# CS684 :: Lab 2: Heptagon code for Line following


```
Submitted by:
- Santanu Sahoo (23m0777)
- Sm Arif Ali (23m0822)
- Soumik Dutta (23m0826)
- Arnab Bhakta (23m0835)
```


## Overview

This document explains the implementation of a PID-based line-following robot in Heptagon. The robot uses five white-line sensors to detect a black line on a white surface and adjusts its motor velocities accordingly to stay on track.

## Inputs and Outputs

### Inputs

- Five white-line sensors (`sen0, sen1, sen2, sen3, sen4`), each providing values between `0-1023`.
- Lower values indicate a white surface, and higher values indicate a black surface.

### Outputs

- `v_l`: Velocity of the left motor (`0-100`).
- `v_r`: Velocity of the right motor (`0-100`).
- `dir`: Robot movement direction:
    - `0` - Stop
    - `1` - Forward
    - `2` - Left
    - `3` - Right
    - `4` - Backward

## Scenarios Considered

- **straight** line => **neutral** (left and right motor speed @50)
  - if reading **straight is continued** for some cycles => **accelerate** (increase both motor speed by 5)

- differnet types of turns in both left and right direction. [+- means one motor speed is increased while the other is decreased based on direction]
  1. **Acute Turn**: Less than 45° => **speed +- ~10**
  2. **Sharp Turn**: 45° to 90° => **speed +- ~25**
  3. **Right-Angle Turn**: 90°-135° => **speed +- ~35**
  4. **Fast Corner**: More than 135°-180° => **speed +- ~50**
- robot **losing the track** => **backward**
- robot reaching the **finish line** (all black strip) => **stop**

## Assumptions

- The robot is moving on a **well-defined black track on a white surface**.
- Sensors provide reliable values between `0-1023`.
- If the sensor is on **white** surface, it gives reading in range **[0, 100]**.
- If the sensor is on **black** surface, it gives reading in range **[800, 1023]**.
- The **black line width is 2cm**.
- There is a **finish line** which is all black strip ie. all sensors reads high value.
- Some random **`kp = 0.06`** has been selected as proportional gain that balances responsiveness without excessive oscillations.
- **kd and `ki`**: Set to zero initially;
- **(kp, kd, ki)** will be tuned empirically for optimal stability and steady-state performance.
- Motor speeds are in the range `0-100`.
- The weight distribution for sensors is symmetrical around the center.
- `You are free to make assumptions regarding readings of white line sensors at different instances, distance covered by robot for different motions in single step etc.` ??

## Implementation Details

PID Controller with direct sensor values has been used.

1. First weighted average of the 5 sensor values is calculated.
2. Then, PID Error is calculated on this Weighted Average Value.
3. Based on the PID error direction is determined and motor speed is updated.

Following are the details.

### 1. Weighted Average of five Sensors

The five sensor values are processed to compute the weighted avg of readings. The `sensor_avg` calculation formula is:

$$sensor\_avg = \frac{\sum(sensor_i \times weight_i)}{\sum sensor_i}$$

**Weights:**

- weights choosen: `[-1000, -500, 0, 500, 1000]`
- The `middle sensor's weight is zero` because it should contribute no error when perfectly aligned.
- `Negative weights` for left-side sensors represent deviations to the `left`.
- `Positive weights` for right-side sensors represent deviations to the `right`.
- Ensure error direction matches deviation (negative for left, positive for right).
- Provide sufficient resolution for integer-only calculations by scaling values appropriately.

### 2. PID Error Calculation

The ideal `sensor_avg` should be 0. So, the error is `sensor_avg` itself. The PID error is computed using proportional, integral, and derivative components:

```
node calPidError(value: int) returns (pid_error: int)
var p, i, d: int;
let
     p = value;
     (* Integral term with overflow prevention *)
     i = if ((0->pre(i) + value) <= max_i)
            then (0->pre(i) + value)
         else
            max_i;
     d = value - 0->pre(value);
     pid_error = (kp*p + ki*i + kd*d) / kscale;
tel
```

$$\text{Here, we use} K\_scale \text{ to approximate floating-point division using integer arithmetic.}$$

- **`max_i = 200,000,000`**: Limits the integral term to prevent overflow during long-term accumulation.

## 3. Direction and motor speed update

### a. Determining Direction

```
     if sensor_sum > (black_thresh*5) then 0  -- all black -> stop
else if sensor_sum < (white_thresh*5) then 4  -- all white -> backward
else if pid_error = 0 then 1  -- no error -> move forward
else if sensor_avg < 0 then 2  -- Negative error -> turn left
else if sensor_avg > 0 then 3  -- Positive error -> turn right
else 0; -- fallback/default
```

### b. Adjusting Motor Speeds

Adjusting motor speeds based on motor reaction while ensuring they remain within the [0,100] range.

In straight line

```
v_l = 50 -> safe_motor_update(pre(v_l), 5);
v_r = 50 -> safe_motor_update(pre(v_r), 5);
```

During turn: pid_error would be negative for left turn and positive for right turn.

```
v_l = safe_motor_update(50, pid_error);
v_r = safe_motor_update(50, -1*pid_error);
```

## Simulation Conditions and Outputs

- YouTube Link

### Input/Output table

| Step | sen0 | sen1 | sen2 | sen3 | sen4 | v_l | v_r | dir | comment |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 9 | 3 | 995 | 2 | 2 | 50 | 50 | 1 | straight |
| 2 | 9 | 5 | 997 | 2 | 2 | 55 | 55 | 1 | straight |
| 3 | 8 | 5 | 790 | 534 | 15 | 64 | 36 | 3 | acute right => +-10 |
| 4 | 8 | 5 | 853 | 214 | 15 | 57 | 43 | 3 | acute right => +-10 |
| 5 | 3 | 9 | 995 | 2 | 2 | 50 | 50 | 1 | straight |
| 6 | 40 | 970 | 460 | 5 | 3 | 26 | 74 | 2 | sharp left => -+25 |
| 7 | 15 | 480 | 620 | 2 | 1 | 35 | 65 | 2 | sharp left => -+25 |
| 8 | 5 | 9 | 997 | 7 | 2 | 50 | 50 | 1 | straight |
| 9 | 980 | 612 | 974 | 1 | 3 | 16 | 84 | 2 | rightangle left => -+35 |
| 10 | 48 | 970 | 460 | 823 | 34 | 48 | 52 | 2 | rightangle left => -+35 |
| 11 | 12 | 24 | 971 | 12 | 2 | 49 | 51 | 2 | straight |
| 12 | 9 | 3 | 995 | 2 | 2 | 50 | 50 | 1 | straight |
| 13 | 3 | 1 | 430 | 995 | 990 | 92 | 8 | 3 | fast right => +-50 |
| 14 | 2 | 1 | 110 | 990 | 980 | 99 | 1 | 3 | fast right => +-50 |
| 15 | 1 | 3 | 8 | 900 | 975 | 100 | 0 | 3 | fast right => +-50 |
| 16 | 2 | 1 | 2 | 1 | 2 | 50 | 50 | 4 | outoftrack => back |
| 17 | 1 | 1 | 1 | 1 | 1 | 50 | 50 | 4 | outoftrack => back |
| 18 | 1 | 1 | 1 | 1 | 1 | 50 | 50 | 4 | outoftrack => back |
| 19 | 9 | 11 | 997 | 5 | 2 | 50 | 50 | 1 | straight |
| 20 | 1 | 3 | 995 | 7 | 4 | 55 | 55 | 1 | accelerate => ++5 |
| 21 | 9 | 11 | 997 | 5 | 2 | 60 | 60 | 1 | accelerate => ++5 |
| 22 | 1 | 3 | 995 | 7 | 4 | 65 | 65 | 1 | accelerate => ++5 |
| 23 | 9 | 11 | 997 | 5 | 2 | 70 | 70 | 1 | accelerate => ++5 |
| 24 | 1 | 3 | 995 | 7 | 4 | 75 | 75 | 1 | accelerate => ++5 |
| 25 | 1000 | 986 | 1000 | 989 | 969 | 0 | 0 | 0 | finish => stop |