Car Racing with Fuzzy Logic

CMP304 AI

Lewis Thomson | 1601828 | 2019

Introduction

For this module, I have chosen the assignment of creating a “racing game” application that features a Fuzzy Inference System that would control the steering of the vehicle. The two inputs for the FIS system are the distance between the car and a theoretical racing line, and the output is a velocity being pushed back into the system. I must also compare the FIS system to an alternative AI system, to which I am choosing an else-If system.

For the implementation of the application, I will be using MATLAB[1] and its Fuzzy System toolkit to create the AI itself, for testing and coding, I will be using Visual Studio C++[2] and making use of SFML[3] for a graphical interface. Finally, I will use an opensource API known as FuzzyLite[4] for the use of ‘.fis’ files in the application.

Methodology

I started by analyzing the task, the first thing that I had to consider, was how many states I would like to FIS to have. I started my thought process of having only 3 states. ‘Left, Neutral, Right’, however once thinking about how this would perform, it would seem quite mechanical and binary (in a sense). So I decided to add another two states, these being ‘FarLeft, FarRight’. This means that the ‘car’ will have a middle state between not moving and moving quickly.

This is due to the nature of Fuzzy Logic systems, these system function by taking in numbers, making them quite vague by ‘fuzzifying’ them, and then performing logic operations on thresholds that are met (on what are called Membership Functions) that then trigger outputs that will be combined to form a ‘crisp’, ‘defuzzified’ output value. The process of fuzzifying numbers, and comparing them to Membership Functions is similar to the human brain of quantifying and identifying data as abstract terms like ‘near’ or ‘far’.

Using these terms of ‘near’ and ‘far’ is what allows the FIS to steer the car towards the line.

After deciding on these five states, I thought it would be a helpful process to create a fuzzy associative map(fig 1) for defining all the rules required for the

			Car position, relative to line.			
		Far Left	Left	Zero	Right	Far Right
	Fast Left	Fast Right	Fast Right	Fast Right	No Change	No Change
	Slow Left	Fast Right	Slow Right	Slow Right	No Change	Slow Left
Current Speed	Neutral	Fast Right	Slow Right	No Change	Slow Left	Fast Left
	Slow Right	Slow Right	No Change	Slow Left	Slow Left	Fast Left
	Fast Right	No Change	No Change	Fast Left	Fast Left	Fast Left

Figure 1

system.

Using MATLAB's Fuzzy toolbox, I created the FIS using the shown input restrains and functions. (fig 2). The 'DistanceFromLine' and 'RateOfChange' both

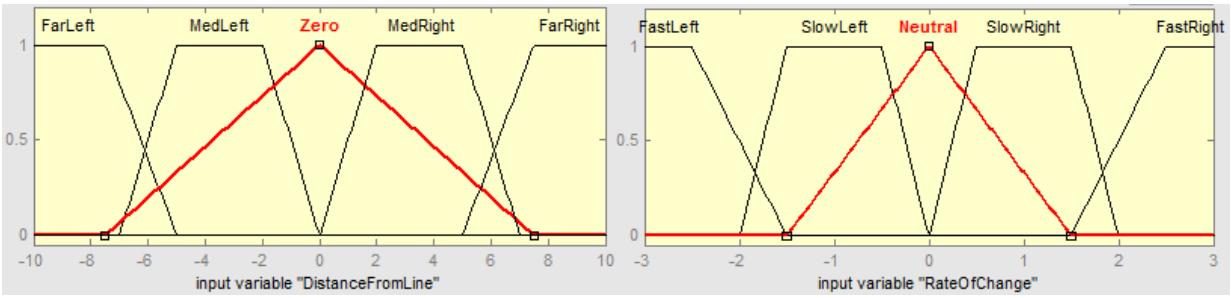

Figure 2

use trapezoids for its Membership Functions, except from Neutral/Zero, which means that these functions are only fully true, when they are at 0. However the trapezoids allow there to be a larger range of values which means the function is true, allowing a smoother transition between states in movement.

However, for the output values only use triangles, which mean that there are a smaller range of combined inputs that allow for each output. Meaning that transitions will be smooth, yet noticable. (fig 3)

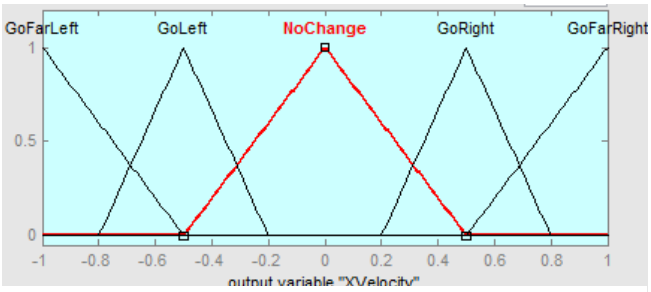

Figure 3

So from here, I implemented the rules as seen in the Fuzzy Associative Map. This means that there will be rule for every combination of inputs, leads to 25 total rules for deciding the output. (fig 4)

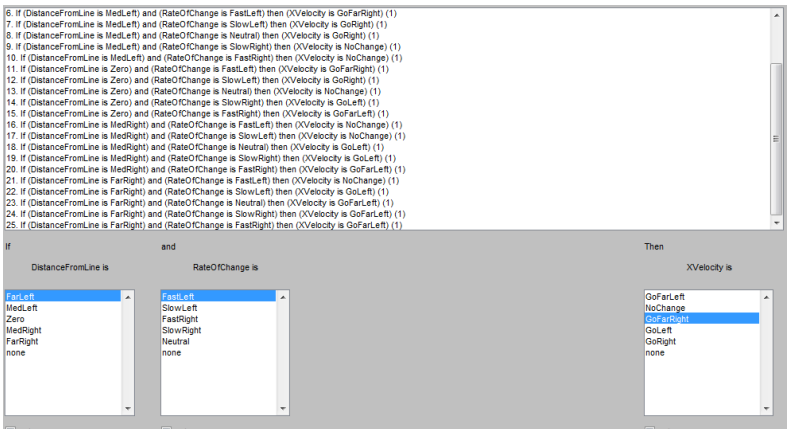

Figure 4

Results

The application performs as about expected. The car accurately follows the line by speeding up or slowing down, depending on how far away the line is from the car, and how fast the car is currently moving. To test these results, I used the

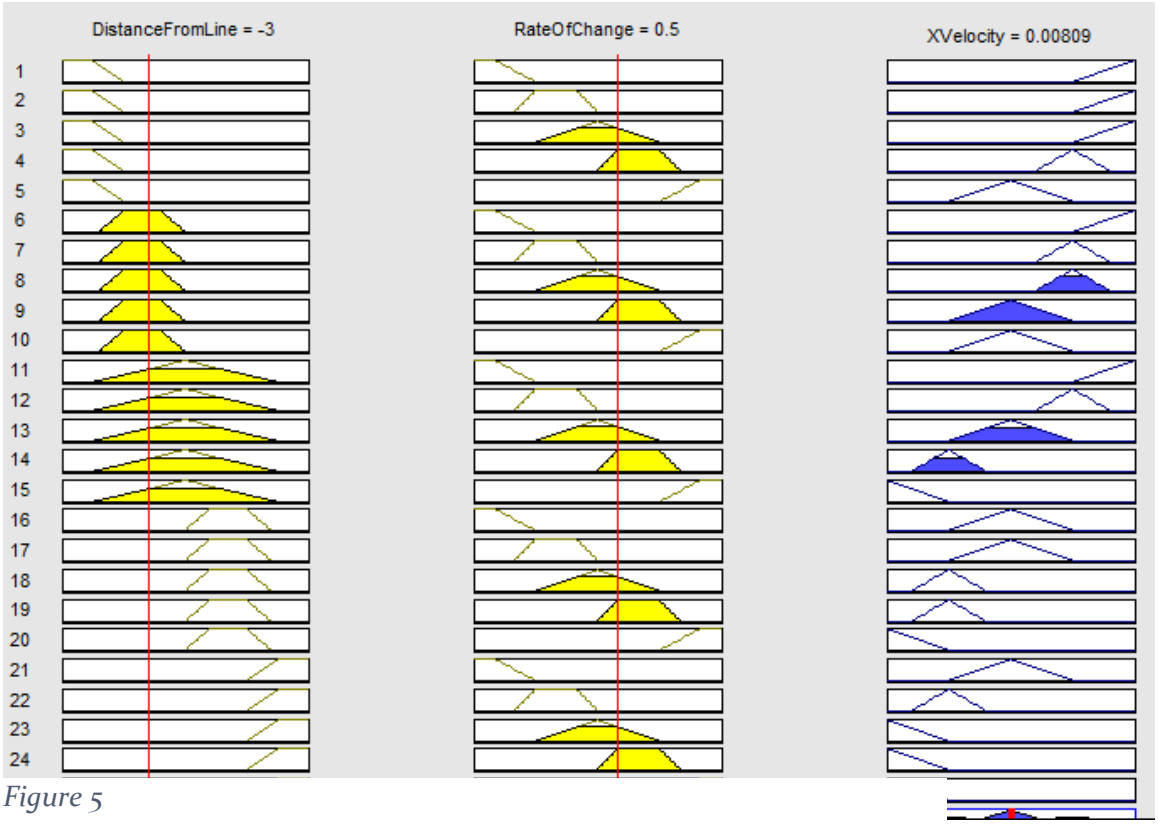

Figure 5

MATLAB toolkit to test a variety of input and output values, and then also test these within the application itself. (fig 5)

Above is one of the many tests I ran, in which I set the car to be units to the left of the line, and already moving at 0.5 units to the right, per frame. The resulting output was to add another 0.008 units to that value, thus still accelerating the car, but only minorly. I also ran several more tests, as shown (fig 6) that give expected results. As mentioned, I also cross-checked these results in my application which gave results within a 0.01~0.05 margin of error. This is most likely due to MATLAB and my application using different accuracies of decimal places.

	Test Cases	
Distance	RoC	Output
-3	0.5	0.00809
3	-0.5	-0.00809
0	2	-0.811
0	-2	0.811
8	3	-0.84
-8	3	1.39E-18
0	0	1.39E-18
10	3	-0.84
-10	3	1.39E-18
-5	1.5	-0.106
5	1.5	-0.5
1.5	3	-0.834
1.5	-3	0.288
-1.5	3	-0.288
-1.5	-3	0.834
7.5	0	-0.84
7.5	-1	-0.585
7.5	-2	1.37E-17
7.5	1	-0.84
7.5	2	-0.811

Figure 6

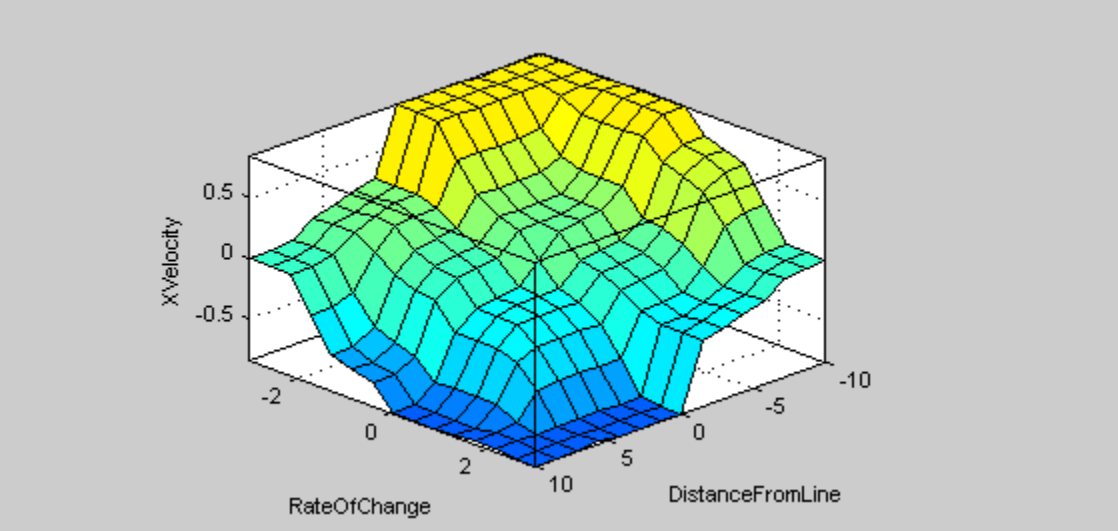

Figure 7

Finally, I used MATLAB's 'surface' feature to show a graphical representation of the possible outputs, corresponding to their respective inputs. (fig 7)

Conclusion

So, to summarize. This project set out to create a FIS that would control a car in a ‘racing game’ application and control the steering. To emulate this, I created a GUI that allows the user to move the race line, and see the dynamic changes in the cars position, thanks to the FIS. In comparison to an else-if system, this is so much more dynamic and allows for a much larger range of outputs, rather than only being limited to and clamped at specific values. An else-if system using similar rules to this Fuzzy system, would only have a total of 5 different speeds, rather than the smooth, ramping up and deacceleration of the created application. If this were to be implemented into a more complex game, I feel as though I would cull off two of the middle-distance values, as adding more inputs for distance (front, back, etc). Would drastically increase the number of rules required, almost exponentially.

Overall, I feel the use of a FIS in this case is warranted. Though if this FIS was being applied to multiple different objects at the same time, I feel as though this would impact performance a lot more than running several else-if systems at the same time.

References

[1] Mathworks.com. (2019). *MATLAB - MathWorks*. [online] Available at: https://www.mathworks.com/products/matlab.html [Accessed 9 Apr. 2019].

[2] Visual Studio. (2019). *C and C++ Coding Tools | Visual Studio*. [online] Available at: https://visualstudio.microsoft.com/vs/features/cplusplus/ [Accessed 9 Apr. 2019].

[3] Sfml-dev.org. (2019). *SFML*. [online] Available at: https://www.sfml-dev.org/ [Accessed 9 Apr. 2019].

[4] FuzzyLite. (2019). *FuzzyLite*. [online] Available at: https://fuzzylite.com/ [Accessed 9 Apr. 2019].