Intelligent Systems and Robotics Assignment

April 21, 2022

Name:	Pankhuri Kulshrestha
Registration number:	2110376
Module Number:	CE801-7-SP
Project:	Intelligent Systems and Robotics Assignment Spring 2022
Link to GitHub:	https://github.com/panks11/CE801

Contents

1 Introduction

A robot can be defined as a physically situated agent capable to accomplish various tasks autonomously, sense and react according to the changes in the real world. A robot should be performing actions and making decisions with a body physically situated in the real world, contrary, to the Software agents similar in their roles and actions, however, existing in the virtual world of software and World Wide Web. Machines which are not bound to one physical location and have an additional capability to move around in their environment are known as 'Mobile robots'. Cleaning Robots, Unmanned Land Rovers, Drones, Entertainment Pets are few examples of such autonomous machines existing in present time. Robots have been fairly adapted in the real world to cover " The 3Ds " : jobs that are Dirty, Dull or Dangerous.

A robot is expected to adapt dynamic changes in the real world and apply human like decision making. A human like behavior requires huge planning and coordination of various components of technologies to work together. The field of locomotion, perception, cognition and navigation are the basic components which can describe the working of a robot. In this report we will discuss the robots under the locomotion, perception and sensor classification. The locomotion is the mechanism which makes a robot capable of moving in its environment of operation. Perception endows the robot the ability to perceive, comprehend, and reason about the surrounding environment. The sensors equip the robots to measure the conditions around them like light, temperature position of obstacles etc.

1.1 Types of Robot Platforms and Mobility

On a higher level, the robots can be classified under three existing modalities : ground, air and water.

1. Land Based:

   The robots designed to operate on the ground are known as Land Based robots. A popular name given to these robots is Unmanned Ground Vehicles or (UGV). UGV can belong to different sizes and shapes effecting their portability. Humanoid or anthropomorphic robots exhibiting human characteristics exist in labs like Sony Asimo and Honda P3. The mobile robots which can be fit within one or two bags are known as 'Man Packable' and the robots which can be carried by two men are known as Man Portable robots. According to their locomotion mechanism, they can be further classified as below :

   Wheeled and Legged Robots : The robots using wheels for movement are wheeled robots. Their kinematics and dynamics model differ as per the number of wheels used for moving a robot. The wheel movements of a robot are easier to control and stable. The robots using articulate limbs such as leg movements to provide locomotion are known as legged robots. These robots are more versatile and can traverse different terrains, however, they have additional complexity and power consumption. The robots having the ability to walk are much expensive and complex to control, however, they have better mobility on uneven terrain and soft surfaces.

   Types of Wheeled Robots:

   (a) Single Wheeled Robot: The unicycle robot is driven using a single a wheel. They are highly unstable and difficult to manage as they require both latitude and longitude control to provide stable movement.

   (b) Two Wheeled Robot: These robots consist of two wheels for movement. The two wheeled robots are easier to balance, however, not statically stable. The balancing problem in these robots is also known as inverted pendulum problem. A sensor is required to keep their balance, the robot moves in the direction it is tumbling. The direction in the robot is provided by the differential steering method.

   (c) Tricycle Drive Mobile Robot: These robots consist of three wheels, with two at the back and one in front, with Centre of Rotation situated in the middle of rear axle. The two configurations exist when the front wheel gives direction and also drive the motion while the back wheels are idle, or the front wheel is only used for direction and the back wheels drive the motion.

   (d) Four Wheeled Mobile Robot: The centre of gravity for these robots is situated inside the rectangle formed by the four wheels which make these robots more stable. The two

configurations exist for these robots are back steer and front drive or Front wheel and drive. The kinematic equations and control systems for these robots are similar to three legged robots.

(e) Differential Drive Robot: These robots have two independently driven wheels and other are castors wheels. The castor wheels have two degree of freedom and turning around an offset steering joint. The castor wheels provide stability to these robots.

Types of Legged Robots:

(a) One legged Robot: One legged robot use a hopping mechanism for navigation. These robots cannot stand still and need to continuously hop in order to maintain balance. The robot would jump slightly in the direction it is falling to catch itself. Uniroo is an example for one legged robot.

(b) Two Legged Robot: Walking robots simulate human walk. The motion of these robots is dependent on dynamic stability. The dynamic stability is the stability required to stand against reaction and inertial forces. They require to maintain the centre of gravity within stable area of dynamic stability. Robots like Atlas, Valkyrie have been developed to simulate tasks equivalent to human being in dangerous environments. The locomotion in these robots require a local path planning module which consists of sensors to provide data of the surrounding area, global path planning using a coordinate system to give the geometric information, footstep planning and motion planner.

(c) Multi Legged Robot: The robots with two or more legs can be classified under this category. These robots require to maintain static stability, that is the ability to stand under reaction forces. The center of gravity is maintained within the triangle formed by the feet contact points. They require high computational speed and power storage systems making them expensive.

Tracked robots: The robots employing treads or caterpillar tracks for movement are known as Tracked robots. Tracked robots make larger ground contact patches which improve their maneuverability on loose surface in comparison to wheeled robots. However, large ground contact patch require a skidding turn to change the direction of the robot , and therefore a large portion of the track slide against the surface. Due to the skidding action it is difficult to predict the accurate position of the robot and the direction of motion is also dependent on the ground friction.

2. Air Based:

   The robots which have the ability to fly commonly addressed as drones and perform aerial tasks. These robots are also known as Unmanned Aerial vehicles (UAVs). The UAV are often categorized on their size and their form.

   (a) Fixed Wing: The robots are similar to aeroplanes and operates without a human pilot onboard. They are usually controlled remotely and widely used for military purposes. U.S. Department of Defense Predators and Global hawks are examples.

   (b) Rotor Craft: The robots are similar to helicopters and perform vertical takeoff and landings. Multi-rotor helicopters are very popular for inspection and many other applications. Yamaha RMAX is a commercial example of an unmanned helicopter used in Japan for crop dusting in small, terraced plots.

3. Water Based:
   The robots that can operate under water are also known as Unmanned Marine Vehicles. The robots are capable to swim in the water. They take advantage of the remotely operated vehicles and humanoid robots to perform various tasks. Ocean One is an example which explores the ocean bed. The design of these robots are inspired by the coordination of bird flocks and fish schools.

1.2 Sensors

In Robotics sensors are required to provide the environment data to a robot. They are required for mobility and navigation planning. The data provided by the sensors is real time and provides a dynamic feedback utilized in navigating the robot on the correct path and also to stabilize. The term transducer is often used interchangeably with a sensor. A transducer is the mechanism which

transforms the energy associated with what is being measured into another form of energy. A sensor receives energy and transmits a signal for processing .The sensors can be broadly classified as :

1. Internal Sensors: The sensors which are required to know the aspects of the robot itself are known as internal sensors. The parameters such as the speed, position, velocity of the robot, any real time changes in the environment are recorded by such sensors.

   (a) Internal Position Sensors: The sensors which are required to sense an object at a small distance with or without contact.
   Contact Sensors are the sensors which measure the parameters with actual contact. For example, Micro switch used to measure the danger situation, potentiometer used to measure position, strain gauges used to measure force make actual contact with the surfaces.
   Potentiometers are three terminal resistors with a rotating contact that forms a voltage divider. They can also be called position transducers. It controls the magnitude of current in a circuit by changing the resistance.

   Non-Contact sensors can measure the parameters from a distance without contacting any object or surface. The Optical sensors, optical encoders are used to measure the position and velocity of a moving object, Synchros to measure angular displacement, Resolvers are used to measure degree of rotation from a distance.
   Optical encoder is a sensor which uses Optical pulses to calculate the distance. The optical encoder consist of a rotating disc with slits, a light emitting source and a photo sensor. When the disc rotates an optical pulse is generated depending on the light emitted from the light emitting source passes through the slits of disc. The optical sensors can be incremental or absolute. The relative encoders capture how many angles have you moved before and after movement. This is also known as incremental method. The incremental Optical encoders are cheap and easy to use however cannot determine the direction of rotation. The absolute angle detection type encoder calculate how many degrees are you from the home position. The output of the absolute optical encoder is a digital serial code or analog voltage in response to instructions from microcomputer. Both incremental and absolute encoders have advantages and disadvantages and should be chosen according to the purpose of detection.

   (b) Internal State Sensors: The internal state sensors are responsible to record the internal state of a robot for example velocity, acceleration, orientation. Velocity sensors, gyroscope, Optical gyroscope, internal measurement unit are some of the sensors utilized for the same.

2. External Sensors: These sensors collect data which is required to control the navigation. They monitor the changes in a robot environment. They measure the ambient environmental energy entering the sensor such as microphones, temperature probes, Charged Coupled Devices. They are expected to know an obstacles location or reaching a goal.

   (a) External State sensors: External State sensors is required to capture the environment data in which the robot is located. These sensors help the robot in path planning and environment learning. The sensors like tactile sensor, proximity sensors and range-finding census help in achieving the same. Computer vision help the robot in performing image analysis and segmentation, this provides the capability to understand the environment in terms of color, edges, texture, motion and depth. **Tactile sensing** is used when there is not adequate illumination to detect objects. The switches, strain gauges and pressure transducers provide information about the contact between the object and the robot.
   Proximity sensors often emit electromagnetic field or infrared radiation and detect any changes in the field or return of the signal. **Range finding** sensors are used to avoid collision map buildings, they measure the time a signal requires to reach and return from an object.

2 Implementation of PID Control Strategy

The PID algorithm also known as "Proportional, Integral, Derivative" algorithm is a control algorithm which is widely accepted across the industries for controlling electronic machines such as

robots. It is a computer algorithm which is used in combination with sensors to calculate the offset of an actual and a desired value. In a closed looped system, the difference in the actual and desired values are provided as a feedback to control and drive the system. The algorithm helps in achieving precise and smooth robot movement. In case of two wheeler robot, it is possible the motors on the left and right chassis are not matched and robot is drifted towards one direction, to overcome such scenarios a PID algorithm can be used to achieve a straight movement.

2.1 PID Theory

1. Proportional Response: The proportional component depends on difference between the desired and the actual value. The output contains the component directly proportional to error.

2. Integral Response: The integral component sums the error term over time. If even a small error term is recorded it will cause the integral component to increase slowly.

3. Derivative Response: The derivative component is proportional to the rate of change of the real time recorded value. If these changes are rapid it decreases the output.

4. Tuning: The process to obtain the optimal values or gains for P, I and D components of the output signal to get an ideal response from the control system is known as Tuning.There are different methods of tuning, we have used "guess and check" method to control our robots.

2.2 PID Strategy

The PID strategy is used to navigate the robot from the starting point, passing between the two gaps, to the stopping point. The difference in the expected Laser range and the Robot heading angle has been used to tune the Robot angular velocity. The robot navigation is divided into three stages:

1. First Stage:PID wallFollowing: The first stage starts from the starting point to the position of the First Gap, where the robot moves forward following the wall on the left side. The distance from the left wall is measured using Left Range Reading of the Laser. The desired left range of 0.3 is maintained using the output of the PID control function which tunes the Forward Right Speed.

2. Second stage: PID pass1stGap: The second stage starts before the first gap to the second gap, where the robot moves in the right direction maintaining the robot head angle at -0.7.The desired heading angle of -0.7 is maintained using the output of the PID control function which tunes the Forward Right Speed.

3. Third stage: PID pass2ndGap: The third stage starts before the second gap to the stopping point, where the robot moves in the right direction maintaining the robot head angle at -1.3. The desired heading angle of -1.3 is maintained using the output of the PID control function which tunes the Forward Right Speed.

2.3 Flow Chart

The Flow Chart in Figure 1 explains the PID strategy used for driving the robot. Figure 2, 3, 4 describe the PID stages in detail.

2.4 Result Graphs

After successfully running the robot from the start to the charger position following charts have been added to the report:

1. Robot Trajectory Graph: Figure 6 plots the graph of the position of Robot in X-Y dimension. The robot moves in the grid of dimensions 2.5(m) X 2.5(m). As visible in the graph, the robot starts from position (0.3,0.3) travels horizontally in Y direction, until the First Gap, makes a Right Turn travels vertically in the X direction, until the Second Gap, makes a Right turn to pass the Second Gap and stops at the Charger location (2.25,0.3) approximately.

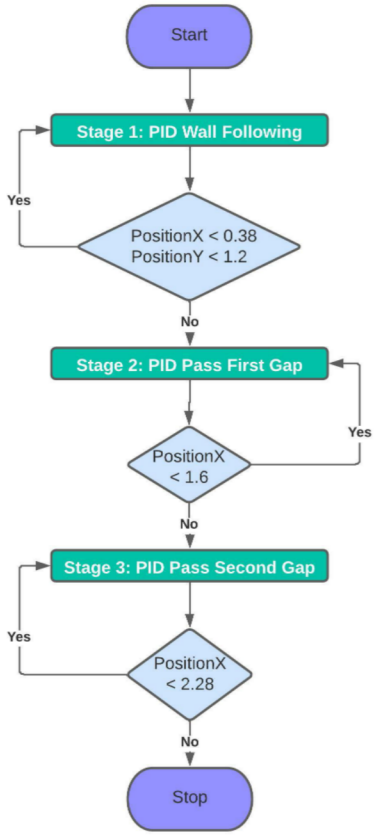

Figure 1: PID Strategy Flow Chart

2. Robot Velocity Graph: Figure 7 plots the Forward or Linear velocity and the Angular Velocity of the robot. The linear velocity does not vary much, and the robot continues to move with a velocity ranging from 0.25-0.3 in forward direction. The angular velocity fluctuates in different Stages to maintain the robot Trajectory. In the first stage, the angular speed is tuned to maintain distance from the wall on the left side. In second and third stage, angular speed is tuned to maintain the Robot heading angle in right direction.

3. Laser Data Graph: Figure 5 plot the Laser readings collected during the robot movement in the grid. The mleft, left, mright, right, front ranges and Robot heading angle captured by the sensors are converted from Polar coordinates to Cartesian coordinates to plot the position of the obstacles and the Grid. As the plot depicts, there are four obstacles present in the grid and the grid is of square shape with dimension of 2.5 X 2.5 (m).

3 Implementation of Fuzzy Control Strategy

A fuzzy control system is based on a mathematical system that analyzes analog input values in terms of logical variables which can take continuous values between 0 and 1, contrary to the discrete values usually used in digital devices. Fuzzy control systems are huge success in controlling various industrial devices as they are capable to make decisions similar to a human being. In a Fuzzy logic controller, the crisp input values are converted into fuzzy values. A Fuzzy Knowledge Base is used to stores the input and fuzzy value relationship. A membership function is used to define the input variables to the fuzzy rule base and the output variables. An Inference Engine simulates human decisions by approximate reasoning. A Defuzzifier convert the fuzzy values from the Inference Engine back to crisp values.

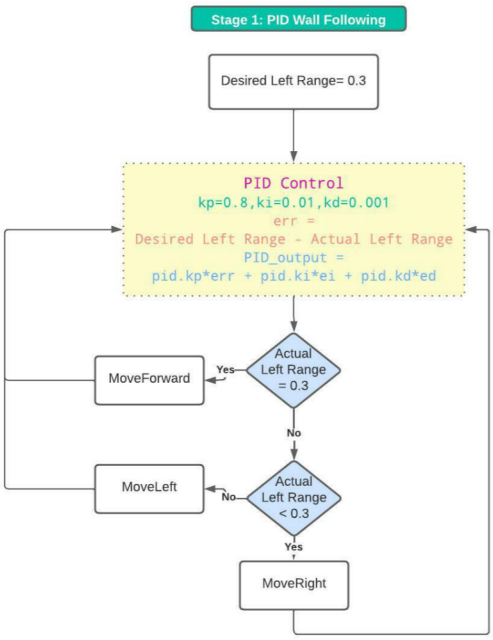

Figure 2: Detailed PID Stage 1

3.1 Fuzzy Control Strategy

The Fuzzy control strategy is used to navigate the robot from the starting point, passing between the two gaps, to the stopping point. The robot velocities are tuned according to the near, medium and far Laser ranges. The robot navigation is divided into four stages:

1. First Stage: Fuzzy Wall Following: The first stage starts from the starting point to the position of the First Gap, where the robot moves forward following the wall on the left side. The leftRange and mleftRange are used as the input of the fuzzy logic controller to control robot velocity and steering speed, so that the robot can follow the left wall smoothly. Since, the expected distance from wall is 0.3 for leftRange and 0.4 for mleftRange we can define
   leftRange: near <0.3m, 0.3m <medium <0.5m, and far >0.5m
   mleftRange: near <0.4m, 0.4m <medium <0.6m, and far >0.6m
   Table in figure 8 present some fuzzy rules used for the robot to follow the left wall smoothly.

2. Second stage: Fuzzy Pass First Gap: The second stage starts before the first gap to the second gap. The leftRange and mleftRange are used as the input of the fuzzy logic controller to control robot velocity and steering speed, so that the robot can pass the first obstacle smoothly. Since, the expected distance from left obstacle is 0.4 for leftRange and mleftRange we can define
   leftRange: near <0.4m, 0.4m <medium <0.6m, and far >0.6m
   mleftRange: near <0.4m, 0.4m <medium <0.8m, and far >0.8m
   Table in figure 8 present some fuzzy rules used for the robot.

3. Third stage: Fuzzy Pass Second gap: The third stage starts before the second gap and ends after passing it, the mleftRange and mRightRange are used as the input of the fuzzy logic controller to control robot velocity and steering speed, so that the robot can pass the second obstacle smoothly. Since, the expected distance from left and right obstacle is 0.5 for mleftRange and mRightRange we can define
   mleftRange: near <0.5m, 0.5m <medium <1m, and far >1m
   mRightRange: near <0.5m, 0.5m <medium <1m, and far >1m
   Table in figure 8 present some fuzzy rules used for the robot.

4. Fourth stage: Fuzzy Reach Charger Position: The fourth stage starts after the second gap to the stopping point. The mleftRange and fronRange are used as the input of the fuzzy logic controller to control robot velocity and steering speed, so that the robot can reach the stopping point smoothly. Since, the expected distance from left and front wall is 0.5 and 0.3

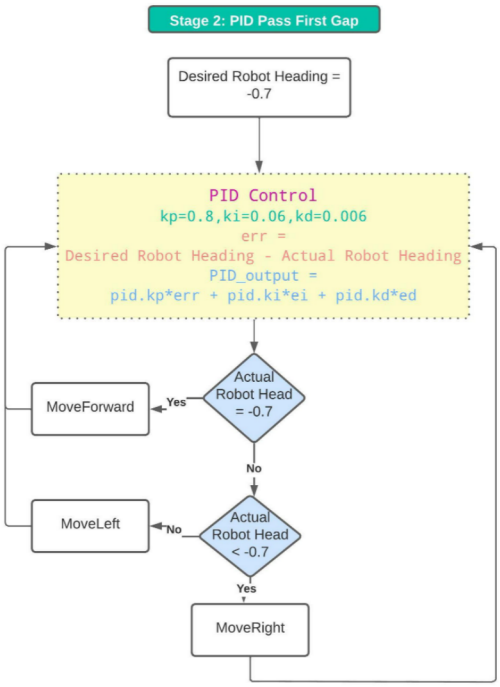

Figure 3: Detailed PID Stage 2

for mleftRange and fronRange we can define
mleftRange: near <0.5m, and far >0.5m
fronRange: near <0.3m, and far >0.3m
Table in figure 8 present some fuzzy rules used for the robot.

3.2 Flow Chart

The Flow Chart in Figure 9 explains the Fuzzy Control strategy used for driving the robot.

3.3 Result Graphs

After successfully running the robot from the start to the charger position following charts have been added to the report:

1. Robot Trajectory Graph: Figure 11 plots the graph of the position of Robot in X-Y dimension explained in Section 2.4: Robot Trajectory Graph

2. Robot Velocity Graph: Figure 12 plots the Forward or Linear velocity and the Angular Velocity of the robot. The linear velocity and the steering or angular velocity varies in different stages of robot trajectory due to fuzzy logic control.

3. Laser Data Graph: Figure 10 plot the Laser readings collected during the robot movement in the grid. Graph explained in Section 2.4: Laser Data Graph

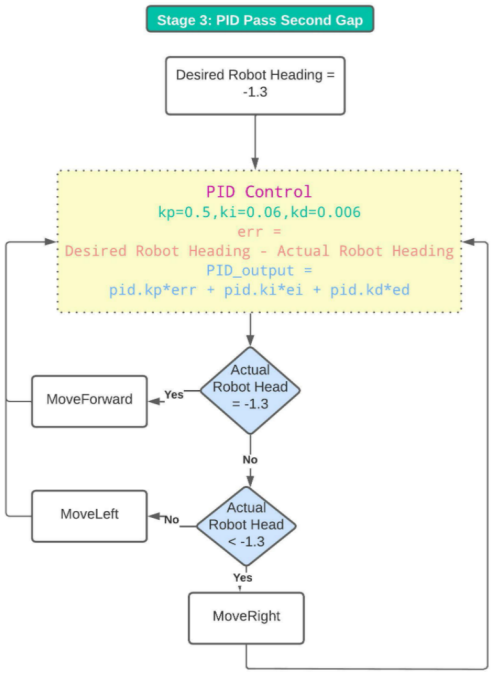

Figure 4: Detailed PID Stage 3

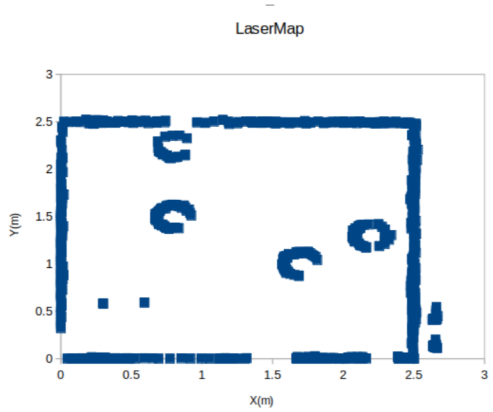

Figure 5: PID Laser Map

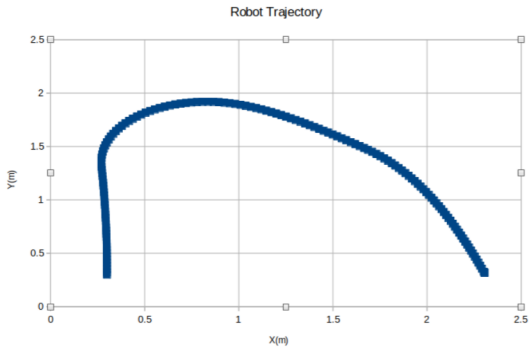

Figure 6: PID Robot Trajectory

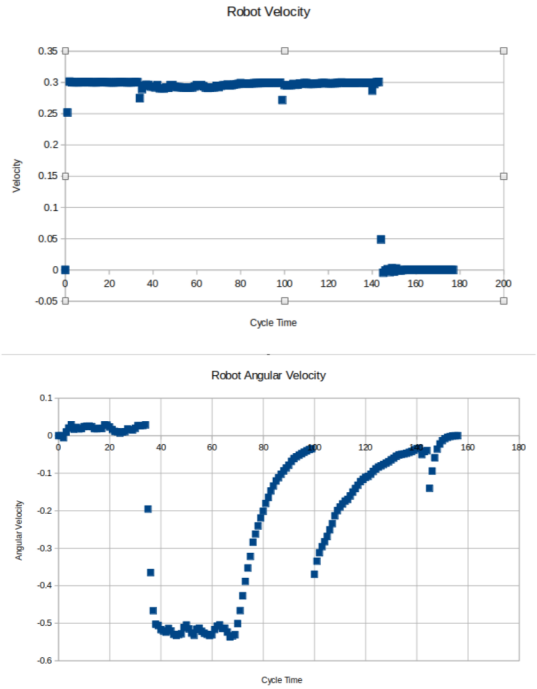

Figure 7: PID Robot Linear Velocity and Angular Velocity

Stage 1: Fuzzy Wall Following			
Left Range	**M Left Range**	**Robot velocity**	**Steering speed**
Near	Near	Low	TURN_RIGHT_SPEED_LOW
Near	Medium	Low	TURN_RIGHT_SPEED_LOW
Near	Far	Low	TURN_LEFT_SPEED_LOW
Medium	Near	Middle	TURN_RIGHT_SPEED_LOW
Medium	Medium	High	TURN_SPEED_ZERO
Medium	Far	Middle	TURN_LEFT_SPEED_LOW
Far	Near	Middle	TURN_RIGHT_SPEED_MIDDLE
Far	Medium	Middle	TURN_RIGHT_SPEED_MIDDLE
Far	Far	High	TURN_LEFT_SPEED_LOW

Stage 2: Fuzzy Pass First Gap			
Left Range	**M Left Range**	**Robot velocity**	**Steering speed**
Near	Near	Low	TURN_RIGHT_SPEED_LOW
Near	Medium	Low	TURN_RIGHT_SPEED_LOW
Near	Far	Low	TURN_LEFT_SPEED_LOW
Medium	Near	Middle	TURN_RIGHT_SPEED_MIDDLE
Medium	Medium	Middle	TURN_RIGHT_SPEED_MIDDLE
Medium	Far	Middle	TURN_RIGHT_SPEED_MIDDLE
Far	Near	Middle	TURN_RIGHT_SPEED_MIDDLE
Far	Medium	Middle	TURN_RIGHT_SPEED_MIDDLE
Far	Far	Middle	TURN_RIGHT_SPEED_MIDDLE

Stage 3: Fuzzy Pass Second gap			
M Left Range	**mRightRange**	**Robot velocity**	**Steering speed**
Near	Near	Low	TURN_SPEED_ZERO
Near	Medium	Low	TURN_RIGHT_SPEED_LOW
Near	Far	Low	TURN_RIGHT_SPEED_MIDDLE
Medium	Near	Middle	TURN_LEFT_SPEED_LOW
Medium	Medium	Middle	TURN_RIGHT_SPEED_LOW
Medium	Far	Low	TURN_RIGHT_SPEED_MIDDLE
Far	Near	Middle	TURN_LEFT_SPEED_MIDDLE
Far	Medium	Middle	TURN_RIGHT_SPEED_MIDDLE
Far	Far	Middle	TURN_SPEED_ZERO

Stage 4: Fuzzy Reach Charger Position			
M Left Range	**Front Range**	**Robot velocity**	**Steering speed**
Near	Near	Zero	TURN_SPEED_ZERO
Near	Far	Low	TURN_RIGHT_SPEED_LOW
Far	Near	Low	TURN_SPEED_ZERO
Far	Far	Middle	TURN_SPEED_ZERO

Figure 8: Fuzzy Logic Table

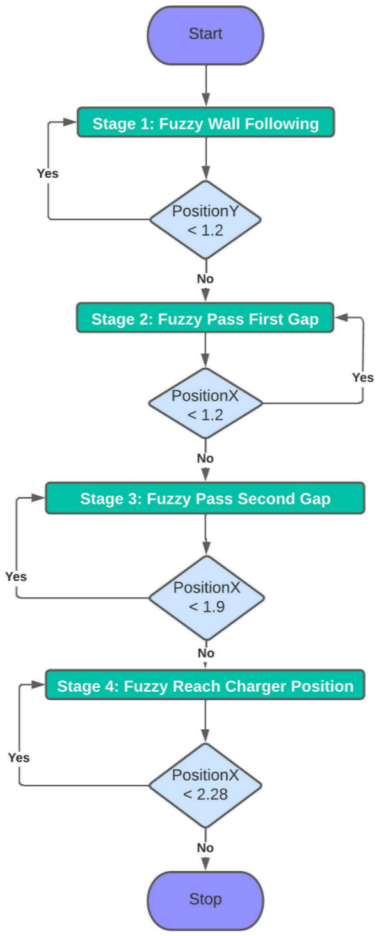

Figure 9: Fuzzy Flow Chart

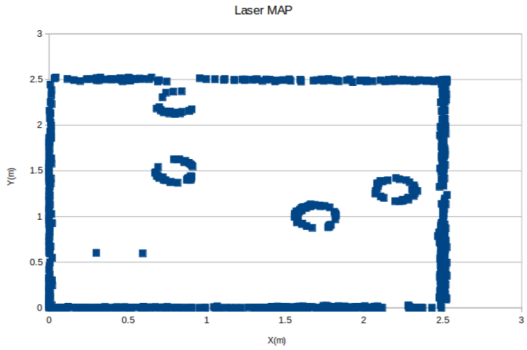

Figure 10: Fuzzy Laser Map

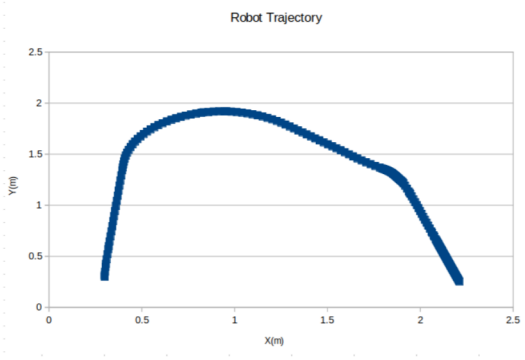

Figure 11: Fuzzy Robot Trajectory

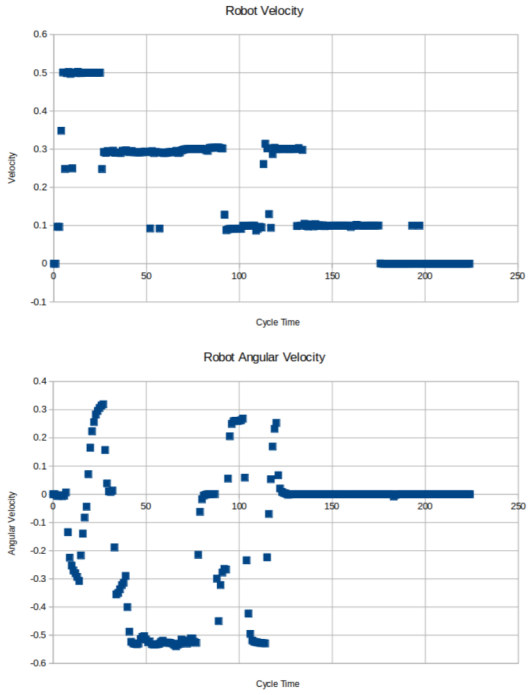

Figure 12: Fuzzy Robot Linear Velocity and Angular Velocity

4 Appendix

```
//***********************************
// PID control Logic created
//***********************************
struct PID_para{double kp, ki, kd, ei_pre, ed_pre, Max_output;};

//***********************************
// Function to calulate PID error and output value
//***********************************
double PID_control(PID_para pid, double setPoint, double measuredData)
{
    //double err = measuredData - setPoint;
    double err = setPoint - measuredData;
    double ei = pid.ei_pre + err;
    double ed = err - pid.ed_pre;
    pid.ei_pre = ei;
    pid.ed_pre = ed;
    double output = pid.kp*err + pid.ki*ei + pid.kd*ed;
    if (output > pid.Max_output )
        {
        output = pid.Max_output;
        }
    else if(output < -pid.Max_output)
        {
            output = -pid.Max_output;
        }
    return output;
}

//***********************************
//PID First Stage function
//***********************************
void Stopper::PID_wallFollowing(double forwardSpeed, double laserData)
{
    PID_para controller;
    double landmark1_toWall = 0.3;
    controller.kp = 0.8, controller.ki = 0.01;
    controller.kd = 0.001, controller.Max_output = 0.6;
    double PID_output = PID_control(controller, landmark1_toWall, laserData);
    cout << "Stage1:Wall Following : Right Speed is :" << PID_output << " Left Distance From Wall
    moveForwardRight(forwardSpeed, PID_output);
}

//***********************************
//PID Second Stage function
//***********************************
void Stopper::PID_pass1stGap(double moveSpeed, double robotHeading)
{
    PID_para controller;
    double robotHeadingGap1 = -0.7; // the robot heading should be 0 degree
    controller.kp = 0.8, controller.ki = 0.06;
    controller.kd = 0.006, controller.ei_pre = 0;
    controller.ed_pre = 0, controller.Max_output = 0.6;
    double PID_output = PID_control(controller, robotHeadingGap1, robotHeading);
    cout << "Stage2:Pass Gap 1 : Right Speed is :" << PID_output << " Heading Angle is : "<< robot
    moveForwardRight(moveSpeed, PID_output);
}

//***********************************
//PID Third  Stage function
```

```
//***********************************
void Stopper::PID_pass2ndGap(double moveSpeed, double robotHeading)
{
    PID_para controller;
    double robotHeadingGap1 = -1.3; // the robot heading should be 0 degree
    controller.kp = 0.5, controller.ki = 0.06;
    controller.kd = 0.006, controller.ei_pre = 0;
    controller.ed_pre = 0, controller.Max_output = 0.6;
    double PID_output = PID_control(controller, robotHeadingGap1, robotHeading);
    cout << "Stage2:Pass Gap 2 : Right Speed is :" << PID_output << " Heading Angle is : "<< robot
    moveForwardRight(moveSpeed, PID_output);
}

//***********************************
// Fuzzy control Logic created
//***********************************
//***********************************
//Fuzzy First Stage function
//***********************************
void Stopper::Fuzzy_wallFollowing(double laserData1, double laserData2)
{
    int fuzzySensor1, fuzzySensor2;
    // sensor data fuzzification
    if (laserData1 < 0.3) fuzzySensor1 = 1; // The robot is near to the wall
    else if (laserData1 < 0.5) fuzzySensor1 = 2; // The robot is on the right distance
    else fuzzySensor1 = 3; // The robot is far from the wall;

    if (laserData2 < 0.4) fuzzySensor2 = 1; // The robot is near to the wall
    else if (laserData2 < 0.6) fuzzySensor2 = 2; // The robot at the right distance;
    else fuzzySensor2 = 3; // The robot is far from the wall;
    // Fuzzy rule base and control output

    if (fuzzySensor1 == 1 && fuzzySensor2 == 1)
        {moveForwardRight(FORWARD_SPEED_LOW, TURN_RIGHT_SPEED_LOW);
        cout << "Stage1:Wall Follow :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuz
    else if (fuzzySensor1 == 1 && fuzzySensor2 == 2)
        {moveForwardRight(FORWARD_SPEED_LOW, TURN_RIGHT_SPEED_LOW);
        cout << "Stage1:Wall Follow :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuz
    else if (fuzzySensor1 == 1 && fuzzySensor2 == 3)
        {moveForwardRight(FORWARD_SPEED_LOW, TURN_LEFT_SPEED_LOW);
        cout << "Stage1:Wall Follow :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuz
    else if (fuzzySensor1 == 2 && fuzzySensor2 == 1)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_LOW);
        cout << "Stage1:Wall Follow :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuz
    else if (fuzzySensor1 == 2 && fuzzySensor2 == 2)
        {moveForwardRight(FORWARD_SPEED_HIGH, TURN_SPEED_ZERO);
        cout << "Stage1:Wall Follow :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuz
    else if (fuzzySensor1 == 2 && fuzzySensor2 == 3)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_LEFT_SPEED_LOW);
        cout << "Stage1:Wall Follow :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuz
    else if (fuzzySensor1 == 3 && fuzzySensor2 == 1)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
        cout << "Stage1:Wall Follow :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuz
    else if (fuzzySensor1 == 3 && fuzzySensor2 == 2)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
        cout << "Stage1:Wall Follow :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuz
    else if (fuzzySensor1 == 3 && fuzzySensor2 == 3)
        {moveForwardRight(FORWARD_SPEED_HIGH, TURN_LEFT_SPEED_LOW);
        cout << "Stage1:Wall Follow :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuz
    else RCLCPP_INFO(this->get_logger(),"Following the left wall");
```

```
}

//***********************************
//Fuzzy Second Stage function
//***********************************
void Stopper::Fuzzy_to1stGap(double laserData1, double laserData2)
{
    int fuzzySensor1, fuzzySensor2;
    // sensor data fuzzification
    if (laserData1 < 0.4) fuzzySensor1 = 1;
    else if (laserData1 < 0.6) fuzzySensor1 = 2;
    else fuzzySensor1 = 3;

    if (laserData2 < 0.4) fuzzySensor2 = 1;
    else if (laserData2 < 0.8) fuzzySensor2 = 2;
    else fuzzySensor2 = 3;

    // Fuzzy rule base and control output
    if (fuzzySensor1 == 1 && fuzzySensor2 == 1)
        {moveForwardRight(FORWARD_SPEED_LOW, TURN_RIGHT_SPEED_LOW);
        cout << "Stage2: Gap1 :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuzzySens
    else if (fuzzySensor1 == 1 && fuzzySensor2 == 2)
        {moveForwardRight(FORWARD_SPEED_LOW, TURN_RIGHT_SPEED_LOW);
        cout << "Stage2: Gap1 :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuzzySens
    else if (fuzzySensor1 == 1 && fuzzySensor2 == 3)
        {moveForwardRight(FORWARD_SPEED_LOW, TURN_LEFT_SPEED_LOW);
        cout << "Stage2: Gap1 :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuzzySens
    else if (fuzzySensor1 == 2 && fuzzySensor2 == 1)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
        cout << "Stage2: Gap1 :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuzzySens
    else if (fuzzySensor1 == 2 && fuzzySensor2 == 2)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
        cout << "Stage2: Gap1 :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuzzySens
    else if (fuzzySensor1 == 2 && fuzzySensor2 == 3)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
        cout << "Stage2: Gap1 :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuzzySens
    else if (fuzzySensor1 == 3 && fuzzySensor2 == 1)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
        cout << "Stage2: Gap1 :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuzzySens
    else if (fuzzySensor1 == 3 && fuzzySensor2 == 2)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
        cout << "Stage2: Gap1 :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuzzySens
    else if (fuzzySensor1 == 3 && fuzzySensor2 == 3)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
        cout << "Stage2: Gap1 :" << "fuzzySensor1: " <<fuzzySensor1<<" ,fuzzySensor2 :"<<fuzzySens
    else RCLCPP_INFO(this->get_logger(), "Going through the 1st gap");
}

//***********************************
//Fuzzy Third Stage function
//***********************************
void Stopper::Fuzzy_to2ndGap(double laserData1, double laserData2)
{
    int mleftSensor, mrightSensor;
    // sensor data fuzzification
    if (laserData1 < 0.5) mleftSensor = 1;
    else if (laserData1 < 1) mleftSensor = 2;
    else mleftSensor = 3;

    if (laserData2 < 0.5) mrightSensor = 1;
    else if (laserData2 < 1) mrightSensor = 2;
```

```
    else mrightSensor = 3;

    // Fuzzy rule base and control output
    if (mleftSensor == 1 && mrightSensor == 1)
        {moveForwardRight(FORWARD_SPEED_LOW, TURN_SPEED_ZERO);
        cout << "Stage3: Gap2 :" << "mleftSensor: " <<mleftSensor<<" ,mrightSensor :"<<mrightSens
    else if (mleftSensor == 1 && mrightSensor == 2)
        {moveForwardRight(FORWARD_SPEED_LOW, TURN_RIGHT_SPEED_LOW);
        cout << "Stage3: Gap2 :" << "mleftSensor: " <<mleftSensor<<" ,mrightSensor :"<<mrightSens
    else if (mleftSensor == 1 && mrightSensor == 3)
        {moveForwardRight(FORWARD_SPEED_LOW, TURN_RIGHT_SPEED_MIDDLE);
        cout << "Stage3: Gap2 :" << "mleftSensor: " <<mleftSensor<<" ,mrightSensor :"<<mrightSens
    else if (mleftSensor == 2 && mrightSensor == 1)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_LEFT_SPEED_LOW);
        cout << "Stage3: Gap2 :" << "mleftSensor: " <<mleftSensor<<" ,mrightSensor :"<<mrightSens
    else if (mleftSensor == 2 && mrightSensor == 2)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_LOW);
        cout << "Stage3: Gap2 :" << "mleftSensor: " <<mleftSensor<<" ,mrightSensor :"<<mrightSens
    else if (mleftSensor == 2 && mrightSensor == 3)
        {moveForwardRight(FORWARD_SPEED_LOW, TURN_RIGHT_SPEED_MIDDLE);
        cout << "Stage3: Gap2 :" << "mleftSensor: " <<mleftSensor<<" ,mrightSensor :"<<mrightSens
    else if (mleftSensor == 3 && mrightSensor == 1)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_LEFT_SPEED_MIDDLE);
        cout << "Stage3: Gap2 :" << "mleftSensor: " <<mleftSensor<<" ,mrightSensor :"<<mrightSens
    else if (mleftSensor == 3 && mrightSensor == 2)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
        cout << "Stage3: Gap2 :" << "mleftSensor: " <<mleftSensor<<" ,mrightSensor :"<<mrightSens
    else if (mleftSensor == 3 && mrightSensor == 3)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_SPEED_ZERO);
        cout << "Stage3: Gap2 :" << "mleftSensor: " <<mleftSensor<<" ,mrightSensor :"<<mrightSens
    else RCLCPP_INFO(this->get_logger(), "Going through the 1st gap");
}

//***********************************
//Fuzzy Fourth Stage function
//***********************************
void Stopper::Fuzzy_ReachStop(double laserData1, double laserData2)
{
    int leftSensor, frontSensor;
    // sensor data fuzzification
    if (laserData1 < 0.5) leftSensor = 1;
    else leftSensor = 2;

    if (laserData2 < 0.3) frontSensor = 1;
    else frontSensor = 2;

    // Fuzzy rule base and control output
    if (leftSensor == 1 && frontSensor == 1)
        {moveForwardRight(TURN_SPEED_ZERO, TURN_SPEED_ZERO);
        cout << "Stage3: Gap2 :" << "leftSensor: " <<leftSensor<<" ,frontSensor :"<<frontSensor<<
    else if (leftSensor == 1 && frontSensor == 2)
        {moveForwardRight(FORWARD_SPEED_LOW, TURN_SPEED_ZERO);
        cout << "Stage3: Gap2 :" << "leftSensor: " <<leftSensor<<" ,frontSensor :"<<frontSensor<<
    else if (leftSensor == 2 && frontSensor == 1)
        {moveForwardRight(FORWARD_SPEED_LOW, TURN_SPEED_ZERO);
        cout << "Stage3: Gap2 :" << "leftSensor: " <<leftSensor<<" ,frontSensor :"<<frontSensor<<
    else if (leftSensor == 2 && frontSensor == 2)
        {moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_SPEED_ZERO);
        cout << "Stage3: Gap2 :" << "leftSensor: " <<leftSensor<<" ,frontSensor :"<<frontSensor<<
    else RCLCPP_INFO(this->get_logger(), "Going through the 1st gap");
}
```