Mobile Robots : Feedback Control

De Montfort University | Thawatchai Sangthep | P2681054 | MSc Intelligent Systems | IMAT5121 | Week 6 Assignment Report

Contents

1. **Introduction**

This paper presents an implementation of PID controller to control a robot within an environment. The controller gives out the desired actuator output by calculating proportional (P), integral (I) and derivative (D) responses and summing those three components to compute the output. The robot randomly moves to detect a wall when running and after that it follows the wall by using PID. So, PID is used to improve accuracy during the implementation of the tasks using sensors to setting the distance between the robot and the detected the wall.

The control design process begins by defining the performance requirements, control system performance is often measured by applying a step function as the set point command variable, and then measuring the response of the process variable. Commonly, the response is quantified by measuring defined waveform characteristics. Rise time is the amount that the process variable overshoots the final value, expressed as a percentage of the final values. Settling time is the time required for the process variable to settle to within a certain percentage (commonly 5%) of the final value.

P = Proportional = difference between actual and planed / present

I = Integral = Sums (or average) of errors over time / past

D = Derivative = adapts the output to decrease if some variables are increasing rapidly / future Tunin

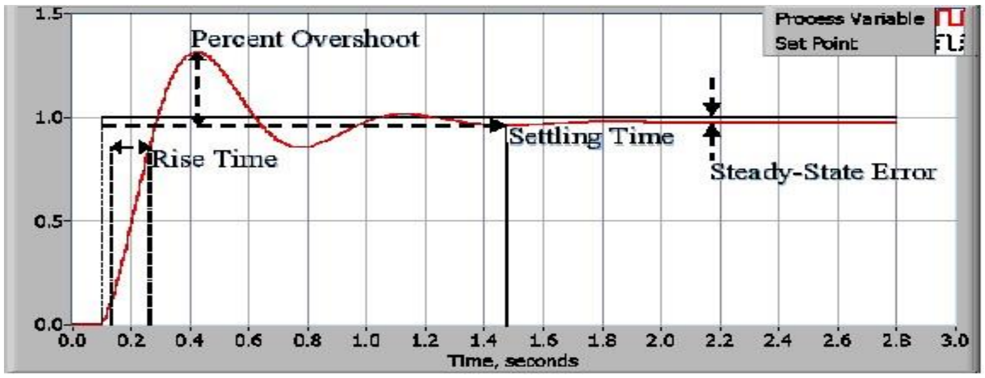


Figure 1 : Response of a typical PID closed loop system.

After using one or all of these quantities to define the performance requirements for a control system, it is helpful to define the worst-case conditions in which the control system that affects the process variable or the measurement of the process variable. It is important to design a control system that performs satisfactorily during worst case conditions. [1]

2. Task

From previous assessments, where we had a target to set a robot run randomly in the environment if the robot detects the wall, it uses sonar sensors to track and follow the wall. The task in this assessment has 2 missions to do.

Firstly, implement a PID controller, start with the proportional controller, find a way to store a set of previous error values that it will be used for the integral and derivative part of PID, implement the Derivative section, keep and analysis the Integral section.

Lastly, Tune the PID controller using the technique from the lecture, compare different tuning parameters to each other, and find out which one is the best.

3. Environment & Tools

The robot

Pioneer 3DX Mobile Robot is a small lightweight two-wheel two-motor differential drive robot ideal for indoor laboratory or classroom use. The robot is surrounded with SONAR sensors, on battery, wheel encoders, a microcontroller with ARCOS firmware and the Pioneer SDK advanced mobile robotics software development package. It can be used with a combination of various sensors such as video camera, ultrasonic sensors, gyroscope, etc. So, It is suitable for research and application involving mapping, teleoperation, localization, autonomous navigation etc. [2]

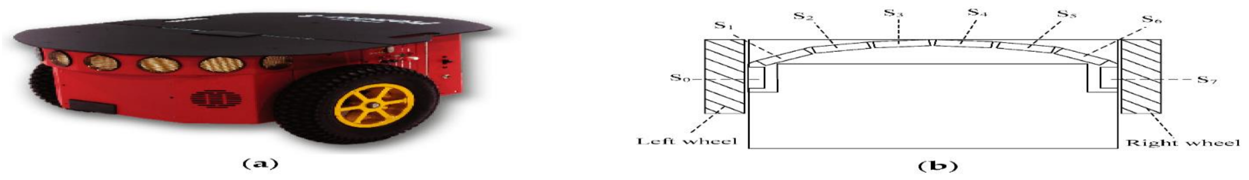


Figure 2 : Robot 3DX and Sensors

The Program simulator

A robot simulator CoppeliaSim (V-REP). It is an integrated development environment based on a distributed control architecture, where each object/model can be individually controlled via an embedded script, a plugin or remote API client. It can be control by using C/C++, Python, Java, Lua, MATLAB, or Octave. [3]

Figure 3 : CoppeliaSim from the creators of V-REP

The Environment

The environment is created in CoppeliaSim and given, the map has a wall which is the skeleton of the path to follow.

The surrounding box offers above compromises 15*15 meters walls

The wall inside the box measure 8*4 meters and the top right has a mini wall measure 2*2 meters

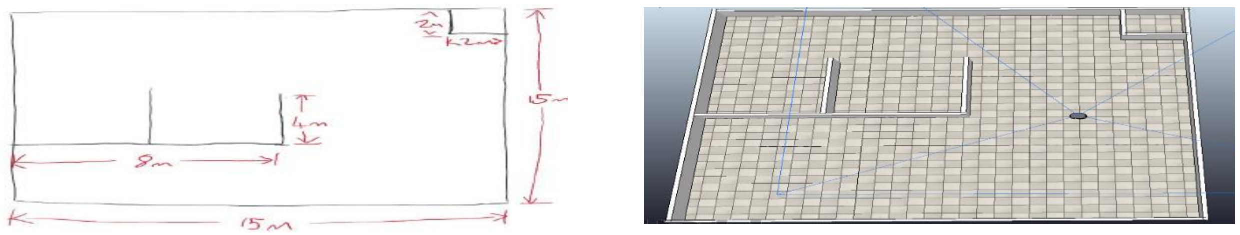


Figure 4 : Environment and blueprint of the map

The coding

I will be using Python language to continue the assignment. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Python is simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. So, the debugger is written in Python itself, testifying to Python's introspective power. [4]

Figure 5 : Python language

PID Controller

A PID Controller stand for proportional integral derivative and it is feedback method based device used to control different process variables like pressure, flow, temperature, and speed in industrial applications. In this controller, a control loop feedback device is used to regulate all the process variables. Several types of control is used to drive a system in the direction of an objective location otherwise level. It is used to maintain the real output when comparing the robot to where it should be to where it actually is. The process of setting the optimal gains for PID to get an ideal response from a control system is called **tuning**. These are different methods of tuning of which the "guess and check". [5]

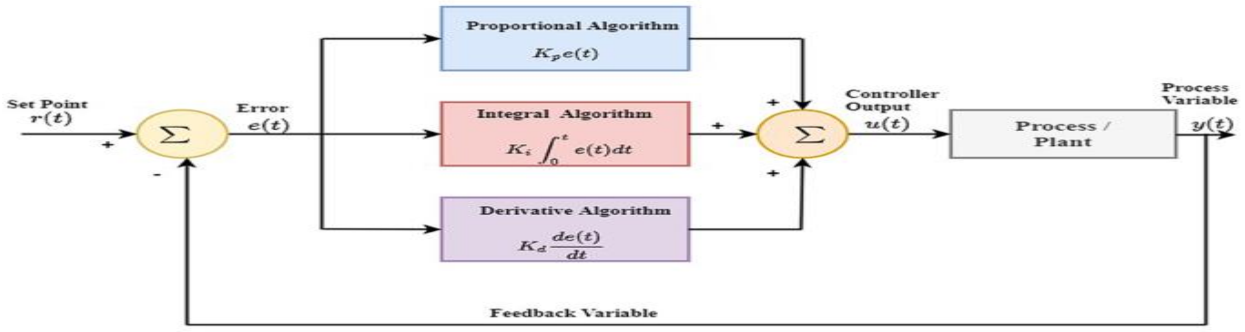
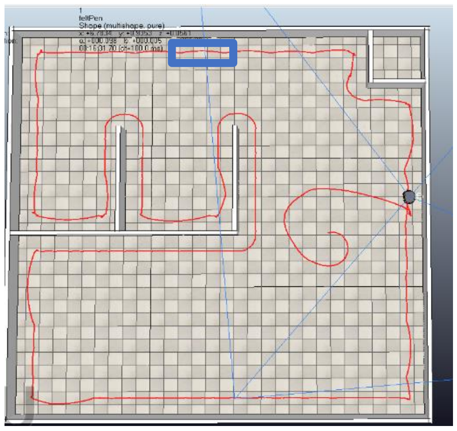


Figure 6: Working of PID controller

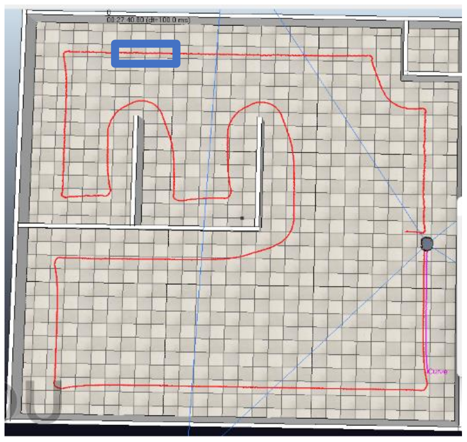
- Proportional part (P) = Kp * e(t), Kp is the gain of the proportional controller and e(t) is the error at time t.
- Integral part (I) = Ki * ∫ e(r) * d(t), Ki is the gain of the integral controller, e is the error and t is the integration variable (time interval)
- Derivative part (D) = Kd * de(t) / dt, Kd is the gain of the derivative controller, e is the error and t is the time.

4. Result

First of all, PID is a good tool for tuning the robot to run suitable to configuration. We had a problem with controlling the robot to move straight. As last assignment we did not put PID, so when the robot detected the wall its followed the path but not in straight line. The use of PID helped solve this problem. The result of our robot maintains a constant distance from wall.

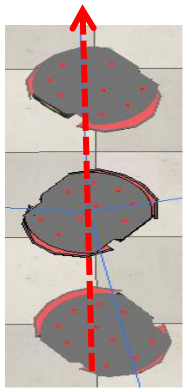


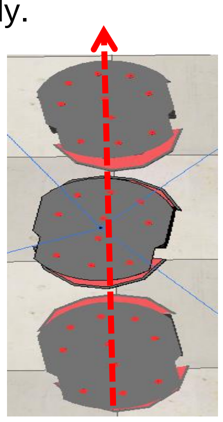
Using Value



Using PID controller

Next, when using only Proportional (P) value, the robot can run but it is not stable when running. The front of robot always oscillates but runs in a good line. So, we have to introduce rest of the values to help robot move steady.


Using only (p)


Using PID

When using the Proportional (P), Integral (I), Derivative (D) combined, the robot changed behavior and runs better than when only using Proportional (P). The result is the front of robot oscillates minimum and runs smoother than the past testing's.

5. Conclusion

In this lab assessment we implemented a PID controller. It is a tool for tuning the robot to solve errors by reading current errors and calculating values to fix errors and repeat the process again to modify the best solution. So, the values received from PID, are highly accurate and correct.

Firstly, the values of PID has affected on the robot such as if we input only Proportional (P) the robot will oscillate, if we input all values Proportional (P), Integral (I), Derivative (D) and the suitable value into the PID, the robot will change behavior and smoothly run, which is the expected result. So, we used the values because it is a good result, it is Kp = 2, Ki = 0.05, Kd = 5.

Secondly, the robot detected the wall and used the PID controller to calculate, when the robot follows the wall, it will calculate the errors with the values of PID by controls the speed and the angle that it is suitable and perform the task with updated feedback. If the robot detects in front, it will turn left by the angle of PID calculates. If the robot detected the wall in sensor 8 (90-degree front-right) and 9 (90-degree back-right) and if the values are not equal, the robot will decide to which side to turn based on the wall. If it has detected the wall and it has a value equal between sensor 8 and sensor 9, the robot will run perfectly straight.

Next, to calculate the distance between the robot and the wall, is simple. We have to add the distance between the sensor 8 (90-degree front-right) & 9 (90-degree back-right) to the wall and divide by 2, to get the average.

Next, how many degrees the robot has to use for the turn angle function. We have used difference between the sensor 8 (90-degree front-right) and 9 (90-degree back-right). So, we get the angle that the robot needed.

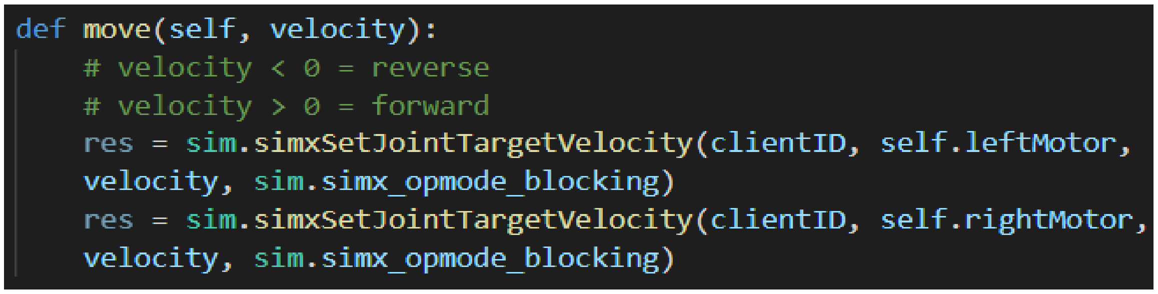
Therefore, this assignment implementation was a good task to learn about PID in mobile robot to calculate, tune, etc. If we compare the result between last assignment and the current one, the use of PID has made path following better.

To challenge programming and also reserve energy and movement, a battery could have been used.

6. Appendix

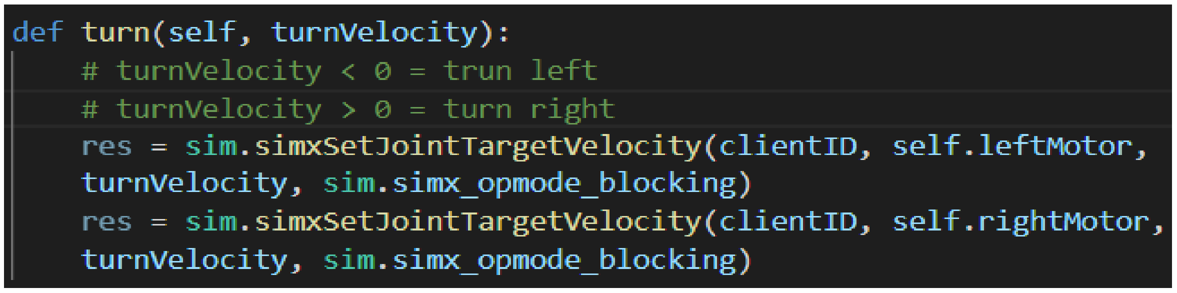
When we look at our assessment, we must design a strategy where the robot is run by using PID and stores all the previous error value to new calculate the output, combine all three values proportional gain, integral gain and derivative gain to get the new result.

6.1 Design for the movement of the robot
If Speed Motor Left = Right && >= 1 : The robot run “Straight Forward”
If Speed Motor Left = Right && <= -1 : The robot run “Straight Backward”



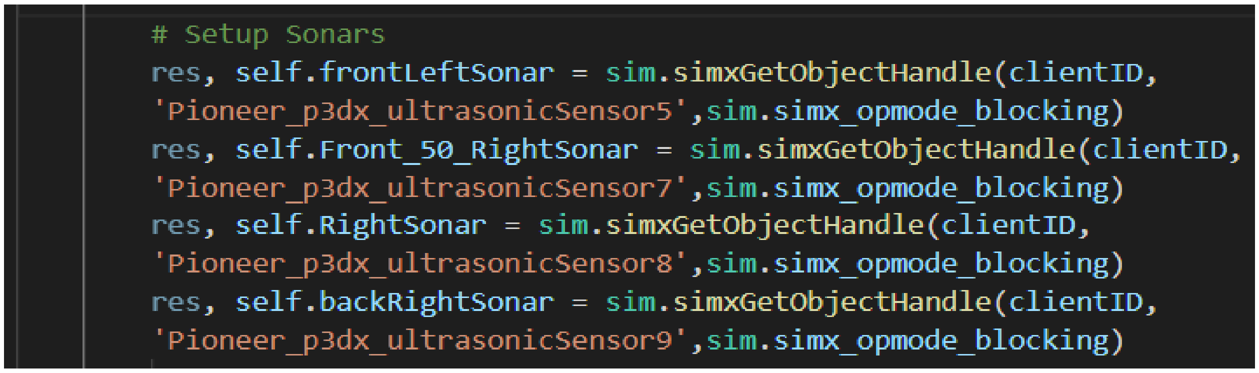
```
def move(self, velocity):
    # velocity < 0 = reverse
    # velocity > 0 = forward
    res = sim.simxSetJointTargetVelocity(clientID, self.leftMotor,
    velocity, sim.simx_opmode_blocking)
    res = sim.simxSetJointTargetVelocity(clientID, self.rightMotor,
    velocity, sim.simx_opmode_blocking)
```

If Speed Motor Left > Right : The robot turn “Right”
If Speed Motor Left < Right : The robot turn “Left”



```
def turn(self, turnVelocity):
    # turnVelocity < 0 = trun left
    # turnVelocity > 0 = turn right
    res = sim.simxSetJointTargetVelocity(clientID, self.leftMotor,
    turnVelocity, sim.simx_opmode_blocking)
    res = sim.simxSetJointTargetVelocity(clientID, self.rightMotor,
    turnVelocity, sim.simx_opmode_blocking)
```

6.2 Design to using Ultrasonic sensors that choose sensors 5, 7, 8 and 9
Sensor 5 and 7 are used for detecting the wall on the front to make a robot turn
Sensor 8 and 9 are used for detecting the wall adjacent to the robot and to compare velocity of the robot



```
# Setup Sonars
res, self.frontLeftSonar = sim.simxGetObjectHandle(clientID,
'Pioneer_p3dx_ultrasonicSensor5',sim.simx_opmode_blocking)
res, self.Front_50_RightSonar = sim.simxGetObjectHandle(clientID,
'Pioneer_p3dx_ultrasonicSensor7',sim.simx_opmode_blocking)
res, self.RightSonar = sim.simxGetObjectHandle(clientID,
'Pioneer_p3dx_ultrasonicSensor8',sim.simx_opmode_blocking)
res, self.backRightSonar = sim.simxGetObjectHandle(clientID,
'Pioneer_p3dx_ultrasonicSensor9',sim.simx_opmode_blocking)
```

6.3 Implementing a PID

The initial values of implementing a PID is set all gain to 1

```
    Kp = 1 # Proportional Gain
    Ki = 1 # Integral Gain
    Kd = 1 # Derivative Gain
```

Variables needed for the PID controller are all set to 0

```
    # Variables
    cp = 0
    current_error = 0
    previous_error = 0
    sum_error = 0
    previous_error_derivative = 0
    current_error_derivative = 0
```

Creating 2 variables to get readings from detecting the wall

```
   # Distance variables
    d_R_front = robot.getDistanceReading(robot.RightSonar)
    d_R_back = robot.getDistanceReading(robot.backRightSonar)
```

The calculation to detect the wall on the right side.

```
  #calculate distance

    dist_robot_and_wall = d_R_front + d_R_back / 2 # distance between robot and
wall
    angle = d_R_front - d_R_back # angel of the wall
    dist_to_wall = math.cos(angle) == (d_R_front / d_R_back) # desired distance
between robot and wall
```

The angle distance is the distance between front and back (right sided) sonars and the wall, is our target. It is an error calculated to keep the robot adjacent to the wall. Using the calculation by implementing a PID, we get an output to keep the robot right beside the wall.

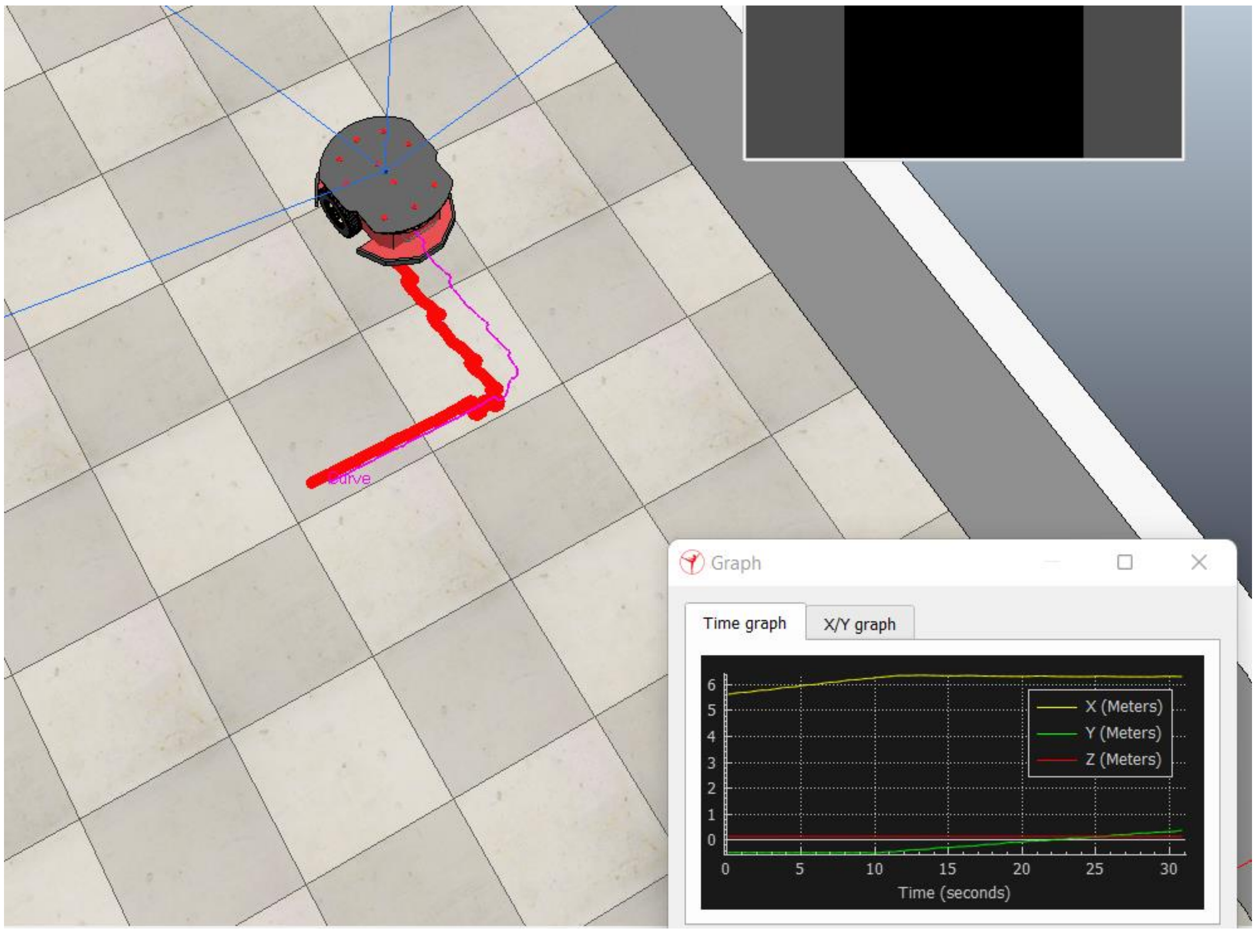
```
        # calculate
        current_error = dist_to_wall - dist_robot_and_wall
        sum_error = sum_error + (current_error * Dt)
        current_error_derivative = (current_error - previous_error) / Dt
        previous_error = current_error

        # PID Process
        cp = (Kp * current_error) + (Ki * sum_error) + (Kd *
current_error_derivative)
```

6.4 The wall following

There are four conditions the robot follows by using PID to detect and follow the wall

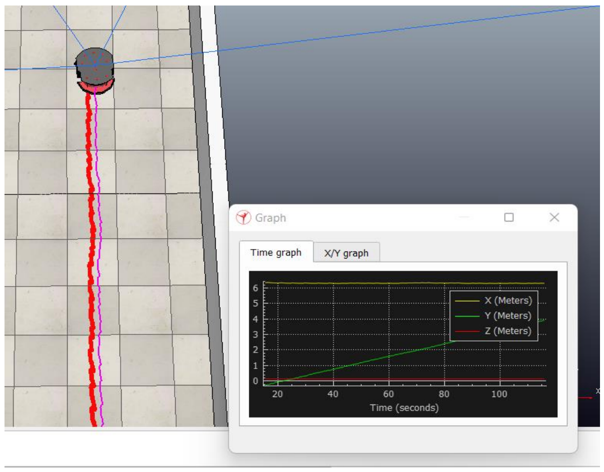
Firstly, If the robot detected the wall in front of robot (sensor 5 or 7). it will turn left to avoid the wall by using values of PID

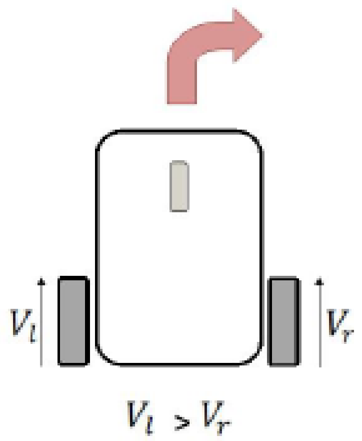
```
        if robot.getDistanceReading(robot.frontLeftSonar) <= 1: #threshold
value
            robot.curveCorner(-cp, cp) # velocity adjectment
            print("Wall detected in (L)front cp")  # the robot will turn
if a wall is detected

        elif robot.getDistanceReading(robot.Front_50_RightSonar) <=
dist_to_wall:
            robot.curveCorner(-cp, cp)  # velocity adjustment
            print("Wall detected in Right_front_50_degrees ") # the robot
will trun if a wall is detected;
```



Secondly, if the robot detectes no wall after path following for a while (right side). It will turn right to decrease the distance between the wall and using values of PID.

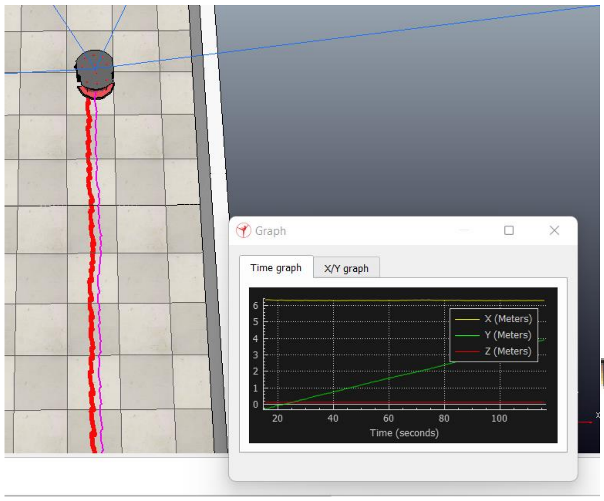
```
        elif robot.getDistanceReading(robot.backRightSonar) <= dist_to_wall and
detectionStateR == 1.0:
            robot.curveCorner(cp, angle)  # velocity adjustment
            print("fix position on the Right (1)") # the robot will turn if
detected on the right within the threshold value
```

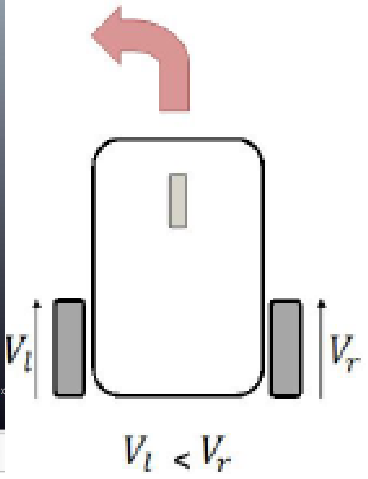




Next, if the robot detected a wall in front and on the right side, it will turn left to increase distance between the wall and using value of PID

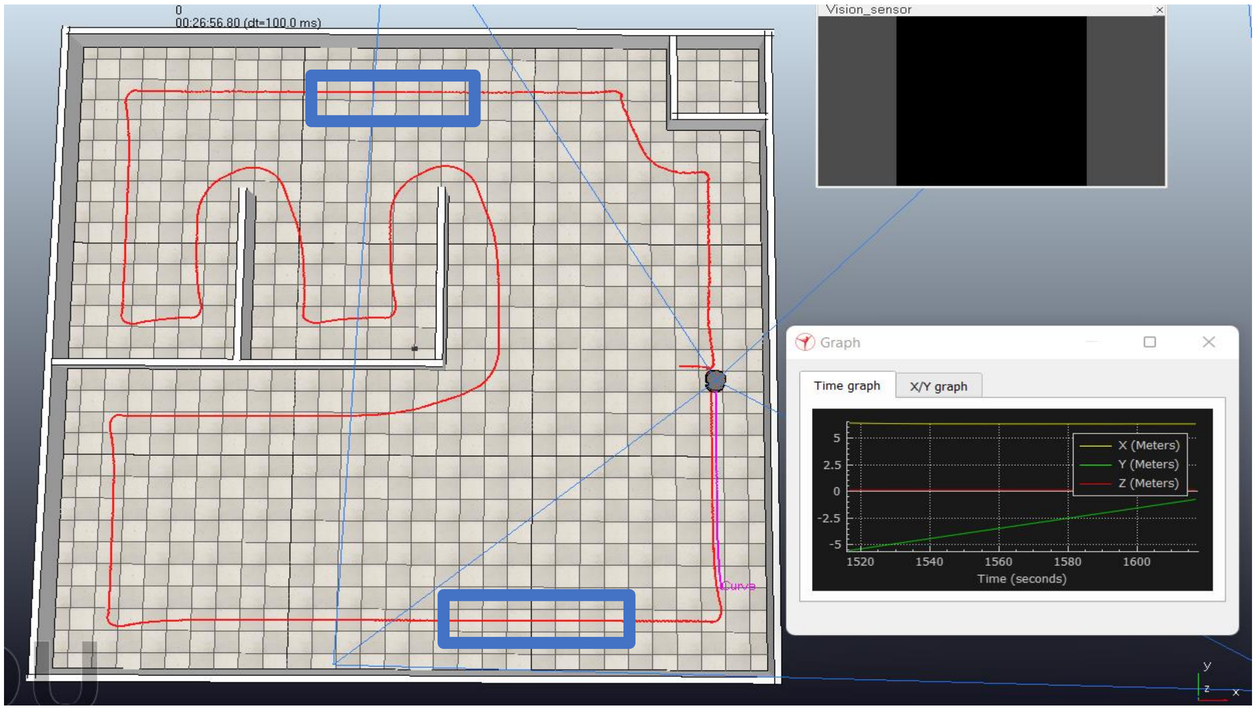
```
        elif robot.getDistanceReading(robot.backRightSonar) >= dist_to_wall and
detectionStateR == 1.0:
            robot.curveCorner(angle, cp)  # velocity adjustment
            print("fix position on the Left (angle,cp)") # the robot will turn to
adjust direction
```





Lastly, if the robot moves adjacent to the wall it compares between two sensors (90 degree (8) and (9)). If both the sensors are in equal distance from the wall, then the robot will move straight.

```
elif robot.getDistanceReading(robot.backRightSonar) == dist_to_wall and
robot.getDistanceReading(robot.RightSonar) == dist_to_wall and detectionStateR ==
1.0: # threshold value
            robot.move(1)   #velecity adjustment
            print("--------- [B-F] Perfectly Forward fix velocity ---------")   #
the robot will turn to adjust direction
```



As a result of the code, path following is better after the use of PID controller, which we used for tuning the robot to run randomly and follow the wall and automatic calculate the error values that is easy to modify or tuning.

7. References

[1] PID Theory Explained. https://www.ni.com/en-gb/innovations/white-papers/06/pid-theory-explained.html (accessed Nov. 23, 2021).

[2] What is Pioneer 3DX. https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf (accessed Nov. 23, 2021).

[3] What is Python. https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python (accessed Nov. 23, 2021).

[4] The robot simulator CoppeliaSim. https://www.coppeliarobotics.com (accessed Nov. 23, 2021).

[5] What is a PID controller : Working & Its applications. https://www.elprocus.com/the-working-of-a-pid-controller/ (accessed Nov.23, 2021)