Spark V robot, ensuring it can successfully follow a continuous track using the robot’s IR sensors within a target time of 30 seconds

## Control Algorithm

In this implementation, a variety of motion control algorithms are utilized alongside a PID controller, which includes functions for moving forward, backward, left, right, and making soft movements (such as soft left or soft right). The primary control mechanism is the PID logic, which adjusts the robot’s motion based on sensor data to maintain alignment with the track.

- **PID Control Logic:** The robot is equipped with three sensors (left, right, and center) connected to ADC channels. It calculates positional errors by comparing the values from these sensors. The proportional (P) and derivative (D) terms in the PID controller are applied to the differences between sensor readings (left-right and left-center) to determine the necessary corrections for keeping the robot centered on the track.
  - **Proportional Term (Kp):** When the robot is significantly off the line, the P term generates a large correction to quickly realign it. However, relying solely on this term can lead to oscillations or overshooting, particularly if the gain is set too high.
  - **Derivative Term (Kd):** The D term reduces overshooting and oscillations by providing a corrective action that moderates the response when the error changes rapidly. This results in smoother movements, preventing abrupt or jerky corrections.
  - **Integral Term (Ki):** The I term addresses any steady-state errors, such as slight misalignments or drifts that may develop over time. It ensures that even small, persistent deviations are corrected, helping the robot stay centered on the track.
  - The velocity function adjusts the motor speed based on the output from the PID controller.
- **Motion Set Functions:** A set of helper functions (left, right, forward, soft left, soft right, etc.) is employed to define the robot's movements based on the PID control output. Each motion function controls the rotation of the wheels by configuring the motor direction through PORTB settings

| Parameter | Value |
|---|---|
| $K_p$ | 4 |
| $K_i$ | 0.01 |
| $K_d$ | 8 |

Table 1: PID Controller Parameters

## Challenges Faced

**Sensor Calibration**: The IR sensors on the robot required precise calibration to accurately detect the track. Minor changes in lighting or surface conditions impacted sensor performance, necessitating frequent recalibration.

**PID Tuning**: Finding the right values for Kp and Kd proved difficult. Excessively high Kp values led to overshooting, while too low values resulted in unresponsiveness. The derivative term, Kd, was effective in reducing oscillations but needed careful fine-tuning.

**Navigating Sharp Turns and Corners**: Handling sharp turns and corners on the track was particularly challenging, as the robot had to adjust quickly to avoid overshooting or straying off the path. The PID controller had to respond rapidly to these sudden changes while maintaining stability, which often required precise tuning of the derivative term.

**Speed Control**: Achieving a target completion time of 30 seconds while ensuring the robot stayed on track required meticulous control of its speed. At higher speeds, the robot had less time to correct its path, making it more susceptible to veering off the line. While slower speeds improved accuracy, they risked exceeding the target time, necessitating a careful balance between speed and control.

**Battery Voltage** Fluctuations: As the robot’s battery drained during operation, the voltage supplied to the motors fluctuated, impacting the robot’s speed and responsiveness. These variations made it challenging for the PID controller to maintain consistent performance, requiring adjustments to the control parameters.

## Code

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "lcd.c"
void motion_pin_config() {
    DDRB |= 0x0F;
    PORTB &= 0xF0;
    DDRD |= 0x30;
}
void motion_set(unsigned char Direction) {
    PORTB = (PORTB & 0xF0) | (Direction & 0x0F);
}
void forward() { motion_set(0x06); }
void back() { motion_set(0x09); }
void left() { motion_set(0x05); }
void right() { motion_set(0x0A); }
void soft_left() { motion_set(0x04); }
void soft_right() { motion_set(0x02); }
void hard_stop() { motion_set(0x00); }
void soft_stop() { motion_set(0x0F); }
void adc_init() {ADMUX = 0x20; ADCSRA = 0x86; }
void init_devices() {
    cli();
    adc_init();
    motion_pin_config();
  sei();
}
void timer1_init() { = 0xA1; TCCR1B = 0x0D; }
unsigned char ADC_Conversion(unsigned char Ch) {
    ADMUX = 0x20 | (Ch & 0x07);
    ADCSRA |= 0x40;
    while (!(ADCSRA & 0x10));
    return ADCH;
}
void velocity(unsigned char left_motor, unsigned char right_motor) {
OCR1AL = left_motor;OCR1BL = right_motor;
}
int main() {timer1_init();init_devices();lcd_set_4bit();lcd_init();
    unsigned char left_sensor, right_sensor, center_sensor;
    int error, error_prev = 0, integral = 0, v0 = 69;
    float Kp = 8, Kd = 4, Ki = 0.01, minThres = 3, offset = 5, ki_limit
= 0.091;
    velocity(64, 64);
    while (1) {
        left_sensor = ADC_Conversion(3);
        right_sensor = ADC_Conversion(5);
        center_sensor = ADC_Conversion(4);
        lcd_print(1, 1, right_sensor, 3);
        lcd_print(1, 5, center_sensor, 3);
        lcd_print(1, 9, left_sensor, 3);
        forward();
        error = right_sensor - left_sensor;
        integral = (integral + error > ki_limit) ? ki_limit : (integral
+ error < -ki_limit) ? -ki_limit : integral + error;
        float pid_drive = Kp * error + Kd * (error - error_prev) + Ki *
integral;
        lcd_print(1, 1, pid_drive, 3);
        velocity(pid_drive > v0 * 2 ? pid_drive : v0 * 2, pid_drive > v0
* 2 ? pid_drive : v0 * 2);
        if (abs(pid_drive) < minThres) {
            forward();
        } else if (center_sensor < 50) {
            (pid_drive < 0) ? ((abs(pid_drive) < offset) ? soft_left() :
left(), velocity(pid_drive > v0 ? pid_drive : v0, v0))
                             : ((abs(pid_drive) < offset) ? soft_right()
: right(), velocity(v0, pid_drive > v0 ? pid_drive : v0));
        }
        error_prev = error;
        _delay_ms(10);
    }
}
```

## Results

The PID control system was successfully implemented, enabling the robot to follow the track efficiently. The robot completed the track in just 26 seconds, well within the required 30-second timeframe, thereby achieving the experiment's objectives.

The following outcomes were observed:

- The robot traced the track with minimal deviations.
- Fine-tuning the PID values significantly reduced oscillations, allowing for smoother navigation along the track.
- The system's response time was optimized, enabling the robot to manage tight turns without losing track.

## Observations and Inferences

- **Importance of Fine-Tuning for Stability**: Achieving stable and accurate line-following necessitated precise tuning of the PID gains. Even minor adjustments to the P, I, and D values significantly influenced the robot's ability to stay on course, highlighting the sensitivity of the PID controller. The proportional and derivative terms effectively maintained the robot's alignment with the track; proportional control provided a quick response, while derivative control helped reduce oscillations.
- **Sensor Sensitivity**: The IR sensors were essential for providing

feedback, but environmental factors such as lighting conditions and surface reflectivity affected the sensors' accuracy. Future enhancements might involve implementing more robust sensor filtering or adaptive calibration techniques.

- **Tuning Parameter**s: Achieving the right balance between Kp and Kd was vital to ensure the robot followed the track without overshooting or lagging. Optimal tuning contributed to a smooth and responsive control action.