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

Assumptions

- The robot is moving on a well-defined black track on a white surface.
- The PID gains (`kp, kd, ki`) are set based on field trials.
- Motor speeds are in the range `0-100`.
- Sensors provide reliable values between `0-1023`.
- 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. Following are the details

1. Sensor Processing

The five sensor values are processed to compute the weighted sum of readings.

```
const sensor_weights: int^5 = [-10, -5, 0, 5, 10]
const sensor_weight_epsilon: int = 1

node senWeightedSum(sen: int^5) returns (sensor_sum: int)
var weighted_sum, epsilon_sum: int;
let
    weighted_sum = fold<<5>>weightedSum(sen, sensor_weights, 0);
    epsilon_sum = fold<<5>>(+)(sen, 0) + sensor_weight_epsilon;
    sensor_sum = weighted_sum / epsilon_sum;
tel
```

The `sensor_sum` calculation formula is:

$$sensor\_sum = \frac{\sum(sensor_i \times weight_i)}{\sum sensor_i + \epsilon}$$

ϵ is very small value $(\text{near } 1)$ to ensure the denominator is never zero.

2. PID Error Calculation

The PID error is computed using proportional, integral, and derivative components:

```
const kp: int = 10
const kd: int = 2
const ki: int = 1
const kscale: int = 10
const max_i: int = sensor_max  -- to prevent integral overflow

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

- The equation used is:

$$PID_{error} = \frac{K_p P + K_i I + K_d D}{K_{scale}}$$

Here, we useK_scale to approximate floating-point division using integer arithmetic.

- Empirically tuning gains (`kp, ki, kd`) based on real world testing.

3. Mapping PID Error to Motor Speed

The PID error is mapped to a motor reaction value using Min-Max scaling:

```
node piderr_2_motor_reaction(pid_error: int) returns (motor_reactio
var sensor_range, motor_range, pid_diff, temp: int;
let
    pid_diff = (pid_error - sensor_min);
    sensor_range = (sensor_max - sensor_min);
    motor_range = (motor_max - motor_min);
    temp = pid_diff * motor_range;
    motor_reaction = temp / sensor_range;
tel
```

Formula:

$$motor_{reaction} = \frac{(PID_{error} - sensor_{min}) \times (motor_{max} - motor_{min})}{sensor_{max} - sensor_{min}}$$

4. Determining Direction

```
dir = if sensor_sum = 0 then 1
      else if sensor_sum < 0 then 2
      else if sensor_sum > 0 then 3
      else if fold<<5>>(+)(sen, 0) < 5 then 4
      else 0;
```

Conditions:

- `sensor_sum = 0` → Move forward.
- `sensor_sum < 0` → Turn left.
- `sensor_sum > 0` → Turn right.
- `All sensors detect white` → Move backward.
- `Else` → Stop.

5. Adjusting Motor Speeds

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

```
v_l = if (50 + motor_reaction) > 100 then 100
      else if (50 + motor_reaction) < 0 then 0
      else (50 + motor_reaction);

v_r = if (50 - motor_reaction) > 100 then 100
      else if (50 - motor_reaction) < 0 then 0
      else (50 - motor_reaction);
```

Simulation Conditions and Outputs

- YouTube Link

Scenario	Sensor Inputs (sen0 to sen4)	Computed sensor_sum	PID Error	v_l	v_r	Direction
Straight	[10, 20, 50, 20, 10]	0	0	50	50	Forward
Left Turn	[50, 50, 10, 5, 2]	Negative	High	30	70	Left
Right Turn	[2, 5, 10, 50, 50]	Positive	High	70	30	Right
Backward	[2, 0, 1, 3, 0]	N/A	N/A	40	40	Backward
Off Track	[0, 0, 0, 0, 0]	N/A	N/A	0	0	Stop