Assessment 1

Knowledge Representation & Reasoning

CS551J

Dr. George Coghill

Yassin Dinana

52094480

8th March 2021

Task 1:

Introduction:

Fuzzy logic is based on the theory of fuzzy sets, it is used in many different systems nowadays, where it has the advantage of dealing with vague systems and using linguistic variables. This paper will be focusing on the research of fuzzy logic in autonomous vehicles where it can control many areas in the car such as steering wheel control, speed / cruise control, and the braking system. The research will be centring on a system with one consequent which is the ***Autonomous Vehicles Braking-System,*** where different bands are used as antecedents to the fuzzy system which control one-consequent. Such antecedents initially include the distance from the surrounded cars and the speed of the vehicle, different antecedents will be introduced later in the paper as part of the tests and experiments. All of the antecedents provided play a role in the solid level of the braking [1].

Introducing & Designing the system:

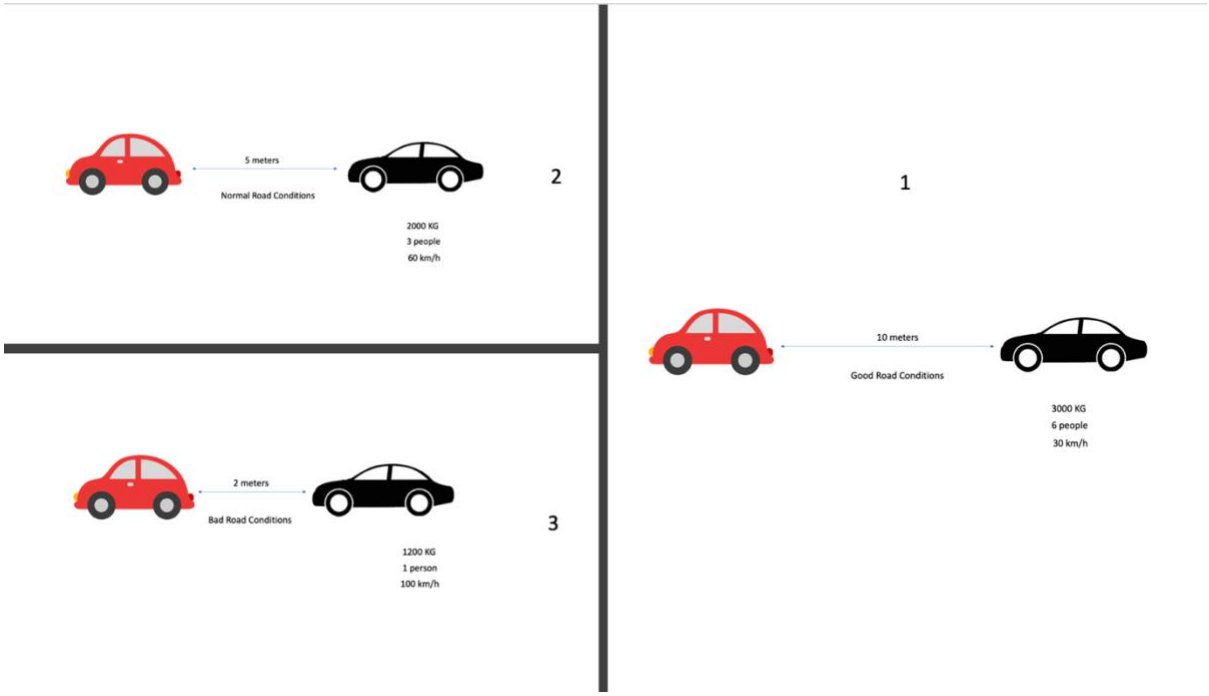


Figure 1: Different scenarios for the braking-system level computation

Looking at figure 1 above it shows different situations that can be available when controlling the brakes power in an autonomous vehicle.

The first situation (1) shows the autonomous vehicle in black and a nearby vehicle in red. The distance between both vehicles is 10 meters, it is also specified that the vehicle is moving forward in a speed of 30 km/h. In such situation:

- Distance is far
- Car is moving slow

Braking should be soft

The second situation (2) shows the autonomous vehicle in black and a nearby vehicle in red. The distance between both vehicles is 5 meters, it is also specified that the vehicle is moving forward in a speed of 60 km/h. In such situation:

- Distance is equidistant
- Car is moving at a moderate speed

Braking should be standard

Finally, the third situation (3) shows the autonomous vehicle in black and a nearby vehicle in red. The distance between both vehicles is 2 meters, it is also specified that the vehicle is moving forward in a speed of 100 km/h. In such situation:

- Distance is very close
- Car is moving at a fast speed

Braking should be strong

For such system a set of if-then rules can be provided, such that:

- If distance is far **and** car is moving slow, **then** apply soft brakes.

- If distance is equidistant **and** car is moving at a moderate speed, **then** apply standard brakes.

- If distance is close **and** car is moving fast, **then** apply strong brakes.

Those rules will later be changed when applying different experiments and as well as applying different defuzzification methods.

In order for the system to understand different terms represented such as heavy, slow, fast, soft, strong, those terms are defined in the code by being applied within a specific range of values. The representation is done using *4-tuple representation*.

As there two initial variables controlling the toughness of the brakes which are speed and distance – those are used to compute a decision as well as representing the states of the variables in a specific level in graphs as well as the fuzziness / vagueness of the system situation.

When defining the variables and parameters responsible for the decision making and fuzziness, the 4-tuple parametric representation is presented as the following:

$$(a, b, \alpha, \beta)$$

Where *A & B* represent the width of the nucleus.
α & β represent the width of the vagueness part [2].

Speed:

- As stated, a car is considered driving *slow* if it is moving forward in a speed between 0 km/h and 30 km/h.
- It is also stated that a car is driving on an *average* speed if it is moving forward in a speed between 30 km/h and 60 km/h.
- A car is considered moving fast if it is moving forward at a speed for 80 km/h+

```
speed['slow'] = fuzz.trapmf(speed.universe, [0,0,20,30])
speed['average'] = fuzz.trapmf(speed.universe, [30, 45, 55, 60])
speed['fast'] = fuzz.trapmf(speed.universe, [60, 80, 100, 100])
```

Figure 2: Defining speed variables & parametric

By looking at the figure 2 above, different attributes are given to the variable speed, where the speed can be slow, average, or fast. The figure shows vehicle speed is slow when it is moving from 0 km/h to 30 km/h. It represents the average speed between 30 km/h to 60 km/h, and finally representing a vehicle moving fast to be moving at a speed for 80 km/h+ in the form of 4-tuple representation. The shape representing it is a "Trapezoid" which is given to be *fuzz.trapmf*.

The membership function on the Y-Axis maximum is 1, which represents the highest values for slow (30 km/h), maximum for the average speed (60 km/h), and finally the maximum membership of 1 represents the highest fast speed of (80 km/h+).

The figure below shows the fuzzy graph generated by the speed functions above.

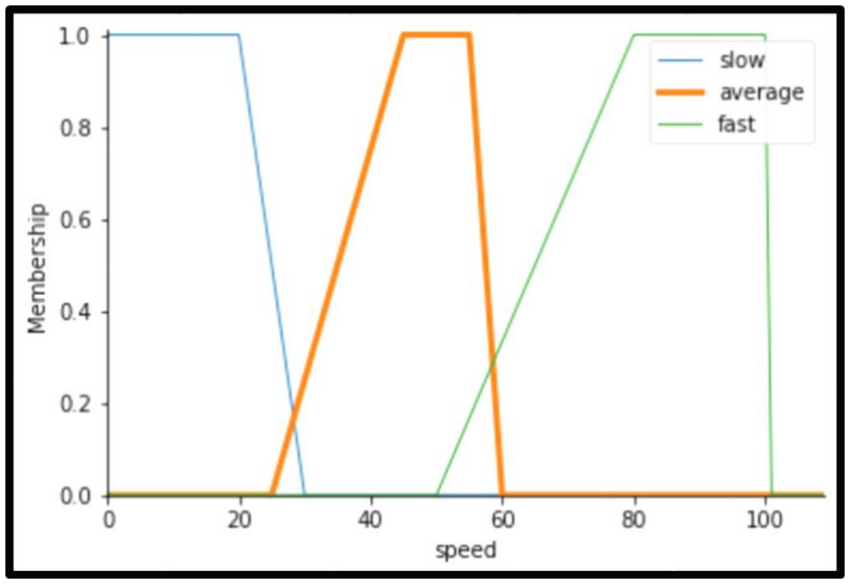


Figure 3: Fuzzy graph generated for the speed effect on brakes

Looking at figure 3, there are three trapezoids where the red trapezoid represents the slow speed, the orange trapezoid represents the average speed, and finally the purple trapezoid represents the fast speed. It can also be seen that there is an overlap happening between the trapezoids, the area in the overlap represents the fuzziness or vagueness of the system. The trapezoids representation can also be seen on the top right legend box.

Distance:

- As stated, a car is considered at light if the weight of the vehicle ahead is 0 to 3 meters away.
- It is also stated that a car is considered at equidistant (not too close, not too far) if the vehicle ahead is 3 to 6 meters away.
- A car is considered far if the vehicle ahead is 7 to 10 meters away – 10 meters away is the highest membership of 1.

```
5 distance['close'] = fuzz.trapmf(distance.universe, [0,0,2,3])
6 distance['equidistant'] = fuzz.trapmf(distance.universe, [3,4,5,6])
7 distance['far'] = fuzz.trapmf(distance.universe, [7, 8, 9, 10])
```

Figure 4: Defining distance variables & parametric

By looking at the figure 4 above, different attributes are given to the variable distance, where the distance can be close, equidistant, or far. The figure shows vehicle is in close distance when the distance from the car ahead is0 to 3 meters. It represents the average equidistant if the vehicle ahead is 3 to 6 meters away, and finally representing a vehicle that is far if the vehicle ahead is 7 to 10 meters away in the form of 4-tuple representation. The shape representing it is a "Trapezoid" which is given to be *fuzz.trapmf*.

The figure below shows the fuzzy graph generated by the speed functions above.

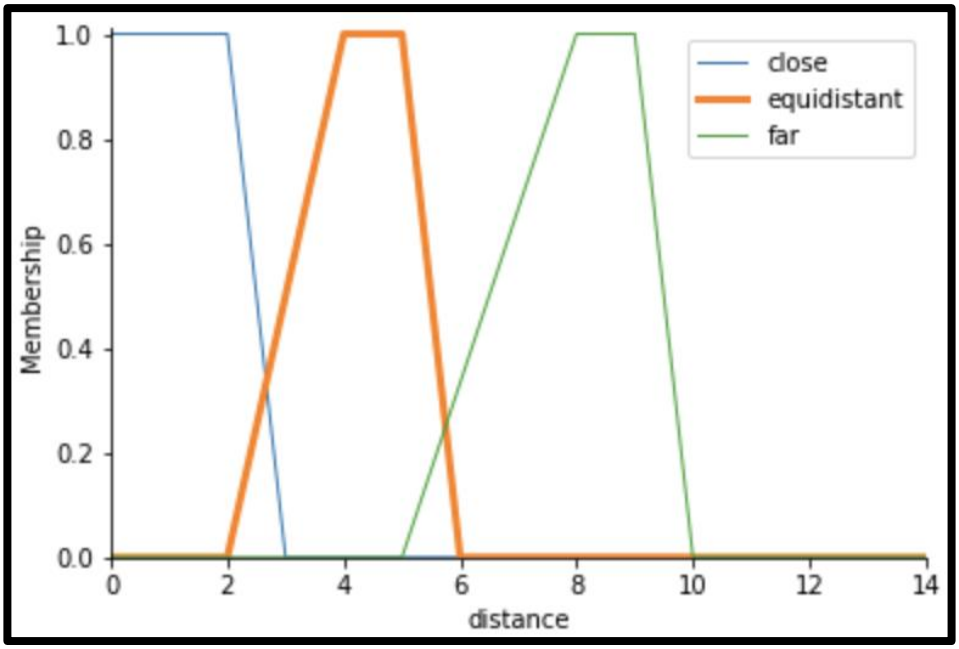


Figure 5: Fuzzy graph generated for the distance effect on brakes

Looking at figure 5, there are three trapezoids where the red trapezoid represents the close distance, the purple trapezoid represents the equidistant, and finally the brown trapezoid represents the far distance. The trapezoids representation can also be seen on the top right legend box.

The graph in figure 6 above shows the states that the brakes can be applied upon, it shows three trapezoids, the blue trapezoids shows where the brakes are soft, the orange trapezoid shows where the brakes are standard, and the green trapezoid shows where the brakes are strong. The state of the brakes is applied upon the rules given before being computed to generate the crisp output.

```
[20]  1 brakes.view()
      2 plt.legend(loc="upper right")
```

```
<matplotlib.legend.Legend at 0x7f7c213508d0>
```

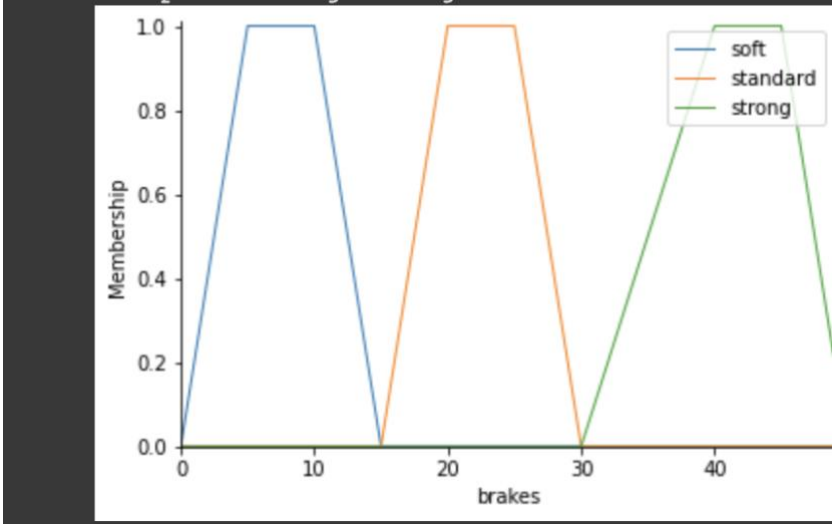


Figure 6: Fuzzy graph representing the consequent (Brakes)

Figure 7 below shows the set of if-then rules specified initially for the system, where the antecedents are given inputs and the consequent which is the brakes in this paper acts upon it. When changing the input given to the system, this will result in the rules changing and therefore, the output will change in the form of the computed output and crisp values. Different results will be shown by having more / less fuzziness and accuracy.

```
[12]  1 #Defining Rules
      2
      3 rule1 = ctrl.Rule(distance['far']  & speed['slow'], brakes['soft'])
      4 rule2 = ctrl.Rule(distance['far'] & speed['average'] , brakes['standard'])
      5 rule3 = ctrl.Rule(distance['far']  & speed['fast'] , brakes['strong'])
      6 rule4 = ctrl.Rule(distance['equidistant']  & speed['slow'], brakes['soft'])
      7 rule5 = ctrl.Rule(distance['equidistant']  & speed['average'] , brakes['standard'])
      8 rule6 = ctrl.Rule(distance['equidistant'] & speed['fast'] , brakes['strong'])
      9 rule7 = ctrl.Rule(distance['close']  & speed['slow'], brakes['strong'])
     10 rule8 = ctrl.Rule(distance['close'] & speed['average'] , brakes['strong'])
     11 rule9 = ctrl.Rule(distance['close'] & speed['fast'] , brakes['strong'])
```

Figure 7: Set of rules provided to the system

After implementing the rules, it is needed to change the fuzzy values which were given in the membership graphs generated in figures 3, 5, and 6 to normal numerical values known as the crisp output. This process is called the defuzzification, it produces a quantifiable result in the form of crisp logic. There are different defuzzification methods that will be addressed where different results are generated and compared in the experiments section.

1.1-

Some experiments are implemented on this system which will generate different results that can be discussed and compared – the experiments will enhance the understanding of the fuzzy system and its outputs.

Experiment 1:

The first experiment involves changing the input values provided to the system as well as representing the system mathematically to establish better understanding and analysis. The rationale for choosing this experiment is because the output should be changing when varying the input values, changing the output includes changing the range from soft to strong brakes, choosing input values outside the provided range, and finally changing the rationale of the input values such as adding a maximum speed and a minimum distance. it is a good experiment to test the efficiency of the system and making sure it is working well in different conditions using the given antecedents and rules.

Experiment 2:

The second experiment involves editing and altering the rules. Adding more rules, less, rules, or editing the rules themselves should have effect on the output of the system compared to the 9 initial rules presented in figure 7 of this paper. The rationale for choosing this experiment is to visualise on the crisp output the different when applying the rules initially present it and when changing them or adding more/less rules. It is a good experience to understand and analyse the system response in case different rules occur.

Experiment 3:

The third experiment involves adding more antecedents to the system. Meaning that not only the speed and distance will compute the power of the braking which is the consequent. Different antecedents will be added such as the weight of the car, the road condition, and how many passengers are in the vehicle. When adding more antecedents, the rules will have to include the new antecedents as well as the number of inputs will increase. The rationale of this experiment is to visualise and compare the output with only two antecedents and then the output with an increasing number of inputs and antecedents. The goal of this experiment is to try and establish which output has less vagueness and more predictable crisp output.

Experiment 4:

The fourth experiment involves applying different defuzzification methods. There are different defuzzification methods that can be applied such as Centroid, Bisector, Mean of Maximum, Min of Maximum, and Max of Maximum. Each crisp output that will be generated after the defuzzification will have different results. The rationale of such experiment is to compare the output of all the defuzzification methods listed above and then decide which method is the most compatible and attuned for the system.

1.2-

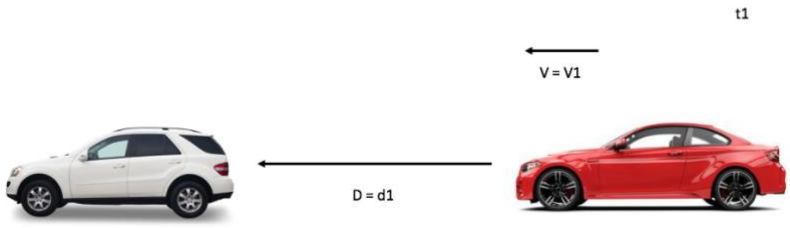


Figure 8: Representation to produce mathematical explanation of the system

Experiment 1:

The figure above, represents the situation at time = t1, distance = d1, and velocity = v1.

The equations below represent the system in the next state, where the output is either acceleration (no action in our system) or braking which is represented below as de-acceleration.

$$Vehicle\ at\ time = t2$$

$$d2 = d1 - v1 \times \Delta t$$

$$v2 = v + v1 \times (a \times \Delta t)$$

$$a\ (braking) = FuzzyOutput(v1, d1)$$

Looking at the equations above, it can be seen that the output is represented mathematically as the fuzzy output depending on the speed and the distance of the vehicles. Representing the equations above in a graph form should inherit the following graphical representation:

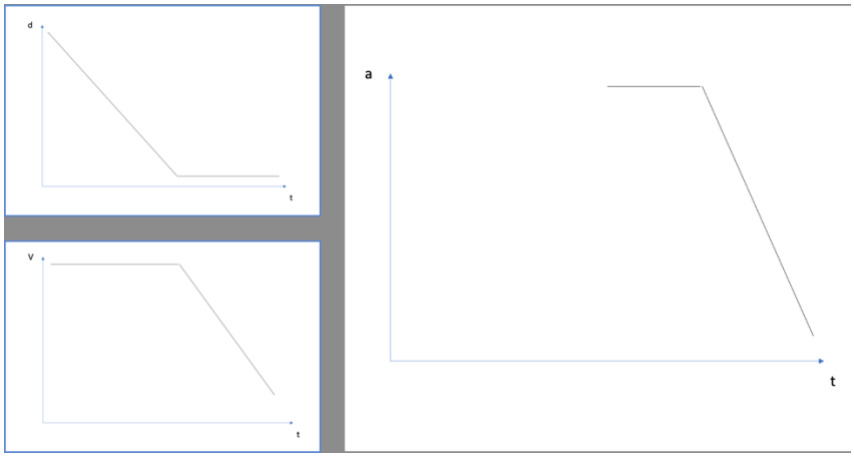


Figure 9: Graphical representation of the mathematical equations presented above

The graphs in figure 9 above represent the output represented as “a” with regards to “d – distance” and “v – velocity”, the graphs are developed from the equations above and this should be the output of the system implemented along with the fuzziness like seen in the equations. Now initially, the system is tested to be working efficiently with different inputs for all its states where the output should be soft, standard, or strong states.

Firstly, testing the system with inputs that should compute the output to be soft brakes. Remembering that the range of the speed is between 0 - 100, the distance range is between 0 – 10. Giving low values inputs representing slow speed and far distance should compute that the brakes should be soft.

```
1 braking.input['distance'] = 2
2 braking.input['speed'] = 20
3
4 braking.compute()
5 print(braking.output['brakes'])
6 brakes.view(sim=braking)
```

Figure 10: Testing the system with low input

Looking at figure 2 in this paper, it can be seen that the speed 20 is slow and distance 2 is far. The output should be soft brakes.

The blue trapezoid represents the soft brakes.
The orange trapezoid represents the standard brakes.
The green trapezoid represents the strong brakes.

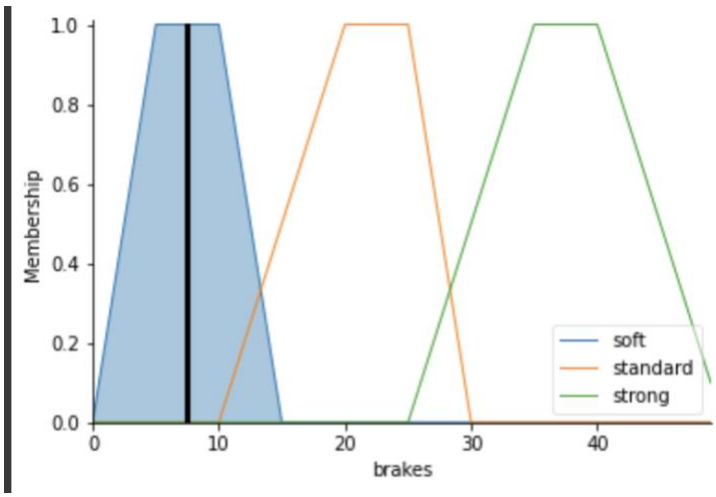


Figure 11: Output of low values input

Figure 11 above shows the output when the speed is slow, and the distance is far. The output is soft brakes with a membership of 1 meaning that it is the highest membership in the soft brakes as the numbers are very low. The output shows that the system is working for the slow values.

The next step is to verify what should be the output of the system in the case of adding the values of a medium speed as an input of the speed and a distance that is close as the input of the distance. This shows that both inputs are not from the same range. The expected outcome for such system should be that there will be vagueness and fuzziness as part of the output. Figure 12 below shows the input values to the system.

```
1 braking.input['distance'] = 8
2 braking.input['speed'] = 55
3
4 braking.compute()
5 print(braking.output['brakes'])
6 brakes.view(sim=braking)
```

Figure 12: Testing the system with fluctuating input values

Figure 13 below shows the output of the input values above.

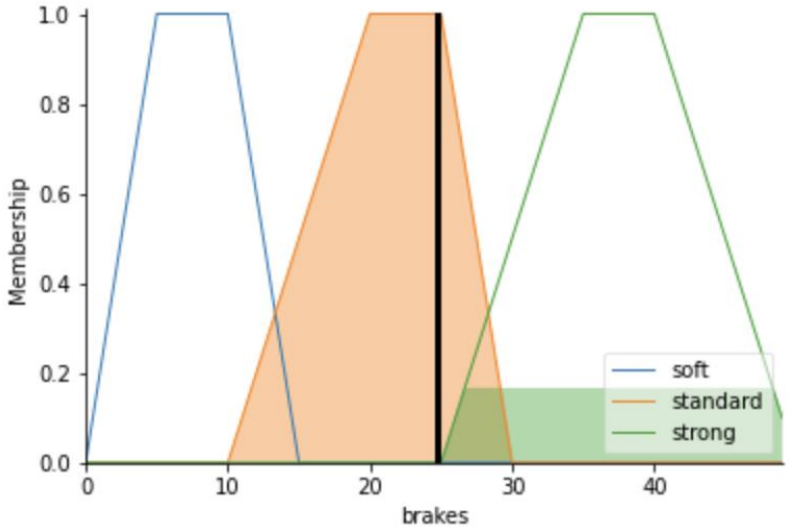


Figure 13: Output the system with fluctuating input values

As it can be seen in figure 13 above, the output of the system has some vagueness and fuzziness, the system is oriented towards the standard (medium braking) as it has a membership of 1 but there's also some fuzziness towards the strong baking with a membership of only 0.2.

Another part of this experiment could be to try and see what the output of the system will be if the values given as input are out of the range specified in figure 2 of this paper. The maximum speed provided is 100 and the maximum distance is 10 meters. Specifying ranges for the antecedents is important to allow the visualisation of the system graphically where the range of values specified is between the membership of 0 and 1. A predicted output when adding values outside the range provided would be problematic when concluding the output as the area under the curve is then unlimited.

Adding out of range values such as 12 for the distance and 150 km/h for the speed produced an error that can be seen in the figure below.

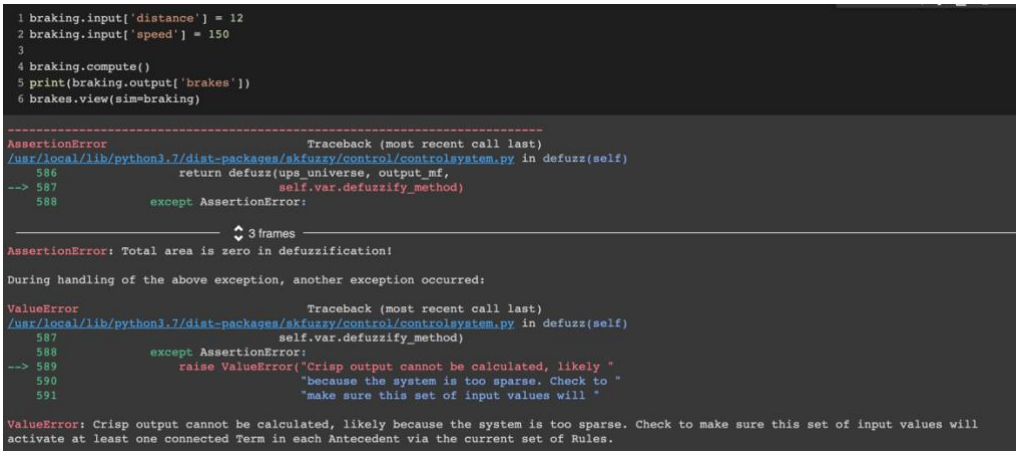


Figure 14: Output / Error when providing values out of range

Looking at the error in figure 14, it means that the system cannot provide a crisp output from the input values given relating to the rules provided. The system cannot compute an output with undefined values / range along with the given rules, meaning that when adding input values which are out of range, the system won't be able to compute a crisp decision and decide whether the brakes should be soft, standard, or strong.

Experiment 2:

As stated, when defining the rules, the second experiment involves altering and changing the rules provided to the system. The initial rules which generated all the outputs discussed previously in this paper are computed using the rules that can be seen in figure 7 of this paper.

The first part of this experiment is to decrease the number of rules provided, initially there are 9 rules provided to the system. The aim of this part is to check whether the system will be able to generate an output with only 6 rules interchanged. Only 6 rules are chosen to be added to the system, the rules can be seen in the figure below and the 3 remaining rules are commented in the code.

```
[55]  1 #Defining Rules
      2
      3 #rule1 = ctrl.Rule(distance['far']  & speed['slow'], brakes['soft'])
      4 rule2 = ctrl.Rule(distance['far'] & speed['average'] , brakes['standard'])
      5 rule3 = ctrl.Rule(distance['far']  & speed['fast'] , brakes['strong'])
      6 rule4 = ctrl.Rule(distance['equidistant']  & speed['slow'], brakes['soft'])
      7 #rule5 = ctrl.Rule(distance['equidistant']  & speed['average'] , brakes['standard'])
      8 rule6 = ctrl.Rule(distance['equidistant'] & speed['fast'] , brakes['strong'])
      9 rule7 = ctrl.Rule(distance['close']  & speed['slow'], brakes['soft'])
     10 rule8 = ctrl.Rule(distance['close'] & speed['average'] , brakes['standard'])
     11 #rule9 = ctrl.Rule(distance['close'] & speed['fast'], brakes['strong'])
     12
```

Figure 15: Set of 6 rules provided for the experiment

After running the code on the rules provided in the figure above, it can be seen that the code can only compute the output when the input values are provided in the rules. When trying to add values that satisfy rule 9 which is commented in the code above, the system is not able to compute it and provides an error. This concludes that in order for a system to work

efficiently, a system with two antecedents, must have at least 3 rules in order to operate efficiently where there 3 rules for each antecedent and 1 consequent output for each.

When changing the braking power in the brakes, meaning changing the output of some rules, the output is also changed directly including fuzziness in all the system. As part of the experiment, the rule:

```
rule3 = ctrl.Rule(distance['far']  & speed['fast'] , brakes['strong'])
```

is changed to:

```
rule3 = ctrl.Rule(distance['far']  & speed['fast'] ,brakes['standard'])
```

When applying the input values and computing the output, it can be seen in the figure below that the main membership lands on the standard braking as mentioned in the rule, but also there is fuzziness and vagueness between the soft braking and strong braking which can be seen in the area under the graph merging with the standard.

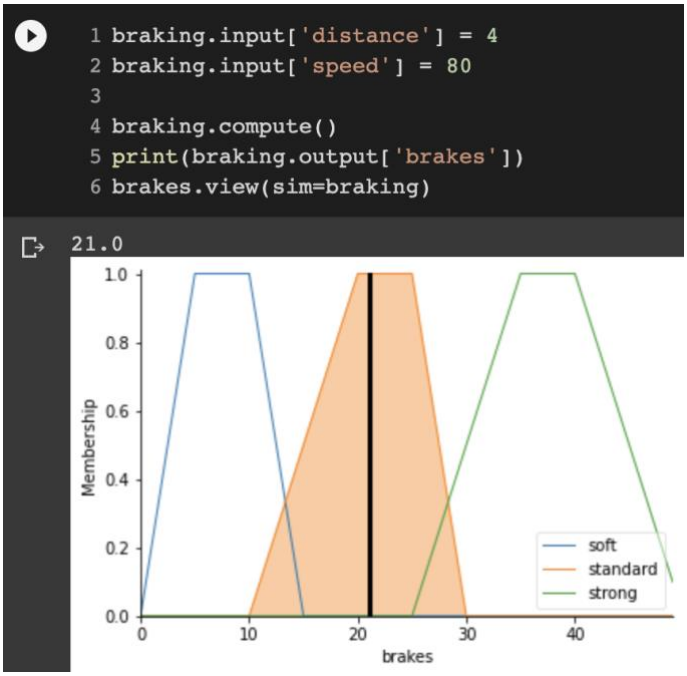


Figure 16: Output produced when changing the brake states in the rules

This part of the experiment can prove that changing the rules can result in more or less vagueness to the system. Therefore, keeping the rules tuned and correct will result in more accuracy and efficiency of the system.

Another test case in this experiment is to use the OR rule instead of the AND rule. An example of this could be:

if distance is ‘close’ OR speed is ‘fast’ then apply ‘strong’ brakes.

This set of rules is applied on some of the rules provided to the system which will help visualise the different reaction of the system’s output.

Changing some of the rules from & to OR can be observed in the set of rules below:

```
#Defining Rules

rule1 = ctrl.Rule(distance['far']  | speed['slow'], brakes['soft'])
rule2 = ctrl.Rule(distance['far'] & speed['average'] , brakes['standard'])
rule3 = ctrl.Rule(distance['far']  & speed['fast'] , brakes['strong'])
rule4 = ctrl.Rule(distance['equidistant']  | speed['slow'], brakes['soft'])
rule5 = ctrl.Rule(distance['equidistant']  & speed['average'] , brakes['standard'])
rule6 = ctrl.Rule(distance['equidistant'] & speed['fast'] , brakes['standard'])
rule7 = ctrl.Rule(distance['close']  | speed['slow'], brakes['soft'])
rule8 = ctrl.Rule(distance['close'] & speed['average'] , brakes['standard'])
rule9 = ctrl.Rule(distance['close'] & speed['fast'], brakes['strong'])
```

Figure 17: Changing some rules to have OR instead of AND.

The set of rules with only AND inputs can be seen in figure 7, after changing some rules to OR as seen in figure 17 above, the output changed. Looking at the figure, it states that if the distance is far OR the speed is slow, then the soft brakes must be applied regardless any of them.

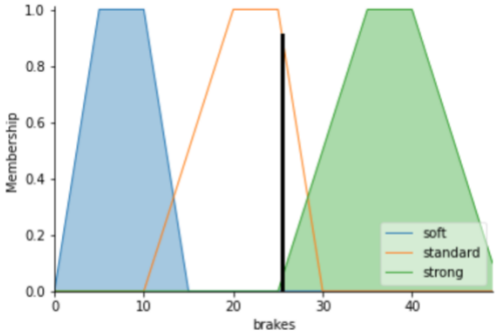
Looking at figure 18 below, the output of the rules in figure 7 and the output of the rules in figure 17 can be seen using the exact input numbers given to the system.

OR – AND Rules

```
braking.input['distance'] = 8
braking.input['speed'] = 80

braking.compute()
print(braking.output['brakes'])
brakes.view(sim=braking)
```

```
25.452237808951235
```



AND- RULES

```
braking.input['distance'] = 8
braking.input['speed'] = 80

braking.compute()
print(braking.output['brakes'])
brakes.view(sim=braking)
```

```
37.46042363433668
```

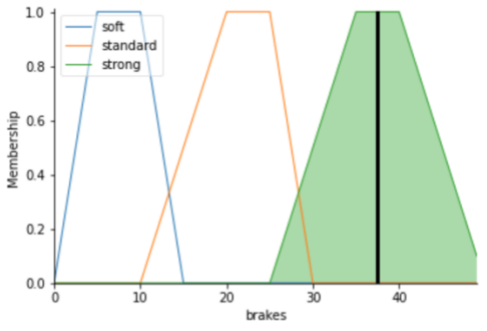


Figure 18: Comparing OR-AND rules output with only AND rules output from figure 7

The output in figure 18 above shows the difference between the output when using OR-AND rules and the output of AND rules from the rules given in figure 7 of this report. This shows the effects the change of rules has on the output of the fuzzy system. When using OR-AND rules it can be seen that there is fuzziness between the soft, standard, and strong brakes. The output of the AND rules have a membership of 1 with the strong breaks and some fuzziness – vagueness with the standard brakes.

Experiment 3:

This experiment is based on increasing the number of antecedents and visualise if the output of the system will be more accurate using more antecedents. Two more antecedents are added to the system. Those antecedents are:

- Weight: This antecedent checks for the weight of the vehicle for a range between 0 and 3000. Where 0 is lightest weight and 3000 is the heaviest weight.

- Road Conditions: This antecedent checks for the road condition where the vehicle is driving on, with a range between 0 and 10. Where 0 represents bad road condition and 10 represents the best road condition.

```
[27]  1 speed = ctrl.Antecedent(np.arange(0, 100, 1), 'speed')
      2 distance = ctrl.Antecedent(np.arange(0, 10, 1), 'distance')
      3 weight = ctrl.Antecedent(np.arange(0, 3000, 1), 'weight')
      4 road_condition = ctrl.Antecedent(np.arange(0, 10, 1), 'weight')
      5
      6 brakes = ctrl.Consequent(np.arange(0, 50, 1), 'brakes')
```

```
 1 speed['slow'] = fuzz.trapmf(speed.universe, [0,0,20,30])
 2 speed['average'] = fuzz.trapmf(speed.universe, [25, 45, 55, 60])
 3 speed['fast'] = fuzz.trapmf(speed.universe, [50, 80, 100, 100])
 4
 5 distance['far'] = fuzz.trapmf(distance.universe, [0,0,2,3])
 6 distance['equidistant'] = fuzz.trapmf(distance.universe, [2,4,5,6])
 7 distance['close'] = fuzz.trapmf(distance.universe, [5, 8, 9, 10])
 8
 9 weight['light'] = fuzz.trapmf(weight.universe, [0,0,1000,1200])
10 weight['midweight'] = fuzz.trapmf(weight.universe, [1000,1500,1700,2000])
11 weight['heavy'] = fuzz.trapmf(weight.universe, [1700, 2400, 2700, 3000])
12
13 road_condition['bad'] = fuzz.trapmf(road_condition.universe, [0,0,1,2])
14 road_condition['normal'] = fuzz.trapmf(road_condition.universe, [1,3,4,5])
15 road_condition['good'] = fuzz.trapmf(road_condition.universe, [3,7,8,10])
16
17 brakes['soft'] = fuzz.trapmf(brakes.universe, [0,5,10,15])
18 brakes['standard'] = fuzz.trapmf(brakes.universe, [10,20,25,30])
19 brakes['strong'] = fuzz.trapmf(brakes.universe, [25,35,40,50])
```

Figure 19: Initialising the variables for the new antecedents

Looking at figure 19 above, the new antecedents which are the weight, and the road condition are initialised. The new variables are represented visually in figure 1 of this report.

As new antecedents are added, those antecedents need to be initialised in the rules.

```
[64]  1 #Defining Rules
      2
      3 rule1 = ctrl.Rule(distance['far']  & speed['slow']  & weight['light'] & road_condition['good'], brakes['soft'])
      4 rule2 = ctrl.Rule(distance['far'] & speed['average'] & weight['light'] & road_condition['good'], brakes['standard'])
      5 rule3 = ctrl.Rule(distance['far']  & speed['fast'] & weight['light'] & road_condition['good'], brakes['strong'])
      6 rule4 = ctrl.Rule(distance['equidistant']  & speed['slow'] & weight['midweight'] & road_condition['normal'], brakes['soft'])
      7 rule5 = ctrl.Rule(distance['equidistant']  & speed['average'] & weight['midweight'] & road_condition['normal'], brakes['standard'])
      8 rule6 = ctrl.Rule(distance['equidistant'] & speed['fast'] & weight['midweight'] & road_condition['normal'], brakes['standard'])
      9 rule7 = ctrl.Rule(distance['close']  & speed['slow'] & weight['heavy'] & road_condition['bad'], brakes['soft'])
     10 rule8 = ctrl.Rule(distance['close'] & speed['average'] & weight['heavy'] & road_condition['bad'], brakes['standard'])
     11 rule9 = ctrl.Rule(distance['close'] & speed['fast'] & weight['heavy'] & road_condition['bad'], brakes['strong'])
```

Figure 20: New rules initialised after adding new antecedents

Looking at figure 20 above, the new antecedents are added as part of the rules. Therefore, the new antecedents need to receive an input value before computing the system. When running the program with the new rules, the error that can be seen in figure 14 appears. After researching the error, it was concluded that because there are four antecedents now more rules

must be applied, a set of 48 rules must be added where there are 12 rules must be applied. As this is a research problem, it is exhausting to be adding that many rules, it was conducted by research that the computation of the crisp output develops an output of zero as it can't compute the output with only 9 rules. The problem is understood and now the system works efficiently with only two antecedents as initialised.

After doing some research, it was concluded that the more antecedents, the merrier. When a system has more antecedents, it has more factors to depend on, therefore the output will be better and dealing with fuzziness and vagueness would be better [3].

Experiment 4:

The fourth and the final experiment includes changing the defuzzification method. There are different defuzzification methods applied on the system. The following defuzzification methods are applied:

- Centroid: The centroid defuzzification, also known as the centre of area method, is the most commonly used defuzzification method. It determines the centre of area of the fuzzy set and the return is the matching crisp value of the system. Where the similarities defined as clusters are known as centroid.

- Bisector: The bisector method treasures the line that divides the fuzzy set into different regions. All regions are given to be of equal area. It might interfere with the centroid line but not always.

- Mean of Maximum: This defuzzification is only for the active rules, meaning that all other rules which are not active are not taken into consideration, it checks for the highest degree of performance.

- Min of Maximum: This checks for the smallest output with the highest membership, the highest membership is usually 1. It chooses the smallest amount in the as an output with the most value of it happening. Looking at the graphs generated in this report, the Y-Axis maximum value is 1, this is defined as the highest membership.

- Max of Maximum: This checks for the largest amount in all the values, it defines the highest amount in the crisp values after it is generated. It comes out like all other defuzzification methods as a vertical line in the graph.

The output of all the defuzzification methods is presented as a vertical line for each, where the vertical line aligns under a trapezoid presented as the area under the curve, generating the crisp value using different defuzzification methods.

All the methods are presented in the system, the output gives all the outputs of all the defuzzification methods at once.

Each output of each defuzzification method is presented with a colour, where each colour reference is shown in a legend box on the left-top part of the graph.

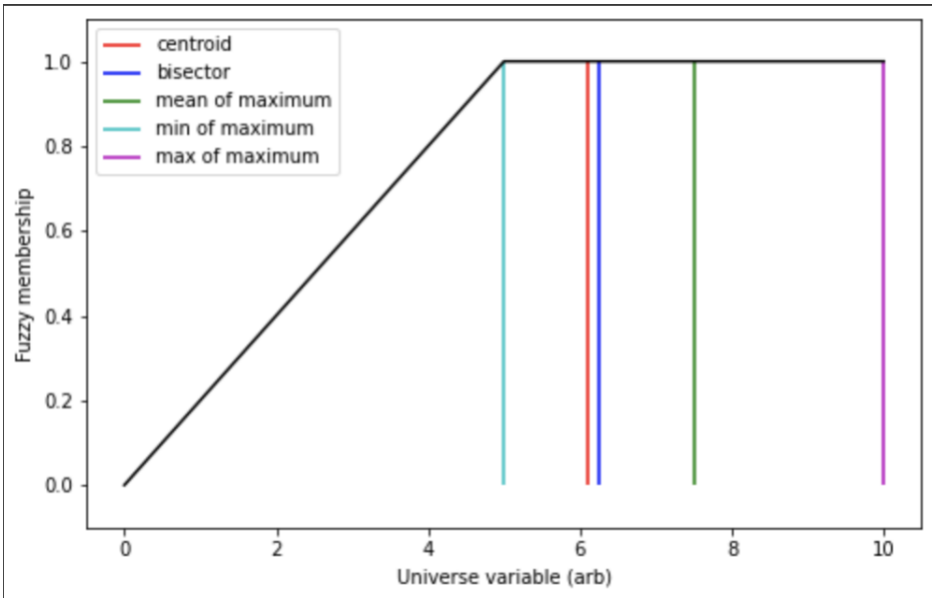


Figure 21: Output of all defuzzification methods

There are different areas when defining which is the best defuzzification method to be used, the most used defuzzification method is the centroid method. When defining which is the best defuzzification method used depends on the performance assessment, the model description, and the mathematical calculation of the method used. The Mamdani-type fuzzy inference system plays an important role as well as the Sugeno-type. The two-types of inference systems help determine the output. The Mamdani-type is very used as it expects the output membership to be fuzzy sets.

After doing the research, for the car braking system, the best method is the centroid method, as we return crisp logic from the fuzzy membership functions. As seen in figure 21, the centroid method vertical line highest.

1.3-

The main idea behind such system is to increase the safety dynamic when driving. The system in this project is based on the autonomous car braking system which adapts to the braking pressure using the distance from other vehicles and the speed. Playing important roles in this research are the fuzzy rules, the fuzzy rules given in the system mainly affects the output of the system. The system output was concluded and developed using different steps.

As the system depends on two antecedents which are the speed and the distance, the output is based on the inputs of those antecedents as well as the fuzzy rules provided. As there are only two antecedents, there are 9 different fuzzy rules applied to the system. In this project, fuzzy rules are defined to specify the logic mapping and the state for the control parameters of the system.

Analysing the results:

Firstly, discussing the different outputs when applying different inputs and rules. Looking at figure 10, it can be seen that the inputs of the system are 2 meters for the distance and 20 km/h, those values are specified in the code that the distance is far, and the speed is slow, as both those values are in the range presenting those outputs, those are defined using the lines below.

```
speed['slow'] = fuzz.trapmf(speed.universe, [0,0,20,30])
distance['far'] = fuzz.trapmf(distance.universe, [0,0,2,3])
```

It can be seen from the lines above that both the number representing the 2 meters for the distance and the 20 km/h for the speed are presented in the slow / far sets.

Linking the fuzzy rule provided for such scenario, the brakes must be applied softly, this is presented as rule 1 in the rules:

```
rule1 = ctrl.Rule(distance['far']  & speed['slow']  , brakes['soft'])
```

When applying those inputs to the program, the rule above is active to the system, therefore the output will be that the brakes are applied softly, this means that the output depends on the state of the input to the system and it also depends on the rate of change of this state as well as depending on the probability of the state of the input. The rate of change is displayed in the output. It can be seen in figure 11 that the action of the brakes being applied softly has a membership of 1, which means it has the highest probability of happening, but there is also a probability of 0.2 that the brakes might be applied in a standard pressure, that is where the soft and standard trapezoids meet.

In some cases, different rules might apply such as the distance of the vehicle is close to the vehicle ahead, but also the speed is average and not fast. Such situation is also presented in the system; by looking at figures 12 and 13. It can be seen that the distance in 8 meters, meaning that it is very close to the vehicle ahead, but the speed is average with 55 km/h. In such situation, the brakes do not need to be applied hard.

```
speed['average'] = fuzz.trapmf(speed.universe, [25, 45, 55, 60])
```

```
distance['close'] = fuzz.trapmf(distance.universe, [5, 8, 9, 10])
```

In such case, the output will have more vagueness depending on the rule provided for such situation. In the rules, it is specified that if the speed is average and the distance the close, the braking should be average, but also because the two inputs are from different sets and ranges, there will be vagueness and fuzziness and seen in figure 13.

```
rule8 = ctrl.Rule(distance['close'] & speed['average'] ,
brakes['standard'])
```

Rule 8 above presents such situation, the output in figure 13 might be changing the probability in the case of the input values changing whilst being in the same set, such as being at speed of 60 km/h instead of 55 km/h, although both values are presented as an average speed.

The rules defined for the system are mainly if-then rules, in order to explain the if then rules, they are represented is such form:

IF distance is far AND speed is slow THEN (apply soft brakes)
IF distance is equidistant AND speed is average THEN (apply standard brakes)
IF distance is close AND speed is fast THEN (apply strong brakes)

The form of rules above explains the rules that can be seen in figure 17. Another form of rules that was tested as part of the experiments was explaining the OR rules, such rules are also defined in a way such that:

IF distance is far OR speed is slow THEN (apply soft brakes)
IF distance is equidistant OR speed is average THEN (apply standard brakes)
IF distance is close OR speed is fast THEN (apply strong brakes)

When applying the OR rules on the system, it means that for example, if the distance is far OR the speed is slow, not the two of them are a must like in the AND rules, then the consequent should apply. Using this form of rules changes the results and fuzziness of the system like seen in figure 18 with the comparison. In a system such as the autonomous braking, when the AND rules are provided, a sustainable output is produced meeting the requirements of the system.

As seen in experiments 3, adding more antecedents to the system makes the system exhaustive as concluded by research, makes the system in need of many other rules. When applying new rules to the system in order to include new antecedents such as weight and road condition as well as the system will be in need of more inputs, as the system included many 4 inputs, it was not able to compute the output because of some rules were missed. It is concluded that a system might have better results with more antecedents, but it is more exhaustive and would require more rules, calculations, and might decrease the computational efficiency.

As by research, there are different defuzzification methods, which were provided and listed in experiment 4. All the methods were implemented and could be seen in the output of figure 21, unlike the fuzzification which converts the crisp quantity to a fuzzy quantity, the defuzzification does the opposite. After the research has been done, it is concluded that the Centroid defuzzification method which is represented in red in figure 21 is the most used method and the best method as it determines the centre of area of the fuzzy set, where the total area of the membership function is used to represent the combined full action depending on the fuzzy rules provided, the area is calculated and finds the value of the discrete set.

Task 2:

Different systems that are used daily are demonstrated using numerical methods, this is known as the quantitative modelling, it deals with discrete values by modelling the system mathematically [primarily using differential equations and calculus to describe the world]. Systems that are modelled using quantitative modelling like modelling the rate of flow in a tank, modelling the current flow in an electronic circuit, or measuring the success of a business campaign or calculating orbital mechanics for spacecrafts. In order to model a system quantitatively, the laws of maths and physics, are used. for example, Newton's laws of motion can facilitate the calculation of the trajectories of projectiles only if all the exact numerical values are available such as the mass, air resistance, and initial velocity. Such models are as accurate and useful as the accuracy of the numerical information they depend on. However, in numerous systems and applications, exact values cannot be known in order to facilitate modelling a system, for example modelling environmental and biological systems often suffers from incomplete knowledge or uncertainty in the underlying parameters, such systems rarely use the exact numerical information and differential equations. That is why Qualitative Reasoning is introduced.

Qualitative reasoning systematises reasoning about the physical world, it enables us to model systems where our knowledge is incomplete numerically. It also offers the ability for A.I systems to infer knowledge about the outside world which is a cornerstone of A.I technology, qualitative reasoning has been a fundamental research topic in Artificial Intelligence as it has been implemented in different applications such as autonomous space-craft support, failure analysis of vehicle systems, and real-time monitoring and hazard identification.

Qualitative reasoning has many benefits, such benefits include:

- It can deal with incomplete knowledge
- Produces qualitative predictions as opposed to exact numerical outputs
- Enables easy explanation of alternatives
- Provides an automatic interpretation of results

Qualitative reasoning can be classified into two main approaches such are *Qualitative Physics* and *Qualitative Spatial and Temporal Reasoning*.

Qualitative Physics focuses on automating the reasoning on the physical world, it mainly deals with physical systems where the information is incomplete, where the main goal of qualitative physics is to apprehend both the common-sense knowledge and the quantitative knowledge. Applying qualitative physics will result in better outcomes and analysis for specific systems. All computer programs that are implemented to adjust physical systems use qualitative physics, it makes interpretations of the real-world and some assumptions in order to represent and predict the future state. Such systems can generate if-then rules, for example:

IF temperature of room is low
AND humidity is high
THEN open the Air-Conditioner

The terms, low, normal, and high are qualitative terms, terms are sensible linguistically to humans. By implementing such terms in a computer program, it is simpler to preserve the vital distinctions without invoking differential equations and mathematics [4].

Qualitative Spatial and Temporal Reasoning has widely emerged in the research field in the past decade, it is based on temporal and spatial aspects, where spatial refers to space, and temporal refers to time. It combines both aspects to solve problems that consist of many steps by envisioning how the objects are active in space and time. An example is the use of Geographical Information System, where the surroundings are represented by conceptual domains which are continuous and discrete. An example is that a person is in a car, there is a traffic jam, the person decides that he will leaving at the next exit and take a different road. This is spatial temporal reasoning, as the reasoning included space and time based on real-world surroundings.

Qualitative reasoning has different features, where some of them are used to develop theories for specific applications and systems, those features are:

- Ontologies: This features states that qualitative reasoning concludes a representation of a formation modelling layer such as devising a plan for the system, not only represent a mathematical equation to model the system, but many models also that are built count on the ontologies in order to automate the system.

- Causality: This feature works on analysing the performance of a system using the cause-effect reasoning, it is a method that proves or denies that an action causes another – where the effect always follows the cause, an example of the causality feature is when told that A causes B, and B causes C, then it can be concluded that action A causes effect C.it has been used widely in qualitative reasoning

- Compositional Modelling: It is mainly used in industrial applications, it uses the modelling assumptions in order to provide qualitative representations, this can support the formulation of models being automatically generated and can also allow re-using specific models – In factory for example, models can be automated and re-used in order to model the output of a system qualitatively as long as its in the same field [5].

Recently, Ross King, professor in Machine Intelligence, used qualitative reasoning to design a robot scientist that is able to generate and test thousands of hypotheses autonomously. The findings of the robot scientists he and his team built were validated to be correct and of substances by human scientists. Indeed, futuristically one can imagine that robot scientists will be driving the discovery of new knowledge.

Qualitative reasoning is still a huge research topic in Artificial Intelligence, it mainly focuses on modelling physical systems. The example mentioned earlier about the robot designed by Dr.Ross King is one of the latest inventions in qualitative reasoning which will open many doors for the research area in such topic, in order for qualitative reasoning to move forward, it needs to model real-life settings and develop representation of human activity.

References:

[1] C.Moraga. 2005. “Introduction to Fuzzy Logic”. [Online]. Available at: http://www.doiserbia.nb.rs/img/doi/0353-3670/2005/0353-36700502319M.pdf [Accessed 3 March 2021].

[2] K. Uplenshwar, R.Kokate. 2020. “Applications of Fuzzy Logic”. [Online]. Available at: https://www.irjet.net/archives/V7/i4/IRJET-V7I4399.pdf [Accessed 3 March 2021]

[3] A.Hassan, S.A. Hammad. 1999. “Proposed Tuning tips for fuzzy controllers using offline and online experimental aids”. [Online]. Available at: https://asat.journals.ekb.eg/article_25165_b75efcbd317d4d6c3e2287cc054a9eee.pdf [Accessed 6 March 2021]

[4] S.Grantham. “Qualitative Physics”. [Online]. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.154.9031&rep=rep1&type=pdf [Accessed 6 March 2021]

[5] Forbus. K, Falkenhainer. B. “Setting up large scale qualitative models”. [Online]. Available at: https://www.qrg.northwestern.edu/papers/Files/QRG_Dist_Files/QRG_1988/Falkenhainer_1988_Setting_up_Qualitative_Models.pdf [Accessed 7 March 2021]