Fuzzy Cruise Controller

Shrirang Mhalgi	**422014**
Ria Mittal	**422016**
Pranav Shirude	**422055**
Honey Talreja	**422062**

DEPARTMENT OF COMPUTER ENGINEERING
VISHWAKARMA INSTITUTE OF INFORMATION TECHNOLOGY, Pune
Affiliated to
Savitribai Phule Pune University
[2019-20]

ACKNOWLEDGEMENT

We are immensely gratified to receive the opportunity to present this report on ‘Soft Computing and Optimization Algorithms’. It has been an excellent learning curve under the guidance of Prof. M.P. Mali and Prof. N. G. Kharate, who always continued to share their valuable suggestions throughout the project work. The smooth completion of the project activities would not have been possible without their guidance. We would also thank all our colleagues for their valuable support throughout the engineering career.

Shrirang Mhalgi
Ria Mittal
Pranav Shirude
Honey Talreja

Table of Contents

TITLE

Solve Greg Viot‟s fuzzy cruise controller using MATLAB Fuzzy logic toolbox or Octave or Python.

PROJECT DEFINITION

This project depicts the working of a fuzzy cruise controller in vehicles for their locomotion. It uses the concepts of Physics along with fuzzy inferences to build an integrated model which describes the relationship between the gradient (slope) of the land and velocity of the vehicle along with the gas pedal (accelerator) orientation.

OBJECTIVES

The main objective is to learn the working of fuzzy systems and how they can be integrated with simple science to produce adaptive locomotion in any vehicle. Even the relationship between the components of the vehicle can be understood in an exceptional manner.

ABSTRACT

In this current world scenario, vehicles have become an important part of everyone’s lives. They serve as the only mode of transfer from a place to another. In this case, a controller which enables automatic driving along with suggestions to speed control is a boon to the world. This report provides a detailed description about the working of a cruise controller and its components, their relationship with each other and how can fuzzy systems be integrated within them.

Keywords: Fuzzy Logic, Cruise Controller, Fuzzy Systems, Locomotion.

1. Introduction

Fuzzy logic is an approach to computing based on "degrees of truth" rather than the usual "true or false" (1 or 0) Boolean logic on which the modern computer is based. The idea of fuzzy logic was first advanced by Dr. Lotfi Zadeh of the University of California at Berkeley in the 1960s. Fuzzy logic is a form of many-valued logic in which the truth values of variables may be any real number between 0 and 1 both inclusive. It is employed to handle the concept of partial truth, where the truth value may range between completely true and completely false.

1.1 Background

The purpose of a cruise control system is to accurately maintain a speed set by the driver without any outside intervention by controlling the throttle-accelerator pedal linkage. The earliest variants of cruise control were actually in use even before the creation of automobiles. The inventor and mechanical engineer James Watt developed a version as early as the 17th century, which allowed steam engines to maintain a constant speed up and down inclines. Cruise control as we know it today was invented in the late 1940s, when the idea of using an electrically-controlled device that could manipulate road speeds and adjust the throttle accordingly was conceived. In this report, we have considered the slope of land and corresponding gas pedal level.

Figure 1: Cruise Controller Symbol

1.2 Working of Cruise Controller

The cruise control system controls the speed of your car the same way you do – by adjusting the throttle (accelerator) position. However, cruise control engages the throttle valve by a cable connected to an actuator, rather than by pressing a pedal. The throttle valve controls the power and speed of the engine by limiting how much air it takes in (since it's an internal combustion engine). The driver can set the cruise control with the cruise switches, which usually consist of ON, OFF, RESUME, SET/ACCEL and COAST. These are commonly located on the steering wheel or on the windshield wiper or turn signal stalk. The SET/ACCEL knob sets the speed of the car. One tap will accelerate it by 1 mph, two by 2 mph and so on. Tapping the knob in the opposite direction will decelerate the vehicle. As a safety feature, the cruise control system will disengage as soon as you hit the brake pedal.

Figure 2: Input and Output Variables

1.3 Types of Cruise Controls

- **Normal:** Vehicles move on the fixed speed as per the rules in the system as shown.

Figure 3: Normal Control

- **Predictive:** Works with help of GPS imaging and predicts whether the upcoming road has a slope or not.

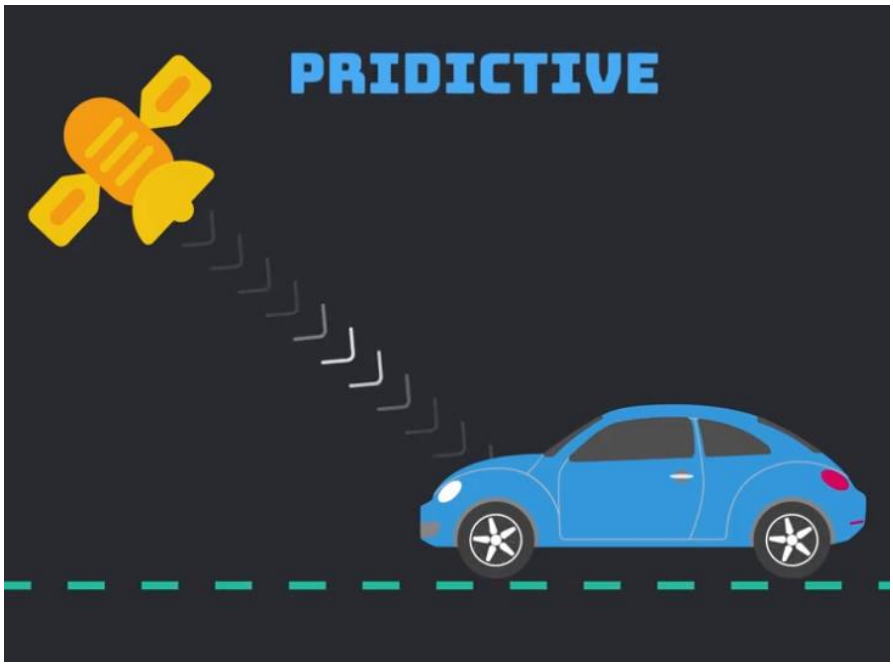

Figure 4: Predictive Control

- **Adaptive:** Vehicle adapts to the environment by mapping through sensors and other vehicles.

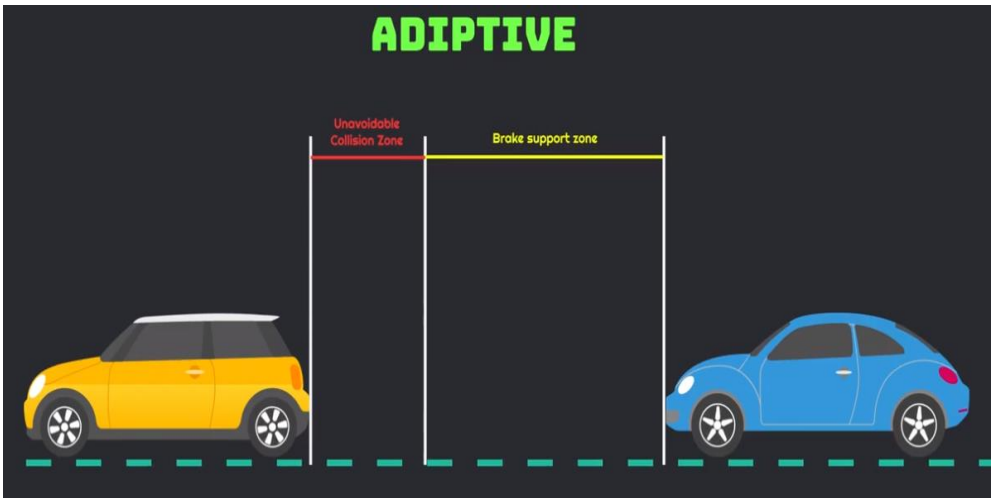

Figure 5: Adaptive Control

2. Requirements

2.1 Software Requirements

Python and its libraries like PySimpleGUI, tkinter, numpy, scikit-learn, etc.

2.2 Hardware Requirements

Any machine that supports python.

2.3 User Interface

The user interface works on running the files as the GUI is built in PySimpleGUI, which is a Python library for creating Python GUIs. It supports all types of inputs.

3. Outcomes

Using the aforementioned requirements, we have implemented a system that enables users to view the relationship between the vehicle acceleration and speed with the slope of land on which it runs. It shows the error as well which creeps into calculating the speed because of natural forces like friction and gravity.

4. Related Physics

Suppose, a self-driving car company has requested a speed controller for their new model of electric autonomous vehicles. Unlike standard cruise control systems, this speed controller must manage transitions between all velocities ranging from 0 to 25 m/s (56 mph or 90 km/hr).

The speed controller changes a gas pedal (%) input that adjusts electrical current to the motor, creates a torque on the drive train, applies forward thrust from wheel contact on the road, and changes the velocity of the vehicle. Regenerative braking is part of this application by allowing the gas pedal to go from -50% to 100%. Decreases in velocity are also due to resistive forces. Negative velocities (backwards driving) is highly undesirable.

4.1 Momentum Balance

A momentum balance is the accumulation of momentum for a control volume equal to the sum of forces *F* acting on the automobile.

$$\frac{d(m\,v)}{dt} = \sum F$$

with *m* as the mass in the vehicle and passengers and *v* as the velocity of the automobile. Assuming constant mass of the vehicle, a constant drag coefficient, C_d, and a linear gain between gas pedal position and force, the momentum balance becomes

$$m\frac{dv(t)}{dt} = F_p u(t) - \frac{1}{2}\rho\, A\, C_d v(t)^2$$

with *u(t)* as gas pedal position (%pedal), *v(t)* as velocity (m/s), *m* as the mass of the vehicle (500 kg) plus the mass of the passengers and cargo (200 kg), C_d as the drag coefficient (0.24), ρρ as the air density (1.225 kg/m^3), *A* as the vehicle cross-sectional area (5 m^2), and F_p as the thrust parameter (30 N/%pedal).

4.2 Control System Design

A first step in controller design is to list the PV (process variable), OP (controller output), SP (controller set point), and any potential DVs (disturbance variables). The three elements that create feedback control are a sensor, actuator, and controller.

Element	**Cruise Control**
OP: Actuator	Gas Pedal Position
CV: Controller Set Point	Desired Speed (mph or km/hr)
PV: Measurement (Sensor)	Speedometer, measured velocity
DV: Disturbance variables	Hills, wind, cargo + passenger load

4.3 Fit to First-Order Model

A first step in determining acceptable PID tuning parameters is to approximate the system response with a **first-order plus dead-time (FOPDT) model**. This can either be done approximately with graphical methods and a step response or more precisely with optimization. In this case, use the step response obtained from the simulation above to obtain parameters KpKp, τpτp, and θpθp.

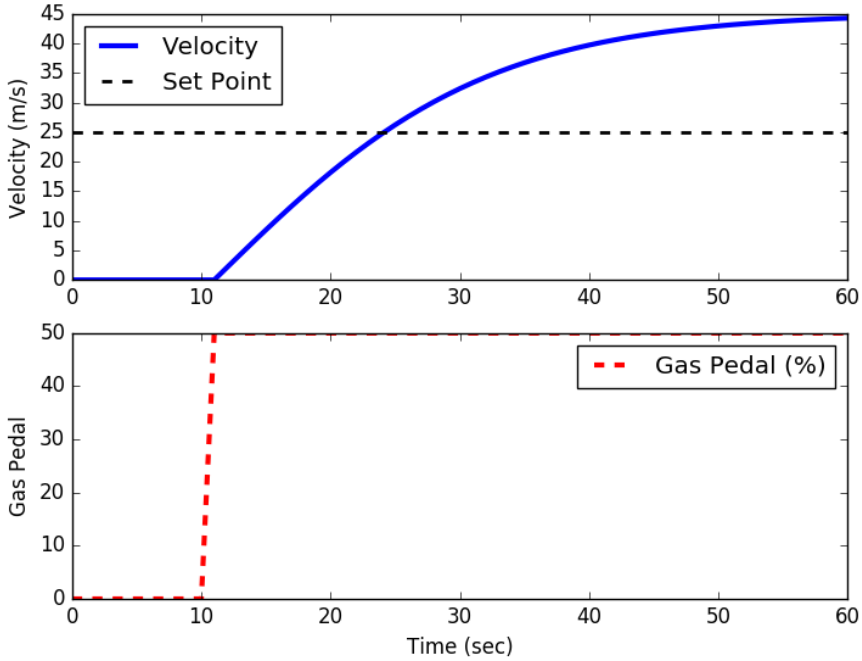

Figure 6: FOPDT checks

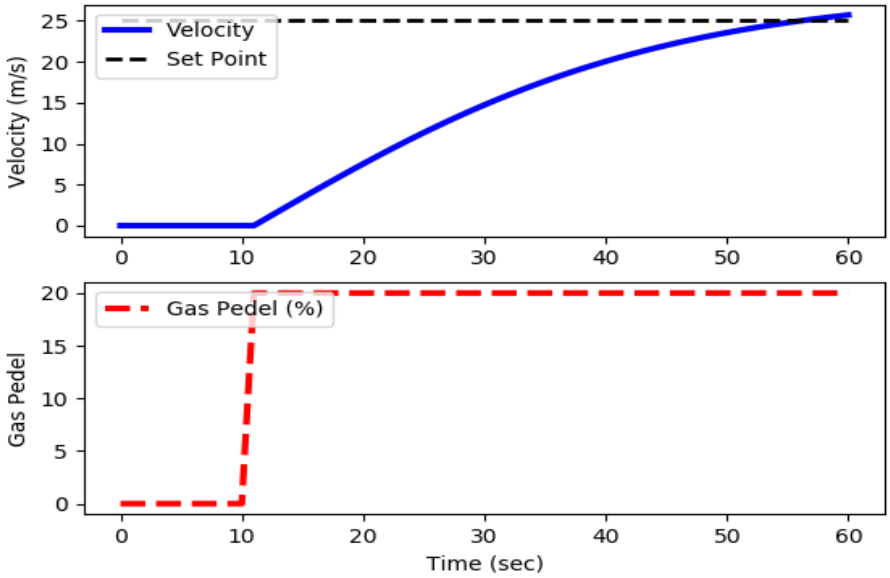

Figure 7: Step-Up as 20

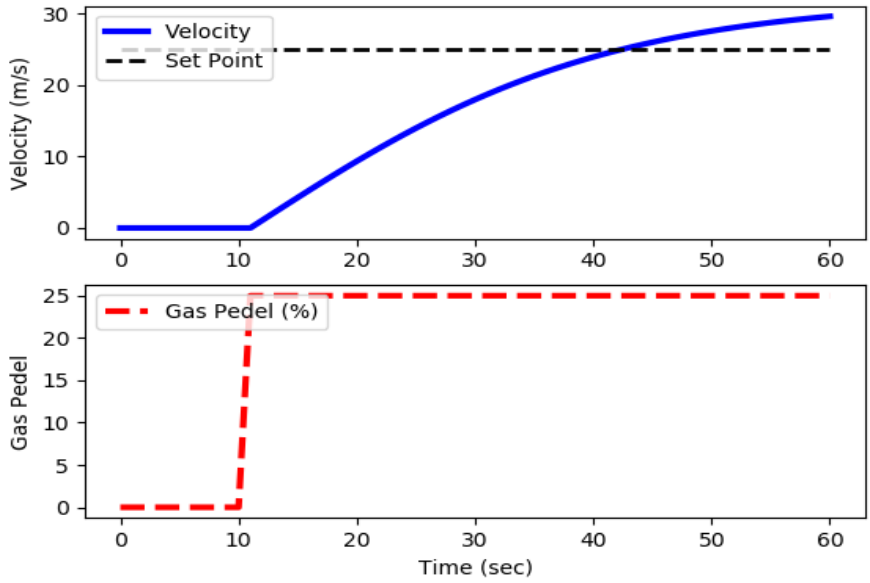

Figure 8: Step-Up as 25

So as per the figures, it has been observed that the least amount of error occurs when step-up value is kept 20.0, thus, it is chosen as threshold for this project.

5. Source Code

Sample code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

def vehicle(v, t, u, load):
    # v = vehicle velocity (m/s)
    # t = time (s)
    # u = gas pedel position (-50% to 100%)
    # load = passenger load + cargo (kg)
    Cd = 0.24 # coefficient of drag
    A = 5.0 # cross sectional area (m^2)
    rho = 1.225 # density of air (kg/m^3)
    Fp = 30 # Thrust parameter (N/%pedel)
    # m = 500 # mass of vehicle (kg)

    # calculate the derivative of velocity
    dv_dt = (1.0 / load) * ((Fp * u) - (0.5 * rho * Cd * A * v ** 2))
```

```
    return dv_dt

def start(tf, load, breakpoint1, breakpoint2, breakpoint3, breakpoint4, terra
in1, terrain2, terrain3, terrain4):
    # tf = time for simulation
    nsteps = tf + 1 # number of time steps
    delta_t = tf / (nsteps - 1) # how long is each step
    ts = np.linspace(0, tf, nsteps) # linearly spaced time vector

    # Simulate the step test operation
    step = np.zeros(nsteps) # u = valve % open

    # passenger(s) + cargo load
    # load = 200.0 # kg
    v0 = 0.0 # initial velocity

    # for storing the results
    vs = np.zeros(nsteps)

    # PI controller
    ubias = 0.0
    Kc = 1.0 / 1.2
    taui = 20.0
    sum_int = 0.0
    es = np.zeros(nsteps) # error
    ies = np.zeros(nsteps) # integral of error
    sp_store = np.zeros(nsteps) # set point store
    sp = 25

    # simulate with odeint
    for i in range(nsteps - 1):
        # Schedule changes in set point
        if i == breakpoint1:
            sp = terrain1
        elif i == breakpoint2:
            sp = terrain2
        elif i == breakpoint3:
            sp = terrain3
        elif i == breakpoint4:
            sp = terrain4

        sp_store[i + 1] = sp
        error = sp - v0
        es[i + 1] = error
        sum_int = sum_int + error * delta_t
```

```
        u = ubias + (Kc * error) + ((Kc / taui) * sum_int)

        # clip the inputs to -50% to 100%
        if u >= 100.0:
            u = 100.0
            sum_int = sum_int - error * delta_t
        elif u <= -50.0:
            u = -50.0
            sum_int = sum_int - error * delta_t
        ies[i + 1] = sum_int
        step[i + 1] = u
        v = odeint(vehicle, v0, [0, delta_t], args=(u, load))
        v0 = v[-1] # take the last value
        vs[i + 1] = v0 # store for plotting

    # plot the results
    plt.figure()
    plt.subplot(2,2,1)
    plt.plot(ts, vs, 'b-', linewidth=3)
    plt.plot(ts, sp_store, 'k--', linewidth=2)
    plt.ylabel("Velocity (m/s)")
    plt.legend(["Velocity", "Set Point"], loc='best')
    plt.subplot(2,2,2)
    plt.plot(ts, step, 'r--', linewidth=3)
    plt.ylabel("Gas Pedel")
    plt.legend(["Gas Pedel (%)"], loc='best')
    plt.subplot(2,2,3)
    plt.plot(ts, es, 'b-', linewidth=3)
    plt.legend(["Error (SP - PV)"])
    plt.xlabel("Time (sec)")
    plt.subplot(2,2,4)
    plt.plot(ts, ies, 'k--', linewidth=2)
    plt.legend(["Integral of Error"])
    plt.xlabel("Time (sec)")
    plt.show()
```

6. Output

6.1 Client Side GUI

Figure 9: GUI

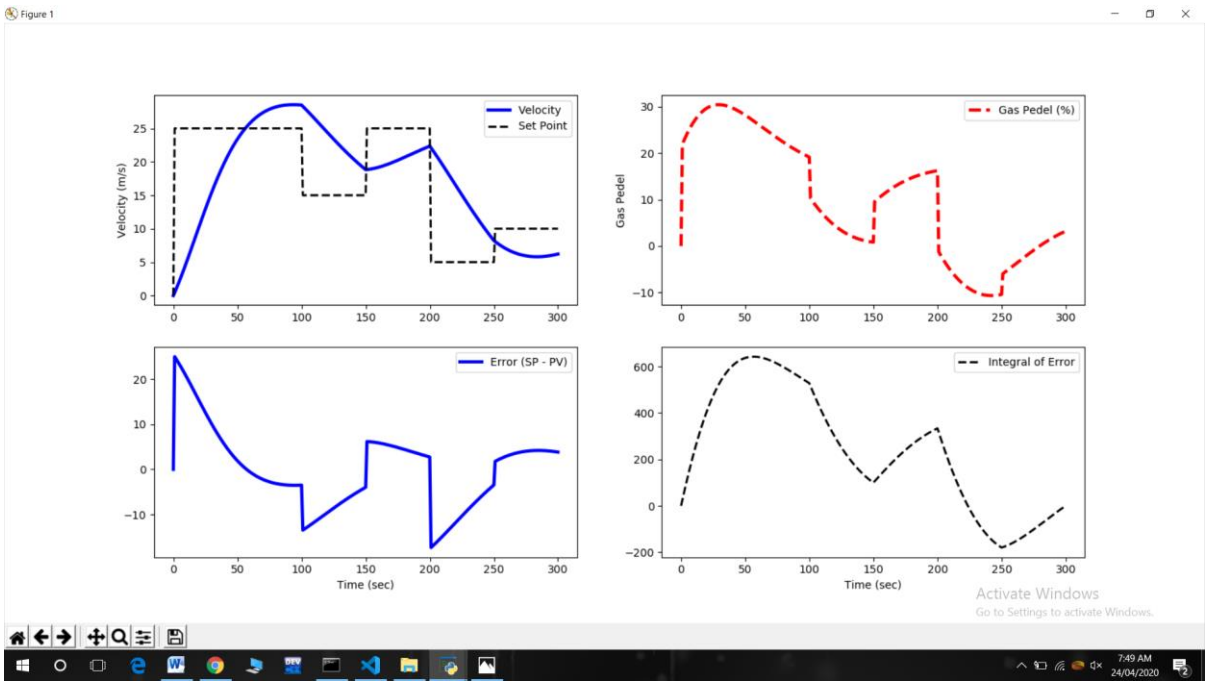

Figure 10: Output

7. Conclusion

The key features of the system including the usage of concepts of physics to apply them in a manner which will enhance the performance of the system in real time. The fuzzy nature of input makes it more probable to decide the velocity and angle of the gas pedal. Thus, by the application of science, this project has taught us to build a fuzzy system that can be used to automate the locomotion of a vehicle.