

# Sistema de Controle III

## Projeto Final

### Robô Equilibrista



FACULDADE DE  
ENGENHARIA

#### **Discentes:**

**1 - Gabriel Ribeiro Bastos de Sousa Rebouças**

**| RGA: 201821902004**

**2 - Jonathan Victor Da Silva Santos**

**| RGA: 201911902014**

**3 - João Gabriel da Silva Ferreira**

**| RGA: 201821902009**

# Índice

**01**

**Introdução**

**02**

**Objetivos**

**03**

**Construção**

**07**

**Desafios**

**04**

**Programação  
Arduino**

**05**

**Montagem**

**06**

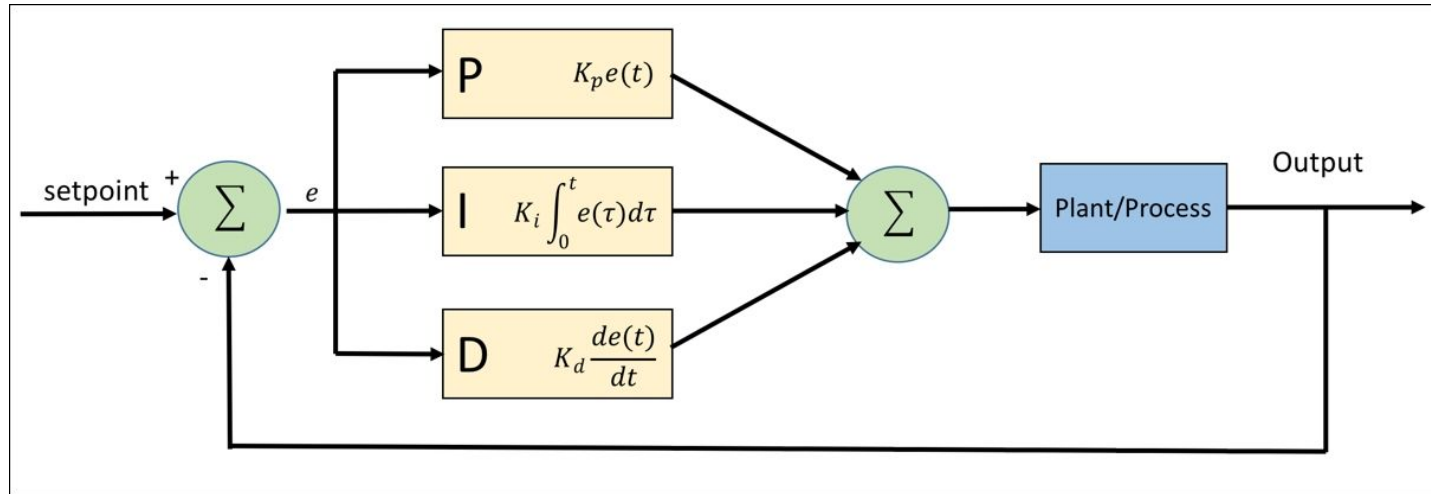
**Teste**

# 1. Introdução

- **1. Papel Vital do Sistema PID na Engenharia de Controle:** Destacamos como o sistema PID é um elemento essencial no controle de sistemas dinâmicos.
- **2. Desafio Intrincado da Autoequilibração do Robô Segway:** Abordamos a complexidade de manter o equilíbrio constante em um robô segway, especialmente durante seu deslocamento autônomo.
- **3. Controle Dinâmico para Precisão e Eficiência:** Exploramos como o sistema PID desempenha um papel fundamental na busca pela precisão e eficiência do controle, adaptando continuamente as ações com base no feedback sensorial.
- **4. Demonstração da Tecnologia de Controle em Ação:** Ilustramos como a aplicação do sistema PID em um robô segway é um exemplo notável de como a tecnologia de controle permite que máquinas executem tarefas complexas, como locomoção autônoma e equilíbrio dinâmico.

# 1. Introdução

- Um controlador PID é um instrumento usado em aplicações de controle industrial e
- Usam um mecanismo de feedback de loop de controle para controlar as variáveis do processo e são os controladores mais precisos e estáveis.



# 1. Introdução

- **Controle Proporcional:** Aqui o **signal de atuação** para a ação de controle em um sistema de controle é **proporcional ao signal de erro**. O signal de erro é a diferença entre o signal de entrada de referência e o signal de feedback obtido da entrada.
- **Controle Derivativo:** O **signal de atuação** consiste em **signal de erro proporcional somado com derivativo do signal de erro**. Portanto, o signal de atuação para a ação de controle derivativo é dado por:

$$e_a(t) = e(t) + T_d \frac{de(t)}{dt} \quad \text{onde } T_d \text{ é uma constante.}$$

- **Controle Integral:** Para ação de controle integral o signal de atuação **consiste em signal de erro proporcional somado com integral do signal de erro**. Portanto, o signal de atuação para a ação de controle integral é dado por:

$$e_a(t) = e(t) + K_i \int e(t) dt \quad \text{onde } K \text{ é uma constante.}$$

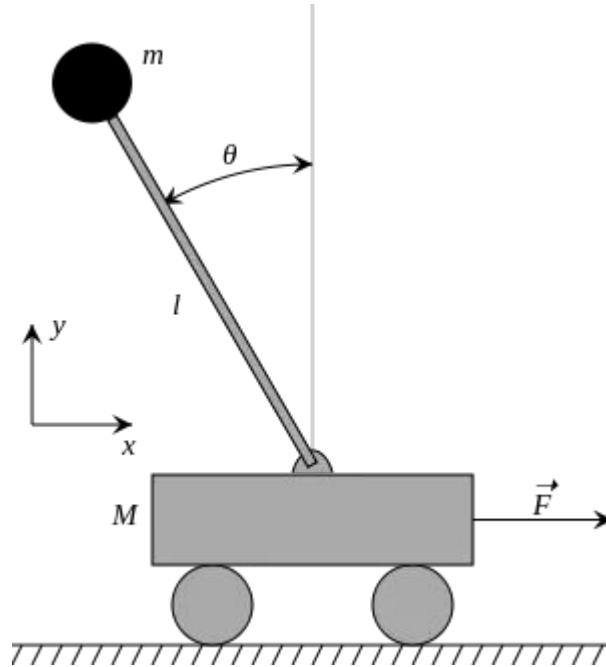
# 1. Introdução

Resposta de controle	Tempo de Subida	Tempo de Acomodação	Overshoot	Erro de Estado Estacionário
Kp	Diminui	Pequena Mudança	Aumenta	Diminui
Kd	-----	Diminui	Diminui	Não altera
Ki	Diminui	Aumenta	Aumenta	Eliminado

## 2. Objetivos

- **1. Estabilidade do Equilíbrio:** O principal objetivo de um sistema PID em um robô segway é manter a estabilidade do equilíbrio do robô, garantindo que ele permaneça na posição vertical, evitando quedas e oscilações excessivas.
- **2. Resposta Rápida a Distúrbios:** O sistema PID visa garantir que o robô possa responder rapidamente a distúrbios, como mudanças na inclinação do terreno ou perturbações externas, restaurando o equilíbrio de forma eficaz.
- **3. Suavidade de Movimento:** Um dos objetivos é garantir que o robô se mova suavemente e com transições fluidas entre aceleração, desaceleração e mudanças de direção, proporcionando uma experiência de passeio segura e confortável.
- **4. Consumo de Energia Eficiente:** Um sistema PID busca otimizar o consumo de energia do robô segway, ajustando o esforço necessário para manter o equilíbrio, o que é vital para a autonomia da bateria e a eficiência operacional do dispositivo.

### 3. Construção do Segway



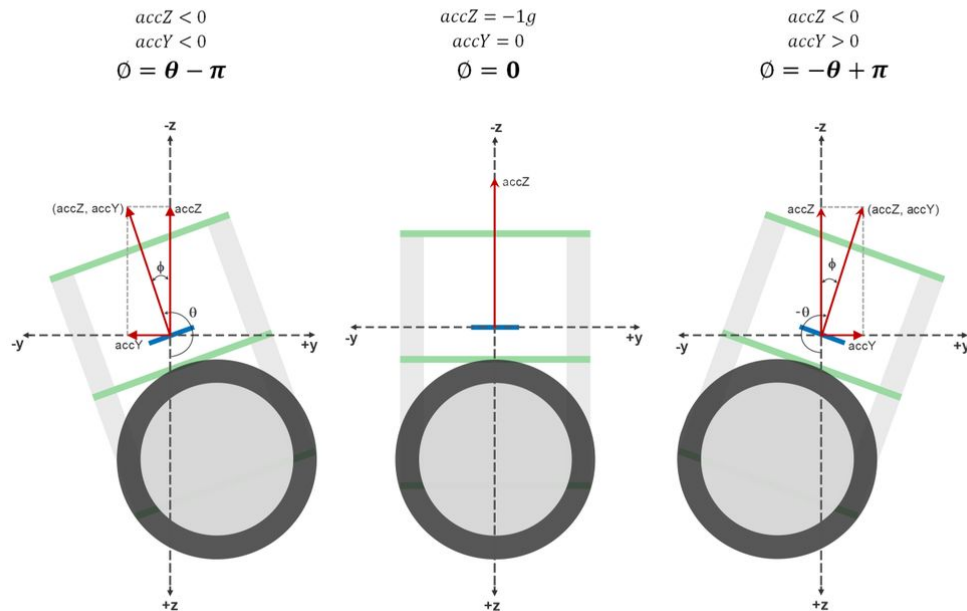
Semelhante a um pêndulo invertido, mas ao contrário de um pêndulo normal que continua balançando depois de movido, este pêndulo invertido não pode permanecer equilibrado por si só.



# 3. Construção do Segway

## O que estamos tentando fazer aqui?

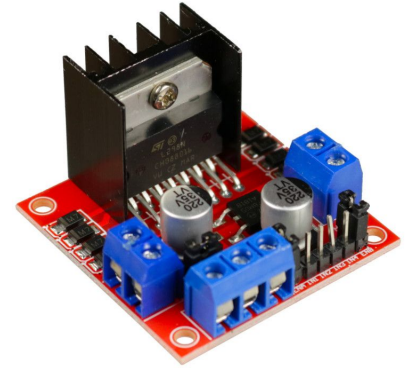
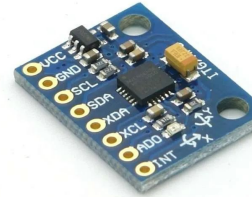
Manter o centro de gravidade exatamente acima do ponto de articulação, esse processo é chamado de cinemática inversa porque o robô faz o oposto de sua tendência.



# 3. Construção do Segway

## Hardware:

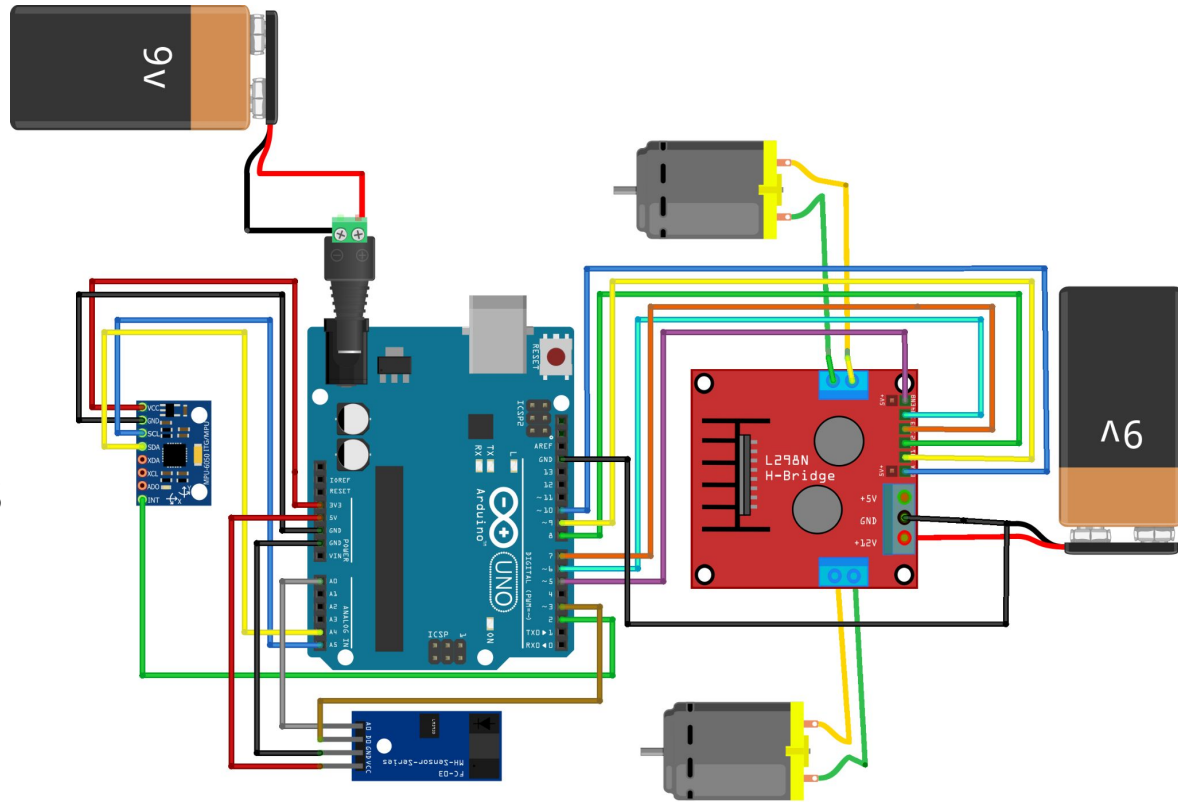
- Arduino Uno
- MPU6050
- 2 motores DC 3-6V
- Ponte H L298N
- 2 Baterias de 9V
- 1 par de rodas
- Sensor de Velocidade FC-03



# 3. Construção do Segway

## Esquemático no Fritzing:

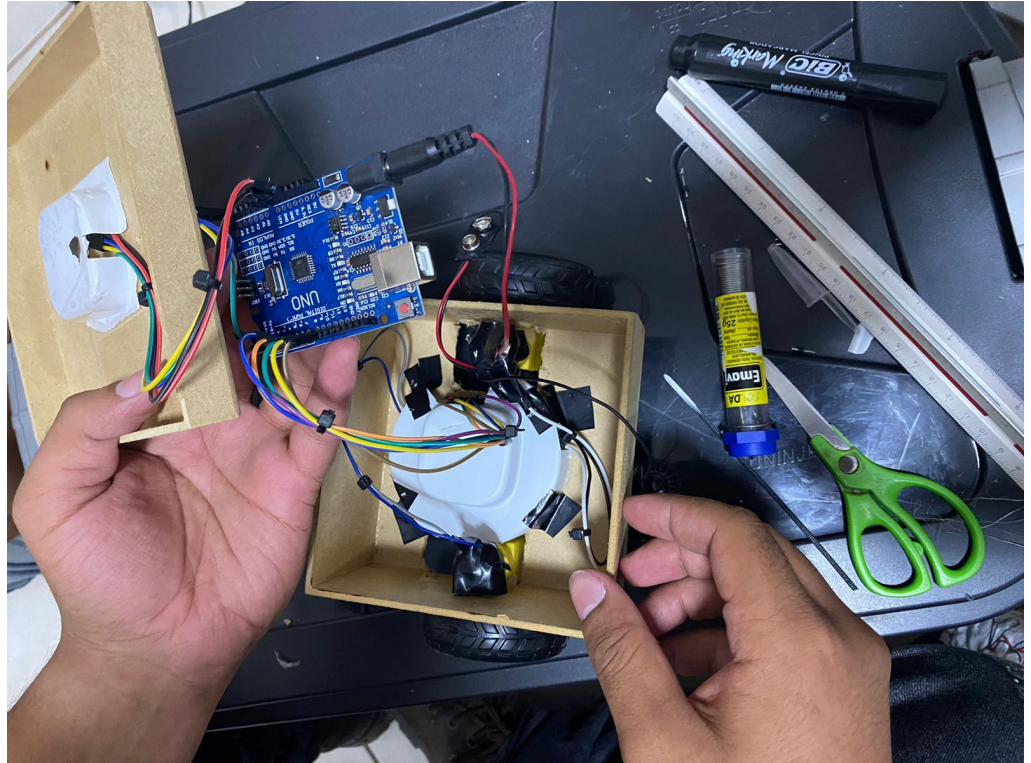
- Arduino Uno
- MPU6050
- 2 motores DC 3-6V
- Ponte H L298N
- 2 Baterias de 9V
- 1 par de rodas
- Sensor de Velocidade FC-03



# 3. Construção do Segway

## Esquemático real:

- Arduino Uno
- MPU6050
- 2 motores DC 3-6V
- Ponte H L298N
- 2 Baterias de 9V
- 1 par de rodas
- Sensor de Velocidade FC-03



# 4. Programação Arduino

## Código Arduino:

```
teste2

1 #include <PID_v1.h>
2 #include <IMotorController.h>
3 #include "I2Cdev.h"
4 #include "MPU6050_6Axis_MotionApps20.h"
5 #include "MPU6050_6Axis_MotionApps_V6_12.h"
6
7 #include <SoftwareSerial.h> //Ver dados do FC-03
8
9 SoftwareSerial gpsSerial(2, 3); // Pinos 2 (RX) e 3 (TX)
10 int gpsBaud = 9600; // Ajuste de acordo com a taxa de baud do sensor FC-03
11
12 #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
13 #include "Wire.h"
14 #endif
15
16 #define MIN_ABS_SPEED 20 // Velocidade minima absoluta
17
18 MPU6050 mpu;
19
20 // Variáveis de controle/status do MPU
21 bool dmpReady = false; // Definido como verdadeiro se a inicialização do DMP for bem-sucedida
22 uint8_t mpuIntStatus; // Armazena o byte real de status de interrupção do MPU
23 uint8_t devStatus; // Status de retorno após cada operação do dispositivo (0 = sucesso, !=0 = erro)
24 uint16_t packetSize; // Tamanho esperado do pacote DMP (o padrão é 42 bytes)
25 uint16_t fifoCount; // Contagem de todos os bytes atualmente na FIFO
26 uint8_t fifoBuffer[64]; // Buffer de armazenamento da FIFO
27
28 // Variáveis de orientação/movimento
```

Compilação terminada.

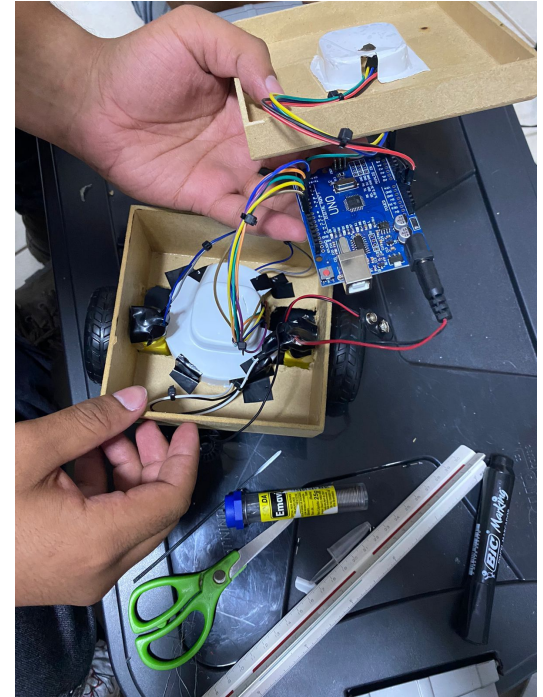
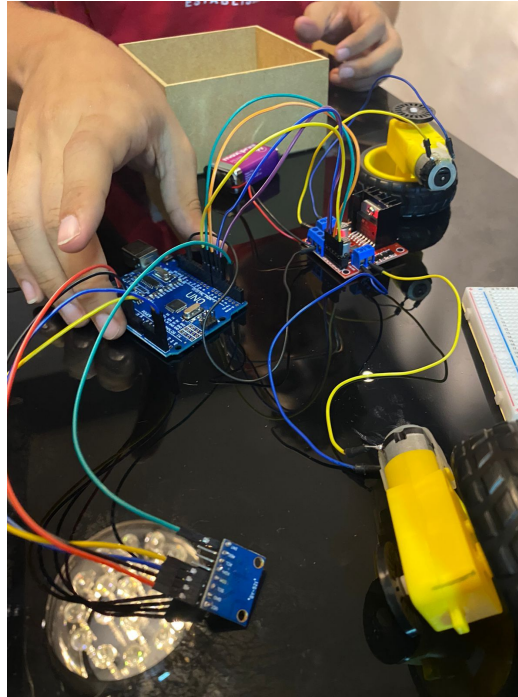
O sketch usa 15550 bytes (48%) de espaço de armazenamento para programas. O máximo são 32256 bytes.  
Variáveis globais usam 760 bytes (37%) de memória dinâmica, deixando 1288 bytes para variáveis locais. O máximo são 2048 bytes.

120 Arduino Uno em COM4

Abrir programa

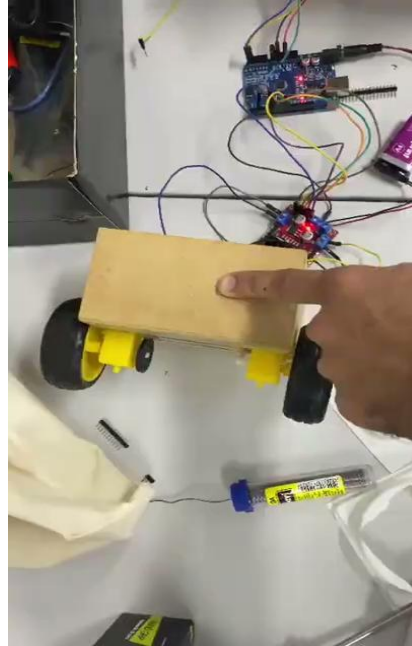
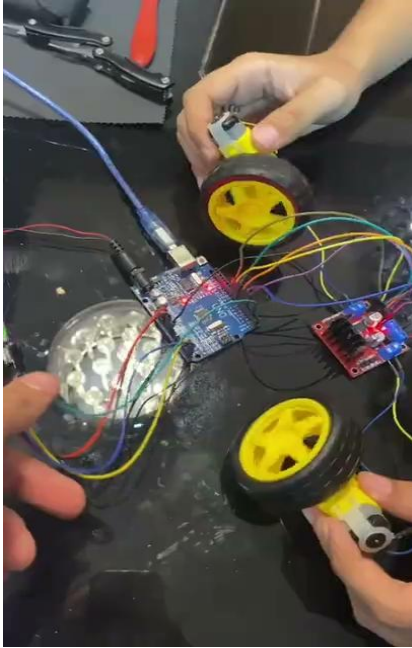


## 5. Montagem



## 6. Testes

**Configuração do código e testes práticos:**



## 6. Testes

**Configuração do código e testes práticos:**





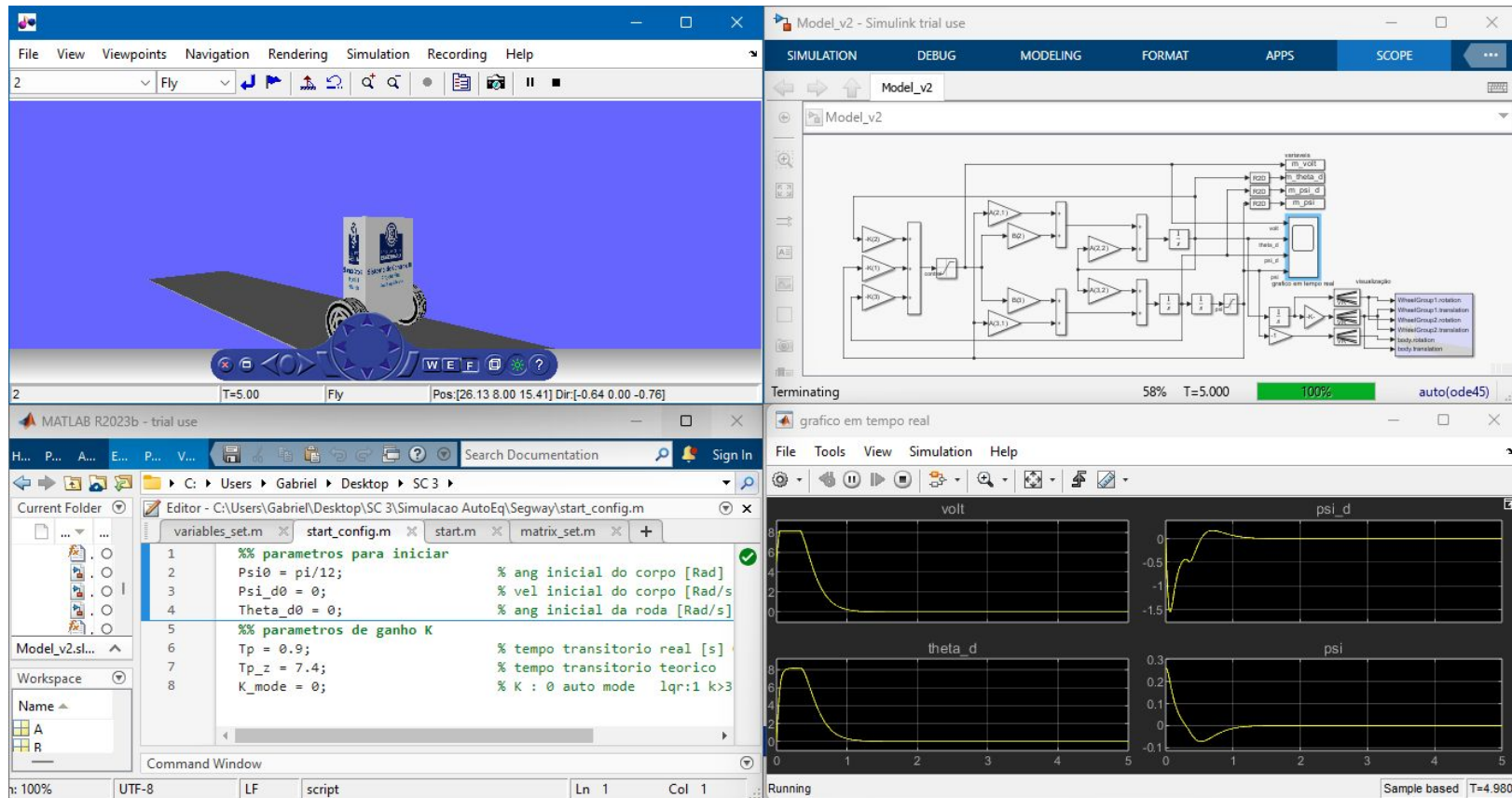
# 7. Desafios

- **1. Integração de Sensores:** O desafio inicial é integrar corretamente o MPU6050 para coletar dados de inclinação e aceleração do robô segway. Isso envolve a comunicação eficaz entre o sensor e o Arduino Uno.
- **2. Controle dos Motores DC:** Configurar o controle dos motores DC (que podem variar de 3-6V) por meio da Ponte H L298N é um desafio, pois requer programação precisa para garantir a resposta apropriada às ações do PID.
- **3. Alimentação e Eficiência Energética:** Gerenciar o uso de duas baterias de 9V para alimentar o robô e garantir eficiência energética é um desafio crítico. É necessário projetar circuitos para otimizar a energia utilizada.
- **4. Estabilidade do Equilíbrio:** A parte mais significativa do desafio é a implementação do sistema PID em si para manter o equilíbrio do robô. Calibrar corretamente os parâmetros PID para evitar oscilações e quedas requer experimentação e ajustes precisos.

# 7. Desafios

- **5. Feedback de Velocidade:** Integrar o Sensor de Velocidade FC-03 para medir a velocidade do robô e fornecer feedback para o sistema PID é um desafio adicional que envolve a interpretação dos dados do sensor e sua incorporação no controle.
- **6. Programação e Lógica Complexa:** Implementar a lógica do controle PID e garantir que todos os componentes e sensores se comuniquem efetivamente requer habilidades de programação avançadas e resolução de problemas.
- **7. Resposta a Distúrbios e Condições Variáveis:** Projetar o sistema PID para responder adequadamente a distúrbios externos e condições variáveis, como diferentes superfícies de terreno, é um desafio importante.
- **8. Design Mecânico e Distribuição de Peso:** Garantir que o robô segway seja fisicamente equilibrado e bem construído, com o peso distribuído de maneira adequada, também é essencial para o sucesso do projeto.

# 8. Simulação Simulink/Matlab



# 8.1 Desafios Simulação

- 1. Modelagem do Robô Segway:** Incluir todas as suas características físicas e dinâmicas.
- 2. Sintonia do Controlador PID:** A sintonia adequada dos parâmetros do controlador PID é um desafio crítico para garantir o equilíbrio estável do robô. Isso exige experimentação e análise.
- 4. Lidar com Não-Linearidades:** Os robôs segways geralmente apresentam não-linearidades em sua dinâmica, como atrito e aceleração variável.
- 5. Feedback de Velocidade e Aceleração:** Integrar feedback de sensores de velocidade e aceleração, se necessário, requer modelagem e manipulação adequada desses dados no Simulink.
- 6. Simulação em Ambientes Realistas:** Levar em consideração distúrbios, como vento ou inclinações variáveis, é um desafio importante.
- 7. Avaliação de Estabilidade e Desempenho:** Medir e avaliar a estabilidade do robô segway, envolve a análise de dados de simulação para ajustes finos.
- 8. Comportamento em Tempo Real:** Assegurar que o sistema PID funcione em tempo real na simulação do Simulink é um desafio adicional, exigindo ajustes e otimização do código.

# Referências

The screenshot shows the GitHub interface for the repository 'SC3\_Final' by user 'gabereboucas'. The repository is private and has 12 commits. The file list includes 'Codigos Arduino', 'Modelo - LQR Simulação Tempo Con...', 'Modelo 1-Simulacao AutoEquilibrio', 'Modelo 3 - PID', 'Referências', '1.png', 'README.md', 'Self\_balance\_bb.png', and 'Simulacao AutoEq.zip'. The 'Referências' file is selected, showing a preview of a document titled 'Documento de Planejamento de Trabalho de Sistema e Controle 3'. The document preview shows a MATLAB/Simulink interface with a block diagram of a control system. The right sidebar shows repository statistics: 1 star, 1 watching, 0 forks. It also lists 'Releases' (no releases published), 'Packages' (no packages published), and 'Languages' (MATLAB 59.4%, C++ 40.6%). Suggested workflows for 'MSBuild based projects' and 'C/C++ with Make' are also visible.

Disponível em: [https://github.com/gabereboucas/SC3\\_Final](https://github.com/gabereboucas/SC3_Final)

# Referências

ANDRADE JÚNIOR, Braz da Costa et al. **Modelagem e análise do sistema de suspensão dianteira de uma motocicleta ao frear**. 2021.

ESHKABILOV, Sulaymon L.; ESHKABILOV, Sulaymon L. Spring-Mass-Damper Systems. **Practical MATLAB Modeling with Simulink: Programming and Simulating Ordinary and Partial Differential Equations**, p. 295-344, 2020.

KHOT, S. M.; YELVE, Nitesh P. Modeling and response analysis of dynamic systems by using ANSYS© and MATLAB©. **Journal of Vibration and Control**, v. 17, n. 6, p. 953-958, 2011.

LAKHLANI, Bhargavkumar; YADAV, Himansh. Development and analysis of an experimental setup of spring–mass–damper system. **Procedia Engineering**, v. 173, p. 1808-1815, 2017.

OGATA, K. et al. **Modern control engineering**. Upper Saddle River, NJ: Prentice hall, 2010.

ROZLAN, Syaiful Azmirul Mohd et al. Theoretical modelling of a beam with attached spring-mass-damper system. In: **MATEC Web of Conferences**. EDP Sciences, 2017. p. 01030.

SHARMA, Rajkumar; PATHAK, D. K.; DWIVEDI, V. K. Modeling & simulation of spring mass damper system in simulink environment. **Proceedings of SOM**, v. 2014, 2014.



**Obrigado  
pela atenção!**