Autonomous obstacle avoidance using fuzzy logic

Jelle van Dijk
University of Amsterdam
Student ID: 10989048

Gawan Dekker
University of Amsterdam
Student ID: 11025654

Victor Gladys
University of Amsterdam
Student ID: 10523626

Abstract—**Self driving cars can no longer be avoided, every major company in the world is working on them. But how can fuzzy logic be used in this new branch of engineering.A simulation of an autonomous robot was created, the movements of the robot where guided by a fuzzy logic system. The goal was to create a fuzzy logic system that would guide the robot through the obstacles placed in the simulation without colliding with them. We succeeded in create such a fuzzy logic system, to some degree. The robot avoided tried to avoid obstacles but was not always successful. There is a lot fuzzy logic can do in the branch of autonomous vehicles.**

I. Introduction

Self driving cars are being developed at almost every major car and tech company in the world right now. Although they are still relatively early in development, they are considered by many to be the future of personal transportation. A lot of research has been done on using fuzzy logic in self driving cars and other autonomous moving vehicles. The different application for fuzzy logic in autonomous vehicles range from Cabrera-Cosetl, 2009 [2] where fuzzy logic was used allow a vehicle to park itself, to Gerdelan et al. 2006 [3] where a combination of A* and fuzzy logic was used to create a path finding robot with obstacle avoidance.

This paper will be about the obstacle avoidance aspect of fuzzy logic in autonomous vehicles. The goal is to better understand how fuzzy logic can be used for obstacle avoidance. To try and gain a better understanding of autonomous moving vehicles, in this project an attempt will be made to recreate a simple one using a fuzzy logic system (FLS).

II. Literature Review

There is already a lot of literature about trying to make a robot traverse an environment using fuzzy logic. In this section, we will briefly discuss three papers that are relevant to our problem.

In [1], X. Bajrami et al. create and compare two fuzzy logic implementations to guide a robot through a simulated environment with obstacles.

The first implementation uses a combination of FLSs with mamdani-type inferrence and handmade rulebases and membership functions to guide the robot. The input (distances to closest objects and goal, angle towards goal and preferred turn) is used to determine acceleration levels for the left and right wheels of the robot for different objectives (like obstacle avoidance) and to determine different weights for those objectives.

The second implementation used an FLS with a Sugeno-type inference system, where the membership functions were trained using a neuro-adaptive learning method.

After running tests in a simulated environment, the two implementations were compared.

Bajrami et al. [1] describes a FLS system but does not explain the design choices or in some cases do net specify the settings used in there FLS. There systems cannot be recreated for due to a lack of information. In this paper the results are also not given clearly given, they describe one of there test setups but do not specify the others.

In [3], Gerdelan et al. created a hybrid system to control a robot. The first layer of the system consisted of an A* algorithm. The A* algorithm is a path finding algorithm and is considered to be very fast. The A* path finding layer calculated the optimal route from the position of the robot to the end point. When traversing this route, the robot followeda set of way points. The second layer of the system is a fuzzy logic system. This system had as input information about the next way point as well as about the nearest obstacle. With these pieces of information a speed and turning speed for the robot were chosen.

This paper used some techniques that can be very useful for our project. Since they used a moving end point in their simulations, their test scenarios where harder. However, the same principles they used to control their robot will apply to the robot in the simulations used in our project.

One problem that the researchers in [3] encountered, was that in some situation the fuzzy logic system would try to avoid an obstacle and by doing so, would not be able to move to the next way point. This is something to keep in mind for our project, but could be fixed by tweaking the membership functions.

In [4], M. Mohammadian and R. J. Stonier used a genetic algorithm to adapt fuzzy rules in a two layer fuzzy logic system, which is then used by two robots to navigate towards a target without colliding. The genetic learning has been applied to generate a new layer of fuzzy rules that can be integrated into an already existing rulebase.

The first of the two fuzzy layers is used to determine the angle at which to continue moving, whilst the second layer determines the speed of the robot. By encoding the rulebase into a bitstring, Mohammadian *et al.*[4] were able to modify the rulebases with a genetic algorithm using cross-over and single-point mutations over a number of generations. To do this they used a modified cross-over procedure that ensured that these bitstrings were cut only at points that defined boundaries between rules.

The paper concludes in noting that, although most of their tests resulted in a positive outcome, some had trouble at the corners of the tested driving areas. This suggests that genetic algorithms find a maximum in optimizing fuzzy systems that is not necessarily easily applicable to a change in, or extension of, the initial learning environment.

III. Proposed Approach

A. Design

The fuzzy logic system used to control the robot has three inputs and two outputs. The inputs that are used correspond the distances in pixels to the nearest object in three different directions: front, front-left and front-right. Both the output variables are an angle, the angle represent how much the FLS wants to move left and right. The direction the robot will move in is the average angle between the two output values.

The FLS has a complete rule base, consisting of 27 rules. The rules are chosen such that the robot moves left or right if the distance to the other side becomes to small. The rules favor a left turn over a right one, meaning that if the distance to the left and to the right are equal the system moves left.

The three input variables all have the same membership functions. Consisting of the following meberships functions
low = trapezoid: [-1 -1 5 25]
medium = triangle: [5 25 70]
high = trapezoid: [25 70 1000 1000]
The input variables right most membership function goes to a 1000 because that is more then the maximum distance which can be measured in the created simulation.

The two output variables also have the same membership functions:
low' = trapmf: [0 0 0.09817477 0.19634954]
medium' = trimf: [0.09817477 0.19634954 0.29452431]
high = trapmf: [0.19634954 0.29452431 0.4 0.4]
The values of the output membership functions are angles in radials.

B. Implementation

The project implementations can be divided in to two parts, the Fuzzy Logic System and the simulation[1]. Both parts are coded in python[2], the following library's where used, numpy[3] for mathematical calculations and pygame[4] for visualizing the simulated environment.

Fuzzy Logic System implementation: The FLS implementation is an extension from the lab1 python exercise as found on blackboard. The FLS implementation contains a Reasoner class which given the input variables, output variables and rule base for the FLS can calculate the crisp value of the outputs given the input data points.

[1]The code for both can be found here: https://github.com/Yelvd/Fuzzy_logic_final_project

[2]Python 3.5.2 from www.python.org

[3]Numpy 1.11.0 from http://www.numpy.org/

[4]Pygame 1.9.3 from www.pygame.org/

The following options are available to use in the FLS implementation used in this paper.

Membership functions:

- Trapezoid
- Triangular
- Gaussian
- Bell shaped
- Sigmoidal curve

Aggregation methods:

- sum
- max

Defuzzification methods

- Smallest of max(SOM)
- Largest of max(LOM)
- Mean of max(MOM)
- Centroid

And/Or methods for rules

- Min (AND)
- Prod (AND)
- Max (OR)
- probor (OR)

Simulated Environment: The simulation environment consists of a checkerboard field. In this field a variable number of square obstacles and 'players' can be placed. The player movements can be controlled by either using the keyboard or giving it a FLS system to the player. If a FLS is given to a player it cannot be controlled manually. All placed obstacles have the ability to make random movements around the field, when this option is turned on the obstacle chooses a random direction and moves in that direction for a given number of frames before picking a new random direction.

When a moving object, either a player or an obstacle, tries to make a move which would result in two objects intersecting, this move is not allowed and the object will not move that frame. If this object is an obstacle the obstacle will rotate its moving direction 45 degrees clockwise.

When the player object, from now on referred to as robot, has an FLS assigned to it, then every frame the inputs for the FLS are calculated and given to the Reasoner. The crisp values returned by the FLS are then used to set the direction the robot will move in for that frame. In this paper three inputs are used for the movement of the robot. All inputs correspond to the distance in pixels from the edge of the robot to the next obstacle, which can either be a placed obstacle or the wall. Each input is the distance at a different angle ranging from [-85, 85]. These angles are based on the rotation of the robot, the front-left input uses the [25, 45, 65, 85] degree angles to determine the distance to the closest obstacle, the front-right uses the angles [-60, -45, -25, -85] and the front inputs uses [-15, -5, 5, 15]. From the multiple distance the lowest is given to the FLS as input, the minimum value for multiple angles is used to give the robot a 'broader' range of vision. The robot has a speed, which is set in the beginning of the simulation, it will constantly move at this speed unless it is colliding with something. The FLS can only choose the direction of the robot.

IV. Experiments and Results

Two starting situation were created for the simulation. These starting situations consisted of the robot in the corner at the bottom left and square obstacles of varying sizes placed around the field, see Figure 1 and Figure 2.

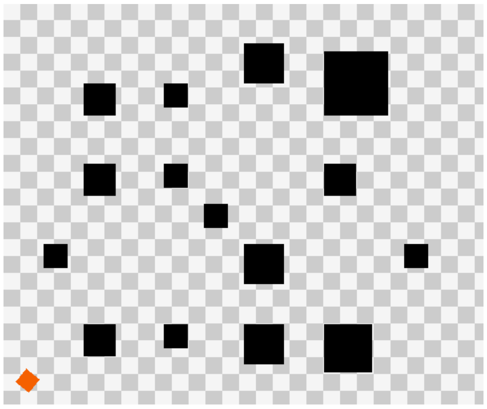
Fig. 1: starting situation 1 for the simulation

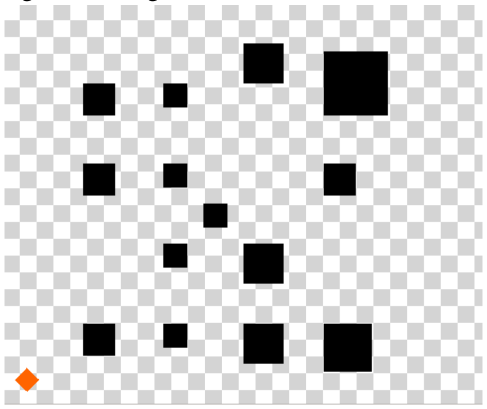
Fig. 2: starting situation 2 for the simulation

The experiment counts the number of collisions the robot makes with the obstacles and the walls. The simulation stops when 5000 ticks have been completed or when the robot gets stuck somewhere. Ticks were used as a time measurement unit to improve accuracy when the experiment is run on different systems. To collect data on the FLS multiple experiments multiple settings could be changed for each experiment. The settings that could be changed and their possible values were:

- Starting situation: s1 or s2, as seen in Figure 1 and Figure 2
- Aggregeation method: max or sum
- Defuzzification method: lom, som, mom or centroid
- Robot moving speed: 5 pixels/tick or 10 pixels/tick

All setting combinations are tested, because the simulation and the FLS are discrete each combination only needs one run.

A. Results

		N ticks	*N collisions*	$\frac{Ncollisions+1}{Nticks+1}$
De-fuzzifi-cation	*som*	286	3	0.01394
	mom	411	3	0.00971
	lom	1023	2	0.00293
	centroid	61	1	0.03226

TABLE I: player-speed=5; environment=1; aggregation=max

		N ticks	*N collisions*	$\frac{Ncollisions+1}{Nticks+1}$
De-fuzzifi-cation	*som*	69	1	0.02857
	mom	449	1	0.00444
	lom	62	1	0.03175
	centroid	61	1	0.03226

TABLE II: player-speed=5; environment=1; aggregation=sum

		N ticks	*N collisions*	$\frac{Ncollisions+1}{Nticks+1}$
De-fuzzifi-cation	*som*	263	1	0.00758
	mom	863	1	0.00231
	lom	292	1	0.00682
	centroid	156	1	0.01274

TABLE III: player-speed=5; environment=2; aggregation=max

		N ticks	*N collisions*	$\frac{Ncollisions+1}{Nticks+1}$
De-fuzzifi-cation	*som*	549	3	0.00727
	mom	1804	7	0.00443
	lom	68	1	0.02899
	centroid	5000	16	0.00340

TABLE IV: player-speed=5; environment=2; aggregation=sum

		N ticks	*N collisions*	$\frac{Ncollisions+1}{Nticks+1}$
De-fuzzifi-cation	*som*	14	1	0.13333
	mom	14	1	0.13333
	lom	14	1	0.13333
	centroid	32	1	0.06061

TABLE V: player-speed=10; environment=1; aggregation=max

		N ticks	*N collisions*	$\frac{Ncollisions+1}{Nticks+1}$
De-fuzzifi-cation	*som*	33	1	0.05882
	mom	32	1	0.06061
	lom	15	1	0.12500
	centroid	32	1	0.06061

TABLE VI: player-speed=10; environment=1; aggregation=sum

		N ticks	*N collisions*	$\frac{Ncollisions+1}{Nticks+1}$
De-fuzzifi-cation	*som*	72	1	0.02740
	mom	117	3	0.03390
	lom	70	1	0.02817
	centroid	24	1	0.08000

TABLE VII: player-speed=10; environment=2; aggregation=max

		N ticks	*N collisions*	$\frac{Ncollisions+1}{Nticks+1}$
De-fuzzifi-cation	*som*	72	1	0.02740
	mom	24	1	0.08000
	lom	35	1	0.05556
	centroid	24	1	0.08000

TABLE VIII: player-speed=10; environment=2; aggregation=sum

V. Discussion

Judging from our results, there were two visible problems with our FLS.

The first being that the robot seems to get stuck very easily. All results in the tables shown in the results section, except for one result in table IV have the simulation shut down before the maximum number of ticks (5000) has passed. Moreover, all results, except for two show that the robot got stuck before 1000 ticks passed.

Furthermore, the tables show that the robot gets stuck at a small amount of collisions. Most of the time the first collision the robot has with an object makes it get stuck. This is yet another sign that the robot gets stuck on collisions easily.

This behavior could probably be explained by a combination of an insufficient amount of information that is provided to the robot and FLS-rules that are sub optimal. The rules might make the robot ill-equipped to recover from a collision (currently the robot turns left as much as possible when encountering very nearby objects). However, the information provided to the robot might be the biggest problem. The robot currently only sees objects within a 170 degree window in front of it. As soon as an object leaves that window, the robot will act as if it does not exists. This could cause it to get stuck on corners of an object, or make it repeatedly rotate back and forth between seeing the object and not seeing the object (for example when another object is also close at its other side).

A third possibility is that we are dealing with an implementation issue. Mainly that the robot takes its rotation into account when calculating its distance from other objects. However, when checking for a collision, we only use squares parallel to the grid axis, since pygame only supports collision-checking in that way. This could cause inconsistencies between the robots perceptions and the simulated reality.

The final way this could be explained is that there are to many places in the simulation where the robot is not able to move through. Because the FLS has no way of telling the robot to move backwards it could get stuck due to limitations in the possible maneuvers of the robot.

The second problem with our FLS that we can extrapolate from our results is that, when the speed of the robot increases, the duration of the simulations decreases (the robot gets stuck very early on). This can be explained by the fact that the mfs use a distance in pixels to calculate the closeness of objects. The membership functions don't change based on the speed. The FLS also does not have speed as one of its inputs. This means that the FLS has no way to correct for speed. When the robot moves at a higher speed the something that was 'far' for a lower speed might now be reached in one frame. In data that was not added to this paper it can be seen that when the robot moves at a very slow speed, 1 or 2 pixels per tick, it is far more successful in avoiding the obstacles.

VI. Conclusions and Future Work

Although the FLS we created is limited in it's methods to avoid obstacles, it shows that there are possibility's for a fuzzy logic system to help in this area. Most of the ways that where concluded to be reasons of failure where design choices that limited the outcome. An example of this is the limited directions the robot could move is, it was not allowed to move backwards. But when looking at the movement from in the simulation it can be clearly seen that the robot has a basic idea of what to do to avoid the obstacles.

For future work the major issues that need to be addressed the fact that the FLS does not use the speed of the robot and the limited directions the robot can move in. After this the next step would be to put the robot in an environment with moving obstacles, when doing this most of the input variable need to be altered to allow backwards vision. When a FLS is created that is successful in avoiding moving obstacles it should be compared to other approaches to see if there are benefits to using a FLS over a different method.

References

[1] X. Bajrami, A. Dërmaku, and N. Demaku, "Artificial neural fuzzy logic algorithm for robot path finding," *IFAC-PapersOnLine*, vol. 48, no. 24, pp. 123–127, 2015.

[2] R. Cabrera-Cosetl, M. Z. Mora-Alvarez, and R. Alejos-Palomares, "Self-parking system based in a fuzzy logic approach," in *2009 International Conference on Electrical, Communications, and Computers*, Feb 2009, pp. 119–124.

[3] A. P. Gerdelan and N. H. Reyes, "A novel hybrid fuzzy a* robot navigation system for target pursuit and obstacle avoidance," in *Proceedings of the First Korean-New Zealand Joint Workshop on Advance of Computational Intelligence Methods and Applications*, vol. 1, 2006, pp. 75–79.

[4] M. Mohammadian and R. J. Stonier, "Adaptive two layer fuzzy control of a mobile robot system," in *Evolutionary Computation, 1995., IEEE International Conference on*, vol. 1. IEEE, 1995, p. 204.

[5] C. von Altrock, B. Krause, and H. J. Zimmermann, "Advanced fuzzy logic control technologies in automotive applications," in *[1992 Proceedings] IEEE International Conference on Fuzzy Systems*, Mar 1992, pp. 835–842.

VII. Workload Distribution

Jelle van Dijk: 10989048

The goal was that every person in the team helped on each part of the project and more importantly understood every part of the project. Although I did not work evenly on all parts I did contribute every where and am pretty confidant that I know how every part of the project works. Most of my work can be found in the simulation and the .fis file parser, which was not discussed in the paper. The .fis file parser I made to have a way to store our FLS, it is a Python script that can read and parse a .fis file, the same files that are generated by Matlab's fuzzy logic toolbox, and create an instance of our FLS with that. Most of my work can be found in the simulation, I created the foundation where the others worked on.

I learned a lot from this project, allot of the theory that was given to us in the classes was turned into something that work during this project. Most of all I am happy that we got a working system, although it is not perfect it is a good attempt.