Introduction

The Application (henceforth referred to as “the App”) uses a Fuzzy Inference System (FIS) to steer a Car towards a moveable Racing Line.

Rapid advancements in Machine Learning techniques such as Artificial Neural Networks have brought Artificial Intelligence to the forefront of much academic, personal and political discussion. The abilities of computers to make complex and accurate decisions across various fields, and the far-reaching socioeconomic implications of this are years ahead of previous projections. When it comes to AI that is specific for games applications, there is still a wide variety of techniques available set apart from other popular techniques by two key factors;

Firstly, the current majority of game-related AI applications are expected to run in real-time, ruling out many of the more computationally expensive techniques available.

Secondly, it is important to bear in mind that games AI is not necessarily required to perform a given task as quickly and as accurately as possible. The primary purpose of every element of a video game should be to provide the player with an enjoyable and engaging experience. As such it is often better to create AI that appears human and believable by providing players with a reasonable challenge and often by feeling human rather than performing a task “perfectly.”

With that second point in mind, the App aims to emulate a natural, “imperfect” human reaction to stimuli on a straight stack, such as a motorway. The car gently corrects for small changes in course, and responds to more drastic changes by steering sharply, oversteering, and then having to correct for that oversteer before settling back into its course.

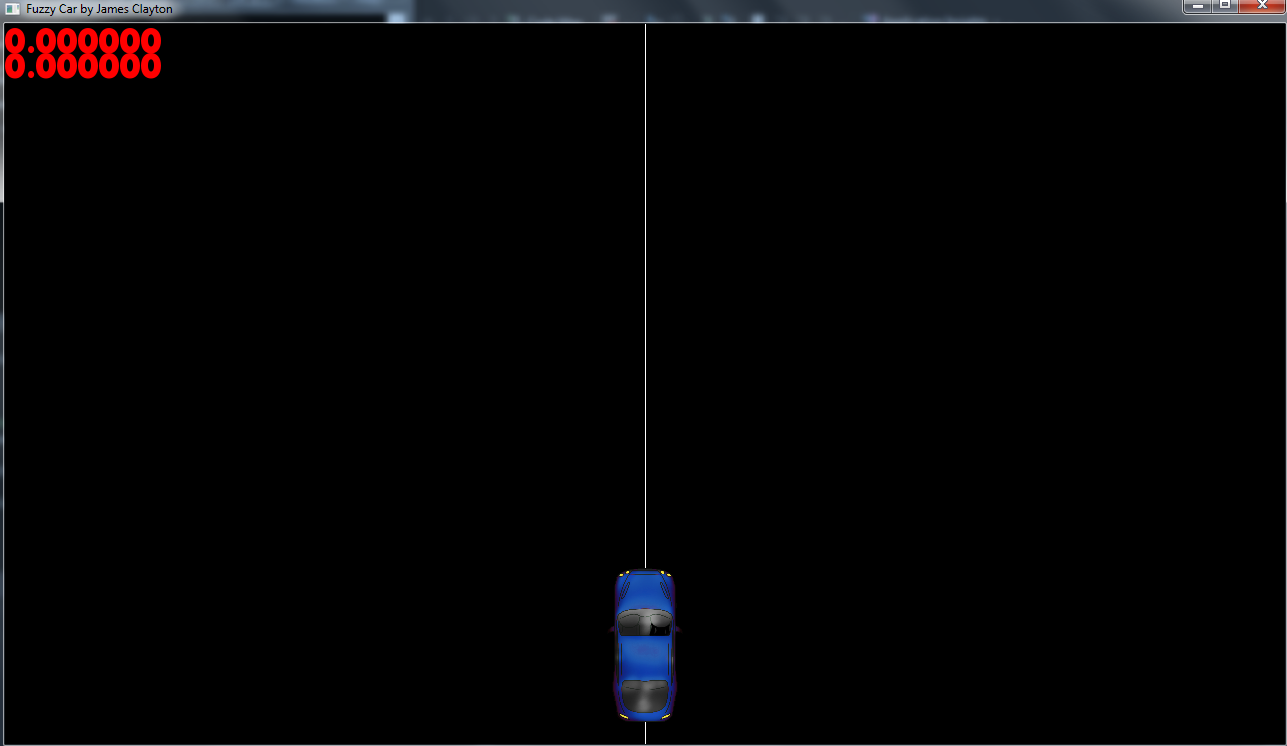
The key principle of a Fuzzy Inference System is in its “fuzziness” mimicking human perception and analysis of the world. Where computers are innately rigid in their logic and reasoning as a result of clearly defined, “crisp” data, Fuzzy Logic allows for vaguer, human-like perception and decision making by classifying data into less strictly defined sets and making decisions accordingly. There are definite limits to computerised Fuzzy Logic, as all data and computation is ultimately rigid and binary, but a FIS allows for an acceptable approximation to this with relatively low computational costs.

The Application

When started, the application presents the user with the option of running the application in one of two states;

1. Test Mode, where the user can enter arbitrary values within a specified range and the application will display the corresponding output. A sample of these test values in included in the Results section.
2. Application Mode, where the graphical representation of the application is displayed allowing the user to move the line and observe the car adjust its course accordingly.

The Graphical representation of the FIS is written is C++ using SFML 2.4.1 (SFML Team, 2007), and uses a window size of 1280 by 720 pixels. In the top right are two numbers displaying the current Velocity of the Car, measured in pixels per frame, and the current displacement of the car in pixels relative to the Racing Line in that order.



As shown above, the application includes a sprite representing the Car and moveable Racing Line for the car to follow. The Line is moved via the left and right arrow keys, and the car is operated by the Fuzzy Inference System.

The Front End of the application was developed before the FIS in order to establish the FIS’s Universe of Discourse, as detailed in the following section

The FIS was implemented in the application using FuzzyLite Version 6.0 (Juan Rada-Vilela, 2010) which handled the mathematics behind implementing the fuzzzy desing into usable code.

The Design

As per the coursework specifications, the FIS makes decisions based on the car’s displacement from the racing line and the car’s velocity. The Membership Functions were kept simple and intuitive;

With regard to displacement, the car is either to the left of the line, to the right of the line, or is roughly on the line.

For the velocity, the car is either following the line, moving left or moving right.

Velocity

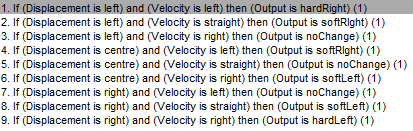
Displacement

|  |  |  |  |
| --- | --- | --- | --- |
| Inputs | Moving Left | Straight | Moving Right |
| Left of Line | Hard Right | Soft Right | No Change |
| Centre | Soft Right | No Change | Soft Left |
| Right of Line | No Change | Soft Left | Hard Left |

As the table shows, the FIS’s Output was given five membership functions in order to produce a more natural range of responses. The outputs are as follow;

1. Hard Right Turn
2. Soft Right Turn
3. No Change
4. Soft Left Turn
5. Hard Right Turn

The Rules linking these together were produced as follows, and stayed consistent throughout the development process.



The Defuzzification method chosen for the App was the Centre of Gravity of the Output. This method may be more mathematically complex and more computationally expensive than some of its common alternatives, but the inherent simplicity of the App meant that the computational costs for this more accurate method were easily met.

The Universe of Discourse (U.o.D.) for the Inputs and Outputs were determined based on the Front End of the Application.

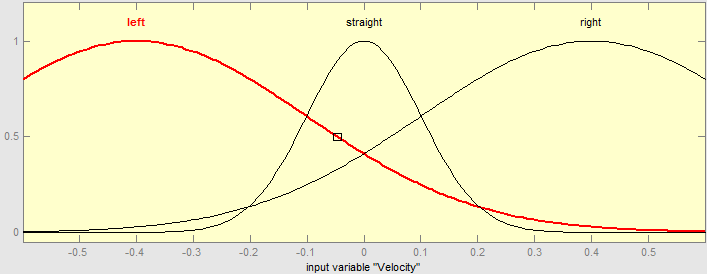
For the Displacement, the U.o.D. extends for a screen’s width in either direction of the line. This may have been excessive, as the user would have to navigate to well outside of the App’s window for any potential issues to occur.

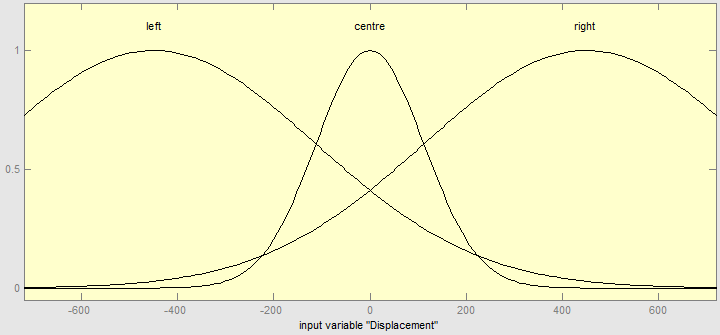
The Velocity’s U.o.D. is the “top speed” of the car in either direction. This top speed was chosen based on how the car’s movement appeared during testing; it is fast enough to respond quickly and visibly, but not so fast that Car’s human-like behaviour is imperceptible or that the sprite fails to move smoothly. This was to fulfil game’s AI primary purpose of appearing believable and human to a user.

The Output’s Domain was set to -1 to 1 in order to “normalize” the returned crisp output in the event that scaling was required.

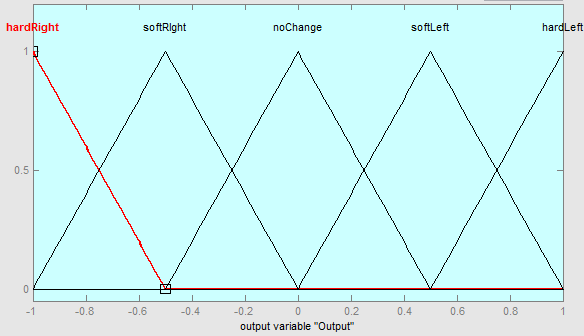
The design described was then implemented in MATLAB (MathWorks, 1984) and refined over three iterations.

Version 1

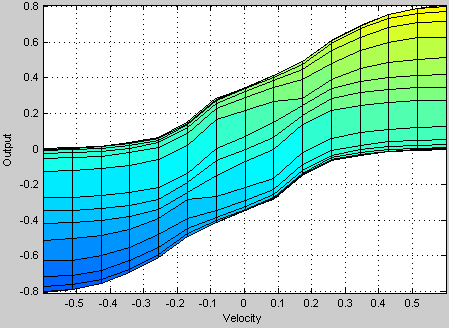
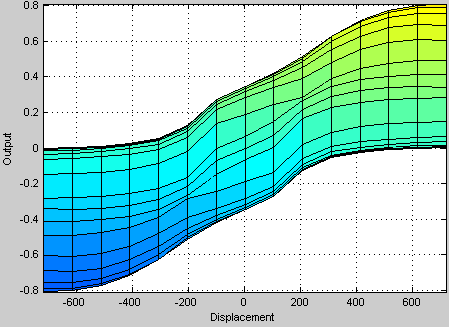
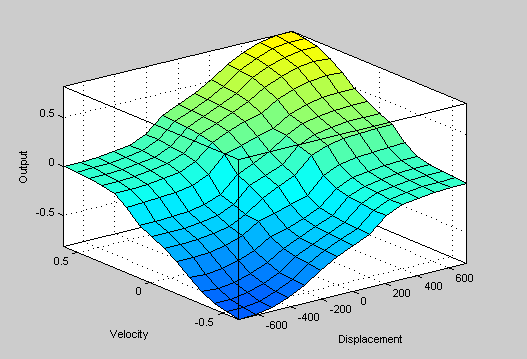




If both cases, the wide domains for the Left and Right values paired with the narrow domains for the central values was intended to produce sizable overlap within the central membership function. This was intended to produce oversteer.



The Membership functions for the Output variable were initially simple uniform triangles for the purpose of evaluating the Input variables. This was modified in subsequent iterations to better match the Input variables.

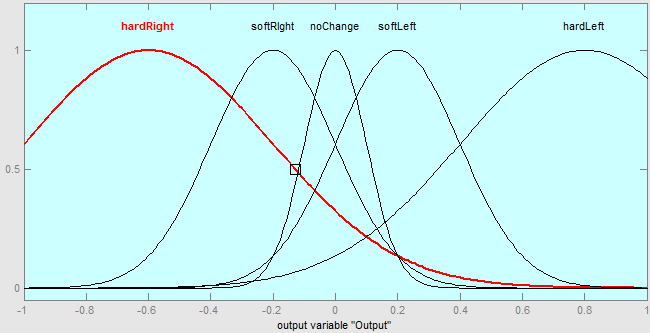
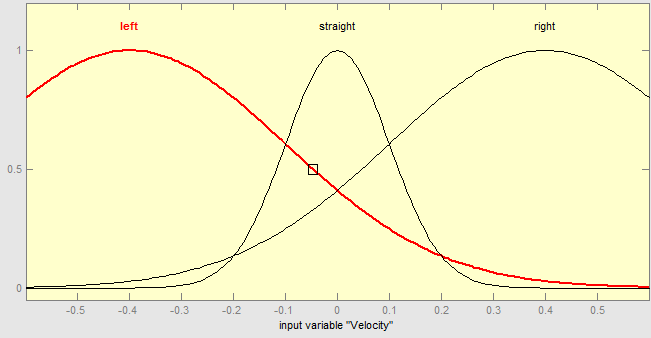
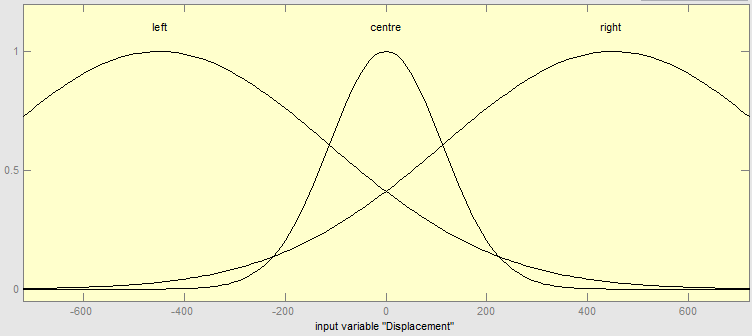


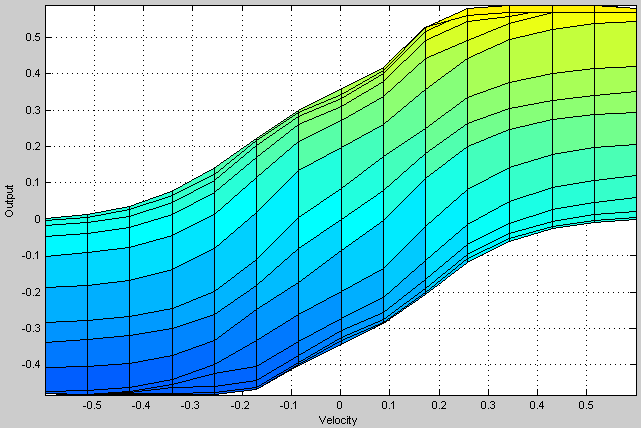
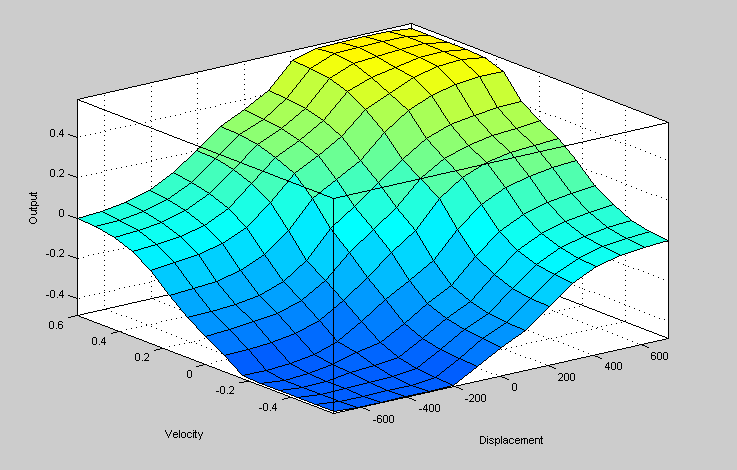
The way in which the output graph start to level out towards the origin indicated that the design was on the correct track, but testing proved that this was not yet sufficient for the desired behaviour.

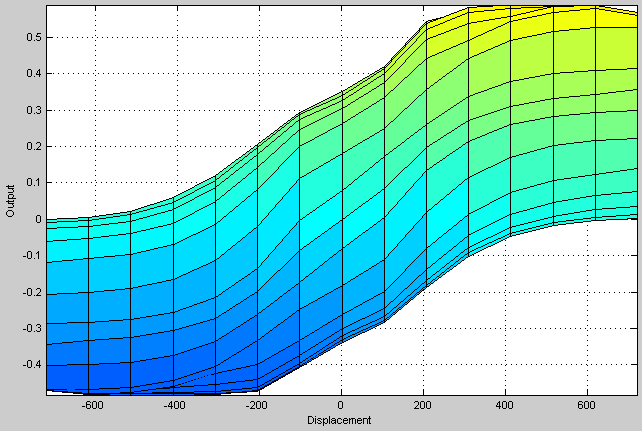
It was also noted that the output graph was smoother than anticipated.. This was taken into account in subsequent iterations.

Version 2

Following the results of Version 1, the Output Functions were modified to use a Gaussian curve and repositioned in a similar fashion to the Inputs. Namely, the Membership Functions have a smaller domain towards the centre and a larger domain moving outwards.

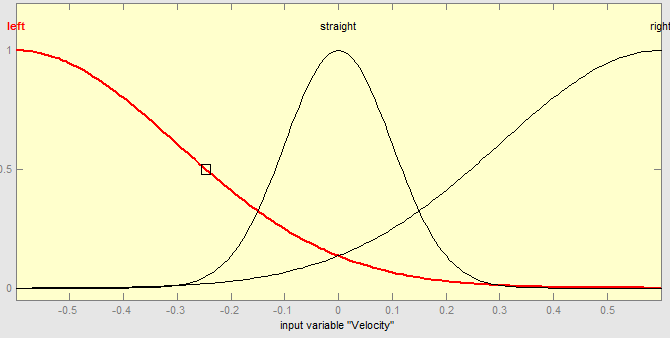


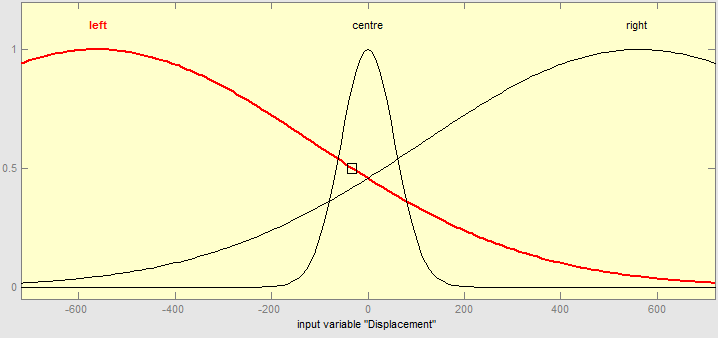




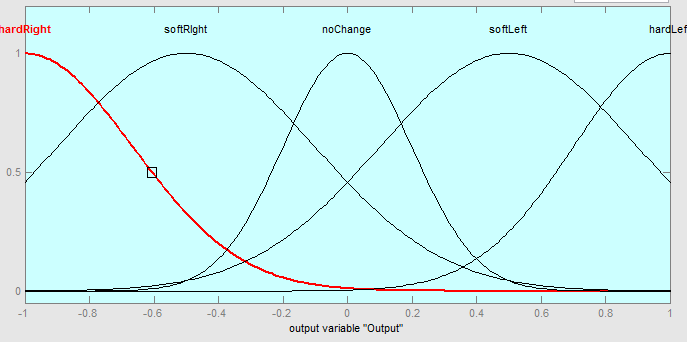
Once again, it was noted that the surface was smoother than anticipated, and thus incapable of producing the deliberate oversteering the App was intended to produce. However, it can be seen that the surface levels out at all four of its extremes. This was intended to produce more dramatic steering, and upon implementation, was deemed to have succeeded in that aspect.

Final Version (3)

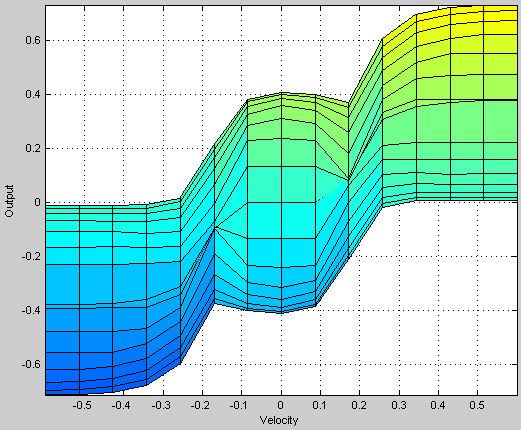
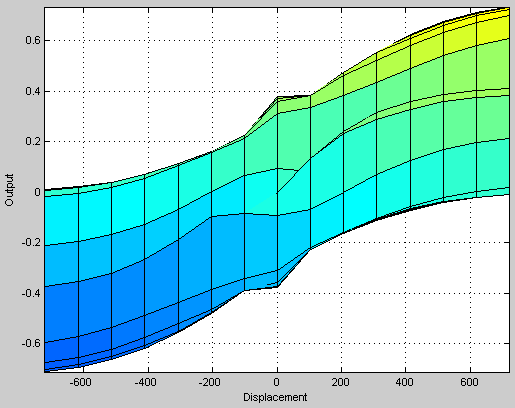
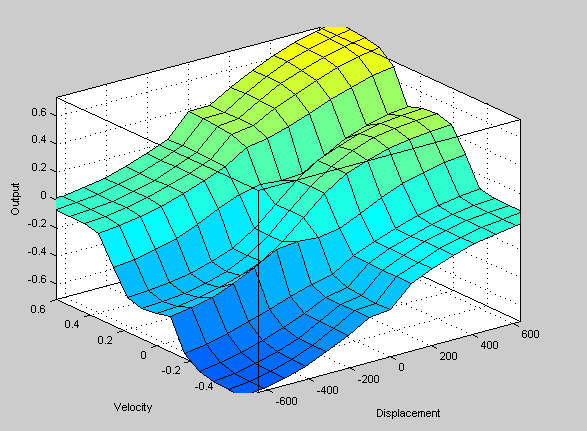




Prior iterations indicated that the key to producing the swerving was in the velocity graph. This makes sense, as “swerving” in reality results from cars attempting to change course too quickly. As such, the Velocity graph was modified to shift the bulk of the membership function from the centre to either side of the centre. Conversely, the Displacement had its Centre Membership Function narrowed and its Left and Right Membership Functions widened to allow for smaller steering corrections.



The domains of the Output’s Membership functions were refined in a similar manner to the Inputs. That is to say, the domains are narrower towards the middle and expand further out. This reflects the applications focus on the car’s behaviour near to the racing line.



Despite an unorthodox appearance, this output graph represents the intended behaviour for the Application. The Displacement can be seen to reach a sharp turning point towards the centre, from whence the car’s ability to make small corrections for gentle changes in course comes from.

The Velocity features clear flat sections at either end, corresponding to the car having a maximum speed at which it can turn. The flat section in the middle, and the steep sections on either side of it it produce the sharp steering towards the line in the event of a drastic change.

However the key to the application is in the “ditches” that exist on either side of the central flat area, as indicated by red arrows. They are only accessed when approached with a high velocity from the other side of the line, and they cause the car to steer back towards the line once oversteering has occurred.

Test Data

The following are tables of example Velocity and Displacement values that have been run first through the design of the FIS in MATLAB, and then through the FIS in the application itself. In both cases, the Velocity and Displacement were varied separately while the other was fixed at 0.

|  |  |  |  |
| --- | --- | --- | --- |
| MATLAB | | | |
| Keeping Displacement Fixed at 0 | | Keeping Velocity Fixed at 0 | |
| Velocity | Output | Displacement | Output |
| 0.6 | 0.378 | 640 | 0.41 |
| 0.5 | 0.377 | 600 | 0.407 |
| 0.4 | 0.368 | 500 | 0.395 |
| 0.3 | 0.343 | 400 | 0.371 |
| 0.2 | 0.196 | 300 | 0.33 |
| 0.1 | -5E-05 | 200 | 0.256 |
| 0 | -4.5E-05 | 100 | 0.145 |
| -0.1 | -5E-05 | 0 | -4.5E-05 |
| -0.2 | -0.196 | -100 | -0.145 |
| -0.3 | -0.343 | -200 | -0.256 |
| -0.4 | -0.368 | -300 | -0.33 |
| -0.5 | -0.377 | -400 | -0.371 |
| -0.6 | -0.378 | -500 | -0.395 |
|  |  | -600 | -0.404 |
|  |  | -640 | -0.41 |

|  |  |  |  |
| --- | --- | --- | --- |
| Application | | | |
| Keeping Displacement Fixed at 0 | | Keeping Velocity Fixed at 0 | |
| Velocity | Output | Displacement | Output |
| 0.6 | 0.418466 | 640 | 0.417531 |
| 0.5 | 0.418454 | 600 | 0.416887 |
| 0.4 | 0.417826 | 500 | 0.413003 |
| 0.3 | 0.402865 | 400 | 0.401385 |
| 0.2 | 0.234197 | 300 | 0.370366 |
| 0.1 | 0.0196924 | 200 | 0.297934 |
| 0 | 3.6572E-17 | 100 | 0.176976 |
| -0.1 | -0.0196924 | 0 | 3.63572E-17 |
| -0.2 | -0.234197 | -100 | -0.176976 |
| -0.3 | -0.402865 | -200 | -0.297934 |
| -0.4 | -0.417826 | -300 | -0.370366 |
| -0.5 | -0.418454 | -400 | -0.401385 |
| -0.6 | -0.418466 | -500 | -0.413003 |
|  |  | -600 | -0.416887 |
|  |  | -640 | -0.417531 |

It was noted that there is a discrepancy between the two sets of data. This can be attributed to differences in centroid calculation methods, number storage (floats vs doubles,) rounding error and the like. The results were graphed using Microsoft’s Excel’s Scatter Graph generator and are as follows;

# Velocity Output Graphs

# Displacement Output Graphs

As these graphs suggest, the discrepancy between the Design and the Application is most pronounced around the origin of the Displacement values, and manifests itself as a more uniform gradient in the Application.

Conclusions

The Fuzzy Inference System behaves almost exactly as intended as designed. The behaves much as a human on a motorway might; it makes small corrections for gentle changes in course, and has an “overreaction” to dramatic, abrupt changes in course.

As can be observed in the tables of test data, there is a discrepancy of up to ~0.1 between the design in MATLAB and the implementation in FuzzyLite. While the FIS behaves almost exactly as intended. Unfortunately, the slight differences between MATLAB and FuzzyLite were most impactful at the focal point of the application.

The result is extremely close to the intended design, but it is likely that the application would have been more nuanced and believable had this not occurred. This likely cannot be proven with certainty without modifications to either FuzzyLite or MATLAB.

The Application serves as a solid example for some of the strengths of Fuzzy Inference Systems. It is simple to implement, easy to understand, has acceptable computational costs and is capable of producing natural, human-like behaviour.

References

* Dr. D. King, (2018) ‘Fuzzy Logic and Fuzzy State Machines’ [PowerPoint presentation, [*MAT301.2017-8.S2.A Mathematics and Artificial Intelligence*](https://blackboard.abertay.ac.uk/webapps/blackboard/execute/courseMain?course_id=_7666_1). Available at :<https://blackboard.abertay.ac.uk/webapps/blackboard/execute/content/file?cmd=view&content_id=_468300_1&course_id=_7666_1> (Accessed: February 18th, 2018)
* SFML Team (2007) *Simple and Fast Multimedia Library* (Version 2.4.1) [Computer Program] Available at <https://www.sfml-dev.org/download/sfml/2.4.1/> (Accessed : February 18th, 2018)
* Juan Rada-Vilela (2010) *FuzzyLite* (Version 6.0) [Computer Program] Available at <http://www.fuzzylite.com/>(Accessed: February 27th, 2018)

OR

* Juan Rada-Vilela. fuzzylite: a fuzzy logic control library, 2017. URL <http://www.fuzzylite.com/> (referencing style requested by author.)
* MathWorks (1984) *MATBALB* (Version R2012a) [Computer Program]. Available at <https://www.mathworks.com/products/matlab.html> (accessed February 18th, 2018)
* Pears, R. and Shields, G. (2016) *Cite Them Right*, Pear Tree Books, Volume 10.
* Rabin, S. (ed.) 2005