ASSESSMENT REPORT

MAT301

FRASER BARKER
1600196
1600196@uad.ac.uk

Contents

Introduction

From the simple enemies in Space Invaders to the hierarchical behemoths as seen in Alien: Isolation, AI in games have rapidly progressed to more complex systems to help the player become more engrossed in the game they are playing.

From finite state machines and rule based systems to genetic algorithms and artificial neural networks there's an almost gargantuan range of techniques that could be used to implement AI in video games. It all depends on the complexity of the application and what the AI should do within that application.

For this application, the two AI techniques chosen for comparison were Finite State Machine and Fuzzy Logic State Machine. Due to their simplistic nature, and the simplicity of the application, they seemed like the best fit.

Application

The application uses a finite state machine to control the red car, and a fuzzy logic state machine (utilising a fuzzy inference system) to control the green car.

The graphical representation of the application is written in C++ using SFML 2.5.1 and uses a window size of 640 by 480 pixels.

On the left side of the screen there is a debug menu which can be maximised/minimised and has been implemented via the use of ImGui and ImGui-SFML to allow the user to see the values calculated for each car in real time and also allow the user to alter how fast each car can move via changing the "Speed Modifier" value.

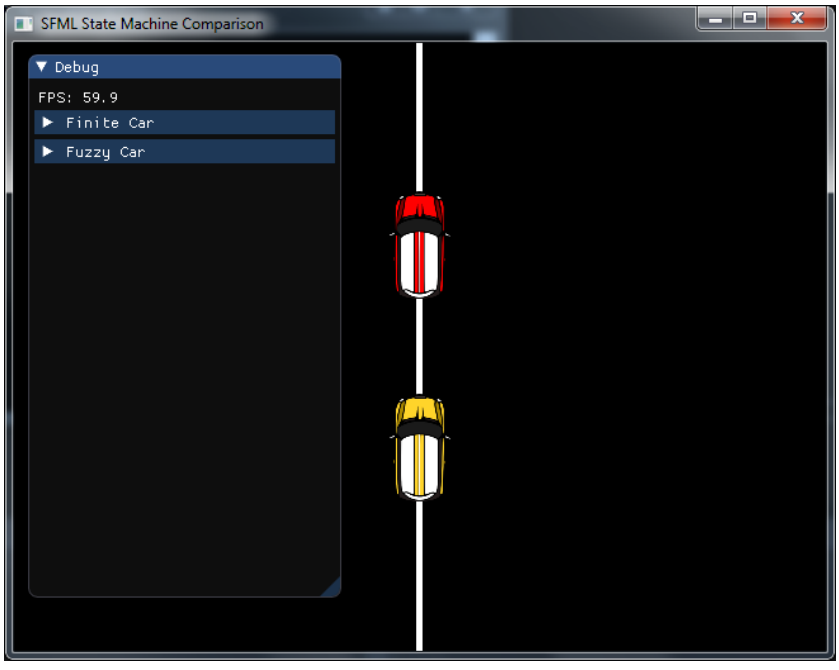


Figure 1: Application

The ImGui debug menu has two sections – one for each car – that will show the current velocity of the car and the distance it is away from the racing line. The Fuzzy Car has an additional section showing what direction has been calculated via the fuzzy inference system.

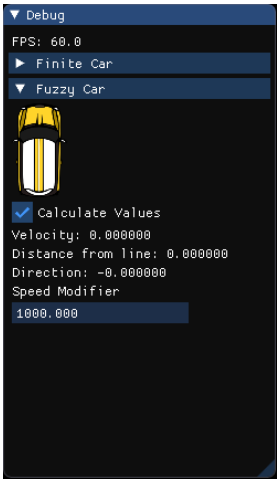


Figure 2: Fuzzy Car Debug Values

The Fuzzy Car specifically can have its distance and velocity set by the user if they untick the "Calculate Values" box.

Figure 3: Manual Input Values

This will then allow the user to input values for the distance from the racing line, and velocity of the car which will result in the output of a different value for direction. Such input will result in values similar to what is documented in the Test Data section.

Controls

The user can move the racing line left/right using the A/Left arrow and D/Right arrow keys respectively.

AI Techniques

Finite State Machine

A finite state machine was chosen as the first AI technique as it was easy to implement when considering the application would be moving a car sprite left/right depending on its distance away from a racing line and the speed it was moving at.

The ability for extensions also helps in that it could be refined further using a hierarchical approach to allow it to have other priorities alongside moving back towards the racing line such as obstacle avoidance.

Fuzzy Logic State Machine

A fuzzy logic state machine was used to compare against the finite state machine due to the concept of "fuzziness" where the application is given some error in order to make it react more human-like. This builds upon the finite state machine approach where it causes the AI to be more error prone and less robotic in nature which would mean players would believe it acts close to another human than a pre-programmed array of options.

The fuzzy inference system was generated in MATLAB. This was then imported into the project using the FuzzyLite library which calculates all the necessary variables required for the system to operate. The result moves the car left/right depending on its distance from the racing line and its speed.

Reasoning

As both techniques are state machines, the only differing factor is how the input data is used to calculate what direction the car should move in.

This made them an ideal comparison as they both function similarly with the exception that the fuzzy logic state machine uses a fuzzy ruleset to mimic a more human-like behaviour.

Computational Efficiency

Finite State Machine

The finite state machine is largely made up of floats and an enum to depict what state the car is in. Aside from SFML variables to render the sprite it's pretty simplistic. Generally, it will be faster than the fuzzy logic state machine as it isn't dependant on external libraries to calculate variables and just processes through a switch statement to decide which state to go to next.

Fuzzy Logic State Machine

Similar to the finite state machine except the direction of the fuzzy car is dependent on its direction being calculated via FuzzyLite. Utilising a fuzzy ruleset this allows the fuzzy logic state machine to display more human-like behaviour.

Ease of Coding

Both systems were easy to set up. The trickiest part was getting the FuzzyLite library into the project which required building files via CMake. After that, as both systems have a finite state machine base, only the fuzzy logic state machine had to be altered in order to make it follow the fuzzy logic ruleset and have its data calculated via FuzzyLite.

Design

Based on the cars distance away from the racing line, and its current velocity, the fuzzy inference system outputs what direction the car should move in.

Regarding distance, the car is either negative (left of the racing line), centre (roughly on the racing line) or positive (right of the racing line).

Regarding velocity, the car is either has a negative velocity (moving left), centre velocity (close to the racing line) or has a positive velocity (moving right).

Inputs		Velocity		
		Negative	Centre	Positive
Distance	Negative	Far Left	Left	Centre
	Centre	Left	Centre	Right
	Positive	Centre	Right	Far Right

As shown in the table, the fuzzy inference systems output was given five membership function in order to produce a more natural response to the stimuli. The outputs are as follows;

1. Far Left
2. Left
3. Centre
4. Right
5. Far Right

The fuzzy ruleset generated from these membership functions are:

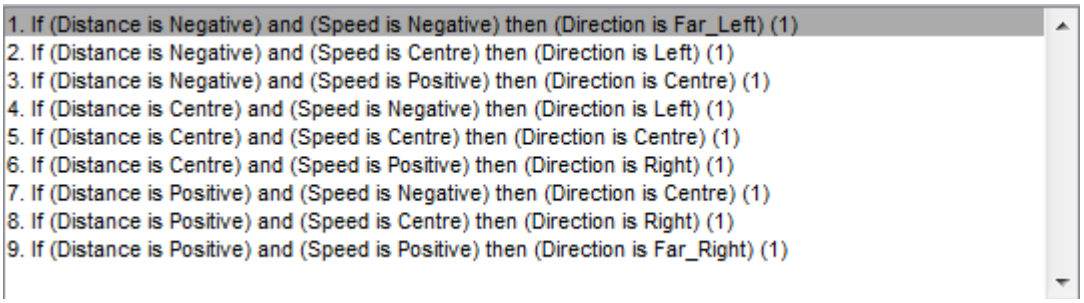


Figure 4: Final Fuzzy Ruleset

The method chosen for defuzzification of the fuzzy inference system was centroid (centre of gravity) of the output. This method, while it may be more computationally complex to calculate the output value compared to other methods, the simplicity of the application means that these costs are easily met.

The universe of discourse for the inputs and output were determined based on wanting to have a normalised input and output.

For distance, the universe of discourse works for the screen's width in either direction of the racing line.

For velocity, the universe of discourse is the maximum speed the car can reach. This was chosen carefully that the car can respond quickly enough to stimuli but not so fast as to nullify the purpose of the AI appearing human-like.

Both the distance and velocity are normalized before being given to the FuzzyLite engine to ensure they match the range as depicted in the input and output graphs of the fuzzy inference system.

The range for all was set from -1 to 1 in order to normalize the returned crisp output to allow it to be easily scaled if required.

This design was implemented in MATLAB and refined over two iterations.

Method

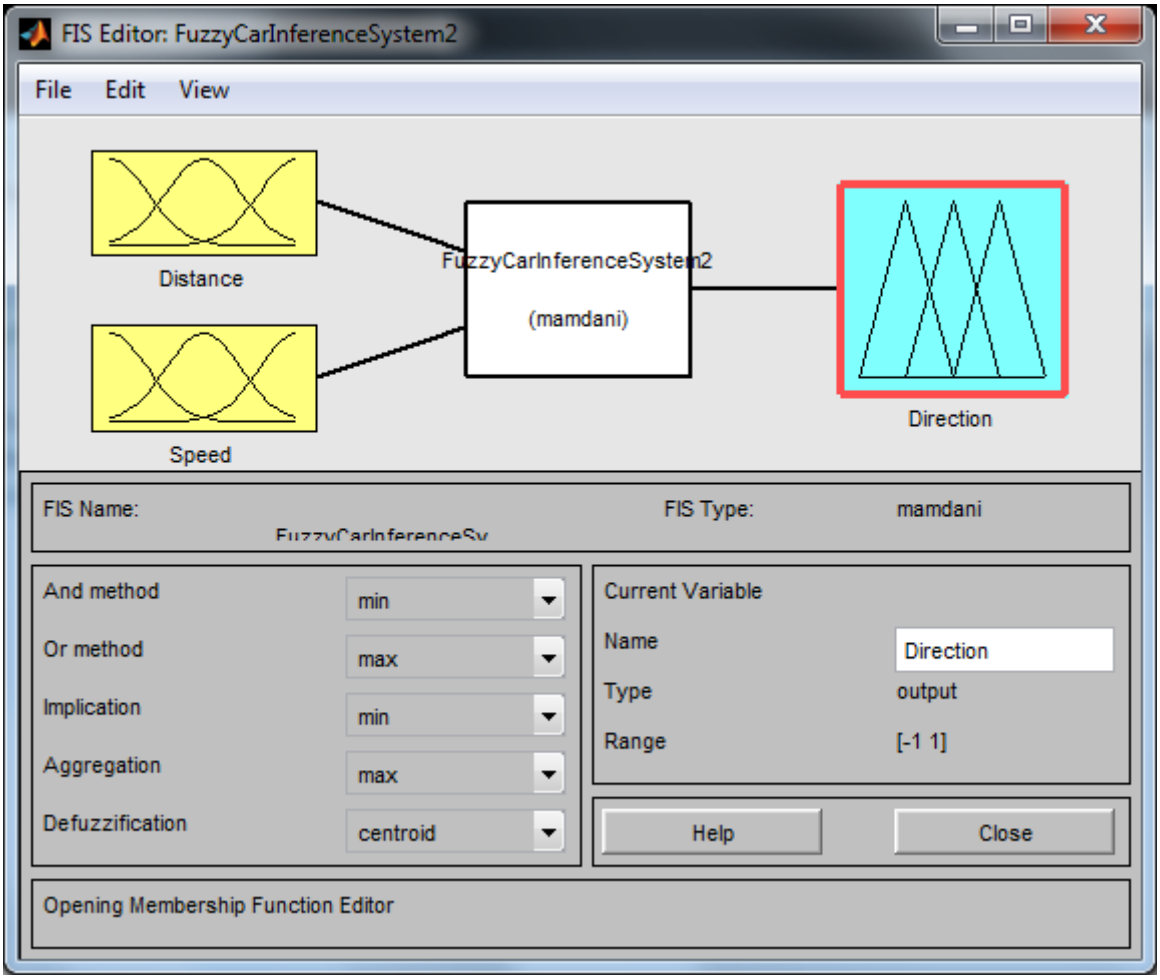


Figure 5: MATLAB Methodology

For the fuzzy inference system, two inputs (distance and speed) were specifically chosen for the application.

Distance from the racing line was a pivotal choice as it is the major component when calculating what direction the car will move in.

Speed was the secondary factor to input into the system is it would be used to determine how fast the car should be moving based on the distance it was away from the racing line. E.g. the car would be expected to slow down once nearing the racing line (as would be the case in human-based testing) otherwise it would just seem illogical.

Direction is the output of the fuzzy inference system and it is used to make the car move in a certain direction.

First Iteration

Input

Both systems take the distance from the racing line and their current velocity as inputs to their systems. The fuzzy logic state machine takes this a step further as it utilises the FuzzyLite library to calculate its direction via the centroid of the two input graphs.

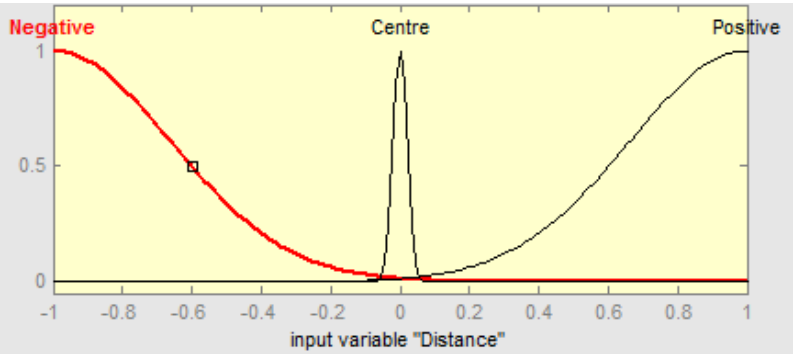


Figure 6: First Iteration Distance Input

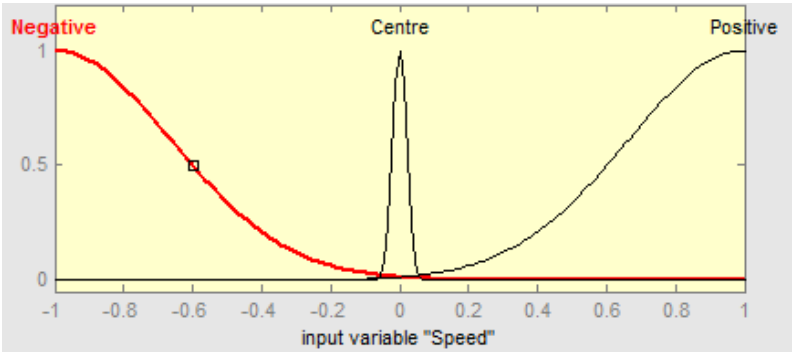


Figure 7: First Iteration Speed Input

Initially the input graphs for distance and speed were identical. But such a sharp curve when coming to the centre cause the fuzzy car to behave rather erratically.

Output

The output for the fuzzy state machine is direction which is determined via the given inputs.

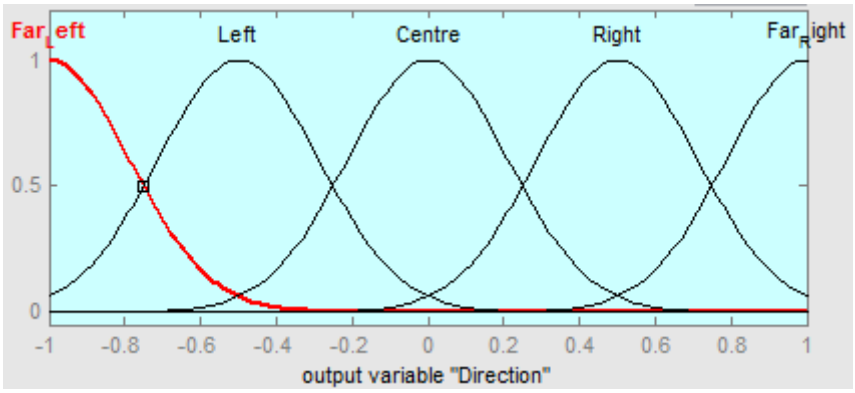


Figure 8: First Iteration Direction Output

The equal curves for each of the directions caused the car to be rather uniform when manoeuvring towards the racing line.

Surface

The surface generated in MATLAB is helpful in identifying problematic spots when it comes to assessing how the fuzzy state machine will perform.

Due to an incorrect ruleset in the first iteration, the surface generated was not equal when Distance was equal to 0.

1. If (Distance is Negative) and (Speed is Negative) then (Direction is Far_Left) (1)
2. If (Distance is Negative) and (Speed is Centre) then (Direction is Left) (1)
3. If (Distance is Centre) and (Speed is Negative) then (Direction is Left) (1)
4. If (Distance is Centre) and (Speed is Centre) then (Direction is Centre) (1)
5. If (Distance is Centre) and (Speed is Positive) then (Direction is Right) (1)
6. If (Distance is Positive) and (Speed is Centre) then (Direction is Right) (1)
7. If (Distance is Positive) and (Speed is Positive) then (Direction is Far_Right) (1)
8. If (Distance is Negative) and (Speed is Positive) then (Direction is Left) (1)
9. If (Distance is Positive) and (Speed is Negative) then (Direction is Right) (1)

Figure 9: First Iteration Ruleset

This means the car would always be veering to the left/right when it was at the centre which was incorrect.

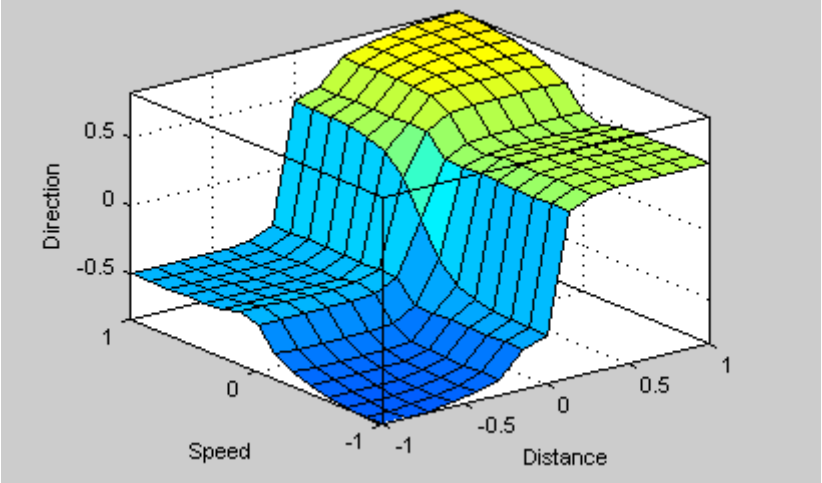


Figure 10: First Iteration Surface

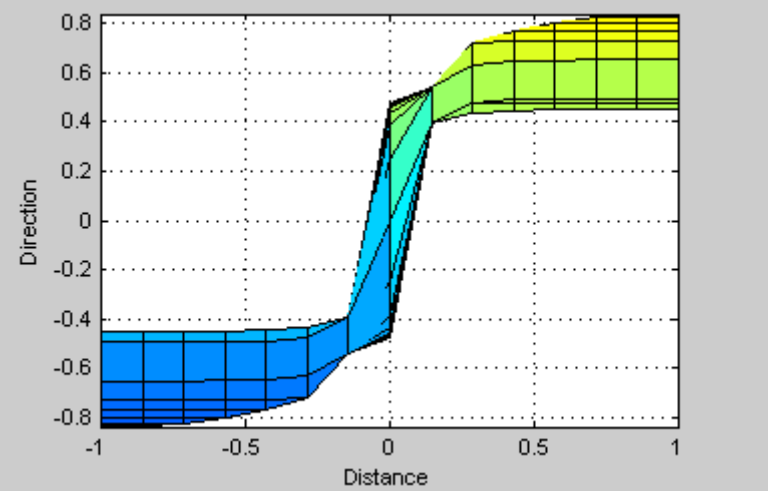


Figure 11: First Iteration Surface X-Z

The surface output was also quite rigid in places and this was reflected in the fuzzy cars movement within the application.

Second Iteration

Input

Using the same inputs as the first iteration, it was decided that the centre point should be widened to allow more flexibility within the input for the system.

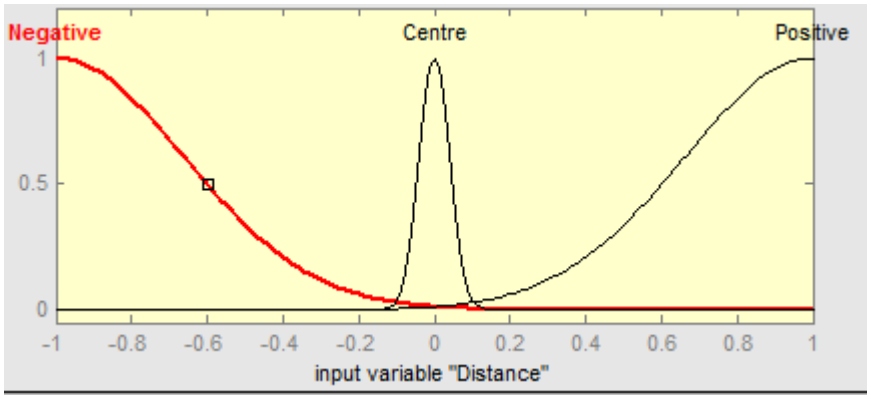


Figure 12: Second Iteration Distance Input

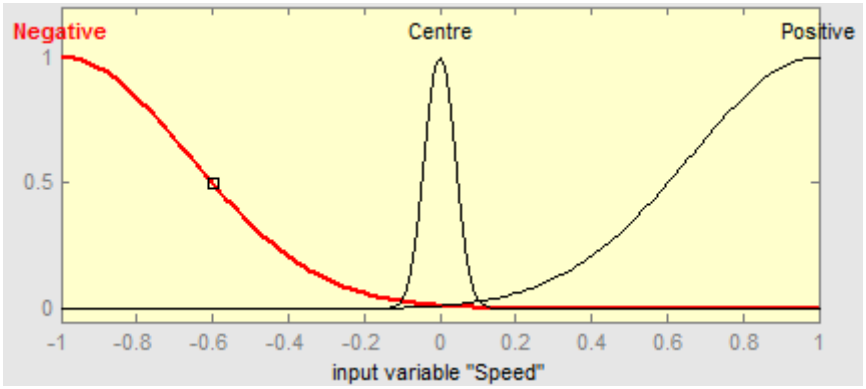


Figure 13: Second Iteration Speed Input

Output

The output was also altered accordingly due to the errors that cropped up during the testing of the first iteration.

With a slightly narrower curve in the centre and larger curves for the left/right directions this caused the fuzzy car to behave as expected.

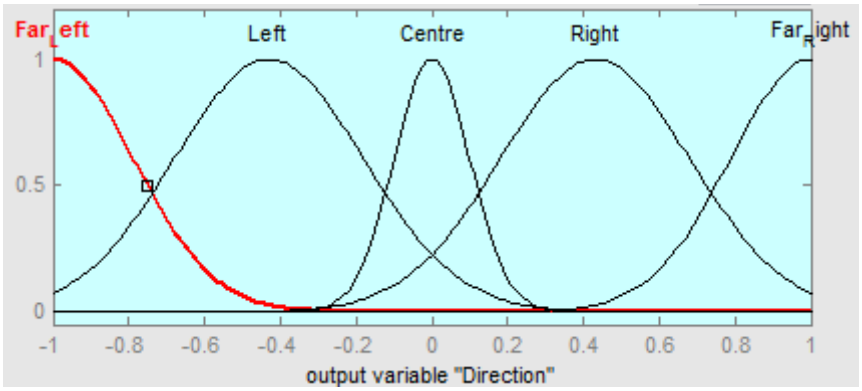


Figure 14: Second Iteration Direction Output

Surface

Due to a change in the ruleset, the surface generated was also different (albeit expected) due to correcting the error in the fuzzy logic ruleset.

1. If (Distance is Negative) and (Speed is Negative) then (Direction is Far_Left) (1)
2. If (Distance is Negative) and (Speed is Centre) then (Direction is Left) (1)
3. If (Distance is Negative) and (Speed is Positive) then (Direction is Centre) (1)
4. If (Distance is Centre) and (Speed is Negative) then (Direction is Left) (1)
5. If (Distance is Centre) and (Speed is Centre) then (Direction is Centre) (1)
6. If (Distance is Centre) and (Speed is Positive) then (Direction is Right) (1)
7. If (Distance is Positive) and (Speed is Negative) then (Direction is Centre) (1)
8. If (Distance is Positive) and (Speed is Centre) then (Direction is Right) (1)
9. If (Distance is Positive) and (Speed is Positive) then (Direction is Far_Right) (1)

Figure 15: Second Iteration Ruleset

This produced even plateaus for the centre point in both distance and speed at Direction = 0. It also smoothed the surface more which allowed the fuzzy car to manoeuvre in a more human-like fashion.

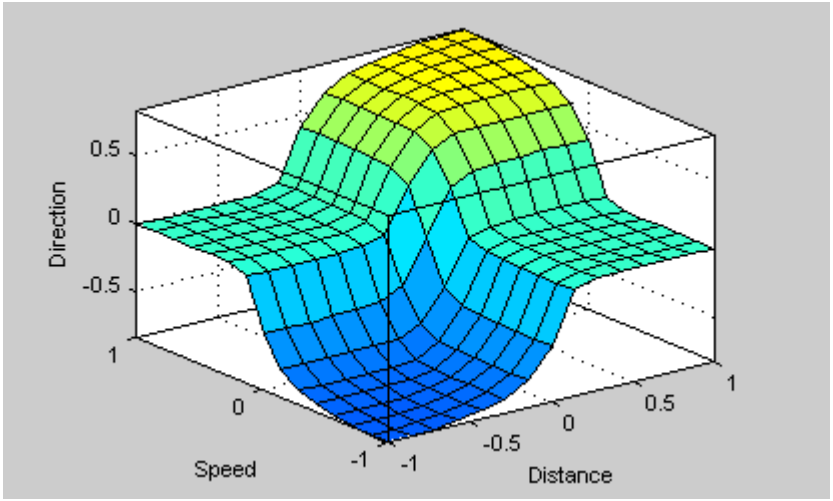


Figure 16: Second Iteration Surface

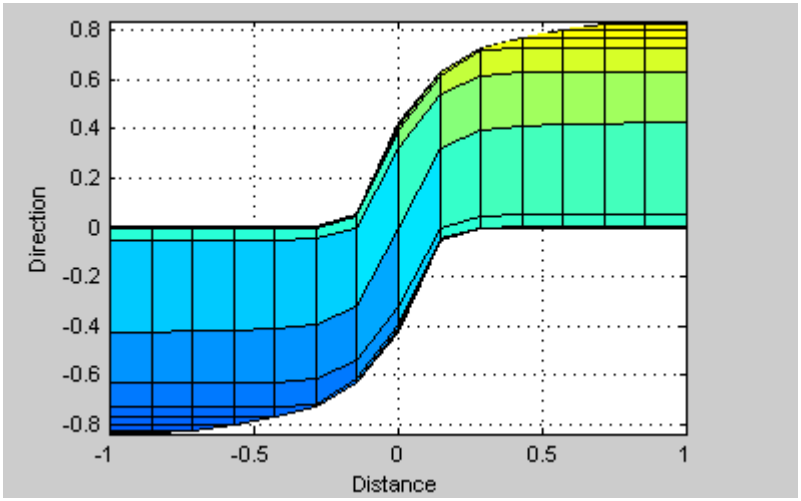


Figure 17: Second Iteration Surface X-Z

Test Data

Fuzzy Logic State Machine

For the fuzzy system, its output was calculated both in MATLAB and the application itself. One variable was altered whilst the other variable was set to a constant value of 0.

MATLAB			
Velocity = 0		**Distance = 0**	
Distance	**Output**	**Velocity**	**Output**
-1	-0.423	-1	-0.423
-0.75	-0.422	-0.75	-0.422
-0.5	-0.417	-0.5	-0.417
-0.25	-0.385	-0.25	-0.385
-0.1	-0.143	-0.1	-0.143
0	$8.01e^{-18}$	0	$8.01e^{-18}$
0.1	0.143	0.1	0.143
0.25	0.385	0.25	0.385
0.5	0.417	0.5	0.417
0.75	0.422	0.75	0.422
1	0.423	1	0.423

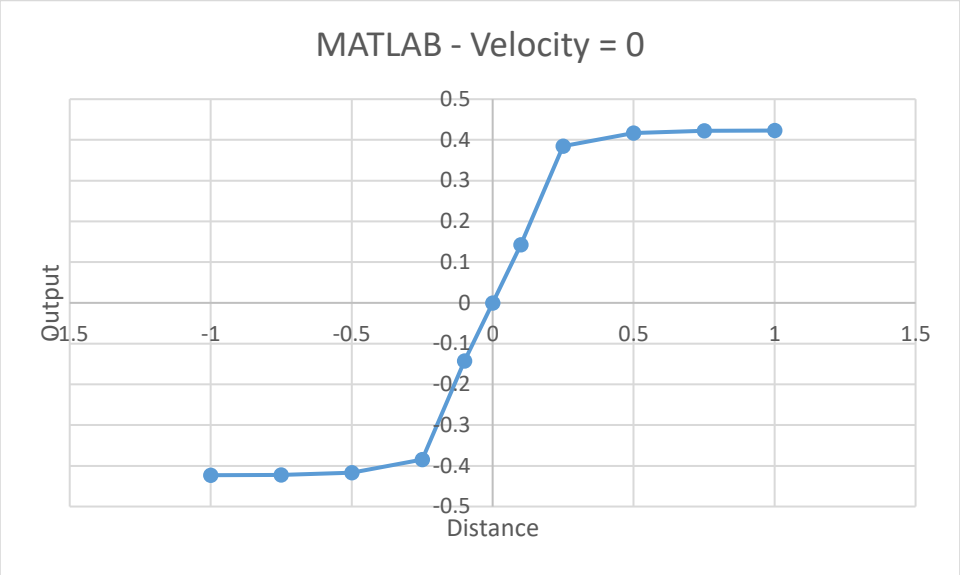
Application			
Velocity = 0		**Distance = 0**	
Distance	**Output**	**Velocity**	**Output**
-1	-0.422	-1	-0.422
-0.75	-0.421	-0.75	-0.421
-0.5	-0.415	-0.5	-0.415
-0.25	-0.380	-0.25	-0.380
-0.1	-0.140	-0.1	-0.140
0	$-3.231e^{-18}$	0	$-3.231e^{-18}$
0.1	0.140	0.1	0.186
0.25	0.380	0.25	0.380
0.5	0.415	0.5	0.415
0.75	0.421	0.75	0.421
1	0.422	1	0.422

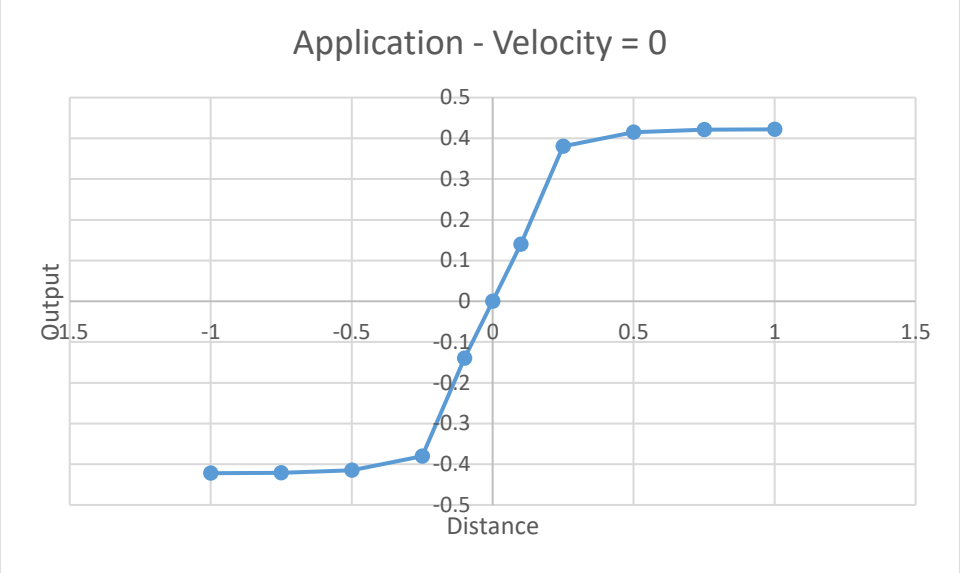
Whilst almost identical there is error of ~0.001 – 0.005 between the application and the MATLAB calculations. This error can be accounted for due the difference in centroid calculation methods between MATLAB and the FuzzyLite system, float point numbers vs doubles, rounding errors and so on.

These tables have also been mapped to graphs using Microsoft Excel's scatter graph plotter and are shown below.

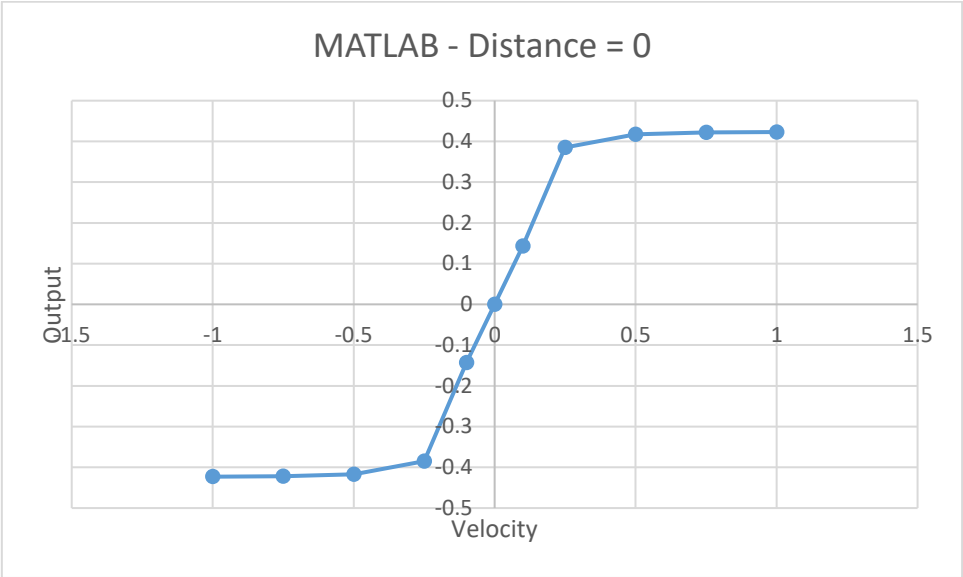
Graphs

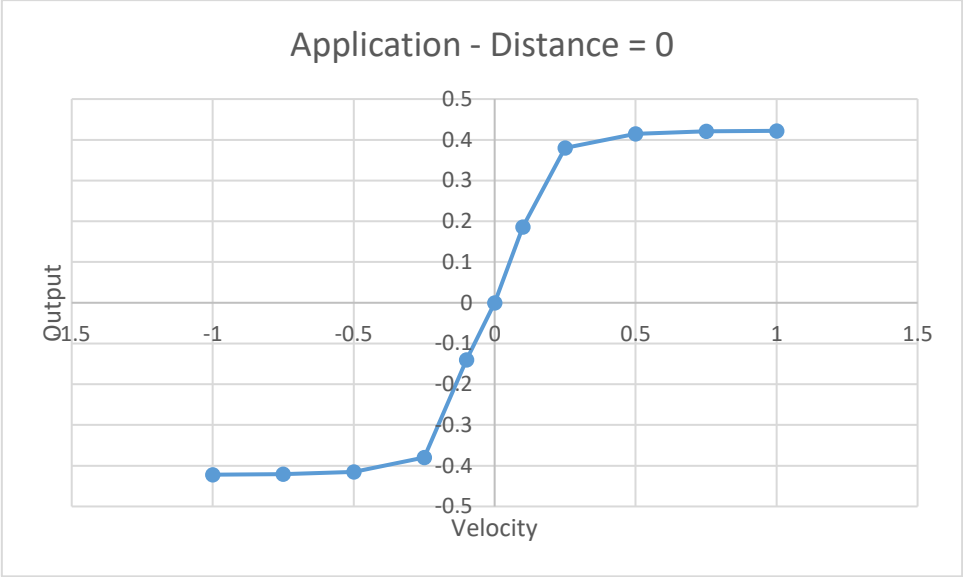
Constant Velocity





Constant Distance





As you can see, the application has a bit more of a curve to both graphs whereas MATLAB is quite rigid in nature.

Finite vs Fuzzy Logic State Machines

Aside from timing how long each systems takes to finish its update function, the only other perceivable way to see how each system is performing is to look at which car gets back to the racing line quicker and the sway it has during this process.

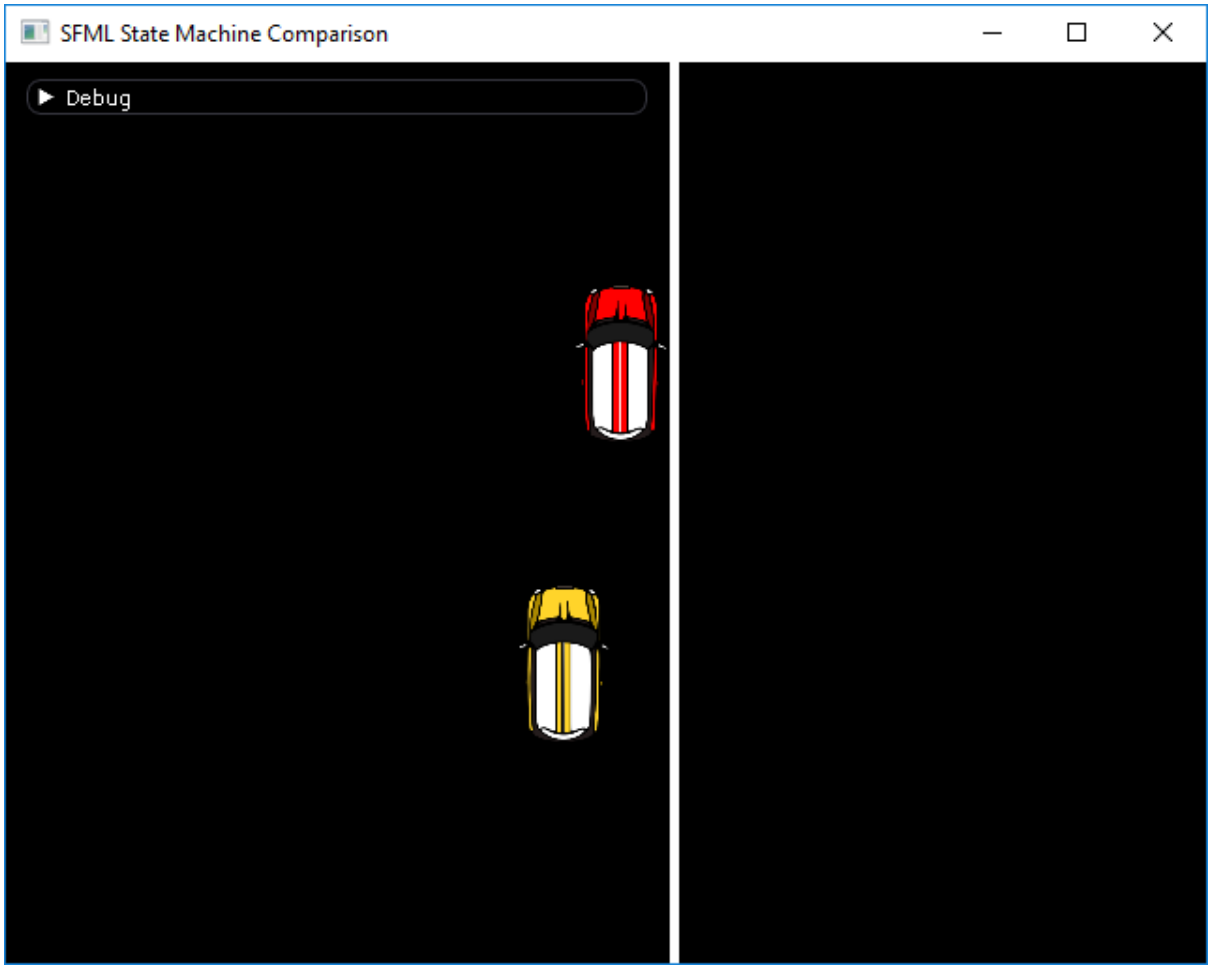
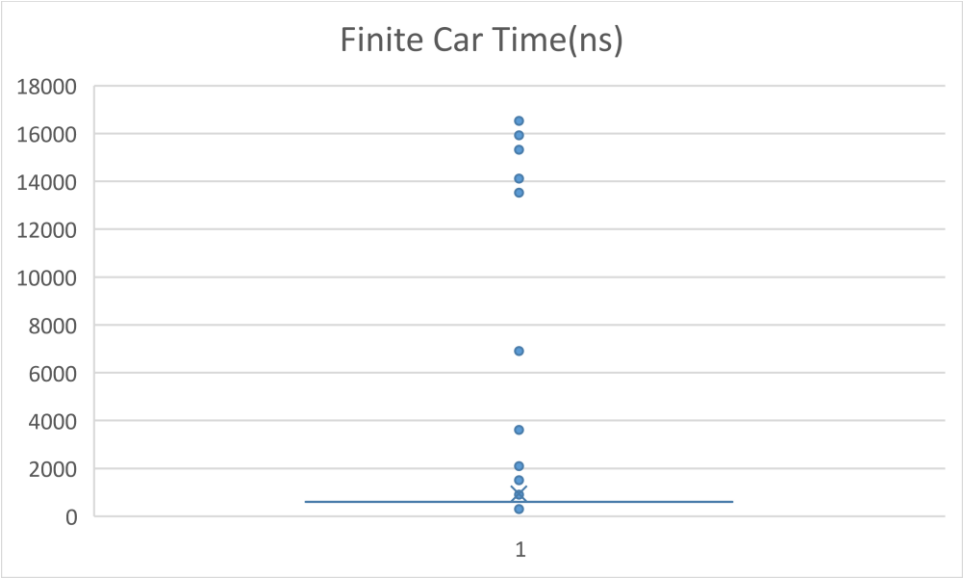
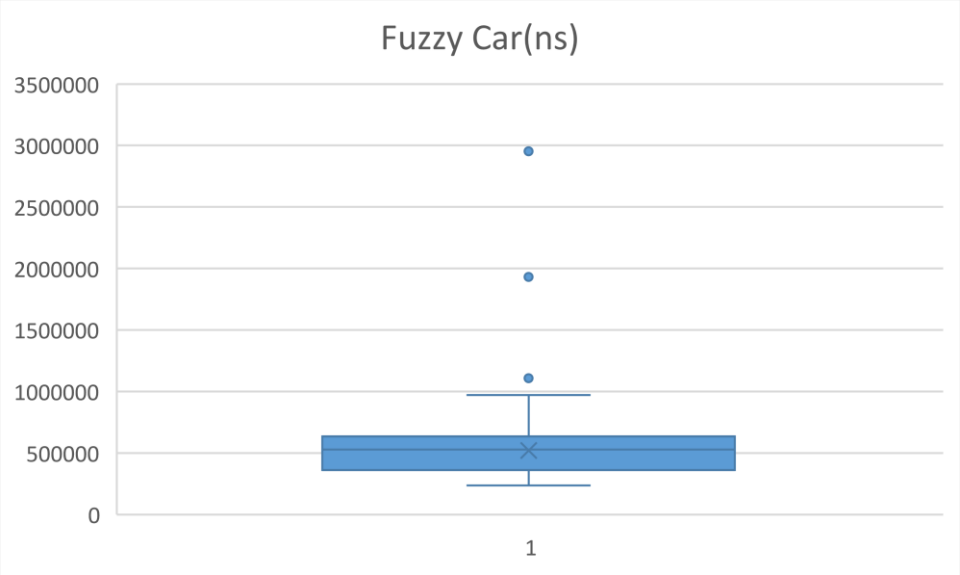


Figure 18: Application Car Sway

Details of timing the two systems can be found below.



Average time (ns): **942.4471**



Average time (ns): **522261.1**

Whilst not as detailed as the MATLAB vs. Application calculation of the fuzzy inference system, it can be seen quite clearly that the finite state machine is ~554 times faster than the fuzzy logic state machine. (Bear in mind, this is the time taken in nanoseconds for both systems to finish their update function). This would be due to the fact that the finite state machine is much simpler in nature and does not need to calculate variables using the centroid of graphs as the fuzzy state machine must.

Results & Conclusions

Overall, the fuzzy logic state machine seems to perform as expected. Whilst a bit slower this can be altered by the user to make the two systems react almost as quickly.

As can be seen in the test data, there is difference of up to ~0.005 between the results in MATLAB and the applications calculations using FuzzyLite. While the fuzzy inference system performs as intended, the application does tend to lose out on some accuracy.

Choosing Gaussian curves in comparison to triangles or trapezoids for the membership function graphs allowed the system to be more fluid in nature and display less rigid behaviour.

Extensions to both systems could be made such as including obstacle avoidance and altering what priority it would take over following the racing line.

Other extensions could include managing a group of racing cars (like that which is used in the Boid flocking simulation) which would essentially be an improved version of the obstacle avoidance alteration in that the obstacles would now move, and the system would have to keep a set distance whilst avoiding obstacles.

References

Taito. 1978. Space Invaders. [ONLINE] Available at:

https://en.wikipedia.org/wiki/Space_Invaders [Accessed 17/04/2019]

Sega. 2014. Alien: Isolation. [ONLINE] Available at:

https://www.alienisolation.com/en [Accessed 17/04/2019]

Unknown. 2010. A Short Fuzzy Logic Tutorial. [ONLINE] Available at:

http://cs.bilkent.edu.tr/~zeynep/files/short_fuzzy_logic_tutorial.pdf [Accessed 17/04/2019]

Juan Rada-Vilela. 2017. FuzzyLite. [ONLINE] Available at:

https://fuzzylite.com/ [Accessed 17/04/2019]

Ocornut. 2018. ImGui-SFML. [ONLINE] Available at:

https://github.com/eliasdaler/imgui-sfml [Accessed 17/04/2019]

Ocornut. 2019. ImGui. [ONLINE] Available at:

https://github.com/ocornut/imgui [Accessed 17/04/2019]

Bahi. 2011. Cartoon Car- Mini. [ONLINE] Available at:

https://opengameart.org/content/cartoon-car-mini [Accessed 17/04/2019]

Laurent Gomila. 2019. SFML. [ONLINE] Available at:

https://www.sfml-dev.org/learn.php [Accessed 17/04/2019]

MathWorks. 2019. MATLAB. [ONLINE] Available at:

https://uk.mathworks.com/products/matlab.html [Accessed 17/04/2019]

KitWare. 2019. CMake. [ONLINE] Available at:

https://cmake.org/ [Accessed 17/04/2019]

Craig Reynolds. 1986. Boids. [ONLINE] Available at:

https://en.wikipedia.org/wiki/Boids [Accessed 17/04/2019]