K18KM

SOFT COMPUTING TECHNIQUE CA-1

on

A FUZZY LOGIC CONTROL SYNTHESIS FOR AN AIRPLANE ANTILOCK-BRAKING SYSTEM

Submitted by

Atul Shukla

Registration No : 11804538

Programme Name ; Btech. CSE

Under the Guidance of

USHA MITTAL

School of Computer Science & Engineering

Lovely Professional University, Phagwara

(Aug-Sep,2020)

Contents

1 OVERVIEW

Human judgment error may cause serious accidents due to inaccurate reaction time and time consumed to perform full braking. In many such cases, the main cause of such accidents is the distraction through driving and failure to react in such enough time. Moreover, the fully intelligent vehicle, which is called a driverless car, will totally automate without any human interaction. Each vehicle will utilize both its speed sensor which measures the wheel speed and distance sensor which is based on ultrasonic waves to compute the safety distance between the vehicle and its neighbours by controlling the brake pressure force on the brake pedal.

AN AIRPLANE ANTILOCK-BRAKING SYSTEM, to stop the vehicle as safely and shortly as possible. This implies, to begin with, the avoiding of the vehicle lateral instability as a result of wheel slip increasing beyond a crucial level, where the ability to steer the vehicle will be compromised. The cause isn't the loss of the longitudinal friction coefficient, however the lateral friction coefficient, which decreases proportionally to the slip. Given the ABS main objective, the controller releases or applies the brakes, aiming to achieve a tradeoff between braking effectiveness and lateral stability. Many successful proprietary algorithms exist for ABS control logic.

. The main problem of this project is as follows:

- Human driving, which involves reaction times, delays, and judgment errors is not enough to avoid emergency braking or road accidents.
- Moreover, and in many such cases, the cause of the accident is driver distraction and failure to react in time.
- So, in this project, we design a human free brake system that maintains both the vehicle speed and inter-vehicle distance, to automatically decelerate the vehicle on urgent demands by controlling the brake force pressure.

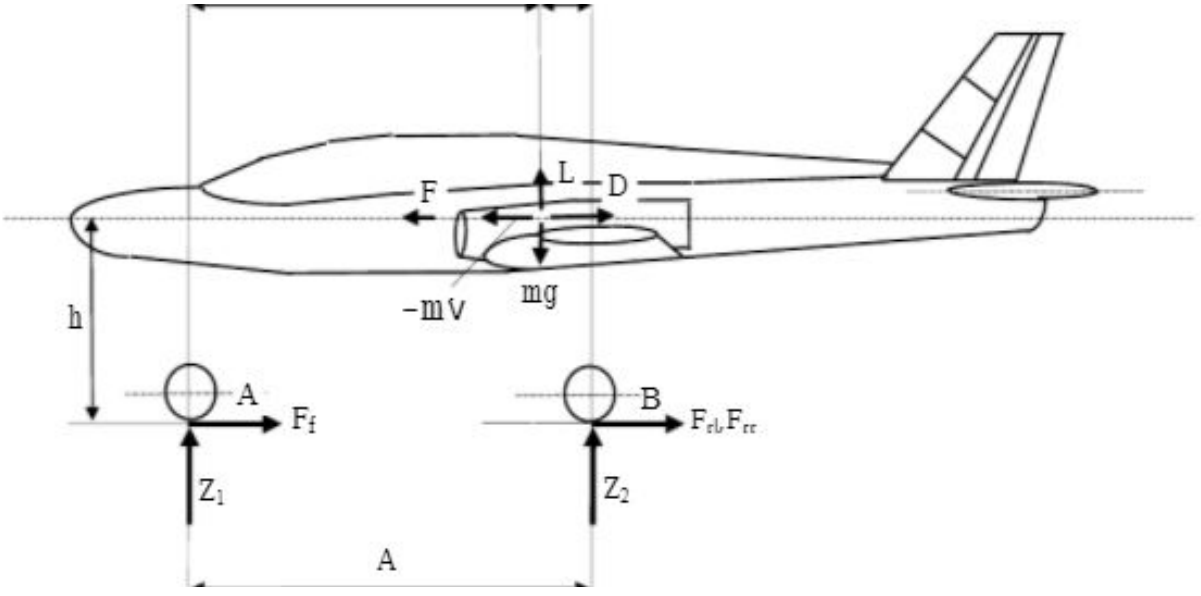
2 INPUT-OUTPUT OF AIRPLANE ANTILOCK-BRAKING SYSTEM

We have two inputs and one output, the inter-distance is based on ultrasonic sensor and speed is based on the Hall effect principle, as will be explained later. The output of our fuzzy proposed controller is brake force pressure.

For measuring purposes of the distance between two consequence vehicles, usually ultrasonic sensors are used which is considered very popular and low cost. Figure 1 shows a type of ultrasonic sensor. Simply, it comes with a separate transmitter, called TRIGGER and a receiver, called ECHO. The sender sends 8 pulses of 40 kHz, which when hits an obstacle stands at the front of the ultrasonic module, gets back and receives from the receiver side. By simple calculation on the receiver side and depending on the duration of the sum of the travelling time, when the signal transferred until when it received back, the distance between the vehicle and the next one could be estimated.

3 DESIGN STRUCTURE

Our fuzzy logic design is consisting of inputs, output, and Mamdani-Fuzzy inference engine. The output is the brake force and is defuzzied using the centroid method. The block diagram below depicts that.



Fuzzification

We divide the distance range from 0-10 m to three sub ranges, very close(V_cls), close, and far. The inter-vehicle distance is classified as follows:

- Very close distance (0-5 m), Triangular.
- Close distance (2-8 m), Triangular.
- Far distance (5-10 m), Triangular.

We have chosen triangle membership function since as the vehicle becomes close, the value of the crisp input (V_cls) increases dramatically, at the middle membership, there is marginal space starting from 4 and ends at 6m, after that, the distance is considered far.

Then, we divide the vehicle speed from 0 to 10 m/s to 5 different ranges as follows:

- Too slow (0-2 m/s), Trapezoidal.
- Slow (1-4 m/s), Trapezoidal.
- Optimum (3-6 m/s), Trapezoidal.
- Fast (5-8 m/s), Trapezoidal.
- Too Fast (7-10 m/s), Trapezoidal.

For the speed since its relative and couldn't rely on a specific value, we select trapezoidal membership function. The trapezoidal shape makes the values wave with reset range.

The output of our fuzzy system is the amount of brake force and is classified as follows:

- Decrease brake force greatly (0-8 N)
- Decrease brake force slightly (4-12 N)
- No brake reaction (8-16 N)
- Increase brake force slightly (12-20 N)
- Increase brake force greatly (16 -24 N)

The output membership function consists of 5 members, triangle membership is the best choice, since for each division range, the value increases sharply until the middle then decreases to interfere with the adjutant membership. Figure 4 below depicts the input and output membership functions.

CONCLUSION

Fuzzy logic is shown to be a successful tool in our simplified ABS problem. "here the aircraft speed is assumed constant. The uncertain and non-linear features of the ABS design find fuzzy logic as an ideal selling to cope with such Other aerospace and vehicle control issues. such as the adaptive suspension system. autonomous navigation, and engine control. to name. I just a few will also find the fuzzy logic approach useful. Future research calls for the ABS design for the realistic system where the aircraft dynamics are coupled with the dynamics of several wheels.

PYTHON CODE

```
import numpy as
np import
skfuzzy as fuzz
import matplotlib.pyplot as
plt import matplotlib.pyplot
as plt2 from skfuzzy
import control as ctrl



# Generate universe variables
# * distanceance is on subjective ranges [0,
10] in units of m. # *Speed is on subjective
ranges [0, 10] in units of m/s.
# * Brake pedal force (deceleration) has a range of [0,
24] in units of N. dist = np.arange(0, 11, 1)
speed = np.arange(0, 11, 1)
Brake_F = np.arange(0, 25, 1)
# Generate fuzzy
membership functions
Dist_V_cls = fuzz.trimf(dist,
[0, 0, 4])
Dist_cls = fuzz.trapmf(dist, [2, 4, 6, 8])
Dist_far = fuzz.trimf(dist, [6, 10, 10])
################################################
########
```

```
speed_2_slow = fuzz.trapmf(speed, [0, 0, 1, 2])

speed_slow = fuzz.trapmf(speed, [1, 2, 3, 4])

speed_op = fuzz.trapmf(speed, [3, 4, 5, 6])

speed_fast = fuzz.trapmf(speed, [5, 6, 7, 8])

speed_2_fast = fuzz.trapmf(speed, [7, 8, 10,10])

##############################################

########

Dec_brake_greatly = fuzz.trimf(Brake_F, [0, 4, 8])

Dec_brake_slightly = fuzz.trimf(Brake_F, [4, 8, 12])

No_brake = fuzz.trimf(Brake_F, [8, 12, 16])

Inc_brake_slightly = fuzz.trimf(Brake_F, [12, 16, 20])

Inc_brake_greatly = fuzz.trimf(Brake_F,

[16, 20, 24]) # Visualize these universes

and membership functions

fig, (ax0, ax1, ax2) = plt.subplots(nrows=3,

figsize=(10, 10)) ax0.plot(dist, Dist_V_cls, 'b',

linewidth=1.5, label='Very-Close') ax0.plot(dist,

Dist_cls, 'g', linewidth=1.5, label='Close')

ax0.plot(dist, Dist_far, 'r', linewidth=1.5,

label='Far') ax0.set_title('Inter-vehicle distance')

ax0.legend()

ax1.plot(speed, speed_2_slow, 'b', linewidth=1.5,

label='Too Slow') ax1.plot(speed, speed_slow, 'y',

linewidth=1.5, label='Slow') ax1.plot(speed,

speed_op, 'g', linewidth=1.5, label='optimun')

ax1.plot(speed, speed_fast, 'r', linewidth=1.5,
```

```
label='Fast') ax1.plot(speed, speed_2_fast, 'k',
linewidth=1.5, label='Too Fast')
ax1.set_title('Speed')
ax1.legend()
ax2.plot(Brake_F, Dec_brake_slightly, 'b', linewidth=1.5,
label='Dec-Slightly') ax2.plot(Brake_F, Dec_brake_greatly,
'y', linewidth=1.5, label='Dec-Greatly') ax2.plot(Brake_F,
No_brake, 'g', linewidth=1.5, label='No_brake')
ax2.plot(Brake_F, Inc_brake_slightly, 'r', linewidth=1.5,
label='Inc_Slightly') ax2.plot(Brake_F, Inc_brake_greatly,
'k', linewidth=1.5, label='Inc_Greatly')
ax2.set_title('Brake_Force_Amount')
ax2.legend()
# Turn off top/right
axes for ax in
(ax0, ax1, ax2):
  ax.spines['top'].set_visible(False)
  ax.spines['right'].set_visible(False)
  ax.get_xaxis().tick_bottom()
  ax.get_yaxis().tick_left()
plt.tight_layout()
plt.show() # To show the membership inputs

distance = ctrl.Antecedent(dist,
'distance') speed =
ctrl.Antecedent(speed, 'speed')
```

```
Brake_F =
ctrl.Consequent(Brake_F,'Brake_F'
)
#####################################################
# Generate fuzzy membership functions
distance['V_cls'] = fuzz.trimf(distance.universe, [0, 0, 4])
distance['cls'] = fuzz.trapmf(distance.universe, [2, 4, 6, 8])
distance['far'] = fuzz.trimf(distance.universe, [6,
10, 10]) #distance.view()
#########################################
###############
speed['2_slow'] = fuzz.trapmf(speed.universe, [0, 0, 1, 2])
speed['slow'] = fuzz.trapmf(speed.universe, [1, 2, 3, 4])
speed['op'] = fuzz.trapmf(speed.universe, [3, 4, 5, 6])
speed['fast'] = fuzz.trapmf(speed.universe, [5, 6, 7, 8])
speed['2_fast'] = fuzz.trapmf(speed.universe, [7,
8, 10,10]) #speed.view()
#########################################
###############
Brake_F['DG'] = fuzz.trimf(Brake_F.universe, [0, 4, 8])
Brake_F['DS'] = fuzz.trimf(Brake_F.universe, [4, 8, 12])
Brake_F['NF'] = fuzz.trimf(Brake_F.universe, [8, 12, 16])
Brake_F['IS'] = fuzz.trimf(Brake_F.universe, [12, 16, 20])
```

```
Brake_F['IG'] = fuzz.trapmf(Brake_F.universe,
[16, 20, 24, 24]) #Brake_F.view()
# Visualize these universes and
membership functions #distance.view()

"""
The Fuzzy design rules
rule1= If too close ,      then    increase
greatly. rule2= If close and too fast ,
then increase slightly. rule3= If close
and optimum, then increase slightly.
rule4= If far and optimum, then no
reaction.
If far and slow ,   then slightly
decrease. If far or too slow,   then
greatly decrease. """
rule1 = ctrl.Rule(distance['V_cls'] , Brake_F['IG'])
rule2 = ctrl.Rule(distance['cls'] & speed['2_fast'] ,
Brake_F['IS']) rule3 = ctrl.Rule(distance['cls'] &
speed['op'] , Brake_F['IS']) rule4 =
ctrl.Rule(distance['far'] & speed['op'] ,
Brake_F['NF']) rule5 = ctrl.Rule(distance['far'] &
speed['slow'] , Brake_F['DS'])
rule6 = ctrl.Rule(distance['far'] & speed['2_slow'] , Brake_F['DG'])

brake_ctrl = ctrl.ControlSystem([rule1, rule2, rule3,
```

```
rule4, rule5, rule6]) braking =

ctrl.ControlSystemSimulation(brake_ctrl)

# Pass inputs to the ControlSystem using Antecedent

labels with Pythonic API # Note: if you like passing many

inputs all at once, use .inputs(dict_of_data)

braking.input['distance'] = 8

braking.input['speed'] = 3
# Crunch the numbers braking.compute()
print

(braking.output['Brake_F

'])

Brake_F.view(sim=braki

ng)



"""""""""

sim = ctrl.ControlSystemSimulation(brake_ctrl,

flush_after_run=24 * 24 + 1) # We can simulate at higher

resolution with full accuracy

upsampled = np.linspace(0, 10, 25)

x, y = np.meshgrid(upsampled,

upsampled) z = np.zeros_like(x)



# Loop through the system 21*21 times to collect the

control surface for i in range(10):

  for j in range(10):

    sim.input['distance']

    = x[i, j]
```

```
        sim.input['speed'] =
        y[i, j] sim.compute()
        z[i, j] = sim.output['Brake_F']

# Plot the result in pretty 3D with
alpha blending import
matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D # Required for 3D plotting

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, projection='3d')

surf = ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap='viridis',
```



```
                linewidth=0.4, antialiased=True)

cset = ax.contourf(x, y, z, zdir='z', offset=-2.5,
cmap='viridis', alpha=0.5) cset = ax.contourf(x, y, z,
zdir='x', offset=3, cmap='viridis', alpha=0.5) cset =
ax.contourf(x, y, z, zdir='y', offset=3, cmap='viridis',
alpha=0.5)

ax.view_init(10
, 20) plt.show()
```

REFERENCE

Aras, A. A. 2013. Design of a Controller for Abs Anti Lock Breaking System Using Fuzzy Logic Control. California State University, Northridge,

Mamat, M. & Ghani, N. 2009. Fuzzy Logic Controller on Automated Car Braking System. *2009 IEEE International Conference on Control and Automation*, pp. 2371-2375.