Name – Keshari S L

Index- 15743

Degree Program – Bioinformatics Special

Fuzzy inference system of the self-driving car (Mega) to navigate through a simulated urban environment – Report

Introducing Mega

Mega is a self-driving car which is developed to do its driving process without a driver by its own intelligence. Mega is equipped with an intelligent decision-making system powered by a fuzzy inference system. This system enables Mega to navigate seamlessly through a complex and dynamic simulated urban environment and making real-time decisions based on a combination of inputs.

Mega is developed to **maintain its own speed according to the speed limit of the urban area** which a vehicle can have within that area.

And to **avoid an obstacle presence in front**.

These inputs are obtained by car sensors from the road signs.

Step 1- Define Input Variables

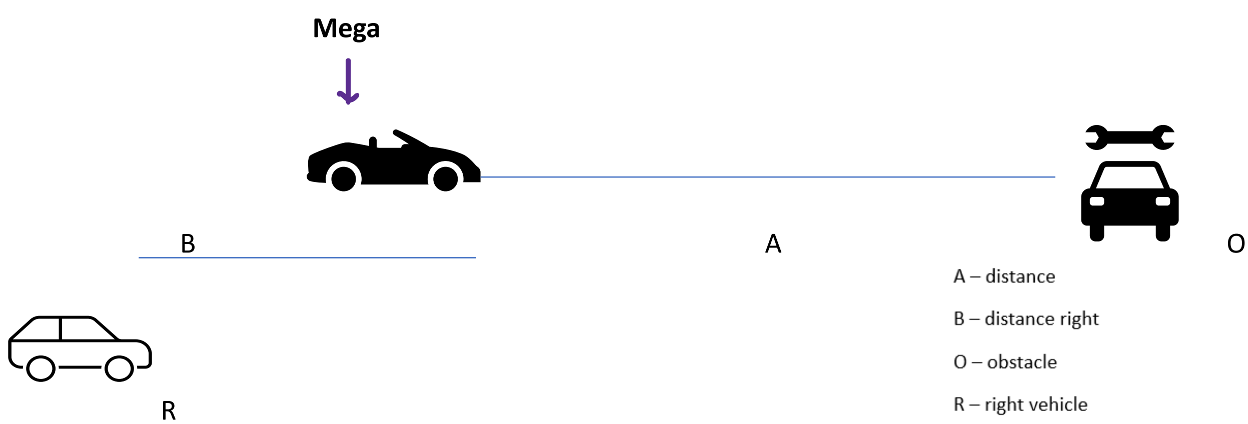
speed - This is current speed of the Mega

speed limit - maximum speed which a vehicle can go in that particular area

distance - distance to the obstacle when mega detecting it (maximum distance mega able to detect an obstacle is 100m)

distance left – Distance to the vehicle on the left side. This distance is measured by mega from its front line to back. The maximum is 10m. If close vehicle is present mega will not steer, if no vehicle presence or vehicle presence in 10m   it will steer left.

distance right –   Distance to the vehicle in right side



Jupyter Untitled2 Last Checkpoint: an hour ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help Trusted Python [conda env:my_env] *

Run Code

In [244]:
```python
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

In [246]:
```python
#this fuzzy system is defined to avoid a sudden obstacle appear in front cars' current track like pedestrians
                            #and other vehicles, stationary objects like road signs , barriers and etc.

#input variables
#this inputs are obtain by the advance sensors of the self driving car and they are taking input within
                                                              # a optimum range which they can be accessed

#current speed of the self-driving car km per hour
#maximum speed of the car is 100 km per hour
speed = ctrl.Antecedent(np.arange(0,101,1),'speed')

#speed limit of the current location
speed_limit = ctrl.Antecedent(np.arange(0,101,1),'speed_limit')

# distance to the sudden obstacle in meter
#the car sensors can recognize a obstacle in front is within a maximum distance of 100m (assuming road is a straight line)
distance = ctrl.Antecedent(np.arange(0,101,1),'distance')

# distance to vehicle in right side
# The car sensors recognize a another vehicle in right side is within a maximum distance of 10m  from cars front line
distance_left = ctrl.Antecedent(np.arange(0,11,1),'distance_left')

# distance to vehicle in left side
# The car sensors recognize a another vehicle in left side is within a maximum distance of 10m  from cars front line
distance_right = ctrl.Antecedent(np.arange(0,11,1),'distance_right')
```

```python
#input membership functions
speed['slow'] = fuzz.trimf(speed.universe, [0,0,50])
speed['moderate'] = fuzz.trimf(speed.universe, [0,50,100])
speed['fast'] = fuzz.trimf(speed.universe, [50,100,100])

speed_limit['slow'] = fuzz.trimf(speed_limit.universe, [0,0,50])
speed_limit['moderate'] = fuzz.trimf(speed_limit.universe, [0,50,100])
speed_limit['fast'] = fuzz.trimf(speed_limit.universe, [0,100,100])

distance['very_close'] = fuzz.trimf(distance.universe, [0,0,10])
distance['close'] = fuzz.trimf(distance.universe, [0,25,100])
distance['moderate'] = fuzz.trimf(distance.universe, [0,50,100])
distance['far'] = fuzz.trimf(distance.universe, [50,100,100])

distance_left['close'] = fuzz.trimf(distance_left.universe, [0,0,6]) #risky distnce, steering caused to an accident
distance_left['moderate'] = fuzz.trimf(distance_left.universe, [0,5,10])
distance_left['far'] = fuzz.trimf(distance_left.universe, [4, 10 ,10]) #non risky, enough distance to steer

distance_right['close'] = fuzz.trimf(distance_right.universe, [0, 0, 6 ])
distance_right['moderate'] = fuzz.trimf(distance_right.universe, [0, 5,10 ])
distance_right['far'] = fuzz.trimf(distance_right.universe, [4, 10, 10])
```

Step 1- Define output Variables and membership functions

brake – stop mega (high brake) or reduce the current speed (low brake)

horn – warning obstacle by horning

steering_action – shift the mega to left or right side to avoid the obstacle.

adjust_speed - increase or decrease speed of mega by accelerating or decelerating.

```
#Defining output variables
#Brake
brake = ctrl.Consequent(np.arange(0,101,1), 'brake')
# warning to the obstacle by horn (pedestrian or other vehicle or animal or etc)
horn = ctrl.Consequent(np.arange(0,2,1), 'horn')
# Car's action (50: Maintain , 0: steer Left side and  100: steer Right side)
# the car is sheduled to the correct angle to steer acoording this output (Eg- if output=50 angle=0 degrees , if output=0 angle =
steering_action = ctrl.Consequent (np.arange(0,101,1), 'steering_action')
#speed adjustment
adjust_speed = ctrl.Consequent (np.arange(-20,21,1), 'adjust_speed')

#Define membership functions for output variables
brake['low'] = fuzz.trimf(brake.universe,[0,0,50])
brake['medium'] = fuzz.trimf(brake.universe,[10,50,90])
brake['high'] = fuzz.trimf(brake.universe,[50,100,100])

horn['yes'] = fuzz.trimf(horn.universe,[0,1,1])
horn['no'] = fuzz.trimf(horn.universe,[0,0,1])

steering_action ['steer_left'] = fuzz.trimf(steering_action.universe, [0, 0, 50])
steering_action ['maintain'] = fuzz.trimf(steering_action.universe, [0,50,100])
steering_action ['steer_right'] = fuzz.trimf(steering_action.universe, [50,100,100])


adjust_speed['decelerate'] = fuzz.trimf(adjust_speed.universe, [-20, -20, 0])
adjust_speed['maintain'] = fuzz.trimf(adjust_speed.universe, [-20, 0, 20])
adjust_speed['accelerate'] = fuzz.trimf(adjust_speed.universe, [0, 20, 20])
```

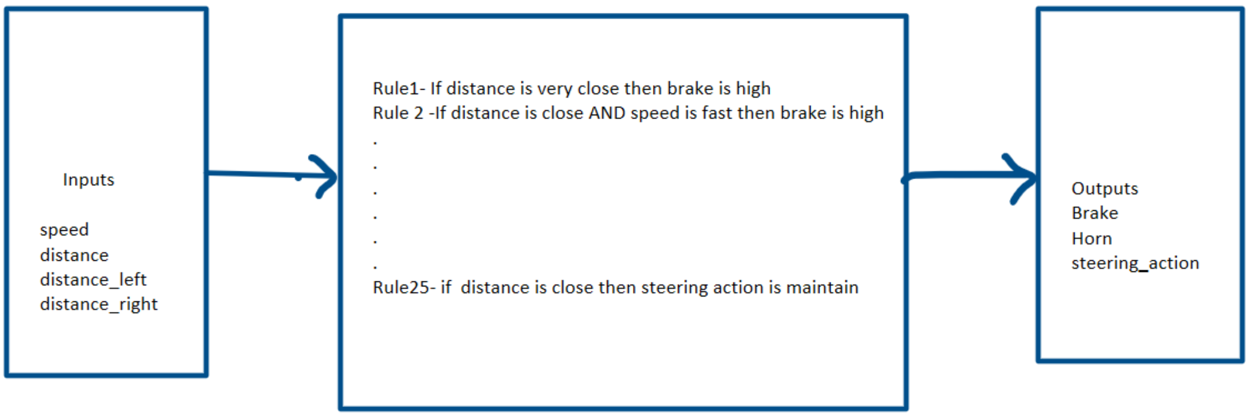
Fuzzy rules and Implement Fuzzy inference (obstacle avoidance)

In [259]:

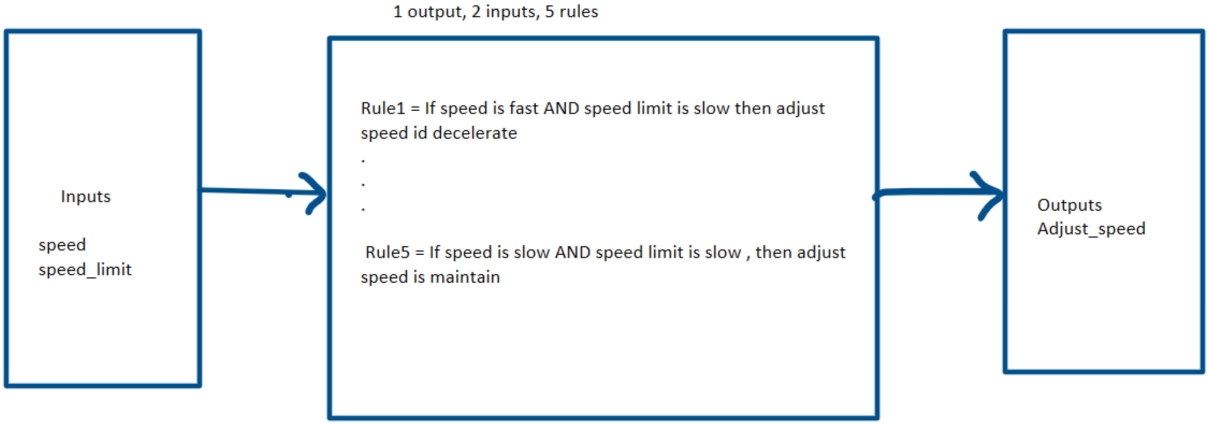
```
#Defining the fuzzy rules
rule1 = ctrl.Rule(distance['very_close']    , brake['high'])
rule2 = ctrl.Rule(distance['close'] & speed['fast']    , brake['high'])
rule3 = ctrl.Rule(distance['close'] & speed['moderate']    , brake['medium'])
rule4 = ctrl.Rule(distance['close'] & speed['slow']    , brake['low'])
rule22 = ctrl.Rule(distance['far']    , brake['low'])
rule5 = ctrl.Rule(distance['moderate'] & speed['fast'] & distance_right['far'] & distance_left['far'], steering_action ['steer_le
rule6 = ctrl.Rule(distance['moderate'] & speed['fast'] & distance_right['far'] & distance_left['close'], steering_action ['steer_
rule7 = ctrl.Rule(distance['moderate'] & speed['fast'] & distance_right['close'] & distance_left['far'], steering_action ['steer_
rule8 = ctrl.Rule(distance['moderate'] & speed['fast'] & distance_right['close'] & distance_left['close'], steering_action ['mair
rule9 = ctrl.Rule(distance['moderate'] & speed['moderate'] & distance_right['far'] & distance_left['far'], steering_action ['stee
rule10= ctrl.Rule(distance['moderate'] & speed['moderate'] & distance_right['far'] & distance_left['close'], steering_action ['st
rule11 = ctrl.Rule(distance['moderate'] & speed['moderate'] & distance_right['close'] & distance_left['far'], steering_action ['s
rule12 = ctrl.Rule(distance['moderate'] & speed['moderate'] & distance_right['close'] & distance_left['close'], steering_action [
rule13 = ctrl.Rule(distance['moderate'] & speed['slow'] & distance_right['far'] & distance_left['far'], steering_action ['steer_l
rule14 = ctrl.Rule(distance['moderate'] & speed['slow'] & distance_right['far'] & distance_left['close'], steering_action ['steer
rule15 = ctrl.Rule(distance['moderate'] & speed['slow'] & distance_right['close'] & distance_left['far'], steering_action ['steer
rule16 = ctrl.Rule(distance['moderate'] & speed['slow'] & distance_right['close'] & distance_left['close'], steering_action ['mai
rule17 = ctrl.Rule(distance['far'] ,horn['yes'] )
rule23 = ctrl.Rule(speed['fast']  ,horn['yes'] )
rule24 = ctrl.Rule(distance['close'],horn['yes'] )
rule18 = ctrl.Rule(distance['far'] & distance_right['close'] & distance_left['close'], steering_action ['maintain'] )
rule19 = ctrl.Rule(distance['far'] & distance_right['far'] & distance_left['far'], steering_action ['maintain'] )
rule20 = ctrl.Rule(distance['far'] & distance_right['close'] & distance_left['far'], steering_action ['maintain'] )
rule21 = ctrl.Rule(distance['far'] & distance_right['far'] & distance_left['close'], steering_action ['maintain'] )

#creating control system and simulation
system = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9, rule10, rule11, rule12, rule13, rule1
simulation = ctrl.ControlSystemSimulation(system)

#Set input values obtained by self driving car sensors
simulation.input['speed'] =90  # speed km/h
simulation.input['distance'] =5 #meters
simulation.input['distance_left'] = 1  #in meters to left vehicle
simulation.input['distance_right'] = 2 #in meters to right vehicle

simulation.compute()

print("Brake:", simulation.output['brake'])
print("Horn:", simulation.output['horn']) #if output value<0.5; horn = no , if output value>0.5; horn = yes
print("Steering_action :", simulation.output['steering_action']) #  if output=50 steering_action is maintain (No steer) ; if outp
```

```
Brake: 66.55349412492268
Horn: 0.6555555555555554
Steering_action : 50.00000000000009
```

3 outputs, 4 inputs, 25 rules



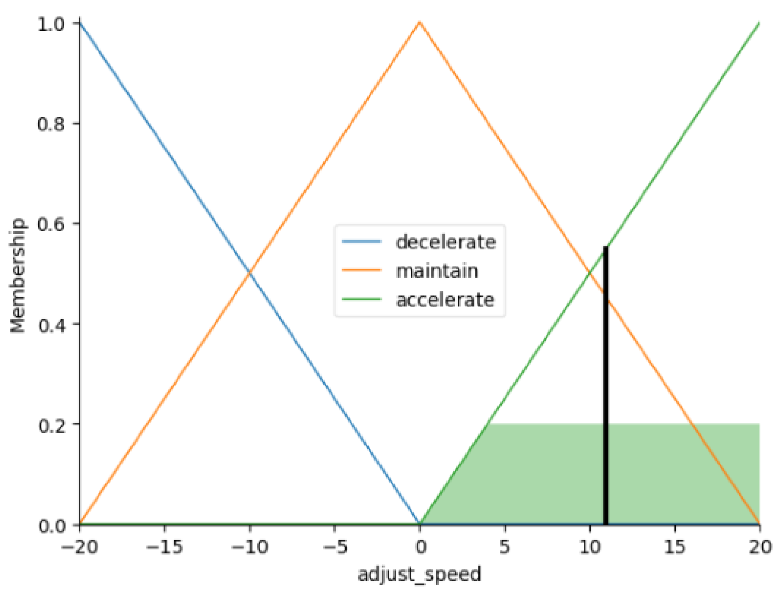
1 output, 2 inputs, 5 rules



Fuzzy rules and Implement Fuzzy inference (Maintain Speed)

In [273]:
```
rule26 = ctrl.Rule(speed['fast'] & speed_limit['slow']  , adjust_speed['decelerate'] )
rule27 = ctrl.Rule(speed['slow'] & speed_limit['fast']  , adjust_speed['accelerate'] )
rule28 = ctrl.Rule(speed['moderate'] & speed_limit['moderate']  , adjust_speed['maintain'] )
rule29 = ctrl.Rule(speed['fast'] & speed_limit['fast']  , adjust_speed['maintain'] )
rule30 = ctrl.Rule(speed['slow'] & speed_limit['slow']  , adjust_speed['maintain'] )

fis_ctrl = ctrl.ControlSystem([rule26, rule27, rule28, rule29, rule30])
fis = ctrl.ControlSystemSimulation(fis_ctrl)

#Input values as per the scenario
fis.input['speed'] = 40  #Example value 40 km/h
fis.input['speed_limit'] =100

#perform fuzzy inference
fis.compute()
#output result
print(fis.output['adjust_speed'])

#view
adjust_speed.view(sim=fis)
```

```
10.96296296296296
```



Analyzing the Megas' performance.

This car is developed to detect an obstacle in front within 100 m distance.

Scenario1 = Mega detects an obstacle in front it in a far distance.

If Mega detects an obstacle in front it 90m (far) ?

If Mega detects obstacle too far **horn to take off the obstacle** and go towards. When detecting an obstacle in far distance it will **not go to give high** break or steer for its efficiency. It will try to remove obstacle by horning.

```
rule17 = ctrl.Rule(distance['far'] ,horn['yes'] )
rule23 = ctrl.Rule(speed['fast']  ,horn['yes'] )
rule24 = ctrl.Rule(distance['close'],horn['yes'] )
rule18 = ctrl.Rule(distance['far'] & distance_right['close'] & distance_left['close'], steering_action ['maintain'] )
rule19 = ctrl.Rule(distance['far'] & distance_right['far'] & distance_left['far'], steering_action ['maintain'] )
rule20 = ctrl.Rule(distance['far'] & distance_right['close'] & distance_left['far'], steering_action ['maintain'] )
rule21 = ctrl.Rule(distance['far'] & distance_right['far'] & distance_left['close'], steering_action ['maintain'] )

#creating control system and simulation
system = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9, rule10, rule11, rule12, rule13, rule1
simulation = ctrl.ControlSystemSimulation(system)

#Set input values obtained by self driving car sensors
simulation.input['speed'] = 45  # speed km/h
simulation.input['distance'] = 90  #meters
simulation.input['distance_left'] = 8  #in meters to left vehicle
simulation.input['distance_right'] = 2 #in meters to right vehicle

simulation.compute()

print("Brake:", simulation.output['brake'])
print("Horn:", simulation.output['horn']) #if output value<0.5; horn = no , if output value>0.5; horn = yes
print("Steering_action :", simulation.output['steering_action']) #  if output=50 steering_action is maintain (No steer) ; if outp

Brake: 26.394037039834217
Horn: 0.6555555555555554
Steering_action : 48.97310513447433
```

If **obstacle still appear when mega reach to moderate distance** of the same obstacle.

Will it accident?

No, when reach to moderate distance if **obstacle is still remaining** and **present vehicle** at least one of the sides (left or right) **not close to mega**. Mega **will steer to avoid the obstacle**. Not give high brake to stop. For efficiency. But try to avoid obstacle by steering. For the safety of all parties.

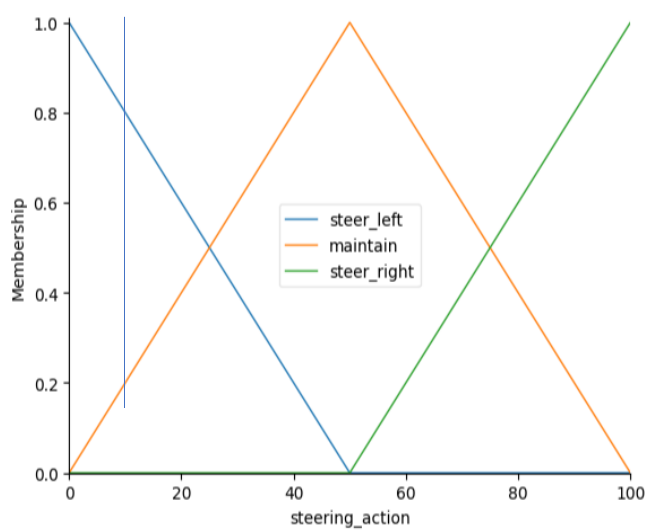
```
rule24 = ctrl.Rule(distance['close'],horn['yes'] )
rule18 = ctrl.Rule(distance['far'] & distance_right['close'] & distance_left['close'], steering_action ['maintain'] )
rule19 = ctrl.Rule(distance['far'] & distance_right['far'] & distance_left['far'], steering_action ['maintain'] )
rule20 = ctrl.Rule(distance['far'] & distance_right['close'] & distance_left['far'], steering_action ['maintain'] )
rule21 = ctrl.Rule(distance['far'] & distance_right['far'] & distance_left['close'], steering_action ['maintain'] )

#creating control system and simulation
system = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9, rule10, rule11, rule12, rule13, rule1
simulation = ctrl.ControlSystemSimulation(system)

#Set input values obtained by self driving car sensors
simulation.input['speed'] = 45  # speed km/h
simulation.input['distance'] = 50  #meters
simulation.input['distance_left'] = 9  #in meters to left vehicle
simulation.input['distance_right'] = 1 #in meters to right vehicle

simulation.compute()

print("Brake:", simulation.output['brake'])
print("Horn:", simulation.output['horn']) #if output value<0.5; horn = no , if output value>0.5; horn = yes
print("Steering_action :", simulation.output['steering_action']) #  if output=50 steering_action is maintain (No steer) ; if outp

Brake: 48.56529625151143
Horn: 0.6388888888888888
Steering_action : 17.06349206349206
```

Give **low break (48.56).** As low break applies in steering **but not stop**(efficient)

The steering action is steer left (17.06). As left side vehicle is far to mega (9). **Very safe & efficient.**

No accident with right side vehicle and avoid obstacle at moderate distance by steering left.

```
In [173]: steering_action.view()
```



But if vehicles are presence very close to mega at both sides. Will Mega accident in obstacle?

No. It will maintain current steer action and go forward with a low break or high break according to the closeness of distance and current speed of Mega. Safe no accidents happen. This may be a high break to stop or may be low break to reduce speed to stop near to obstacle.

```
rule9 = ctrl.Rule(distance['moderate'] & speed['moderate'] & distance_right['far'] & distance_left['far'], steering_action ['stee
rule10= ctrl.Rule(distance['moderate'] & speed['moderate'] & distance_right['far'] & distance_left['close'], steering_action ['st
rule11 = ctrl.Rule(distance['moderate'] & speed['moderate'] & distance_right['close'] & distance_left['far'], steering_action ['s
rule12 = ctrl.Rule(distance['moderate'] & speed['moderate'] & distance_right['close'] & distance_left['close'], steering_action [
rule13 = ctrl.Rule(distance['moderate'] & speed['slow'] & distance_right['far'] & distance_left['far'], steering_action ['steer_l
rule14 = ctrl.Rule(distance['moderate'] & speed['slow'] & distance_right['far'] & distance_left['close'], steering_action ['steer
rule15 = ctrl.Rule(distance['moderate'] & speed['slow'] & distance_right['close'] & distance_left['far'], steering_action ['steer
rule16 = ctrl.Rule(distance['moderate'] & speed['slow'] & distance_right['close'] & distance_left['close'], steering_action ['mai
rule17 = ctrl.Rule(distance['far'] ,horn['yes'] )
rule23 = ctrl.Rule(speed['fast']  ,horn['yes'] )
rule24 = ctrl.Rule(distance['close'],horn['yes'] )
rule18 = ctrl.Rule(distance['far'] & distance_right['close'] & distance_left['close'], steering_action ['maintain'] )
rule19 = ctrl.Rule(distance['far'] & distance_right['far'] & distance_left['far'], steering_action ['maintain'] )
rule20 = ctrl.Rule(distance['far'] & distance_right['close'] & distance_left['far'], steering_action ['maintain'] )
rule21 = ctrl.Rule(distance['far'] & distance_right['far'] & distance_left['close'], steering_action ['maintain'] )

#creating control system and simulation
system = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9, rule10, rule11, rule12, rule13, rule1
simulation = ctrl.ControlSystemSimulation(system)

#Set input values obtained by self driving car sensors
simulation.input['speed'] = 45  # speed km/h
simulation.input['distance'] = 20  #meters
simulation.input['distance_left'] = 1 #in meters to left vehicle
simulation.input['distance_right'] = 1 #in meters to right vehicle

simulation.compute()

print("Brake:", simulation.output['brake'])
print("Horn:", simulation.output['horn']) #if output value<0.5; horn = no , if output value>0.5; horn = yes
print("Steering_action :", simulation.output['steering_action']) #  if output=50 steering_action is maintain (No steer) ; if outp
```

```
Brake: 48.6683501683501
Horn: 0.6555555555555554
Steering_action : 50.00000000000004
```

This can be differ with the speed and distance. Can be shown below.

```
rule24 = ctrl.Rule(distance['close'],horn['yes'] )
rule18 = ctrl.Rule(distance['far'] & distance_right['close'] & distance_left['close'], steering_acti
rule19 = ctrl.Rule(distance['far'] & distance_right['far'] & distance_left['far'], steering_action [
rule20 = ctrl.Rule(distance['far'] & distance_right['close'] & distance_left['far'], steering_action
rule21 = ctrl.Rule(distance['far'] & distance_right['far'] & distance_left['close'], steering_action

#creating control system and simulation
system = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9, rule10,
simulation = ctrl.ControlSystemSimulation(system)

#Set input values obtained by self driving car sensors
simulation.input['speed'] = 90  # speed km/h
simulation.input['distance'] = 3  #meters
simulation.input['distance_left'] = 1 #in meters to left vehicle
simulation.input['distance_right'] = 1 #in meters to right vehicle

simulation.compute()

print("Brake:", simulation.output['brake'])
print("Horn:", simulation.output['horn']) #if output value<0.5; horn = no , if output value>0.5; hor
print("Steering_action :", simulation.output['steering_action']) #  if output=50 steering_action is
```

```
Brake: 73.45458450993172
Horn: 0.6555555555555554
Steering_action : 50.000000000000085
```

Adaptability – also high according to the situation. Even if it has no way to be safe at all distance finally at close distance it will be safe. Will give priority to safe of Mega and other vehicles as well as the obstacle.

Scenario 2 – Mega detects obstacle suddenly in very close distance in a very high speed.

```
#Set input values obtained by self driving car sensors
simulation.input['speed'] = 100  # speed km/h
simulation.input['distance'] = 2 #meters
simulation.input['distance_left'] = 8 #in meters to left vehicle
simulation.input['distance_right'] = 7 #in meters to right vehicle

simulation.compute()

print("Brake:", simulation.output['brake'])
print("Horn:", simulation.output['horn']) #if output value<0.5; horn = no , if output value>0.5; horn = yes
print("Steering_action :", simulation.output['steering_action']) #  if output=50 steering_action is maintain (No steer) ; if outp
```

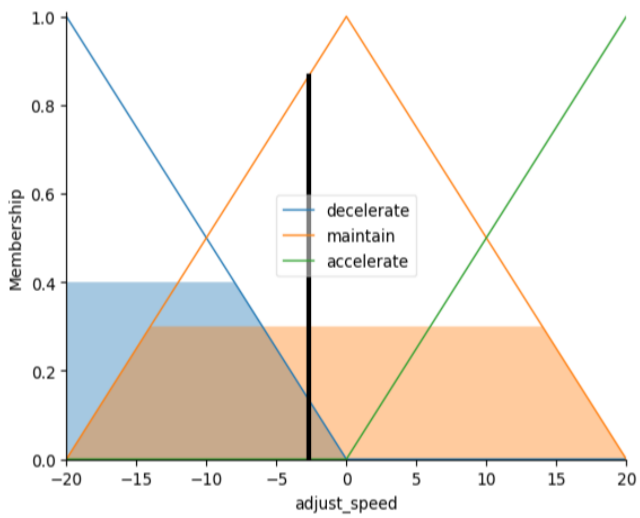
```
Brake: 82.77777777777777
Horn: 0.6666666666666666
Steering_action : 49.74438687392052
```

Will give a high break and stop. Mega is enough for more than 50 breaks to completely stop within 0.7 seconds. Under 82 break it will definitely stop less than 2m. Safety is secured. High safety and adaptability to the condition.

Scenario 3 – Current area maximum speed limit is 30 km/h. Mega speed is 90 km/h.

```
fis_ctrl = ctrl.ControlSystem([rule18, rule19, rule20, rule21, rule22])
fis = ctrl.ControlSystemSimulation(fis_ctrl)

#Input values as per the scenario
fis.input['speed'] =90  #Example value 40 km/h
fis.input['speed_limit'] = 30

#perform fuzzy inference
fis.compute()
#output result
print(fis.output['adjust_speed'])

#view
adjust_speed.view(sim=fis)
```

```
-2.7204301075268806
```

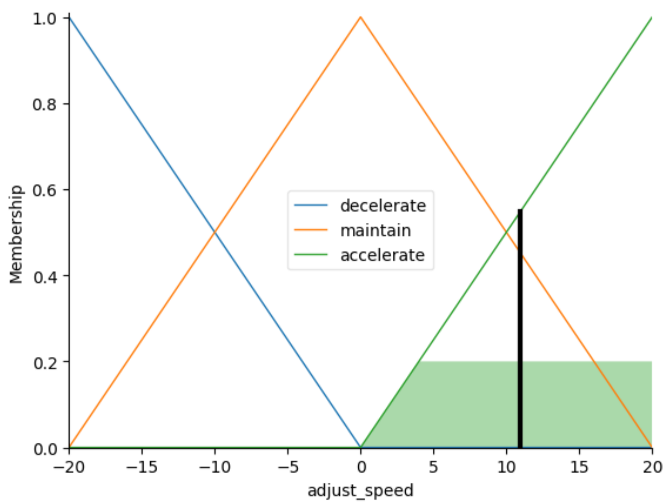


Mega decelerates by -2.72. And obtain the relevant speed limit. Will not caught by police and will not damage to pedestrians and other vehicles. Adapt to the specified speed. (This is adaptability of Mega, refers to its ability to adjust and respond appropriately to changes in the environment and road conditions)

Scenario 4 – Current area maximum speed limit is 80 km/h. Mega speed is 40 km/h.

```
fis_ctrl = ctrl.ControlSystem([rule18, rule19, rule20, rule21, rule22])
fis = ctrl.ControlSystemSimulation(fis_ctrl)

#Input values as per the scenario
fis.input['speed'] = 40  #Example value 40 km/h
fis.input['speed_limit'] =100

#perform fuzzy inference
fis.compute()
#output result
print(fis.output['adjust_speed'])

#view
adjust_speed.view(sim=fis)
```

```
10.96296296296296
```



Mega accelerates by 10.96. And obtain the maximum speed limit. Will keep its efficiency by maintaining maximum speed in possible areas. Adapt to the specified speed. (This is the adaptability of Mega, refers to its ability to adjust and respond appropriately to changes in the environment and road conditions). Mega will bring passengers to the destination safely and efficiently.

Whole Code in jupyter NoteBook

In [26]:

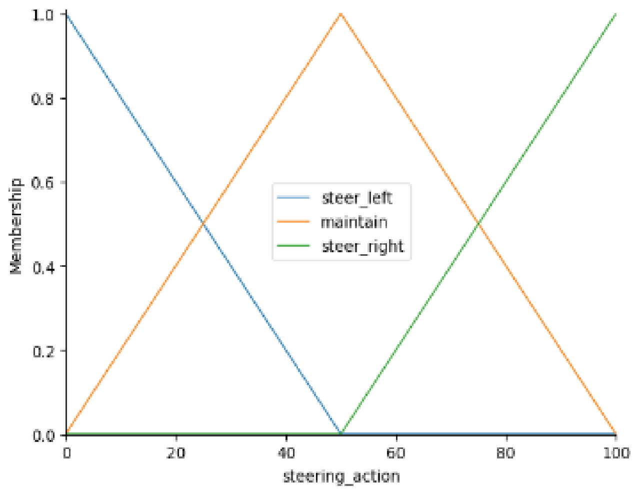
```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

In [27]:

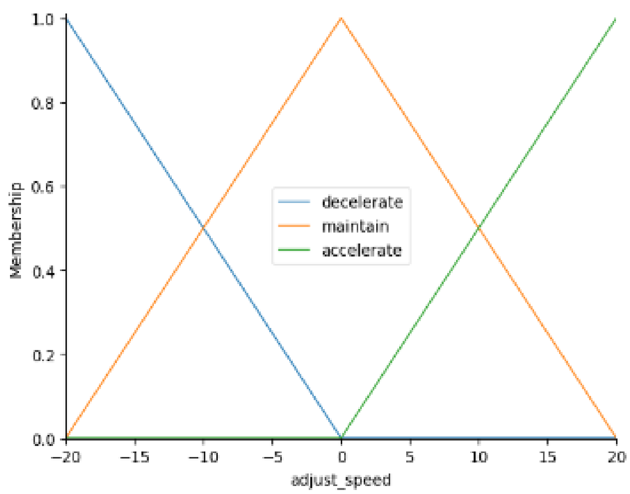
```
#this fuzzy system is defined to avoid a sudden obstacle appear in front cars' current track like pedestrians
                            #and other vehicles, stationary objects like road signs , barriers and etc.

#input variables
#this inputs are obtain by the advance sensors of the self driving car and they are taking input within
                                                             # a optimum range which they can be accessed

#current speed of the self-driving car km per hour
#maximum speed of the car is 100 km per hour
speed = ctrl.Antecedent(np.arange(0,101,1),'speed')

#speed limit of the current location
speed_limit = ctrl.Antecedent(np.arange(0,101,1),'speed_limit')

# distance to the sudden obstacle in meter
#the car sensors can recognize a obstacle in front is within a maximum distance of 100m (assuming road is a straight line)
distance = ctrl.Antecedent(np.arange(0,101,1),'distance')

# distance to vehicle in right side
# The car sensors recognize a another vehicle in right side is within a maximum distance of 10m  from cars front line
distance_left = ctrl.Antecedent(np.arange(0,11,1),'distance_left')

# distance to vehicle in left side
# The car sensors recognize a another vehicle in left side is within a maximum distance of 10m  from cars front line
distance_right = ctrl.Antecedent(np.arange(0,11,1),'distance_right')


#input membership functions
speed['slow'] = fuzz.trimf(speed.universe, [0,0,50])
speed['moderate'] = fuzz.trimf(speed.universe, [0,50,100])
speed['fast'] = fuzz.trimf(speed.universe, [50,100,100])

speed_limit['slow'] = fuzz.trimf(speed_limit.universe, [0,0,50])
speed_limit['moderate'] = fuzz.trimf(speed_limit.universe, [0,50,100])
speed_limit['fast'] = fuzz.trimf(speed_limit.universe, [0,100,100])

distance['very_close'] = fuzz.trimf(distance.universe, [0,0,10])
distance['close'] = fuzz.trimf(distance.universe, [0,25,100])
distance['moderate'] = fuzz.trimf(distance.universe, [0,50,100])
distance['far'] = fuzz.trimf(distance.universe, [50,100,100])

distance_left['close'] = fuzz.trimf(distance_left.universe, [0,0,6]) #risky distnce, steering caused to an accident
distance_left['moderate'] = fuzz.trimf(distance_left.universe, [0,5,10])
distance_left['far'] = fuzz.trimf(distance_left.universe, [4, 10 ,10]) #non risky, enough distance to steer

distance_right['close'] = fuzz.trimf(distance_right.universe, [0, 0, 6 ])
distance_right['moderate'] = fuzz.trimf(distance_right.universe, [0, 5,10 ])
distance_right['far'] = fuzz.trimf(distance_right.universe, [4, 10, 10])

#Defining output variables
#Brake
brake = ctrl.Consequent(np.arange(0,101,1), 'brake')
# warning to the obstacle by horn (pedestrian or other vehicle or animal or etc)
horn = ctrl.Consequent(np.arange(0,2,1), 'horn')
# Car's action (50: Maintain , 0: steer left side and  100: steer Right side)
# the car is sheduled to the correct angle to steer acoording this output (Eg- if output=50 angle=0 degrees , if output=0 angle
steering_action = ctrl.Consequent (np.arange(0,101,1), 'steering_action')
#speed adjustment
adjust_speed = ctrl.Consequent (np.arange(-20,21,1), 'adjust_speed')

#Define membership functions for output variables
brake['low'] = fuzz.trimf(brake.universe,[0,0,50])
brake['medium'] = fuzz.trimf(brake.universe,[10,50,90])
brake['high'] = fuzz.trimf(brake.universe,[50,100,100])

horn['yes'] = fuzz.trimf(horn.universe,[0,1,1])
horn['no'] = fuzz.trimf(horn.universe,[0,0,1])

steering_action ['steer_left'] = fuzz.trimf(steering_action.universe, [0, 0, 50])
steering_action ['maintain'] = fuzz.trimf(steering_action.universe, [0,50,100])
steering_action ['steer_right'] = fuzz.trimf(steering_action.universe, [50,100,100])


adjust_speed['decelerate'] = fuzz.trimf(adjust_speed.universe, [-20, -20, 0])
adjust_speed['maintain'] = fuzz.trimf(adjust_speed.universe, [-20, 0, 20])
adjust_speed['accelerate'] = fuzz.trimf(adjust_speed.universe, [0, 20, 20])
```

brake

In [33]:
```
steering_action.view()
```



In [34]:
```
adjust_speed.view()
```

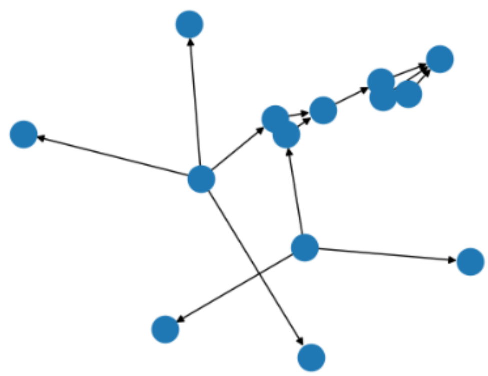


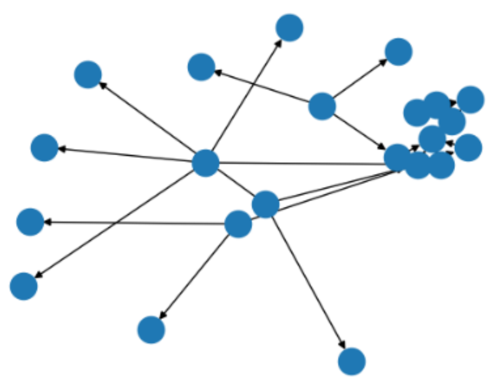
In [35]:
```
#Defining the fuzzy rules
rule1 = ctrl.Rule(distance['very_close']   , brake['high'])
rule2 = ctrl.Rule(distance['close'] & speed['fast']   , brake['high'])
rule3 = ctrl.Rule(distance['close'] & speed['moderate']   , brake['medium'])
rule4 = ctrl.Rule(distance['close'] & speed['slow']   , brake['low'])
rule22 = ctrl.Rule(distance['far']   , brake['low'])
rule25 = ctrl.Rule(distance['close']   , steering_action ['maintain'])
rule5 = ctrl.Rule(distance['moderate'] & speed['fast'] & distance_right['far'] & distance_left['far'], steering_action ['steer_le
rule6 = ctrl.Rule(distance['moderate'] & speed['fast'] & distance_right['far'] & distance_left['close'], steering_action ['steer_
rule7 = ctrl.Rule(distance['moderate'] & speed['fast'] & distance_right['close'] & distance_left['far'], steering_action ['steer_
rule8 = ctrl.Rule(distance['moderate'] & speed['fast'] & distance_right['close'] & distance_left['close'], steering_action ['mair
rule9 = ctrl.Rule(distance['moderate'] & speed['moderate'] & distance_right['far'] & distance_left['far'], steering_action ['stee
rule10= ctrl.Rule(distance['moderate'] & speed['moderate'] & distance_right['far'] & distance_left['close'], steering_action ['st
rule11 = ctrl.Rule(distance['moderate'] & speed['moderate'] & distance_right['close'] & distance_left['far'], steering_action ['s
rule12 = ctrl.Rule(distance['moderate'] & speed['moderate'] & distance_right['close'] & distance_left['close'], steering_action [
rule13 = ctrl.Rule(distance['moderate'] & speed['slow'] & distance_right['far'] & distance_left['far'], steering_action ['steer_
rule14 = ctrl.Rule(distance['moderate'] & speed['slow'] & distance_right['far'] & distance_left['close'], steering_action ['steer
rule15 = ctrl.Rule(distance['moderate'] & speed['slow'] & distance_right['close'] & distance_left['far'], steering_action ['steer
rule16 = ctrl.Rule(distance['moderate'] & speed['slow'] & distance_right['close'] & distance_left['close'], steering_action ['mai
rule17 = ctrl.Rule(distance['far'] ,horn['yes'] )
rule23 = ctrl.Rule(speed['fast']  ,horn['yes'] )
rule24 = ctrl.Rule(distance['close'],horn['yes'] )
rule18 = ctrl.Rule(distance['far'] & distance_right['close'] & distance_left['close'], steering_action ['maintain'] )
rule19 = ctrl.Rule(distance['far'] & distance_right['far'] & distance_left['far'], steering_action ['maintain'] )
rule20 = ctrl.Rule(distance['far'] & distance_right['close'] & distance_left['far'], steering_action ['maintain'] )
rule21 = ctrl.Rule(distance['far'] & distance_right['far'] & distance_left['close'], steering_action ['maintain'] )

#creating control system and simulation
system = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9, rule10, rule11, rule12, rule13, rule1
simulation = ctrl.ControlSystemSimulation(system)

#Set input values obtained by self driving car sensors
simulation.input['speed'] = 100  # speed km/h
simulation.input['distance'] = 2 #meters
simulation.input['distance_left'] = 8 #in meters to left vehicle
simulation.input['distance_right'] = 7 #in meters to right vehicle

simulation.compute()

print("Brake:", simulation.output['brake'])
print("Horn:", simulation.output['horn']) #if output value<0.5; horn = no , if output value>0.5; horn = yes
print("Steering_action :", simulation.output['steering_action']) #  if output=50 steering_action is maintain (No steer) ; if outp
```

```
Brake: 82.77777777777777
Horn: 0.6666666666666666
Steering_action : 49.74438687392052
```

In [36]:
```
#view connections of a specific rule
r2 = rule2.view()
print("rule1",r2)
r5 =rule5.view()
print("rule5",r5)
```

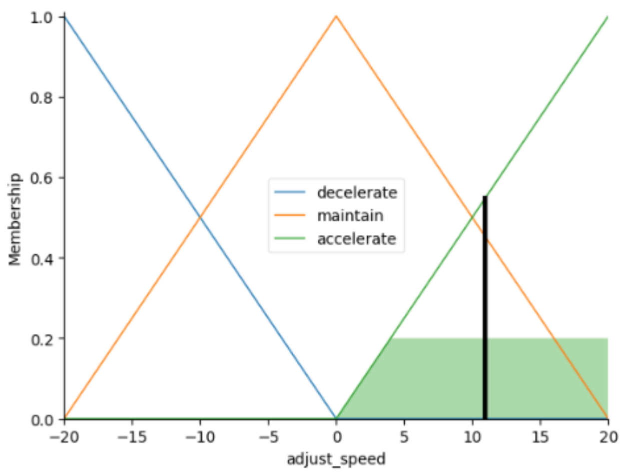
```
rule1 (<Figure size 640x480 with 1 Axes>, <Axes: >)
rule5 (<Figure size 640x480 with 1 Axes>, <Axes: >)
```





In [37]:
```
rule26 = ctrl.Rule(speed['fast'] & speed_limit['slow']  , adjust_speed['decelerate'] )
rule27 = ctrl.Rule(speed['slow'] & speed_limit['fast']  , adjust_speed['accelerate'] )
rule28 = ctrl.Rule(speed['moderate'] & speed_limit['moderate']  , adjust_speed['maintain'] )
rule29 = ctrl.Rule(speed['fast'] & speed_limit['fast']  , adjust_speed['maintain'] )
rule30 = ctrl.Rule(speed['slow'] & speed_limit['slow']  , adjust_speed['maintain'] )

fis_ctrl = ctrl.ControlSystem([rule26, rule27, rule28, rule29, rule30])
fis = ctrl.ControlSystemSimulation(fis_ctrl)

#Input values as per the scenario
fis.input['speed'] = 40  #Example value 40 km/h
fis.input['speed_limit'] =100

#perform fuzzy inference
fis.compute()
#output result
print(fis.output['adjust_speed'])

#view
adjust_speed.view(sim=fis)
```

```
10.962962962962962
```



References

[Fuzzy Control Systems: The Tipping Problem — skfuzzy v0.2 docs (pythonhosted.org)]

https://www.cs.princeton.edu/courses/archive/fall07/cos436/HIDD EN/Knapp/fuzzy004.htm

https://youtu.be/XACvF3TtywM. YouTube Tutorial (Accessed on 2nd August 2023)