CS684 Spring 2024-25

CS684: Embedded System Course

Lab 2: Heptagon code for Line following

AIM

- Writing heptagon code for Line Following.

- Inputs:

  - Values from **5** White Line Sensors.
  - Each Sensor value ranges between: 0 to 1023.
  - On white surface, sensor values are low and for black surface, sensor values are high.
- Outputs:

  - **3** for controlling the motion of the Robot (1 for Direction and 2 for Motor's velocities).
  - From the final heptagon node, 3 outputs are required: 2 velocity values (of left and right motors) and direction.
    - Directions list:

Direction	Representation
Stop	0
Forward	1
Left	2
Right	3

| Backward | 4 |

-
  Motor velocity ranges between 0 to 100. It should not be negative or greater than 100.

- The heptagon node for line following must be defined according to the format given:

- node main(sen0, sen1, sen2, sen3, sen4: int) returns (v_l, v_r, dir: int)
- var sen: int^5;
- let
-     sen = [sen0, sen1, sen2, sen3, sen4];
- tel

- Using the above information, write a heptagon for Line following of the robot. Consider a black line on white surface.
- Use PID controller for line following.

  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.

Algorithm

- White Line following has to be done using the 5 white line sensors provided.
- Each sensor value ranges between: 0 to 1023.
- On white surface, sensor values are low and for black surface, sensor values are high.
- So depending upon the value of the particular sensor, robot has to decide whether it is on the black line or it is deviated from the line.
- To adjust the position of the robot if it deviated from the black line, use PID.
- Likewise, team has to assume different scenarios or test cases which may come across while robot is traversing the arena.
- You are free to make assumptions regarding readings of sensors at different instances.

Algorithms for Line Following

1. Bang-bang Controller

Here all the sensor values are converted to binary using thresholding and will be directly used to control the speed and direction of the motors. It uses an if-else ladder structure for

logic implementation. The controller that was developed in Lab 1 uses the same logic. This method is not reliable as numerous cases must be defined for a set of sensor values which will be difficult to implement.

2. PID Controller with binary sensor values

PID controller is a feedback loop that utilises the error in the sensor values to correct the position of the robot. Here all the sensor values are converted to binary using thresholding and error value is calculated. This error value is used for controlling the spees and direction of the motor.

3. PID Controller with direct sensor values

This is similar to method 2 but here no thresholding is used for sensor values while calculating error. The error calculation can be done using various methods like weighted average or using the sensor values as such.

How does PID work?

The system calculates the 'error', or 'deviation' of the physical quantity from the set point, by measuring the current value of that physical quantity using a sensor(s). To get back to the set point, this 'error' should be minimized, and should ideally be made equal to zero. Also, this process should happen as quickly as possible. Ideally, there should be zero lag in the response of the system to the change in its set point.

More information can be found in many books and websites, including here:

http://en.wikipedia.org/wiki/PID_controller

Implementing PID

(i) Error Term (e):

This is equal to the difference between the set point and the current value of the quantity being controlled.

error = set_point – current_value (in our case is the error variable get from the position of Robot over the line

(ii) Proportional Term (P):

This term is proportional to the error.

P = error

This value is responsible for the magnitude of change required in the physical quantity to achieve the set point. The proportion term is what determines the control loop rise time or how quickly it will reach the set point.

(iii) Integral Term (I):

This term is the sum of all the previous error values.

I = I + error

This value is responsible for the quickness of response of the system to the change from the set point. The integral term is used the eliminate the steady state error required by the proportional term. Usually, small Robots doesn't use the integral term because we are not concerned about steady state error and it can complicate the "loop tuning".

(iv) Differential or Derivative Term (D):

This term is the difference between the instantaneous error from the set point, and the error from the previous instant.

D = error - previousError

This value is responsible to slow down the rate of change of the physical quantity when it comes close to the set point. The derivative term is used to reduce the overshoot or how much the system should "over correct".

Equation:
 PIDvalue = (Kp*P*) + *(Ki*I) + (Kd*D)

Where:

Kp is the constant used to vary the magnitude of the change required to achieve the set point. Ki is the the constant used to vary the rate at which the change should be brought in the physical quantity to achieve the set point. Kd is the constant used to vary the stability of the system.

Based on the above approach, the bellow function was implemented:

- void calculatePID()
- {
- P = error;
- I = I + error;
- D = error-previousError;
- PIDvalue = (Kp*P) + (Ki*I) + (Kd*D);
- previousError = error;
- }

- Folder should contain following:
  1. `.ept` file: Heptagon code file
  2. `Readme.txt` file file:
     - Containing information about the assumptions made and description about the conditions or the scenarios if considered, inputs used for simulation and outputs obtained in a tabular form.
     - Also add youtube link of the video - shoot the screen recording of the simulation of algorithm for atleast 20 steps with different input combinations.
  3. `Contribution.txt` file: stating detailed contribution of each member
- Compress the folder and rename it as `<GroupName>_Lab_2.tar.gz`
- Upload the file on Moodle
- There should be only one submission per group. Member having highest roll no should do the submission.