Fuzzy Logic Controller for Obstacle Avoidance and Swarm Formation Control

Anvith Pabba and Hema Landa

Contributing authors: ee19b070@smail.iitm.ac.in;
ee19b036@smail.iitm.ac.in;

Abstract

In this Report, we take a look at the design and implementation of a **fuzzy logic controller** capable of guiding a swarm of robots from a start position to a fixed target position, while ensuring that none of the robots collide with any obstacles present along the way. Each robot is modelled so that it has 3 sensors, one on the top, one on the left, and one on the right. We implement the decentralized (i.e individual robot level) obstacle avoidance using a Mamdani Fuzzy Inference system. Where the inputs are the fuzzified values of the obstacle distances given by the sensors. Finally, we implement the centralized swarm control via a leader-follower implementation in which the leader sends the expected path each follower has to move in. The interplay of choosing the path necessary to avoid obstacles and the path given by the leader for swarm control is done using a dynamic weight alpha, which is also explained in detail.

1 Introduction: What exactly is Swarm Control and Obstacle avoidance?

Swarm control consists of maintaining a fixed arrangement of robots, that make up the "swarm". This swarm has to travel from the start to the end while maintaining its formation. If the swarm encounters any obstacles, then the swarm separates or deforms in order to pass the obstacle, and when possible it regains its swarm formation. Any swarms that follows this is said to have "obstacle avoidance" and "swarm control".

2 Motivation: Why and Where is this useful?

Swarms can be used in a myriad of applications such as:

- In military operations such as aerial drones.
- In Exploration of foreign lands.
- In industries such as agriculture.
- In mining and excavation.
- A substitute for natural swarms, such as swarms of bees to pollinate fields.
- etc...

Swarms are mainly useful when singular robots are not sufficient to complete a proposed task. Swarms are theorised to almost entirely replace conventional armies in the future. They can also be scaled up to create swarms of artificial bees that can pollinate entire fields and prevent disastrous famines.

As the terrain the swarm will move in is not known to the swarm beforehand, to safely traverse and avoid hitting obstacles, the swarm needs to have a mechanism to avoid these obstacles while also maintaining its formation. Fuzzy logic is a great way of imparting this domain knowledge into the controller in the form of intuitive IF-THEN statements, and hence it is a good approach to try and implement. **This is the main inspiration for this project**.

3 Problem Statement:

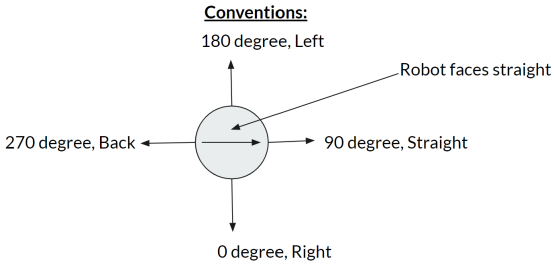
Swarm Formations are groups of robots aligned in a certain spatial pattern. This spatial pattern is also referred to as a "formation", and has to be consistent as much as possible throughout a simulation irrespective of the presence of obstacles. In order to do this, we need to provide accurate speeds and directions (angle) such that each robot in the swarm has the ability to break the formation ,maneuver through an obstacle and join back into the formation as intended. These speeds and directions are to be given using a controller. **Hence in this project, our goal is to design this control system using fuzzy logic (hence a "fuzzy logic controller") that provides this speed and angle such that**:

- No robot in the swarm collides with an obstacle (**obstacle avoidance**)
- And that the robots go to the initial formation when possible (**formation control**)

4 Assumptions Taken:

1. The environment the robot traverses in is a 101x101 matrix, in which the obstacles can be inserted by the user.
2. The obstacles are static.
3. Each robot has 3 sensors, one on the top (90 degree), one on the left (180 degree) and one on the right (0 degree).

4. The top half of the robot consisting of the sensors can only look towards the right at all times, but the bottom half containing the wheels can twist in any direction/speed.
5. One robot is made the leader, and all the other robots try to get in the correct position with respect to this leader robot.
6. The leader can communicate (i.e send information) to all the follower robots.
7. Each robot knows its exact coordinates at all times. (This assumption can easily be implemented in real life using a dedicated processor)
8. the swarm control is **Centralized** and is given to each robot by the leader
9. the obstacle avoidance is **Decentralized** and is computed by the fuzzy controller at the robot level.



5 Non-Triviality Of The Problem:

Obstacle avoidance is done using the readings of the sensors and analysing the best way to move. Previous implementations such as PID controllers use complex mathematical formulas and negative feedback loops to achieve this goal. which requires specialised knowledge of the domain. Another implementation is using deep learning, such as in autonomous vehicles. Notable examples of this are Tesla and Waymo which are essentially data science companies. The issue with this approach, is that u need to have extremely large amounts of data to get a good model that performs well. Getting this dataset is not easy and requires a lot of testing/modelling and collection of data from these tests which takes a significant amount of money and time to achieve.

Hence, designing a good controller with a lower model complexity is the challenge that we attempt to tackle using Fuzzy logic, As this control system can make decisions based on logic rules that mimic an expert.

6 Literature Review:

[1] Syed Ahemed (2019) uses a PID controller in order to achieve single robot obstacle avoidance, [2] Aqeel-Ur-Rehman and C. Cai (2020) use a fuzzy-PID controller to do the same, [3] Omrane et al. (2016) implemented a fuzzy logic

controller for obstacle avoidance, but all the above are unable to handle swarms of robots. [4] rendyansyah et al. explains the improvement of using fuzzy logic over PIDs. [5] Dewi et al. (2018) show the results of fuzzy logic obstacle avoidance and swarm control, but do not mention anywhere how the swarm control was implemented! only the results are shown, hence our improvement, is that we will also be showing how to implement all the features along with the relevant code.

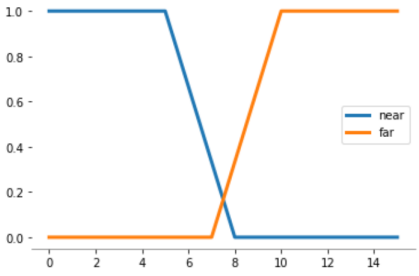
7 Proposed approach:

7.1 Decentralized Obstacle avoidance

To do this, we use a Mamdani Inference System that takes in the input distances of the 3 sensors and based on 8 rules, accordingly gives the output speed and angle the robot has to take in order to avoid the obstacle.

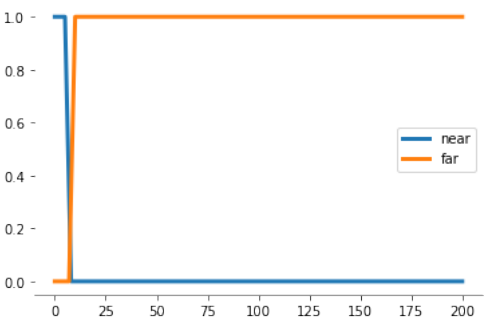
7.1.1 The Fuzzified Sensor values

The 3 sensors s0,s90,s180 give the integer value of the closest obstacle in the respective direction. We then fuzzify these crisp values using the following fuzzy sets, which consist of "near" and "far":



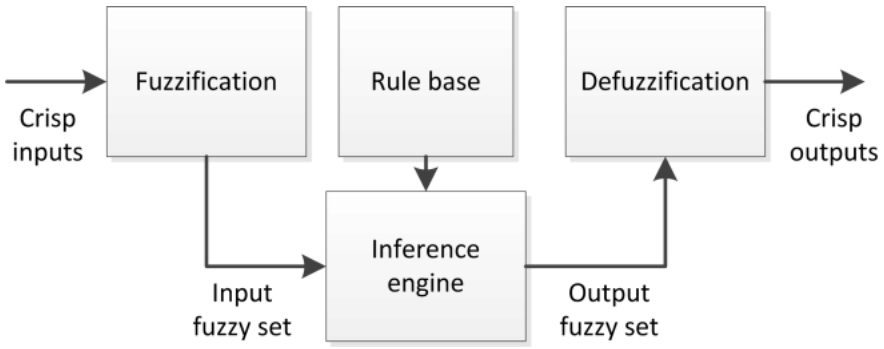
... upto the length of the maze used

If the range is up to 200, the fuzzy set is as follows:



7.1.2 The Fuzzy Logic Controller (FLC)

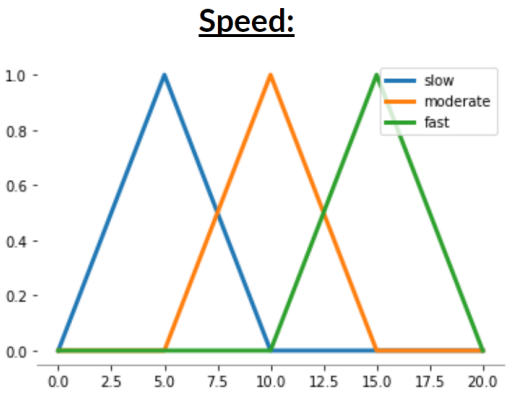
We now, input these fuzzy inputs into the Mamdani inference system, which consists of 8 IF-THEN rules. The output MAX-MIN truncated speed and angle is then defuzzified using the "centre of gravity" method to get the final results.



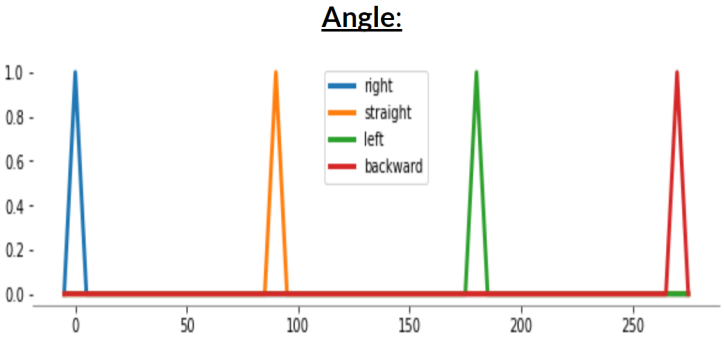
7.1.3 The Fuzzy Outputs

The outputs are the speed and the angle, these fuzzy sets are as follows:

Speed consists of "slow","moderate", and "fast".



Angle consists of "right","straight","left" and "backward"



7.1.4 The Mamdani Rules

There are 8 Mamdani rules as follows:

S.No.	If s0 is	And s90 is	And s180 is	Then speed is	And angle is
1	near	near	near	slow	backwards
2	near	near	far	moderate	left
3	near	far	near	fast	straight
4	near	far	far	fast	straight
5	far	near	near	moderate	right
6	far	near	far	moderate	right
7	far	far	near	fast	straight
8	far	far	far	fast	straight

These rules are very easy to understand and are extremely intuitive. This is how we impart domain knowledge into the controller.

7.1.5 Implementation

We implement all of this in python using the "Fuzzy_Expert" package. First, we put the input and output fuzzy sets. We then add the rules in the IF-THEN format specified. We then build the model and use it in all further experiments.

7.1.6 Sanity check

Example 1: when s0 is 20 (far), s90 is 5 (near), s180 is 5 (near), then the output is moderately fast (10) and right (angle is close to 0). The model also shows the same!

```
In [243]:
model(
    variables=variables,
    rules=rules,
    sensor0=20,
    sensor90=5,
    sensor180=5,
)
```

```
Out[243]: ({'speed': 9.999999999999996, 'angle': -2.2794266474335246e-16}, 1.0)
```

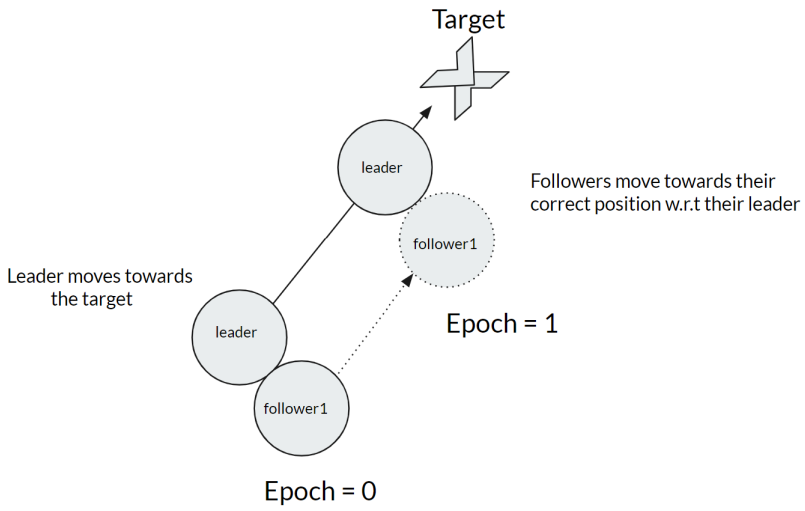
7.2 Centralized Swarm Control

In this, a swarm consists of a group of robots with one robot designated to be the leader. All the other robots control the swarm by moving into the correct position w.r.t the leader. As the leader knows all the other robots positions, and where they need to be w.r.t, it can accurately tell them how they need to move.

The leader always moves towards the **target**.

The robots always move towards their correct position w.r.t their **leader**.

The figure below illustrates this:



This can be scaled up for any number of robots in the swarm. A vector with 3 values is given to each robot where, the first and third represent how the follower should move in the y-direction, and the second value represents how the follower should move in the x-direction. This is calculated by the following process:

- direction = position it needs to go to - current position
- vector = [0,0,0]
- vector[0] = max(0, -1*direction[1])
- vector[2] = max(0, direction[1])
- vector[1] = direction[0]
- normalize direction
- direction = round(direction*speed of follower) (speed was taken as 3)

Now, "direction" represents how the robot/leader needs to move for swarm control!

7.3 Interplay between the two: Introduction of Alpha

We now have 2 paths that a robot can take,

1. First, it can go the obstacle avoidance route.
2. Second, it can go the formation control route.

So in order to decide which path it should take, we use **alpha**, where alpha is in [0,1], such that:

Final path = alpha * Obstacle Avoidance + (1-alpha) * Formation Control

Now, its easy to see that whenever the robot is about to hit an obstacle, alpha has to be 1 and when no obstacles are nearby, alpha has to be 0.

7.4 Determining Alpha

We determine alpha as follows:

If the robot follows the "Formation Control" path, but hits an obstacle on its way, then alpha is made 1 and it follows Obstacle avoidance. If the robot doesn't hit any obstacle, then alpha is 0. This is implemented using the "pathchecker" function.

Hence, the combination of these 3 is what determines how each robot moves.

8 Useful Functions in the Code Implementation

The code implementation: https://colab.research.google.com/drive/1NXJedxerg9alWTLd5kL8Dt3U_uEcwKVh?usp=sharing

"numbots" = number of bots

"actualwrtleader" = position of these bots wrt to the leader, the first is always [0,0] to signify that its the leader. Example, [-5,-5] indicates that the second robot is 5 units below and 5 units to the left of the leader!

"Target" = coordinates of the final target

"insertrectangle(maze,left corner position, width w, length l)" can be used to insert a rectangle obstacle at the desired location with a width of "w" and a length of "l"

"insertrectanglecoords" can be used to get the x and y coordinates of all the points in this obstacle, which is useful in the final plotting.

"insertline(maze, corner, length, horizontal or vertical)" can be used to insert either a horizontal or a vertical line in the maze.

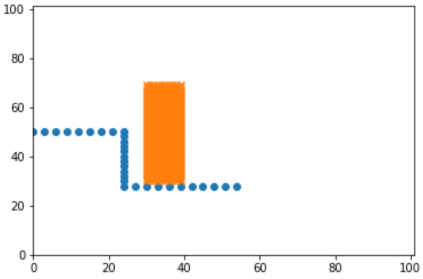
Similarly, "insertlinecoords" can be used to get the x and y coordinates of the points in the line.

9 Experimental Results:

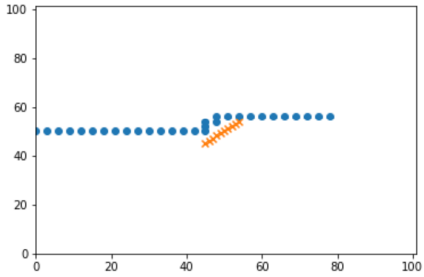
9.1 Pure Obstacle Avoidance:

Here, the robot just needs to avoid the obstacle. Blue is the path of the robot, And orange represents the Obstacle.

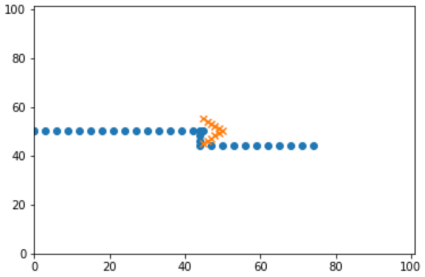
Rectangle at the center



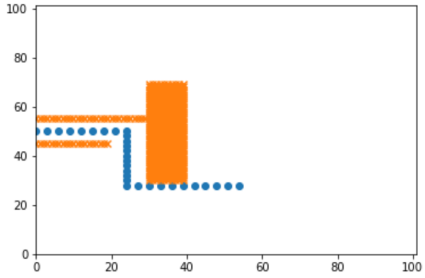
Slanted Object

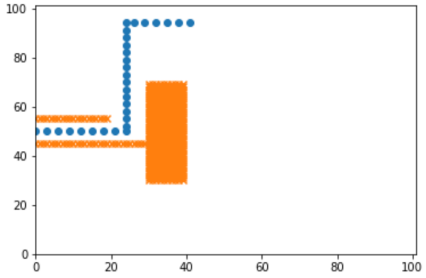


2 Sided Object



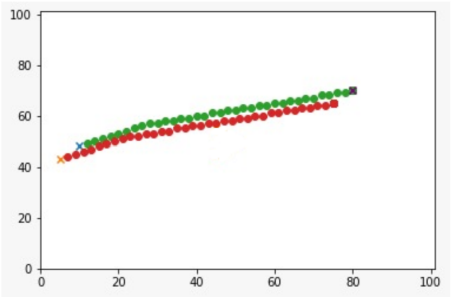
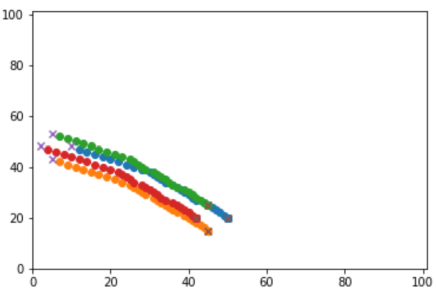
Either wall Closed off



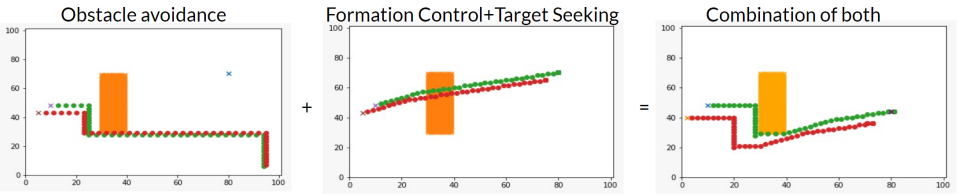


9.2 Pure Formation Control:

in this, an n-bot system just goes to its target, as there are no obstacles, this is to demonstrate how the formation control works.

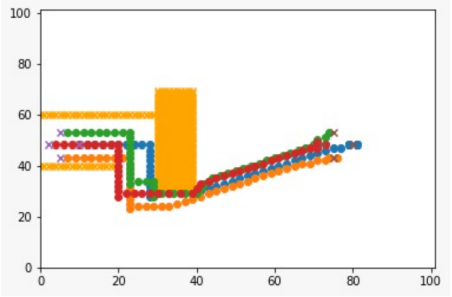
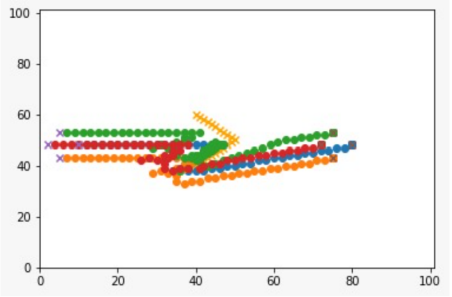



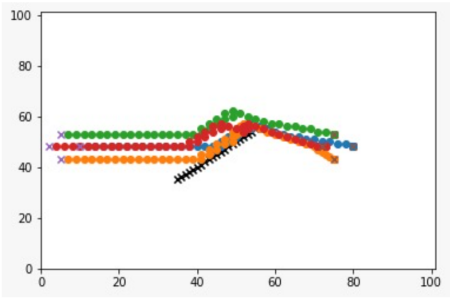
9.2.1 Showing the combination of both:

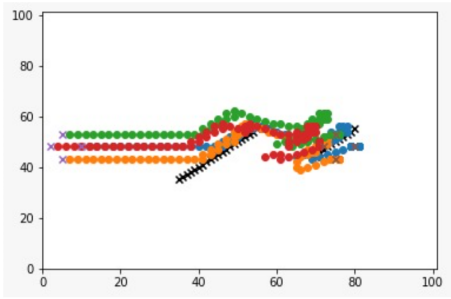


9.3 Obstacle Avoidance and Formation Control:

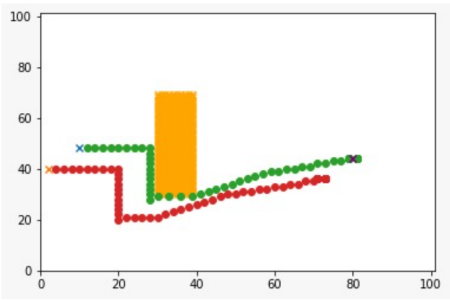
9.3.1 For a 4 robot diamond shape swarm

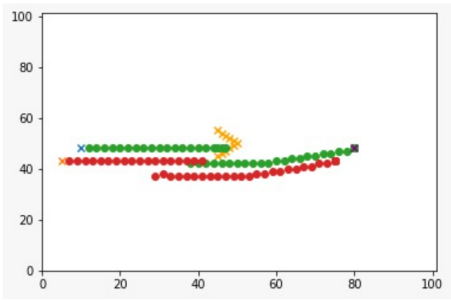



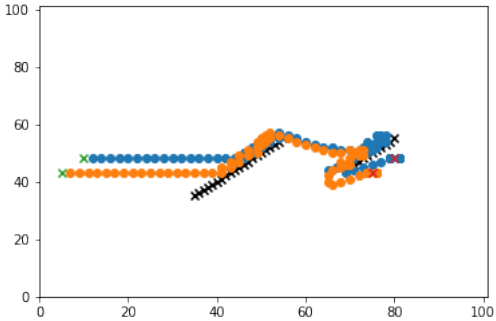




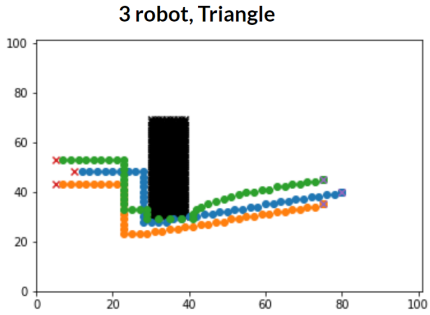
9.3.2 For a 2 robot diagonal shape swarm

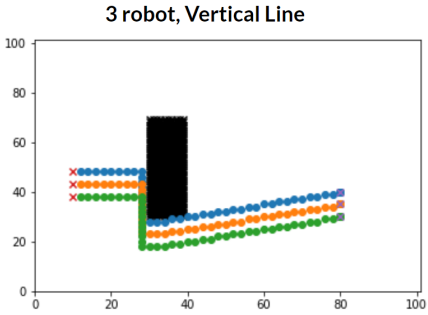






9.3.3 3 robot with different shapes





This g-drive has all the videos:

- https://drive.google.com/drive/folders/1rOQco5PnT8t8jBK86hK6$_1$n2Uc7EL0Ma?usp = *sharing*

10 Results and Analysis:

- Mamdani engine slows considerably as the number of rules increases. Initially, 3 fuzzy sets of near, medium and far were used with 27 rules, but lead to considerably slower times with no real benefits.
  1. for 27 rules, time taken for 1 inference was 43 seconds on my local machine and 13 seconds on google colab.
  2. for 8 rules, time taken for 1 inference was around 7 seconds on my local machine and roughly 2 seconds on google colab. **Much faster!**
- The angle outputs have to be extremely sharp in order for the robots to consistently avoid obstacles.
- Symmetric outputs lead to deadlocks which were solved using **backtracking** and taking a hard right turn to escape the deadlock.
- This method works well for right biased obstacles, i.e obstacles present on the right side. To fix this, we have implemented a system where the robot turns its head along the specified angle apart from just movement in order to avoid right bias. (as now the robot can turn to the left side)
- We have tested different variations of alpha such as "alphaforAvoidancevs-Control" and "alphadash", and found that **"alphadash"** was the best, hence we have included that in the alpha section.
- The system works best when speed of fuzzy control is 2, and the scaling factor for obstacle avoidance is 0.2 .
- ***Timing analysis:*** 100 epochs with 3 robots takes roughly 9 minutes, 100 epochs with 4 robots takes roughly 12 minutes and 50 epochs with 4 robots

takes roughly 6 minutes. Hence, the time it takes scales **Linearly** with the number of robots and the number of epochs.
- This implementation is **NOT good when the obstacles are dynamic and can move!**

11 Critical analysis of the Fuzzy logic Implementation

Although it works well for most cases, fuzzy logic would not perform as well when the obstacles are more frequent and narrow.

In that case, we would need to have a system that performs multiple runs over the environment. Each run would have to improve the knowledge of the maze until the robot knows exactly how to go. After mapping the entire map, we could run path finding algorithms such as **A star** algorithm or **Yen's k-Shortest Paths** algorithm to find the shortest path for each robot in the swarm.

Fuzzy logic takes a lot of time to run when there are a considerable number of robots in the swarm and the number of epochs is high. This is an issue, as the velocity has to be as slow as possible, so that the robots do not hit any obstacles, but this then increases the number of epochs required, leading to longer times.

Another issue is that swarms as usually much harder to control when there are an extremely large number of robots. In this case, decentralised swarm control could potentially be better.

12 Potential Future Work

In the future, we could try to implement more sensors such as 5 sensors in the front and 3 sensors on the back. This would need exponentially more number of rules, and as a result, exponentially more time to run. So pruning these rules to get an optimised set is one option. Another is to implement our own version of the Mamdani inference system such that it is optimised to the maximum extent.

We could also try analysing how adding more number of exploration runs would affect the results, along with running another path finding algorithm in parallel to this controller.

Finally, we could try optimising the fuzzy sets used by using reinforcement learning or deep learning.

13 Conclusion

In this project, we have successfully implemented a Fuzzy Logic Controller that works on **ANY** formation of robots such that it performs obstacle avoidance and formation control, while also seeking a fixed target. This has also been implemented in python code, and We have then run multiple experiments with Simulations of the swarms and how they interact with obstacles. Finally, We also give our results and our analysis.

References

[1] Ahmed, Syed. (2019). Navigation and Obstacle Avoidance Control of an Autonomous Differential Wheeled Robot Using PID Controller in Matlab Simulation - Academic Project Report. 10.13140/RG.2.2.21943.14246.

[2] Aqeel-Ur-Rehman and C. Cai, "Autonomous Mobile Robot Obstacle Avoidance Using Fuzzy-PID Controller in Robot's Varying Dynamics," 2020 39th Chinese Control Conference (CCC), 2020, pp. 2182-2186, doi: 10.23919/CCC50068.2020.9188467.

[3] Omrane, Hajer Masmoudi, Mohamed Masmoudi, Mohamed. (2016). Fuzzy Logic Based Control for Autonomous Mobile Robot Navigation. Computational Intelligence and Neuroscience. 2016. 1-10. 10.1155/2016/9548482.

[4] RENDYANSYAH, Nurmaini, Siti SARI, Sartika RAMARTA, Dimas. (2020). Implementation of Fuzzy Logic Control for Navigation System in Mobile Robot Omnidirectional. 10.2991/aisr.k.200424.036.

[5] Dewi, Tresna Wijanarko, Yudi Risma, Pola Oktarina, Yurni. (2018). Fuzzy Logic Controller Design for Leader-Follower Robot Navigation. 298-303. 10.1109/EECSI.2018.8752696.