FUNDAÇÃO UNIVERSIDADE FEDERAL DO TOCANTINS
CAMPUS UNIVERSITÁRIO DE PALMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

GABRIEL CINTRA

MODELAGEM E CONTROLE DE UM PÊNDULO INVERTIDO SOBRE DUAS RODAS

Palmas / TO
2021

GABRIEL CINTRA

MODELAGEM E CONTROLE DE UM PÊNDULO INVERTIDO SOBRE DUAS RODAS

Monografia foi avaliada e apresentada à UFT – Universidade Federal do Tocantins – Campus Universitário de Palmas, Curso de Engenharia Elétrica para obtenção do título de Bacharel em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: Prof. Me. Alex Vilarindo Menezes

Palmas / TO
2021

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da Universidade Federal do Tocantins

C575m Cintra, Gabriel.

Modelagem e Controle de um Pêndulo Invertido Sobre Duas Rodas. / Gabriel Cintra. – Palmas, TO, 2021.

103 f.

Monografia Graduação - Universidade Federal do Tocantins – Câmpus Universitário de Palmas - Curso de Engenharia Elétrica, 2021.

Orientador: Alex Vilarindo Menezes

1. Pêndulo Invertido. 2. Sistemas de Controle. 3. Controlador PID. 4. Modelagem Matemática. I. Título

CDD 621.3

GABRIEL CINTRA

MODELAGEM E CONTROLE DE UM PÊNDULO INVERTIDO SOBRE DUAS RODAS

Monografia foi avaliada e apresentada à UFT – Universidade Federal do Tocantins – Campus Universitário de Palmas, Curso de Engenharia Elétrica para obtenção do título de Bacharel em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Data de aprovação: 16 / 04 / 2021

Banca Examinadora:

Alex Vilarindo Menezes

Prof. Me. Alex Vilarindo Menezes, UFT

Priscila da S. Oliveira

Profª. Drª. Priscila da Silva Oliveira, UFT

Prof. Dr. Orlando Fonseca Silva, UFPA

Palmas / TO

2021

RESUMO

O pêndulo invertido é um sistema naturalmente instável, com várias aplicações, muito utilizado em lançamento de foguetes, carros de duas rodas, entre outras. Devido sua natureza instável é muito utilizado no estudo de Teoria de Controle para medir o desempenho de vários sistemas de controle. Este trabalho aborda a modelagem, implementação e controle de um sistema pêndulo invertido sobre duas rodas utilizando um controlador PID com o objetivo de manter o pêndulo em equilíbrio na posição vertical. Para manter o pêndulo em equilíbrio é utilizado um motor em cada roda para compensar as forças que o fazem instável, controlados por um microcontrolador ATMEGA32u4 utilizando a plataforma Arduino e um sensor MPU6050 que é composto por um giroscópio e acelerômetro que faz a leitura do ângulo de inclinação do pêndulo.

Palavras-chaves: Pêndulo Invertido, Modelagem, Controle, PID.

ABSTRACT

The inverted pendulum is a naturally unstable system, with several applications, widely used in launching rockets, two-wheel cars, among others. Due to its unstable nature it is widely used in the study of Control Theory to measure the performance of various control systems. This work addresses the modeling, implementation and control of an inverted pendulum system on two wheels using a PID controller in order to keep the pendulum in balance in the vertical position. To keep the pendulum in balance, a motor is used on each wheel to compensate for the forces that make it unstable, controlled by an ATMEGA32u4 microcontroller using the Arduino platform and an MPU6050 sensor that is composed of a gyroscope and accelerometer that reads the inclination angle of the pendulum.

Key-words: Inverted Pendulum, Modeling, Control, PID.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CAD	Computer Aided Design (Desenho Assistido por Computador)
MATLAB	Matrix Laboratory
PIDR	Pêndulo Invertido sobre Duas Rodas
PID	Controlador Proporcional Integral Derivativo
LQR	Regulador Linear Quadrático
PWM	Pulse Width Modulation (Modulação por Largura de Pulso)
Motor CC	Motor Corrente Contínua
RPM	Rotações Por Minuto
PPR	Pulsos Por Revolução
DIR	Sentido de Rotação do Motor de Passo
STEP	Passo do Motor de Passo

LISTA DE SÍMBOLOS

K_p	Constante proporcional
K_i	Constante integral
K_d	Constante derivativa
V_a	Tensão aplicada no motor CC (V)
R_a	Resistência de armadura do motor CC (**Ω**)
L_a	Indutância de armadura do motor (H)
I_a	Corrente de Armadura (A)
V_n	Tensão nominal do motor CC (V)
V_e	Tensão da força contra eletromotriz do motor CC (V)
k_e	Constante da força contra eletromotriz (Vs/rad)
k_t	Constante de torque do motor CC (Nm/A)
ω	Velocidade angular no eixo do motor (rad/s)
N	Velocidade rotação do motor (RPM)
n	Relação de velocidade da caixa de redução
T_m	Torque do motor CC (Nm)
m_r	Massa do conjunto rodas (kg)
R	Raio da roda (m)
H_{fD}	Força de fricção entre a roda direita e a superfície (N)
H_{fE}	Força de fricção entre a roda esquerda e a superfície (N)
H_D	Componente horizontal da força de reação da roda direita (N)
H_E	Componente horizontal da força de reação da roda esquerda (N)
P_D	Componente vertical da força de reação da roda direita (N)
P_E	Componente vertical da força de reação da roda esquerda (N)
C_D	Torque aplicado pelo motor à roda direita (Nm)
C_E	Torque aplicado pelo motor à roda esquerda (Nm)
I_r	Momento de inércia da roda (kgm²)
θ_r	Ângulo entre o eixo y e a roda (°)
θ_r'	Velocidade angular da roda (°/s)
θ_r''	Aceleração angular da roda (°/s²)
m_p	Massa do conjunto pêndulo (kg)
J_p	Momento de inércia do pêndulo (kgm²)

l	Distância do centro de massa (pêndulo) até o centro das rodas (m)
g	Aceleração da gravidade (m/s²)
x	Distância x para o sistema (m)
x'	Velocidade linear (m/s)
x''	Aceleração linear (m/s²)
L	Largura do pêndulo (m)
H	Altura do pêndulo (m)
θ_p	Ângulo entre o eixo y e o pêndulo (°)
θ_p'	Velocidade angular do pêndulo (°/s)
θ_p''	Aceleração angular do pêndulo (°/s²)
θ_{passo}	Ângulo de passo do motor de passo (°)
f_{con}	Frequência de controle do motor de passo

SUMÁRIO

1 INTRODUÇÃO

O sistema pêndulo invertido é utilizado em várias áreas de controle e automação, é o princípio do controle de atitude de foguetes, onde o objetivo é se manter na posição vertical. O pêndulo é um sistema instável, por só ter um ponto de apoio em sua base, ele pode cair para qualquer direção, a menos que uma força de controle seja aplicada a ele (OGATA, 2010).

Alves (2018) faz uma analogia do controle do pêndulo invertido ao equilibrar um cabo de vassoura na ponta dos dedos. Para manter a vassoura em equilíbrio deve-se ficar a todo o momento movendo a mão a fim de compensar as variações do ângulo de inclinação de modo a manter o cabo de vassoura em equilíbrio vertical.

O sistema pêndulo invertido, como já mencionado, é um sistema instável. Este fato pode ser confirmado utilizando o gráfico do lugar das raízes. O gráfico do lugar das raízes representa a posição dos polos e zeros da função de transferência de um sistema e a posição dos polos pode ser utilizada para determinar sua estabilidade. Segundo Nise (2013), um polo localizado do semiplano direito do plano s, caracteriza um sistema instável.

Segundo Paula (2014), existem vários modelos de pêndulos invertidos, alguns utilizando um pêndulo sobre um carro, um pêndulo sobre duas rodas, pêndulo sobre trilhos, porém as modelagens matemáticas do sistema são todas bem próximas.

O sistema pêndulo invertido é um sistema não linear (TEIXEIRA, 2006), então para facilitar o controle, algumas aproximações e simplificações por meio de linearizações serão efetuadas. Uma destas simplificações será a restrição do ângulo de inclinação do pêndulo, ao reduzir o deslocamento angular do pêndulo pode-se tratar o pêndulo como um sistema linear, tornando o controle mais simples (PAULA, 2014).

1.1 Justificativa

O sistema pêndulo invertido é um sistema naturalmente instável que pode ser comparado a alguns problemas da engenharia como o controle de atitude de um foguete, meios de transportes de duas rodas e até mesmo a postura de uma pessoa. O sistema pêndulo invertido é uma referência no estudo da Teoria de Controle por ter uma dinâmica complexa e ser um sistema não linear, é utilizado para medir a eficiência de várias técnicas de controle (GADELHA, 2018).

O estudo deste sistema engloba várias áreas da engenharia como mecânica, eletrônica, programação e controle. Portanto o controle desse sistema instável é ideal para aplicar todos os conceitos obtidos ao longo da graduação.

1.2 Objetivo geral

O objetivo geral deste trabalho é desenvolver e implementar um sistema do tipo pêndulo invertido sobre duas rodas, que se mantenha em equilíbrio vertical e que suporte pequenas perturbações.

1.3 Objetivos específicos

1. Estudar os modelos de pêndulos invertido;
2. Modelagem matemática das forças exercidas sobre o pêndulo;
3. Desenvolver um sistema de controle para manter o pêndulo em equilíbrio;
4. Simulação do sistema em malha fechada com o controle projetado;
5. Programação do controle em linguagem de alto nível, C++;
6. Manter o pêndulo em equilíbrio e ser capaz de suportar pequenos distúrbios.

1.4 Metodologia

Primeiramente será feito um levantamento bibliográfico sobre o tema.

Após um bom embasamento teórico, a próxima etapa será à modelagem matemática do sistema do pêndulo invertido.

Em seguida será desenvolvido um sistema de controle utilizando a modelagem já desenvolvida.

Com auxilio do MATLAB e Simulink serão feitas simulações do sistema de controle projetado.

Depois de elaborar o sistema de controle, este deve ser programado em um microcontrolador, para controlar o pêndulo.

Após a programação do controlador serão feitos ajustes para melhorar o desempenho do sistema;

Para finalizar, serão realizados testes a fim de cumprir o objetivo do projeto que é manter o pêndulo na posição vertical, e ainda ser capaz de suportar pequenos distúrbios.

2 REVISÃO BIBLIOGRÁFICA

O sistema pêndulo invertido é um modelo muito estudado na área da Teoria de Controle e, portanto existem vários trabalhos sobre este tema. A seguir serão mostrados alguns destes trabalhos.

2.1 *Wheelie* Plataforma Auto balanceada em duas rodas (CARVALHO, 2014).

Neste projeto o autor utiliza uma plataforma chamada *Wheelie*, que é uma plataforma com duas rodas, onde uma pessoa fica sobre a plataforma e controla seus movimentos. O autor faz melhorias no sistema original do *Wheelie*, integrando micro-giroscópios e acelerômetros para observar mudanças no terreno e a posição do corpo do condutor e também o desenvolvimento de um sistema de navegação semi-autônoma, utilizando sensores de obstáculos. A Figura 1 mostra o resultado de Carvalho (2014), o sistema em equilíbrio transportando uma pessoa.

Figura 1 - Plataforma auto balanceada Wheelie

Fonte: (CARVALHO, 2014)

2.2 Análise Comparativa de Controladores Aplicados a Sistemas de Auto Equilíbrio (FRAGA et al., 2014).

Neste projeto os autores fazem a comparação de dois tipos de controladores no sistema do pêndulo invertido sobre duas rodas comparando a eficiência entre os dois tipos de controles. Foi feita a comparação entre controladores PID e LQR.

O Regulador Linear Quadrático (LQR), do inglês *Linear Quadratic Regulator*, é um controlador ótimo que vêm sendo amplamente difundido na indústria. A estratégia deste controlador é a realimentação dos estados, que são ponderados de forma a minimizar uma função custo. Nesta estratégia de controle os ganhos associados aos estados da lei de controle são obtidos através da solução de uma equação algébrica de Ricatti, relacionada ao problema.

Os autores dizem que ambos os controladores estabilizam o sistema, entretanto o controlador PID teve uma resposta mais rápida que o controlador LQR, porém a o controlador PID apresenta uma pequena oscilação na saída em contraste do controlador LQR para o qual a saída ficou praticamente sem variação. A Figura 2 mostra o sistema que FRAGA et al. (2014) utilizou para fazer a comparação dos dois tipos de controle.

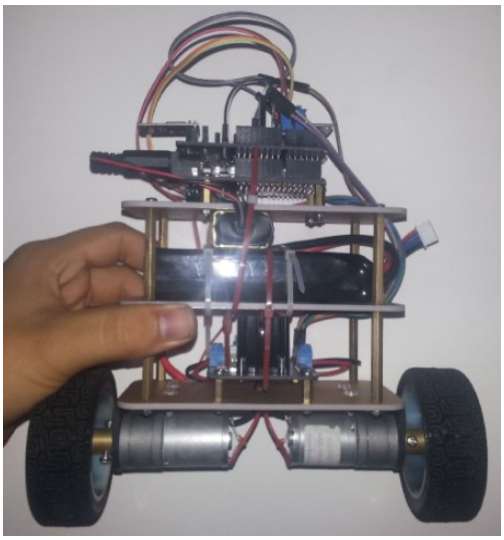
Figura 2 - Protótipo do pêndulo para fazer comparação dos controladores

Fonte: (FRAGA et al., 2014).

2.3 Projeto de um controlador neuro-fuzzy para um robô de equilíbrio (TREVISANI, 2015)

O autor deste projeto utiliza controladores neuro-fuzzy para estabilizar o pêndulo, ele menciona que os controladores neuro-fuuzy utilizam técnicas de redes neurais artificiais, também chamados de controladores inteligentes, pois sua operação é análoga ao pensamento humano. A vantagem deste tipo de controlador é que a modelagem física do sistema não é estritamente necessária, ao contrário dos controladores PID onde a modelagem da planta é fundamental. Na Figura 3 é possível observar o sistema montado por Trevisani (2015).

Figura 3 - Robô montado



Fonte : (TREVISANI, 2015)

2.4 *Segway i2 SE Personal Transporter* (SEGWAY, 2021)

Quando se fala em pêndulo invertido sobre duas rodas a primeira coisa que vem a cabeça são os veículos de transporte da *Segway*. O transportador funciona de maneira semelhante ao corpo humano, utilizando uma tecnologia de estabilização dinâmica para manter o equilíbrio enquanto anda para frente ou para trás. Ao andar o corpo se inclina para frente, dando um passo à frente para manter o equilíbrio. Ao inclinar para trás, da um passo para trás. No transportador, ao inclinar para frente, as rodas são acionadas na direção em que se inclina para manter o equilíbrio e fazendo-o andar. A Figura 4 mostra o transportador *Segway i2 SE*.

Figura 4 - *Segway i2 SE Personal Transporter*

Fonte : (SEGWAY, 2021)

3 FUNDAMENTAÇÃO TEÓRICA

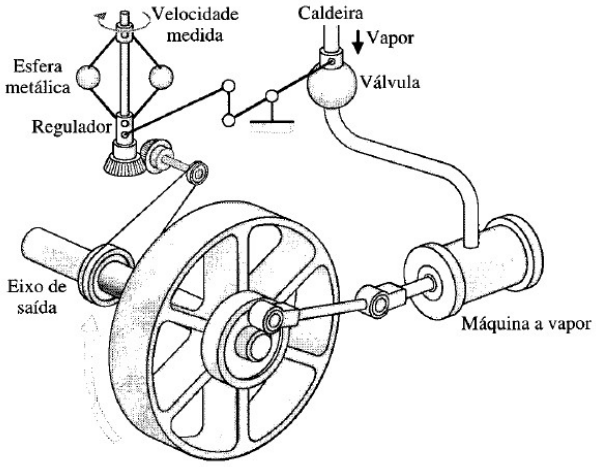
Este capítulo fornece os aspectos teóricos sobre os principais assuntos que serão utilizados para o projeto de um pêndulo invertido sobre duas rodas.

3.1 Histórico da Teoria de Controle

A primeira aplicação de controle com retroação foi o mecanismo regular de bóia na Grécia no período entre 300 a.C. O relógio d`água de Ktesibios usava um regulador de bóia. Um lampião de óleo inventado por Philton em 250 a.C. usava um regulador de bóia para manter o nível de óleo constante (DORF; BISHOP, 2001).

O primeiro controlador automático com retroação usado em um processo industrial é o regulador de esferas de James Watt, desenvolvido em 1769 para controlar a velocidade de máquinas a vapor. O dispositivo, totalmente mecânico, mostrado na Figura 5, media a velocidade do eixo de saída e utilizava a inércia das esferas para controlar a abertura da válvula de vapor (DORF; BISHOP, 2001).

Figura 5 - Regulador de esferas de James Watt



Fonte: (DORF; BISHOP, 2001).

Até o ano de 1868 os sistemas de controle eram desenvolvidos através de intuição e invenção. Os esforços para melhorar os sistemas de controle conduziam para menor atuação das oscilações e até tornavam os sistemas instáveis. Então James Clerk Maxwell formulou uma teoria matemática de controle usando equações diferenciais (DORF; BISHOP, 2001).

A teoria e a prática do controle automático receberam um grande degrau durante a Segunda Guerra Mundial quando surgiu a necessidade de projetar pilotos automáticos para

aviões, sistemas de posicionamento de canhões, sistema de posicionamento de antenas para radares, entre outros sistemas para uso militar (DORF; BISHOP, 2001).

Antes de 1940, os sistemas de controle em sua maior parte, era uma arte que envolvia abordagem de ensaio e erro. Durante 1940, surgiram métodos matemáticos e analíticos que garantiram a consolidação de uma área específica para sistemas de controle (DORF; BISHOP, 2001).

Após a Segunda Guerra Mundial com o aumento do uso da Transformada de Laplace e o plano de frequência complexa, as técnicas no domínio da frequência dominaram o campo de controle. Durante os anos 1950, a ênfase na área de controle foi o desenvolvimento e o uso de métodos no plano s e, particularmente, a abordagem do lugar das raízes. Em 1980, a utilização de computadores digitais como componentes de controle tornou possível a execução de cálculos rápidos e precisos, anteriormente impossíveis (DORF; BISHOP, 2001).

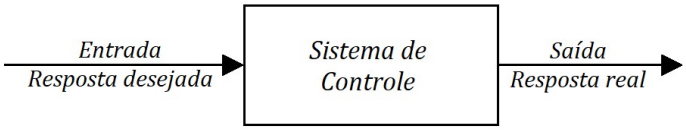
Com o advento da era espacial, novos estímulos foram dados à engenharia de controle. Os sistemas se tornaram muito mais complexos e a precisão era imprescindível para controle de lançamentos de mísseis, espaçonaves e sondas espaciais. Os métodos de controle no domínio do tempo desenvolvidos por Liapunov, Minorsky e outros, tem sido objeto de grande interesse. Torna-se evidente que a engenharia de controle deve considerar o domínio do tempo e o domínio da frequência simultaneamente na análise e projeto de sistemas de controle (DORF; BISHOP, 2001).

3.2 Controle

Sistemas de controle são muito importantes para a sociedade moderna. Em muitas aplicações se não for aplicado algum tipo de controle o sistema não terá o funcionamento desejado e em alguns casos o colapso do sistema.

Um sistema de controle é composto por um processo com o objetivo de se obter uma saída desejada com um desempenho desejado, dada uma entrada especificada (NISE, 2013). A Figura 6 mostra o modelo de um sistema de controle simplificado, onde é representado uma entrada, o sistema de controle e a saída desejada.

Figura 6 - Modelo simplificado de um sistema de controle



Fonte: Adaptado de Nise (2013)

Segundo Nise (2013), a construção de sistemas de controle é dada principalmente por quatro razões:

1. Amplificação de potência;
2. Controle remoto;
3. Conveniência da forma de entrada;
4. Compensação de perturbações.

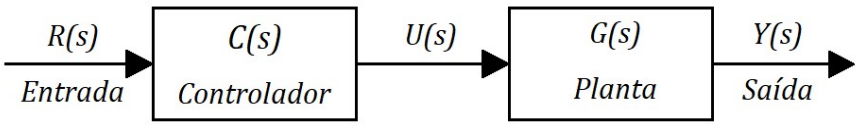
Como exemplo do controle de uma antena, com o ajuste de um pequeno botão é possível controlar a posição da antena que requer uma grande quantidade de potência, e ainda se manter na posição desejada mesmo com perturbações externas, como o vento. (NISE, 2013).

Os sistemas de controle podem ser divididos em duas configurações, controle em malha aberta e malha fechada.

3.2.1 Sistema em malha aberta

Os sistemas de controle em malha aberta, Figura 7, são aqueles em que o sinal de saída não exerce nenhuma ação de controle no sistema. Portanto o sinal de saída não é medido e nem comparado com a entrada (OGATA, 2010).

Figura 7 - Sistema em malha aberta



Fonte: Próprio autor

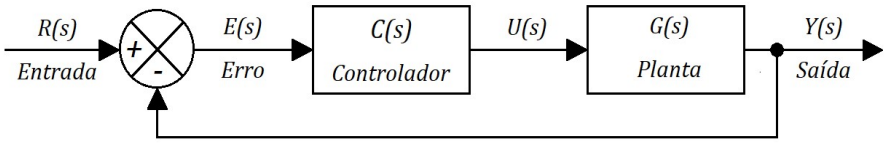
Nos sistemas em malha aberta, a saída não tem realimentação na entrada, assim a entrada é ajustada para uma condição fixa de operação. Então se ocorrer qualquer perturbação o sistema não tem nenhuma forma de correção, ou seja, o sistema não vai executar a tarefa desejada. A Equação (1) mostra a função de transferência de um sistema em malha aberta (OGATA, 2010).

$$\frac{Y(s)}{R(s)} = C(s)G(s) \tag{1}$$

3.2.2 Sistema em malha fechada

Os sistemas de controle em malha fechada é um sistema em malha aberta com uma realimentação do sinal da saída, mostrado na Figura 8. Em um sistema de malha fechada, o sinal de erro, a diferença entre o sinal de entrada e o sinal de saída, realimenta o controlador, minimizando o erro e acertando a saída do sistema ao valor desejado (OGATA, 2010).

Figura 8 - Sistema em malha fechada



Fonte: Próprio autor

Uma das vantagens do sistema em malha fechada é que a realimentação faz com que a resposta do sistema seja menos suscetível a distúrbios externos e variações internas nos parâmetros do sistema. Dessa forma pode-se utilizar componentes relativamente imprecisos e em decorrência disso, mais baratos para obter um controle de determinado sistema, ao passo que isso é impossível num sistema em malha aberta. A Equação (2) mostra a função de transferência de um sistema em malha fechada (OGATA, 2010).

$$\frac{Y(s)}{R(s)} = \frac{C(s)G(s)}{1 + C(s)G(s)} \tag{2}$$

3.3 Controlador PID

O Controlador PID, é a sigla para Controlador Proporcional, Integral e Derivativo, muito utilizado nos controles industriais. Segundo Ogata (2010), mais da metade dos controladores em uso atualmente utilizam o controle tipo PID. A função de transferência do controlador PID é dada pela Equação (3).

$$G_c(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_p s + K_i + K_d s^2}{s} \tag{3}$$

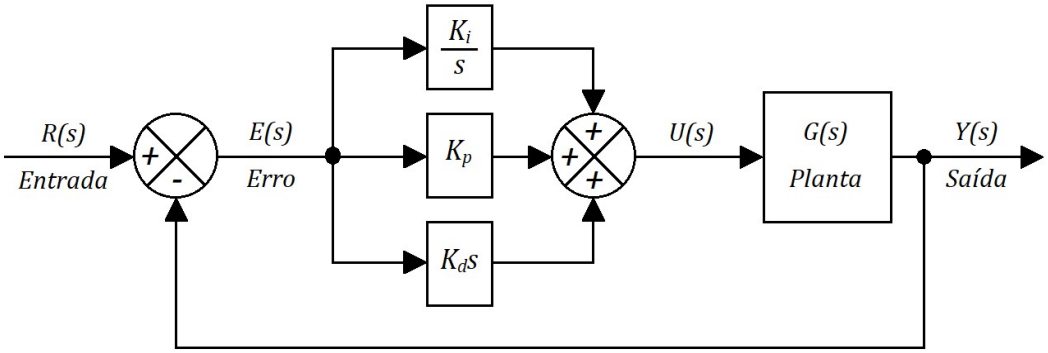
Observando a função de transferência do controlador PID, ele insere um polo na origem e dois zeros que podem ser posicionados em qualquer lugar do semiplano esquerdo do plano s, lembrando que o lugar das raízes começa nos polos e termina nos zeros (DORF; BISHOP, 2001).

O controlador oferece um termo proporcional, um termo de integração e um termo derivativo, a equação para a saída no domínio do tempo é mostrada na Equação (4).

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{d}{dt} e(t) \tag{4}$$

A popularidade dos controladores PID pode ser atribuída parcialmente pelo seu bom desempenho em vastas condições de operação e parcialmente à sua simplicidade funcional que permite sua operação de modo simples e direto, o diagrama de blocos do controlador PID é mostrado na Figura 9. Para implementar um controlador PID, é necessário determinar as constantes do ganho proporcional K_p, o ganho integral K_i e o ganho derivativo K_d (DORF; BISHOP, 2001).

Figura 9 - Controlador PID



Fonte: Adaptado de Nise (2013)

O funcionamento do controlador PID pode descrito como mostrado na Figura 9. A variável E representa o erro, a diferença entre a saída Ye a referência R. Este sinal de erro E é a entrada do controlador PID, o controlador calcula a derivada e a integral desse erro em relação ao tempo. O sinal de controle U da planta é igual a soma do erro multiplicado pela constante proporcional K_p, a integral do erro vezes o ganho integral K_i e o ganho derivativo K_d vezes a derivada do erro. Este sinal de controle U é à entrada da planta e a nova saída Y obtida é então realimentada e comparada com a referência encontrando um novo valor de erro. Este processo continua enquanto o controlador estiver em funcionamento (CTMS, 2020).

3.3.1 Proporcional

O ganho proporcional K_p produz um valor na saída proporcional ao erro obtido. Aumentar o ganho proporcional aumenta o sinal de controle fazendo o sistema reagir mais rapidamente, mas aumenta o sobre-sinal. Outro efeito do aumento de K_p é que ele reduz o erro, mas não anula completamente (CTMS, 2020).

3.3.2 Integral

A função integral soma todos os erros instantâneos e a somatória é multiplicada pela constante K_i. Se houver um erro persistente e constante o integrador soma estes erros, aumentando o sinal de controle reduzindo o erro até que o erro seja eliminado. Um problema é que o termo integral torna o sistema mais lento, outro problema é que com a soma dos erros o sinal de controle pode aumentar muito fazendo ocorrer um sobre-sinal (CTMS, 2020).

3.3.3 Derivativo

A função derivativa traz um caráter antecipativo ao controlador em relação a tendência futura do erro. A única maneira do sinal de controle aumentar é se o erro aumentar, com o controle derivativo o sinal de controle pode aumentar rapidamente se o erro ainda for relativamente pequeno. Essa antecipação tende a adicionar amortecimento ao sistema, diminuindo o sobre-sinal. Entretanto o termo derivativo não tem efeito sobre o erro em estado estacionário (CTMS, 2020).

Os efeitos gerais de cada parâmetro do controlador K_p, K_d e K_i num sistema podem ser resumidos na Tabela 1. Essas diretrizes são válidas para muitos casos, mas não em todos. Esta informação é útil caso se queira determinar os valores dos ganhos por tentativa e erro (CARVALHO, 2014).

Tabela 1 - Efeito da variação das constantes de um controlador PID

Constante	Tempo de subida	Sobre-sinal	Tempo de assentamento	Erro
K_p	Diminui	Aumenta	Pouca alteração	Diminui
K_i	Diminui	Aumenta	Aumenta	Elimina
K_d	Pouca alteração	Diminui	Diminui	Não afeta

Fonte: (CTMS, 2020).

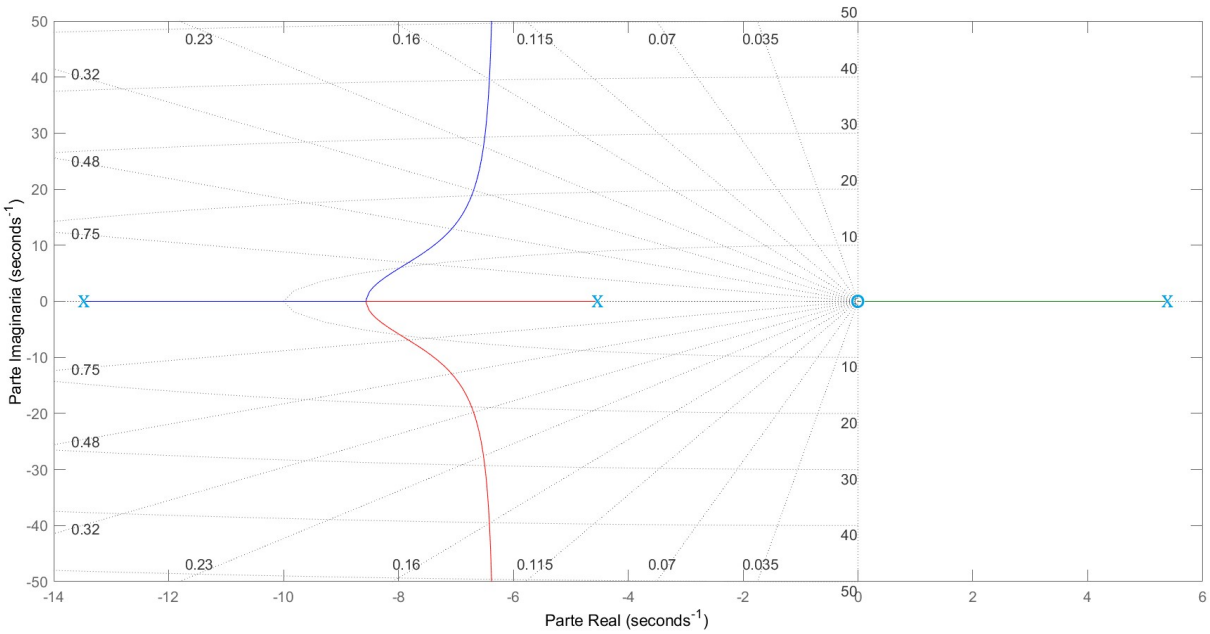
3.4 O Lugar das Raízes

A estabilidade relativa e o desempenho de um sistema de controle em malha fechada estão diretamente relacionados com a localização das raízes da equação característica em malha fechada no plano s. Para obter os resultados desejados no sistema deve-se fazer alguns ajustes nos parâmetros de modo a obter o posicionamento adequado das raízes (DORF; BISHOP, 2001).

Em alguns sistemas, o simples ajuste do ganho pode mover os polos de malha fechada para as localizações desejadas. Então, o problema é resolvido apenas escolhendo um valor de ganho apropriado. Se apenas o ajuste do ganho não produzir o resultado desejado, será necessário adicionar um compensador ao sistema (OGATA, 2010).

A Figura 10, representa um exemplo de gráfico do lugar das raízes de um sistema pêndulo invertido. Ao analisar o gráfico observa-se a presença de um polo no semiplano da direita, ou seja, são polos que produzem respostas naturais exponencialmente crescentes puras ou senoides exponencialmente crescentes. Essas respostas naturais tendem ao infinito à medida que o tempo tende ao infinito. Portanto se os polos em malha fechada estiverem na metade direita do plano s e consequentemente na parte real positiva, o sistema será instável (NISE, 2013).

Figura 10 - Exemplo de gráfico do lugar das raízes



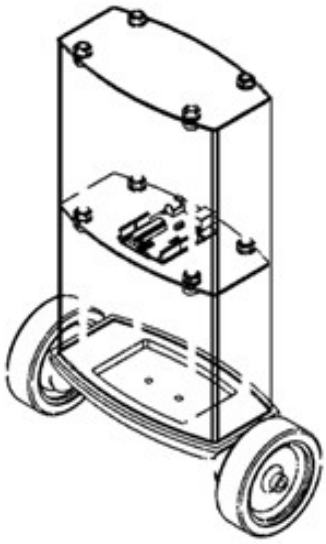
Fonte: Próprio autor

4 PÊNDULO INVERTIDO

O pêndulo invertido é composto basicamente de uma haste presa a um carrinho, onde a haste tem movimento livre em uma direção, deste modo o ângulo entre a haste em relação à superfície normal do carrinho é facilmente encontrado. A modelagem deste sistema pode ser encontrada em (OGATA, 2010).

Entretanto neste trabalho será utilizada uma variação do modelo tradicional do pêndulo, conhecido como Pêndulo Invertido sobre Duas rodas (PIDR), como pode ser visto na Figura 11. Neste modelo a haste é fixada em duas rodas paralelas que farão os ajustes das forças para manter o equilíbrio. A referência para estabilização é o vetor de aceleração da gravidade, assim pode-se corrigir o ângulo entre a haste e à gravidade fazendo com que o pêndulo permaneça na posição vertical (PAULA, 2014).

Figura 11 - Esquema do pêndulo invertido sobre duas rodas



Fonte: Adaptado de Hanna e Henrik (2015)

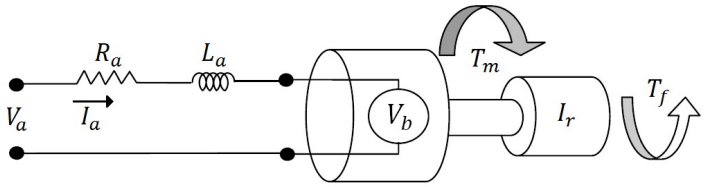
Para simular o sistema em MATLAB / Simulink, um modelo matemático que descreva a dinâmica do sistema deve ser desenvolvido.

Para facilitar a modelagem matemática o sistema será dividido e analisado em partes separadas. A estrutura do PIDR será dividida em motores CC, rodas, e haste. Este procedimento é utilizado por Ooi (2003) e por Kankhunthod et. al. (2019).

4.1 Modelo do motor CC

O circuito simplificado de um motor CC é mostrado na Figura 12. As bobinas do rotor são representadas pela conexão em série de R_a e L_a. Quando uma tensão V_a é aplicada aos terminais do motor, uma corrente I_a flui nas bobinas. O rotor torna-se um eletroímã e reage com os ímãs permanentes no estator fazendo-o girar. O torque gerado pelo motor é proporcional a intensidade do campo magnético do rotor, que por sua vez depende da corrente I_a. O torque produzido pelo motor é ampliado pela caixa de redução e apresentado no eixo de saída. Os parâmetros k_t e k_b são coeficientes que dependem da construção dos motores CC.

Figura 12 - Circuito elétrico do motor CC



Fonte: Adaptada de Ooi (2003)

O torque do eixo do motor, T_m, precisa superar a inércia, o amortecimento viscoso e o atrito do motor. Como são valores difíceis de obter, de magnitudes tão pequenas, e com pequena relevância para a modelagem, elas foram desprezadas (SUNDIN; THORSTENSSON, 2012). A Equação (5), descreve o torque no eixo do motor.

$$T_m = k_t I_a \tag{5}$$

Onde k_t é a constate de torque e I_a é a corrente de armadura.

Quando o motor está girando a bobina do rotor gira no campo magnético criado pelos ímãs permanentes no estator, uma força contra eletromotriz é gerada na bobina em virtude da Segunda Lei de Faraday, a intensidade dessa força depende da velocidade em que o rotor está girando em relação ao estator, a força contra eletromotriz V_b é mostrado na Equação (6).

$$V_b = k_b \theta'_m \tag{6}$$

Aplicando a lei de Kirchoff no circuito do motor CC da Figura 12 resulta em:

$$V_a = R_a I_a + L_a I'_a + k_b \theta'_m \tag{7}$$

Onde R_a é a resistência de armadura, I_a é a corrente de armadura, L_a é a indutância de armadura do motor, k_b é constante da força contra eletromotriz, θ'_m é a velocidade angular no eixo do motor.

De acordo com Sundin e Thorstensson (2012), a indutância do motor é desprezível, pois a constante de tempo elétrica do motor é bem menor que a constante de tempo mecânica, então reescrevendo a Equação (7), desprezando a indutância do motor tem-se a Equação (8).

$$V_a = R_a I_a + k_b \theta'_m \tag{8}$$

Utilizando as Equações (5) e (8), a expressão que descreve o torque para cada motor é dada pela Equação (9).

$$T_m = \frac{k_t}{R_a} V_a - \frac{k_t k_b}{R_a} \theta'_m \tag{9}$$

Como há uma caixa de engrenagens entre o motor e o eixo de saída, com relação de engrenagem N_g, o torque entre o eixo de saída e o motor é dada pela Equação (10):

$$T_r = N_g T_m \tag{10}$$

Usando a relação descrita pela Equação (10), a expressão do torque no eixo de saída em função da tensão aplicada e velocidade angular do motor é mostrada na Equação (11):

$$T_r = \frac{N_g k_t}{R_a} V_a - \frac{N_g k_t k_b}{R_a} \theta'_m \tag{11}$$

Para encontrar as constantes k_t e k_b serão feitas aproximações mostradas nas Equações (12) e (14). Em motores de médio a grande porte, ou em motores mais caros, estes valores podem ser consultados nas folhas de dados, porém em motores pequenos e mais baratos estes dados não estão disponíveis (SUNDIN; THORSTENSSON, 2012). A constante de torque é calculada utilizando a Equação (12), com os parâmetros de rotor bloqueado.

$$k_t = \frac{T_{r_{bloqueado}}}{N_g I_{a_{bloqueado}}} \tag{12}$$

Onde $I_{a_{bloqueado}}$ é a corrente de armadura com rotor bloqueado e $T_{r_{bloqueado}}$ o torque do motor com rotor bloqueado e N_g é a relação da caixa de redução.

A velocidade do motor geralmente é dada em RPM, dessa forma a Equação (6) deve ser reescrita como mostrada na Equação (13):

$$V_b = \frac{N60}{2\pi} k_b \tag{13}$$

A constante da força contra eletromotriz é calculada utilizando a Equação (14), com os parâmetros de rotor a vazio.

$$k_b = \frac{\left(V_a - R_a I_{a_{vazio}}\right)60}{2\pi N_g N} \tag{14}$$

Onde, V_a é a tensão de armadura, R_a é a resistência de armadura, $I_{a_{vazio}}$ é a corrente de armadura com motor sem carga, N_g é a relação da caixa de redução e N é a velocidade em RPM do motor a vazio.

Como o moto-redutor está conectado à roda, a velocidade angular da roda pode ser relacionada à velocidade angular do motor. Se não há escorregamento entre à roda e a superfície, a velocidade linear do pêndulo é relacionado com a velocidade das rodas, como mostrado na Equação (15).

$$x' = \theta_r' R = N_g \theta_m' R \tag{15}$$

Reorganizando a Equação (15), a velocidade angular do motor em função do deslocamento é dada pela Equação (16).

$$\theta_m' = \frac{x'}{R N_g} \tag{16}$$

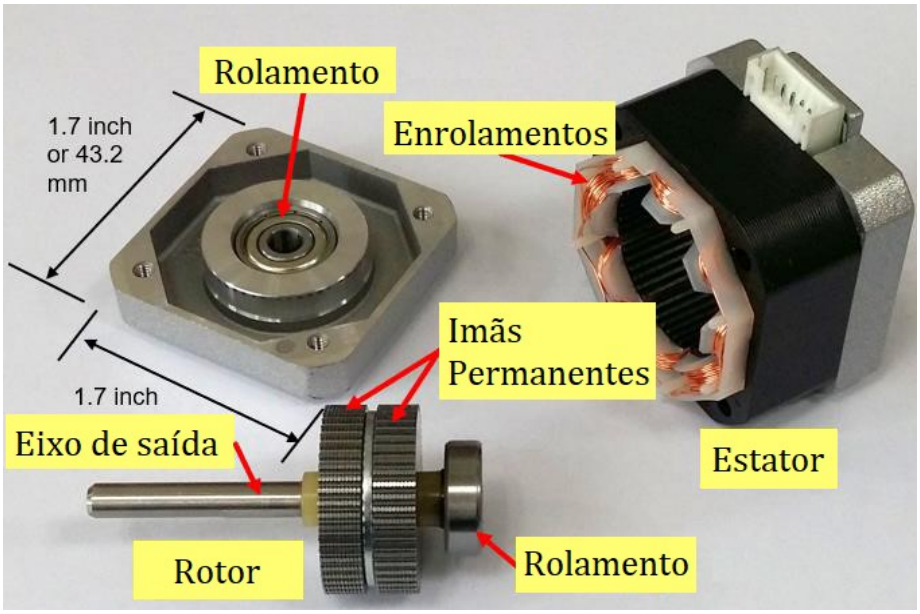
Após as manipulações algébricas, a equação do torque é dada pela Equação (17):

$$T_r = \frac{N_g k_t}{R_a} V_a - \frac{k_t k_b}{R_a R} x' \tag{17}$$

4.1.1 Motor de Passo

O nome motor de passo é dado devido ao funcionamento do motor, o eixo gira em "passos" quando pulsos são aplicados na sequência correta. A velocidade do motor de passo é definida pela frequência com que esses pulsos são enviados e o número de voltas do eixo é definido pela quantidade de pulsos. O dispositivo que faz o controle desses pulsos elétricos que são enviados para o motor, é chamado *driver* (NEOMOTION, 2021). A Figura 13 mostra um motor de passo desmontado do tamanho NEMA-17.

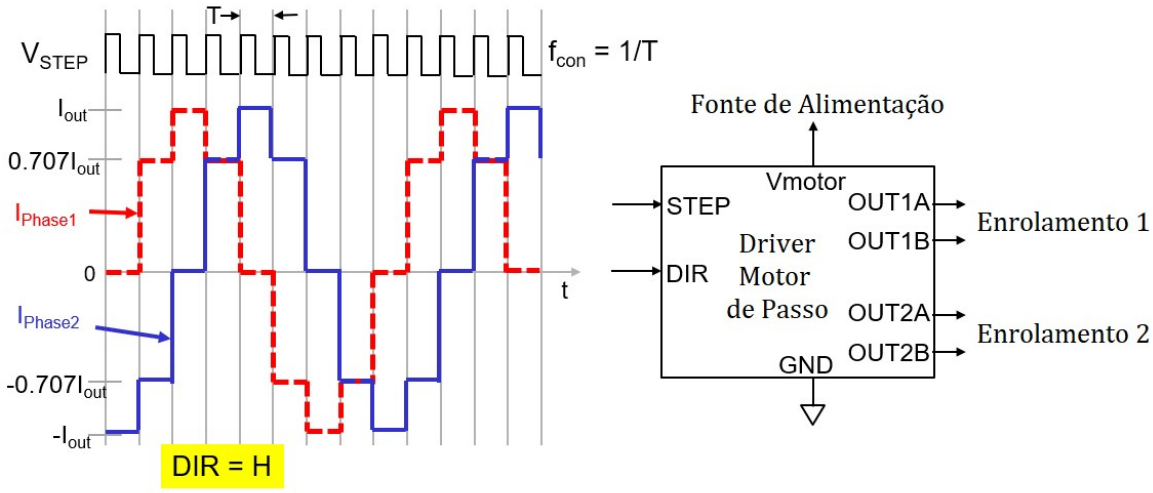
Figura 13 - Motor de passo desmontado



Fonte: Adaptada de Kung (2019)

Para fazer um motor de passo girar no sentido horário ou anti-horário, a corrente elétrica com a sequência correta precisa ser injetada nos enrolamentos. Normalmente a corrente em cada enrolamento é uma série de etapas que se assemelham a uma onda senoidal, as formas de onda entre o enrolamento 1 e o enrolamento 2 são defasadas em 90°. O *driver* recebe uma entrada na forma de pulsos de tensão periódicos. Com cada pulso, a sequência correta de corrente elétrica será imposta aos enrolamentos, girando o eixo do motor de passo em uma quantidade fixa, normalmente 1,8° para passo completo ou 0,9° para operação de meio passo (1/4, 1/8, 1/16 passos também são possíveis). O *driver* também contém outra entrada que determina a direção de rotação. Este conceito é ilustrado na Figura 14 para um *driver* comercial A4988 para operação de meio passo, com *STEP* e *DIR* sendo as entradas para controle de passo e direção (KUNG, 2019).

Figura 14 - Relação entre pulso e as correntes para operação em meio-passo.



Fonte: Adaptada de Kung (2019)

Os motores de passo mais comuns, são os motores que cada passo corresponde à 1,8°. Uma volta completa tem 360°, portanto esse motor precisa de 200 passos para completar uma volta, como mostrado na Equação (18) (MURTA, 2018).

$$PPR = \frac{360}{1{,}8} = 200 \tag{18}$$

Onde PPR é a sigla para Pulsos por Revolução. Quanto menor o modo de passo, maior é o número PPR, permitindo uma maior precisão no controle do motor. Em contra partida, o modo micro-passo produz um torque menor do que o modo passo completo, devido à redução de corrente usada nesse modo (MURTA, 2018). A Tabela 2 mostra os modos de passo que podem ser utilizados nos motores de passo.

Tabela 2 - Modos de passo do motor

Modo	**Pulsos por Revolução (PPR)**
Passo Completo (*Full Step*)	200
Meio Passo (*Half Step*)	400
Micro Passo (MP 1/4)	800
Micro Passo (MP 1/8)	1600
Micro Passo (MP 1/16)	3200

Fonte: Adaptada de Murta (2018)

Por exemplo, assumindo a operação de meio passo (0,9° / passo), para girar o eixo do motor de passo em 45°, deve-se aplicar 50 pulsos no *driver*. Se esses 50 pulsos forem enviados em 1 segundo (50Hz), a velocidade de rotação será de 0,125 RPS ou 45°/s. Se isso for feito dentro de 0,25s (200Hz), a velocidade de rotação será de 0,5 RPS ou 180°/s, desde que a carga no eixo do motor esteja dentro do limite de torque do motor de passo. Assim, a velocidade de rotação do eixo de saída do motor de passo é proporcional à frequência do controle de passo (KUNG, 2019).

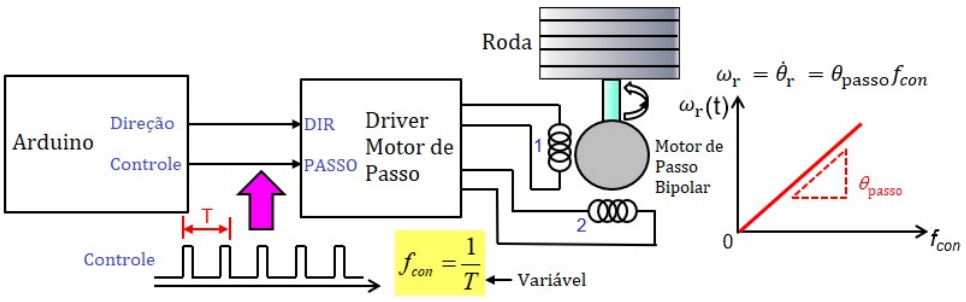
Usando os motores de passo bipolares de tamanho NEMA 14 e NEMA 17, para velocidade de rotação lenta a moderada abaixo de 600 RPM ou 10 RPS, o torque em baixas velocidades do motor de passo pode ser aproximado como constante (a tensão contra eletromotriz induzida nos enrolamentos não é significativa em baixa velocidade de rotação). A maioria dos sistemas de pêndulo invertido não exige uma velocidade de rotação da roda acima de 300 RPM para se manter na posição vertical, portanto, esta aproximação é válida na maioria dos casos (KUNG, 2019).

Portanto, é vantajoso considerar o motor de passo como um atuador rotacional de primeira ordem cujo torque de saída é constante e a velocidade rotacional é proporcional à frequência do sinal de controle. Seja θ_{passo} o ângulo de passo e f_{con} a frequência dos pulsos de tensão aplicados à entrada *STEP*, θ_r é o ângulo da roda e θ_r' é a velocidade angular da roda. A roda é conectada ao eixo de saída do motor de passo. Então, a relação de entrada-saída para o motor de passo é mostrada na Equação (19) (KUNG, 2019):

$$\theta_r' = \frac{d\theta_r}{dt} \cong \theta_{passo} f_{con} \tag{19}$$

Ângulo de passo θ_{passo} será positivo ao girar no sentido horário e negativo no sentido anti-horário. Esse relacionamento é ilustrado na Figura 4.

Figura 15 - A relação de entrada-saída de primeira ordem para motor de passo.

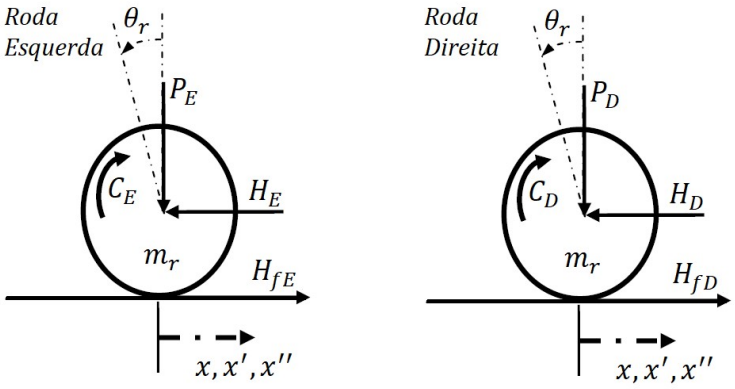


Fonte: Adaptada de Kung (2019)

4.2 Equações para as rodas

O diagrama de corpo livre de ambas as rodas é mostrado na Figura 16. Considerando que as duas rodas são idênticas e o conjunto do sistema é simétrico, as equações para ambas as rodas são análogas, então basta fazer o equacionamento para uma roda.

Figura 16 - Forças que atuam sobre a roda



Fonte: Adaptado de Ooi (2003)

4.2.1 Roda direita

Utilizando-se das leis de movimento definidas por Newton, as forças horizontais atuantes sobre a roda são descritas pela Equação (20).

$$m_r x'' = H_{fD} - H_D \tag{20}$$

Onde H_D a componente horizontal da força de reação do chassi e a roda direita, H_{fD} a força de fricção entre a roda e a superfície e m_r a massa da roda.

A soma dos momentos em torno do eixo da roda é mostrada na Equação (21):

$$J_r \theta_r'' = C_D - H_{fD} R \tag{21}$$

Sendo J_r o momento de inércia da roda, C_D o torque aplicado pelo motor à roda direita, θ_r'' a aceleração angular da roda e R o raio da roda.

Isolando a força de reação H_{fD} da Equação (21), resulta na Equação (22):

$$H_{fD} = \frac{C_D}{R} - \frac{J_r}{R}\theta_r'' \tag{22}$$

Substituindo a Equação (22) na (20), tem como resultado a Equação (23):

$$m_r x'' = \frac{C_D}{R} - \frac{J_r}{R}\theta_r'' - H_D \tag{23}$$

Assumindo que não há escorregamento nas rodas, a rotação das rodas pode ser expressa em termos de deslocamento como mostrada na Equação (24):

$$\theta_r'' R = x'' \Rightarrow \theta_r'' = \frac{x''}{R} \tag{24}$$

Substituindo a Equação (24) na (23), resulta na Equação (25), a equação que modela o comportamento da roda direita.

$$\left(m_r + \frac{J_r}{R^2}\right) x'' = \frac{C_D}{R} - H_D \tag{25}$$

4.2.2 Roda esquerda

A modelagem da roda esquerda é feita de maneira análoga ao apresentado para a roda direita na Seção 4.2.1. Considerando que a roda esquerda é idêntica a roda direita e que estão sobre uma superfície de mesmo material, logo a equação da roda esquerda é dada pela Equação (26):

$$\left(m_r + \frac{J_r}{R^2}\right) x'' = \frac{C_E}{R} - H_E \tag{26}$$

4.2.3 Ambas as rodas

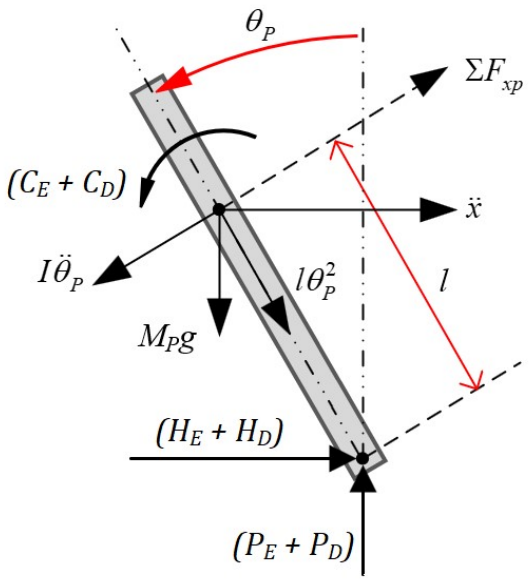
A equação para as duas rodas é obtida através da soma entre as Equações (25) e (26), resultando na Equação (27).

$$2\left(m_r + \frac{J_r}{R^2}\right) x'' = \frac{C_D + C_E}{R} - (H_D + H_E) \tag{27}$$

4.3 Equações para o pêndulo

A Figura 17 mostra o diagrama de corpo livre do pêndulo, o ângulo θ_p é a variável de saída do sistema, que deve ser mantida próximo de 0° para manter o sistema em equilíbrio.

Figura 17 - Forças que atuam sobre pêndulo



Fonte: Adaptado de Kankhunthod (2019)

O centro de massa da haste está localizado no ponto (x_g, y_g), em que x_g e y_g são mostrados, respectivamente, pelas Equações (28) e (29).

$$x_g = x + l \sin\theta_p \tag{28}$$

$$y_g = l \cos\theta_p \tag{29}$$

Derivando as equações de posição (28) e (29), resulta nas equações de velocidade (30) e (31):

$$x_g' = x' + l\theta_p' \cos\theta_p \tag{30}$$

$$y_g' = -l\theta_p' \sin\theta_p \tag{31}$$

E derivando as equações de velocidade (30) e (31), resulta nas equações de aceleração (32) e (33).

$$x_g'' = x'' + l\theta_p'' \cos\theta_p - l\theta_p'^2 \sin\theta_p \tag{32}$$

$$y_g'' = -l\theta_p'' \sin\theta_p - l\theta_p'^2 \cos\theta_p \tag{33}$$

4.3.1 Forças horizontais

Novamente, utilizando as leis de movimento definidas por Newton, a somatória das forças horizontais que atuam sobre o pêndulo é mostrada nas Equações (34) e (35).

$$(H_D + H_E) = m_p x_g'' \tag{34}$$

$$(H_D + H_E) = m_p x'' + m_p l\theta_p'' \cos\theta_p - m_p l\theta_p'^2 \sin\theta_p \tag{35}$$

Onde m_p é a massa do pêndulo, H_D e H_E são as componentes horizontais da forças de reação entre o chassi e cada roda, l é à distância do centro de massa até o centro das rodas, θ_p'' é a aceleração angular, θ_p' é a velocidade angular e θ_p é o ângulo entre o eixo y e o pêndulo.

4.3.2 Forças perpendiculares

As Equações (36) e (37) mostram as forças perpendiculares que atuam sobre o pêndulo.

$$m_p y_g'' = (P_D + P_E) - m_p g \tag{36}$$

$$(P_D + P_E) = m_p g - m_p l\theta_p'' \sin\theta_p - m_p l\theta_p'^2 \cos\theta_p \tag{37}$$

Onde P_De P_E são as componentes perpendiculares das forças de reação entre o chassi e cada roda e g é a aceleração da gravidade na Terra que corresponde a $9{,}81\ m/s^2$.

4.3.3 Momentos

As leis de Newton também são aplicadas aos momentos angulares da haste. A soma dos momentos em torno do centro de gravidade da haste é dada pelas Equações (38) e (39).

$$\sum T = I\theta_p'' \tag{38}$$

$$J_p\theta_p'' = (P_D + P_E)l \sin\theta_p - (H_D + H_E)l \cos\theta_p - (C_D + C_E) \tag{39}$$

4.4 Combinando as equações

Os principais componentes de cada elemento do PIDR foram modelados, agora deve-se associá-los e resumi-los para obter um sistema completo.

Substituindo as Equações (35) e (37) na Equação (39), os termos $(P_D + P_E)$ e $(H_D + H_E)$ são eliminados resultando na Equação (40), a primeira equação de movimento do sistema.

$$\left(J_p + m_p l^2\right)\theta_p'' = m_p g l \sin\theta_p - m_p l \cos\theta_p \, x'' - (C_D + C_E) \tag{40}$$

O termo $(H_D + H_E)$ da Equação (27), também deve ser removido da dinâmica do sistema. Para isso substitui-se a Equação (35) em (27), resultando na Equação (41) a segunda equação de movimento do sistema.

$$\left(m_p + 2m_r + \frac{2J_r}{R^2}\right)x'' = -m_p l \cos\theta_p \, \theta_p'' - m_p l \sin\theta_p \, \theta_p'^2 + \frac{C_D + C_E}{R} \tag{41}$$

Para um deslocamento retilíneo os torques das rodas direita e esquerda, C_D e C_E, respectivamente, são iguais, portanto podem ser descritos como mostrado na Equação (42):

$$C_D + C_E = 2T_r \tag{42}$$

Reescrevendo as Equações (40) e (41) encontram-se as equações não-lineares que descrevem o comportamento físico do sistema, mostrada nas Equações (43) e (44):

$$\left(J_p + m_p l^2\right)\theta_p'' = m_p l g \sin\theta_p - m_p l \cos\theta_p \, x'' - 2T_r \tag{43}$$

$$\left(m_p + 2m_r + \frac{2J_r}{R^2}\right)x'' = -m_p l \cos\theta_p \, \theta_p'' - m_p l \sin\theta_p \, \theta_p'^2 + \frac{2}{R}T_r \tag{44}$$

4.5 Linearizando as equações

As Equações de movimento (43) e (44) são equações não lineares, devido à presença do seno, cosseno e termos quadrados. Na maior parte do tempo quando em operação a haste estará próxima a posição vertical, o movimento de balanço ou rotação ao longo do eixo da roda é muito pequeno, ou seja, essas equações podem ser linearizadas em torno do ponto de operação. O ponto de operação é o ponto de equilíbrio vertical do pêndulo quando $\theta_p = 0$, pois essa é a região onde se deseja manter o sistema controlado. Se o desvio do pêndulo da posição vertical for pequeno, ou seja, $|\theta_p| \leq 10°$, as linearizações são mostradas nas Equações (45), (46) e (47) (KANKHUNTHOD et. al., 2019):

$$\cos\theta_p \approx 1 \tag{45}$$

$$\sin\theta_p \approx \theta_p \tag{46}$$

$$\theta_p'^2 \approx 0 \tag{47}$$

Substituindo as simplificações para linearização das Equações (45), (46) e (47) nas Equações (43) e (44) tem como resultado as Equações Linearizadas (48) e (49).

$$(J_p + m_p l^2)\theta_p'' = m_p l g \theta_p - m_p l x'' - 2T_r \tag{48}$$

$$\left(m_p + 2m_r + \frac{2J_r}{R^2}\right)x'' = -m_p l \theta_p'' + \frac{2}{R}T_r \tag{49}$$

Ao introduzir os símbolos das Equações (50) a (54), as equações de movimento linearizadas podem ser escritas de uma forma compacta.

$$\mathbb{A} = (J_p + m_p l^2) \tag{50}$$

$$\mathbb{B} = m_p g l \tag{51}$$

$$\mathbb{C} = m_p l \tag{52}$$

$$\mathbb{D} = \left(m_p R + 2m_r R + \frac{2J_r}{R}\right) \tag{53}$$

$$\mathbb{E} = m_p l R \tag{54}$$

Reescrevendo as Equações de movimento (48) e (49) de forma compacta, resulta nas Equações (55) e (56):

$$\mathbb{A}\theta_p'' = \mathbb{B}\theta_p - \mathbb{C}x'' - 2T_r \tag{55}$$

$$\mathbb{D}x'' = -\mathbb{E}\theta_p'' + 2T_r \tag{56}$$

4.6 Transformada de Laplace

Como a análise do sistema no domínio do tempo é mais complexa, e no domínio da frequência tem-se vantagens de simplificar muitas operações matemáticas, realizar-se-á modelagem do sistema no domínio da frequência, utilizando a transformada de Laplace nas Equações (55) e (56), considerando condições iniciais nulas, resulta nas Equações (57) e (58):

$$\mathbb{A}s^2\Theta_{\mathrm{p}}(\mathrm{s}) = \mathbb{B}\Theta_{\mathrm{p}}(\mathrm{s}) - \mathbb{C}s^2X(\mathrm{s}) - 2T_r(\mathrm{s}) \tag{57}$$

$$\mathbb{D}s^2X(\mathrm{s}) = \mathbb{E}s^2\Theta_{\mathrm{p}}(\mathrm{s}) + 2T_r(\mathrm{s}) \tag{58}$$

4.7 Função de transferência

Com as equações de movimento definidas agora deve-se escolher quem será a entrada e a saída do sistema. O ângulo de inclinação com a vertical θ_p será à saída do sistema, então a entrada do sistema deve ser o Torque do motor T_r ou o deslocamento X.

O motor escolhido neste projeto será o motor de passo por possuir algumas vantagens sobre o motor CC, por exemplo: fixação das rodas diretamente no eixo do motor sem utilizar caixa de engrenagens que geram folgas, controle preciso de posição e velocidade via contagens de pulsos, sem atraso de aceleração e desaceleração e torque constante a baixas rotações.

Como já mencionado na Seção 4.1.1, o motor de passo tem um torque de saída constante, então como o torque T_r é constante, a variável para entrada do sistema será o deslocamento X, derivando o deslocamento resulta na velocidade linear X'.

O que será controlado é a velocidade angular do eixo do motor de passo e como a roda está acoplada diretamente ao eixo do motor à velocidade de rotação da roda e do eixo do motor são iguais. Assumindo o movimento antiderrapante, a velocidade do pêndulo x' e a velocidade angular da roda θ_r' é relacionada ao raio da roda R, mostrado na Equação (59).

$$R\theta_r' = x' \xrightarrow{\mathcal{L}} sR\Theta_r = sX \tag{59}$$

Isolando o termo $T_r(s)$ da Equação (58) e substituindo na Equação (57), após algumas manipulações algébricas resulta na Equação (60).

$$\mathrm{G(s)} = \frac{\Theta_{\mathrm{p}}(\mathrm{s})}{sX(\mathrm{s})} = \frac{-(\mathbb{C}+\mathbb{D})s}{(\mathbb{A}-\mathbb{E})s^2 - \mathbb{B}} \tag{60}$$

Substituindo a relação da Equação (59) em (60) e reorganizando, resulta na função de transferência mostrada na Equação (61), onde a entrada é a velocidade angular da roda, ou do motor, dado por $s\Theta_r(s)$ e a saída o ângulo de inclinação do pêndulo dado por $\Theta_{\mathrm{p}}(\mathrm{s})$.

$$\mathrm{G(s)} = \frac{\Theta_{\mathrm{p}}(\mathrm{s})}{s\Theta_r(s)} = \frac{-(\mathbb{C}+\mathbb{D})Rs}{(\mathbb{A}-\mathbb{E})s^2 - \mathbb{B}} \tag{61}$$

A Equação (61) indica que, com o ajuste adequado da velocidade angular Θ'_r do motor de passo, é possível manter Θ_{p} próximo a zero grau, ou seja, posiciona o sistema na vertical.

Analisando a função de transferência na Equação (61), pode-se determinar a estabilidade do sistema. De acordo com Nise (2013), uma condição suficiente para que o sistema seja instável é que os sinais do denominador da função de transferência sejam diferentes. Portando como pode ser visto na Equação (61), os sinais dos coeficientes do polinômio do denominador não são iguais e ainda a ausência do termo s, indicando que este sistema é instável.

4.8 Identificação dos parâmetros do sistema

As equações obtidas para o sistema pêndulo invertido contêm várias constantes como a massa das rodas m_r, massa do chassi m_p, raio da roda R, entre outras. Cada constante será medida e analisada separadamente. Em seguida, as especificações observadas foram utilizadas para modelagem matemática e simulação do sistema.

4.8.1 Massa do chassi

A massa do chassi foi obtida com o uso de uma balança. Foi medida a massa apenas do chassi, as rodas e o rotor dos motores foram removidos.

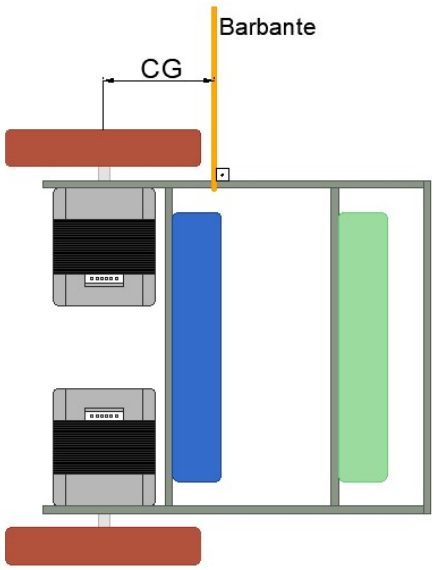
4.8.2 Massa das rodas

As massas das rodas é a soma da massa da roda e a massa do rotor do motor.

4.8.3 Localização do centro de gravidade

A localização do centro de gravidade foi obtida pendurando o pêndulo horizontalmente com um barbante, ajustando a posição do barbante até que o pêndulo ficasse na horizontal. Em seguida, foi medida a distância entre o barbante, localização do centro de gravidade, e o eixo do motor. A Figura 18 mostra o procedimento realizado, onde CG é a medida do centro de gravidade do pêndulo.

Figura 18 - Localização do centro de gravidade



Fonte: Próprio autor

4.8.4 Momento de inércia do chassi

Considerando que o pêndulo tenha formato retangular e massa uniforme, o momento de inércia pode ser calculado usando a Equação (62), L e H são a largura e altura, respectivamente.

$$J_P = \frac{1}{12} m_p (L + H)^2 \tag{62}$$

4.8.5 Momento de inércia das rodas

Considerando a distribuição de massa da roda uniforme, o momento de inércia é mostrado na Equação (63), onde m_r é a massa e R é o raio da roda.

$$J_r = \frac{1}{2} m_r R^2 \tag{63}$$

4.8.6 Parâmetros físicos do sistema

A Tabela 3 contém todos os parâmetros físicos do sistema pêndulo invertido.

Tabela 3 - Parâmetros físicos do sistema

Símbolo	Valor	Unidade	Parâmetro
g	9,807	m/s^2	Aceleração da gravidade
m_p	0,850	kg	Massa do chassi
J_p	0,0039	kgm^2	Momento de inércia do chassi
R	0,040	m	Raio das rodas
J_r	0,0000928	kgm^2	Momento de inércia da roda
m_r	0,116	kg	Massa das rodas
H	0,160	m	Altura do chassi
L	0,075	m	Largura do chassi
l	0,045	m	Distância do centro de massa até o centro da roda

Fonte: Próprio autor

4.9 Função de transferência finalizada

Substituindo os valores da Tabela 3 na Equação (61), tem-se a função de transferência do sistema, dada pela Equação (64):

$$G(s) = \frac{\Theta_p(s)}{s\Theta_r(s)} = \frac{-0{,}003447s}{0{,}004103s^2 - 0{,}3751} = \frac{-0{,}8401s}{s^2 - 91{,}4209} \tag{64}$$

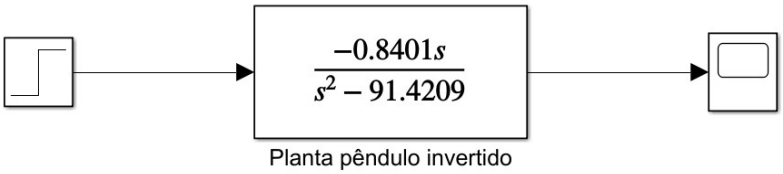
5 PROJETO DO CONTROLADOR

Antes de projetar o controlador é necessário verificar o comportamento do sistema em malha aberta e em malha fechada.

5.1 Comportamento em Malha Aberta

A Figura 19 mostra o sistema em Malha Aberta com a função de transferência encontrada na Equação (64), considerando uma resposta à entrada degrau.

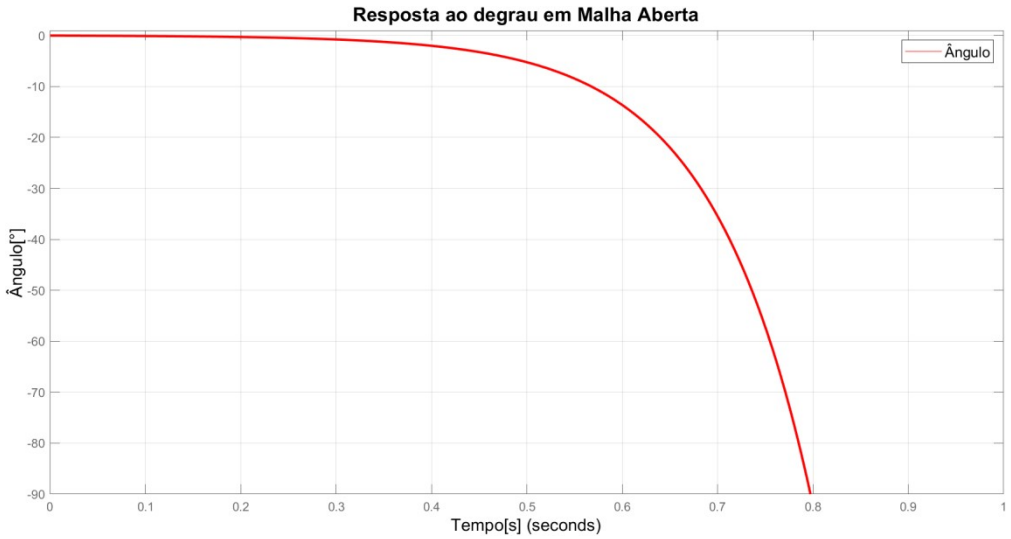
Figura 19 - Diagrama em blocos em Malha Aberta



Fonte: Próprio autor

Uma resposta ao degrau do sistema em malha aberta é mostrada na Figura 20. O ângulo de inclinação θ_p aumenta rapidamente resultando na queda do pêndulo. Esta resposta indica que o sistema é instável.

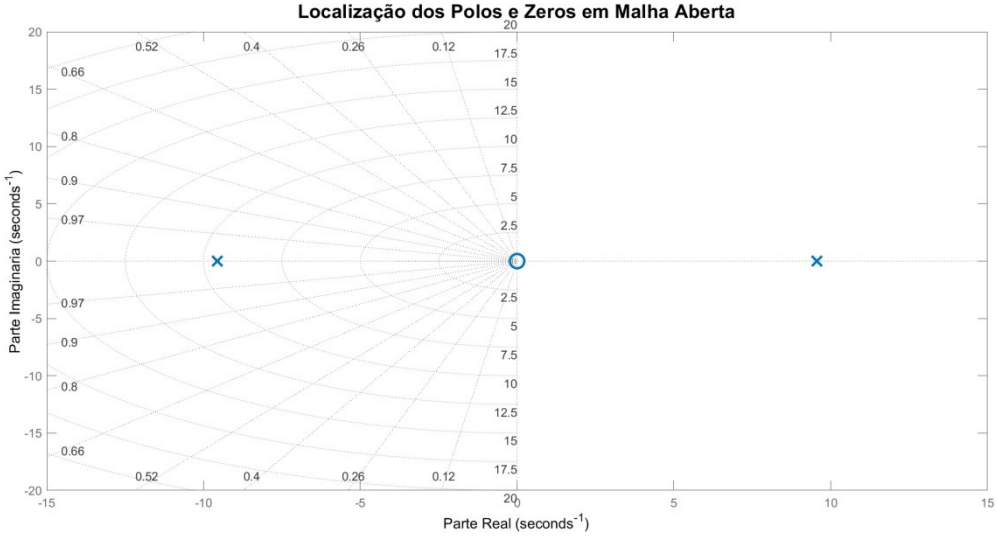
Figura 20 - Resposta ao degrau do sistema em Malha Aberta



Fonte: Próprio autor

A Figura 21 apresenta o gráfico com a localização dos polos e zeros em malha aberta no plano s. Como um dos polos da função de transferência encontra-se no semiplano direito do plano s significa que o sistema é instável.

Figura 21 – Localização dos Polos e Zeros em Malha Aberta

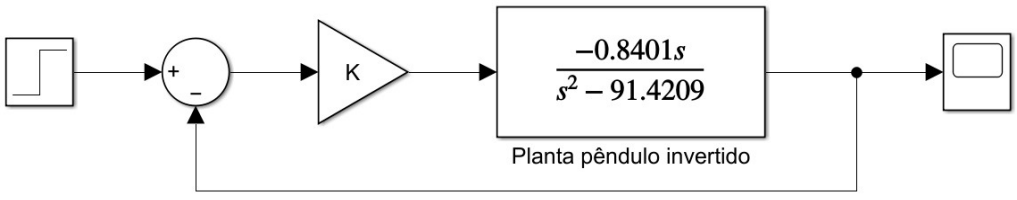


Fonte: Próprio autor

5.2 Comportamento em Malha Fechada

Em seguida antes de projetar a compensação é interessante observar a resposta do sistema em malha fechada com realimentação unitária. Muitos sistemas instáveis em malha aberta passam a ser estáveis em malha fechada. O contrário também é possível que um sistema estável em malha aberta se torne instável em malha fechada, embora muito raros. O diagrama em blocos do sistema em malha fechada com realimentação unitária e compensador Proporcional é ilustrado na Figura 22.

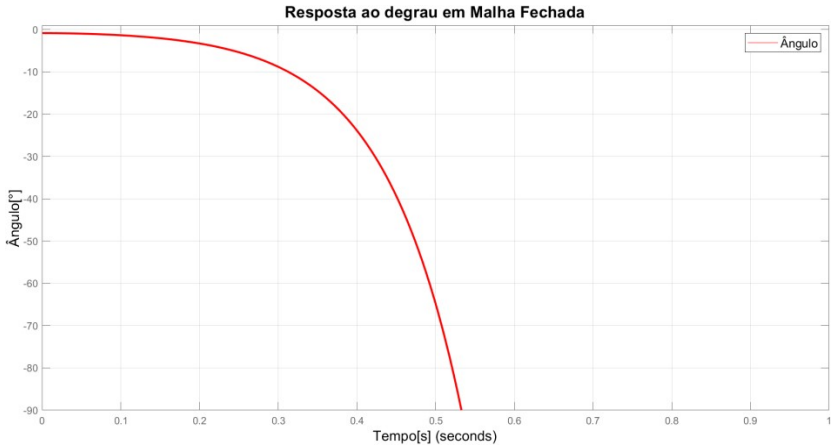
Figura 22 - Diagrama em blocos em Malha Fechada



Fonte: Próprio autor

A resposta ao degrau do sistema em malha fechada é mostrada na Figura 23. O ângulo aumenta rapidamente, indicando instabilidade.

Figura 23 - Resposta ao degrau do sistema em Malha Fechada

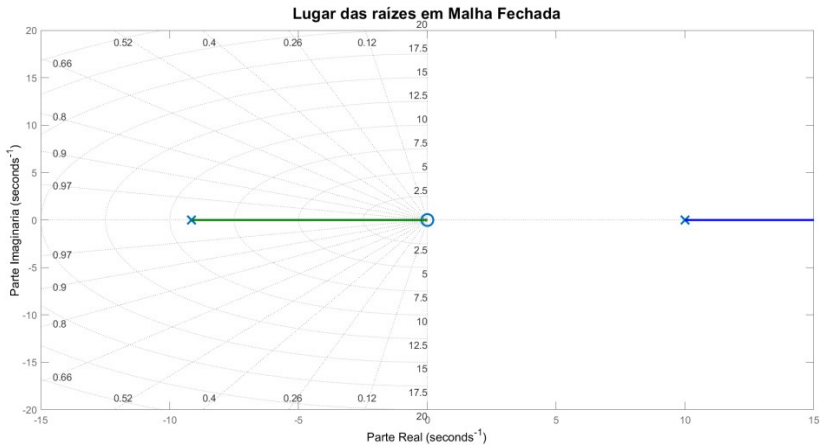


Fonte: Próprio autor

O numerador da função de transferência em malha fechada, conforme pode ser visto na Equação (65), tem um sinal negativo, portanto para traçar o lugar das raízes são utilizadas as regras para um sistema com realimentação positiva. Nos sistemas com realimentação positiva, o lugar das raízes existe à esquerda de um número par de polos e/ou zeros finitos sobre o eixo real (NISE, 2013). O lugar das raízes em malha fechada é mostrado na Figura 24.

$$G(s) = \frac{-0{,}8401s}{(s - 9{,}991)(s + 9{,}151)} \tag{65}$$

Figura 24 - Lugar das Raizes em Malha Fechada



Fonte: Próprio autor

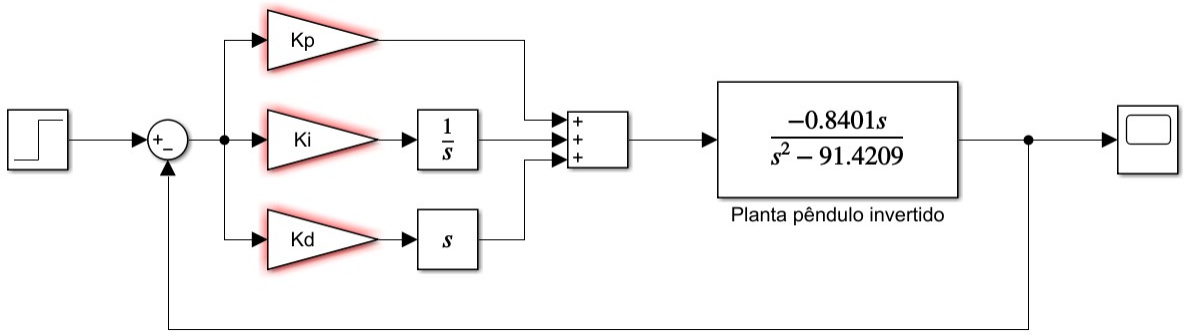
O gráfico mostra que o sistema não pode ser controlado apenas por um controlador Proporcional em malha fechada. Qualquer que seja o valor do ganho de malha fechada K, um ramo do lugar das raízes ainda permanece do lado direito (região instável) do plano s.

A realocação do lugar das raízes do sistema é necessária para que para certa faixa de ganhos, o sistema tenha todas suas raízes do lado esquerdo (região estável) do plano s.

5.3 Projeto do compensador

É necessário um compensador para estabilizar o sistema, pois como observado na Sessão 5.2, mesmo o sistema em malha fechada permanece instável. Portanto deve-se projetar um compensador de modo que o lugar das raízes tenha as suas raízes no semiplano esquerdo do plano s, colocando-as assim na região estável. O compensador será um PID que é constituído de três componentes: um termo proporcional, um integral e outro derivativo como já mostrado na Sessão 3.3. O sistema com compensador PID é mostrado na Figura 25. O projeto constitui no cálculo dos ganhos K_p, K_i e K_d.

Figura 25 - Sistema com Compensador PID



Fonte: Próprio autor

A técnica escolhida para projeto do controlador é o método do Lugar das Raízes (*Root Locus*). Esta técnica consiste em analisar o lugar das raízes e adicionar polos e zeros no compensador de modo a modelar o local das raízes e encontrar os ganhos K_p, K_i e K_d para atender os requisitos de projeto. Portanto o sistema deve atender os seguintes requisitos de projeto:

- Tempo de acomodação T_a até 0.5s;
- Sobre-sinal OS (*overshoot*) máximo de 10%;
- Erro de regime permanente inferior a 2%.

Com estes requisitos serão encontrados os polos de projeto, ou seja, o local onde os requisitos são atendidos e as constantes K_p, K_i e K_d do controlador PID.

O projeto do controlador foi dividido em três partes, primeiro é calculado os polos de projeto, em seguida um controlador PD e depois um controlador PI. Após o projeto dos controladores PD e PI, a combinação dos dois controladores forma o controlador PID.

5.3.1 Cálculo do polo de projeto

Primeiro são obtidos os polos de projeto, ou seja, o local onde o sistema deve operar para satisfazer os requisitos do projeto. Os polos de projeto são obtidos pela Equação (66):

$$P_d = -\xi\omega_n \pm j\omega_n\sqrt{1-\xi^2} \tag{66}$$

Para calcular o fator de amortecimento ξ e a frequência natural ω_n são utilizadas as Equações (67) e (68), respectivamente.

$$\xi = \frac{-\ln(OS/100)}{\sqrt{\pi^2 + \ln^2(OS/100)}} * 100 = \frac{-\ln(10/100)}{\sqrt{\pi^2 + \ln^2(10/100)}} = 0{,}591 \tag{67}$$

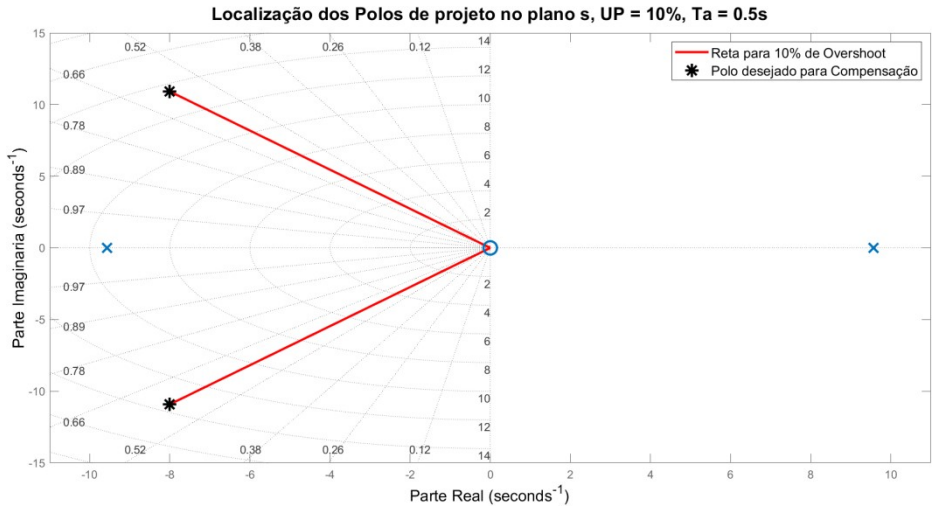
$$\omega_n = \frac{4}{T_a\xi} = \frac{4}{0.5 * 0{,}591} = 13{,}533\ rad/s \tag{68}$$

Substituindo o resultado de (67) e (68) na Equação (66) os polos de projeto são dados pela Equação (69):

$$P_d = -8 \pm j10{,}915 \tag{69}$$

A localização dos polos desejados são marcados no gráfico do pano s, como mostrado na Figura 26. A inclinação da reta vermelha está associada ao fator de amortecimento para um sobre-sinal de 10% e o polo desejado é o asterisco preto.

Figura 26 - Destaque dos polos de projeto no plano s



Fonte: Próprio autor

5.3.2 Projeto do Compensador Proporcional Derivativo (PD)

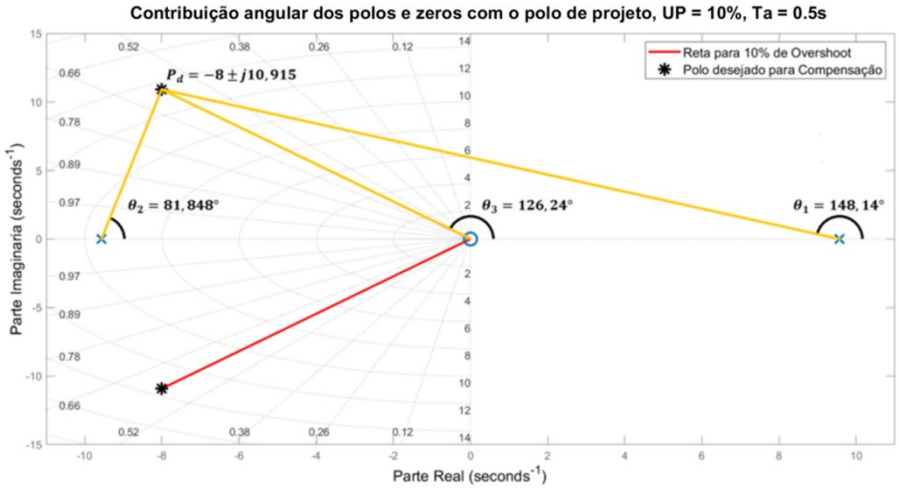
Como o lugar das raízes do sistema não compensado em malha fechada não passa pelo polo de projeto, o lugar geométrico das raízes deve ser modificado de modo que o lugar das raízes com compensação passe pela posição escolhida. Para modificar o lugar das raízes, polos e zeros podem ser adicionados para produzir uma nova função de transferência cujo lugar das raízes passe pelo ponto de projeto no plano s (NISE, 2013).

O compensador PD, cuja função de transferência é dada pela Equação (70), adiciona um zero ao sistema, a adição deste zero tende a aumentar a velocidade do sistema original.

$$G_{PD}(s) = K_{PD}(s + z_c) \tag{70}$$

A posição do zero é calculada utilizando a condição de fase, encontrando a contribuição de ângulo de cada polo e zero do lugar das raízes, como mostra a Figura 27.

Figura 27 - Contribuição angular dos polos e zeros com o polo de projeto



Fonte: Próprio autor

Como a função de transferência é negativa, serão utilizas as regras do lugar das raízes para um sistema com realimentação positiva. Portanto a soma dos ângulos dos polos e zeros de malha aberta deve ser igual a 0° ± k360° conforme a Equação (71) (OGATA, 2013). A contribuição de fase do zero do compensador PD é calculada pela condição de fase como mostrado na Equação (72).

$$\sum \hat{a}ngulos\ do\ zeros + \theta_z - \sum \hat{a}ngulos\ dos\ polos = k360° \qquad (k = 0, 1, 2, \ldots) \tag{71}$$

$$\theta_z = (148{,}14 + 81{,}858) - (126{,}24) = 103{,}758° \tag{72}$$

De posse do ângulo do zero do controlador PD, utilizando relações trigonométricas é possível calcular a posição do zero do controlador PD, conforme Equação (73).

$$\tan \theta_z = \frac{\omega_d}{z_c - \sigma_d} \tag{73}$$

Utilizando a Equação (74) resulta no valor de z_c, a posição do zero do controlador PD.

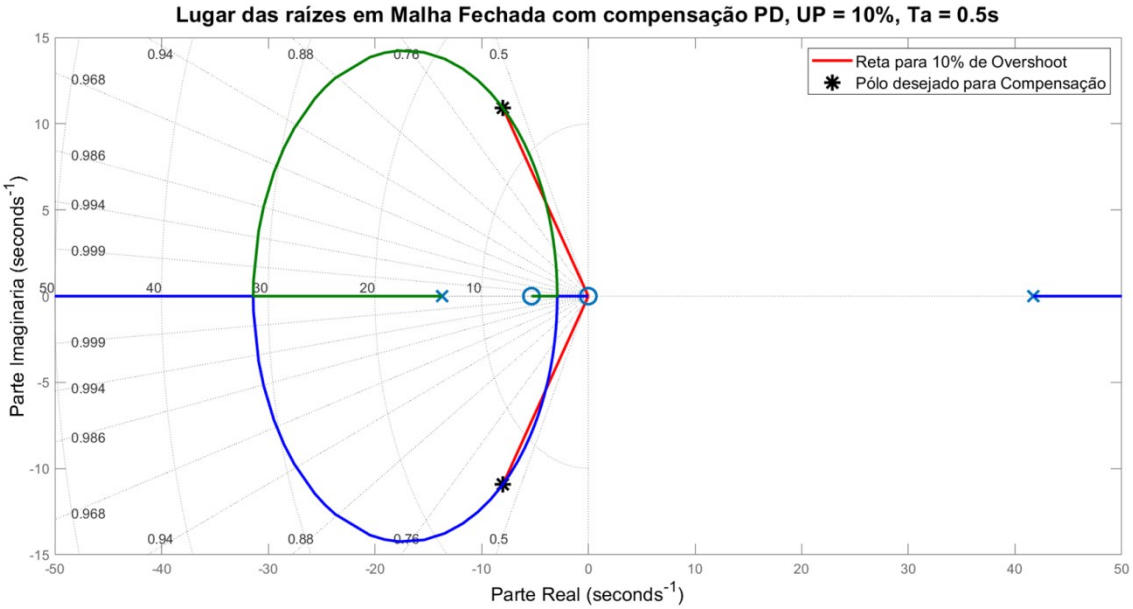
$$z_c = \frac{\omega_d}{\tan \theta_z} + \sigma_d = \frac{10{,}915}{\tan 103{,}758} + 8 = 5{,}3277 \tag{74}$$

Portanto substituindo o valor encontrado na Equação (74) em (70), resulta na função de transferência do controlador PD como mostrada na Equação (75).

$$G_{PD}(s) = K_{PD} \cdot (s + 5{,}3277) \tag{75}$$

O lugar geométrico das raízes para o sistema em malha fechada com compensação PD é mostrado na Figura 28.

Figura 28 - Lugar da raízes com compensação PD



Fonte: Próprio autor

O circulo formado pela adição do zero do controlador PD possibilita a estabilidade do sistema ao aplicar um ganho (K_{PD}) tal que o sistema opere na região desejada.

Para encontrar o valor do ganho K_{PD} para que o sistema possa operar em estabilidade, é utilizada a condição de módulo, mostrado na Equação (76) (NISE, 2013):

$$K = \frac{1}{|G_{PD}(s)| \cdot |G(s)|} \tag{76}$$

Substituindo as funções de transferência do sistema $G(s)$ e do compensador PD $G_{PD}(s)$, e fazendo s o polo desejado, o ganho K_{PD} para que o sistema opere de acordo com os requisitos de projeto é dada pela Equação (77).

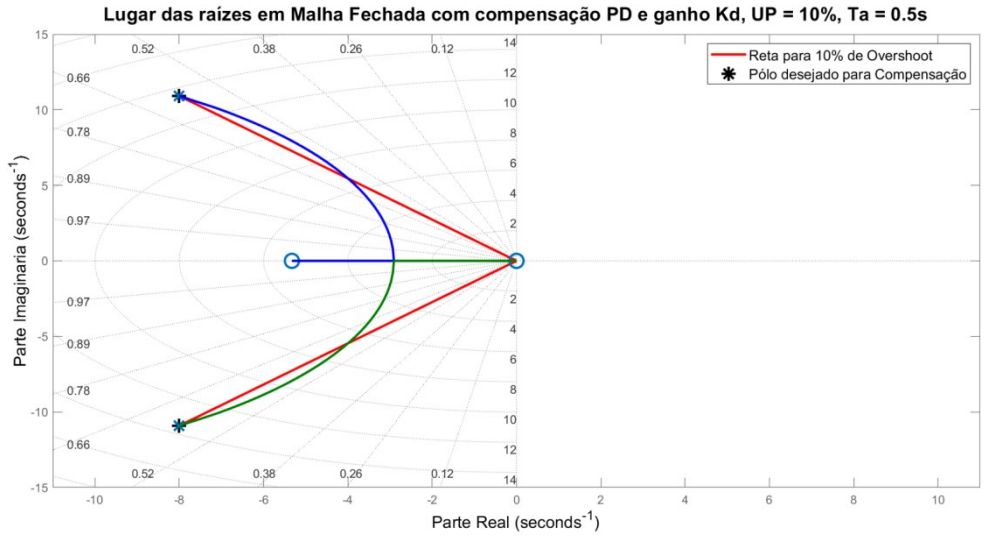
$$K_{PD} = \left| \frac{1}{(s + 5{,}3277) \cdot \left(\frac{-0{,}8401s}{s^2 - 91{,}4209}\right)} \right|_{s=-8+j10{,}915} = 1{,}7846 \tag{77}$$

Logo, a função de transferência do controlador PD é dada pela Equação (78).

$$G_{PD} = 1{,}7846 \cdot (s + 5{,}3277) \tag{78}$$

O lugar das raízes do sistema com controlador PD é mostrado na Figura 29. Pode-se observar que não existem polos do lado direito do plano s, ou seja, o sistema está estável, e aplicando o ganho K_{PD}, o sistema pode operar de acordo com os requisitos de projeto.

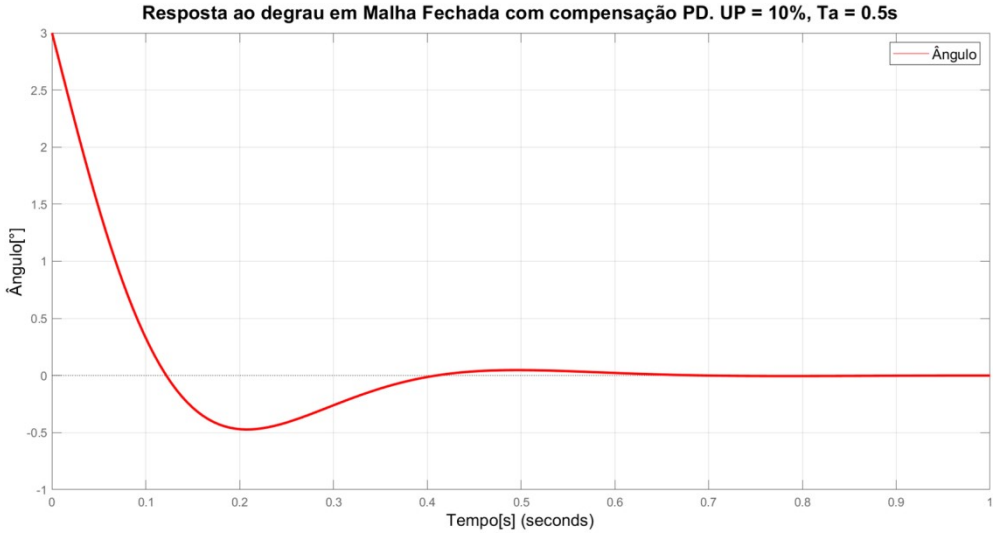
Figura 29 - Gráfico lugar das raízes com controlador PD aplicando ganho K_{PD}



Fonte: Próprio autor

A resposta ao degrau do sistema com controlador PD é mostrada na Figura 30. É possível observar que o sistema tem um pico no inicio do pulso que vai se estabilizando, e que em até 0,5s depois da entrada degrau o sistema já está estabilizado.

Figura 30 - Resposta ao degrau do sistema com controlador PD



Fonte: Próprio autor

5.3.3 Projeto do Compensador Proporcional Integral (PI)

O controlador PI reduz o erro em regime permanente para uma entrada em degrau para zero. O controlador PI adiciona um polo na origem e um zero próximo a origem.

O zero do compensador PI foi calculado de forma análoga ao zero do controlador PD, inserindo o polo na origem e calculando a somatória dos ângulos e fazendo as relações trigonométricas. Entretanto a função de transferência utilizada para o cálculo do controlador PI é a função do sistema em malha aberta, $G(s)$ incluindo o controlador PD sem o ganho, ou seja, a função de transferência usada para calcular o controlador PI é dada pela Equação (79).

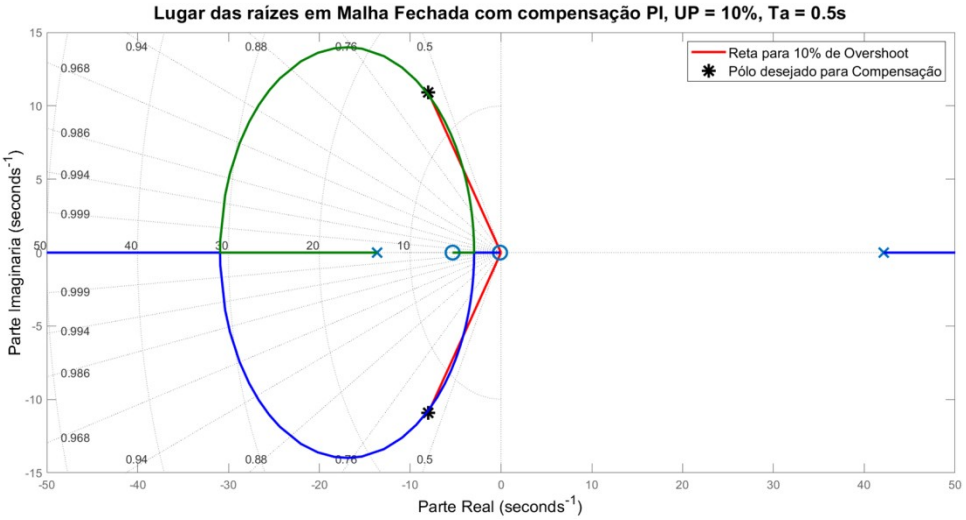
$$G(s) \cdot G_{PD} = \frac{-0{,}8401s^2 - 4{,}5134s}{s^2 - 91{,}4209} \tag{79}$$

Após os cálculos para encontrar o zero do controlador, a função de transferência do controlador PI é mostrada na Equação (80):

$$G_{PI}(s) = K_{PI} \cdot \frac{s + 0{,}1}{s} \tag{80}$$

O lugar geométrico das raízes para o sistema com compensação PI é mostrado na Figura 31. O lugar das raízes é bem parecido com o da Figura 28, do sistema com compensação PD, a diferença é que o controlador PI adicionou um polo na origem anulando o zero da função, e adicionou outro zero em 0,1, próximo a origem

Figura 31 - Lugar da raízes com compensação PI



Fonte: Próprio autor

Deforma análoga ao executado com o controlador PD o ganho do controlador PI é encontrado utilizando a condição de módulo como mostrado na Equação (81).

$$K = \frac{1}{|G_{PD}(s)| \cdot |G_{PI}(s)| \cdot |G(s)|} \tag{81}$$

Portanto o ganho do controlador PI e dado pela Equação (82).

$$K_{PI} = \left| \frac{1}{(s + 5{,}3277) \cdot \left(\frac{s + 0{,}1}{s}\right) \cdot \left(\frac{-0{,}8401s}{s^2 - 91{,}4209}\right)} \right|_{s=-8+j10{,}915} = 1{,}7924 \tag{82}$$

Com o ganho encontrando na Equação (82), a função de transferência do controlador PI é dada pela Equação (83).

$$G_{PI}(s) = 1{,}7924 \cdot \frac{s + 0{,}1}{s} \tag{83}$$

5.3.4 Controlador Proporcional Integral Derivativo (PID)

O controlador PID é a combinação dos controladores PD e PI. Ao conectar os controladores PI e PD em cascata resulta na Equação (84).

$$G_{PID}(s) = G_{PI}(s) \cdot G_{PD}(s) = \frac{K \cdot (s + 0{,}1) \cdot (s + 5{,}3277)}{s} \tag{84}$$

O ganho K do controlador é o mesmo encontrado no projeto do controlador PI, ou seja, $K = K_{PI} = 1{,}7924$. Substituindo K na Equação (86) e após algumas manipulações matemáticas, resulta na Equação (85).

$$G_{PID}(s) = \frac{1{,}7924 \cdot (s + 0{,}01) \cdot (s + 5{,}3277)}{s} = \frac{1{,}7924s^2 + 9{,}7289s + 0{,}9549}{s} \tag{85}$$

Comparando a Equação (85) com (3), os ganhos do controlador PID são:

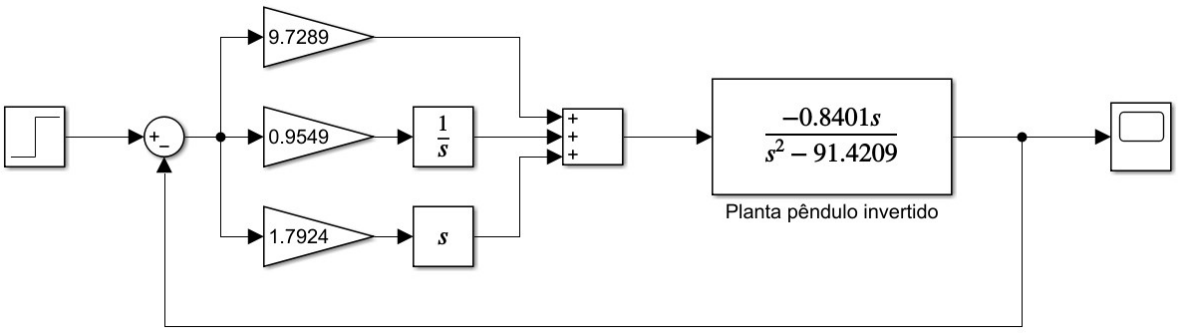
$$K_p = 9{,}7289 \tag{86}$$

$$K_i = 0{,}9549 \tag{87}$$

$$K_d = 1{,}7924 \tag{88}$$

Substituindo os constantes do controlador PID no diagrama em blocos da Figura 25, resulta no diagrama em blocos completo do sistema com a o controlador PID e a planta, com ilustrado na Figura 32.

Figura 32 - Diagrama de blocos do sistema com controlador PID

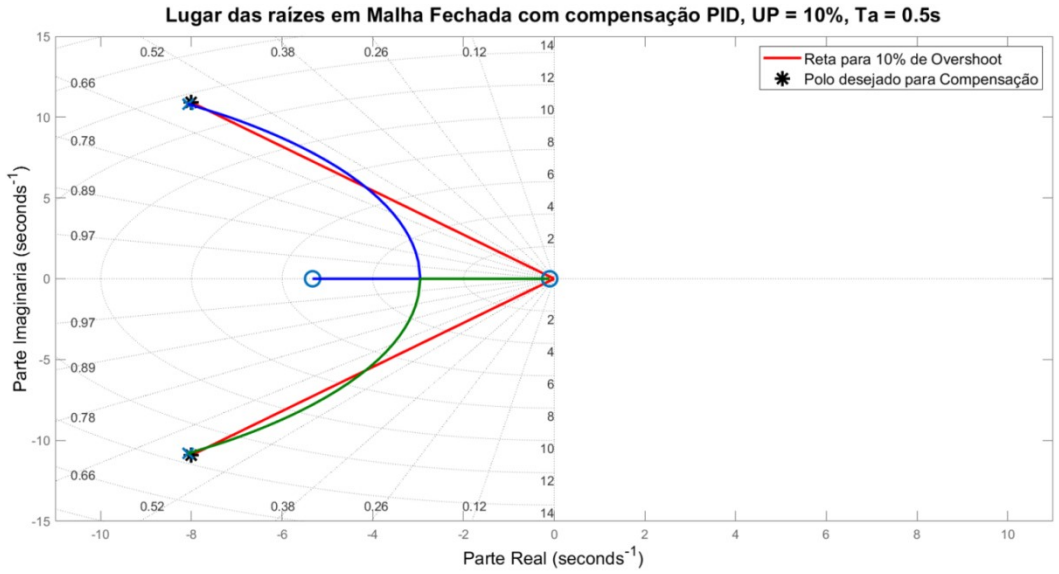


Fonte: Próprio autor

O gráfico do lugar das raízes do sistema completo adicionando as constantes do controlador PID é mostrado na Figura 33. A função de transferência do sistema em malha fechada é mostrada na Equação (89).

$$G_{MF}(s) = \frac{2{,}9772 \cdot (s + 5.3277) \cdot (s + 0{,}1)}{s^2 + 16{,}16s + 182{,}4} \tag{89}$$

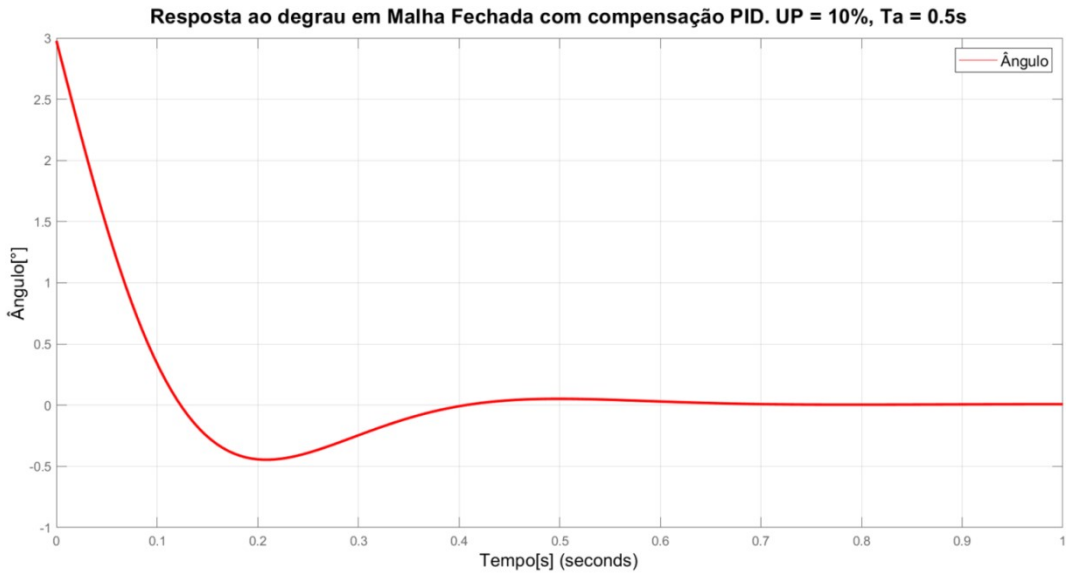
Figura 33 - Lugar da raízes do sistema completo com ganhos do controlador PID



Fonte: Próprio autor

A resposta do sistema ao degrau com o controlador PID é mostrado na Figura 34.

Figura 34 - Resposta ao degrau do sistema com controlador PID



Fonte: Próprio autor

5.4 Discretização do Controlador

O controlador $G_{PID}(s)$ encontrado na seção anterior foi projetado em tempo contínuo, entretanto será utilizando um microcontroloador para fazer o controle do pêndulo. Os microcontroladores trabalham em tempo discreto, ou seja, existe um tempo de amostragem entre as leituras dos sensores. A discretização faz que as integrais e derivadas do controlador em tempo contínuo seja substituída por uma equação de diferenças, e então essa equação de diferenças será implementada no microcontrolador.

Para discretizar o controlador PID contínuo da Equação (4), a parte integral é discretizada utilizando o método de Tustin, também denominado transformação bilinear ou método trapezoidal, mostrada na Equação (90).

$$\int e(t)dt = \frac{(z+1)T_a}{(z-1)2}E(z) \tag{90}$$

Sendo T_a o tempo de amostragem do sinal. A parte derivativa é discretizada utilizando o método de Euler reverso, mostrado na Equação (91).

$$\frac{d}{dt}e(t) = \frac{(z-1)}{zT_a}E(z) \tag{91}$$

Substituindo as Equações (90) e (91) na equação do controlador PID contínuo (4) resulta na Equação (92) e após algumas manipulações algébricas tem-se a Equação (93):

$$\frac{U(z)}{E(z)} = K_p + K_i\frac{T_a(z+1)}{2(z-1)} + K_d\frac{(z-1)}{T_a z} \tag{92}$$

$$\frac{U(z)}{E(z)} = \frac{\left(K_p + \frac{KiT_a}{2} + \frac{K_d}{T_a}\right)z^2 + \left(-K_p + \frac{K_iT_a}{2} - \frac{2K_d}{T_a}\right)z + \frac{K_d}{T_a}}{(z^2 - z)} \cdot \frac{z^{-2}}{z^{-2}} \tag{93}$$

Os termos z^n, quando $n > 0$, não interessam, pois estes são termos futuros do sistema, o que se deseja são os termos anteriores para então aplicar o controle. Para contornar este inconveniente, como visto na Equação (93), basta multiplicar-la por z^{-2}/z^{-2}. Para facilitar a escrita, os termos constantes são substituídos por símbolos como mostrado nas Equações (94), (95) e (96), resultando na Equação (97).

$$a = K_p + \frac{KiT_a}{2} + \frac{K_d}{T_a} \tag{94}$$

$$b = -K_p + \frac{K_i T_a}{2} - \frac{2K_d}{T_a} \tag{95}$$

$$c = \frac{K_d}{T_a} \tag{96}$$

$$\frac{U(z)}{E(z)} = \frac{a + bz^{-1} + cz^{-2}}{(1 - z^{-1})} \tag{97}$$

Rearranjando os termos da Equação (97), resulta na Equação (98):

$$U(z) - U(z)z^{-1} = aE(z) + bE(z)z^{-1} + cE(z)z^{-2} \tag{98}$$

Da propriedade do deslocamento do tempo mostrada nas Equações (99) e (100).

$$\mathcal{Z}\{x[k-n]\} = z^{-n}X[z] \tag{99}$$

$$\mathcal{Z}\{x[k]\} = X[z] \tag{100}$$

Substituindo as relações das Equações (99) e (100) na Equação (98), resulta na equação do controlador PID discreto conforme mostrado na Equação (101).

$$u(k) = u(k-1) + ae(k) + be(k-1) + ce(k-2) \tag{101}$$

O vetor $u[]$ representa à saída do controlador e o vetor $e[]$ representa o sinal de erro que é o valor de referência menos o valor de saída.

Substituindo os valores de K_p, K_i e K_d calculados na Seção 5.3.4 e considerando um tempo de amostragem T_a de 5ms, o controlador PID discreto é dado pela Equação (102).

O tempo de amostragem T_a foi obtido do tempo de execução do programa pelo microcontrolador, que é de aproximadamente 5ms. Também é possível obter o tempo de amostragem em função do tempo de acomodação, geralmente é utilizado um tempo de amostragem quinze vezes menor que o tempo de acomodação do sistema.

$$u(k) = u(k-1) + 368{,}22\, e(k) - 726{,}7\, e(k-1) + 358{,}49\, e(k-2) \tag{102}$$

6 CONSTRUÇÃO DO PÊNDULO INVERTIDO

Após a modelagem do sistema e o projeto do controlador PID, foi feita a construção do protótipo de um pêndulo invertido sobre duas rodas para avaliar o desempenho do controlador que foi projetado.

O protótipo do pêndulo invertido foi construído pelo próprio autor e foram utilizados materiais de fácil acesso e de preço acessivo.

6.1 Lista de materiais

A lista de materiais que foram utilizados para a confecção do protótipo é mostrada na Tabela 4. Os preços foram retirados da internet em vários sites, pois não foi possível encontrar todos os produtos na mesma loja, os preços podem sofrer variações e o frete não foi levado em consideração.

Tabela 4 - Lista de Materiais

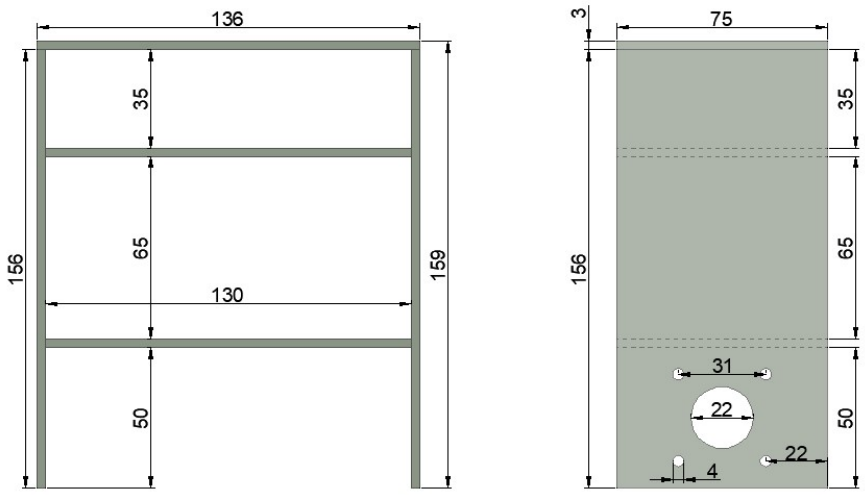
Produto	**Quantidade**	**Preço Unitário**	**Preço Total**
Placas de acrílico 3mm, 30x30cm	1	R$ 37,90	R$ 37,90
Cola instantânea 20g	1	R$ 14,78	R$ 14,78
Placa de desenvolvimento Pro Micro	1	R$ 42,90	R$ 42,90
MPU-6050 Giroscópio e Acelerômetro	1	R$ 16,50	R$ 16,50
Driver motor de passo A4988	2	R$ 16,49	R$ 32,98
Motor de passo 17HS4401	2	R$ 68,21	R$136,42
Módulo *Bluetooth* HC-05 ou HC-06	1	R$ 27,06	R$ 27,06
Bateria Li-Ion 18650 2200mAh	6	R$ 16,24	R$ 97,44
Controlador de carga (BMS 3S)	1	R$ 27,90	R$ 27,90
Regulador de tensão 7805	1	R$ 1,99	R$ 1,99
Capacitor 470uF x 25V	4	R$ 0,40	R$ 1,60
Placa de circuito impresso (opcional)	1	R$ 3,10	R$ 3,10
Matriz de contatos	1	R$ 42,19	R$ 42,19
Cabos, conectores, parafusos	-	-	-
Placa de madeira compensada 15mm	-	-	-
		Preço Total	**R$ 482,76**

Fonte: Próprio autor

6.2 Estrutura mecânica

O chassi do pêndulo foi feito de acrílico com 3 mm de espessura e as rodas com madeira compensada de 15mm de espessura com raio de 40mm. As dimensões do chassi são mostradas na Figura 35.

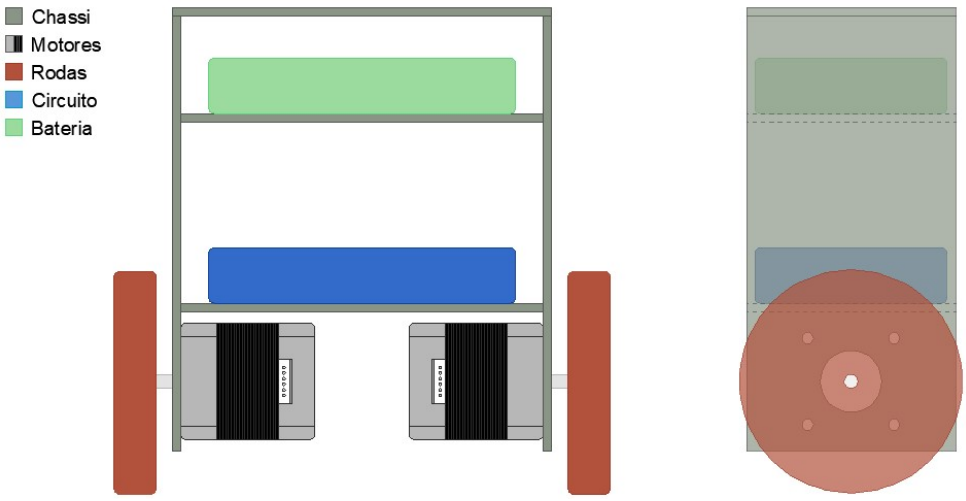
Figura 35 - Dimensões chassi pêndulo invertido



Fonte: Próprio autor

A estrutura é dividida em três partes para facilitar a acomodação dos elementos do pêndulo. Na parte inferior são instalados os motores de passo, logo acima os circuitos eletrônicos para controle e acima é fixada a bateria. A plataforma de cima fica a disposição para o transporte de algum objeto sobre o pêndulo. As rodas são fixadas diretamente ao eixo do motor. A Figura 36 ilustra o pêndulo com a disposição de todos os elementos.

Figura 36 – Pêndulo invertido com a disposição de todos os elementos



Fonte: Próprio autor

6.3 Componentes

Os componentes utilizados durante o desenvolvimento do projeto serão apresentados a seguir.

6.3.1 Microcontrolador

O microcontrolador é o cérebro do projeto, é nele onde os cálculos são efetuados. O microcontrolador faz a leitura dos dados do sensor de inclinação, verifica a posição do pêndulo, faz o cálculo do controle e dependendo do resultado manda o sinal do sentido de rotação para os motores e a velocidade necessária para corrigir o ângulo.

O microcontrolador utilizado é o ATmega32U4, incorporado à uma placa de desenvolvimento com os periféricos básicos para fazer o microcontrolador funcionar, como cristal ressonador, regulador de tensão, porta USB para cominação e alimentação. A placa de desenvolvimento utilizada é a Leonardo Pro Micro, esta foi escolhida devido a sua praticidade, disponibilidade de mercado e baixo custo. A tabela mostra as características principais do ATmega32U4.

Tabela 5 - Caractrísticas principais da placa Leonardo Pro Micro (ATmega32U4)

Microcontrolador	ATmega32U4
Regulador	5V integrado
Tensão de operação	5V
Tensão de entrada (pino RAW)	6-12V
Conexão	USB
Corrente máxima de saída	120mA
Pinos digitais I/O	12 (dos quais 5 oferecem saída PWM)
Pinos de entrada analógica	4
Memória *flash*	32kB (2kB usados pelo *bootloader*)
EEPROM	1kB
SRAM	2kB
Velocidade de *Clock*	16MHz

Fonte: (ATEMEL, 2016)

6.3.2 Sensor de inclinação

O sensor de inclinação utilizado foi o MPU-6050 produzido pela InvenSense. O MPU-6050 é um sensor com seis eixos, ou seja, correspondente aos eixos X, Y e Z para a aceleração através do acelerômetro e os eixos X, Y e Z da velocidade angular através do giroscópio.

Em um instante em que o pêndulo está em repouso sem a ação de forças externas, a única força que atua sobre o pêndulo é a aceleração gravitacional, o ângulo entre a aceleração total e o eixo Z (aceleração gravitacional) é o ângulo θ_p.

Tanto o acelerômetro quando o giroscópio são encapsulados no mesmo *chip*. A comunicação é feita utilizando o protocolo I2C. A Tabela 6 mostra as especificações do sensor MPU-6050.

Tabela 6 - Especificações Acelerômetro e Giroscópio MPU-6050

Chip	MPU-6050
Tensão de operação	3-5V
Conversor AD	16 bits
Comunicação	I2C
Faixa do Giroscópio	±250, ±500, ±1000, e ±2000°/s
Faixa do Acelerômetro	±2g, ±4g, ±8g e ±16g

Fonte: (INVENSENSE, 2013)

6.3.3 *Driver* motor de passo

O módulo *driver* A4988 possui um *chip* A4988, resistores e capacitores, praticamente pronto para utilizar. O *chip* A4988 possui duas pontes H com transistores FET DMOS para fazer o controle de motores de passo bipolares.

Pode ser configurado em até cinco modos de controle: Passo completo, meio passo, um quarto de passo, um oitavo de passo e um dezesseis - avos de passo, também chamados de *micro-stepping*.Tem capacidade de controlador motores com corrente nominal de até 2A e tensão de 35V, além de possuir proteções de curto-circuito e temperatura. A Tabela 7 mostra as especificações do *driver* para motor de passo A4988.

Tabela 7 - Especificações *driver* motor de passo A4988

Tensão de operação	8-35V
Corrente de saída	2A
Tensão lógica	0.3 – 5.5V
Resoluções micropassos	Completo, 1/2, 1/4, 1/8 e 1/16
Proteções	Curto-circuito e sobre aquecimento

Fonte: (ALLEGRO, 2020)

6.3.4 Bateria

A bateria utilizada é um conjunto contendo seis pilhas do modelo 18650 ligadas em um arranjo série/paralelo conhecido como 2P3S, ou seja, são duas pilhas ligadas em paralelo e então os 3 conjuntos em paralelos são ligadas em série, formando um conjunto de 10,8V por 4400mAh. As especificações da pilha 18650 são mostradas na Tabela 8.

Tabela 8 - Especificações pilha 18650

Modelo	18650
Capacidade nominal	2200mAh
Tensão nominal	3,6V
Tensão de carga	4,2V
Tensão de corte de descarga	2,75V
Composição	Íons de Lítio
Peso	45,5g
Comprimento	65mm
Diâmetro	18mm

Fonte: (EEMB, 2015)

6.3.5 Motores

Os motores utilizados no projeto foram os motores de passo modelo 17HS4401. A escolha do motor de passo foi devido ao melhor controle de velocidade e posição e a vantagem de não precisar de caixa de redução para acoplamento da roda, evitando folgas que atrapalham o controle. E ainda o menor custo em relação a um motor com caixa de redução embutida. As especificações do motor de passo estão listadas na Tabela 9.

Tabela 9 - Especificações do motor de passo

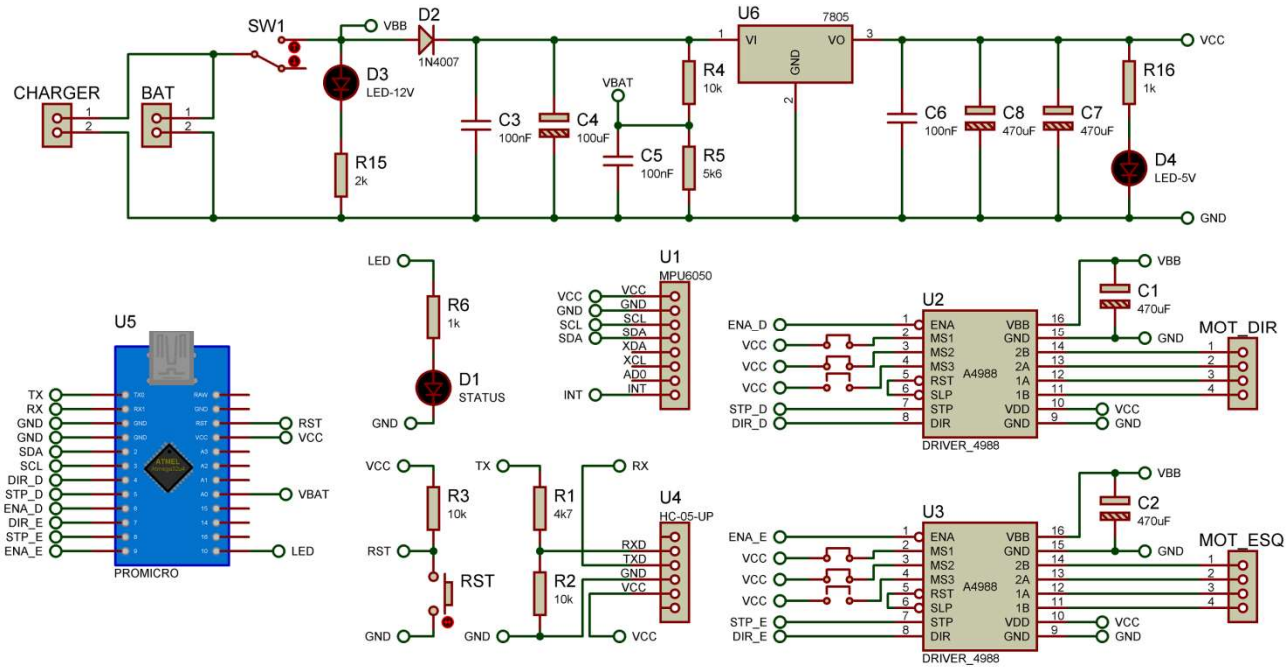
Modelo	17HS4401
Ângulo do passo	1,8°
Número de fase	2 (bipolar)
Corrente	1,7A
Resistência de fase	1,5ohm
Indutância da fase	2,8mH
Torque de retenção	40N.cm
Torque detente	2,2N.cm
Inércia do rotor	54g.cm²
Número de fios	4
Peso	280g

Fonte: (MOTIONKING, 2011)

6.4 Diagrama esquemático

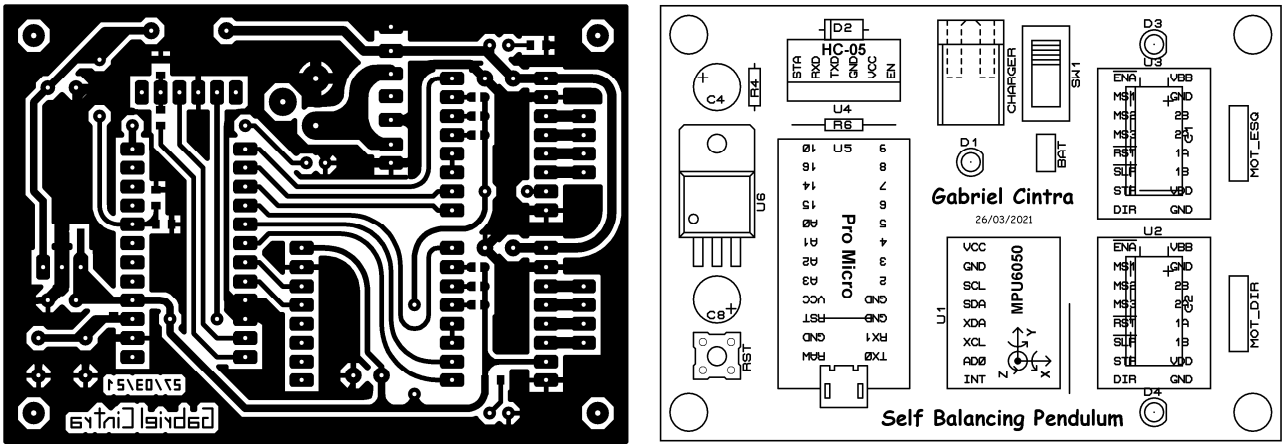
A representação do diagrama esquemático do projeto é mostrada na Figura 37. O diagrama da placa de circuito impresso é mostrado na Figura 38

Figura 37 – Diagrama esquemático do projeto



Fonte: Próprio autor

Figura 38 – Diagrama da placa de circuito impresso do projeto



Fonte: Próprio autor

6.5 Programa

O programa foi desenvolvido em linguagem C++ utilizando a IDE Arduino, com base em pesquisas de projetos similares e bibliotecas de código aberto.

6.5.1 Leitura do ângulo de inclinação

Para medir o ângulo de inclinação são utilizados dois sensores, um acelerômetro e um giroscópio, cada um deles possui duas vantagens e desvantagens, com a combinação da resposta dos dois sensores obtém-se leituras mais precisa do ângulo de inclinação.

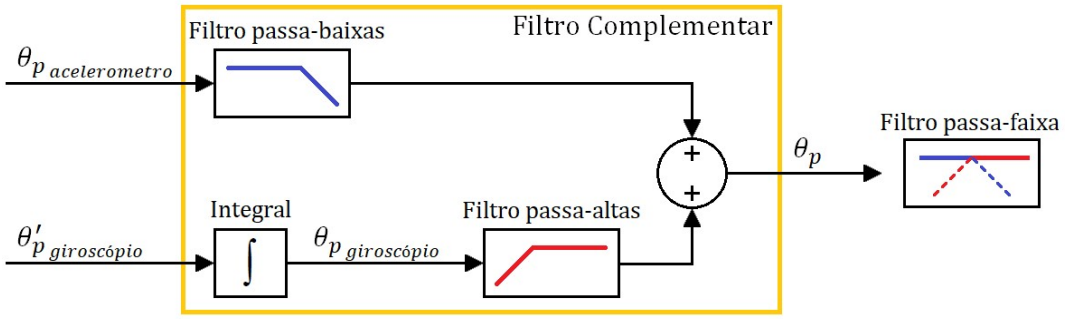
O acelerômetro mede a aceleração em relação ao campo gravitacional da Terra, e com essa informação já é possível obter o ângulo de inclinação, porém o acelerômetro é extremamente susceptível ruído, as vibrações do pêndulo influenciam nas leituras (BUENO; ROMANO, 2011).

O giroscópio mede a velocidade angular, que é integrada em relação ao tempo, resultando no ângulo de inclinação, o problema é que essa integração tende a acumular erro ao passar do tempo e o ângulo se desviará do ângulo real (SCHILING, 2017).

Esses problemas podem ser resolvidos com a combinação de ambos os sensores, isso é chamado de fusão de sensores, existem alguns métodos de combinação, o Filtro de Kalman e o Filtro Complementar. Neste projeto será utilizando o filtro complementar, devido ao seu baixo custo computacional e simples implementação, mostrada na Equação (103). O filtro complementar é uma combinação de um filtro passa alta e um filtro passa baixa. O filtro passa baixa filtra o sinal obtido pelo acelerômetro e o filtro passa alta o sinal do giroscópio já com a integração do sinal (OLIVEIRA; ROSSI, 2015). A Figura 39 mostra o diagrama em bloco do Filtro Complementar.

$$\theta_p = 0{,}995 * \left(\theta_p + \theta'_{p_{giroscópio}} * dt\right) + 0{,}005 * \theta_{p_{acelerômetro}} \tag{103}$$

Figura 39 - Diagrama em blocos Filtro Complementar

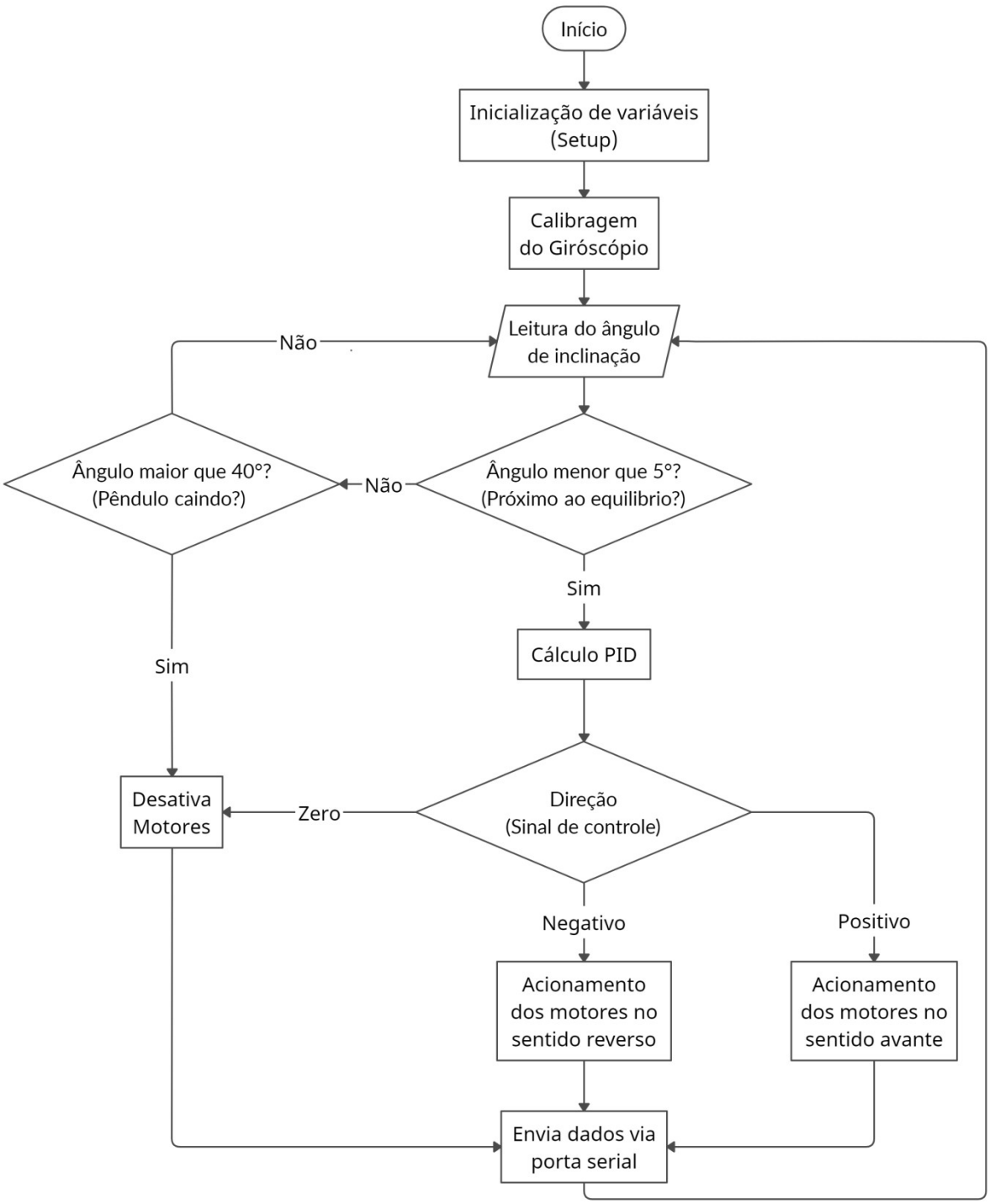


Fonte: Próprio autor

6.5.2 Fluxograma do programa

O programa funciona em um *loop* de aproximadamente 5ms, a Figura 40 mostra o fluxograma do programa de controle do sistema.

Figura 40 - Fluxograma programa de controle



Fonte: Próprio autor

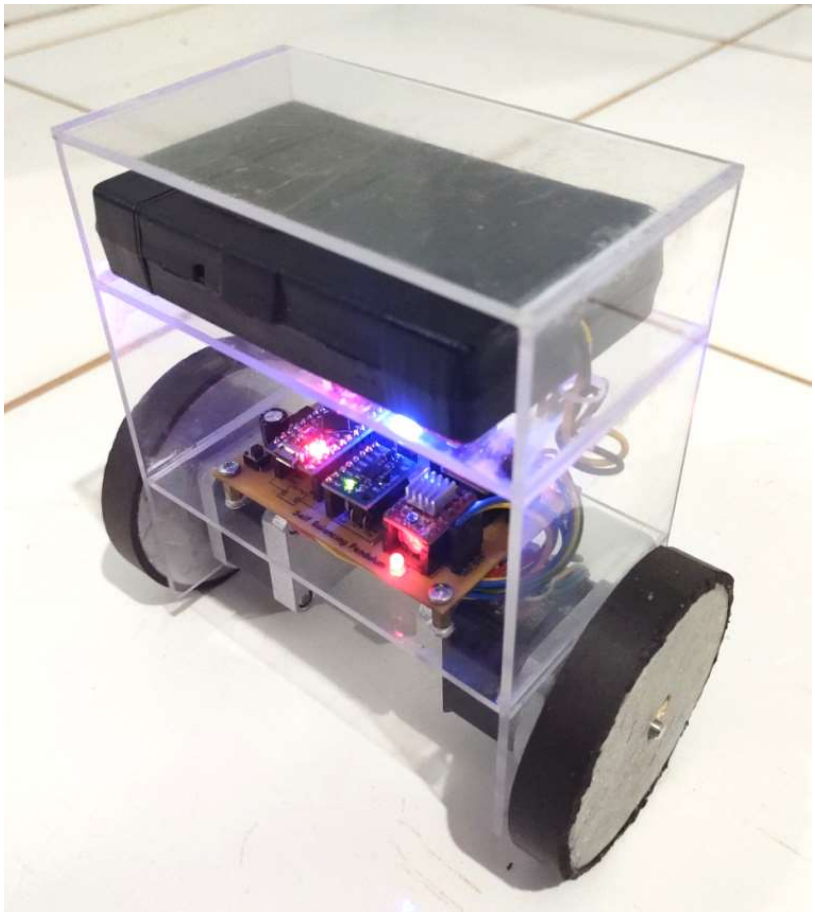
7 RESULTADOS

Os resultados são divididos em duas partes, os resultados relativos à construção do protótipo e os resultados simulados e experimentais.

7.1 Resultados construtivos

O pêndulo foi montado utilizando chapas de acrílico transparente com 3mm de espessura, as partes foram cortadas, e coladas utilizando adesivo instantâneo. As rodas foram cortadas de uma chapa de madeira compensada de 15mm de espessura com raio de 40mm, para melhorar a aderência, as rodas foram encapadas com borracha. A Figura 41 mostra o pêndulo finalizado.

Figura 41 - Protótipo do pêndulo invertido finalizado



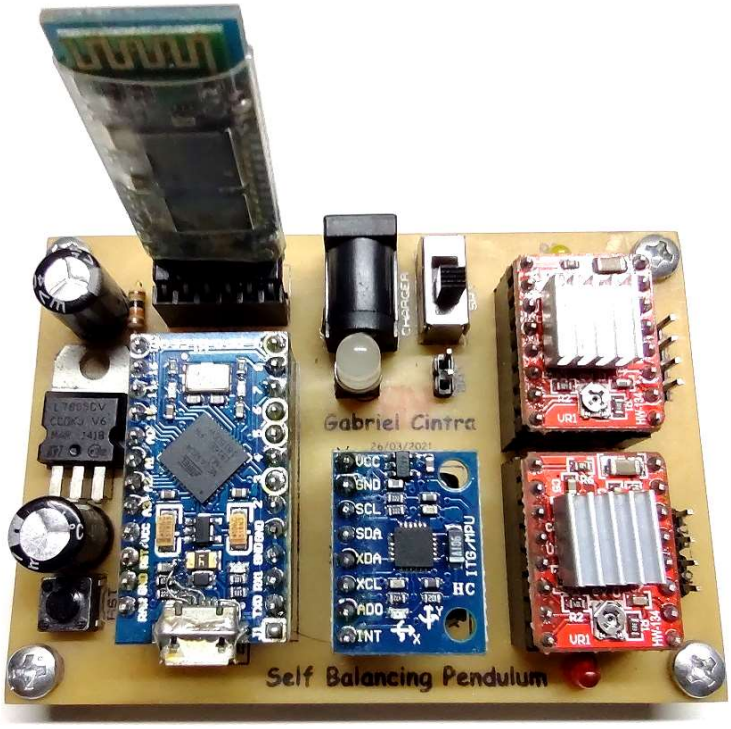
Fonte: Próprio autor

A placa de circuito impresso, mostrada na Figura 42, foi feita de forma artesanal utilizando o método de transferência térmica.

Os componentes da placa são:

- Módulo *Bluetooth* HC-05;
- Placa de desenvolvimento Pro Micro com micontrolador ATmega32U4;
- Módulo Acelerômetro e Giroscópio MPU-6050;
- *Drivers* para motor de passo A4988;
- Regulador de tensão 7805;
- Capacitor eletrolítico;
- Botão de *reset* do microcontrolador;
- LED para indicação do estado do pêndulo;
- Chave liga-desliga;
- Conector para recarga da bateria;
- Conexão para os motores de passo;
- Conexão para bateria.

Figura 42 - Placa de circuito impresso



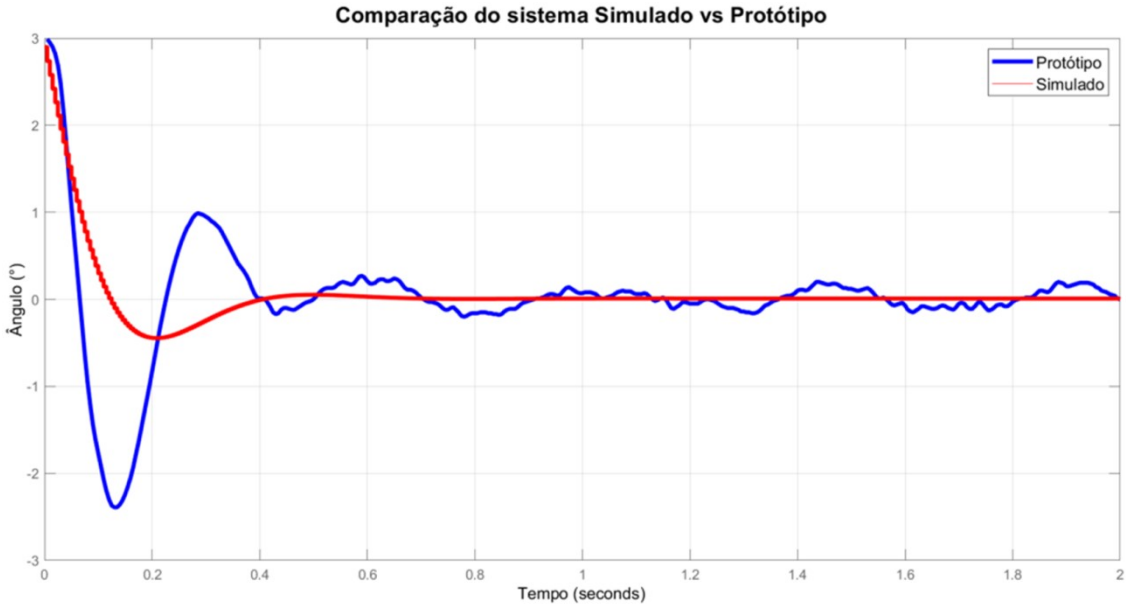
Fonte: Próprio autor

7.2 Resultados simulados e experimentais

Os dados são enviados via porta serial da placa Arduino e capturados utilizando o programa MATLAB, com o código do Apêndice VIII. Após a captura dos dados, estes são plotados em um gráfico, para isso foi utilizando o código do Apêndice VIX, fazendo a comparação dos resultados do sistema simulado e experimental.

Para obtenção destes resultados o pêndulo foi posicionado manualmente até ângulo de inclinação de aproximadamente 3° até que o sistema de controle entrasse em ação fazendo o controle da velocidade dos motores para manter o pêndulo em equilíbrio. A Figura 43, ilustra o comportamento do sistema simulado e do protótipo, em vermelho e azul respectivamente.

Figura 43 - Comparação do sistema Simulado e o Protótipo

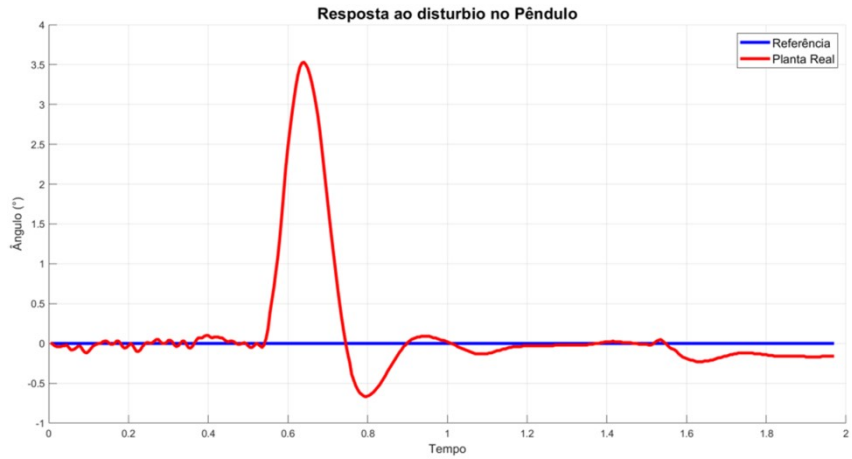


Fonte: Próprio autor

Utilizando o controlador PID projetado, o pêndulo se equilibra com pequenas oscilações e com tempo de acomodação dentro do tempo determinado, entretanto ocorreu um sobre-sinal maior que o projetado, mas logo é estabilizado. A divergência entre os resultados simulados e experimentais pode ser devido à linearização das equações na modelagem matemática, pois foram feitas algumas aproximações de modo a facilitar a modelagem isso pode ocasionar alguns erros nos resultados experimentais. Porém para este projeto estes erros são toleráveis.

Quando o pêndulo está em equilíbrio, ao provocar um distúrbio, ele logo retorna ao equilíbrio. A Figura 44 mostra o comportamento do pêndulo à resposta ao distúrbio.

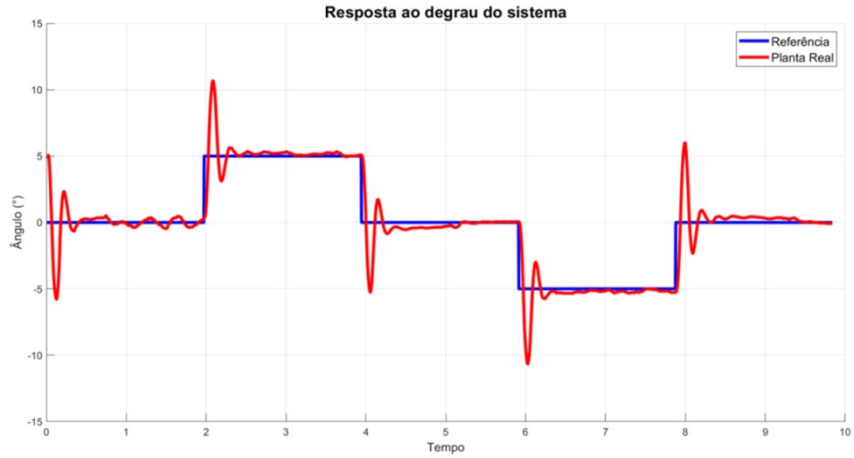
Figura 44 - Resposta ao distúrbio do pêndulo



Fonte: Próprio autor

Para verificar os resultados da mudança de ângulo do pêndulo, ou seja, uma entrada degrau, uma rotina de testes foi programada onde o *setpoint* é alterado a cada 2 segundos. Inicialmente o pêndulo foi posicionado em um ângulo próximo de 5°, logo o sistema de controle é acionado mantendo o pêndulo em equilíbrio, em seguida entra na rotina de testes. O comportamento do sistema à rotina de testes é mostrado na Figura 45.

Figura 45 - Resposta ao degrau do protótipo



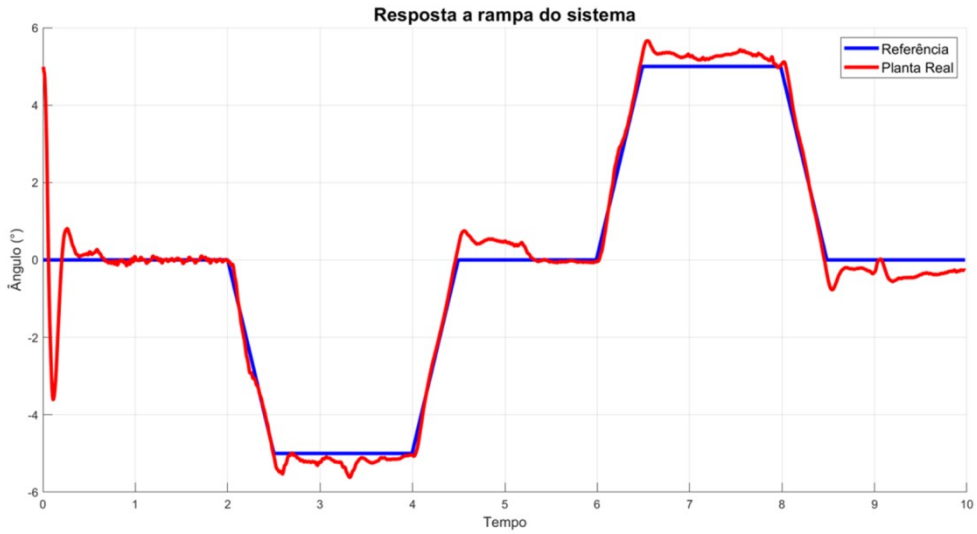
Fonte: Próprio autor

Ao mudar o ângulo de referência o pêndulo responde forma imediata, o problema é que ocorre um sobre-sinal indesejado, que rapidamente é corrigido. Foram feitos ajustes nas constantes do controlador PID para reduzir o sobre-sinal, porém sem melhorias significativas. Ao reduzir o sobre-sinal, o controle não funcionou adequadamente.

Ao aplicar um sinal em rampa, o sobre sinal é reduzido, como ilustrado na Figura 46. Os picos do sinal são em decorrência das imperfeições da superfície do local de teste.

O sinal de referência em azul foi programado de modo análogo ao teste anterior, a diferença é que neste caso o sinal tem uma transição mais suave de um estado para outro, formando uma rampa de subida ou de descida.

Figura 46 – Resposta a rampa do protótipo



Fonte: Próprio autor

Estas alterações no ângulo de inclinação são utilizadas para fazer o pêndulo se deslocar para frente ou para trás. Ajustando o ângulo de inclinação com um valor maior que 0° o pêndulo anda para frente, e mudando o ângulo de inclinação com um valor menor que 0° o pêndulo se desloca para trás. A velocidade do deslocamento é proporcional ao ângulo ajustado, ou seja, quanto maior o ângulo de inclinação, maior é a velocidade de deslocamento do pêndulo. Porém se o ajuste for muito alto o pêndulo pode não ser capaz de se sustentar fazendo-o cair.

8 CONCLUSÃO

O controle do sistema pêndulo invertido apresentou resultados muito satisfatórios. O pêndulo entra em equilíbrio com rapidez e se mantém equilibrado mesmo sob ação de perturbações.

Ao ajustar o ângulo de inclinação do pêndulo um dos lados, o pêndulo tende a se mover para este lado tentando recuperar o equilíbrio, este é um modo de fazer o pêndulo se mover para frente e para trás. Para fazer o pêndulo girar, basta subtrair o sinal de controle de um dos motores e somar ao outro, assim o pêndulo gira, podendo fazer curvas.

Num primeiro momento foram feito testes com motores de corrente contínua com caixa de redução, a implementação do código para controle de velocidade dos motores é mais simples quando comparado ao controle dos motes de passo. Entretanto os motores com caixa de redução apresentam alguns problemas.

O primeiro problema encontrado foi à folga nas engrenagens, o pêndulo ficou muito instável, a resposta do sinal de controle até fazer as rodas girarem demorava, dificultando o controle. Outro problema é devido às características de cada motor, pois os motores não são idênticos, ao enviar o mesmo sinal de controle para ambos motores, um motor girava mais que o outro fazendo o pêndulo andar em circulo ou fazendo curvas.

Por estes motivos o projeto foi refeito utilizando motores de passo, como as rodas são fixadas diretamente no eixo do motor, não existe problema com folga de engrenagens, e como eles funcionam com passos não tem problemas com diferença de velocidade entre os motores.

Ao desenvolver um projeto dessa natureza, obtêm-se um grande aproveitamento das habilidades adquiridas no decorrer da graduação, colocando em prática a teoria estudada.

8.1 Trabalhos futuros

O projeto cumpriu com o objetivo principal de se manter em equilíbrio, mas ainda é possível fazer algumas melhorias:

Utilizar técnicas avançadas de controle como lógica Fuzzy, redes neurais, controle LQR, utilizar sistema MIMO (múltiplas entradas e múltiplas saídas), entre outras.

Instalação de sensores de obstáculos para que o pêndulo possa andar de forma autônoma sem colidir com nenhum objeto.

REFERÊNCIAS

ALLEGRO. **A4988**. 2020. Disponível em: https://www.allegromicro.com/~/media/Files/Datasheets/A4988-Datasheet.ashx. Acesso em: 14 mar. 2021.

ALVES, Rafael Gustavo. **Controle de um pêndulo invertido utilizando técnica de linearização por realimentação**. 2018. 77 f. TCC (Graduação) - Curso de Engenharia de Controle e Automação, Universidade Federal de Ouro Preto, Ouro Preto, 2018.

ATMEL. **ATmega32U4**. Disponível em: https://ww1.microchip.com/downloads/ en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf. Acesso em: 14 mar. 2021.

BUENO, Aline Grotewold; ROMANO, Rodrigo Alvite. **Filtro complementar aplicado a medida de inclinação de plataformas móveis**. 2011. Disponível em: https://maua.br/files/122014/filtro-complementar-aplicado-a-medida-de-inclinacao-de-plataformas-moveis.pdf. Acesso em: 15 mar. 2021.

BRINKEBY, Axel. Mini balancing robot. **Axel's DIY page**, 2017. Disponível em: http://axelsdiy.brinkeby.se/?page_id=1447. Acesso em: 24 mar. 2021.

CARVALHO, Jacinto Luis Azevedo. **Wheelie Plataforma Auto balanceada em duas rodas**. 2014. 141 f. Dissertação (Mestrado) – Controle e Eletrônica Industrial, Instituto Politécnico de Tomar. 2014.

CTMS - Control Tutorials for Matlab & Simulink. **Introduction : PID Controller Design**. Disponível em: https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID#7. Acesso em: 3 nov. 2020.

DORF, Richard C.; BISHOP, Robert H. **Sistemas de controle moderno**. 8. ed. Rio de Janeiro: LTC, 2001.

EEMB. **Li-ion Battery Specification**. Model: LIR18650. 2015. Disponível em: https://www.eemb.com/public/image/download/LIR18650.pdf. Acesso em: 14 mar. 2021.

FRAGA, Guilheme; SILVA, Hugo Bernardino da; FILHO, Marius David Covelli; FREITAS, Renan de; PIRES, Ricardo. **Análise Comparativa de Controladores Aplicados a Sistema de Auto Equilíbrio**. In: Congresso Científico da Semana Tecnológica - IFSP, 5., 2014, Bragança Paulista. Anais do CONCISTEC. Bragança Paulista: IFSP, 2014, 8f.

GADELHA, **Nayana de Freitas. Controle de um pêndulo invertido utilizando lógica fuzzy**. 2018. 75 f. TCC (Graduação) - Curso de Engenharia Mecânica, Universidade Federal Rural do Semi-árido, Caraúbas, Rio Grande do Norte, 2018.

INVENSENSE. Datasheet: **MPU-6000 and MPU-6050 Product Specification Revision 3.4**. 2004. Disponível em: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf. Acesso em 14 mar. 2021.

KANKHUNTHOD, K.; KONGRATANA, V.; NUMSOMRAN, A.; TIPSUWANPORN, V. **Self-balancing Robot Control Using Fractional-Order PID Controller**. Proceedings of the International MultiConference of Engineers and Computer Scientists. 2019. 6f. Hong Kong, 2019.

KUNG, Fabian. **A Tutorial on Modelling and Control of Two-Wheeled Self-Balancing Robot with Stepper Motor**. 2019. Faculty of Engineering, Multimedia University, Cyberjaya, Selangor, Malaysia, 2019.

MATLAB. **The MathWorks - MATLAB & Simulink**. Versão 9.10.0.1602886 (R2021a). Disponível em: https://www.mathworks.com. Acesso em: 22 abr. 2021.

MOTIONKING. **HB Stepper Motor**. 2011. Disponível em: http://www.svaltera.ua/catalogs /knowledge-base/brands/motionking/HB_Stepper_Motor_E.pdf. Acesso em: 14 mar. 2021.

MURTA, Gustavo. Tudo sobre DRIVER A4988 e Motor de Passo > Usando o Arduino. **Eletrogate**, 2018. Disponível em: https://blog.eletrogate.com/driver-a4988-motor-de-passo-usando-o-arduino/. Acesso em: 19, fev. 2021.

NEOMOTION. **Motores de Passo**. Disponível em: https://neomotion.com.br/wp-content/uploads/2020/08/Catálogo-Datasheet-Motores-de-Passo.pdf. Acesso em 19 fev. 2021.

NISE, Norman S. **Engenharia de sistemas de controle**. 6. ed. Rio de Janeiro: LTC, 2013.

OGATA, K. **Engenharia de controle moderno**. 5. ed. São Paulo: Peason Prentice Hall, 2010.

OLIVEIRA, Diego Delgado Colombo de; ROSSI, Leonardo Novak. **Modelagem, controle e prototipação de veículo baseado em Pêndulo Invertido**. 2015. 68 f. TCC (Graduação) - Engenharia Mecatrônica, Escola Politécnica da Universidade de São Paulo, São Paulo, 2015.

OOI, Rich Chi. **Balancing a Two-Wheeled Autonomous Robot**. 2003. 72 f. Tese (Graduação em Engenharia Mecatrônica) – Faculty of Engineering and Mathematical Sciences, The University of Western Australia, Perth, 2003.

PAULA, Adriano Rodrigues de. **Modelagem e controle do pêndulo invertido sobre duas rodas**. 2014. 62 f. TCC (Graduação) - Curso de Engenharia Elétrica, Departamento de Engenharia Elétrica, Universidade Federal do Ceará, Fortaleza, 2014.

SEGWAY. **Segway i2 SE**. 2021. Disponível em: https://www.segway.com/segway-i2-se-pt/. Acesso em: 16 de mar. de 2021.

SCHILING, Darlei Elias. **Modelagem e Controle de um Robô de Eixo Único em Sitema de Pêndulo invertido**. 2017. 53 f. TCC (Graduação) – Curso de Engenharia Elétrica, Universidade Regional do Nordeste do Estado do Rio Grande do Sul, Ijuí, 2017.

SUNDIN, Christian; THORSTENSSON, Filip. **Autonomous balancing robot**: Design and construction of a balancing robot. 2012. 76 f. Department of Signals and Systems - Chalmers University of Technology, Suécia, 2012.

TEIXEIRA, Frederico Santos; SODRÉ, José Luis de Freitas; JUNIOR, Marcos Rangel Junior. **O sistema dinâmico pêndulo invertido**: modelagem e projeto de controladores por simulação computacional. 2006. 68 f. TCC (Graduação) – Curso Superior de Tecnologia em Automação Industrial, Centro Federal de Educação Tecnológica de Campos, Campos dos Goytacazes, 2006.

TREVISANI, Marcelo Duarte. **Projeto de um controlador neuro-fuzzy para um robô de equilíbrio**. 2015. 48 f. Monografia (Graduação) - Curso de Graduação em Engenharia de Controle e Automação, Universidade Federal de Minas Gerais, Belo Horizonte, 2015.

APÊNDICE I – CONTROLADOR PID EM CASCATA

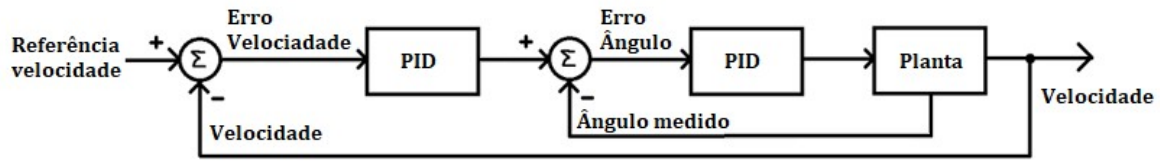
Outro método para fazer o controle do sistema é utilizando dois controladores PID em cascata. Um controlador PID, a malha interna, pega a diferença entre o ângulo desejado e o ângulo atual e calcula a velocidade dos motores. O segundo controlador PID, a malha externa, pega a diferença entre a velocidade do motor e velocidade de referência ajustado pelo usuário e calcula o ângulo de inclinação correto para estabilização do pêndulo (BRINKEBY, 2017).

A vantagem em utilizar este sistema de controle é o que pêndulo retorna a posição original ao sofrer um distúrbio. Também é capaz de fazer o ajuste do ângulo de equilíbrio para fazer a compensação se o centro de gravidade for movido, ou se o pêndulo estiver um plano inclinado (BRINKEBY, 2017).

Para que o pêndulo fique parado basta fazer a velocidade de referência igual a zero. Ao mover o pêndulo para frente, para trás, ou se o centro de gravidade for alterado ao colocar um objeto sobre a plataforma do pêndulo, o termo Integral do controle externo ajustará o ângulo de inclinação a fim de manter a velocidade em zero. Isto acontece devido à integral da velocidade é a posição. O ângulo de inclinação será proporcional à distância do ponto inicial. A componente Proporcional do controle externo faz com que o pendulo sempre se incline ligeiramente no sentido oposto à velocidade. Sem a parte P, o pêndulo sempre se moveria e não pararia. O termo Derivativo não é necessário para este controle (BRINKEBY, 2017).

Portanto a malha de controle externa calcula o ângulo para que o pêndulo fique em equilíbrio e a malha de controle interna calcula a velocidade do motor para manter a estabilidade.

Figura 47 – Controle PID em cascata



Fonte: Adaptada de Brinkeby (2017)

As constantes K_p e K_i do controlador externo foram feitas de forma empírica.

O programa para Arduino do Apêndice XI, já está programado para utilizar o controlador PID em cascata, basta mudar a variável $speedStatus$ para *true*.

APÊNDICE II - CÁLCULO DOS PARÂMETROS E FUNÇÃO DE TRANSFERÊCIA DA PLANTA G(S)

```
% Software para calcular os parâmetros do sistema e a função G(s)
% Autor: Gabriel Cintra
% Engenharia Elétrica - Universidade Federal do Tocantins
% Data: 12/03/2021
% Nome do arquivo: parametros.m

clc
clear all
close all
format short g

% Especificações do sistema
    Ta = 0.5;       % Tempo de acomodação até 1,0s;
    OS = 10;        % Sobre-sinal (overshoot) máximo de 10%;

% Parâmetros físicos do pêndulo
    g  = 9.807;                 % m/s² - Aceleração da gravidade
    mp = 0.850;                 % kg   - Massa do chassi
    H  = 0.160;                 % m    - Altura do chassi
    L  = 0.075;                 % m    - Largura do chassi
    l  = 0.045;                 % m    - Distância do centro de massa até
da centro da roda
    Jp = (mp*(H+L)^2)/12;       % kgm² - Momento de inércia do chassi
    R  = 0.040;                 % m    - Raio das rodas
    mr = 0.116;                 % kg   - Massa da roda
    Jr = (mr*R*R/2);            % kgm² - Moemento de inércia da roda

%para usar s como função da frequência
    s=tf('s');

% Função de transferência da planta
    A = Jp + mp*l^2;
    B = mp*g*l;
    C = mp*l;
    D = mp*R + 2*mr*R + (2*Jr/R);
    E = mp*l*R;

    gs = -(C+D)*R*s / ((A-E)*s^2 - B)
```

APÊNDICE III – LUGAR DAS RAÍZES DO SISTEMA EM MALHA ABERTA E MALHA FECHADA SEM COMPENSAÇÃO

```
% Software para verificar o comportamento do sistema e plotar os gráficos
% do lugar das raízes e das reposta ao degrau e degrau em malha aberta e
% malha fechada do sistema sem compensação.
% Autor: Gabriel Cintra
% Engenharia Elétrica - Universidade Federal do Tocantins
% Data: 12/03/2021
% Nome do arquivo: sistemaSemControlador.m

parametros

% 1 - Verificar o sistema sem compensação
% 1.1 - Sistema em malha aberta
% 1.1.1 - Localização dos Polos e Zeros em Malha Aberta
    figure
    pzmap(gs)
    change(12, 2, 1);
    axis([-15 15 -20 20])
    title('Localização dos Polos e Zeros em Malha Aberta', 'FontSize', 16);
    xlabel('Parte Real', 'FontSize', 12)
    ylabel('Parte Imaginaria', 'FontSize', 12)
% 1.1.2 - Resposta ao degrau em malha aberta
    figure
    step(gs, 'r');
    axis([0 1 -90 1])
    change(12, 2, 1);
    title('Resposta ao degrau em Malha Aberta', 'FontSize', 16)
    ylabel('Ângulo[°]', 'FontSize', 14)
    xlabel('Tempo[s]', 'FontSize', 14)
    legend('Ângulo', 'FontSize', 12)
% 1.2 - Sistema em malha fechada
% 1.2.1 - Lugar das raízes em malha fechada
    figure
    gsmf = feedback(gs, 1)
    rlocus(gsmf)
    change(12, 2, 1);
    axis([-15 15 -20 20])
    title('Lugar das raízes em Malha Fechada', 'FontSize', 16);
    xlabel('Parte Real', 'FontSize', 12)
    ylabel('Parte Imaginaria', 'FontSize', 12)
% 1.2.2 - Resposta ao degrau em malha fechada
    figure
    impulse(gsmf, 'r');
    axis([0 1 -90 1])
    change(12, 2, 1);
    title('Resposta ao degrau em Malha Fechada', 'FontSize', 16)
    ylabel('Ângulo[°]', 'FontSize', 14)
    xlabel('Tempo[s]', 'FontSize', 14)
    legend('Ângulo', 'FontSize', 12)
```

APÊNDICE IV – PROJETO DO CONTROLADOR PID

```
% Software para calcular o controlador PID
% Primeiro é feito o projeto do controlador PD, em seguida o controlador PI
% Com a combinação dos controladores PD e PI é formado o controlador PID
% Autor: Gabriel Cintra
% Engenharia Elétrica - Universidade Federal do Tocantins
% Data: 12/03/2021
% Nome do arquivo: projetoCompensadorPID.m

parametros

% 1 - Calculo do pólo de projeto
% 1.1 - Calcula o fator de amortecimento.
    csi = ( -log(OS/100) / (sqrt(pi^2 + (log(OS/100))^2)))
% 1.2 -  Calcula a frequência natural.
    wn = 4 / (csi * Ta)
% 1.3 - % Calcula a posição do polo dominante desejado.
    Pd = (-csi*wn) + (wn*sqrt(1-csi^2)*i)

% 1.4 - Localização dos polos e zeros em Malha Aberta com marcação do pólo
de projeto
    figure
    poloProjeto(OS, Ta);
    pzmap(gs);
    change(12, 2, 1);
    axis([-11 11 -15 15]);
    title(['Localização dos polos e zeros em Malha Aberta com marcação do
pólo de projeto, UP = ', num2str(OS), '%, Ta = ', num2str(Ta), 's'],
'FontSize', 16)
    legend(['Reta para ', num2str(OS), '% de Overshoot'], 'Polo desejado
para Compensação', 'FontSize', 12)
    xlabel('Parte Real', 'FontSize', 14)
    ylabel('Parte Imaginaria', 'FontSize', 14)

% 2 - Projetar o controlador PD para atender as especificações da resposta
%transitória adicionando um zero ao sistema.
% 2.1 - Cálculo do local do zero do controlador PD
    zcd = zeroCompensador(gs, Pd)
% 2.2 - Lugar das raízes em Malha Fechada com controlador PD
    figure
    poloProjeto(OS, Ta);
    gpd = (s + zcd);
    gspd = gs*gpd;
    rlocus(feedback(gspd, 1));
    change(12, 2, 1);
    axis([-50 50 -15 15])
    legend(['Reta para ', num2str(OS), '% de Overshoot'], 'Pólo desejado
para Compensação', 'FontSize', 12)
    title(['Lugar das raízes em Malha Fechada com compensação PD, UP = ',
num2str(OS), '%, Ta = ', num2str(Ta), 's'], 'FontSize', 16)
    xlabel('Parte Real', 'FontSize', 14)
    ylabel('Parte Imaginaria', 'FontSize', 14)
    Kcd = abs(1/((evalfr(gpd, Pd) * evalfr(gs, Pd))))
```

```
% 2.3 - Lugar das raízes em Malha Fechada com controlador PD com ganho Kd
    figure
    poloProjeto(OS, Ta);
    gspdmf = feedback(Kcd*gspd, 1);
    rlocus(gspdmf)
    change(12, 2, 1);
    axis([-11 11 -15 15])
    legend(['Reta para ', num2str(OS), '% de Overshoot'], 'Pólo desejado
para Compensação', 'FontSize', 12)
    title(['Lugar das raízes em Malha Fechada com compensação PD e ganho
Kd, UP = ', num2str(OS), '%, Ta = ', num2str(Ta), 's'], 'FontSize', 16)
    xlabel('Parte Real', 'FontSize', 14)
    ylabel('Parte Imaginaria', 'FontSize', 14)
% 2.4 - Resposta ao degrau do sistema em Malha Fechada com compensação PD
    figure
    step(gspdmf, 'r')
    axis([0 1 -1 3])
    change(12, 2, 1);
    title(['Resposta ao degrau em Malha Fechada com compensação PD. UP = ',
num2str(OS), '%, Ta = ', num2str(Ta), 's'], 'FontSize', 16)
    ylabel('Ângulo[°]', 'FontSize', 14)
    xlabel('Tempo[s]', 'FontSize', 14)
    legend('Ângulo', 'FontSize', 12)

% 3 - Projetar o controlador PI para produzir o erro de estado estacionário
%zero adicionando um Pólo na origem e um Zero próximo a este pólo.
% 3.1 - Cálculo do local do zero do controlador PI, incluindo o pólo na
origem
    zci = zeroCompensador(gspd*(1/s), Pd)
    zci = 0.1
% 3.2 - lugar das raízes em Malha Fechada com compensação PI
    figure
    poloProjeto(OS, Ta);
    gpi = (s + zci) / s
    gspid = gpi*gspd
    rlocus(feedback(gspid, 1));
    change(12, 2, 1);
    axis([-50 50 -15 15])
    legend(['Reta para ', num2str(OS), '% de Overshoot'], 'Pólo desejado
para Compensação', 'FontSize', 12)
    title(['Lugar das raízes em Malha Fechada com compensação PI, UP = ',
num2str(OS), '%, Ta = ', num2str(Ta), 's'], 'FontSize', 16)
    xlabel('Parte Real', 'FontSize', 14)
    ylabel('Parte Imaginaria', 'FontSize', 14)
    K = abs(1/((evalfr(gpi, Pd) * evalfr(gpd, Pd) * evalfr(gs, Pd))))

% 3.3 - lugar das raízes em Malha Fechada com compensação PI com ganho Ki
    figure
    poloProjeto(OS, Ta);
    gspidmf = feedback(K*gspid, 1);
    rlocus(gspidmf)
    change(12, 2, 1);
    axis([-11 11 -15 15])
    legend(['Reta para ', num2str(OS), '% de Overshoot'], 'Pólo desejado
para Compensação', 'FontSize', 12)
    title(['Lugar das raízes em Malha Fechada com compensação PI  com ganho
Ki, UP = ', num2str(OS), '%, Ta = ', num2str(Ta), 's'], 'FontSize', 16)
    xlabel('Parte Real', 'FontSize', 14)
    ylabel('Parte Imaginaria', 'FontSize', 14)
```

```
% 3.4 - Resposta ao degrau do sistema em Malha Fechada com compensação PID
    figure
    step(gspidmf, 'r')
    axis([0 1 -1 3])
    change(12, 2, 1);
    title(['Resposta ao degrau em Malha Fechada compensação PI. UP = ',
num2str(OS), '%, Ta = ', num2str(Ta), 's'], 'FontSize', 16)
    ylabel('Ângulo[°]', 'FontSize', 14)
    xlabel('Tempo[s]', 'FontSize', 14)
    legend('Ângulo', 'FontSize', 12)

% 4 - Cálculo das constantes do controlador PID de acordo com o ganho
obtido
% no gráfico do lugar das raízes do passo anterior.
% O ganho encontrado foi K = 2.7949, substituindo na equação do PID e após
% aglumas manipulações matemáticas as constantes Kp, Ki e Kd são dadas por:
%   gpid = K*(s + zpd)*(s + zpi) / s
%   gpid = ( Kd*s^2 + Kp*s + Ki ) / s
    gpid = K*(s + zci)*(s + zcd) / s
    [num,den] = tfdata(gpid);
    Kp = num{1}(2)
    Ki = num{1}(3)
    Kd = num{1}(1)
% 4.1 - lugar das raízes em Malha Fechada com compensação PID
    figure
    poloProjeto(OS, Ta);
    gpidgs = feedback(gpid*gs, 1);
    rlocus(gpidgs)
    change(12, 2, 1);
    axis([-11 11 -15 15])
    legend(['Reta para ', num2str(OS), '% de Overshoot'], 'Polo desejado
para Compensação', 'FontSize', 12)
    title(['Lugar das raízes em Malha Fechada com compensação PID, UP = ',
num2str(OS), '%, Ta = ', num2str(Ta), 's'], 'FontSize', 16)
    xlabel('Parte Real', 'FontSize', 14)
    ylabel('Parte Imaginaria', 'FontSize', 14)
% 4.2 - Resposta ao degrau do sistema em Malha Fechada com compensação PID
    figure
    step(gpidgs, 'r')
    axis([0 1 -1 3])
    change(12, 2, 1);
    title(['Resposta ao degrau em Malha Fechada com compensação PID. UP =
', num2str(OS), '%, Ta = ', num2str(Ta), 's'], 'FontSize', 16)
    ylabel('Ângulo[°]', 'FontSize', 14)
    xlabel('Tempo[s]', 'FontSize', 14)
    legend('Ângulo', 'FontSize', 12)
```

APÊNDICE V – FUNÇÃO - AUMENTAR MARCAÇÃO DOS POLOS E ZEROS

```
% Função para aumentar o tamanho e a largura das linhas dos gráficos
% Entrada: markersize - tamanho do marcador, linewidth - largura da linha,
% grade - habilita ou desabilita a grade
% Autor: Gabriel Cintra
% Engenharia Elétrica - Universidade Federal do Tocantins
% Data: 12/03/2021
% Nome do arquivo: change.m

function change(markersize, linewidth, grade)

    a = findobj(gca,'type','line');

    for i = 1:length(a)
        set(a(i), 'markersize', markersize); %change marker size
        set(a(i), 'linewidth', linewidth);  %change linewidth
    end

    if grade == 1
        grid on
    else
        grid off
    end

end
```

APÊNDICE VI – FUNÇÂO - MARCAR POLO DE PROJETO

```
% Função para marcar o pólo desejado no gráfico do lugar das raízes
% Entrada: OS - Overshoot, Ta - Tempo de acomodação
% Autor: Gabriel Cintra
% Engenharia Elétrica - Universidade Federal do Tocantins
% Data: 12/03/2021
% Nome do arquivo: poloProjeto.m

function poloProjeto(OS, Ta)

    % Calcula o fator de amortecimento.
        csi = ( -log(OS/100) / (sqrt(pi^2 + (log(OS/100))^2)));
    % Calcula a frequência natural.
        wn = 4 / (csi * Ta);
    % Calcula a posição do polo dominante desejado.
        Pd = (-csi*wn) + (wn*sqrt(1-csi^2)*i);

    x = linspace(0,-real(Pd));
    y = tan(acos(csi))*x;
    plot(-x,y,'r');
    hold on
    plot(real(Pd),imag(Pd),'k*');
    plot(-x,-y,'r');
    plot(real(Pd),-imag(Pd),'k*');

end
```

APÊNDICE VII – FUNÇÃO - CÁLCULO DO ZERO DO COMPENSADOR

```
% Função para calcular o local onde inserir um zero do compensador PD ou PI
% Entrada: FT - Função de Transferência, Pd - Pólo desejado
% Autor: Gabriel Cintra
% Engenharia Elétrica - Universidade Federal do Tocantins
% Data: 12/03/2021
% Nome do arquivo: zeroCompensador.m

function [zc] = zeroCompensador(TF, Pd)

    p = pole(TF);
    z = zero(TF);

    thetaPolos = zeros(size(p));
    thetaZeros = zeros(size(z));
    thetaPolosSum = 0;
    thetaZerosSum = 0;

    for n = 1:length(p)
        if real(Pd) < p(n)
            thetaPolos(n) = abs(atand(imag(Pd)/abs(real(Pd)-p(n)))-180)
        else
            thetaPolos(n) = abs(atand(imag(Pd)/abs(real(Pd)-p(n))))
        end
        thetaPolosSum = thetaPolosSum + thetaPolos(n);
    end

    for n = 1:length(z)
        if real(Pd) < z(n)
            thetaZeros(n) = abs(atand(imag(Pd)/abs(real(Pd)-z(n)))-180)
        else
            thetaZeros(n) = abs(atand(imag(Pd)/abs(real(Pd)-z(n))))
        end
        thetaZerosSum = thetaZerosSum + thetaZeros(n);
    end

    thetaZero = thetaPolosSum - thetaZerosSum

    zc = ( imag(Pd) / tand(thetaZero) ) + abs(real(Pd));

end
```

APÊNDICE VIII – CAPTURA DOS DADOS VIA COMUNICAÇÃO SERIAL

```
% Software para receber os dados do Arduino via comunicação serial.
% Autor: Gabriel Cintra
% Engenharia Elétrica - Universidade Federal do Tocantins
% Data: 15/03/2021
% Nome do arquivo: receiveDataArduino.m

if ~isempty (instrfind)
  fclose (instrfind);
  delete (instrfind);
end

serialArduino = serial('COM7');

set(serialArduino,'BaudRate',500000);
set(serialArduino,'DataBits',8);
set(serialArduino,'Parity','none');
set(serialArduino,'StopBits',1);
set(serialArduino,'Terminator','LF');

set(serialArduino,'InputBufferSize',5000);

fopen(serialArduino)    %Abre a porta Serial

fprintf('Pronto para receber dados.\n')
i=101;
while (serialArduino.BytesAvailable == 0)
  pause(0.1);
  i = i-1;
  if mod(i, 10) == 0
    fprintf('Aguardando...%d\n', i/10)
  end
  if i == 0
    break;
  end
end

i=0;
currentAngle = 0;
targetAngle = 0;
while (serialArduino.BytesAvailable > 0)
  pause(0.005);
  i=i+1;
  currentAngle(i) = fscanf(serialArduino, '%f');
  targetAngle(i) = fscanf(serialArduino, '%f');
end

fprintf('Quantidade de leituras, currentAngle: %d\n', length(currentAngle))
fprintf('Quantidade de leituras, targetAngle: %d\n', length(targetAngle))

%Encerra conexão
fclose(serialArduino)
delete(serialArduino)
clear serialArduino
```

ANPÊNDICE IX – COMPARAÇÃO DOS RESULTADOS: SIMULAÇÃO VS PLANTA REAL

```
% Software para comparar os resultados obtidos através da simulação com as
% funções de transferência e os resultados práticos do sistema real.
% Autor: Gabriel Cintra
% Engenharia Elétrica - Universidade Federal do Tocantins
% Data: 15/03/2021
% Nome do arquivo: SimuladoVsArduino.m

clc
close all

if ~exist('gpid', 'var')
    projetoCompensadorSemGraficos
end

receiveDataArduino

Ts = 0.005;

gsD = c2d(gs,Ts,'tustin');

gpidD = c2d(gpid,Ts,'tustin');

figure
if currentAngle(1) < 0
    plot((1:length(currentAngle))*Ts,-currentAngle, 'b');
else
    plot((1:length(currentAngle))*Ts,currentAngle, 'b');
end
hold on
step(feedback(gpidD*gsD, 1),length(currentAngle)*Ts, 'r')
legend('Protótipo', 'Simulado', 'FontSize', 12)
title('Comparação do sistema Simulado vs Protótipo', 'FontSize', 16);
ylabel('Ângulo (°)', 'FontSize', 12)
xlabel('Tempo', 'FontSize', 12)
change(12, 3, 1);
```

APÊNDICE X - PROJETO DO CONTROLADOR PID (SEM GRÁFICOS)

```
% Software para calcular o controlador PID
% Primeiro é feito o projeto do controlador PD, em seguida o controlador PI
% Com a combinação dos controladores PD e PI é formado o controlador PID
% Autor: Gabriel Cintra
% Engenharia Elétrica - Universidade Federal do Tocantins
% Data: 12/03/2021
% Nome do arquivo: projetoCompensadorSemGraficos.m

parametros

% 1 - Calculo do pólo de projeto
% 1.1 - Calcula o fator de amortecimento.
    csi = ( -log(OS/100) / (sqrt(pi^2 + (log(OS/100))^2)))
% 1.2 -  Calcula a frequência natural.
    wn = 4 / (csi * Ta)
% 1.3 - % Calcula a posição do polo dominante desejado.
    Pd = (-csi*wn) + (wn*sqrt(1-csi^2)*i)

% 2 - Projetar o controlador PD para atender as especificações da resposta
%transitória adicionando um zero ao sistema.
% 2.1 - Cálculo do local do zero do controlador PD
    zcd = zeroCompensador(gs, Pd)
% 2.2 - Função de Transfeência do controlador PD
    gpd = (s + zcd)
% 2.3 - Cálculo do ganho em malha fechada do controlador PD
    Kcd = abs(1/((evalfr(gpd, Pd) * evalfr(gs, Pd))))

% 3 - Projetar o controlador PI para produzir o erro de estado estacionário
%zero adicionando um Pólo na origem e um Zero próximo a este pólo.
% 3.1 - Cálculo do local do zero do controlador PI, incluindo o pólo na
origem
    zci = zeroCompensador(gpd*gs*(1/s), Pd)
    zci = 0.1
% 3.2 - Função de Transfeência do controlador PI
    gpi = (s + zci) / s
% 3.3 - Cálculo do ganho em malha fechada do controlador PD
    K = abs(1/((evalfr(gpi, Pd) * evalfr(gpd, Pd) * evalfr(gs, Pd))))

% 4 - Cálculo das constantes do controlador PID de acordo com o ganho
obtido
% no gráfico do lugar das raízes do passo anterior.
% O ganho encontrado foi K = 2.7949, substituindo na equação do PID e após
% aglumas manipulações matemáticas as constantes Kp, Ki e Kd são dadas por:
%   gpid = K*(s + zpd)*(s + zpi) / s
%   gpid = ( Kd*s^2 + Kp*s + Ki ) / s
    gpid = K*(s + zci)*(s + zcd) / s
    [num,den] = tfdata(gpid);
    Kp = num{1}(2)
    Ki = num{1}(3)
    Kd = num{1}(1)
```

APÊNDICE XI – SOFTWARE ARDUINO PARA CONTROLE DO PÊNDULO INVERTIDO

```
#include <EEPROM.h>
#include <Wire.h>
#include "mpu6050.h"
#include "dPID.h"
#include "rampa.h"

// PINOUT
#define DM1   PORTD, 4    // DIR  Motor1: D4 -> PORTD, 4
#define SM1   PORTC, 6    // STEP Motor1: D5 -> PORTC, 6
#define EM1   PORTD, 7    // ENA  Motor1: D6 -> PORTD, 7
#define DM2   PORTE, 6    // DIR  Motor2: D7 -> PORTE, 6
#define SM2   PORTB, 4    // STEP Motor2: D8 -> PORTB, 4
#define EM2   PORTB, 5    // ENA  Motor2: D9 -> PORTB, 5
#define LED   PORTB, 6    // LED: D4 ---------> PORTB, 6
#define VOL   PORTF, 7    // VOL: A0 ---------> PORTF, 7

// AUX definitions
#define _outputLow(port, pin)   ( port &= ~(1<<pin) )
#define _outputHigh(port, pin) ( port |=  (1<<pin) )
#define outputLow(x)  _outputLow(x)
#define outputHigh(x) _outputHigh(x)

#define _setInput(port, pin)   ( *(&port - 1) &= ~(1<<pin) )
#define _setOutput(port, pin) ( *(&port - 1) |=  (1<<pin) )
#define setInput(x)  _setInput(x)
#define setOutput(x) _setOutput(x)

#define STOP      0
#define BACKWARD -1
#define FORWARD   1
#define LEFT      1
#define RIGHT    -1

//TIMERS
unsigned long sampleTime = 5000;  // unit: microseconds
unsigned long currentTime = 0;    // unit: microseconds
unsigned long previousTime = 0;   // unit: microseconds
float dt = 0;                     // unit: seconds

//MOTORES
#define MAX_SPEED  300
#define MAX_ACCEL  10
volatile int32_t steps1 = 0, steps2 = 0;
float previousSpeed = 0, currentSpeed = 0;
float targetSpeed = 0, estimatedSpeed = 0;
float speedMotor1 = 0, speedMotor2 = 0;
float motor1 = 0, motor2 = 0;
int  dir_M1 = 0, dir_M2 = 0;
float speedFiltered = 0;
float controlOutput = 0;
float steering = 0;
float throttle = 0;
```

```
//MPU 6050
//#define CALIBRATE
#define MAX_ANG 40
Mpu6050 imu = Mpu6050(0x68);
int16_t ax = 0, ay = 0, az = 0, gx = 0, gy = 0, gz = 0;
float previousAngle = 0, currentAngle = 0;
float angularVelocity = 0;
float targetAngle = 0;
float angleOffset = 0;
float angle = 0;

//PID
float stabilityKp stabilityKp = 12.0;
float stabilityKi stabilityKi = 1.00;
float stabilityKd stabilityKd = 2.00;
const float speedLimit = 300.0;
float speedKp speedKp  = 0.300;
float speedKi speedKi  = 0.002;
float speedKd speedKd  = 0.000;
const float angleLimit = 30.00;

dPID speedPID(speedKp, speedKi, speedKd, -angleLimit, angleLimit);
dPID stabilityPID(stabilityKp, stabilityKi, stabilityKd, -speedLimit,
speedLimit);

ramp velocidade(0.25, 0.5, 100.0);
ramp giro(0.25, 0.5, 50.0);
ramp angulo(0.05, 0.1, 5.0);

bool stabilityStatus = true;
bool speedStatus = true;
bool balancing = false;

void setup(){

  // configuração dos pinos
  setOutput(DM1);             // DIR  Motor1: D4 -> PORTD, 4
  setOutput(SM1);             // STEP Motor1: D5 -> PORTC, 6
  setOutput(EM1);             // ENA  Motor1: D6 -> PORTD, 7
  setOutput(DM2);             // DIR  Motor2: D7 -> PORTE, 6
  setOutput(SM2);             // STEP Motor2: D8 -> PORTB, 4
  setOutput(EM2);             // ENA  Motor2: D9 -> PORTB, 5
  setOutput(LED);             // LED: D4 ---------> PORTB, 6
  setInput(VOL);              // VOL: A0 ---------> PORTF, 7

  outputHigh(EM1);            // Desable Motor 1
  outputHigh(EM2);            // Desable Motor 2
  outputLow(LED);             // Desable LED

  delay(1000);

  Serial.begin(500000);
  //while(!Serial);
  Serial1.begin(115200);
  Serial.print("Initializing...");
  Wire.begin();                   // Start the I2C bus as master
  Wire.setClock(400000);          // 400000 =  400kHz I2C clock.
  // initialize device
  imu.init();
  Serial.println(F("Initializing MPU6050"));
```

```
  // verify connection
  Serial.println(F("Testing device connections..."));
  Serial.println(imu.testConnection() ? F("MPU6050 connection successful")
: F("MPU6050 connection failed"));
  delay(1000);

#ifdef CALIBRATE
  // MPU-6050 calibration
  Serial.println(F("MPU-6050 calibration"));
  imu.calibration();
#else
  // calibration offsets for MPU6050
  imu.setXAccelOffset(-49);
  imu.setYAccelOffset(-4097);
  imu.setZAccelOffset(1357);
  imu.setXGyroOffset(-81);
  imu.setYGyroOffset(103);
  imu.setZGyroOffset(-10);
#endif

  // initial angle
  angle = imu.setInitialAngle();
  Serial.print(F("setInitialAngle: "));
  Serial.println(angle);

  // TIMER1 controls motor 1 (left)
  TCCR1A = 0;                           // Timer1 CTC mode 4
  TCCR1B = (1 << WGM12) | (1 << CS11); // Prescaler=8, => 2Mhz
  OCR1A = 65535;                        // longest period, motor stopped
  TCNT1 = 0;

  // TIMER3 controls motor 2 (right)
  TCCR3A = 0;                           // Timer1 CTC mode 4
  TCCR3B = (1 << WGM12) | (1 << CS11); // Prescaler=8, => 2Mhz
  OCR3A = 65535;                        // longest period, motor stopped
  TCNT3 = 0;

  Serial.println("Done. \nStarting main loop...");

  for (int i=0; i<5; i++){
    outputHigh(LED);
    delay(100);
    outputLow(LED);
    delay(100);
  }

  // Enable TIMERs interrupts
  TIMSK1 |= (1 << OCIE1A);    // Enable TIMER1 interrupt
  TIMSK3 |= (1 << OCIE3A);    // Enable TIMER3 interrupt

  readEEprom();

  previousTime = micros();

}//setup
```

```
void loop(){

  while ( sampleTime > ( (currentTime = micros()) - previousTime ) );
  dt = (currentTime - previousTime) * 0.000001;
  previousTime = currentTime;

  previousAngle = currentAngle;
  currentAngle = computeAngle(dt) + angleOffset;

  currentSpeed = (speedMotor1 + speedMotor2) / 2.0;

  angularVelocity = (currentAngle - previousAngle) / dt;
  estimatedSpeed = -currentSpeed + angularVelocity;
  speedFiltered = 0.99*speedFiltered + 0.01*estimatedSpeed;

  // Começa se equilibrar se o pêndulo estiver perto do equilíbrio.
  if ( !balancing && abs(currentAngle) < 5 && stabilityStatus ){
    balancing = true;
    speedPID.resetPID();
    stabilityPID.resetPID();
    outputHigh(LED);                    // LED ON
  }

  if (balancing){
    if(speedStatus){
     // Controle de Velocidade: Utilizando um controlador PI
     //   Entrada: Velocidade desejada
     //   Variável: Velocidade atual do pêndulo
     //   Saída: Ângulo do pêndulo para obter a velocidade desejada
     throttle = velocidade.setpoint();
     targetAngle = speedPID.updatePID(throttle, speedFiltered, dt);
    } else{
     speedPID.resetPID();
     targetAngle = angulo.setpoint();
    }

    if(stabilityStatus){
     // Controle de Estabilidade
     //   Entrada: Ângulo desejado (obtido do Controle de Velocidade)
     //   Variável: Ângulo atual do pêndulo
     //   Saída: Velocidade do motor
     controlOutput = stabilityPID.updatePID(targetAngle, currentAngle, dt);
    } else{
     controlOutput = 0;
     stabilityPID.resetPID();
    }
    // The steering part from the user is injected directly to the output
    steering = giro.setpoint();
    motor1 = controlOutput + steering;
    motor2 = controlOutput - steering;
    // Limit max speed (control output)
    motor1 = constrain(motor1, -MAX_SPEED, MAX_SPEED);
    motor2 = constrain(motor2, -MAX_SPEED, MAX_SPEED);
    // Send the commands to the motors
    setSpeedMotor1(motor1);
    setSpeedMotor2(motor2);
  }
```

```
  // Stop balancing if the robot is leaning to much in any direction.
  if ( abs(currentAngle) > 40 || !stabilityStatus){
    outputLow(LED);                 // LED OFF
    balancing = false;
    targetAngle = 0;
    controlOutput = 0;
    setSpeedMotor1(0); setSpeedMotor2(0);
    steps1 = 0; steps2 = 0;
    throttle = 0;
    steering = 0;
  }

  Serial.print(F("targetAngle:"));
  Serial.print(targetAngle);
  Serial.print(F("\t"));
  Serial.print(F("currentAngle:"));
  Serial.print(currentAngle);
  Serial.println();

  serialData();

}//loop

float readVoltage(){

  float R1 = 10000.0;
  float R2 = 5600.0;
  float value = 0;

  //for(int i=0; i<10; i++)
    value += analogRead(A0);
  //value /= 10;

  float vout = ((value*5.0)/1023.0)/(R2/(R1+R2));

  return vout ;

}//readVoltage

float computeAngle(float dt){

  imu.getMotion(&ax, &ay, &az, &gx, &gy, &gz);

  float roll = atan2(ay, az) * RAD_TO_DEG;

  float complementGyro = 1.0f - dt;
  float complementAcc = dt;

  angle = complementGyro*(angle + gx/131.0 * dt) + complementAcc*roll;

  return angle;

}//computeAngle
```

```
void delay_1us(){
    __asm__ __volatile__ (
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop" "\n\t"
    "nop");
}//delay_1us

// TIMER 1 compare interrupt for driving motor 1
ISR(TIMER1_COMPA_vect){

  if (dir_M1 == 0) // If we are not moving we dont generate a pulse
    return;
  // We generate 1us STEP pulse
  outputHigh(SM1); // STEP MOTOR 1
  delay_1us();
  if(dir_M1>0)
    steps1++;
  else
    steps1--;
  outputLow(SM1);

}//TIMER1_COMPA_vect

// TIMER 3 compare interrupt for driving motor 2
ISR(TIMER3_COMPA_vect){

  if (dir_M2 == 0) // If we are not moving we dont generate a pulse
    return;
  // We generate 1us STEP pulse
  outputHigh(SM2); // STEP MOTOR 2
  delay_1us();
  if(dir_M2>0)
    steps2++;
  else
    steps2--;
  outputLow(SM2);

}//TIMER3_COMPA_vect

void setSpeedMotor1(float tspeed){

  long timer_period;
  // LIMIT MAX ACCELERATION of the motors
  if ((speedMotor1 - tspeed) > MAX_ACCEL)
    speedMotor1 -= MAX_ACCEL;
  else if ((speedMotor1 - tspeed) < -MAX_ACCEL)
    speedMotor1 += MAX_ACCEL;
```

```
  else
    speedMotor1 = tspeed;
  if ( speedMotor1 == 0 ){
    timer_period = 65535;
    dir_M1 = 0;
    outputHigh(EM1);                  // Desabilita Motor 1
  } else if ( tspeed > 0 ){
    timer_period = 2000000.0 / ( speedMotor1 * 25.0 );
    dir_M1 = 1;
    outputLow(EM1);                   // Habilita Motor 1
    outputLow(DM1);                   // Direção Motor 1 (Avante)
  } else{
    timer_period = 2000000.0 / ( -speedMotor1 * 25.0 );
    dir_M1 = -1;
    outputLow(EM1);                   // Habilita Motor 1
    outputHigh(DM1);                  // Direção Motor 1 (Reverso)
  }
  if (timer_period > 65535) // Check for maximun period without overflow
    timer_period = 65535;
  OCR1A = timer_period;
  if (TCNT1 > OCR1A)         // Check  if we need to reset the timer...
    TCNT1 = 0;
}//setSpeedMotor1

void setSpeedMotor2(float tspeed){

  long timer_period;
  // LIMIT MAX ACCELERATION of the motors
  if ((speedMotor2 - tspeed) > MAX_ACCEL)
    speedMotor2 -= MAX_ACCEL;
  else if ((speedMotor2 - tspeed) < -MAX_ACCEL)
    speedMotor2 += MAX_ACCEL;
  else
    speedMotor2 = tspeed;
  if ( speedMotor2 == 0 ){
    timer_period = 65535;
    dir_M2 = 0;
    outputHigh(EM2);                  // Desabilita Motor 2
  } else if ( speedMotor2 > 0 ){
    timer_period = 2000000.0 / ( speedMotor2 * 25.0 );
    dir_M2 = 1;
    outputLow(EM2);                   // Habilita Motor 2
    outputHigh(DM2);                  // Direção Motor 2 (Avante)
  } else{
    timer_period = 2000000.0 / ( -speedMotor2 * 25.0 );
    dir_M2 = -1;
    outputLow(EM2);                   // Habilita Motor 2
    outputLow(DM2);                   // Direção Motor 2 (Reverso)
  }
  if (timer_period > 65535) // Check for maximun period without overflow
    timer_period = 65535;
  OCR3A = timer_period;
  if (TCNT3 > OCR3A)         // Check  if we need to reset the timer...
    TCNT3 = 0;
}//setSpeedMotor2
```

```
void serialData(){

  bool refresh = false;
  char command = ' ';

  if (Serial1.available() > 0) {
   command = Serial1.read();
   switch(command){
    case 'R': refresh = true;                                          break;
    case 'I': speedKp += 0.01;     speedPID.setP(&speedKp);            break;
    case 'J': speedKi += 0.0001;   speedPID.setI(&speedKi);            break;
    case 'K': speedKd += 0.01;     speedPID.setD(&speedKd);            break;
    case 'i': speedKp -= 0.01;     speedPID.setP(&speedKp);            break;
    case 'j': speedKi -= 0.0001;   speedPID.setI(&speedKi);            break;
    case 'k': speedKd -= 0.01;     speedPID.setD(&speedKd);            break;
    case 'X': stabilityKp += 0.5;  stabilityPID.setP(&stabilityKp);    break;
    case 'Y': stabilityKi += 0.05; stabilityPID.setI(&stabilityKi);    break;
    case 'Z': stabilityKd += 0.1;  stabilityPID.setD(&stabilityKd);    break;
    case 'x': stabilityKp -= 0.5;  stabilityPID.setP(&stabilityKp);    break;
    case 'y': stabilityKi -= 0.05; stabilityPID.setI(&stabilityKi);    break;
    case 'z': stabilityKd -= 0.1;  stabilityPID.setD(&stabilityKd);    break;
    case 'O': angleOffset += 0.1;                                      break;
    case 'o': angleOffset -= 0.1;                                      break;
    case 'H': speedStatus = !speedStatus;                              break;
    case 'E': stabilityStatus = !stabilityStatus;                      break;
    case 'S': saveEEprom();                                            break;
    case 'U': readEEprom();                                            break;
    case 'T': velocidade.setDir(STOP);     angulo.setDir(STOP);        break;
    case 'f': velocidade.setDir(FORWARD);  angulo.setDir(FORWARD);     break;
    case 'b': velocidade.setDir(BACKWARD); angulo.setDir(BACKWARD);    break;
    case 't': giro.setDir(STOP);                                       break;
    case 'l': giro.setDir(LEFT);                                       break;
    case 'r': giro.setDir(RIGHT);                                      break;
    default: command = "";                                             break;
   }
   command = ""; // No repeats
  }

  if(refresh){
    Serial1.print(targetAngle);            Serial1.print(F(","));    // 1
    Serial1.print(speedKp, 2);             Serial1.print(F(","));    // 2
    Serial1.print(speedKi, 4);             Serial1.print(F(","));    // 3
    Serial1.print(speedKd);                Serial1.print(F(","));    // 4
    Serial1.print(controlOutput);          Serial1.print(F(","));    // 5
    Serial1.print(stabilityKp);            Serial1.print(F(","));    // 6
    Serial1.print(stabilityKi);            Serial1.print(F(","));    // 7
    Serial1.print(stabilityKd);            Serial1.print(F(","));    // 8
    Serial1.print(currentAngle);           Serial1.print(F(","));    // 9
    Serial1.print(angleOffset);            Serial1.print(F(","));    // 10
    Serial1.print(imu.getTemperature());   Serial1.print(F(","));    // 11
    Serial1.print(readVoltage());          Serial1.print(F(","));    // 12
    Serial1.print(speedStatus);            Serial1.print(F(","));    // 13
    Serial1.println(stabilityStatus);      Serial1.print(F(","));    // 14
  }
}//serialData
```

```
void resetEEprom(){

  for (int i = 0 ; i < 30 ; i++)
    EEPROM.write(i, 0);

}//resetEEprom
void saveEEprom(){

  EEPROM.put(0, speedKp);
  EEPROM.put(4, speedKi);
  EEPROM.put(8, speedKd);
  EEPROM.put(12, stabilityKp);
  EEPROM.put(16, stabilityKi);
  EEPROM.put(20, stabilityKd);
  EEPROM.put(24, angleOffset);

}//saveEEprom

void readEEprom(){

  EEPROM.get(0, speedKp);
  EEPROM.get(4, speedKi);
  EEPROM.get(8, speedKd);
  EEPROM.get(12, stabilityKp);
  EEPROM.get(16, stabilityKi);
  EEPROM.get(20, stabilityKd);
  EEPROM.get(24, angleOffset);

  speedPID.setPID(&speedKp, &speedKi, &speedKd);
  stabilityPID.setPID(&stabilityKp, &stabilityKi, &stabilityKd);

}//readEEprom
```

APÊNDICE XII – CLASSE PID, ARQUIVO: PID.H

```
#ifndef dPID_h
#define dPID_h

#include "Arduino.h"

class dPID{

  public:
    dPID(float Kp, float Ki, float Kd,float Min, float Max);

    float updatePID(float setpoint, float input, float deltaTime);

    void setPID(float *Kp, float *Ki, float *Kd);

    void setP(float *Kp);
    void setI(float *Ki);
    void setD(float *Kd);

    void SetOutputLimits(float Min, float Max);

    void resetPID();

    float getP();
    float getI();
    float getD();

  private:
    float _Kp;
    float _Ki;
    float _Kd;
    float _a;
    float _b;
    float _c;
    float _outMin;
    float _outMax;
    float _uk, _uk1;
    float _ek, _ek1, _ek2;

};

#endif
```

APÊNDICE XIII - CLASSE PID, ARQUIVO: PID.CPP

```
#include "Arduino.h"
#include "dPID.h"

dPID::dPID(float Kp, float Ki, float Kd, float Min, float Max){

  _Kp = Kp;
  _Ki = Ki;
  _Kd = Kd;

  _outMin = Min;
  _outMax = Max;

  _uk = 0; _uk1 = 0;
  _ek = 0; _ek1 = 0; _ek2 = 0;

}//dPID

float dPID::updatePID(float setpoint, float input, float Ta){

    Ta = 1.0;

    _a = _Kp + _Ki*Ta/2.0 + _Kd/Ta;
    _b = -_Kp + _Ki*Ta/2.0 - 2.0*_Kd/Ta;
    _c = _Kd/Ta;

    _ek = setpoint - input;

    _uk = _uk1 + _a*_ek + _b*_ek1 + _c*_ek2;

    _uk = (_uk < _outMin )? _outMin:((_uk > _outMax)? _outMax : _uk);

    _ek2 = _ek1;
    _ek1 = _ek;
    _uk1 = _uk;

    return _uk;

}//updatePID

void dPID::resetPID(){

  _uk = 0; _uk1 = 0;
  _ek = 0; _ek1 = 0; _ek2 = 0;

}//resetPID

void dPID::setPID(float *Kp, float *Ki, float *Kd){

  if(*Kp < 0) *Kp = 0;
  if(*Ki < 0) *Ki = 0;
  if(*Kd < 0) *Kd = 0;

  _Kp = *Kp; _Ki = *Ki; _Kd = *Kd;

}//setPID
```

```
void dPID::setP(float *Kp){

  if(*Kp < 0) *Kp = 0;
  _Kp = *Kp;

}//setP

void dPID::setI(float *Ki){

  if(*Ki < 0) *Ki = 0;
  _Ki = *Ki;

}//setI

void dPID::setD(float *Kd){

  if(*Kd < 0) *Kd = 0;
  _Kd = *Kd;

}//setD

void dPID::SetOutputLimits(float Min, float Max){

  if(Min >= Max) return;

  _outMin = Min;
  _outMax = Max;

}//SetOutputLimits

float dPID::getP(){ return _Kp; }
float dPID::getI(){ return _Ki; }
float dPID::getD(){ return _Kd; }
```

APÊNDICE XIV - CLASSE MPU6050, ARQUIVO: MPU6050.H

```
#ifndef Mpu6050_h
#define Mpu6050_h

#include "Arduino.h"
#include <Wire.h>

class Mpu6050{
  public:
    Mpu6050(int addr);

    void  init();
    bool  testConnection();
    float setInitialAngle();
    void  getMotion(int16_t *ax, int16_t *ay, int16_t *az, int16_t *gx,
int16_t *gy, int16_t *gz);
    void  getAcceleration(int16_t *ax, int16_t *ay, int16_t *az);
    float getTemperature();
    void  getRotation(int16_t *gx, int16_t *gy, int16_t *gz);

    void  calibration();
    void  meansensors();
    void  calcOffsets();
    void  setXAccelOffset(int16_t offset);
    void  setYAccelOffset(int16_t offset);
    void  setZAccelOffset(int16_t offset);
    void  setXGyroOffset(int16_t offset);
    void  setYGyroOffset(int16_t offset);
    void  setZGyroOffset(int16_t offset);

  private:

  int _address;

  //Change variables if you want to fine tune the skecth to your needs.
  int _bufferSize=1000;// Amount of readings used to average(default:1000)
  int _acelDeadzone=8; // Acelerometer error allowed (default:8)
  int _giroDeadzone=1; // Giro error allowed (default:1)

  int16_t _ax, _ay, _az, _gx, _gy, _gz, _tmp;
  int16_t _axMean, _ayMean, _azMean, _gxMean, _gyMean, _gzMean, _state=0;
  int16_t _gxOffset, _gyOffset, _gzOffset, _axOffset, _ayOffset, _azOffset;

};

#endif
```

APÊNDICE XV - CLASSE MPU6050, ARQUIVO: MPU6050.CPP

```
#include "Arduino.h"
#include "mpu6050.h"

Mpu6050::Mpu6050(int addr){ _address = addr; }

void Mpu6050::init(){

  // Reset chip
  Wire.beginTransmission(_address);
  Wire.write(0x6B);                  // PWR_MGMT_1 register (6B hex)
  Wire.write(0b10000000);            // Reset chip
  Wire.endTransmission(true);        // End the transmission with the gyro.
  delay(200);

  // Clear the 'sleep' bit to start the sensor and select clock source
  Wire.beginTransmission(_address);
  Wire.write(0x6B);                  // PWR_MGMT_1 register (6B hex)
  Wire.write(0b00000001);            // Clear the 'sleep' bit to start
  Wire.endTransmission(true);        // End the transmission with the gyro.

  //Gyroscope's sensitivity scale at 250deg/seg  => 131 LSB/deg/s
  Wire.beginTransmission(_address);
  Wire.write(0x1B);                  // GYRO_CONFIG register (1B hex)
  Wire.write(0b00000000);            // 250dps full scale
  Wire.endTransmission();            // End the transmission with the gyro

  //Accelerometer's sensitivity scale at +/- 2g  => 16384 LSB/deg/s
  Wire.beginTransmission(_address);
  Wire.write(0x1C);                  // ACCEL_CONFIG register (1A hex)
  Wire.write(0b00000000);            // +/- 2g full scale range
  Wire.endTransmission(true);        // End the transmission with the gyro

  //Set some filtering to improve the raw data.
  Wire.beginTransmission(_address);
  Wire.write(0x1A);                  // CONFIG register (1A hex)
  Wire.write(0b00000101);            // Set Digital Low Pass Filter to 10Hz
  Wire.endTransmission(true);        // End the transmission with the gyro
}//init

bool Mpu6050::testConnection() {
    Wire.beginTransmission(_address);
    Wire.write(0x75);
    Wire.endTransmission(false);
    Wire.requestFrom(_address, 1, true);
    return (Wire.read() & 0b01111110) == 0b01101000;
}//testConnection

float Mpu6050::setInitialAngle(){
    Wire.beginTransmission(_address);
    Wire.write(0x3B); // Address of acc.x
    Wire.endTransmission(false);
    Wire.requestFrom(_address, 6, true);
    _ax = Wire.read() << 8 | Wire.read();
    _ay = Wire.read() << 8 | Wire.read();
    _az = Wire.read() << 8 | Wire.read();
    return atan2(_ay, _az) * RAD_TO_DEG;
}//setInitialAngle
```

```
void Mpu6050::getMotion(int16_t *ax, int16_t *ay, int16_t *az, int16_t *gx,
int16_t *gy, int16_t *gz) {
  Wire.beginTransmission(_address);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(_address, 14, true);  // request a total of 14 registers
  *ax = Wire.read() << 8 | Wire.read();
  *ay = Wire.read() << 8 | Wire.read();
  *az = Wire.read() << 8 | Wire.read();
  _tmp = Wire.read() << 8 | Wire.read();
  *gx = Wire.read() << 8 | Wire.read();
  *gy = Wire.read() << 8 | Wire.read();
  *gz = Wire.read() << 8 | Wire.read();
}//getMotion

void Mpu6050::getAcceleration(int16_t *ax, int16_t *ay, int16_t *az) {
  Wire.beginTransmission(_address);
  Wire.write(0x3B);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(_address, 6, true);  // request a total of 6 registers
  *ax = Wire.read() << 8 | Wire.read();
  *ay = Wire.read() << 8 | Wire.read();
  *az = Wire.read() << 8 | Wire.read();
}//getAcceleration

float Mpu6050::getTemperature(){
  Wire.beginTransmission(_address);
  Wire.write(0x41);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(_address, 2, true);  // request a total of 2 registers
  _tmp = Wire.read() << 8 | Wire.read();
  return (_tmp / 340.00f) + 36.53f;
}//getTemperature

void Mpu6050::getRotation(int16_t *gx, int16_t *gy, int16_t *gz) {
  Wire.beginTransmission(_address);
  Wire.write(0x43);  // starting with register 0x3B (ACCEL_XOUT_H)
  Wire.endTransmission(false);
  Wire.requestFrom(_address, 6, true);  // request a total of 6 registers
  *gx = Wire.read() << 8 | Wire.read();
  *gy = Wire.read() << 8 | Wire.read();
  *gz = Wire.read() << 8 | Wire.read();
}//getRotation

void Mpu6050::meansensors(){
  long i=0, axBuff=0, ayBuff=0, azBuff=0, gxBuff=0, gyBuff=0, gzBuff=0;
  while (i < (_bufferSize+101)){
   Mpu6050::getMotion(&_ax, &_ay, &_az, &_gx, &_gy, &_gz);
    if (i > 100 && i <= (_bufferSize+100)){ //First 100 measures discarded
      axBuff = axBuff + _ax;
      ayBuff = ayBuff + _ay;
      azBuff = azBuff + _az;
      gxBuff = gxBuff + _gx;
      gyBuff = gyBuff + _gy;
      gzBuff = gzBuff + _gz;
    }
    if (i == (_bufferSize+100)){
      _axMean = axBuff / _bufferSize;
      _ayMean = ayBuff / _bufferSize;
      _azMean = azBuff / _bufferSize;
```

```
      _gxMean = gxBuff / _bufferSize;
      _gyMean = gyBuff / _bufferSize;
      _gzMean = gzBuff / _bufferSize;
    }
    i++;  delay(2); //Needed so we don't get repeated measures
  }
}//meansensors

void Mpu6050::calcOffsets(){
  _axOffset = -_axMean / 8;
  _ayOffset = -_ayMean / 8;
  _azOffset = (16384 - _azMean) / 8;
  _gxOffset = -_gxMean / 4;
  _gyOffset = -_gyMean / 4;
  _gzOffset = -_gzMean / 4;
  while (1){
    int ready=0;
    Mpu6050::setXAccelOffset(_axOffset);
    Mpu6050::setYAccelOffset(_ayOffset);
    Mpu6050::setZAccelOffset(_azOffset);
    Mpu6050::setXGyroOffset(_gxOffset);
    Mpu6050::setYGyroOffset(_gyOffset);
    Mpu6050::setZGyroOffset(_gzOffset);
    meansensors();
    Serial.println("...");
    if (abs(_axMean) <= _acelDeadzone) ready++;
    else _axOffset = _axOffset - _axMean / _acelDeadzone;
    if (abs(_ayMean) <= _acelDeadzone) ready++;
    else _ayOffset = _ayOffset - _ayMean / _acelDeadzone;
    if (abs(16384 - _azMean) <= _acelDeadzone) ready++;
    else _azOffset = _azOffset + (16384 - _azMean) / _acelDeadzone;
    if (abs(_gxMean) <= _giroDeadzone) ready++;
    else _gxOffset = _gxOffset - _gxMean / (_giroDeadzone+1);
    if (abs(_gyMean) <= _giroDeadzone) ready++;
    else _gyOffset = _gyOffset - _gyMean / (_giroDeadzone+1);
    if (abs(_gzMean) <= _giroDeadzone) ready++;
    else _gzOffset = _gzOffset - _gzMean / (_giroDeadzone+1);
    if (ready == 6) break;
  }
}//calcOffsets

void Mpu6050::calibration() {
  Mpu6050::setXAccelOffset(0);
  Mpu6050::setYAccelOffset(0);
  Mpu6050::setZAccelOffset(0);
  Mpu6050::setXGyroOffset(0);
  Mpu6050::setYGyroOffset(0);
  Mpu6050::setZGyroOffset(0);
  Serial.println("Reading sensors for first time...");
  Mpu6050::meansensors();  delay(1000);
  Serial.println("Calculating offsets...");
  Mpu6050::calcOffsets();  delay(1000);
  Mpu6050::meansensors();
  Serial.println("FINISHED!");
  Serial.print("Sensor readings with offsets:\t");
  Serial.print(_axMean);    Serial.print("\t");
  Serial.print(_ayMean);    Serial.print("\t");
  Serial.print(_azMean);    Serial.print("\t");
```

```
  Serial.print(_gxMean);    Serial.print("\t");
  Serial.print(_gyMean);    Serial.print("\t");
  Serial.println(_gzMean);  Serial.print("Offsets:\t");
  Serial.print(_axOffset);  Serial.print("\t");
  Serial.print(_ayOffset);  Serial.print("\t");
  Serial.print(_azOffset);  Serial.print("\t");
  Serial.print(_gxOffset);  Serial.print("\t");
  Serial.print(_gyOffset);  Serial.print("\t");
  Serial.println(_gzOffset);
  Serial.println("\nData is printed: acelX acelY acelZ giroX giroY giroZ");
  Serial.println("Check sensor readings are close to 0 0 16384 0 0 0");
}//calibration

void Mpu6050::setXAccelOffset(int16_t offset) {
  Wire.beginTransmission(_address);
  Wire.write(0x06); //[15:0] XA_OFFS
  Wire.write((uint8_t)(offset >> 8));
  Wire.write((uint8_t)offset);
  Wire.endTransmission();
}
void Mpu6050::setYAccelOffset(int16_t offset) {
  Wire.beginTransmission(_address);
  Wire.write(0x08); //[15:0] YA_OFFS
  Wire.write((uint8_t)(offset >> 8));
  Wire.write((uint8_t)offset);
  Wire.endTransmission();
}
void Mpu6050::setZAccelOffset(int16_t offset) {
  Wire.beginTransmission(_address);
  Wire.write(0x0A); //[15:0] ZA_OFFS
  Wire.write((uint8_t)(offset >> 8));
  Wire.write((uint8_t)offset);
  Wire.endTransmission();
}
void Mpu6050::setXGyroOffset(int16_t offset) {
  Wire.beginTransmission(_address);
  Wire.write(0x13); //[15:0] XG_OFFS_USR
  Wire.write((uint8_t)(offset >> 8));
  Wire.write((uint8_t)offset);
  Wire.endTransmission();
}
void Mpu6050::setYGyroOffset(int16_t offset) {
  Wire.beginTransmission(_address);
  Wire.write(0x15); //[15:0] YG_OFFS_USR
  Wire.write((uint8_t)(offset >> 8));
  Wire.write((uint8_t)offset);
  Wire.endTransmission();
}
void Mpu6050::setZGyroOffset(int16_t offset) {
  Wire.beginTransmission(_address);
  Wire.write(0x17); //[15:0] ZG_OFFS_USR
  Wire.write((uint8_t)(offset >> 8));
  Wire.write((uint8_t)offset);
 Wire.endTransmission();
}
```

APÊNDICE XVI - CLASSE RAMPA, ARQUIVO: RAMPA.H

```
#ifndef rampa_h
#define rampa_h

#include "Arduino.h"

class ramp{

  public:
    ramp(float inc, float dec, float limit);

    float setpoint();
    void  setDir(int dir);
    void  setInc(float inc);
    void  setDec(float dec);
    void  setLimit(float limit);

  private:
    int   _dir;
    float _inc;
    float _dec;
    float _limit;
    float _setpoint;
};

#endif
```

APÊNDICE XVII - CLASSE RAMPA, ARQUIVO: RAMPA.CPP

```
#include "Arduino.h"
#include "rampa.h"

ramp::ramp(float inc, float dec, float limit){

  _inc = inc;
  _dec = dec;
  _limit = limit;
  _dir = 0;
  _setpoint = 0;

}//dPID

float ramp::setpoint(){

  if(_dir == 0){
    if(_setpoint > _dec)        _setpoint -= _dec;
    else if(_setpoint < -_dec) _setpoint += _dec;
    else _setpoint = 0;
  }
  else if(_dir < 0){
    if(_setpoint > -_limit) _setpoint -= _inc;
  }
  else if(_dir > 0){
    if(_setpoint < _limit)  _setpoint += _inc;
  }

  return _setpoint;

}

void ramp::setDir(int dir){

  _dir = dir;

}

void ramp::setInc(float inc){

  _inc = inc;

}

void ramp::setDec(float dec){

  _dec = dec;

}

void ramp::setLimit(float limit){

  _limit = limit;

}
```