FRASER BARKER

1600196

1600196@uad.ac.uk

ASSESSMENT REPORT

MAT301

Contents

[Introduction 2](#_Toc6566794)

[Application 2](#_Toc6566795)

[Controls 3](#_Toc6566796)

[AI Techniques 3](#_Toc6566797)

[Finite State Machine 3](#_Toc6566798)

[Fuzzy Logic State Machine 3](#_Toc6566799)

[Reasoning 4](#_Toc6566800)

[Computational Efficiency 4](#_Toc6566801)

[Finite State Machine 4](#_Toc6566802)

[Fuzzy Logic State Machine 4](#_Toc6566803)

[Ease of Coding 4](#_Toc6566804)

[Design 4](#_Toc6566805)

[Method 5](#_Toc6566806)

[Input 5](#_Toc6566807)

[Output 5](#_Toc6566808)

[Test Data 7](#_Toc6566809)

[Timings 7](#_Toc6566810)

[Graphs 9](#_Toc6566811)

[Constant Velocity 9](#_Toc6566812)

[Constant Distance 10](#_Toc6566813)

[Results & Conclusions 11](#_Toc6566814)

[References 12](#_Toc6566815)

# Introduction

Talk about AI in games in general to put the application into perspective.

# Application

The application uses a finite state machine to control the red car, and a fuzzy 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.

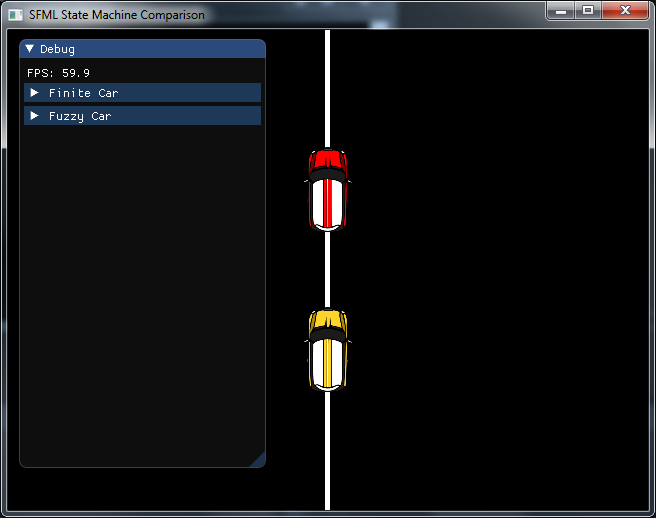


Figure : Application

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 driving line. The Fuzzy Car has an additional section showing what direction has been calculated via the fuzzy inference system.

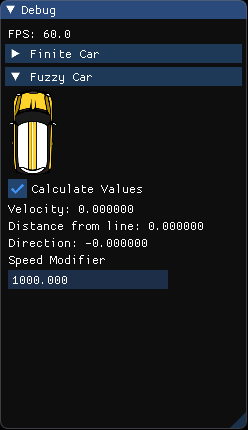


Figure : Fuzzy Car Debug Values

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



Figure : 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 driving 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 driving 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).

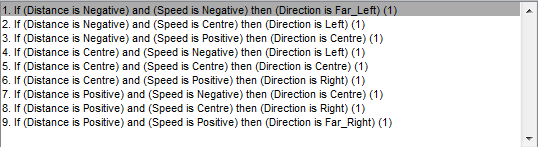
Regarding velocity, the car is either has a negative velocity (moving left), centre velocity (close to the racing line) or has a positive velocity (moving right).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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:

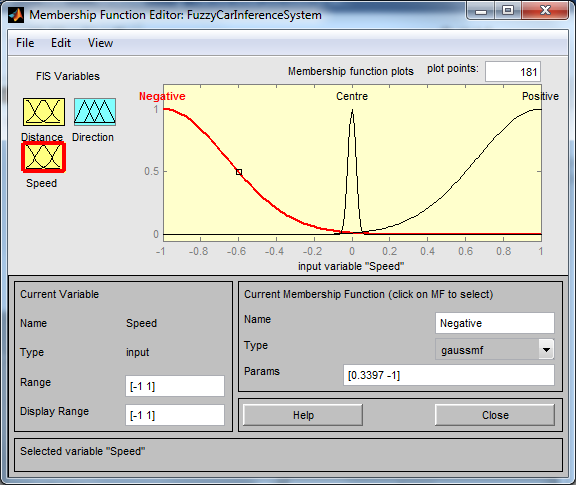
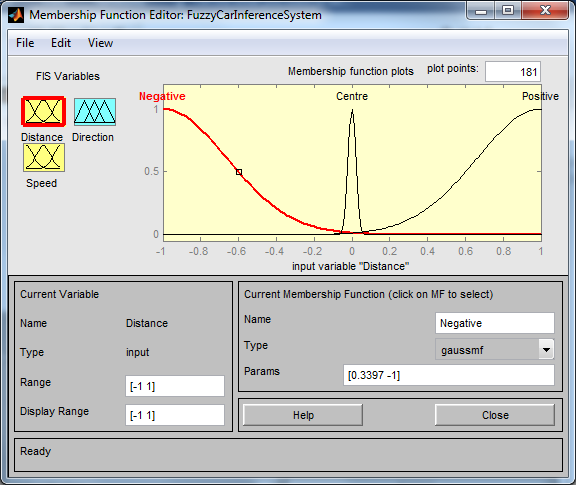


Universe of discourse.

# Method

## Input

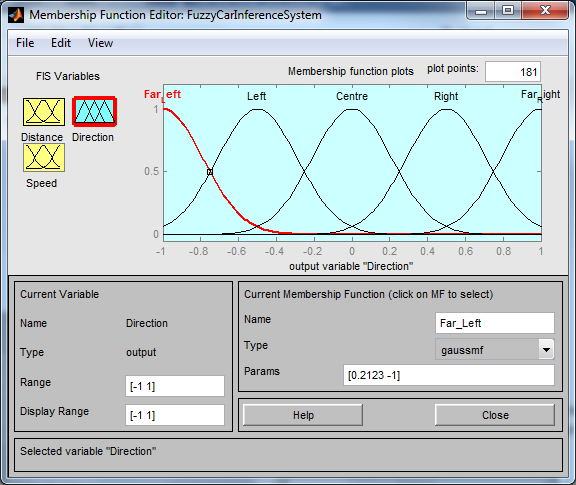
Both systems take the distance from the driving line and their current velocity as inputs to their systems. The fuzzy logic state machine takes this a step further as it utilises the FuzzyLite library to calculate its direction via the centroid of the two input graphs.



## Output

The output for the finite state machine is acceleration which is determined via what state the car is in.

The output for the fuzzy logic state machine is direction which is calculated via centroid of the two input graphs distance and speed at their respective points of distance and velocity input.



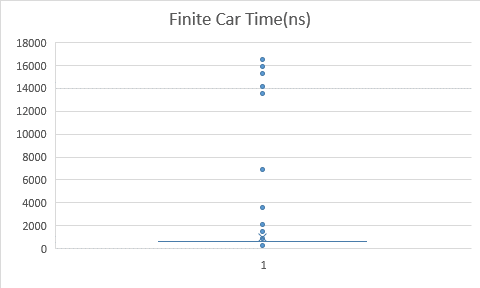
# Test Data

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

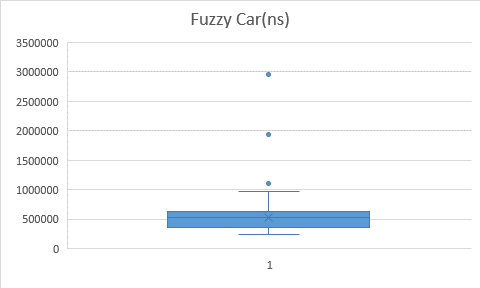
|  |  |  |  |
| --- | --- | --- | --- |
| **Application** | | | |
| **Velocity = 0** | | **Distance = 0** | |
| **Distance** | **Output** | **Velocity** | **Output** |
| -1 | -0.422 | -1 | -0.422 |
| -0.75 | -0.421 | -0.75 | -0.421 |
| -0.5 | -0.415 | -0.5 | -0.415 |
| -0.25 | -0.380 | -0.25 | -0.380 |
| -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 |

Talk about discrepancy.

## Timings



Average time (ns): **942.4471**



Average time (ns): **522261.1**

# Graphs

## Constant Velocity

## Constant Distance

Talk about discrepancy again.

# Results & Conclusions

# References

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]