UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DE ENGENHARIA ELETRÔNICA

FUNDAMENTOS DE VEÍCULOS AUTÔNOMOS

TRABALHO 1: SIMULAÇÃO DE CONTROLE E ATUAÇÃO DE UM VEÍCULO AUTÔNOMO NO COPPELIASIM

Marcone Márcio da Silva Faria 2019021573

1. INTRODUÇÃO

No presente trabalho serão abordadas técnicas relacionadas ao controle e à autonomia de veículos. Começando com o cenário de uma pista reta sem inclinação, o objetivo foi obter os parâmetros do modelo longitudinal do veículo e projetar um sistema de controle capaz de manter uma referência de velocidade constante, levando em consideração incertezas nos parâmetros e distúrbios externos. Posteriormente, a análise foi expandida a outros cenários, como a pista inclinada e o circuito, incorporando o controle lateral em trajetórias variáveis.

Além disso, sistemas de visão computacional foram explorados para medir a orientação do veículo em relação à pista e identificar os limites de velocidade por meio da leitura das placas de trânsito. A integração desses elementos resultou em simulações abrangentes, resultando na avaliação do desempenho dos controladores e do sistema visual em cenários de pista de corrida, mantendo o veículo dentro da faixa de rodagem e respeitando os limites de velocidade de acordo com a referência. Este trabalho também possui anexos complementares com códigos e análises complementares a esse relatório como descrito na seção de anexos.

2. PARÂMETROS DO MODELO LONGITUDINAL

Para a estimação dos parâmetros do modelo longitudinal primeiramente foram coletados os dados da simulação considerando a pista plana. Nesse sentido, em um intervalo de 3 segundos foi setado o torque do motor para vinte por dois segundos e em seguida esse torque foi setado para zero.

```python
while car.t < 3.0:
    # lê senores
```

```
car.step()
car.setU(0.0)

if car.t < 2.0:
    car.setU(20.0)
else:
    car.setU(0.0)

vel.append(car.v)
time.append(car.t)

# lê exibe gráfico
plt.clf()
plt.plot(time, vel, 'r')
plt.show()
plt.pause(0.01)
```

Com os dados coletados, os mesmos foram analisados utilizando o Matlab [1], de modo a obter os parâmetros do modelo em si. Na figura 1 abaixo temos uma representação gráfica desses dados em função do tempo, seja da velocidade do carro ou da potência do motor.

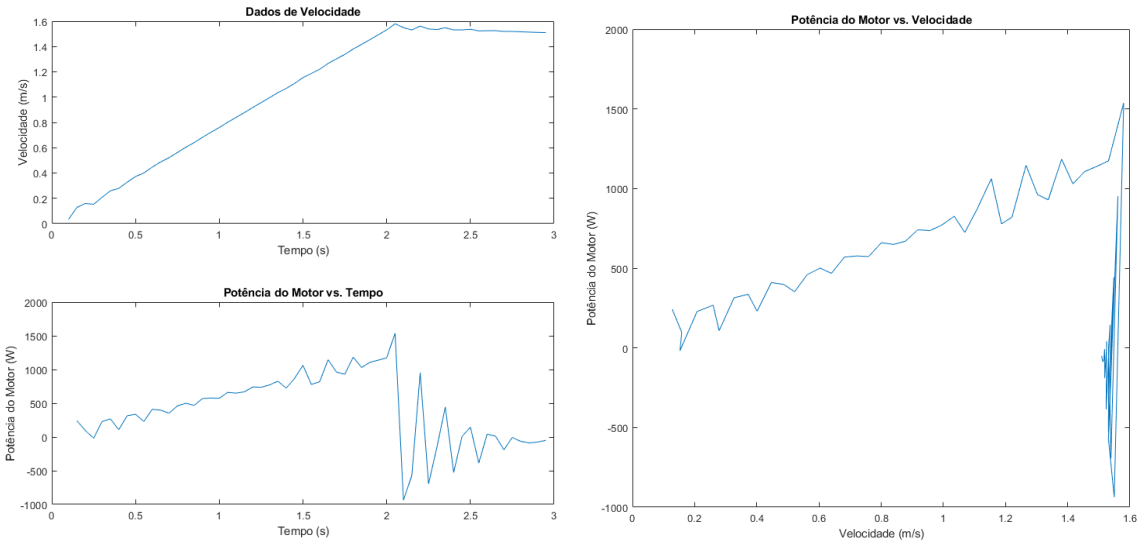

Figura 1: Análise dos dados coletados na simulação.

Em seguida foram definidos alguns parâmetros experimentais do carro, como a massa e os coeficientes de arrasto e resistência de rolamento, sendo para esses dois últimos definidos valores próximos aos encontrados na indústria automobilística. Esses parâmetros são então ajustados com relação ao cálculo aproximado das forças de resistência atuantes no veículo e da potência que seria necessária para vencer essas resistências. De posse dessas análises é possível obter então o modelo linear ajustado considerando os dados experimentais coletados, como mostrado na figura 2.

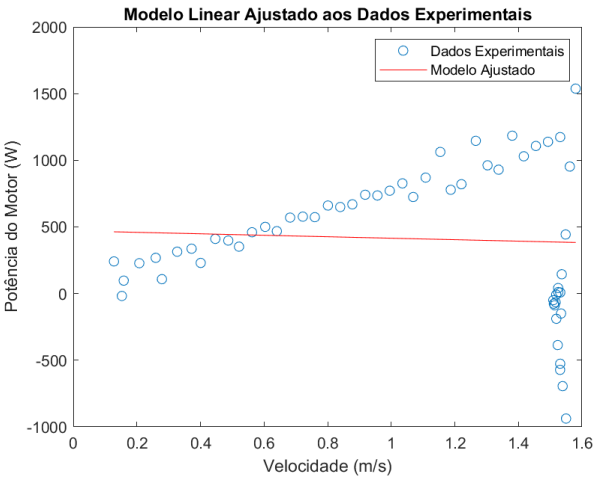

Figura 2: Representação do modelo linear ajustado.

Com base nesse modelo, os parâmetros CDA e CRR são otimizados, sendo as resistências do sistema novamente calculadas com relação a essa otimização. Agora com base no modelo ajustado, plotamos os mesmos gráficos das grandezas com relação ao tempo mostrados na figura 1, obtendo os resultados abaixo:

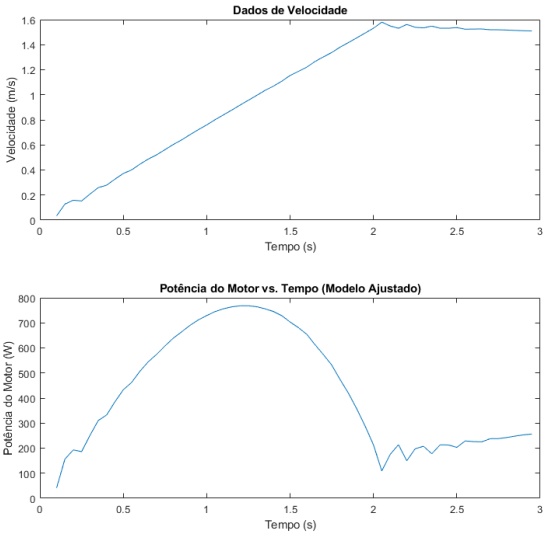

Figura 3: Representação da velocidade e potência ao longo do tempo com relação ao modelo ajustado.

É possível notar com essa representação que a potência do veículo passa a assumir um caráter muito menos oscilatório reagindo muito melhor a queda do torque do motor que ocorre a partir dos dois segundos de simulação, não ocasionando uma queda de potência muito brusca proporcionando uma melhor estabilidade ao carro. Para o coeficiente de resistência ao rolamento (Crr) foi encontrado um valor de 0.12517, enquanto para o coeficiente de arrasto x área Frontal (CdA), foi encontrado um valor de -757.8803, que não é um valor correto ou realista porém no contexto da simulação não foi possível entender qual o erro cometido ou melhorar essa aproximação de uma melhor maneira.

3. CONTROLE LONGITUDINAL

Para o controle longitudinal foi projetado um controlador PID também com a ajuda do Matlab [1] e dos parâmetros calculados anteriormente, sendo também ajustados os ganhos proporcional, integral e derivativo. Para o projeto do controlador foi considerada uma velocidade de referência experimental, no caso vinte metros por segundo constante (aceleração nula) sendo considerados alguns ruídos na medição somente a título de experimentação.

```
Unset
for i = 1:length(tempo)
  % Medida da velocidade (pode ser obtida na prática com sensores)
  velocidade_medida = velocidade_atual(i) + normrnd(0, 0.1);

  % Erro de velocidade
  erro = referencia_velocidade - velocidade_medida;

  % Ação de controle do PID
  integral = integral + erro;
  derivativo = erro - erro_anterior;

  aceleracao = Kp * erro + Ki * integral + Kd * derivativo;

  % Modelagem da dinâmica do veículo
  forca_rolamento = Crr * massa * g;
  forca_arrasto = 0.5 * CdA * densidade_ar * (velocidade_atual(i) ^ 2);

  % Equação de movimento
  forca_resultante = massa * aceleracao + forca_rolamento + forca_arrasto;
  velocidade_atual(i+1) = velocidade_atual(i) + (forca_resultante / massa) *
  (tempo(2) - tempo(1));
  aceleracao_atual(i) = aceleracao;

  erro_anterior = erro;
end
```

Por fim, os dados simulados para o projeto do controlador são então plotados nos gráficos da figura 4. É importante notar na estabilidade do controlador e no tempo de resposta relativamente rápido até atingir os valores de referência para velocidade e aceleração.

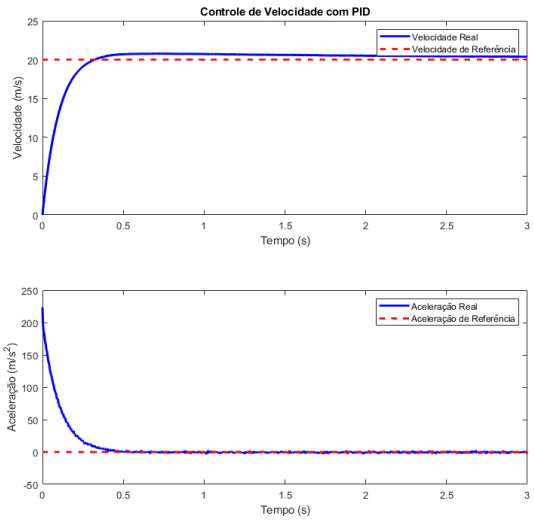

Figura 4: Controle de velocidade do veículo com um controlador PID.

O controlador é então implementado em python para ser de fato testado na simulação do CoppeliaSim nos dois tipos de pista, plana e inclinada.

3.1. Pista Plana

Para o caso da pista plana é possível verificar um overshoot aceitável, demorando um segundo para se estabilizar. O código de ambas as simulações pode ser visto em [2].

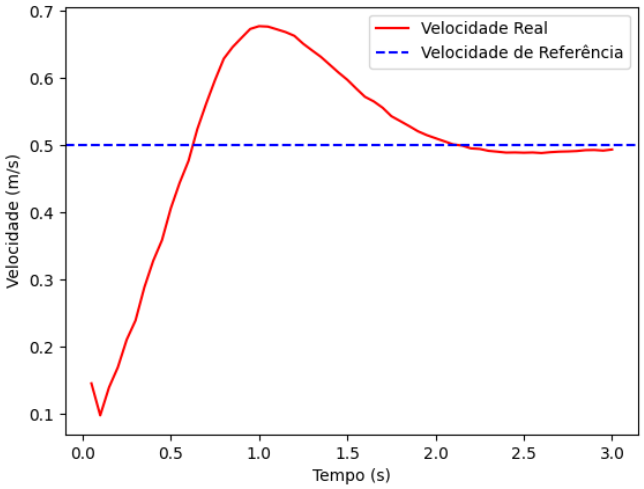

Figura 5: Resposta do sistema a referência no caso da pista plana.

3.2. Pista Inclinada

No caso da pista inclinada o veículo não segue a velocidade de referência durante maior parte do tempo e assim que o carro começa a percorrer a parte inclinada a velocidade despenca, demorando pouco mais de um segundo para que o sistema atue novamente e busque aproximar a velocidade do veículo da referência. É possível notar ainda que mesmo após a estabilização do carro o sistema assume um caráter mais oscilatório, não respondendo bem a inclinação da pista.

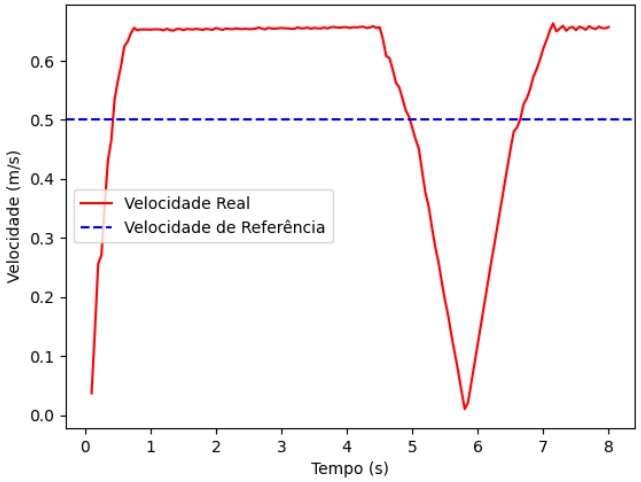

Figura 6: Resposta do sistema a referência no caso da pista inclinada.

4. CONTROLE LATERAL

O comportamento do veículo simulado no CoppeliaSim sem qualquer tipo de controle lateral sobre ele tende a desviar para um dos lados, sem conseguir-se manter em linha reta. É possível notar que todas as perturbações no sistema impedem com que o mesmo siga uma trajetória bem definida. A fim de resolver esse problema, foi projetado um controlador Stanley em python para o seguimento de uma trajetória, como é possível observar no código abaixo:

```python
class StanleyController:
    def __init__(self, k, k_crosstrack, max_steering_angle):
        self.k = k  # Ganho do controlador
        self.k_crosstrack = k_crosstrack  # Ganho do erro lateral
        self.max_steering_angle = max_steering_angle  # Ângulo máximo

    def calculate_steering_angle(self, current_pose, reference_pose=np.array([0, 0])):
        # Calcula o erro lateral (cross-track error) separadamente para as
          #  coordenadas x e y
        erro_x = reference_pose[0] - current_pose[0]
        erro_y = reference_pose[1] - current_pose[1]

        # Calcula o módulo do erro lateral
        erro = math.sqrt(erro_x**2 + erro_y**2)

        # Calcula o ângulo de direção
        steering_angle = self.k * np.arctan2(self.k_crosstrack * erro, 1.0)

        # Limita o ângulo de direção dentro dos limites
        steering_angle = np.clip(steering_angle, -self.max_steering_angle,
                                 self.max_steering_angle)

        return steering_angle
```

4.1. Trajetória em linha reta

No caso do seguimento em linha reta o controlador, apesar de apresentar oscilações, respondeu bem às perturbações, à medida que oscilou em um limite de apenas 6% fora da referência, percorrendo um percurso quase linear. A implementação encontra-se em [3].

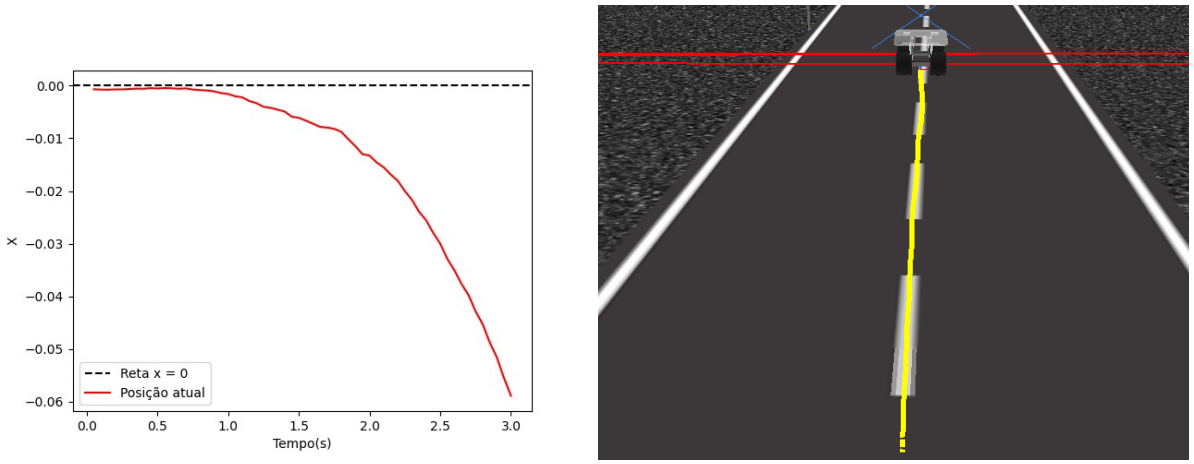

Figura 8: Simulação no CoppeliaSim em um percurso linear.

4.2. Trajetória circular

Considerando o caso da trajetória circular os resultados obtidos foram bem mais promissores, observe que na primeira metade da simulação o controlador segue com precisão a referência, enquanto na segunda metade do percurso o sistema reage mais rápido do que a referência. A implementação dessa simulação pode ser analisada em [4]

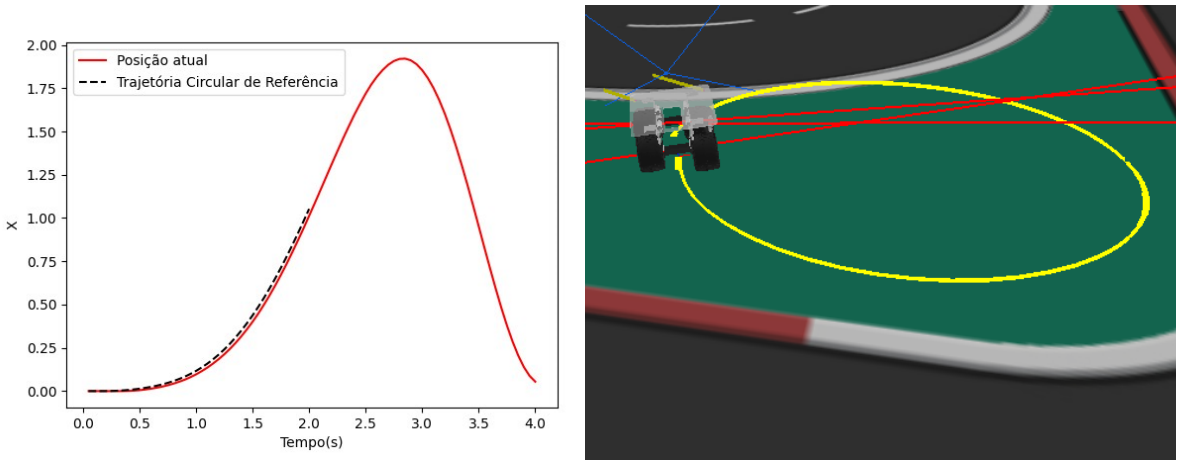

Figura 9: Simulação no CoppeliaSim em um percurso circular.

5. ATUAÇÃO CONJUNTA

Por fim, foi simulado a atuação conjunta dos controladores projetados. Pode-se observar que ambos obtiveram um bom resultado, em especial o controlador PID que realiza o controle longitudinal do veículo. Para o caso do controlador lateral foi possível verificar uma pequena piora dos resultados, mesmo ajustando os ganhos após o primeiro segundo de simulação o carro tende a se desviar ligeiramente da trajetória. A implementação pode ser vista com detalhes em [5].

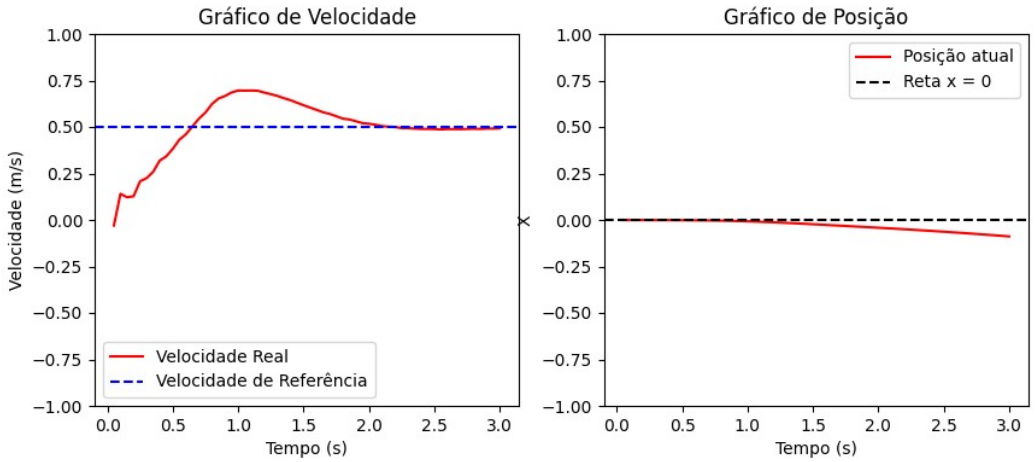

Figura 9: Simulação no CoppeliaSim em um percurso circular.

```python
# Erro de velocidade para o controlador longitudinal
    erro_longitudinal = velocidade_referencia - car.v

    # Ação de controle do PID (controlador longitudinal)
    integrated_error_longitudinal += erro_longitudinal
    derivative_error_longitudinal = erro_longitudinal - previous_error_longitudinal
    control_signal_longitudinal = (
        Kp_longitudinal * erro_longitudinal +
        Ki_longitudinal * integrated_error_longitudinal +
        Kd_longitudinal * derivative_error_longitudinal
    )

    # Calcula o ângulo de direção com base no controlador lateral (Stanley)
    current_pose = np.array([car.getPos()[0], car.getPos()[1]])
    reference_pose = np.array([0, 0])
    steering_controller = StanleyController(k_lateral, k_crosstrack, max_steering_angle)
    steering_angle = steering_controller.calculate_steering_angle(current_pose, reference_pose)

    # Define a abertura do acelerador e o ângulo de direção com base nos controles
    car.setU(control_signal_longitudinal)
    car.setSteer(steering_angle)

    # Atualiza o erro anterior para o controlador longitudinal
    previous_error_longitudinal = erro_longitudinal
```

6. ATUAÇÃO COM VISÃO COMPUTACIONAL

7. ATUAÇÃO CONJUNTA COM VISÃO COMPUTACIONAL

8. SEGUIMENTO DO CIRCUITO

9. ANEXOS

[1] Documento “Parametros_Modelo_Logitudinal.pdf” contendo o Live Script detalhando o código utilizado para a estimação dos parâmetros do modelo longitudinal.

[2] Código “controleLogitudinal.py” contendo o código em python utilizado para a simulação do controle longitudinal no CoppeliaSim.

[3] Código “controleLateral.py” contendo o código em python utilizado para a simulação do controle lateral utilizando o controlador Stanley no CoppeliaSim na trajetória linear.

[4] Código “controleCircularLateral.py” contendo o código em python utilizado para a simulação do controle lateral tomando como base a trajetória simulada obtida em “referenciaCircular.csv”.

[5] Código “controleConjunto.py” contendo o código em python utilizado para a simulação conjunta dos dois controladores projetados.