Project X : Report

- PID Control system :-
  - PID allows us to set a process variable (like direction , temperature etc) to a setpoint value by continuously adjusting to compensate for external effects.

  - PID control system continuously calculates error value between current variable value and final setpoint value in order to shift the process variable value to that of the setpoint variable.

  - It has 3 components :-
    - Propoprtional :-
      - Takes only the error into account
      - Bigger the error , bigger the amount to change the process variable.

    - Integral :-
      - Takes past errors into account
      - Tries to nudge the process variable to ensure that the small errors are removed
    - Derivative :-
      - Takes change of error into account
      - Ensures that process variable does not overshoot the setpoint value

  - By taking these values , (ie P , I and D) we take a weighted sum to get the final output of the controller.
  - Hence the final output of the controller is given by :-

$$OUTPUT = (Ki * I) + (Kp * P) + (Kd * D)$$

  - This final output can be used to adjust the wheel speed to allow for smooth and effective turns while following the line.

- Line Follower Robot:-
    - Line follower bot uses an IR brightness sensor / sensor array to navigate on a black line .

    - By taking the brightness output from the sensor output , the microcontroller figures out the position of the bot with respect to the line to make the necessary adjustments in order for it to continue following the line.

    - To fine tune the rate of turning we use a control system mechanism called PID (Proportional Integral Derivative) control loop to adjust the parameters of rate of turning by using previous turn data.

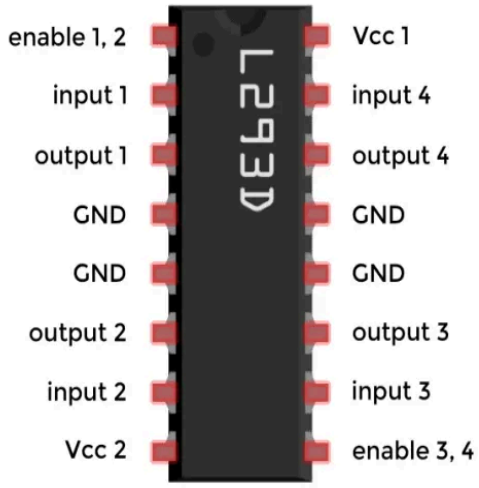
    - We take the input from a linear array of 5 IR sensors and using the digital input of whether the IR sensor detects black or while underneath , we can run a function that causes the bot to rotate until desired orientation has been met.

    - The output of the IR array gives us various pointers on the orientation of the bot :- (0 :- White , 1 :- Black)
        - [0 0 1 0 0] :- Perfectly aligned
        - [0 0 0 1 0] :- Line to the right (turn right)
        - [0 0 0 0 1] :- Line at extreme right (turn hard right)
        - [0 1 0 0 0] :- Line to the left (turn left)
        - [1 0 0 0 0] :- Line at extreme left (turn hard left)

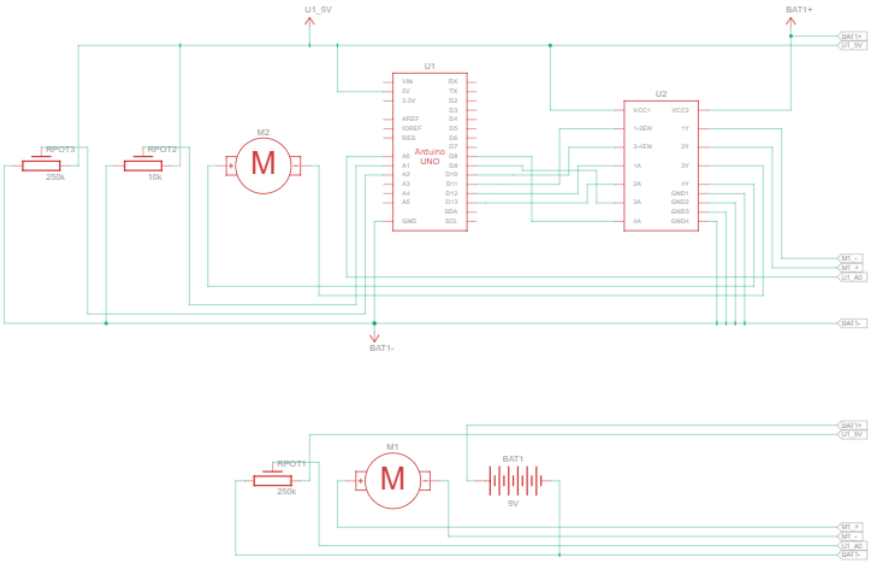
    - To adjust the rate of a “hard” and a “soft” turn we use the PID control loop.
    - To fine tune our robot , we have to adjust Kp , Ki and Kd values of the PID control loop based on trial and error to ensure our robot never overshoots or undershoots turns.

    - To power this line follower robot , we use two 9v hobby gearmotors that use the L293D motor driver.

- Schematic of L293D motor driver :-



- This motor driver can actuate two hobby gear motors and can control their rpm's respectively
- Enable 1 , 2 :- PWM pin to control speed of motor 1
- Input 1 , Input 2 :- Direction Set pins for motor 1
- Output 1, Output 2 :- Power pins for motor 1
- GND :- Ground Pin
- VCC 2 :- 9V power for motors
- VCC 1 :- 5V power for motor driver
- Input 4 , Input 5 :- Direction set pins fo motor 2
- Output 4 , Output 5 :- Power pins for motor 2
- Enable 3,4 :- PWM pin to control speed of motor 2



-

- Maze Solver Robot :-
  - A maze solver robot is a type of line following robot that combines the line following aspect of the bot with a maze that contains dead ends , multiple turns and infinite loops.
  - It involves initially scanning the maze for an optimal and fastest route and then follows it.
  - The scanning and computation of the maze is done locally on the bot
  - After scanning the bot uses various algorithms to solve the maze in the least time possible.
  - Maze solvers convert the maze into a weighted graph.
  - Mapping algorithms used by maze solver :-
    - Breadthfirst Search :-
      - BFS explorers a graph layer , starts at a source node and explores all neighbours and then their unvisited neighbours.
      - Proceeds to visit all neigbours until it meets a dead end or a loop in which case it revers back to previous intersection.
      - Proceeds to do this till it finds the target node of the maze .
    - Depthfirst Search :-
      - Bot travels straight through the maze till it meets a dead end or is surrounded by visited cells
      - When it encounters this the bot backtraces its way to the previous intersection and takes one of the unvisited routes.
      - This process leads to the bot being able to find a valid path between source and target node
    - Flood fill search :-
      - Type of search algorithm that searches for the route by trying to find the path of least resistance.
      - Assigns the distance from the origin to the final node and then constantly updates the distance from the target node to that cell.
      - Takes the path of least resistance by following cells with smaller and smaller weights.
    - Djikstra’s :-
      - If we already have the weighted graph containing the map of the maze then using djikstra’s algorithm we can navigate ourselves between any two nodes in the shortest distance possible.