# Robust Control Lab Report

Ben Frazer*

March 25, 2022

## 1 Introduction

This report will aim to describe the controller implementation of a fuzzy logic way-point following collision avoiding robot.

## 2 Implementation

This Section will discuss the approach taken to solve the control problem.

### 2.1 Collision Avoidance

#### 2.1.1 Sensor Characteristics

The collision avoidance mechanism of the robot is fed entirely by the input of the right and left forward facing sensors (abbreviated to RS and LS respectively). These sensor report the linear distance to the nearest obstruction directly in-front of them, however saturate above a distance of `1m`. Thus as the robot approaches a wall the respective sensors will report a value starting at `1` and shrinking to `0` at the point at which the robot hits the wall. Since the sensors are placed to the right and left of the robot's body, if the robot approaches a wall with an angle it is thus expected that the sensors will report a slight difference in the distances accordingly. This difference in the distance will be exploited for collision avoidance.

*2704250F@student.gla.ac.uk

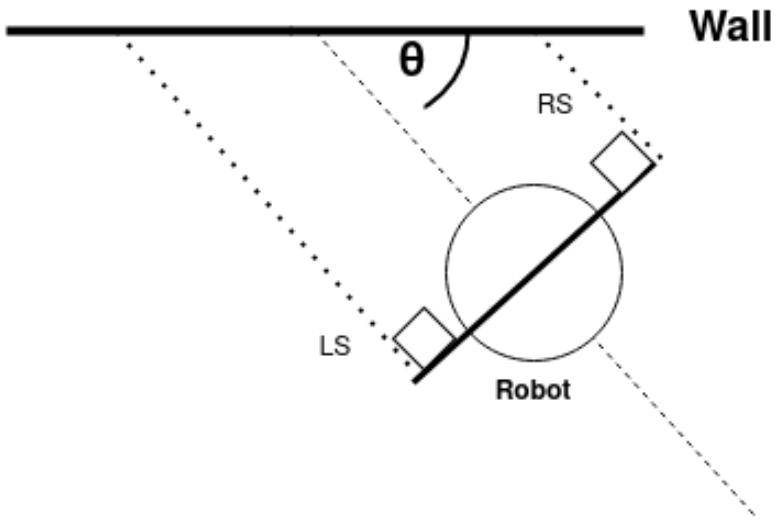



**Figure 2.1:** *Diagram of the robot approaching an obstacle, defining approach angle.*

**Table 2.1:** *The Expected Sensor output for various approach angles towards an obstacle, Approach angle $\theta$ is defined in Figure 2.1.*

| Approach Angle $\theta$ | expected Sensor Report |
|---|---|
| $\theta<\pi/2$ | LS<RS |
| $\theta>\pi/2$ | LS>RS |
| $\theta=\pi/2$ | LS=RS |

### 2.1.2 Desired behaviour

It is obvious that the desired behaviour when approaching an obstacle at an angle is to turn in such a way that the obstacle is no-longer directly in front. It is intuitive to see that the optimal way to turn is always in such a way as to minimise the required angle change. For example the approach angle shown in Figure 2.1 would elicit an intuitive response to turn further left until driving parallel along the wall. The approach angles, their respective sensor readouts and the desired turn can be formulated in Table 2.2.

**Table 2.2:** *Desired Behaviour For a given approach angle*

| Approach Angle $\theta$ | expected Sensor Report | Desired Turn Direction |
|---|---|---|
| $\theta<\pi/2$ | LS<RS | Left |
| $\theta>\pi/2$ | LS>RS | Right |
| $\theta=\pi/2$ | LS=RS | Left |

This table highlights one of the key considerations when designing the collision avoidance system, namely that the frontal approach angel must also elicit a turn in some direction. It is thus decided that the control system will have a tendance to turn in the left direction in the case of a frontal approach.

**Turn 'aggressiveness'** Given the tank staring nature of the robot, the turn rate may be adjusted from turning on the spot (one wheel forward one back) to turning with a large radius (one wheel stationary one wheel forward). For simplicity these turns will be known as "hard turn" and "slow turn" respectively.

Through experimentation is established that the turning radius for one wheel forward one stationary is roughly `0.8m`, while of course the turning circle of one wheel forward one back is $\approx$ `0`. This means that an ideal turn would begin upon detecting a sensor reading less than `0.8m`. If the robot is however within this radius of the wall, and pointing directly at it then it is impossible that this larger turning circle won't intersect the wall. It is therefore required that the robot be able to rotate on the spot when very close to the wall. The desired behaviour may be formulated in the Table below:

| Absolute Distance | Turn aggressiveness |
|---|---|
| d>0.8 | No turn |
| 0.2<d<0.8 | Slow Turn |
| d<0.2 | Hard Turn |

**Amplification of Small differences in sensor measurements**

### 2.1.3 Collision Sensor Membership Functions

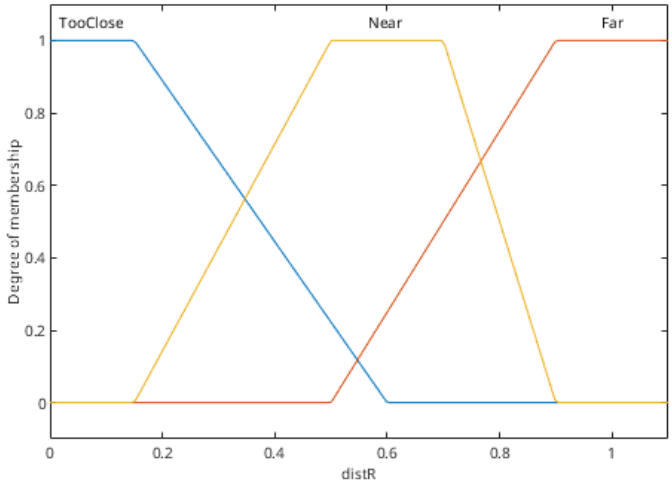



**Figure 2.2:** *Membership Function for a single forward facing sensor (Bias term=0)*

### 2.1.4 Bias Term Implementation

To gain the desired characteristic discussed in Section 2.1.2, it decided that a 'bias' term would be introduced in order to change the symmetry of the sensor feedback membership functions. Since the avoidance behaviour is tuned to respond to differences in sensor readings, this will have the effect, that a front on approach will appear to the controller like an approach of a slight angle and this will induce a turn. The nature of this asymmetry between membership functions is simply a shift along the x axis of $\Delta x = bias$.

The bias term is applied to the left sensor membership function only and may be negative or positive, resulting in a tendance to turn right or left depending on the sign.

**Table 2.3:** *Effect of varying the value of the bias term assuming a frontal approach.*

| bias value | Turn Tendency |
|---|---|
| bias=0 | No tendance |
| bias>0 | Turn Left |
| bias<0 | Turn Right |

### 2.1.5 Membership Function Definition

```
% Left sensor membership Function (with Bias term)
colisCont = addMF(colisCont,"distL","trapmf", [-10 -10 0.15
 ↪  0.6+bias],'Name',"TooClose");
colisCont = addMF(colisCont,"distL","trapmf", [0.5+bias 0.9 1.1
 ↪  2],'Name',"Far");
colisCont = addMF(colisCont,"distL","trapmf", [0.15+bias 0.5+bias
 ↪  0.7+bias 0.9],'Name',"Near");
% Right sensor membership Function
colisCont = addMF(colisCont,"distR","trapmf", [-10 -10 0.15
 ↪  0.6],'Name',"TooClose");
colisCont = addMF(colisCont,"distR","trapmf", [0.5 0.9 1.1
 ↪  2],'Name',"Far");
colisCont = addMF(colisCont,"distR","trapmf", [0.15 0.5 0.7
 ↪  0.9],'Name',"Near");
```

### 2.1.6 Collision Avoidance Rules

As will be discussed in Section **??**, the collision avoidance rules will operate in a manner completely oblivious to the target seeking steering, as such none of the rules defined below make reference to the angle to target measurement $\phi$.

```
%Turn Right Slow
colisCont = addRule(colisCont,"distL==Near & distR==Far => powerL=Fwd,
 ↪  powerR=Off (1)");
%Turn Left Slow
colisCont = addRule(colisCont,"distL==Far & distR==Near => powerL=Off,
 ↪  powerR=Fwd (1)");
%Turn Left Hard
colisCont = addRule(colisCont,"distL==Near & distR==TooClose =>
 ↪  powerL=Rev, powerR=Fwd (1)");
%Turn Right Hard
colisCont = addRule(colisCont,"distL==TooClose & distR==Near =>
 ↪  powerL=Fwd, powerR=Rev (1)");
```

### 2.1.7 Bias Term Implementation

## 2.2 Way-point Seeking

The way-point seeking controller is also implemented using fuzzy logic. This

### 2.2.1 Input

The input for the way-point seeking algorithm is a derived metric referred to in this report as 'relative heading angle' and denoted by $\phi$. This angle refers to the angle between the absolute heading of the robot $\theta$ and the heading angle from the robot to the target (output by the function `computeHeadingAngle()`).

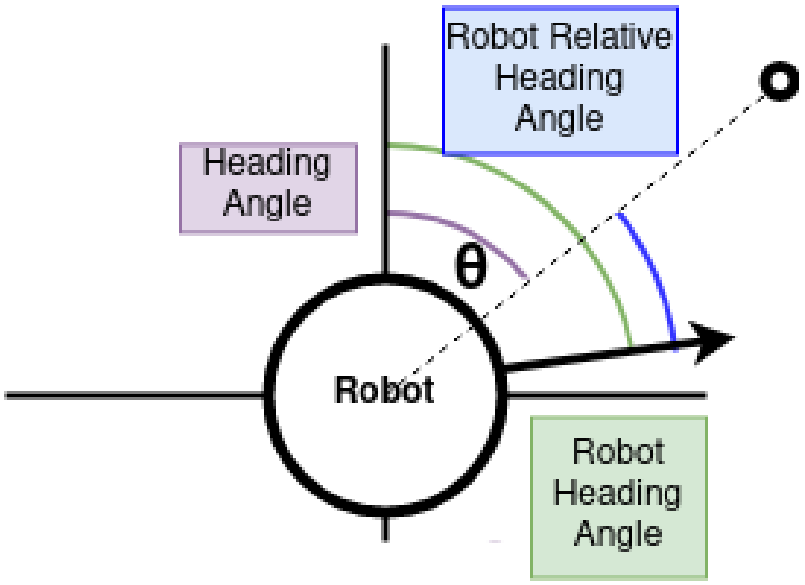



**Figure 2.3:** *Definition of robot relative heading angle $\phi$*

The angle $\phi$ may be defined according to 2.1:

$$\phi = RobotHeading - \theta \tag{2.1}$$

The code for this is shown below:

```
1 Theta = wrapToPi((newHeadingAngle)-
  ↪  (state(timeStep,stateEnum.angHeading)));
```

The `wrapToPi()` function is used so that the angles returned are always within $\pm\ \pi$. This avoids the robot sometimes rotating fully when approaching a target from above.

### 2.2.2 Desired behaviour

When $\phi$ is negative this means the target is to the left of the robot, then when $\phi$ is positive, this means the target is to the right of the robot. The action in either case is fairly intuitive: when the target is to the right, turn right and when the targe is to the left, turn left. These characteristic rules are shown in

**Table 2.4:** *Desired behaviour for Way-point Seeking algorithm with different values of $\phi$*

| $\phi$ | Turn Direction |
|---|---|
| $\phi$=0 | No turn (forward motion) |
| $\phi$>0 | Turn Right |
| $\phi$<0 | Turn Left |

**Dead-zone** To stop the robot constantly oscillating between turning left and right however, a dead-zone is added in the middle which where acceptably low angles will still elicit forward movement.

### 2.2.3 Membership Functions

To get the desired behaviour in Table 2.4, three separate membership functions for the input $\phi$ must exist:

- *TargetRight*
- *TargetLeft*
- *TargetCenter*

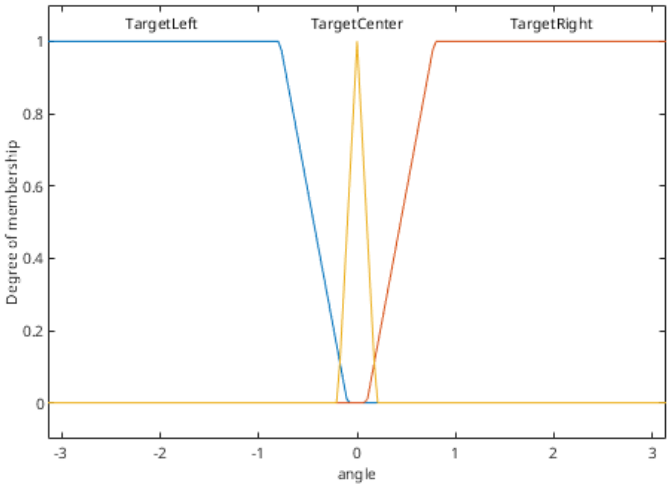



**Figure 2.4:** *Membership Function for relative angle $\phi$ input, Angle given in Radians.*

### 2.2.4 Collision avoidance prioritisation

The rules to elicit the current turning behaviour by themselves are relatively simple, however it is necessary to consider the condition in which an obstacle is present, which is in-between the robot and the target. In this case the way-point seeking behaviour would be steering the robot towards the wall, while the collision avoidance algorithm would be steering the robot away. The intuitive solution to this problem would be of course to prioritize the collision avoidance steering. To do this the rules defined for the collision avoidance are contingent on both left and right sensor inputs being in the far membership function. Thus when the robot is in open terrain the Waypoint seeking algorithm behaves nominally, however close to an obstacle it will lose it's authority.

### 2.2.5 Turn Aggressiveness

Since a condition of the robot reaching a way-points is that it get within a tolerance of `0.05m`, the turning circle guiding the robot must allow for the robot to reach the target to within this tolerance. The robot

is not smart enough to first back away from the target and then approach again, rather it will simply, circle the target eternally at the specified turn rate. As such the turn rate must be smaller than the target tolerance meaning that only hard turns are suitible.

### 2.2.6 Way-point seeking rules

The following Source code is used to define the way-point seeking rules:

```
% Turning (Contingent on Sensor distances being "far")
colisCont = addRule(colisCont,"distL==Far & distR==Far &
    ↪  angle==TargetRight => powerL=Fwd, powerR=Rev (1)");
colisCont = addRule(colisCont,"distL==Far & distR==Far &
    ↪  angle==TargetLeft => powerL=Rev, powerR=Fwd (1)");

% Forward motion in "deadzone" (Contingent on Sensor distances being
    ↪  "far")
colisCont = addRule(colisCont,"distL==Far & distR==Far &
    ↪  angle==TargetCenter=> powerL=Fwd, powerR=Fwd (1)");
```

# 3 Test Plan

## 3.1 Collision avoidance tests

Based on the considerations of Sections 2.1 and 2.2, the controller collision avoidance behaviour will be tested. The values for LS and RS corresponding to these conditions are defined and will then be passed into the controller. The resulting output power for each wheel is then tabulated in Table 3.1.

**Table 3.1:** *Tabulated test conditions and resulting output for collision avoidance behaviour*

| Comment [1] | LS | RS | P_LW | P_RW | Expected Output | Actual Output |
|---|---|---|---|---|---|---|
| Preferentially turn left during frontal approach | 0.7 | 0.7 | -0.77 | 3.08 | turn left | turn left |
| Turn Hard Left When $\theta$<pi/2 and total distance is small | 0.2 | 0.3 | -4.27 | 3.69 | turn hard left | turn hard left |
| Turn Slow Left When $\theta$<pi/2 and total distance is medium | 0.6 | 0.8 | 1.13 | 2.01 | turn slow left | turn slow left |
| Turn Hard Right When $\theta$>pi/2 and total distance is small | 0.3 | 0.2 | 1.72 | -3.30 | turn hard right | turn hard right |
| Turn Slow Right When $\theta$>pi/2 and total distance is medium | 0.8 | 0.6 | 4.79 | -0.37 | turn slow right | turn slow right |

## 3.2 Way-point Steering Test

**Table 3.2:** *Tabulated test conditions and resulting output for Waypoint steering behaviour*

| Comment | LS | RS | $\phi$ | P_RW | P_LW | Expected Output | Actual Output |
|---|---|---|---|---|---|---|---|
| Turn Right when $\phi$ is positive + no Obstacle | 1 | 1 | $\pi/8$ | 8.0391 | -5.5654 | turn hard right | turn hard right |
| Turn Left when $\phi$ is negative + no Obstacle | 1 | 1 | $-\pi/8$ | -5.5654 | 8.0391 | turn hard left | turn hard left |

# 4 Results

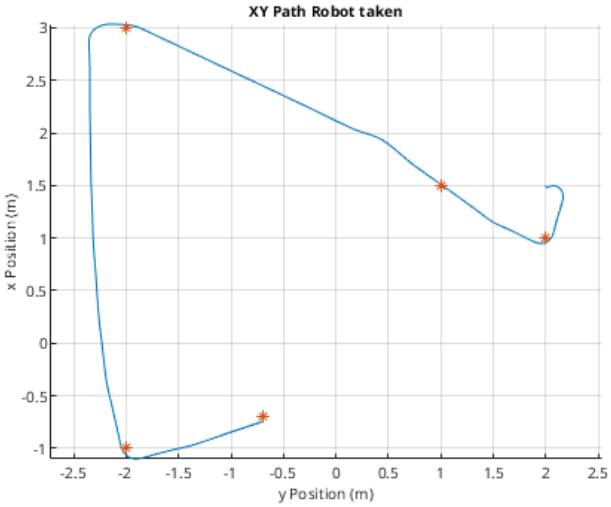
## 4.1 Final Path No Walls

Figure **??** shows the path of the robot when seeking the way-points shown in Table 4.1.

[1] $\theta$ is defined as the approach angle to a wall

**Table 4.1:** *Waypoints followed by robot*

| Point | X | Y |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 1.5 | 1 |
| 3 | 3 | -2 |
| 4 | -1 | -2 |
| 5 | -0.7 | -0/7 |




## 4.2 Final Path With Walls

Figure 4.1 shows the path of the robot with obstacles present following the way-points shown in Table 4.2.

**Table 4.2:** *Waypoints followed by robot in test with obstacles.*

| Point | X | Y |
|---|---|---|
| Start | -1 | 0 |
| 123 | 2 | 3 |
| 214 | 1 | 4 |
| 3 | 3 | -4 |
| 4 | -1 | -2 |
| 5 | 3 | 0 |
| End | 1.5 | 2 |

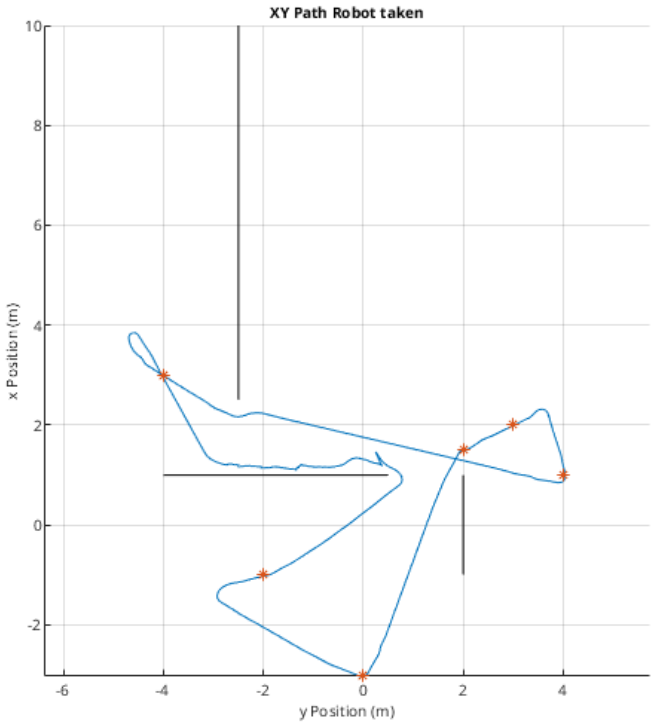



**Figure 4.1:** *XY Path of Waypoint Seeking robot with obstacles present*

# 5 Discussion

## 5.1 'Wall Scrape' With obstacles

One particularly difficult scenario for the robot to deal with is when there is a wall in-between itself and the target, at this point the way-point steering starts to fight with the collision avoidance steering since the former wishes to steer through the wall, while the latter wishes to push away from the wall. Since, by design, the way-point guidance loses it's authority when close to the wall, this results in an angular 'wobbling' as the obstacle avoidance algorithm turns away from the wall and target until the sensor readings are suitable low for the way-point guidance to kick in again and steer back towards the wall. Since the Way-point steering turns with one wheel forward and one wheel back, this means that as soon as it kicks in the entire forward motion of the Robot is arrested, meaning that motion along such a wall results not only the angular oscillation, but also on a jerky forward motion.

The angular oscillation can be seen in Figure 5.1 between t=25-45s [2].

[2]Note that discontinuities in this plot are due to the angle wrapping from $\pm\ \pi$

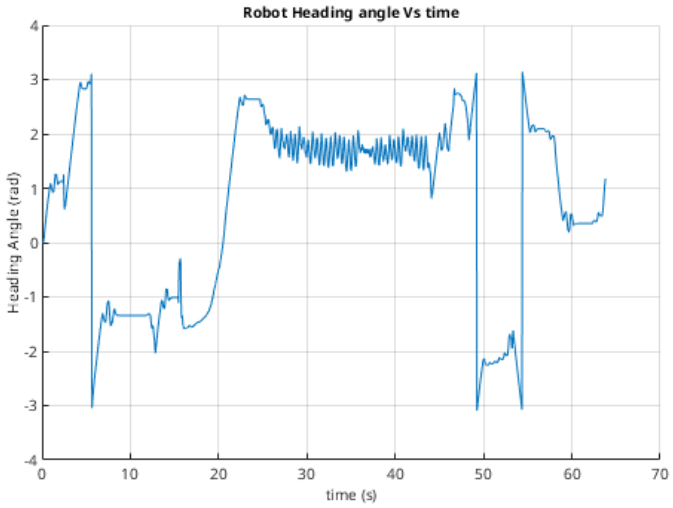



**Figure 5.1:** *The angular oscillating Resulting from Wall Scrape*

# 6 Improvements

## 6.1 Acting on $\Delta$ Distance

The desired behaviour of the collision avoidance algorithm is effectively to maximise the difference between the two sensors, since this is the effect that turning has. Perhaps a better suited input for the controller is the difference between the right and left sensor rather than the absolute value of both. This might allow for a more consice set of rules, since it boils down two separate inputs with their own rule sets into just one.

## 6.2 Acting on absolute distance to target

One mitigation for the 'wall scrape' described in Section 5.1 might be to have different turn aggressiveness depending on the relative distance to the target. As discussed in Section 2.2.5, the tank steering is required to avoid the case in which the robot is close enough to the target that it's turning doing a slow turn would not allow it to intersect the target. This condition only exists however if the target is within the robot's turning circle, so if the distance to the targe could also be used as an input, this could be used as a switch between hard turning and slow turning. Since slow turning just involves stopping/Slowing one wheel relative to the other rather than driving it backwards, this might act to reduce the 'jerkiness' of the motion along a wall.

# 7 Appendices

**ENG5009 Advanced Control 5**

**Laboratory 1 Worksheet Answer Grid**

## Fuzzy Logic

1. *Introduction to Fuzzy Logic Toolbox*

| Defuzzification Method | Service | Food | Tipping Value |
|---|---|---|---|
| Centroid | 5 | 5 | 15 |
| Centroid | 7 | 8 | 20.341 |
| Centroid | 10 | 2 | 16.421 |
| Mean of Maximum | 5 | 5 | 15 |
| Mean of Maximum | 7 | 8 | 24.9 |
| Mean of Maximum | 10 | 2 | 24.9 |
| Bisector | 5 | 5 | 15 |
| Bisector | 7 | 8 | 21.6 |
| Bisector | 10 | 2 | 22.5 |

2. *Integrating the Fuzzy system into the command line*

Defuzzification method used: Centroid

| Service | Food | Value |
|---|---|---|
| 5 | 5 | 15 |
| 7 | 8 | 20.341 |
| 10 | 2 | 16.421 |

3. *Further Fuzzy Logic Example*

| Input Temperature, °C | Output |
|---|---|
| 9 | 15.035 |
| 13.1 | 11.222 |
| 26.5 | -9.5957 |
| 20 | -9.4369e-17 |

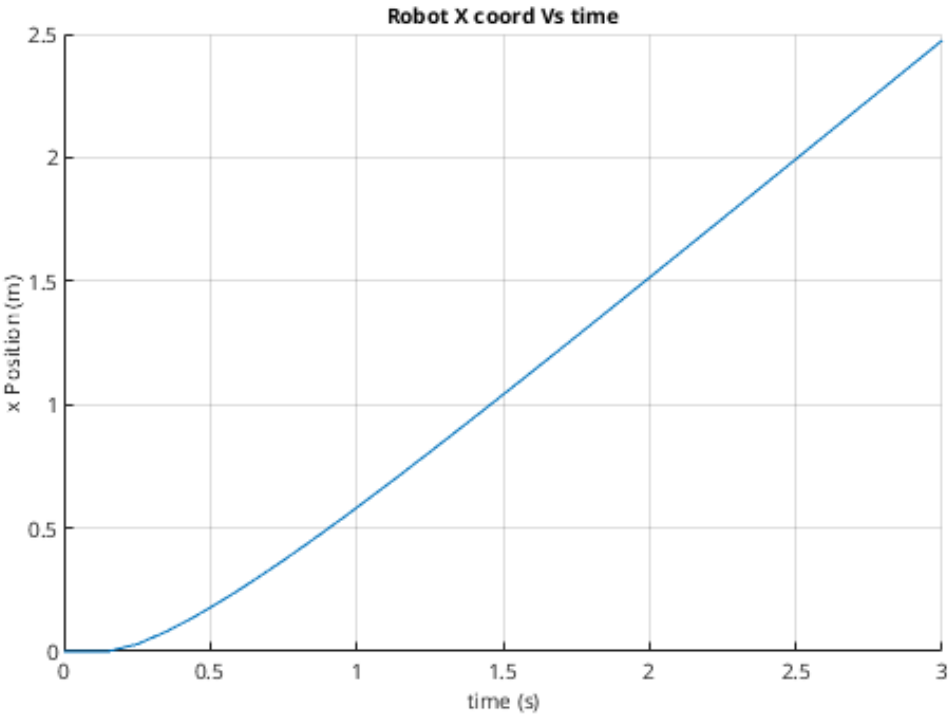
**Lab 2 - Introduction to Model – Answer Grid 2**

The following should be used as a structure for presenting the requested information. This structure can then be copied into an Appendix of the main assignment report.

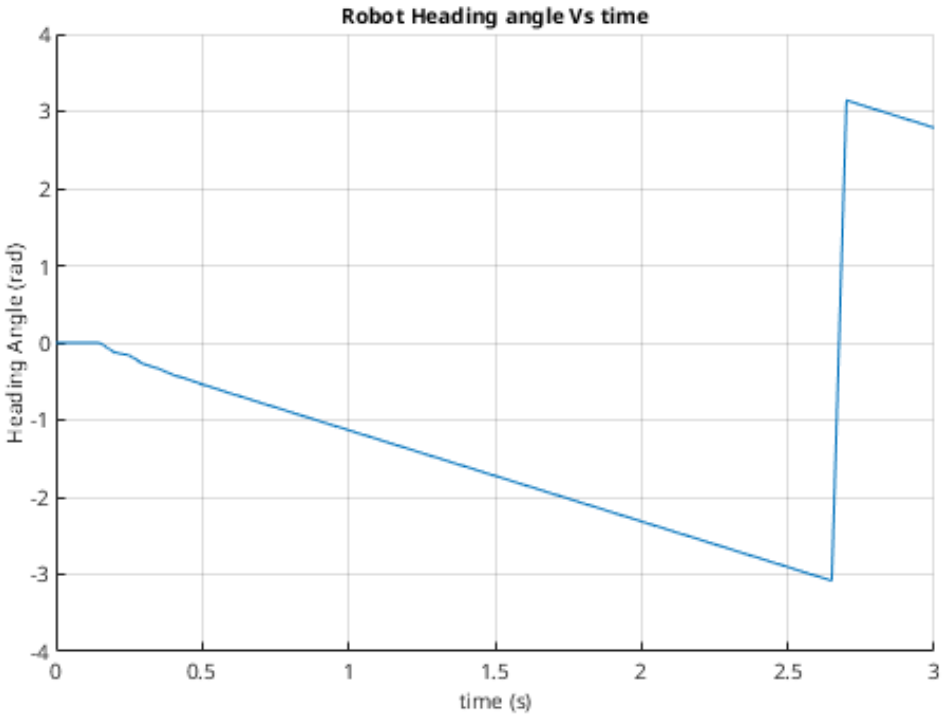
1. *Basic Operation*

The following data shows that you have got the simulation working as expected.

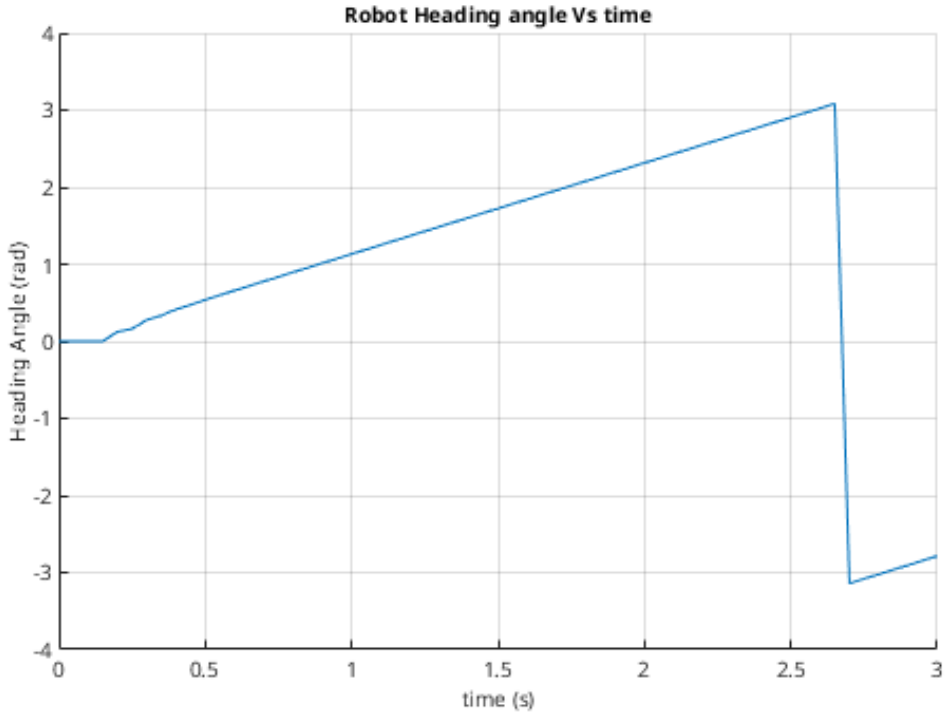
- *Insert a plot for each of the following:*
  - *Straight Line motion*




    - *Insert a plot of the x distance travelled.*
  - *Turn counterclockwise*
    - *Insert a plot of the psi angle.*

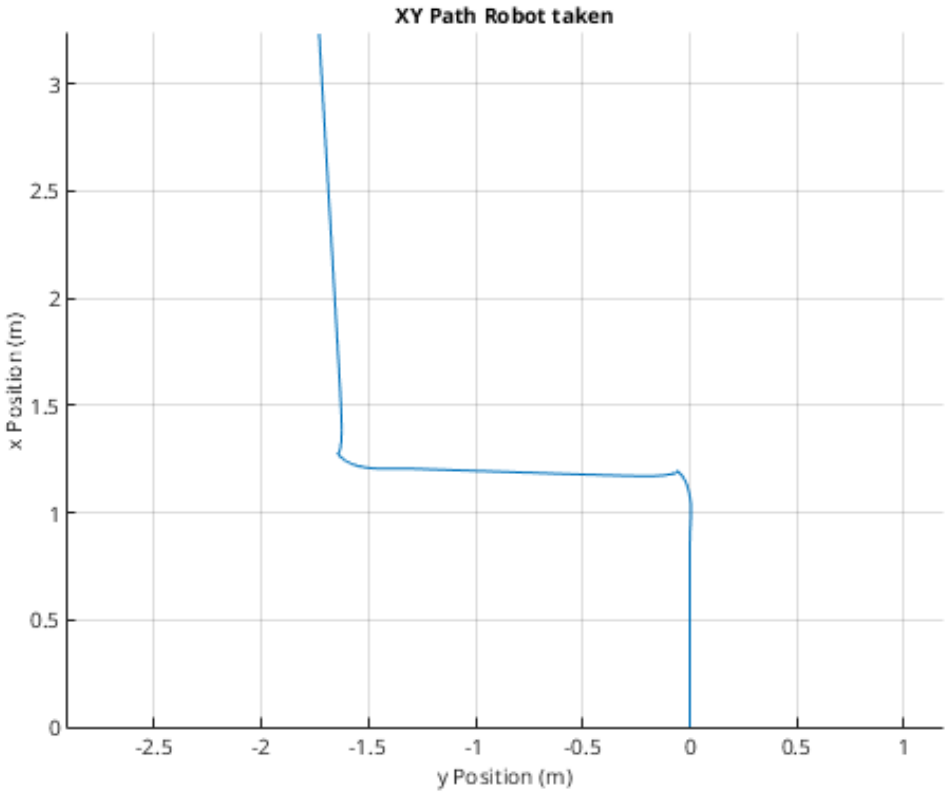



- *Turn Clockwise*
  - *Insert a plot of the psi angle.*




- *Complete a short forward, turn left, forward, turn right*

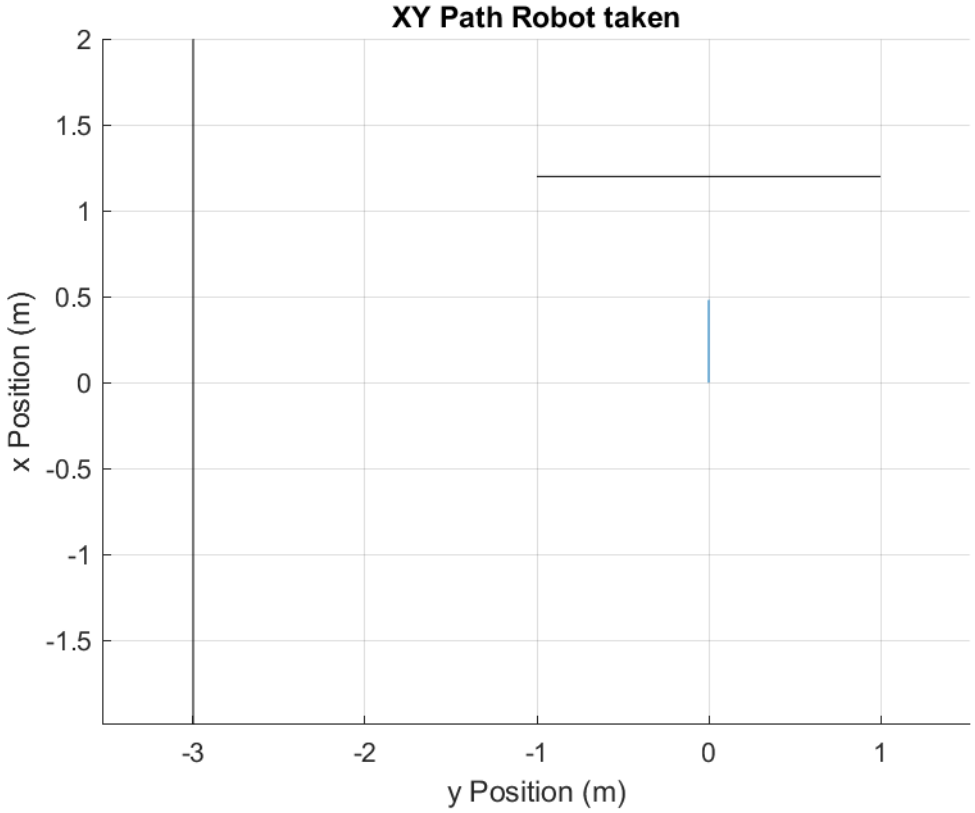
- *Insert a plot of the x/y position*




2. *Running the Model with a Fuzzy Controller*

The following data shows that you have got a basic Fuzzy controller working with the model.

- *Provide a plot of the path of the system using the tutorial example.*




- *Comment on the behaviour of the system*

There were lots of oscillation in the original controller design it would just oscillated in a strait line when it hit a wall from directly front on.

- *Comment on any changes you would make to improve the performance of the system*

To compensate for this osculation I first added a bias term to the left motor membership function as shown below:

*colisCont = addMF(colisCont,"distL","trapmf", [-10 -10 0.25+bias 0.45+bias],'Name',"TooClose");*

*colisCont = addMF(colisCont,"distL","trapmf", [0.2+bias 0.4+bias 0.5+bias 0.7+bias],'Name',"Close");*

*colisCont = addMF(colisCont,"distL","trapmf", [0.5+bias 0.9+bias 1.1 2],'Name',"Near");*

With the bias set to 0.1, this had the effect of the robot preferentially turning left, but the oscillations still remained.

I first tried to remove the oscillation terms by "smoothing out" the transitions between membership functions so they weren't to steep, this did very little to change the behaviour however.

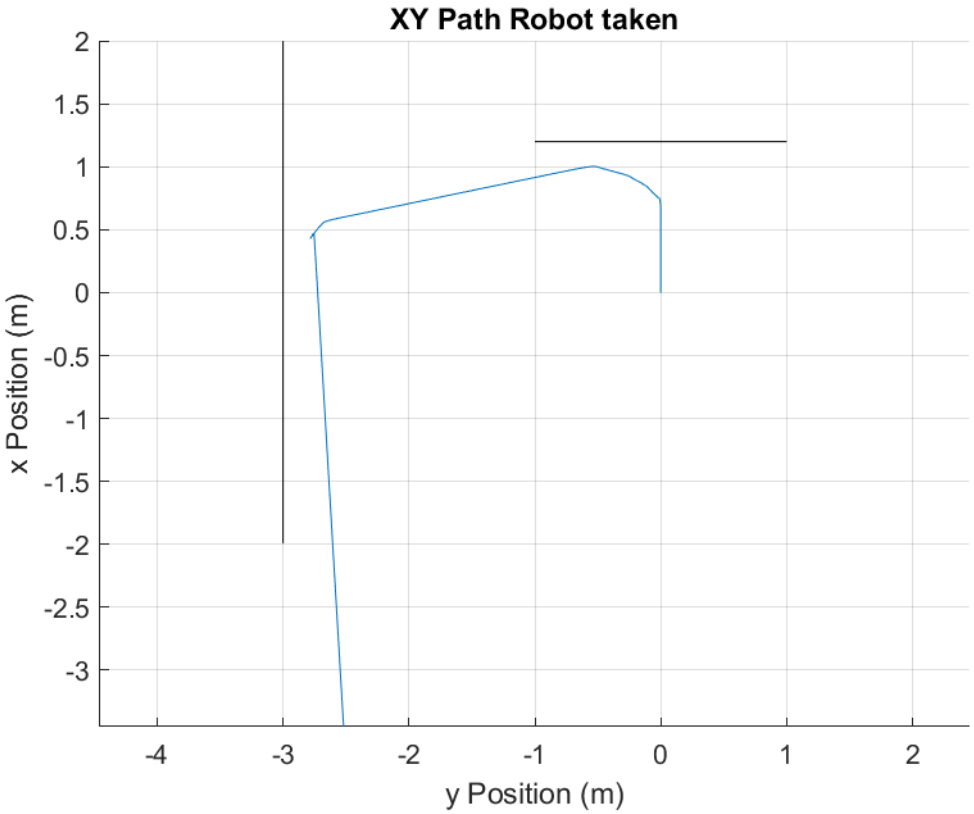
I was suspicious because it seemed that the robot seemed to drive strait until it hit a wall and then immediately started reversing, while I expected it to begin smoothly turning due the bias terms. I dug a little into the rules and found the following suspicious rule:

*colisCont = addRule(colisCont,"distL==Close & distR==Close => powerL=Rev, powerR=Rev (1)");*

After commenting this out the robot began working without any oscitation as can be seen below:

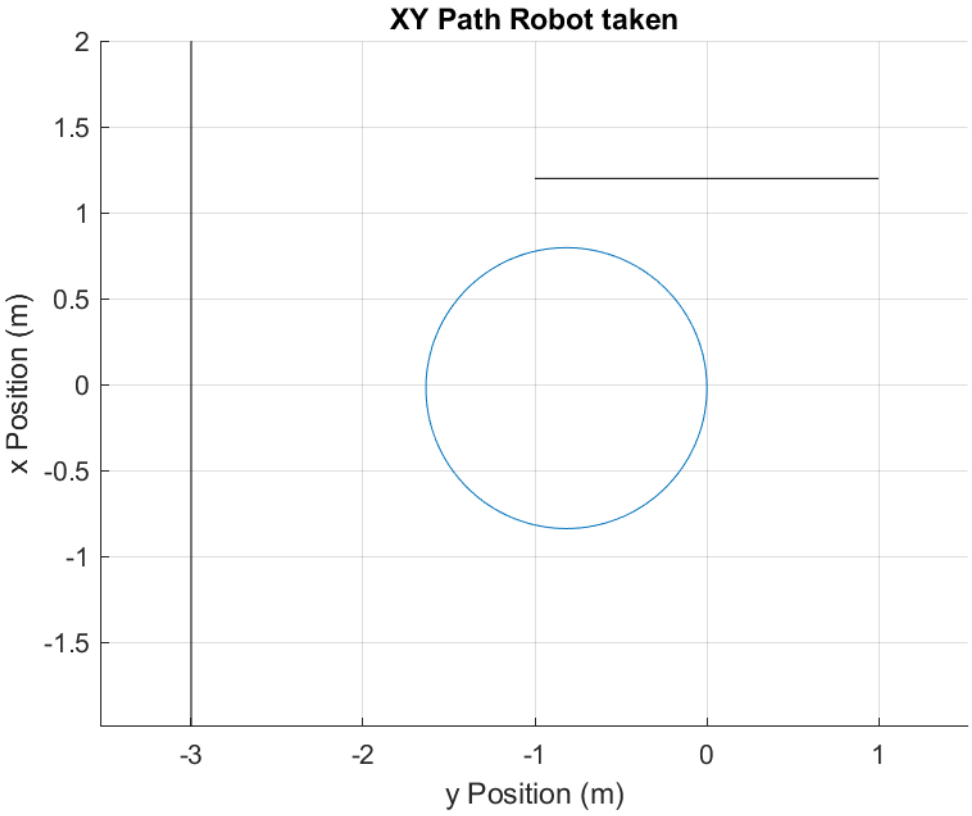
- *Provide a plot of the path of the system with the changes you have implemented*

3.




The following data shows that you have got a basic Neural Network controller working with the model.

- *Provide a plot of the path of the system using the tutorial example.*




- *Briefly comment on the behaviour of the system*

Using the base parameters and the neurons threshold activation function, at max sensor distance (1) the robot still has one wheel going forward and one stationary meaning it turns in a circle.

- *Comment on any changes you would make to improve the performance of the system*

Though not specified, the  activation function for the neural net is assumed to be a simple threshold meaning that the output of the net may only be a one or a zero. this means that any motor may only be in one of two states. these could be:

- forward or off

- forward and backward

*It is possible to implement a controller for either configuration, however the first option is substantially more challenging this is what i cose to implement and describe here.*

*** Design of optimal Gains and Weights

It is worth noting that the limited number of neurons in the network impose fundamental geometric limitations on the logical rules that may be implement. If one considers the inputs to the robot to be a two dimensional state-space, it can be shown that any combination two

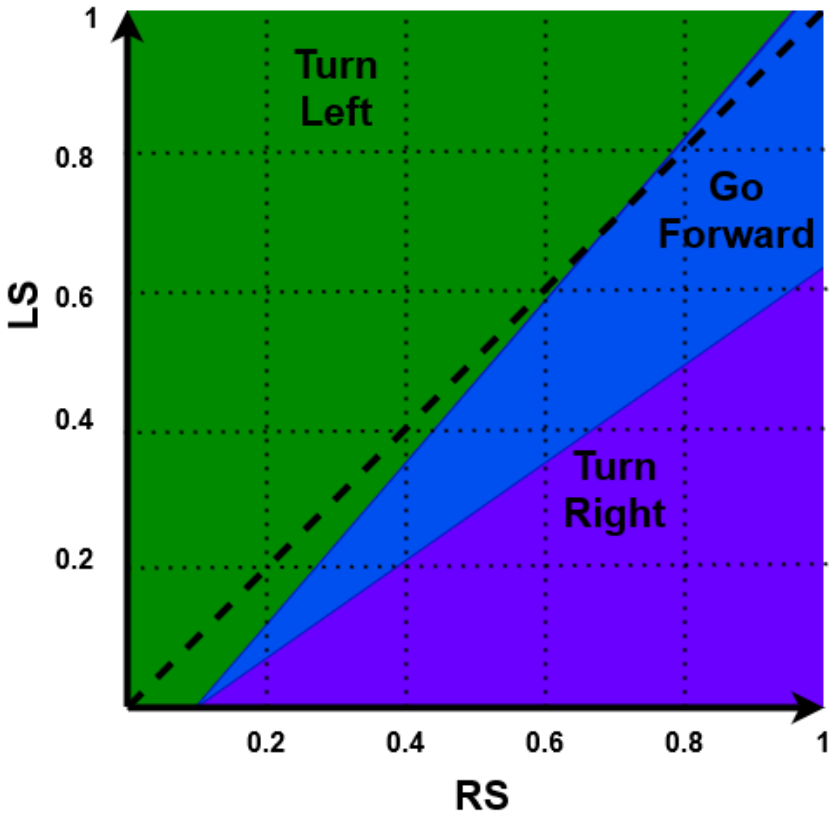
weights and biases must constitute a strait line bisecting the state-space. Depending on the output "wiring" of the neural net to the motors the area enclosed by one of these lines will mean either full forward or reverse/stop for the output of a given wheel as a function of the input states. When the lines for each wheel are drawn on the state-space the intersection of the areas where the wheels are moving forward will constitute forward motion for the robot as a whole, while the areas were both are zero will be either stationary or in full reverse depending on what the output of logical zero from the neural net maps to.

A basic fundamental requirement of the robot be that it should go in a strait line when there are no obstacles in front of it. Since the sensor saturate at distance >1. At minimum, the only thing that must be satisfied in the state-space is that the infinitesimal region around Rs=1 and LS=1 be an intersection of the two regions corresponding to forward motion of each wheel.

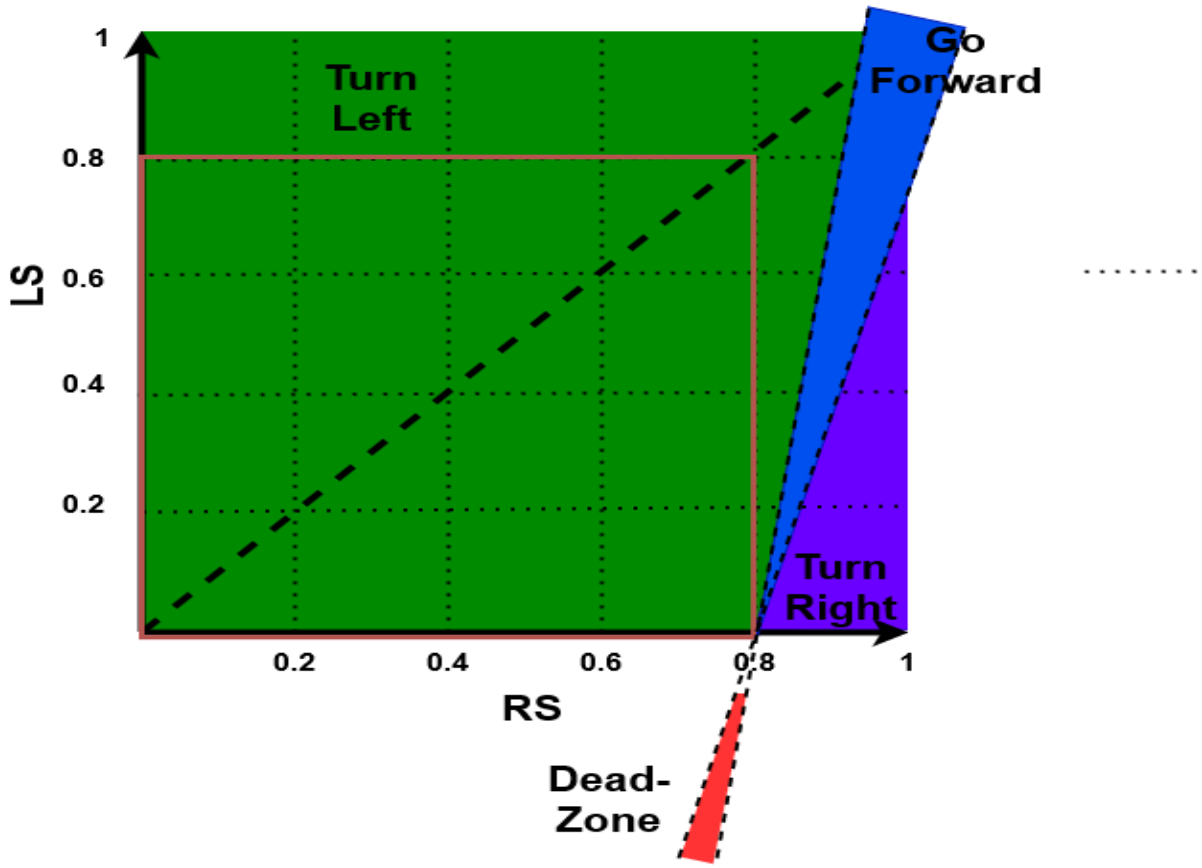
The second constraint relates to the turning circle of the robot, from testing, it was observed that the turning circle with one wheel stationary, is ~0.8, this means that the robot must begin turning almost as soon as the object is in sensory range. This imposes a second geometric constraint, that turning begin out-with the square drawn from the origin to the point LS=0.8,RS=0.8. Stated another way 'going forward' region where LW=1&RW=1 (shown in blue) must not cross into the region, LS=<0.8|RS=<0.8 (in the figure below it does).

A final consideration that in the case of the two selected wheel states being forward and stationary, there exists the possibility for "dead-zones" to exist where both wheels are zero. Once the robot crosses into such a region it is impossible to get out. This is another example where the choice to go with forward and backward wheel activation is superior! One can show that such a region is an inevitable outcome of any overlap in the two regions, just from basic geometry. Restated this means that if we want the robot to move forward, then by definition we must accept a dead-zone. We do however have the freedom to chose where this dead-zone appears. In the figure below the 'go forward' region has been chosen such that the dead-zone is on the negative axis, values which we know the sensor will never take.

0.4




With these geometric constraints in place, and with the understanding that the RW=1,LW=1 region may only be defined by strait lined due to the architecture of the network, the following geometry is proposed:




From the equations of the lines bounding the two regions, it is trivial to derive the corresponding weights and thresholds.

- *Briefly provide a plot of the path of the system with the changes you have implemented*