Tópicos Especiais em Instrumentação I

Relatório

Controlador Fuzzy

Frederico May de Liz e Matheus Quevedo Sivelli

Universidade Federal do Rio Grande do Sul, Departamento de Engenharia Elétrica, Curso de

Engenharia Elétrica, Tópicos Especiais em Instrumentação I , Prof. Dr. Tiago Oliveira Weber

E-Mails: fredericoliz25@gmail.com (F.M.L); Matheus_sivelli@hotmail.com (M.Q.S)

Data Início: 20/06/2020; Data Final: 10/07/2020

Resumo: O presente trabalho tem como principal objetivo utilizar as técnicas aprendidas de aprendizado de máquina juntamente com a lógica Fuzzy para construirmos um controlador. Para atingirmos tal objetivo, foi criado um simulador de corrida em linguagem python para melhor atender nosso problema. O objetivo do controlador é deixar o carro no piloto automático dentro do perímetro permitido, ou seja, caso ocorra algum acidente, quando o carro sair da pista, o jogo é automaticamente fechado. Por fim, foi comparado e discutido a abordagem do controlador Fuzzy com um controlador PD tradicional.

Abstract: The current report focus on the use of machine learning techniques combined with a Fuzzy logic to build a race track controller. To achieve this goal, a racing simulator was created using python language in order to better achieve our problem. The objective is to control the car car within the allowed distance, that is, in the event of an accident, if the car leaves the track, the game is closed automatically. Finally, the Fuzzy controller was analyzed and compared with a PD controller as well.

Palavras Chaves: Fuzzy; Controlador; Aprendizagem de Máquina;

---

1. Introdução

A lógica fuzzy é uma abordagem para o processamento de dados baseado no grau de pertencimento em detrimento do comum verdadeiro ou falso (1 ou 0) da lógica booleana. Esta lógica é mais intuitiva e acessível pois processa dados de forma parecida com a forma que nosso cérebro interpreta no cotidiano. Isso acontece de forma natural, com a associação de dados com intervalos de verdades parciais, de modo que nem todo dado está dentro de um intervalo apenas, podendo ter dupla interpretação. Assim a lógica fuzzy abre mão da precisão porém facilita a resolução de problemas, trata de maneira mais clara das imprecisões e é mais intuitivo, principalmente se a aplicação será utilizada por uma parcela da população que não tem conhecimento intermediário sobre engenharia. A Figura 1 demonstra o processo de descrição de um controlador fuzzy, com a exibição de suas principais partes.

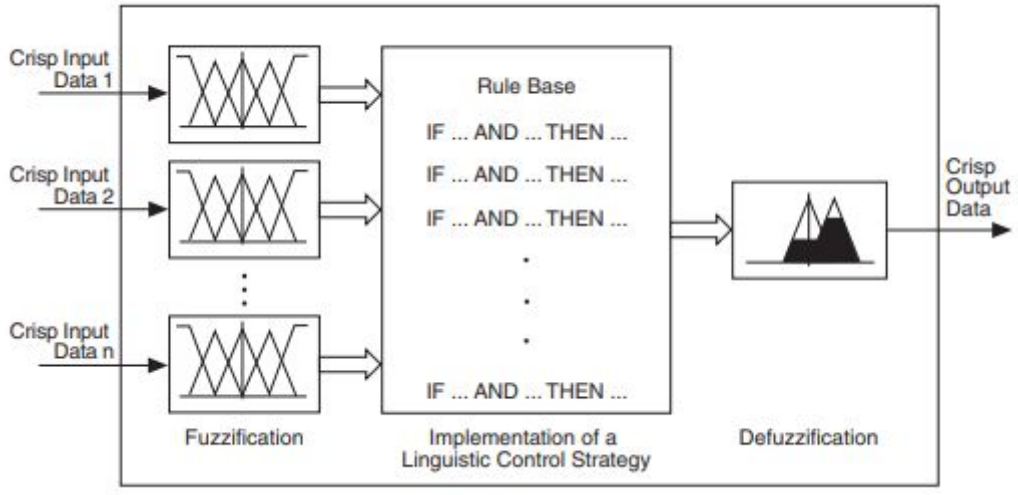

Figura 1. Processos de um controlador *Fuzzy*.

Fonte – LabVIEW: PID and Fuzzy Logic Toolkit User Manual.

Após um controlador fuzzy 'fuzzyficar' os valores de entrada de um sistema, o controlador fuzzy usa os termos linguísticos de entrada correspondentes e a base de regras para determinar os termos linguísticos resultantes das variáveis linguísticas de saída. A lógica fuzzy permite descrever sistemas extremamente complexos apenas com valores qualitativos que não necessitam de amplo conhecimento prévio. Consegue-se implementar lógicas extremamente complexas no ponto de vista físico, mas que partam de uma premissa até então simplória. No caso de um piloto automático, utilizando valores qualitativos como 'devagar', 'rápido' e 'descentralizado' conseguem nos trazer um resultado satisfatório em comparação com os controladores tradicionais que requerem uma matemática mais avançada da planta e da função de transferência aplicada no sinal de controle.

Neste relatório foi utilizado a teoria por trás dos controladores fuzzy para criar um piloto automático para um software de corrida. Inicialmente, foram construídos sensores laterais e frontais para armazenar os dados conforme o carro era manualmente movimentado. Ao obter uma quantidade considerável de dados, foi implementado um controlador sob as premissas fuzzys. Após a

implementação do controlador fuzzy, foi implementado um controlador sob as teorias tradicionais de controle para comparar o desempenho de cada um.

2. Metodologia Experimental

2.1 Simulador de pista de corrida

Foi desenvolvido o modelo virtual de uma pista de corrida para realizar o controle de um carro. Para isso, utilizou-se um código já criado em linguagem *python* de direção de um carro em 2D. Com a biblioteca *pygame*, foi inserido o carro e seus movimentos no jogo e desenvolvido uma pista compatível com o sistema a ser controlado posteriormente. A classe do carro utilizada possui os seguintes valores:

- *Position*: são as coordenadas x e y do carro;
- *Velocity*: velocidade em relação ao carro, em metros por segundo;
- *Angle*: ângulo em relação à tela, em graus;
- *Length*: comprimento do chassi, em metros;
- *Maximum value of acceleration*: valor máximo de aceleração, em metros por segundo ao quadrado;
- *Maximum steering value*: valor máximo do ângulo do volante, em graus;
- *Current acceleration*: aceleração atual, em metros por segundo ao quadrado;
- *Current steering value*: valor atual da direção. Valores negativos significam que as rodas estão viradas para a direita e positiva para a esquerda.

A classe utilizada para o carro no simulador pode ser observada na Figura 2.

```python
class Car:
    def __init__(self, x, y, angle=0.0, length=1, max_steering=30, max_acceleration=1):
        self.position = Vector2(x, y)
        self.velocity = Vector2(0.0, 0.0)
        self.angle = angle
        self.length = length
        self.max_acceleration = max_acceleration
        self.max_steering = max_steering
        self.max_velocity = 3
        self.brake_deceleration = 20
        self.free_deceleration = 5
        self.acceleration = 0.0
        self.steering = 0.0

    def update(self, dt):
        self.velocity += (self.acceleration * dt, 0)
        self.velocity.x = max(-self.max_velocity, min(self.velocity.x, self.max_velocity))

        if self.steering:
            turning_radius = self.length / sin(radians(self.steering))
            angular_velocity = self.velocity.x / turning_radius
        else:
            angular_velocity = 0

        self.position += self.velocity.rotate(-self.angle) * dt
        self.angle += degrees(angular_velocity) * dt

```

Figura 2. Código da classe do carro para o simulador.

Fonte – Py Game 2D Car Tutorial [S. l.], 2019.

Ao iniciar o jogo é criado uma instância para o carro. É inserido posteriormente o carro em uma posição onde se encontra dentro da pista de corrida. A sua posição posteriormente é atualizada juntamente com seus valores de velocidade e ângulo da direção conforme é chamada a função *update* com a variável *dt*, que é a quantidade de tempo que passou desde o último frame do jogo.

A pista é posicionada e juntamente a ela é criada uma máscara onde cada pixel de cor preta na tela é reconhecida como valor ‘1’. Para o loop do jogo, caso o centro da imagem do carro for posicionado em uma região na tela onde o pixel for preto (no caso fora da pista de corrida), é chamada a função de finalização do jogo e o programa é fechado. Sabendo que fora da pista de corrida os valores serão sempre 1 em cada pixel, foram criados 3 sensores de posição para o carro: lateral direito, lateral esquerdo e frontal. Cada sensor fará o cálculo da distância euclidiana entre o centro do carro até o primeiro encontro onde um pixel terá o seu valor de resultado 1. Os sensores implementados no código do simulador são apresentados na Figura 3.

```
            #sensores
            for i in range(1500):
                x =  np.int(pygame_position[0] + math.cos(math.radians(car.angle)) * i)
                y =  np.int(pygame_position[1] - math.sin(math.radians(car.angle)) * i)
                if (track_mask.get_at([x,y])==1):
                    pygame.draw.lines(self.screen,(255,0,0),True,[pygame_position, [x,y]])
                    #print('Sensor Reto: ', math.hypot(x - pygame_position[0], y - pygame_position[1]))
                    a.append(math.hypot(x - pygame_position[0], y - pygame_position[1]))
                    break

            for i in range(1500):
                 x = np.int(pygame_position[0] + math.cos(math.radians(car.angle+90)) * i)
                 y = np.int(pygame_position[1] - math.sin(math.radians(car.angle+90)) * i)
                 if (track_mask.get_at([x,y])==1):
                     pygame.draw.lines(self.screen,(255,0,0),True,[pygame_position, [x,y]])
                     #print('Sensor Esquerda: ', math.hypot(x - pygame_position[0], y - pygame_position[1]))
                     a.append(math.hypot(x - pygame_position[0], y - pygame_position[1]))
                     break

            for i in range(1500):
                x = np.int(pygame_position[0] + math.cos(math.radians(car.angle - 90)) * i)
                y = np.int(pygame_position[1] - math.sin(math.radians(car.angle - 90)) * i)
                if (track_mask.get_at([x,y])==1):
                    pygame.draw.lines(self.screen,(255,0,0),True,[pygame_position, [x,y]])
                    #print('Sensor Direita: ', math.hypot(x - pygame_position[0], y - pygame_position[1]))
                    a.append(math.hypot(x - pygame_position[0], y - pygame_position[1]))
                    break
```

Figura 3. Código dos sensores de distância do carro à borda da pista.

Fonte – Py Game 2D Car Tutorial [S. l.], 2019.

Por fim, quando o evento de fechamento do programa ocorre, o simulador salva cada dado calculado nos sensores, juntamente com os dados de velocidade e ângulo do carro em um arquivo *.csv*. Esses dados foram posteriormente utilizados para realizar a análise de clusterização do sistema a fim de criar regras para o controlador.

2.2 Clusterização de dados obtidos

Foi utilizado o fuzzy c-means, da biblioteca *skfuzzy*, para clusterizar o conjunto de dados. O algoritmo faz uso da lógica fuzzy e permite que determinados elementos pertencem a mais de uma categoria ao mesmo tempo, tendo seu relacionamento com a classificação expresso em grau de pertinência a determinado conjunto.

Inicialmente, foi preciso normalizar o conjunto de dados para que diferentes escalas não sejam influenciadoras na construção do cluster. Posteriormente, foi adotado uma técnica de mensuramento para o conjunto de dados, para decidir quantas categorias ou clusters descreveriam melhor o problema.

Para isso, adotou-se a técnica de otimização *fuzzy partition coefficient* (FPC). Esse parâmetro do algoritmo determina o quão limpo é o nosso dataset ao aumentar a quantidade de *clusters*, sendo o que mais se aproximar do valor 1 como o ideal. A Figura 4 ilustra como foi implementada a técnica de otimização, já com os dados previamente tratados para a clusterização.

```
112
113  """
114  cmeans para relacionar os sensores laterais com o ângulo do volante
115  """
116  c = []
117  for i in range(2,20):
118      cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(alldata, i, 2, error=0.005, maxiter=1000, seed=42)
119      c.append(fpc)
120
```

Figura 4. Código da técnica de otimização dos parâmetros do c-means para encontrar a melhor quantidade de clusters.

Após encontrar a melhor representação para o problema associado com as possíveis respostas do sistema, foi utilizado o algoritmo do fuzzy c-means para encontrar os clusters que representavam nosso conjunto de dados e construir o motor de regras a partir da categorização, como pode ser observado na Figura 5.

```
125  cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(alldata, 3, 2, error=0.005, maxiter=1000, seed=42)
126
127  #Show 3-cluster model
128  fig2, ax2 = plt.subplots()
129  ax2.set_title('Sensores laterais versus ângulo do volante')
130  for j in range(3):
131      ax2.plot(alldata[0, u.argmax(axis=0) == j],
132               alldata[1, u.argmax(axis=0) == j], 'o',
133                label='series ' + str(j))
134
135  ax2.legend()
136  plt.show()
```

Figura 5. Código do algoritmo fuzzy c-means para encontrar os clusters do dataset e seus devidos graus de pertencimento.

2.3 Controlador Fuzzy

Após a clusterização do conjunto de dados através do fuzzy c-means foi adotado a estratégia do "especialista" para o motor de regras, visto que a técnica de clusterização para um problema linear não foi útil. A técnica utilizada é passada para o sistema de controle as funções de pertinência de cada valor linguístico das variáveis de entrada e as regras de associação com a saída esperada.

O controlador implementado no simulador utiliza três variáveis de entrada mensuradas a partir da resposta dos sensores: o erro, diferença entre as distâncias laterais, a derivada do erro, quanto o erro variou de um frame para outro e a distância frontal, utilizada para controlar a velocidade do carro. O erro e a derivada do erro detinham cinco valores linguísticos, são eles:

- *Muito direita;*
- *Direita;*

- *Meio;*
- *Esquerda;*
- *Muito Esquerda;*

A ordem apresentada acima está no menor valor para o maior, tendo adotado a direita do carro como o eixo negativo do problema. Já a distância frontal possui apenas dois valores linguísticos, são eles:

- *Perto;*
- *Longe;*

A partir das variáveis de entrada apresentadas acima, criou-se as regras para controlar as duas variáveis de saída do sistema de controle: velocidade e direção. Foram criadas 24 regras para o motor de inferência do controlador, sendo metade delas para determinar a velocidade e a outra a metade para julgar para qual direção o carro deve se mover. O fato da grande quantidade de regras deve-se ao fato de ter utilizado a derivada do erro para aprimorar o controle do carro. As funções de pertinência e as regras implementadas são apresentadas respectivamente pelas Figuras 6 e 7.

```
def controle_fuzzy():

    # Descrição do conjunto universo para os antecedentes e consequentes
    posicao_lateral = ctrl.Antecedent(np.arange(-90, 90, 0.1), 'Posicao Lateral')
    posicao_derivadaerro = ctrl.Antecedent(np.arange(-3000, 3000, 0.1), 'Derivada Erro')
    posicao_frontal = ctrl.Antecedent(np.arange(30, 1120, 0.1), 'Posicao Frontal')
    angulo_volante = ctrl.Consequent(np.arange(-35, 35, 0.1), 'Angulo Volante')
    velocidade = ctrl.Consequent(np.arange(-1, 8, 0.1), 'Velocidade')

    # Funções de pertinência para a derivada do erro
    posicao_derivadaerro['Muito Direita'] = fuzz.trapmf(posicao_derivadaerro.universe, [-3000, -2900, -800, -90])
    posicao_derivadaerro['Direita'] = fuzz.trapmf(posicao_derivadaerro.universe, [-800, -600, -150, 0])
    posicao_derivadaerro['Meio'] = fuzz.trimf(posicao_derivadaerro.universe, [-150, 0, 150])
    posicao_derivadaerro['Esquerda'] = fuzz.trapmf(posicao_derivadaerro.universe, [0, 150, 500, 600])
    posicao_derivadaerro['Muito Esquerda'] = fuzz.trapmf(posicao_derivadaerro.universe, [600, 800, 2900, 3000])
    # Funções de pertinência para os sensores laterais
    posicao_lateral['Muito Direita'] = fuzz.trapmf(posicao_lateral.universe, [-100, -90, -70, -35])
    posicao_lateral['Direita'] = fuzz.trapmf(posicao_lateral.universe, [-40, -20, -10, 0])
    posicao_lateral['Meio'] = fuzz.trimf(posicao_lateral.universe, [-15 , 0, 15])
    posicao_lateral['Esquerda'] = fuzz.trapmf(posicao_lateral.universe, [0, 10, 25, 40])
    posicao_lateral['Muito Esquerda'] = fuzz.trapmf(posicao_lateral.universe, [35, 70, 90, 100])
    # Funções de pertinência para o sensor frontal
    posicao_frontal['Perto'] = fuzz.trapmf(posicao_frontal.universe, [-1, 0, 100, 150])
    posicao_frontal['Longe'] = fuzz.trapmf(posicao_frontal.universe, [120, 450,900,1130])
    # Funções de pertinência para os sensores laterais
    angulo_volante['Muito Direita'] = fuzz.trapmf(angulo_volante.universe, [-33, -30, -15, -10])
    angulo_volante['Direita'] = fuzz.trapmf(angulo_volante.universe, [-12, -8, -2, 0])
    angulo_volante['Meio'] = fuzz.trimf(angulo_volante.universe, [-2, 0, 2])
    angulo_volante['Esquerda'] = fuzz.trapmf(angulo_volante.universe, [0, 2, 8, 12])
    angulo_volante['Muito Esquerda'] = fuzz.trapmf(angulo_volante.universe, [10, 15, 30, 35])
    # Funções de pertinência para o sensor frontal
    velocidade['Devagar'] = fuzz.trimf(velocidade.universe, [0.5, 2.5, 4])
    velocidade['Rapido'] = fuzz.trimf(velocidade.universe, [3, 7, 10])

```

Figura 6. Código das funções de pertencimento para o controlador fuzzy.

```
    #Regras
    rule1 = ctrl.Rule(posicao_lateral['Meio'] & posicao_frontal['Longe'] & posicao_derivadaerro['Meio'], angulo_volante['Meio'])
    rule1_2 = ctrl.Rule(posicao_lateral['Meio'] & posicao_frontal['Longe'] & posicao_derivadaerro['Meio'], velocidade['Rapido'])

    rule2 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Perto'] & posicao_derivadaerro['Muito Esquerda'], angulo_volante['Muito Direita'])
    rule2_2 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Perto'] & posicao_derivadaerro['Muito Esquerda'], velocidade['Devagar']) #Devagar

    rule3 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Perto'] & posicao_derivadaerro['Muito Direita'], angulo_volante['Muito Esquerda'])
    rule3_2 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Perto'] & posicao_derivadaerro['Muito Direita'], velocidade['Devagar']) #Devagar

    rule4 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Longe'] & posicao_derivadaerro['Esquerda'], angulo_volante['Meio'])
    rule4_2 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Longe'] & posicao_derivadaerro['Esquerda'], velocidade['Rapido'])

    rule5 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Longe']  & posicao_derivadaerro['Direita'], angulo_volante['Meio'])
    rule5_2 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Longe'] & posicao_derivadaerro['Direita'], velocidade['Rapido'])

    rule6 = ctrl.Rule(posicao_lateral['Meio'] & posicao_frontal['Perto']  & posicao_derivadaerro['Meio'], angulo_volante['Meio'])
    rule6_2 = ctrl.Rule(posicao_lateral['Meio'] & posicao_frontal['Perto'] & posicao_derivadaerro['Meio'], velocidade['Rapido']) #Devagar

    rule7 = ctrl.Rule(posicao_lateral['Muito Esquerda'] & posicao_frontal['Perto']  & posicao_derivadaerro['Muito Esquerda'], angulo_volante['Muito Direita'])
    rule7_2 = ctrl.Rule(posicao_lateral['Muito Esquerda'] & posicao_frontal['Perto'] & posicao_derivadaerro['Muito Esquerda'], velocidade['Devagar'])

    rule8 = ctrl.Rule(posicao_lateral['Muito Direita'] & posicao_frontal['Perto']  & posicao_derivadaerro['Muito Direita'], angulo_volante['Muito Esquerda'])
    rule8_2 = ctrl.Rule(posicao_lateral['Muito Direita'] & posicao_frontal['Perto'] & posicao_derivadaerro['Muito Direita'], velocidade['Devagar'])

    rule9 = ctrl.Rule(posicao_lateral['Muito Esquerda'] & posicao_frontal['Longe']  & posicao_derivadaerro['Esquerda'], angulo_volante['Direita'])
    rule9_2 = ctrl.Rule(posicao_lateral['Muito Esquerda'] & posicao_frontal['Longe']  & posicao_derivadaerro['Esquerda'], velocidade['Rapido'])

    rule10 = ctrl.Rule(posicao_lateral['Muito Direita'] & posicao_frontal['Longe'] & posicao_derivadaerro['Direita'], angulo_volante['Esquerda'])
    rule10_2 = ctrl.Rule(posicao_lateral['Muito Direita'] & posicao_frontal['Longe'] & posicao_derivadaerro['Direita'], velocidade['Rapido'])

    rule11 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Longe'] & posicao_derivadaerro['Meio'], angulo_volante['Esquerda'])
    rule11_2 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Longe'] & posicao_derivadaerro['Meio'], velocidade['Rapido'])

    rule12 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Longe'] & posicao_derivadaerro['Meio'], angulo_volante['Direita'])
    rule12_2 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Longe'] & posicao_derivadaerro['Meio'], velocidade['Rapido'])

    rule13 = ctrl.Rule(posicao_lateral['Muito Esquerda'] & posicao_frontal['Perto'] & posicao_derivadaerro['Meio'], angulo_volante['Muito Direita'])
    rule13_2 = ctrl.Rule(posicao_lateral['Muito Esquerda'] & posicao_frontal['Perto'] & posicao_derivadaerro['Meio'], velocidade['Devagar'])

    rule14 = ctrl.Rule(posicao_lateral['Muito Direita'] & posicao_frontal['Perto'] & posicao_derivadaerro['Meio'], angulo_volante['Muito Esquerda'])
    rule14_2 = ctrl.Rule(posicao_lateral['Muito Direita'] & posicao_frontal['Perto'] & posicao_derivadaerro['Meio'], velocidade['Devagar'])

    rule15 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Perto'] & posicao_derivadaerro['Esquerda'], angulo_volante['Muito Direita'])
    rule15_2 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Perto'] & posicao_derivadaerro['Esquerda'], velocidade['Devagar']) #Devagar

    rule16 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Perto'] & posicao_derivadaerro['Direita'], angulo_volante['Muito Esquerda'])
    rule16_2 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Perto'] & posicao_derivadaerro['Direita'], velocidade['Devagar']) #Devagar

    rule17 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Perto'] & posicao_derivadaerro['Meio'], angulo_volante['Muito Esquerda'])
    rule17_2 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Perto'] & posicao_derivadaerro['Meio'], velocidade['Devagar'])

    rule18 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Perto'] & posicao_derivadaerro['Meio'], angulo_volante['Muito Direita'])
    rule18_2 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Perto'] & posicao_derivadaerro['Meio'], velocidade['Devagar'])

    rule19 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Longe']  & posicao_derivadaerro['Esquerda'], angulo_volante['Esquerda'])
    rule19_2 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Longe'] & posicao_derivadaerro['Esquerda'], velocidade['Rapido'])

    rule20 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Perto']  & posicao_derivadaerro['Esquerda'], angulo_volante['Muito Esquerda']) #
    rule20_2 = ctrl.Rule(posicao_lateral['Direita'] & posicao_frontal['Perto'] & posicao_derivadaerro['Esquerda'], velocidade['Devagar'])

    rule21 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Longe']  & posicao_derivadaerro['Direita'], angulo_volante['Direita'])
    rule21_2 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Longe'] & posicao_derivadaerro['Direita'], velocidade['Rapido'])

    rule22 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Perto']  & posicao_derivadaerro['Direita'], angulo_volante['Muito Esquerda'])
    rule22_2 = ctrl.Rule(posicao_lateral['Esquerda'] & posicao_frontal['Perto'] & posicao_derivadaerro['Direita'], velocidade['Devagar'])

    rule23 = ctrl.Rule(posicao_lateral['Muito Esquerda'] & posicao_frontal['Longe']  & posicao_derivadaerro['Meio'], angulo_volante['Direita'])
    rule23_2 = ctrl.Rule(posicao_lateral['Muito Esquerda'] & posicao_frontal['Longe']  & posicao_derivadaerro['Meio'], velocidade['Rapido'])

    rule24 = ctrl.Rule(posicao_lateral['Muito Direita'] & posicao_frontal['Longe'] & posicao_derivadaerro['Meio'], angulo_volante['Esquerda'])
    rule24_2 = ctrl.Rule(posicao_lateral['Muito Direita'] & posicao_frontal['Longe'] & posicao_derivadaerro['Meio'], velocidade['Rapido'])
```

Figura 7. Código das regras implementadas no controlador fuzzy.

2.4 Controlador Proporcional Derivativo

Por fim, para realizar comparações com o controlador criado, foi desenvolvido para o mesmo problema um controlador proporcional derivativo para o carro. Um controlador proporcional derivativo (controlador PD) é um mecanismo de loop de controle que emprega *feedback,* sendo amplamente utilizado em sistemas de controle industrial e em uma variedade de outras aplicações que exigem controle modulado contínuo. Como observado na Figura 8 abaixo, um controlador PD calcula continuamente um valor de erro, sendo a diferença entre um dado setpoint e uma variável do processo medida e aplica uma correção baseada em termos de proporção e derivada.

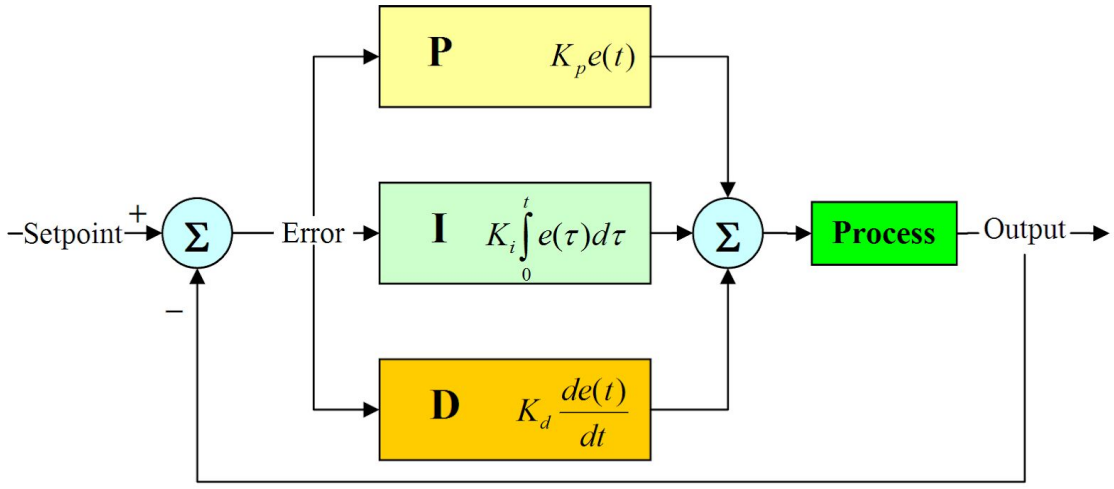

Figura 8. Esquemático de um controlador proporcional integral derivativo.

Fonte – Python ivmech PID Controller [S. l.], 2019.

Para a implementação de um controlador proporcional derivativo no simulador, foi implementada a classe PID já desenvolvida anteriormente por outro usuário. Quando se inicia o controlador em uma variável, é necessário denominar os valores de Kp e Kd para a classe. Ao inicializar o jogo, foi denominado dois controladores para o sistema: um controlador proporcional derivativo para o ângulo do volante e outro controlador proporcional para a velocidade do carro. A cada período de tempo do programa, o controlador é atualizado com o novo erro calculado no sistema. A classe do controlador e a sua implementação no jogo podem ser observadas nas Figuras 9 e 10 respectivamente.

```python
class PID:
    """PID controller."""

    def __init__(self, Kp, Ki, Kd, origin_time=None):
        if origin_time is None:
            origin_time = time.time()

        # Gains for each term
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd

        # Corrections (outputs)
        self.Cp = 0.0
        self.Ci = 0.0
        self.Cd = 0.0

        self.previous_time = origin_time
        self.previous_error = 0.0

    def Update(self, error, current_time=None):
        if current_time is None:
            current_time = time.time()
        dt = current_time - self.previous_time
        if dt <= 0.0:
            return 0
        de = error - self.previous_error

        self.Cp = error
        self.Ci += error * dt
        self.Cd = de / dt

        self.previous_time = current_time
        self.previous_error = error

        return (
            (self.Kp * self.Cp)    # proportional term
            + (self.Ki * self.Ci)  # integral term
            + (self.Kd * self.Cd)  # derivative term
        )

```

Figura 9. Classe do controlador proporcional integral derivativo.

Fonte – Python korfuri PID Controller [S. l.], 2019.

```
#sensores
for i in range(1500):
    x =  np.int(pygame_position[0] + math.cos(math.radians(car.angle)) * i)
    y =  np.int(pygame_position[1] - math.sin(math.radians(car.angle)) * i)
    if (track_mask.get_at([x,y])==1):
        pygame.draw.lines(self.screen,(255,0,0),True,[pygame_position, [x,y]])
        sensor_frontal = math.hypot(x - pygame_position[0], y - pygame_position[1])
        break

for i in range(1500):
     x = np.int(pygame_position[0] + math.cos(math.radians(car.angle+90)) * i)
     y = np.int(pygame_position[1] - math.sin(math.radians(car.angle+90)) * i)
     if (track_mask.get_at([x,y])==1):
         pygame.draw.lines(self.screen,(255,0,0),True,[pygame_position, [x,y]])
         sensor_esquerdo = math.hypot(x - pygame_position[0], y - pygame_position[1])
         break

for i in range(1500):
    x = np.int(pygame_position[0] + math.cos(math.radians(car.angle - 90)) * i)
    y = np.int(pygame_position[1] - math.sin(math.radians(car.angle - 90)) * i)
    if (track_mask.get_at([x,y])==1):
        pygame.draw.lines(self.screen,(255,0,0),True,[pygame_position, [x,y]])
        sensor_direito = math.hypot(x - pygame_position[0], y - pygame_position[1])
        break

distancia = sensor_direito - sensor_esquerdo
error = target_distance - distancia
error_frontal = distancia_ideal - sensor_frontal
correction = controlador_lateral.Update(error)
correction_frontal = controlador_frontal.Update(error_frontal)
car.steering = correction
car.velocity.x = -correction_frontal
car.acceleration = -car.velocity.x / dt
car.acceleration = max(-car.max_acceleration, min(car.acceleration, car.max_acceleration))
car.steering = max(-car.max_steering, min(car.steering, car.max_steering))
```

Figura 10. Implementação do controlador obtido no simulador.

3. Resultados e Discussões

3.1 Simulador de pista de corrida

A partir da utilização do código base para a movimentação do carro e a criação da sistemática da pista de corrida estabelecidas na Seção 2.1, simulou-se o comportamento manual do carro dentro da pista de corrida. O resultado obtido para o simulador encontra-se ilustrado na Figura 11.

Figura 11. Simulador de pista de corrida.

Pela Figura 11 é possível analisar o carro dentro da pista na parte lateral com os 3 sensores formando uma reta vermelha até o primeiro encontro de um pixel de cor preta. O programa consegue calcular as distâncias para os sensores e é possível fazer muitas alterações de forma simples. Inicialmente foi criado mais sensores em diferentes ângulos, porém foi observado que esses sensores não seriam necessários para o sistema, visto que os três sensores disponíveis já conseguiam controlar o carro na pista.

Vale também lembrar que ao fechar o programa, é criado 3 arquivos *.csv* que salvam as distâncias dos sensores, ângulo e velocidade do carro para cada período de tempo. A fim de não reescrever os dados para as simulações posteriores, foram criados mais 2 códigos em *python* para o controlador fuzzy e controlador proporcional derivativo sem a criação dos arquivos *.csv*.

3.2 Clusterização de dados obtidos

Ao executar o programa de simulação, obteve-se o resultado relacionado ao erro do sistema, sendo de entrada a diferença entre os valores apresentados pelos sensores laterais, com a saída esperada, o ângulo do volante. O gráfico pode ser observado pela Figura 12.

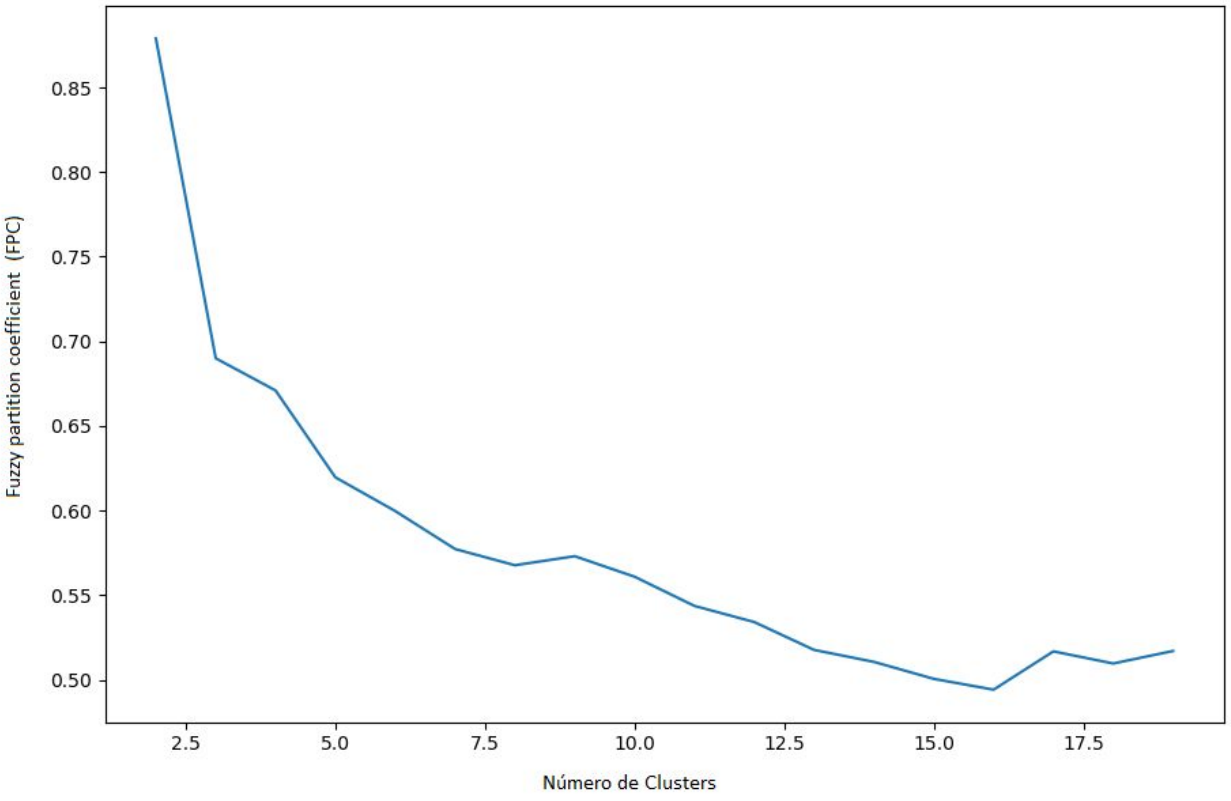

Figura 12. Possíveis quantidades de *clusters* para a relação do erro dos sensores laterais com o ângulo do volante e seus resultados no valor de *fpc*.

Ao analisar a Figura 12, nota-se que o valor que melhor representa o conjunto de dados é quando a quantidade de clusters se equivale a 2. Em contrapartida, ao considerar as possíveis saídas do sistema, percebe-se que a resposta quando se possui 2 clusters não se descreve integralmente as saídas, visto que possui ao menos três saídas possíveis: girar o volante a esquerda, girar o volante a direita ou deixar reto. Portanto, utilizou-se o segundo melhor valor apresentado na figura, quando o valor de clusters equivale a 3. O resultado do algoritmo de clusterização utilizando 3 clusters é apresentado na Figura 13.

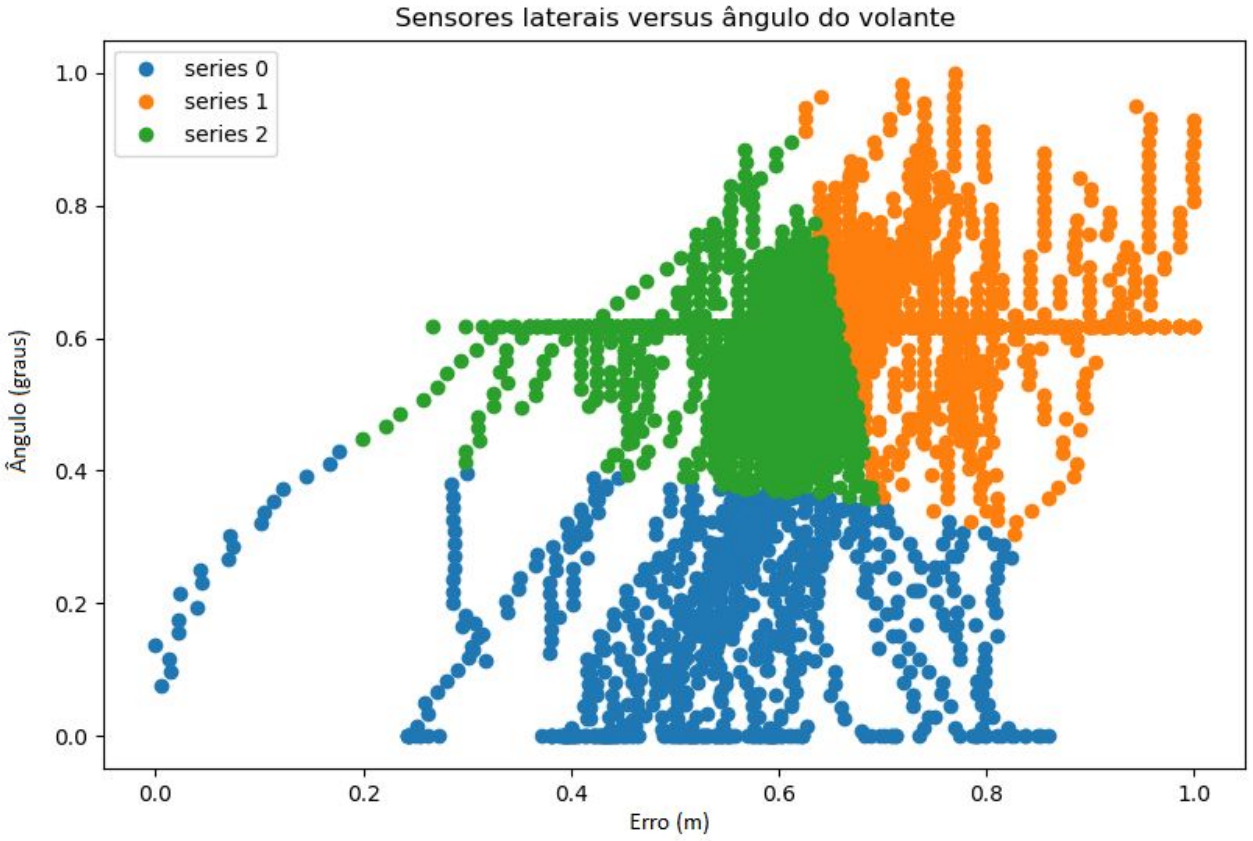

Figura 13. Clusters do erro dos sensores laterais com o ângulo do volante.

Analisando a resposta do nosso algoritmo, nota-se que não foi obtido *clusters* independentes, sendo divididos em categorias distintas por imposição dos próprios hiperparâmetros. Essa resposta condiz muito com a física do problema. Um problema linear e que implica numa condição de causalidade, ou seja, o evento presente é influência do evento passado e, portanto, inviabilizando a premissa em separar categorias distintas. Repetindo a mesma estratégia para visualizar o cluster da distância frontal com a velocidade, pode-se observar o resultado na Figura 14.

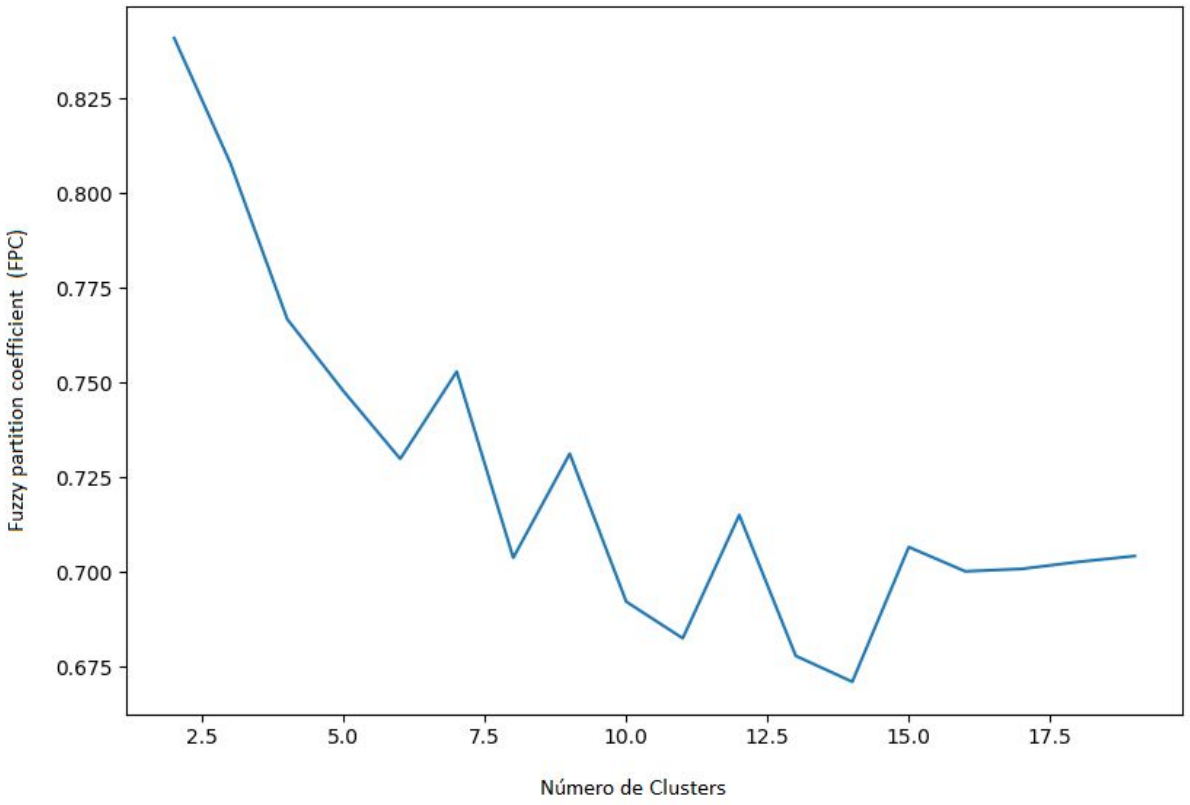

Figura 14. Possíveis quantidades de *clusters* para a relação da distância do sensor frontal com a velocidade do carro e seus resultados no valor de *fpc*.

Neste contexto, ao considerar a física do problema, nota-se que a resposta de dois *clusters* satisfaz o controle do carro, visto que foi adotado a estratégia de descrever as possíveis saídas como perto ou longe. A resposta da clusterização é ilustrada na figura 15.

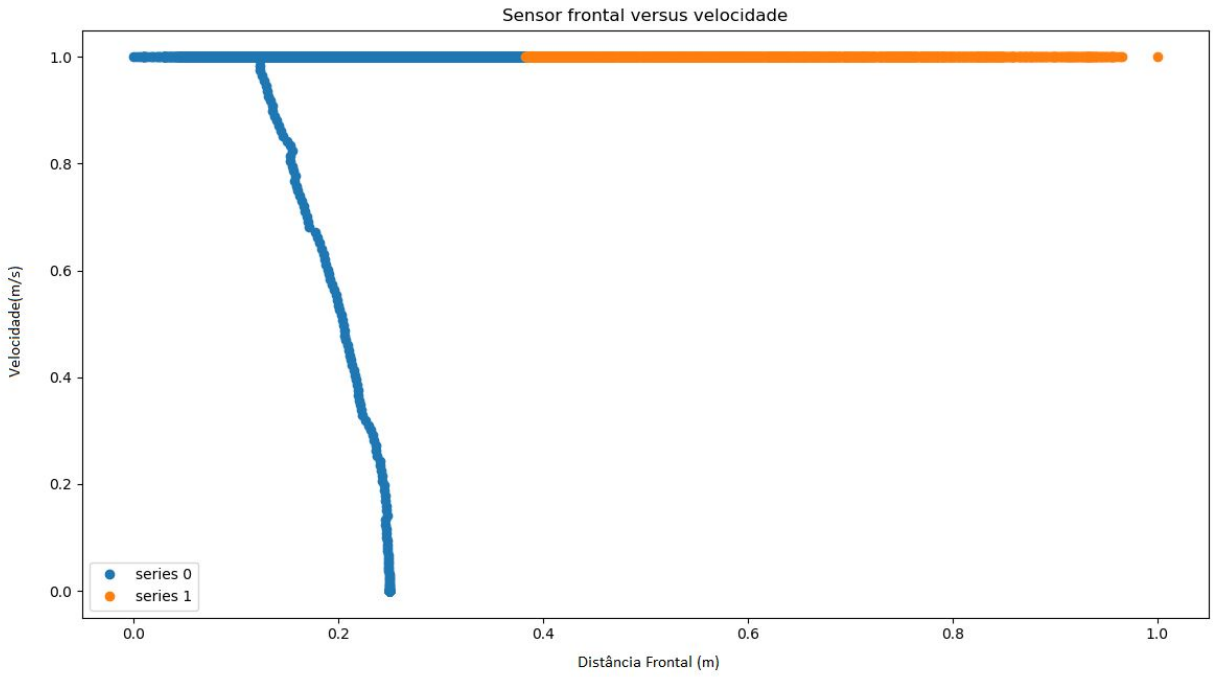

Figura 15. Clusters da distância do sensor frontal com o ângulo do volante.

Ao analisar o resultado obtido dessa clusterização, nota-se que os dados fazem sentido com o simulador em si, visto que os valores são crescentes e estabilizam na velocidade máxima do jogo. Assim como ocorreu na primeira tentativa de clusterizar os dados, outro problema linearmente dependente e causal é observado, inviabilizando a categorização dos dados por meio do fuzzy c-means.

3.3 Controlador Fuzzy

Após constatar que a técnica de clusterização não auxiliaria na implementação do controlador, foi abordado a técnica do especialista, técnica que consiste em passar as funções de pertinência específicas para o contexto do problema e os valores linguísticos associadas a ela para sistema de inferência do controlador. Assim sendo, foi atingido um resultado satisfatório para o controlador, apresenta-se o sinal de controle na Figura 16.

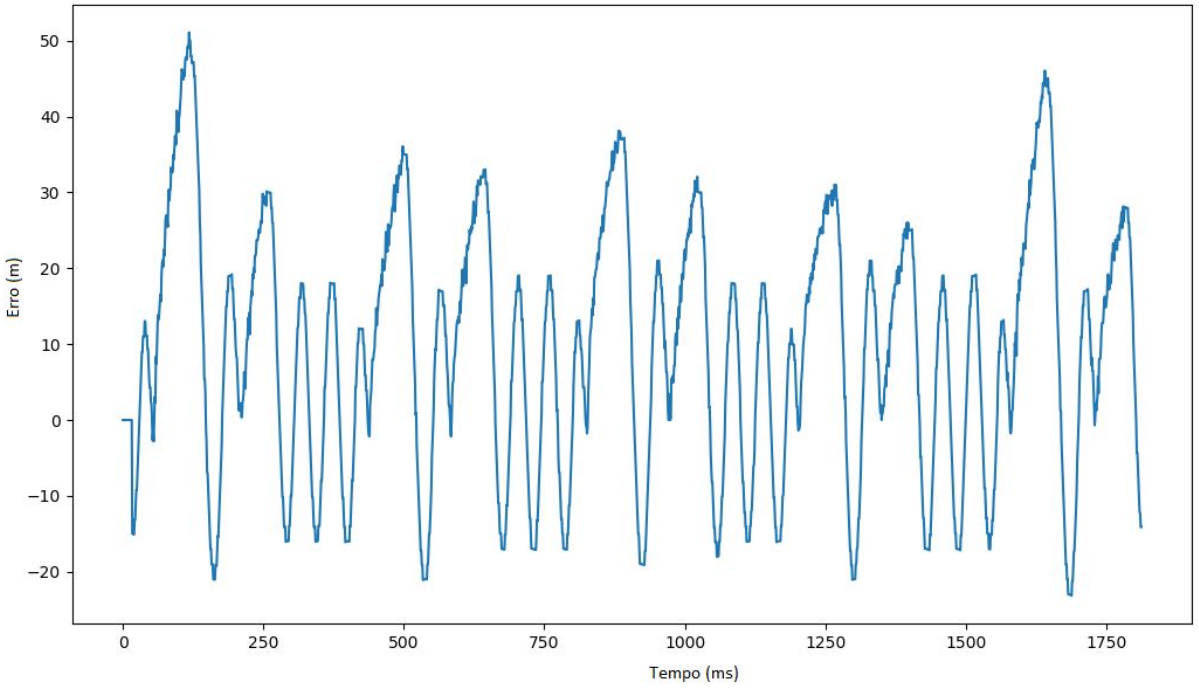

Figura 16. Sinal de controle do fuzzy ao longo do tempo

Percebe-se que o sinal de controle apresentou forma oscilatória, o que não é ideal para um controlador, porém conseguiu atuar de certa forma para minimizar o erro. Nas partes onde o sinal mais se aproxima de zero é quando o carro se encontra nas retas e consegue corrigir o erro remanescente da curva anterior, permitindo que o carro continue em movimento constante e sem colisões.

Foi julgado o resultado como satisfatório, visto que a teoria por trás da lógica e do controle fuzzy preza por um controle mais genérico, na medida que foi utilizado valores linguísticos para operar o sinal, e por um controle mais impreciso, que em contrapartida é muito mais aplicável a problemas extremamente complexos e que necessitam de um respaldo físico e matemático por trás. Além disso, com poucas linhas de códigos, consegue-se desenvolver um controlador que consiga atuar em duas diferentes variáveis, velocidade e direção, de maneira eficiente e sem a complexidade associada às teorias de controle tradicional.

Por fim, avalia-se que o controle fuzzy é um recurso extremamente interessante para disponibilizar uma solução simples a problemas complexos e que demandam amplo conhecimento técnico sobre o contexto. Por exemplo, não foi necessário quantificar nenhuma equação matemática que representasse

o movimento do carro, apenas aplicando os princípios de condução de automóveis e especificidades da pista.

3.4 Controlador Proporcional Derivativo

A partir da utilização do código base do controlador PID, foram implementados 2 controladores para realizar a movimentação do carro. Os valores de ganho para cada valor proporcional e derivativo no sistema foi feito de forma manual com o objetivo de chegar ao melhor equilíbrio do carro no meio da pista de corrida. Sendo assim, foi encontrado para o controle lateral os valores de ganho proporcional (*Kp*) igual à 1 e ganho derivativo (*Kd*) igual à 0,1. Já para o controle frontal, foi utilizado um controlador proporcional com ganho (*Kp*) igual à 1.

Como as escolhas do valor de ganho foram obtidas de forma manual, foi determinado que o controlador não iria possuir o fator integral, pois esse fator acumula o erro ocasionando um overshoot no sistema e consequentemente fazendo que o carro saia da pista. Para evitar esse tipo de comportamento, seria necessário obter outra variável para o windup, dificultando ainda mais a obtenção manual dos valores para o sistema. A implementação dos controladores frontais e laterais e o resultado do sistema para o controlador lateral podem ser observados nas Figuras 17 e 18.

```
122     def run(self):
123         car_image = pygame.image.load('car.png')
124         car_image = pygame.transform.scale(car_image, (45, 20))
125         track_image = pygame.image.load('race_track.png')
126         track_image = pygame.transform.scale(track_image, (1280, 720))
127         track_mask = pygame.mask.from_threshold(track_image,pygame.Color('black'), (1, 1, 1, 255))
128         car = Car(30, 1.5)
129         ppu = 32
130         #data = np.array([])
131         dataSensors = np.zeros(3)
132         dataSteering = np.zeros(1)
133         dataVel = np.zeros(1)
134         steer=0
135         sensor_esquerdo = 0
136         sensor_direito = 0
137         sensor_frontal = 0
138         distancia = 0
139         controlador_lateral = PID(1, 0, 0.1)
140         target_distance = 0
141         controlador_frontal = PID(1, 0, 0)
142         distancia_ideal = 10
143
```

Figura 17. Implementação do controlador proporcional derivativo para controle lateral e proporcional para controle frontal no simulador.

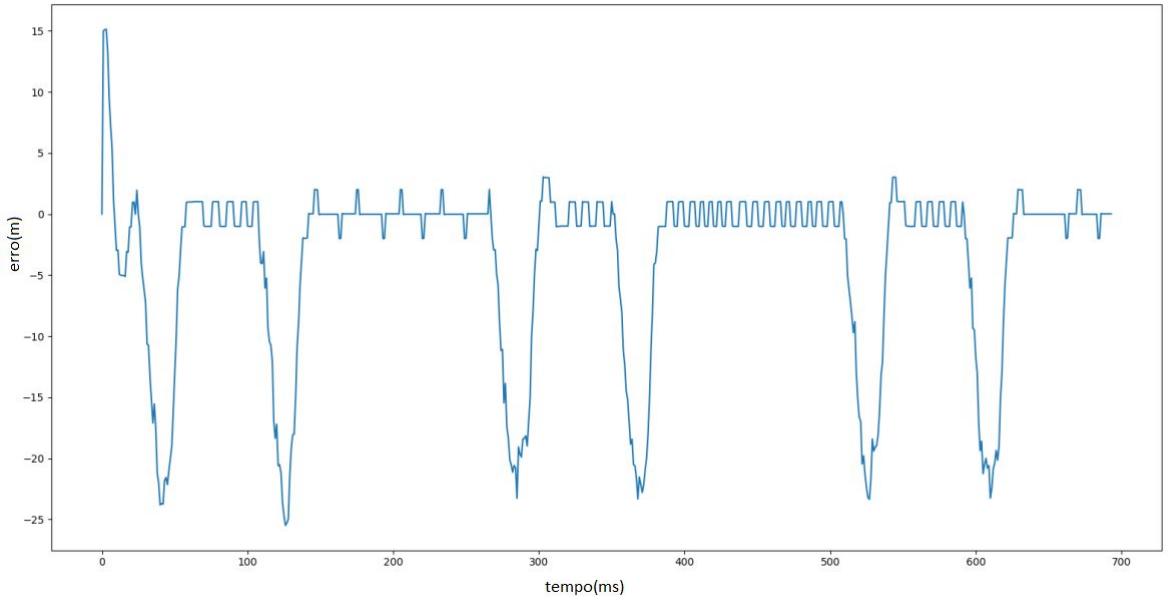

Figura 18. Gráfico do erro dos sensores laterais em relação ao tempo percorrido no jogo.

Pode ser observado pela Figura 18 que o controlador implementado foi bem sucedido. Para esse simulador, pode-se inferir que o controle utilizando um controlador clássico pode ser muito eficaz também, pois controladores PD são controladores automáticos que trabalham bem se o processo é linear. Em contrapartida, para obter um resultado razoável, foi necessário testar diferentes valores de ganho a fim de conseguir controlar o sistema. Como o controle fuzzy nasce da experiência e de experimentos, em vez de modelos matemáticos, uma implementação baseada em regras é muito mais rápida e efetiva de se implementar.

4. Conclusões

Em suma, foi possível perceber a simplicidade que o controlador fuzzy proporciona ao usuário em desenvolver o sistema. Tanto funções de controle lineares quanto não-lineares podem ser implementadas por um sistema baseado em regras, usando o conhecimento de um especialista no assunto a ser abordado.

Os controladores PID são controladores automáticos que trabalham bem se o processo é linear. Mesmo o controlador clássico proporcional derivativo obtendo uma resposta mais estável, foi necessário testar diversos valores de ganho para realizar um controle adequado. Pode-se inferir que uma implementação linguística, como é feito na lógica fuzzy, é muito mais rápida e efetiva de se implementar.

Ao utilizar o método de clusterização de dados obtidos manualmente para adequar as regras fuzzy para o controlador, foi observado que o resultado não é satisfatório. Isso deve-se ao fato do sistema ser linear, fazendo com que não consiga identificar de forma concisa grupos de pertencimento para certos dados. Em contrapartida, o método é muito eficiente para identificar clusters quando os dados são muito mais esparsos e aleatórios no ponto de vista do usuário.

Referências Bibliográficas

1. LabVIEW PID and Fuzzy Logic Toolkit User Manual [S. l.], 2009. Disponível em: http://www.ni.com/pdf/manuals/372192d.pdf. Acesso em: 20 out. 2019.

2. Py Game 2D Car Tutorial [S. l.], 2019. Disponível em: https://github.com/rasmaxim/pygame-car-tutorial. Acesso em: 25 jun. 2020.

3. Python ivmech PID Controller [S. l.], 2019. Disponível em: https://github.com/ivmech/ivPID. Acesso em: 5 jul. 2020.

4. Python korfuri PID Controller [S. l.], 2019. Disponível em: https://github.com/korfuri/PIDController. Acesso em: 5 jul. 2020.

5. Weber, T. ***Tópicos Especiais em Instrumentação I:*** *Controladores Fuzzy*, 25 de jun. de 2020. 20 f. Notas de Aula. Universidade Federal do Rio Grande do Sul.

6. Weber, T. ***Tópicos Especiais em Instrumentação I:*** *Clusterização: K-means, Fuzzy C-means*, 25 de jun. de 2020. 27 f. Notas de Aula. Universidade Federal do Rio Grande do Sul.