

6ELE0067 Intelligent Systems and Robotics

Coursework Assignment - Fuzzy Logic Control of a Mobile Robot

1. FIS Design

1.1 Overall structure

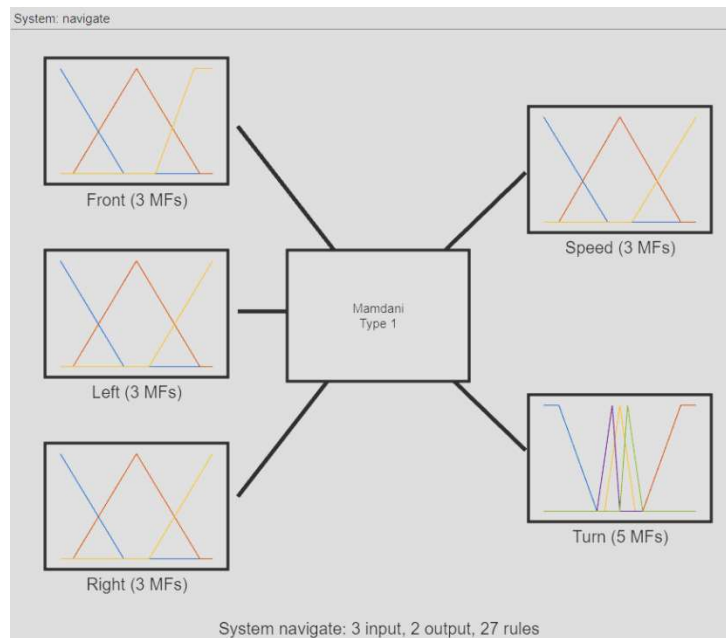


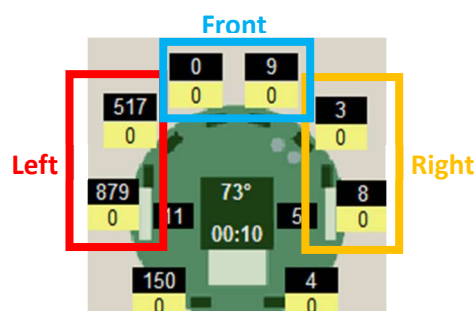
Figure 1: Fuzzy Inference System plot.

For this fuzzy inference system, I have chosen a Mamdani type 1. The main advantage of Mamdani over Sugeno is the ability to output a fuzzy set, which makes the rules more human-like, so the system is intuitive and easier to build from human experience. Additionally, Mamdani is a well-known inference system. However, unlike a Sugeno system, it is slower to compute and can be too complex when working with embedded systems such as drones, where computing power is often limited.

The type-2 inference system was not implemented as it models uncertainty within the degree of membership, which is unnecessary for this simulation. As this is a simulation, there is no need to attempt to reduce noise or measure errors.

In a Mamdani Type 1 FIS, the initial step is fuzzification, whereby input values are converted to a degree of membership for each membership function, resulting in nine values to use in our rules. The rules are then applied using an AND operator, which is equivalent to using a MIN operator on all the fuzzified values. After computing all the rules, the corresponding membership function is selected for each output and used to generate a fuzzy output set. These sets are then combined, resulting in two fuzzy output sets. The centroid of each set is computed to obtain a single value per output.

1.2 Input variables



To cover the front, left, and right sides without overwhelming the FIS with too many inputs, I chose to group the sensors in pairs. This approach also enables a wider area to be covered instead of just a single value.

$$\begin{aligned} Front &= \frac{\text{sum}(\text{reflex}(3:4))}{2}; \\ Left &= \frac{\text{sum}(\text{reflex}(1:2))}{2}; \\ Right &= \frac{\text{sum}(\text{reflex}(5:6))}{2}; \end{aligned}$$

To input group values into the FIS, the values are merged using a mean function. No scaling is applied, so the universe of discourse for each input is equal to the sensor range, which is from 0 to 1024. Here, 1024 represents the closest distance and 0 represents the farthest or no detection.

1.3 Output variables

To define the Universe of Discourse for the two output variables, 'speed' and 'turn', I manually tested the edge values to ensure that the speed is not too high or too low at its maximum. This step of the process involves testing to define proper membership functions. Therefore, the speed should be between 0 and 100, as going backward is not useful in our case. Additionally, the turn rate should be between -10 and 10.

1.4 Membership functions (MFs) of the input and output variables

Fuzzy set	MF's name	Type	Parameters
Front	Far	Linear Z-shaped	[0 427]
	Medium	Triangular	[86 512 938]
	Close	Linear S-shaped	[640 900]
Left / Right	Far	Linear Z-shaped	[0 427]
	Medium	Triangular	[86 512 938]
	Close	Linear S-shaped	[597 1024]
Speed	Slow	Linear Z-shaped	[0 42]
	Medium	Triangular	[10 50 90]
	Fast	Linear S-shaped	[58 100]
Turn	Very Left	Linear S-shaped	[3 8]
	Slight Left	Triangular	[0 1 3]
	Straight	Triangular	[-2 0 2]
	Slight Right	Triangular	[-3 -1 0]
	Very Right	Linear Z-shaped	[-8 -3]

Three membership functions were used for each input (see Appendix B. Membership functions). This was done to avoid complicating the rule definition and creating a system that is more prone to design errors. As we want to follow a wall, the ideal distance is represented by 'Medium', while 'Close' is used when a collision is imminent and in contrary, 'Far' represent an important distance or no detection. To ensure efficient obstacle avoidance, it is necessary to detect front obstacles faster than those on the left or right, as the robot is constantly moving forwards. This is the reason why the 'Near' membership function differs for the 'Front' input.

The robot operates in a relatively simple environment, and therefore only needs to handle a limited number of scenarios. As a result, three types of speed are sufficient. Here, the term 'medium' refers to the cruising speed, 'slow' should be used in narrower environments where obstacles are close, and on the contrary 'fast' should be used when the path is clear.

Finally, the turn rate is the most complex fuzzy set, as it requires precise control, which is why there are five membership functions. All these functions have the same form but are centred on different values.

1.5 The rule set

To design our rule set properly, it is important to spend some time writing down rules inside a Fuzzy Associative Matrix to understand how rules can be merged and quickly spot design problems.

Speed				
Front	Right	Far	Medium	Close
	Left			
Far	Far	Medium	Medium	Slow
	Medium	Fast	Fast	Fast
	Close	Medium	Medium	Slow
Medium	Far	Medium	Medium	Slow
	Medium	Medium	Medium	Slow
	Close	Medium	Medium	Slow
Close	Far	Slow	Slow	Slow
	Medium	Slow	Slow	Slow
	Close	Slow	Slow	Slow

Turning rate				
Front	Right	Far	Medium	Close
	Left			
Far	Far	Slight left	Very left	Very left
	Medium	Straight	Straight	Slight left
	Close	Slight right	Slight right	Straight
Medium	Far	Slight right	Very left	Very left
	Medium	Slight right	Slight right	Very right
	Close	Very right	Slight right	Very right
Close	Far	Very right	Very left	Very left
	Medium	Very right	Very right	Very right
	Close	Very right	Very right	Very right

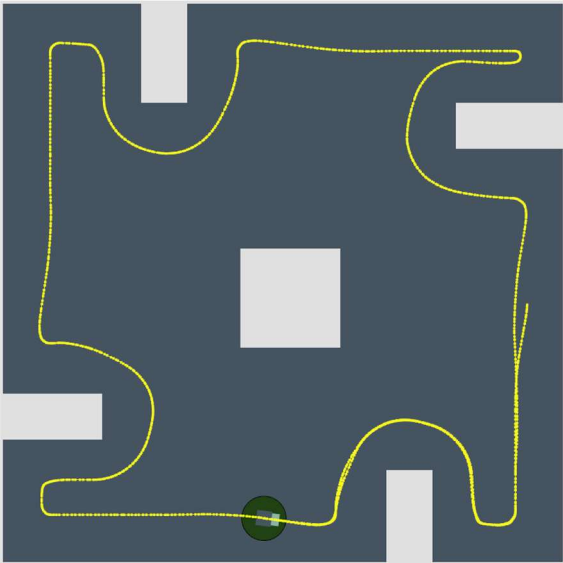
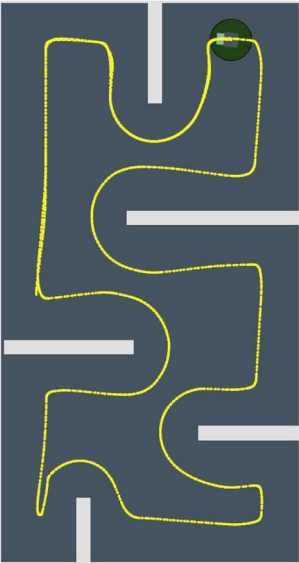
The FAM above can be simplified by dividing it into more general states. Firstly, when the robot is close to an obstacle in front, it should slow down or come to a complete stop to avoid collision. Secondly, for the right and left directions, the robot should slow down only if it is in a narrow corridor where it could hit a wall. Otherwise, it can continue at a medium speed. Finally, when the situation is perfect, meaning that the robot could even travel faster to optimise the distance travelled.

To determine the turning rate, we can apply the same method. When a front obstacle is detected, a sharp right turn should be taken to avoid it while keeping the wall on the left. However, in the edge case where a wall is detected on the right, the direction should be changed to ensure the wall is on the left, requiring a sharp left turn. Then, all the other rules have been designed with the same idea, but including some nuance so that we only take a small turn if we don't need a sharp turn and even keep straight in some situations where turning would result in robot trajectory instability.

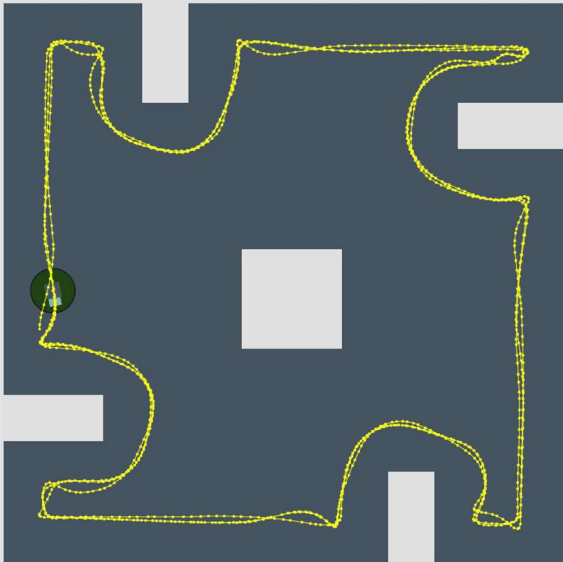
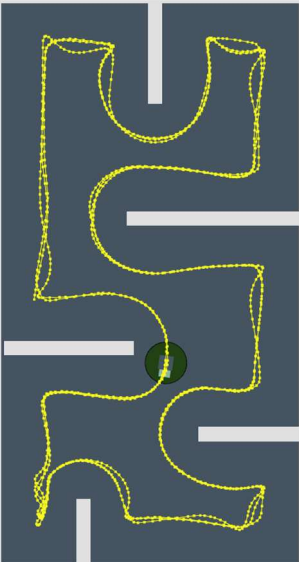
All the rule defining in the FIS file are listed in Appendix C. Rules set.

2. Performance Evaluation

2.1 Testing of wall following

Arena 1	Arena 2
	
<div>distance travelled forwards: 400.6 cm backwards 0.0 cm total: 400.6 cm straight: 43.7 cm</div>	<div>distance travelled forwards: 394.1 cm backwards 0.0 cm total: 394.1 cm straight: 44.3 cm</div>

As our system runs smoothly at native speed and turn rate, we can try to optimise the distance travelled by scaling the FIS outputs before sending them to the motors. Here are the results with x2.5 scaling applied to speed and turn rate.

Arena 1	Arena 2
	
<div>distance travelled forwards: 991.5 cm backwards 0.0 cm total: 991.5 cm straight: 5.3 cm</div>	<div>distance travelled forwards: 937.8 cm backwards 0.0 cm total: 937.8 cm straight: 20.8 cm</div>

The distance travelled by the robot has been increased by almost 2.5 times. However, upon closer examination of the trajectory, it is evident that this increase has come at the cost of stability. Therefore, it is not feasible to continuously enhance the system due to real-life limitations. The value of 2.5 is the highest recorded during tests to improve distance while maintaining acceptable stability for the robot.

All testing from now on will be conducted without any scaling applied, as the focus is on the trajectory rather than speed.

2.2 Testing of obstacle avoidance

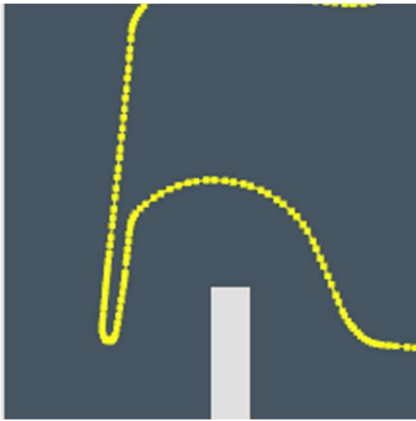


Figure 2: Narrow corridor in arena 2.

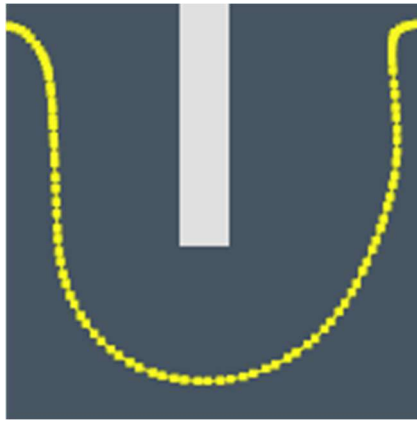


Figure 3: Obstacle's corner in arena 2.

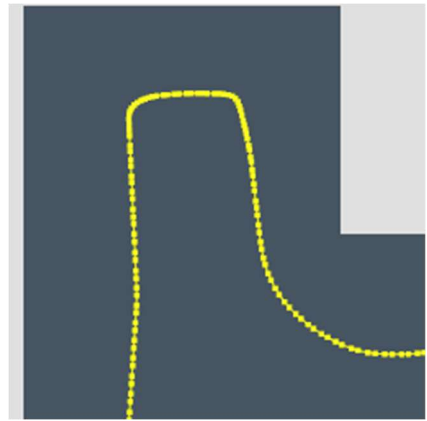


Figure 4: Corner of arena 1.

Based on the screenshots, it appears that our robot performs well in challenging areas such as corners and narrow spaces. However, there is a noticeable difference in its performance between open and closed obstacles. In closed corridors, the robot's path is almost squared, while in open obstacle line wall corners, the trajectory is more circular, which is slightly less efficient.

2.3 Testing of wall finding

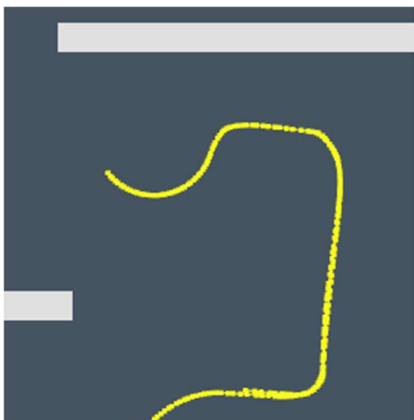


Figure 5: Wall finding from open area in arena 2

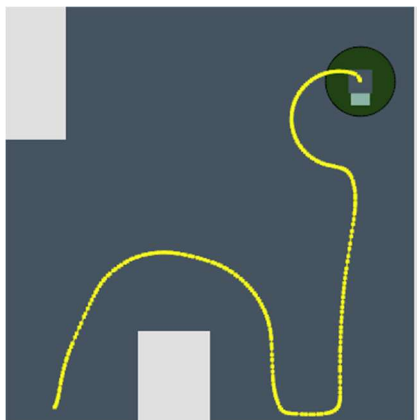


Figure 6: Start from opposite wall in arena 1.

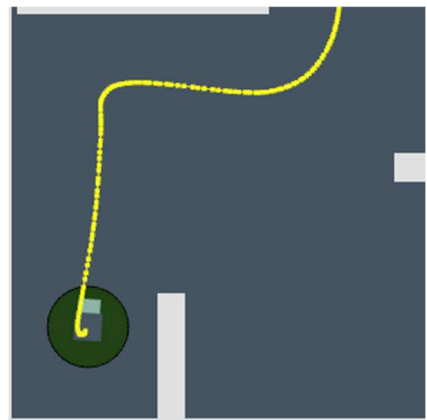


Figure 7: Start facing narrow corridor in arena 2.

The screenshots demonstrate the performance of our FIS in narrow spaces and its ability to correct its position when the wall is not on the left side of the robot as intended.

2.4 Result analysis

To conclude this testing section, it is important to analyse our system and evaluate its performance. The wall following function of the FIS performed well, although the robot's speed is slightly low with the native output. However, the trajectory is smooth and stable without any instances of losing contact with the wall. To address

the speed issue, we can increase the speed and turning rate to 'overclock' our FIS, allowing for greater distance travelled at the cost of some instability.

Additionally, as previously mentioned, the system excels at avoiding obstacles and maintaining a smooth path even in challenging environments such as narrow corridors. However, the system struggles to recover from sharp outer corners, as demonstrated in Figure 8: Obstacle's corner in arena 2. It would be advantageous to find a solution that enables the robot to take sharper turns to increase efficiency.

The most challenging and least performant aspect is the wall finding. The robot attempts to turn around itself while moving forward until it finds a wall. In arenas 1 and 2, finding a wall to follow is almost never a problem due to the close proximity of walls. However, in completely empty areas, the robot may struggle to locate a wall to follow and start spinning around itself indefinitely.

2.5 Further development

During the performance analysis of our FIS, we identified two issues. Firstly, the robot's trajectory when navigating sharp outer corners, and secondly, the difficulty in detecting walls in open areas. As our inputs are limited, when the robot encounters a sharp edge, it immediately loses track of the wall on its left. Therefore, in the scenario where none of the three sensors detect any obstacles, the robot is in a wall-finding situation. To recover from this situation quickly, we instruct the robot to make a slight left turn. If the robot was already close to a wall, it will quickly detect it and continue to follow it as expected. However, if the robot is in an open area, we want it to move straight until it finds a wall to follow. The chosen inputs and hardware limitation do not allow for differentiation between the two scenarios. Therefore, the FIS is implemented with the assumption that the robot will frequently encounter the first scenario. To address this issue, one possible solution is to use sensors with a greater range that can detect walls diagonally. For example, a sensor placed on the back left of the robot could identify whether the robot is approaching an edge or is in an open space. However, it is important to note that adding a new input will significantly increase the number of rules that need to be designed.

Appendix A. My Matlab FIS file.

Listing of FIS file:

- navigate.fis (FIS MATLAB file)
- FIS_navigate.m (Robot control and FIS loader)
- arena_1.kad & arena_2.kad (KiKS default arena for testing)

Appendix B. Membership functions.

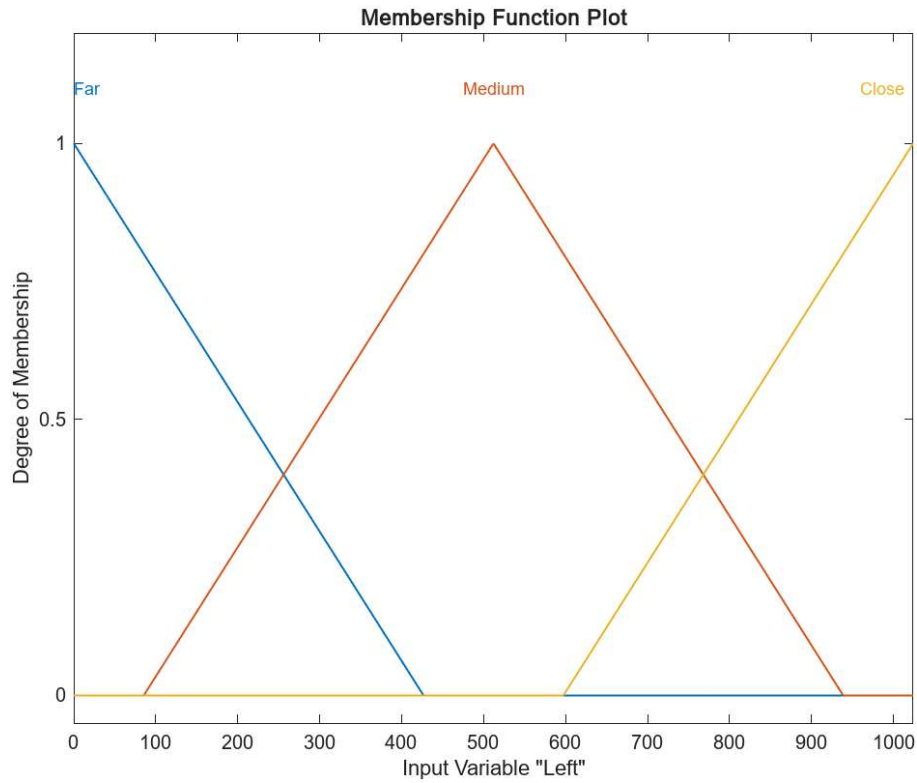


Figure 8: Input Left & Right membership functions.

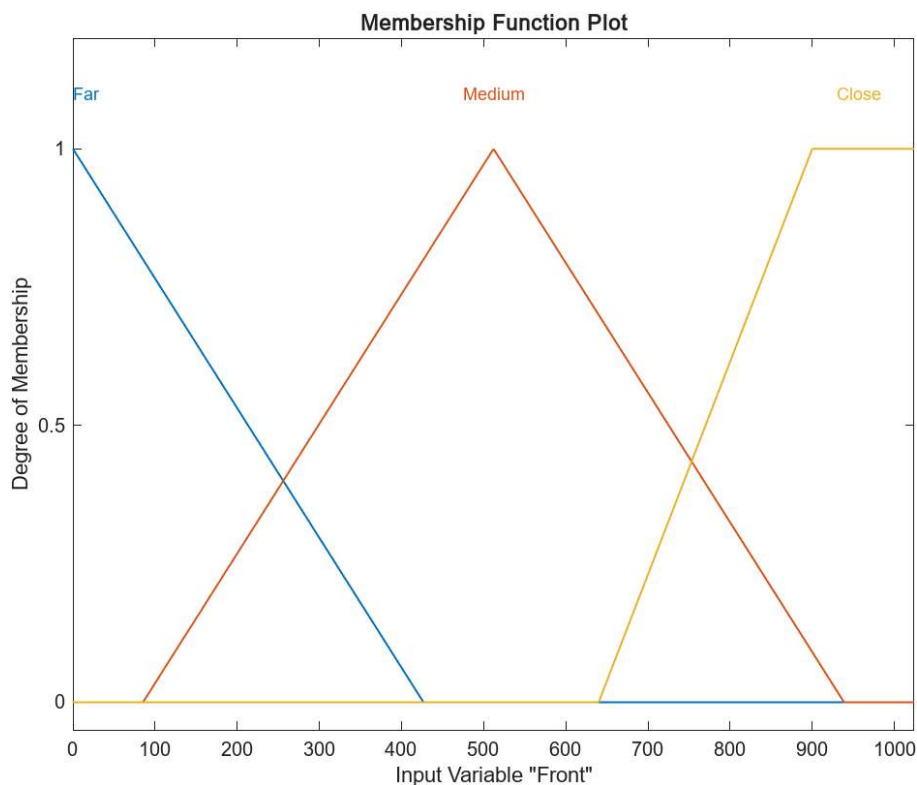


Figure 9: Input Front membership functions.

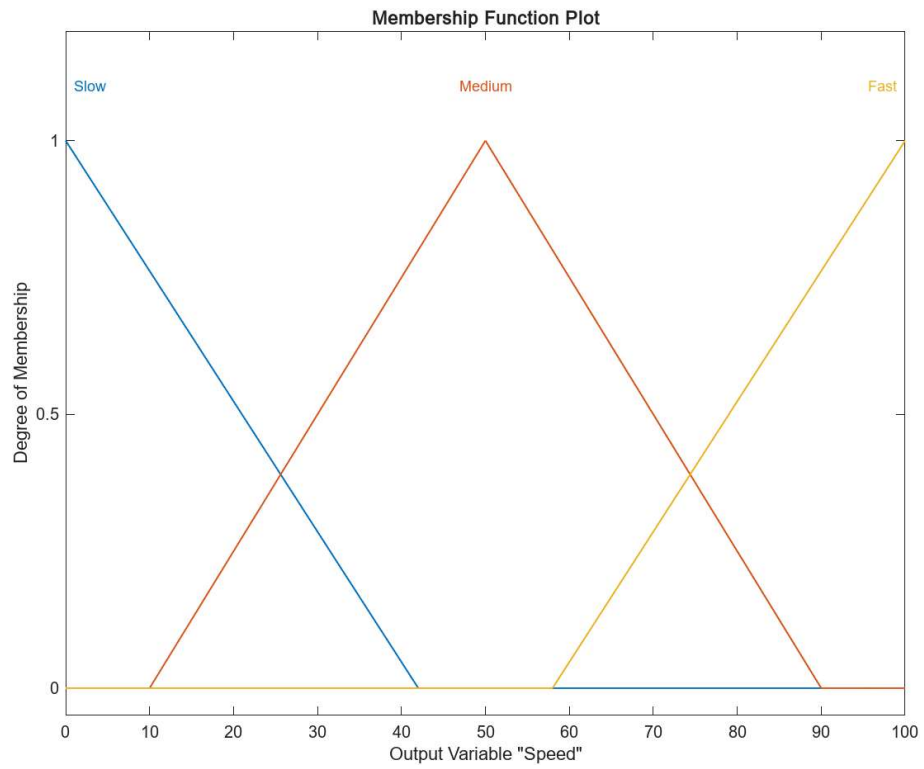


Figure 10: Output speed membership functions.

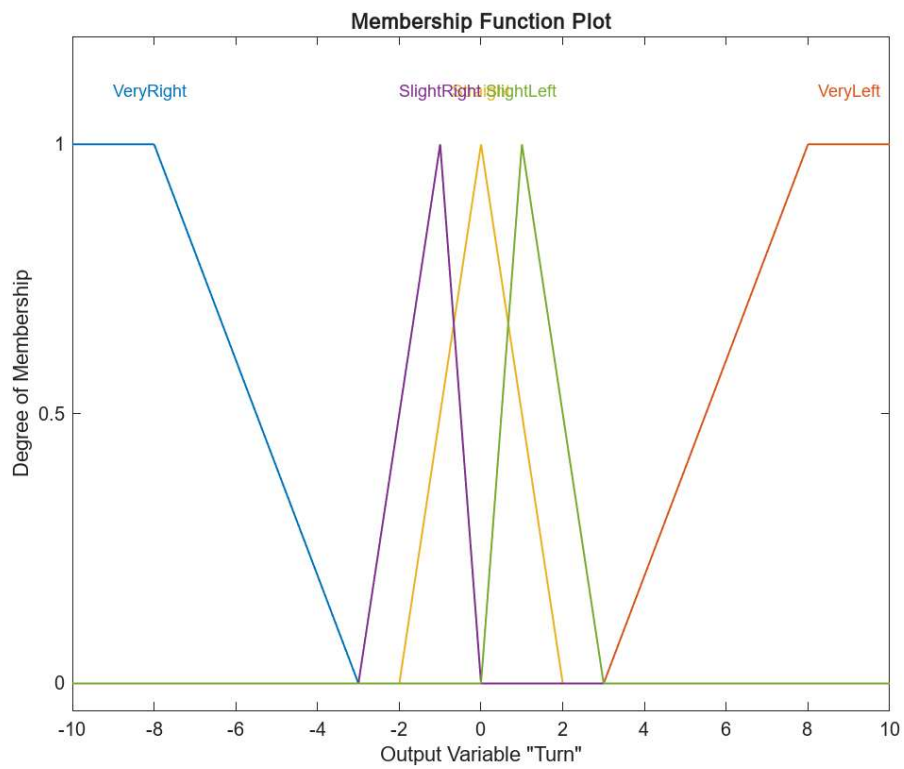


Figure 11: Output Turn membership functions.

Appendix C. Rules set.



1. If (Front is Far) and (Left is Far) and (Right is Far) then (Speed is Medium)(Turn is SlightLeft) (1)
2. If (Front is Far) and (Left is Medium) and (Right is Far) then (Speed is Fast)(Turn is Straight) (1)
3. If (Front is Far) and (Left is Close) and (Right is Far) then (Speed is Medium)(Turn is SlightRight) (1)
4. If (Front is Medium) and (Left is Far) and (Right is Far) then (Speed is Medium)(Turn is SlightRight) (1)
5. If (Front is Medium) and (Left is Medium) and (Right is Far) then (Speed is Medium)(Turn is SlightRight) (1)
6. If (Front is Medium) and (Left is Close) and (Right is Far) then (Speed is Medium)(Turn is VeryRight) (1)
7. If (Front is Close) and (Left is Far) and (Right is Far) then (Speed is Slow)(Turn is VeryRight) (1)
8. If (Front is Close) and (Left is Medium) and (Right is Far) then (Speed is Slow)(Turn is VeryRight) (1)
9. If (Front is Close) and (Left is Close) and (Right is Far) then (Speed is Slow)(Turn is VeryRight) (1)
10. If (Front is Far) and (Left is Far) and (Right is Medium) then (Speed is Medium)(Turn is VeryLeft) (1)
11. If (Front is Far) and (Left is Medium) and (Right is Medium) then (Speed is Fast)(Turn is Straight) (1)
12. If (Front is Far) and (Left is Close) and (Right is Medium) then (Speed is Medium)(Turn is SlightRight) (1)
13. If (Front is Medium) and (Left is Far) and (Right is Medium) then (Speed is Medium)(Turn is VeryLeft) (1)
14. If (Front is Medium) and (Left is Medium) and (Right is Medium) then (Speed is Medium)(Turn is SlightRight) (1)
15. If (Front is Medium) and (Left is Close) and (Right is Medium) then (Speed is Medium)(Turn is SlightRight) (1)
16. If (Front is Close) and (Left is Far) and (Right is Medium) then (Speed is Slow)(Turn is VeryLeft) (1)
17. If (Front is Close) and (Left is Medium) and (Right is Medium) then (Speed is Slow)(Turn is VeryRight) (1)
18. If (Front is Close) and (Left is Close) and (Right is Medium) then (Speed is Slow)(Turn is VeryRight) (1)
19. If (Front is Far) and (Left is Far) and (Right is Close) then (Speed is Slow)(Turn is VeryLeft) (1)
20. If (Front is Far) and (Left is Medium) and (Right is Close) then (Speed is Fast)(Turn is SlightLeft) (1)
21. If (Front is Far) and (Left is Close) and (Right is Close) then (Speed is Slow)(Turn is Straight) (1)
22. If (Front is Medium) and (Left is Far) and (Right is Close) then (Speed is Slow)(Turn is VeryLeft) (1)
23. If (Front is Medium) and (Left is Medium) and (Right is Close) then (Speed is Slow)(Turn is VeryRight) (1)
24. If (Front is Medium) and (Left is Close) and (Right is Close) then (Speed is Slow)(Turn is VeryRight) (1)
25. If (Front is Close) and (Left is Far) and (Right is Close) then (Speed is Slow)(Turn is VeryLeft) (1)
26. If (Front is Close) and (Left is Medium) and (Right is Close) then (Speed is Slow)(Turn is VeryRight) (1)
27. If (Front is Close) and (Left is Close) and (Right is Close) then (Speed is Slow)(Turn is VeryRight) (1)

Figure 12: Rules set.