Lab 7 – Fuzzy Controller for Robot Navigation

In Lab 6, you have created PID controllers to control the robot moving from home, go through two gaps, and reach to the charger position. In this lab, you should create Fuzzy controllers to implement the same navigation task.

Task 2: You should create Fuzzy controllers to control the simulated robot moving from home to the charger in an environment shown in Figure 7.1.

- You should write C/C++ code to implement Fuzzy controllers in a ROS environment.
- Your code should drive the robot to go through two gaps and stop at the charger position.
- You should adjust the parameters of Fuzzy controllers to make the trajectory smooth.
- The odometry and laser data should be collected and saved into files for plotting.

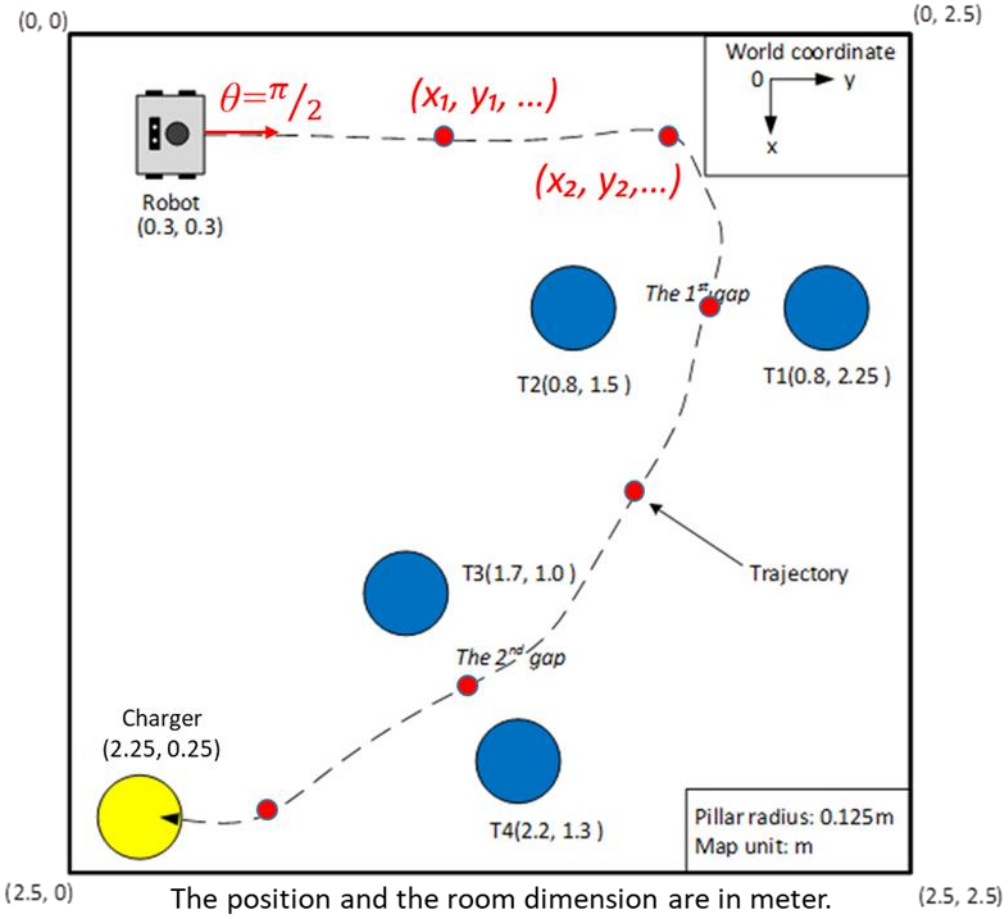


Figure 7.1 The navigation task

7.1 Basic Concepts:

Fuzzy logic is applied with great success in various control applications. Almost all the consumer products have fuzzy control. Some of examples include air-conditioners controlling your room temperature, anti-braking systems used in vehicles, controllers used in traffic lights, washing machines, large economic systems, etc.

Why Use Fuzzy Logic in Control Systems

A control system is an arrangement of physical components designed to alter another physical system so that it exhibits certain desired characteristics. Following are some reasons of using Fuzzy Logic in Control Systems −

- In traditional control systems, we need to build their models and their objective functions precisely, which makes them difficult to apply in many cases.
- By applying fuzzy logic in a control system, we do not need to build the system model and objective functions. Instead, we can utilize the human expertise and experience .
- To build a fuzzy logic controller, we can use the fuzzy control rules, i.e. the IF-THEN rules, in designing a Fuzzy logic controller for a robotic system.

Architecture of Fuzzy Logic Control

The following diagram shows the architecture of Fuzzy Logic Control (FLC).

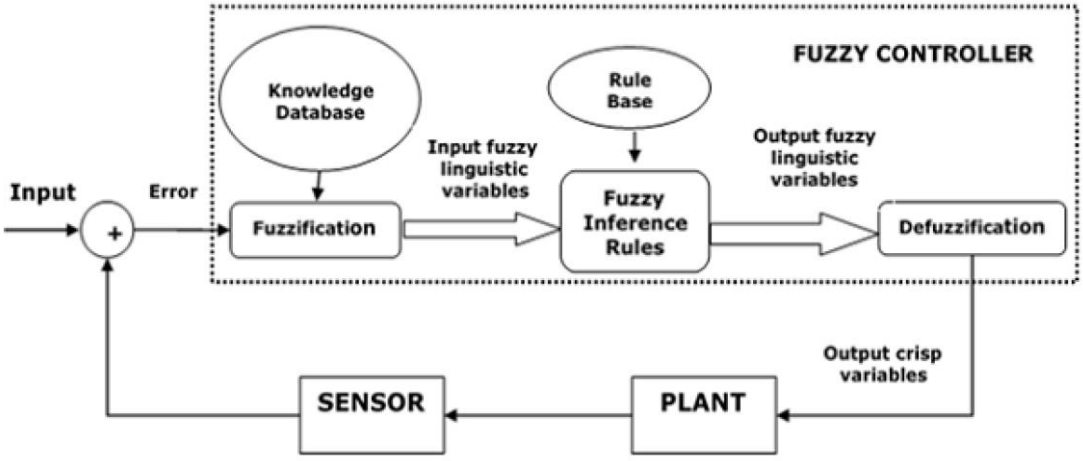


Figure 6.1 Fuzzy Control System Architecture (Plant = Robot)

Major Components of FLC

Followings are the major components of the FLC as shown in the above figure −

- Fuzzifier − Its role is to convert the crisp input values into fuzzy values.
- Fuzzy Knowledge Base − It stores the knowledge about all the input-output fuzzy relationships. It also has the membership function which defines the input variables to the fuzzy rule base and the output variables to the plant under control.
- Fuzzy Rule Base − It stores the knowledge about the operation of the process of domain.
- Inference Engine − It acts as a kernel of any FLC. Basically it simulates human decisions by performing approximate reasoning.
- Defuzzifier − Its role is to convert the fuzzy values into crisp values getting from fuzzy inference engine.

7.2 Design of Fuzzy logic controller

Now, you should design Fuzzy logic controllers for your robot to move from its home, go through two gaps and finally reach to the charger position.

You should create a number of Fuzzy logic based robot behaviours for the robot to move from home to the charger, which are shown in YELLOW text below. Note that you may create different numbers of robot behaviours to implement the specified navigation tasks.

```
// The following lines of code is used to control the robot motion
void RobotMove::startMoving(){
    ros::Rate rate(20);   //Define rate for repeatable operations.
    ROS_INFO("Start moving");
    while (ros::ok()){ // keep spinning loop until user presses Ctrl+C
      switch(stage){
      case 1: // the robot is following the wall
          if (PositionY < landmark1)
              // create "Fuzzy-based wall following behaviour"
          else stage = 2;
          break;
      case 2: // the robot turns right into the 1st gap
           if (PositionX < landmark2)
               // create "Fuzzy-based going through the 1st gap behaviour"
           break;
      case 3: // the robot moves toward the 2nd gap
           if (PositionX < landmark3)
              // create "Fuzzy-based moving toward the 2nd gap behaviour"
           else stage = 4;
           break;
     case 4: // the robot going though the 2nd gap
          if (PositionX < landmark4)
              // create "Fuzzy-based going through the 2nd gap behaviour"
           else stage = 5;
           break;
      case 5: // the robot moves towards the charger
           if (PositionX < landmark5)
              // create "Fuzzy-based goal reaching behaviour"
           else stage = 6;
           break;
       case 6: // stop at the charger position
           moveStop();
           break;
       }
      // some existing code
       ros::spinOnce(); // Allow ROS to process incoming messages
       rate.sleep();   // Wait until defined time passes.
  }
  odomTrajFile.close();
```

```
    odomVelFile.close();
    laserFile.close();
    laserMapFile.close();
    ROS_INFO("Closing files");
}
```

7.3 Create Fuzzy based wall following behaviour

Now you should use Fuzzy logic control algorithm to create the wall following behaviour. To follow the wall, you may use two Laser data, i.e. *leftRange* and *mleftRange* as the input of the fuzzy logic controller to generate the turning control value so that the robot can follow the left wall smoothly. As the desired distance between robot and wall is 0.3m for *leftRange* and 0.42 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.

The fuzzy logic controller should keep the robot within the range of "medium".

Based on our driving knowledge, we can create the following table to present some fuzzy rules used for the robot to follow the left wall smoothly,.

leftRange	mleftRange	Robot velocity	Steering speed
Near	Near	Low	Right-Low
Near	Medium	Low	Right-Low
Near	Far	Low	Left-Low
Medium	Near	Middle	Right-Middle
Medium	Medium	Middle	Left-Low
Medium	Far	Middle	Left-Low
Far	Near	Middle	Right-Middle
Far	Medium	Middle	Right-Middle
Far	Far	High	Left-Low

```
class RobotMove{
public:
    // some existing code……
    onstexpr const static double TURN_SPEED_ZERO = 0;
    void Fuzzy_wallFollowing(double moveSpeed, double Data1,double Data2);
        private:
    // some existing code……
    double Max_Fuzzy_output = 0.6;
}

void RobotMove::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;
```

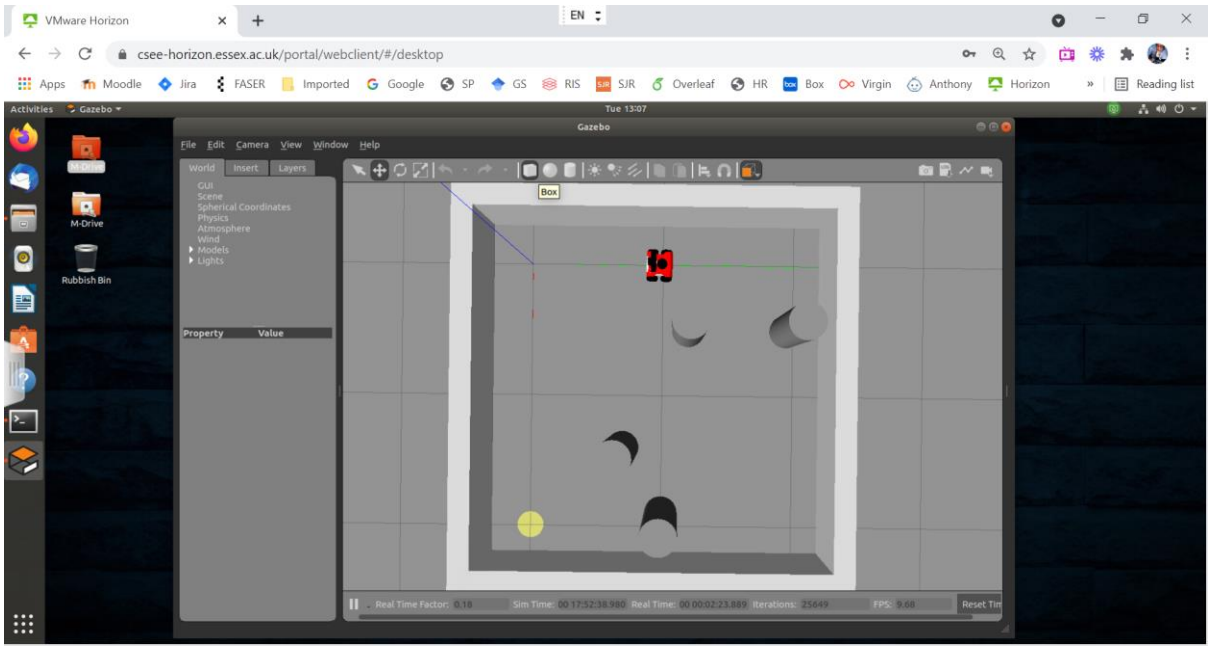
```
        if (laserData2 < 0.4)  fuzzySensor2 = 1; // The robot is near to the wall
        else if (laserData2 < 0.6)      fuzzySensor2 = 2; // The robot at the right distance;
        else fuzzySensor2 = 3;                  // The robot is far from the wall;
        // Fuzzy rule base and control output
        if (fuzzySensor1 == 1 && fuzzySensor2 == 1)
            moveForwardRight(FORWARD_SPEED_LOW, TURN_RIGHT_SPEED_LOW);
        else if(fuzzySensor1 == 1 && fuzzySensor2 == 2)
            moveForwardRight(FORWARD_SPEED_LOW, TURN_RIGHT_SPEED_LOW);
        else if((fuzzySensor1 == 1 && fuzzySensor2 == 3)
            moveForwardRight(FORWARD_SPEED_LOW, TURN_LEFT_SPEED_LOW);
        else if((fuzzySensor1 == 2 && fuzzySensor2 == 1)
            moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_LOW);
        else if((fuzzySensor1 == 2 && fuzzySensor2 == 2)
            moveForwardRight(FORWARD_SPEED_HIGH, TURN_SPEED_ZERO);
        else if((fuzzySensor1 == 2 && fuzzySensor2 == 3)
            moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_LEFT_SPEED_LOW);
        else if((fuzzySensor1 == 3 && fuzzySensor2 == 1)
            moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
        else if((fuzzySensor1 == 3 && fuzzySensor2 == 2)
            moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
        else if((fuzzySensor1 == 3 && fuzzySensor2 == 3)
            moveForwardRight(FORWARD_SPEED_HIGH, TURN_LEFT_SPEED_LOW);
      else ROS_INFO("Following the left wall");
}

void RobotMove::startMoving(){
      // some existing code
      while(ros::pk()){
      switch(stage){
      case 1: // wall following
            if (PositionY < landmark1)
                  Fuzzy_wallFollowing(leftRange, mleftRange);
                 //PID_wallFollowing(FORWARD_SPEED_HIGH, leftRange);
            else stage = 6;
            break;

            // some existing code

      case 6: // stop at the charger position
                moveStop();
                break;
      }

      // some existing code
```

```
        ros::spinOnce(); // Allow ROS to process incoming messages
        rate.sleep();   // Wait until defined time passes.
    }
    odomTrajFile.close();
    odomVelFile.close();
    laserFile.close();
    laserMapFile.close();
    ROS_INFO("Closing files");
}
```

Run your code to test your Fuzzy logic controller:

```
cd  ~/M-Drive/ros_workspace
catkin_make
source devel/setup.sh
roslaunch tutorial_pkg tutorial_rosbot.launch
```

After running your code, the robot will stop at the point you set. The display of Simulator looks like the picture below.



7.4 Create Fuzzy-based going through the 1st Gap behaviour

Now you need to create the go-through the middle of the 1st gap behaviour by using Fuzzy logic control algorithm in a similar way conducted previously. For simplicity, you may use the odometry data to control the robot going through the 1st gap.

During the robot moves toward the 1st gap, *leftRange* and *mleftRange* can be defined below:

- *leftRange*: "near" < 0.3m, 0.3m < "medium" < 0.6m, and "far" > 0.6m.
- *mleftRange*: near" < 0.4m, 0.4m < "medium" < 0.8m, and "far" > 0.8m.

Based on our driving knowledge, we can create the following table to present some fuzzy rules used for the robot to follow the left wall smoothly,.

leftRange	mleftRange	Robot velocity	Steering speed
Near	Near	Low	Right-Low
Near	Medium	Middle	Right-Middle
Near	Far	Middle	Left-Low
Medium	Near	Middle	Right-Middle
Medium	Medium	Middle	Right-Middle
Medium	Far	Middle	Right-Low
Far	Near	Middle	Right-Middle
Far	Medium	Middle	Right-Middle
Far	Far	High	Right-Low

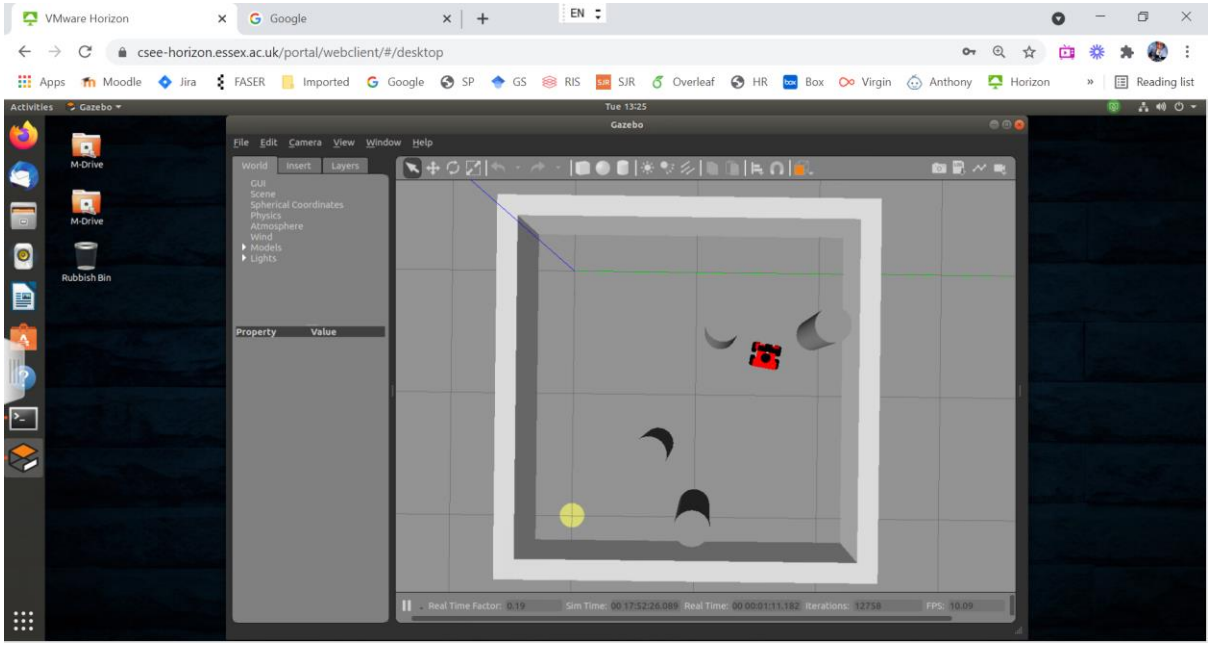
```
class RobotMove{
public:
        // some existing code……
        void Fuzzy_to1stGap(double laserData1, double laserData2);
private:
        // some existing code……
}

void RobotMove::Fuzzy_to1stGap(double laserData1, double laserData2)
{
      int fuzzySensor1, fuzzySensor2;

      // sensor data fuzzification
      if (0 <= laserData1 < 0.3) fuzzySensor1 = 1;
      else if (laserData1 < 0.6)  fuzzySensor1 = 2;
      else fuzzySensor1 = 3;
      if (laserData2 < 0.3)  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);
      else if(fuzzySensor1 == 1 && fuzzySensor2 == 2)
          moveForwardRight(FORWARD_SPEED_LOW, TURN_RIGHT_SPEED_LOW);
    else if(fuzzySensor1 == 1 && fuzzySensor2 == 3)
          moveForwardRight(FORWARD_SPEED_LOW, TURN_LEFT_SPEED_LOW);
    else if(fuzzySensor1 = 2 && fuzzySensor2 == 1)
          moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
    else if(fuzzySensor1 = 2 && fuzzySensor2 == 2)
          moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_MIDDLE);
    else if(fuzzySensor1 = 2 && fuzzySensor2 == 3)
          moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_RIGHT_SPEED_LOW);
    else if(fuzzySensor1 = 3 && fuzzySensor2 == 1)
          moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_LEFT_SPEED_HIGH);
```

```
    else if(fuzzySensor1 = 3 && fuzzySensor2 == 2)
        moveForwardRight(FORWARD_SPEED_MIDDLE, TURN_LEFT_SPEED_MIDDLE);
    else if(fuzzySensor1 = 3 && fuzzySensor2 == 3)
        moveForwardRight(FORWARD_SPEED_HIGH, TURN_RIGHT_SPEED_LOW);
    else ROS_INFO("Going through the 1st gap");
}

void RobotMove::startMoving()
{
   // some existing code
   while(ros::pk()){
       switch(stage){
       case 1: // wall following
           if(PositionY < landmark1)
               Fuzzy_wallFollowing(leftRange, mleftRange);
               //PID_wallFollowing(FORWARD_SPEED_HIGH, leftRange);
           else stage = 2;
               break;
       case 2: // move toward the middle of the 1st gap
           if(PositionX < landmark2)
               Fuzzy_to1stGap(leftRange, mleftRange);
               //PID_to1stGap(FORWARD_SPEED_MIDDLE, robotHeadAngle);
           else stage = 6;
           break;

           // some existing code

           case 6: // stop at the charger position
               moveStop();
               break;
       }
       // some existing code
       ros::spinOnce(); // Allow ROS to process incoming messages
       rate.sleep();   // Wait until defined time passes.
   }
   odomTrajFile.close();
   odomVelFile.close();
   laserFile.close();
   laserMapFile.close();
   ROS_INFO("Closing files");
}
```

Run your code to test your Fuzzy logic controller and adjust its parameters to improve the wall following performance until the smooth trajectory is achieved.

```
cd  ~/M-Drive/ros_workspace
catkin_make
source devel/setup.sh
roslaunch tutorial_pkg tutorial_rosbot.launch
```

After running your code, the robot should be stopped at the point you set, as shown below.



7.5 Create remaining two robot behaviours

Now, you should continue the creation of the remaining robot behaviours so that the robot can be controlled to move toward the 2nd gap, go through it, and reach the charger position in a similar way conducted in the previous steps. You should adjust Fuzzy rules to improve the robot performance.

After all Fuzzy-based robot behaviours are created, you should view some saved odometry and laser data and create the velocity, trajectory, heading and map graphs which will be included into your Assignment report.