Universidade de Brasília
Departamento de Engenharia Elétrica
Professor: Henrique Cezar Ferreira
Disciplina: Controle Digital

Projeto - Controle discreto de um motor DC

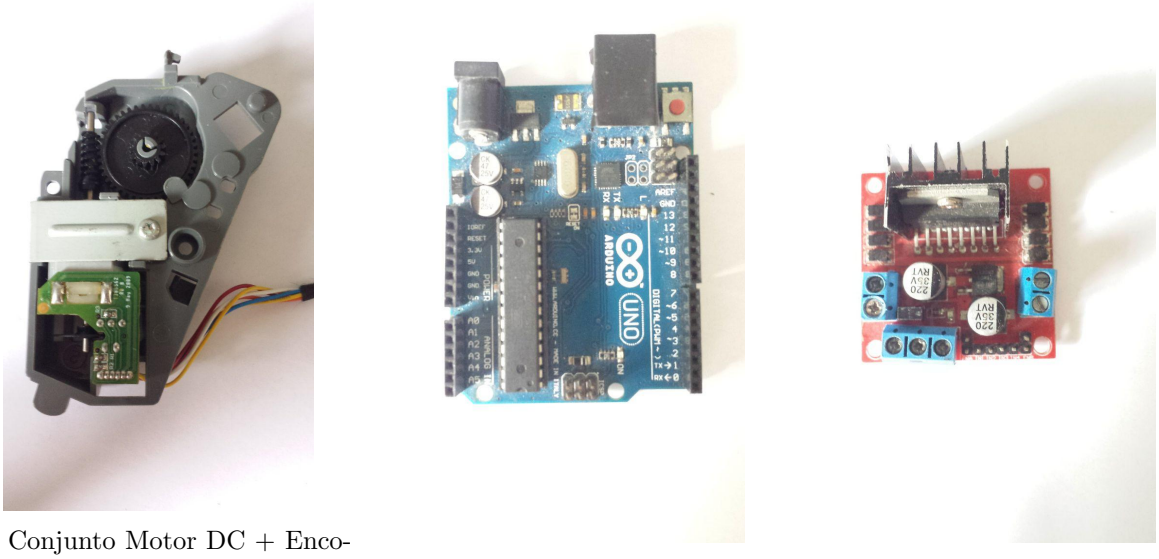
Rafael Lima 10/0131093
Leonardo Cardoso Botelho 11/0154151

Brasília
2020

Conteúdo

1 Introdução

O presente trabalho discorre sobre o desenvolvimento de um servo mecanismo a partir de técnicas de controle discreto com base em uma planta composta por um motor DC (motor de corrente contínua), um sistema de redução e um sensor encoder de maneira a permitir o controle da posição a partir de um sinal de referência. O conjunto de peças foi retirada de um equipamento antigo que havia sido descartado. Para implementação do controle, além da planta foi utilizado um Arduino UNO e uma módulo com uma ponte H para o controle de potência conforme mostrado na figura 1.



(a) Conjunto Motor DC + Encoder

(b) Arduino UNO

(c) Ponte H

Figura 1: Peças para proposta da planta

1.1 Justificativa

Sistemas de posicionamento utilizando motores de corrente contínua está presente nos mais variados equipamentos presentes nos dia de hoje. Isso se dá pois eles possuem bom custo benefício,

Sua complexidade é razoável para o contexto da disciplina de Controle de Digital pois temos nesse sistema pelo menos dois atrasos nos elementos de atuação e sensoreamento. A caixa de redução por sua vez também insere um aspecto interessante ao processo, visto os efeitos de histerese e zona morta.

Do ponto de vista didático, esse sistema possui bons elementos a serem compreendidos no contexto da disciplina de Controle Digital. Podemos destacar as etapas de modelagem, discretização da planta, alocação de polos no domínio Z, projeto em LGR no domínio discreto de um controlador, identificação de sistema e implementação em uma planta real.

Por fim, o controle digital pressupõe um sistema computacional para sua execução, podendo ser operado por um microcontrolador disponível a custo acessível comercialmente. Isso torna esse projeto relevante no contexto das aplicações que utilizam motores de corrente contínua.

1.2 Objetivos

Este projeto têm seguintes objetivos:

(*1*) Obter modelo da planta no domínio da frequência (Motor DC em conjunto com caixa de redução);

(*2*) Obter modelo do sensor no domínio da frequência;

(*3*) Obter modelos discreto da planta em conjunto com o sensor

(*4*) Desenvolver modelo em *Simulink* para o projeto

(*5*) Obter controlado descrito através de equações de diferenças;

(*6*) Implementação e Validação do controlador em teste no sistema real;

1.3 Metodologia de Projeto

Para o desenvolvimento do projeto foi feito a partir das etapas relatadas a seguir. Primeiramente foi feito uma revisão bibliográfica e modelagem da planta e avaliação em ambiente de simulação. Num segunda fase foi ajustado o modelo encontrado para o planta real.

- Estudo Planta em Simulação
  - (*1*) Revisão Bibliográfica para busca do modelo de um servo motor na literatura
  - (*2*) Simulação do modelo
  - (*3*) Implementação do controle em simulação
- Estudo Planta Real
  - (*1*) Simulação da planta no *Tinkercad*
  - (*2*) Identificação de Parâmetro da Planta
  - (*3*) Simulação da planta a partir do modelo identificado
  - (*4*) Implementação do controle em simulação
  - (*5*) Implementação do controle na planta

2 Desenvolvimento

O desenvolvimento foi feitas em 3 fases: implementação em hardware, caracterização do sistema e projeto do controlador. Primeiramente foi feito um estudo e caracterização de cada um dos componentes, depois foram feitos vários testes visando a identificação do modelo da planta bem como suas propriedades do ponto de vista de controle e por fim foi desenvolvido um controlador a partir dos dados obtidos experimentalmente que pudesse permitir o funcionamento do sistema a partir dos requisitos desejados.

2.1 Implementação em Hardware

Para o projeto de um sistema de controle é necessário garantir alguns requisitos básicos do ponto de vista de acionamento, sensoriamento e temporização. Primeiramente é necessário que os atuadores e os sensores deve possuir um faixa de operação compatível com a dinâmica esperada do sistema. Além disto deve ser observado também o tempo necessário para a ação de controle como forma de garantir uma amostragem com período constante, permitindo assim o uso da teoria de controle desenvolvida ao longo da disciplina.

Como efeito, em virtude da natureza dos componentes escolhidos, foi necessário implementar algumas funcionalidades do zero. Em particular um tempo maior foi necessário na leitura do deslocamento angular a partir do sensor óptico em no acionamento do motor conjunto com o Arduino. Este estudo foi implementado primeiramente no *Tinkercad* e posteriormente foi feito diretamente na planta.

2.1.1 Simulação no Tinkercad

Como forma de prevenir danos as peças, foi utilizado o *Tinkercad* para simulação do sistema e como uma plataforma de estudos a respeito do comportamento do motor DC em conjunto com o encoder incremental. O *Tinkercad* foi escolhido por sua simplicidade de uso e facilidade de integração do circuito diretamente com a programação do Arduino.

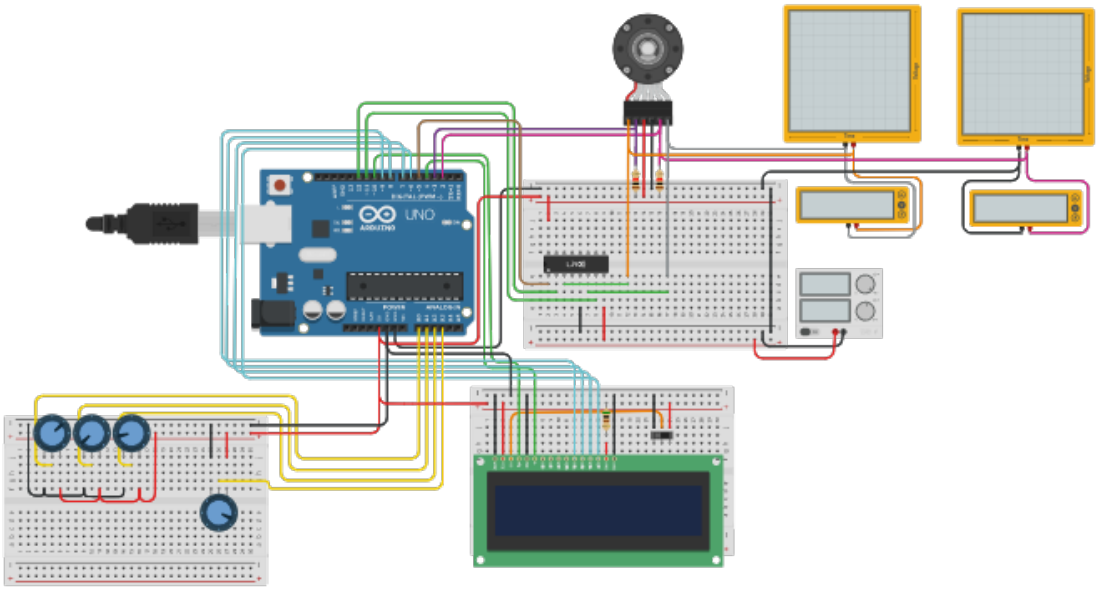


Figura 2: Diagrama Sistema implementado no Tinkercad

Desta forma foi possível avaliar os algoritmos usados para a contagem de passo pelo encoder em conjunto com o acionamento do motor, cálculo do PID e detecção de sentido de rotação. Conforme ilustrado pela figura 3, foram utilizados potenciômetros para um ajuste manual e grosseiro dos ganhos do PID. Também foi adotado um display para mostrar os ganhos adotados. Com isto foi possível chegar a uma combinação de ganhos que permitia o motor alcançar o valor da referência ou ao menos parar após um tempo de movimento.

Uma atenção maior foi dada a como obter e condicionar os dados do encoder e como definir uma estrutura mínima do código para acondicionar tanto o acionamento como a leitura dos sensores dentro do período escolhido. Com isto foi possível gerar alguns gráficos iniciais da variação da posição ao longo do tempo. Porém dado as limitações da plataforma, principalmente nos modelos adotados para o motor e para comportamento das portas e ações do Arduíno não foi possível adotar nenhuma forma de identificação dos parâmetros.

O maior problema encontrado foi que o tempo real gasto pelo funções do Arduino não era obedecido pela simulação e com isto não dava para verificar qual a frequência máxima que poderia ser usada para amostragem. Assim, concluído o estudo preliminar passamos para experimentos feitos diretamente na planta real.

2.1.2 Temporização

Como o estilo de controlador adotado pressupõe uma amostragem fixa, foi necessário adotar um controle de temporização estrito como forma de garantir que tanto o acionamento como a leitura do encoder seria feita dentro do tempo adequado. A leitura de cada passo do encoder é

feita através de interrupção e a frequência de acionamento é feita a partir da comparação do valor de tempo para o instante atual com temporizadores internos. Para avaliar o tempo gasto em um conjunto de instruções foi adota uma estratégia similar registrando os valores intermediários e enviando para a serial, uma vez que não tinha disponível nenhum osciloscópio ou equipamento de medida externo para esta avaliação.

2.1.3 Leitura Encoder

O encoder presente na planta escolhida é sistema ótico e incremental e com isto foi necessário adicionar ao código no Arduíno a contagem de passos a detecção de sentido de rotação. O sinal do encoder é composto por duas ondas quadradas defasadas em 90 graus. Desta forma, o deslocamento absoluto pode ser obtido através da contagem dos pulsos e o sentido de rotação pela comparação do dois sinais.

Como esta variação é relativamente rápida, o sinal de borda de subida do encoder foi associado a uma interrupção no processador. Como efeito a cada vez que é detectado um pulso, o processamento é interrompido e uma função para contagem e detecção da direção é chamada. Desta forma é possível manter o algoritmo de controle em paralelo com a leitura do encoder.

A partir da captura de dados movendo-se manualmente o eixo do motor foi detectado a presença de bastante ruído na leitura. Foi percebido nos primeiros experimentos que contador acabava sendo incrementado ainda que o motor estivesse parado. Para reduzir o ruído algumas medidas foram adotadas: primeiro o sinal de alimentação do encoder foi separado do sinal de alimentação da ponte H, foi adotado um cabo com par trançado como forma de reduzir a interferência devido ao ruído introduzido com o acionamento do motor, a fonte de alimentação do motor e do arduino também foi separada. Com estas medidas, o ruído reduziu bastante.

2.1.4 Acionamento Motor

O acionamento do motor foi feito através de uma ponte H. Um fator que introduz não linearidade ao sistema é a presença de uma zona morta correspondente ao mínimo de tensão necessária para fazer o motor girar partindo do repouso. Além disto o sistema possui uma limitação de tensão máxima que é capaz de fornecer, o que também impõe um limite a região linear para controle do motor.

2.2 Caracterização Sistema

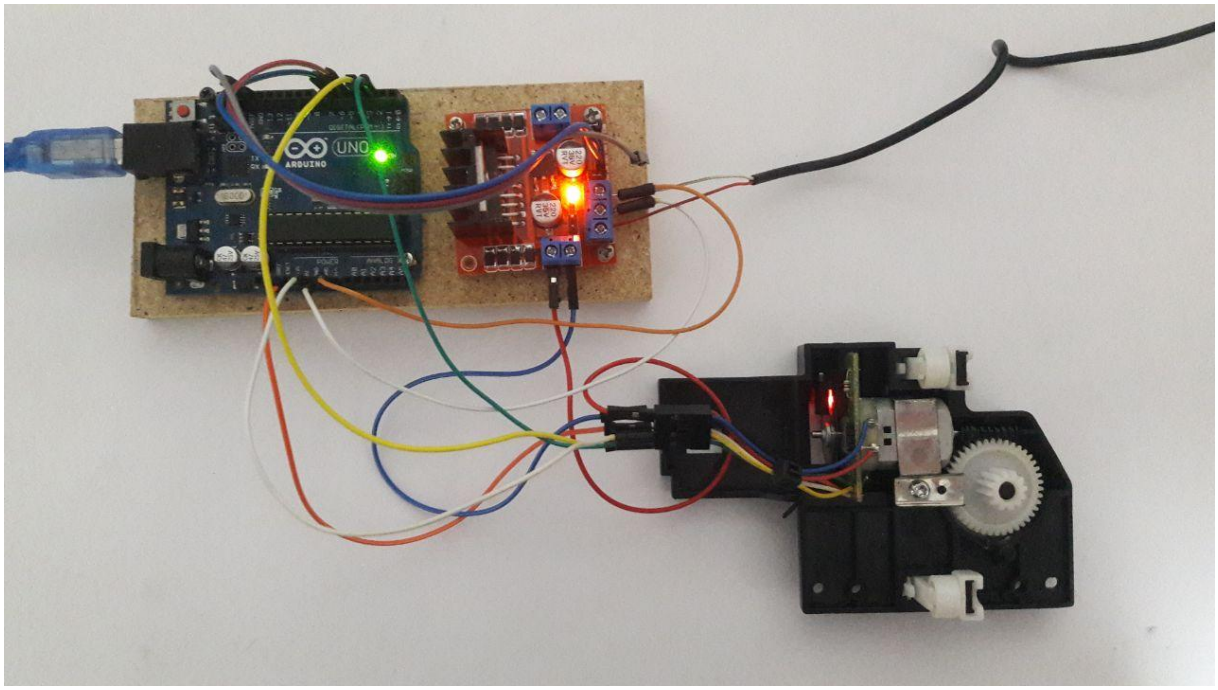


Figura 3: Planta Montada

2.2.1 Região Não Linear

Idealmente o comportamento poderia ser representado através de um modelo linear para todas as condições de operação, No entanto, na prática devido as características do hardware, temos uma região de operação a qual o modelo pode ser utilizado delimitada por uma região inferior e superior.

Para o motor começar a se mover é preciso fornecer um mínimo de energia. Esta quantidade varia de acordo com as características eletromecânicas do motor. Em particular para o motor utilizado, a faixa de operação é de 3V a 6V, sendo 3V o mínimo para fazer o motor girar e 6V uma tensão máxima para não danificar.

A conversão do sinal de controle do Arduino em um sinal de potência foi feita através uma ponte H em conjunto com um sinal PWM. Como efeito, a tensão de alimentação da fonte utilizada para o motor foi amostrada usando 8 bits. Desta forma, chegar ao valor teórico correspondente a zona morta do motor para uma alimentação de 5V basta

$$D_{min} = \frac{3V}{V_{fonte}}(2^8 - 1) = \frac{3 * 255}{5} = 153$$

Para obter este valor experimentalmente foi feito um programa aplicando um sinal de entrada de dente de serra (em anexo). Isto é, o valor de referência passado para a tensão do motor foi gradualmente aumentado e em seguida gradualmente reduzido de forma periódica. Os valores obtidos foram registrados a partir do Matlab através da conexão Serial com o Arduino.

Mesmo que frequência interna do Arduino seja próximo de $14MHz$, as operações de comunicação via Serial são bastante dispendiosas. Por conta disto, o menor período conseguido experimentalmente a qual é mantida um a operação com ciclo em tempo contante foi de $60ms$. Foi escolhido então uma período de $100ms$ de forma a permitir uma margem segura de folga de tempo para todas as operações. Para períodos menores que este valor, o tempo gasto com a comunicação serial, cálculos e as interrupções do encoder acabava sendo maior que o período de amostragem fixo e as garantias de funcionamento em tempo real eram perdidas.

2.2.2 Identificação de Parâmetros

Uma vez definido uma região linear de operação foi adotado o procedimento de identificação do sistema. Pelas especificação do fabricante temos

V_i $[V]$	$\dot{\theta}$ $[RPM]$	$\dot{\theta}$ $[Hz]$
3	3000	50
6	6000	60

Tabela 1: Velocidade de operação sem carga

De forma que poderíamos adotar como uma aproximação razoável do comportamento dinâmico do motor como uma função de transferência de primeira ordem na forma

$$G_m(s) = \frac{\beta}{\alpha s + 1} \tag{1}$$

Neste sistema temos apenas a medida da posição através do encoder. Este processo é feito através do conjunto de várias etapas, incluindo amostragem através da contagem do pulsos e a discretização referente ao instante de contabilização da leitura do sensor. No entanto, dado a grande diferença entre o tempo gasto neste processo com o tempo gasto na comunicação serial, podemos aproximar a dinâmica do sistema de medição de posição como um ganho fixo.

Aplicando a transformada Z podemos representar o sistema em tempo discreto pela seguinte função:

$$G(z) = \frac{Y(z)}{U(z)} = \frac{\beta_1 z + \beta_2}{z^2 + \alpha_1 z + \alpha_2} \tag{2}$$

Do que podemos observar que a discretização traz como efeitos tanto o deslocamento dos polos, como também a inclusão de um zero na planta.

A partir da função 4 podemos obter a representação do sistema a partir de equação de diferenças como:

$$Y[k+2] = -Y[k+1]\alpha_1 - Y[k]\alpha_2 + U[k+1]\beta_1 + U[k]\beta_2 \tag{3}$$

Tomando um conjunto maior de dados podemos reescrever a equação na forma matricial como

$$Y[k+2] = \begin{pmatrix} -Y[k+1] & -Y[k] & U[k+1] & U[k] \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \beta_1 \\ \beta_2 \end{pmatrix} \tag{4}$$

A partir do qual podemos relacionar o sinal de saída com o sinal de entrada do sistema e obter os parâmetros α_1,α_2,β_1 e β_2 por regressão linear. Para obter uma melhor caracterização foi gerado um sinal formado por pulsos de tamanhos e durações variadas diretamente no Arduino. Conforme mostrado na figura 6

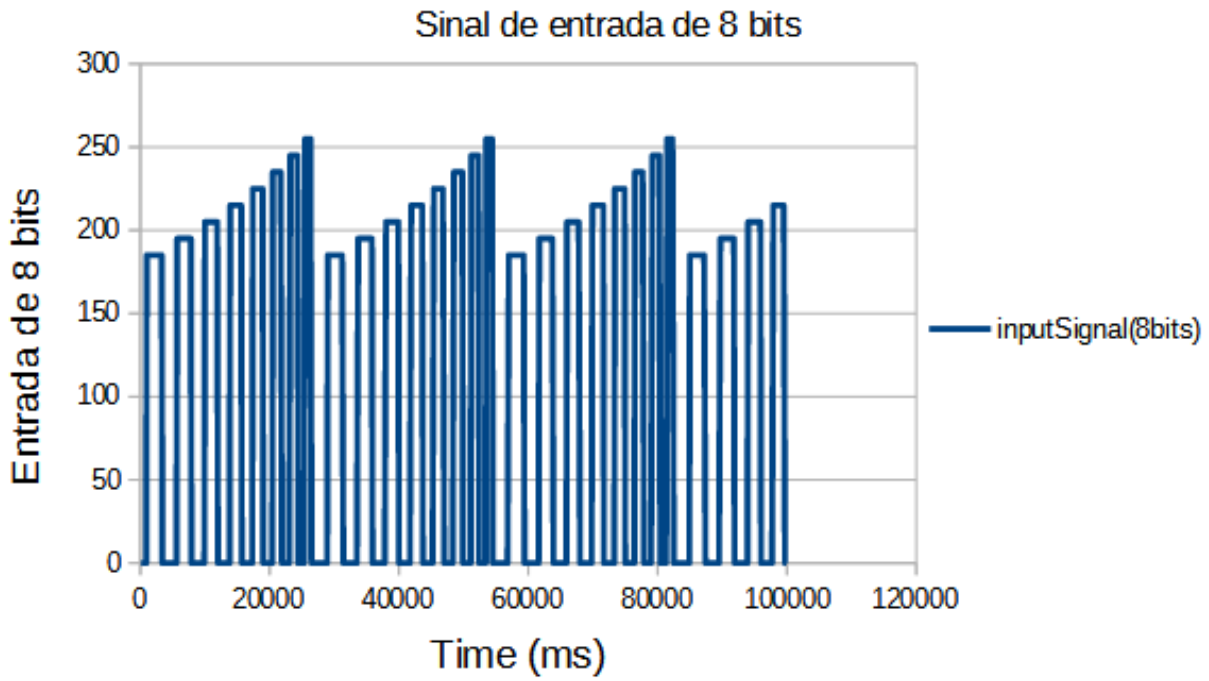


Figura 4: Sinal de 8 bits

A partir deste sinal foi obtido a seguinte resposta:

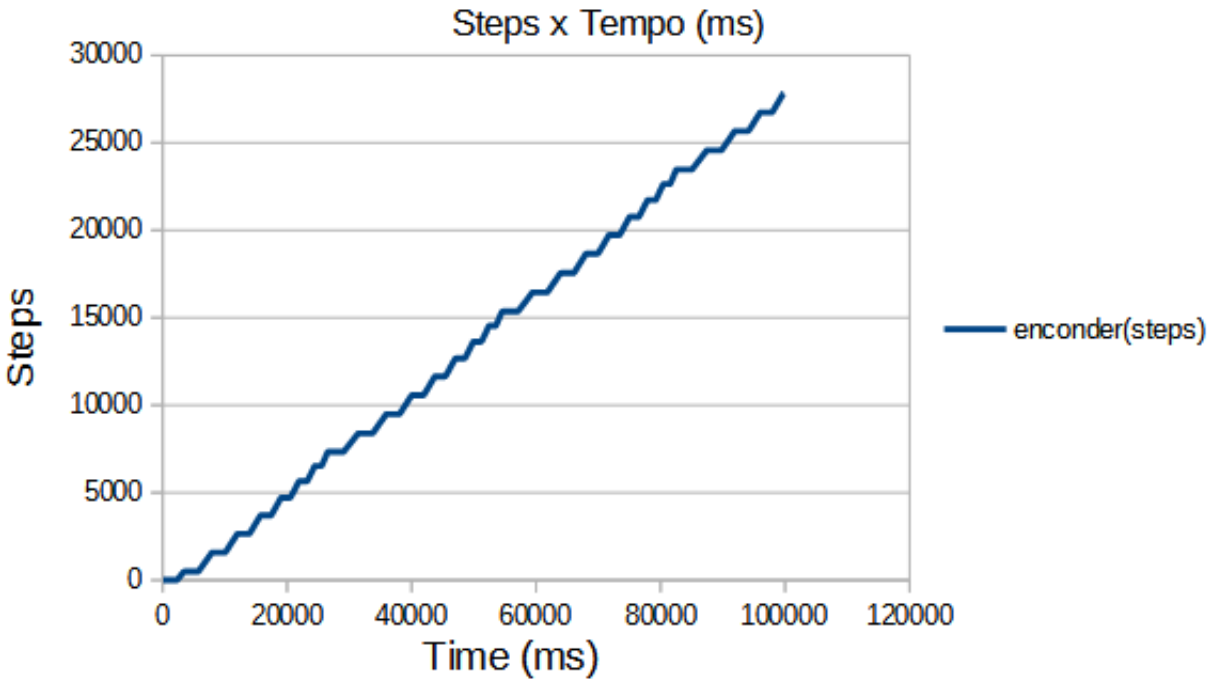


Figura 5: Steps x tempo (ms)

Para visualizar melhor foi calculado a estimativa da velocidade, e como pode ser observado o gráfico da velocidade da planta em malha aberta acompanha o mesmo formato do gráfico do sinal de entrada.

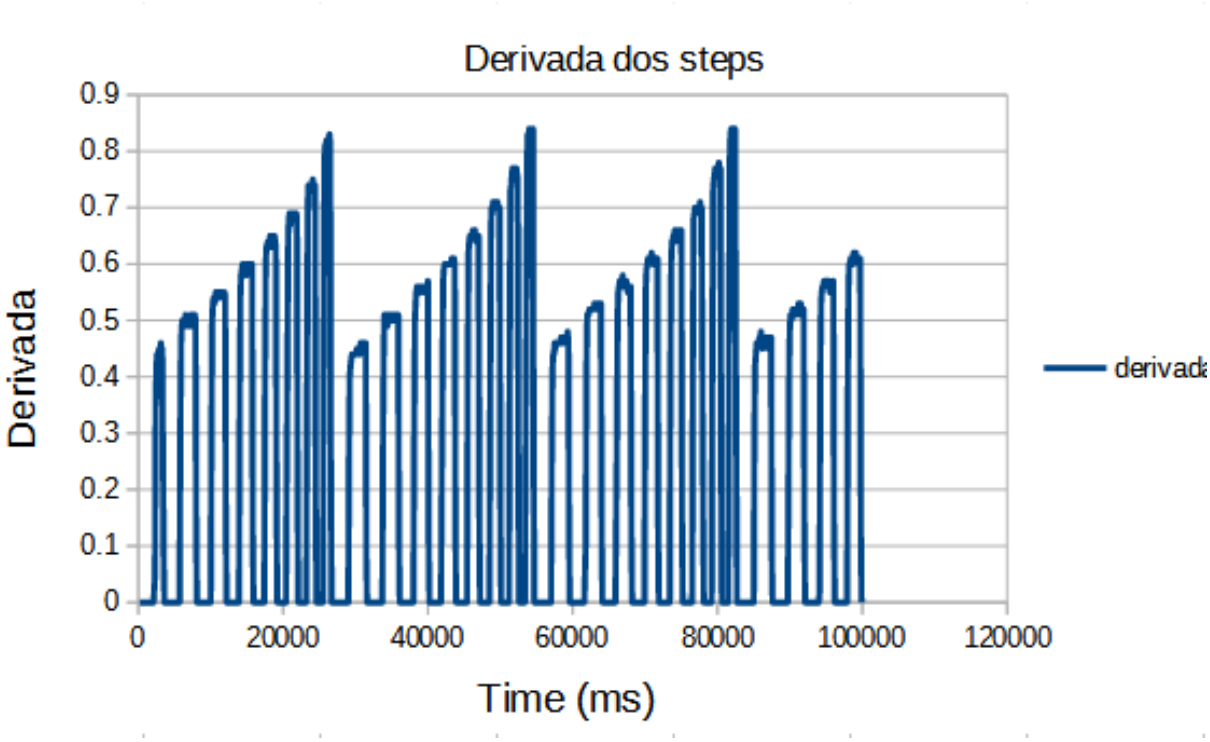


Figura 6: Derivada do sinal

Os dados do tempo, leitura do encoder e sinal de referência foram então enviados para o computador através de comunicação serial e com auxílio do Matlab (código em anexo) foi obtido a seguinte função de transferência:

$$G(z) = \frac{Y(z)}{U(z)} = \frac{0.19422\,z - 0.092392}{z^2 - 1.6576\,z + 0.65762} \tag{5}$$

Onde o período de amostragem é de $T = 100ms$, o tempo é dado em ms e o sinal de entrada e saída em $bits$.

2.3 Projeto do controlador

A planta escolhida para esse trabalho é um dispositivo tipicamente utilizado para posicionamento de cartuchos de impressoras e scanners. Esse equipamentos não tem requisitos tão rígidos quanto velocidade como servos motores de aeromodelos e possuem baixa tolerância para grandes valores de sobressinal.

2.3.1 Definição Requisitos

O desempenho do controlador será algo dentro dos limites físicos dos dispositivos envolvidos. No que inclui, a taxa de variação da posição angular não pode ser maior que $10Hz$, a precisão está bastante limitada as taxas de amostragem de 8 bits do Arduino e quantidade de buracos do disco do encoder. Tendo isto em mente foi definido como requisitos para o sistema:

- Overshoot <= 16 %;
- Tempo de assentamento para entrada degrau: 1 s;

2.4 Primeira proposta de controlador

O primeiro controlador tem seu projeto feito pelo LGR (Lugar Geométrico das Raízes). Analisa-se o LGR da planta discretizada no plano imaginário Z. Para o requisito de overshoot,

procura-se um ponto no LGR em que o fator de amortecimento esteja entre o intervalo de 0.5 e 0.7, visto a seguinte relação em 6:

$$\zeta = \frac{-\ln\left(PO\%/100\right)}{\sqrt{\pi^2 + \ln^2\left(PO\%/100\right)}} \tag{6}$$

Nesse primeiro momento procura-se atender apenas um dos objetivos, tendo como objetivo final dessa etapa encontrar um ganho dentro do LGR da planta que atenda o requisito.

Em momento posterior, analisa-se a resposta ao degrau em malha fechada do sistema para observar se o requisito de tempo de assentamento também foi alcançado utilizando apenas essa proposta.

As etapas desta análise seguem na sequência a seguir:

(*1*) Dispor pólos e zeros da planta no plano imaginário;

(*2*) Obter o LGR da planta no espaço Z;

(*3*) Delimitar conjunto de pontos no plano imaginário que corresponde a fator de amortecimento de 0.7;

(*4*) Determinar ponto de intersecção do LGR com a linha do conjunto de dados de fator de amortecimento;

Utilizando-se a função *rlocus* do Matlab (Função que gera o lugar da Raízes de uma função de transferência) e a função *zgrid* (que plota a área do plano Z no plano imaginário), presentes no Matlab, temos na figura 16 o LGR da função de transferência da planta discretizada em 5:

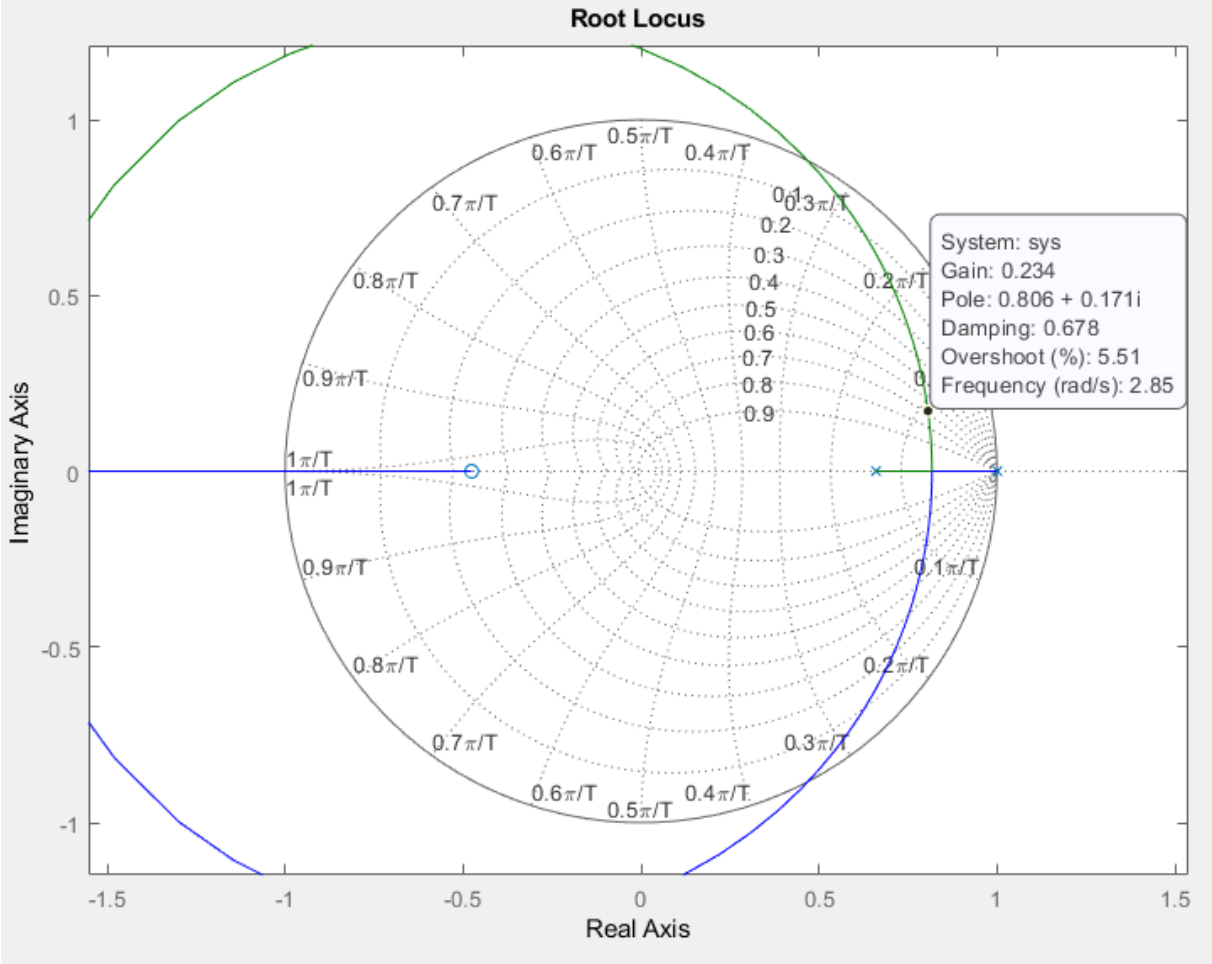


Figura 7: LGR da planta

Observa-se que o ganho para a obtenção do comportamento deseja do sistema é de 0.234. Para validação desse resultado, simulou-se o sistema em malha fechado no Simulink com segue na figura 8:

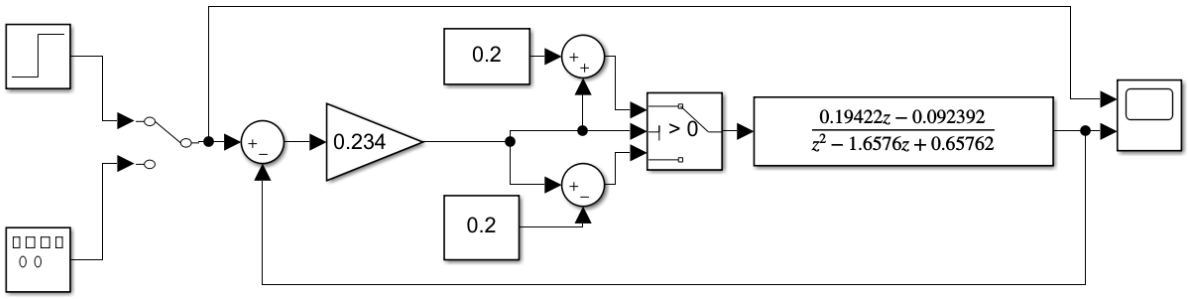


Figura 8: Diagrama da proposta 1 de controle

Os resultados obtidos estão dispostos a seguir nas figuras 9 e 10:

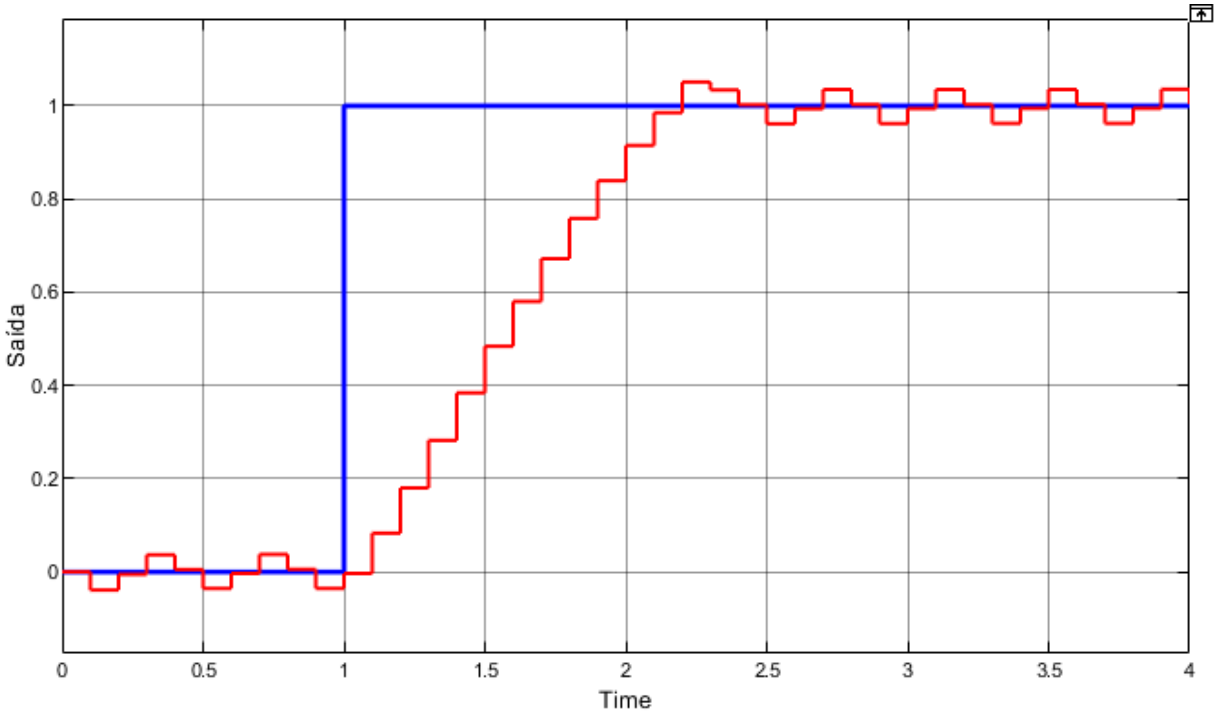


Figura 9: Gráfico da saída da proposta 1 de controle

Observa-se que o requisito de *overshoot* foi alcançado. Na gráfico da figura 10 foi aumentado a escala do gráfico a fim de ser mais preciso quanto a essa análise. O *overshoot* é aproximadamente de 5 % com valores abaixo desse valor para o trecho em que o sinal já está estabilizado em 1.

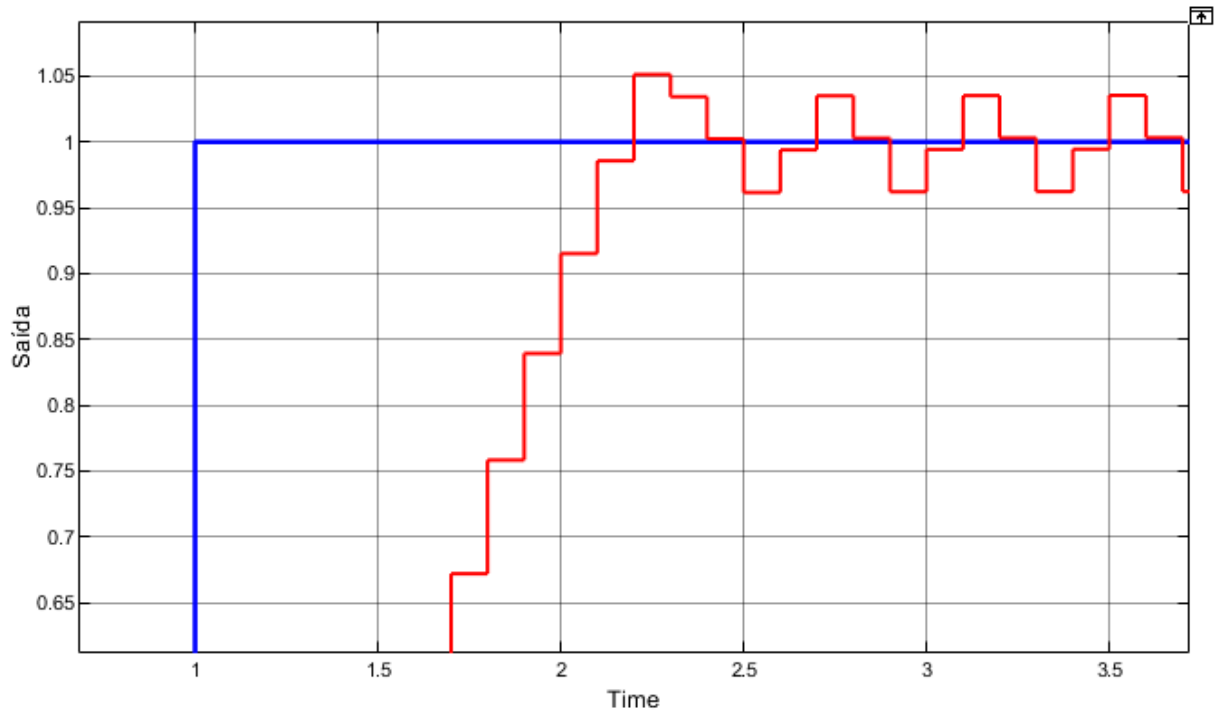


Figura 10: Aumento de escala para observação do *overshoot* da proposta 1 de controle

A implementação em arduino do primeiro controlador segue na figura 11, em que foi utilizado um ganho de 0.25. O valor 0.25 foi utilizado para ganho por permitir a implementação da operação através de um shift e com isto o cálculo teria pouco impacto no tempo do loop.

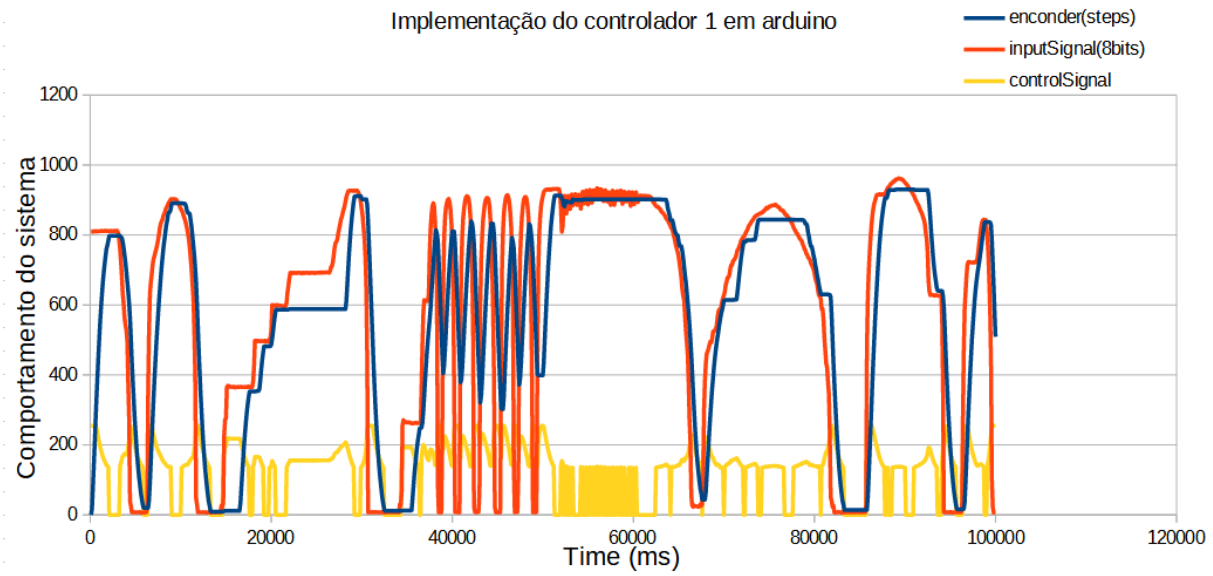


Figura 11: Implementação em arduino do controlador 1

A sinal de referência foi gerado manualmente através de um potênciometro associado ao Arduino. Pelo gráfico podemos perceber que para varições lentas a planta acompanha a referência mas sempre com um atraso e um erro estático. Para uma variação mais rápida o sistema não consegue acompanhar. Também foi notado que para uma variação pequena o erro acaba sendo maior pois o motor fica muito próximo da zona morta e com isto acaba não movimentando.

Somado a todos estes problemas e temos que o tempo de assentamento é maior que 2 segundos em simulação, mais que o dobro do tempo de assentamento do requisito de projeto. Por esse motivo, propõe-se um segundo tipo de proposta de projeto.

2.5 Segunda proposta de controlador

Nessa abordagem iremos utilizar o Diagrama de Bode. O objetivo é atender os dois requisitos realizando um compensador em avanço. Para isso, iremos realizar a transformada inversa de Z sobre a função de transferência discretizada, obtendo sua versão no domínio da frequência, como se segue em 7:

$$G(s) = \frac{0.5011(s + 70.0259)}{(s + 0.00058419)(s + 4.19042)} \tag{7}$$

Utiliza-se a estratégia de anular o polo de 4.19042 da planta e adicionar um outro polo. A escolha do novo polo irá seguir a relação 8:

$$\zeta\omega_n = \frac{4}{T_s} \tag{8}$$

Como se quer tempo de assentamento de 1 segundo, utiliza-se um fator de amortecimento menor sem deixar de se observar *overshoot*. A escolha então será de fator de amortecimento de 0.5. Tem-se então que a frequência natural para o caso é de **8 radianos por segundos**. O controlador tem a seguinte lei do formação 9:

$$G_c(s) = \frac{s + 4.19042}{s + 8} \tag{9}$$

Plota-se então o Diagrama de BODE da planta para realizar a compensação de ganho devido ao avanço do novo controlador. Analisa-se que em próximo ao polo que é anulado na planta, o ganho é de 3.46 dB.

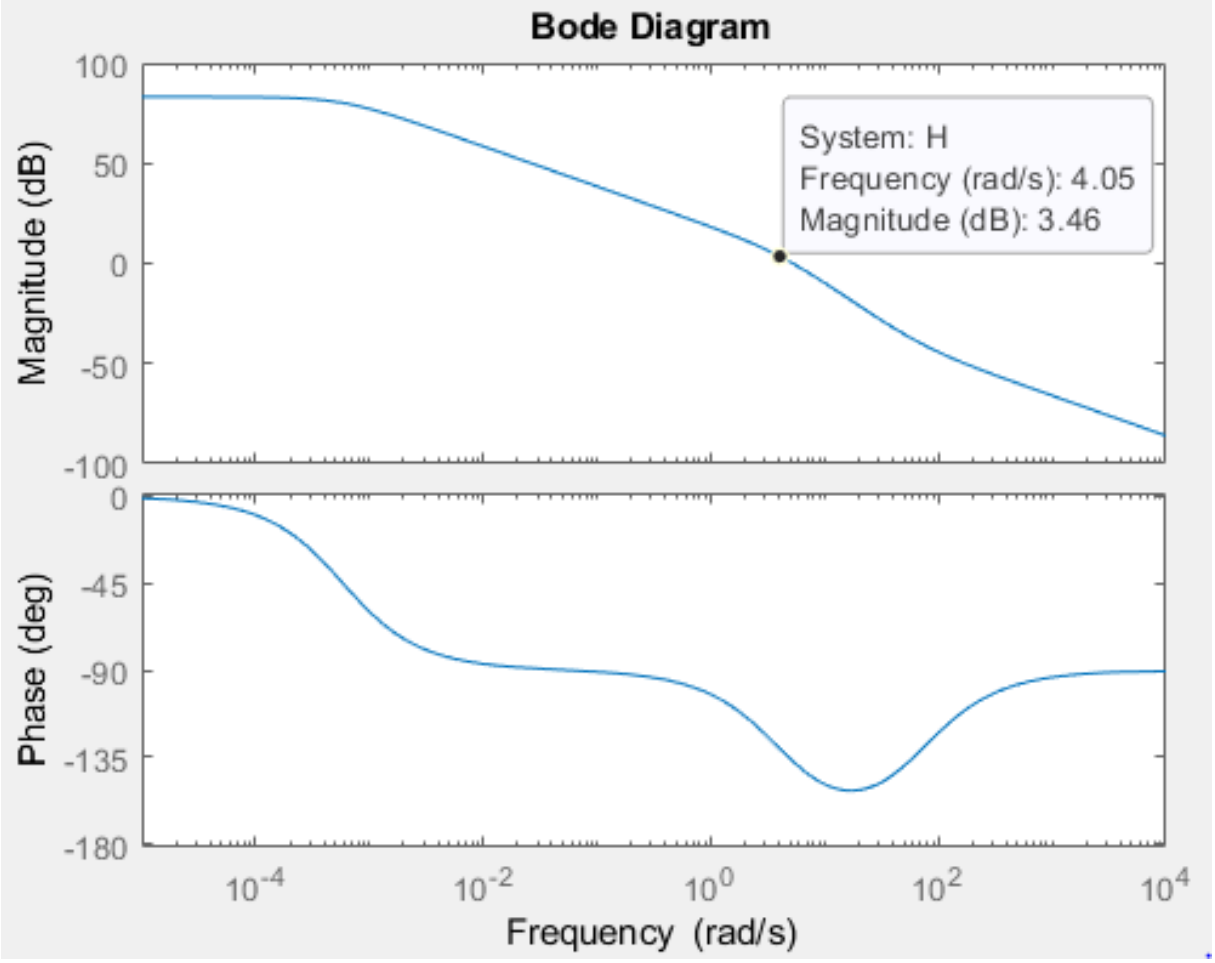


Figura 12: Diagrama da primeira proposta de controle

O controlador final fica então com a seguinte configuração (10):

$$G_c(z) = 3.46\frac{s - 0.7116}{z - 0.4493} \tag{10}$$

Implementa-se então o controlador em Simulink a fim de avaliar o alcance dos requisitos, como segue na figura 13:

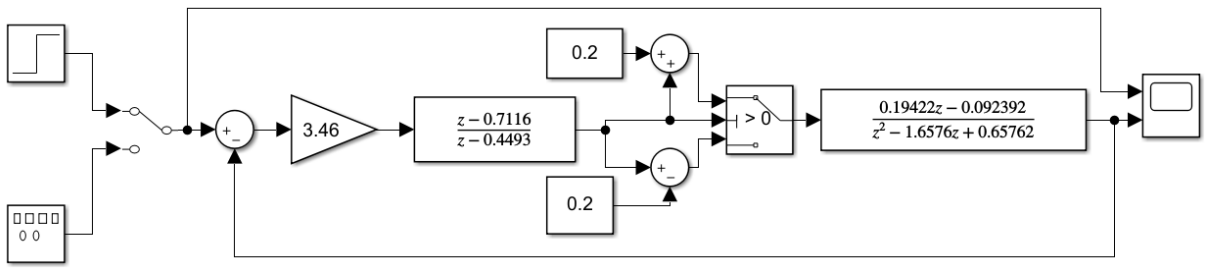


Figura 13: Diagrama da segunda proposta de controle

Observa-se na figura 14 que o requisito tempo de assentamento de 1 segundo é alcançado.

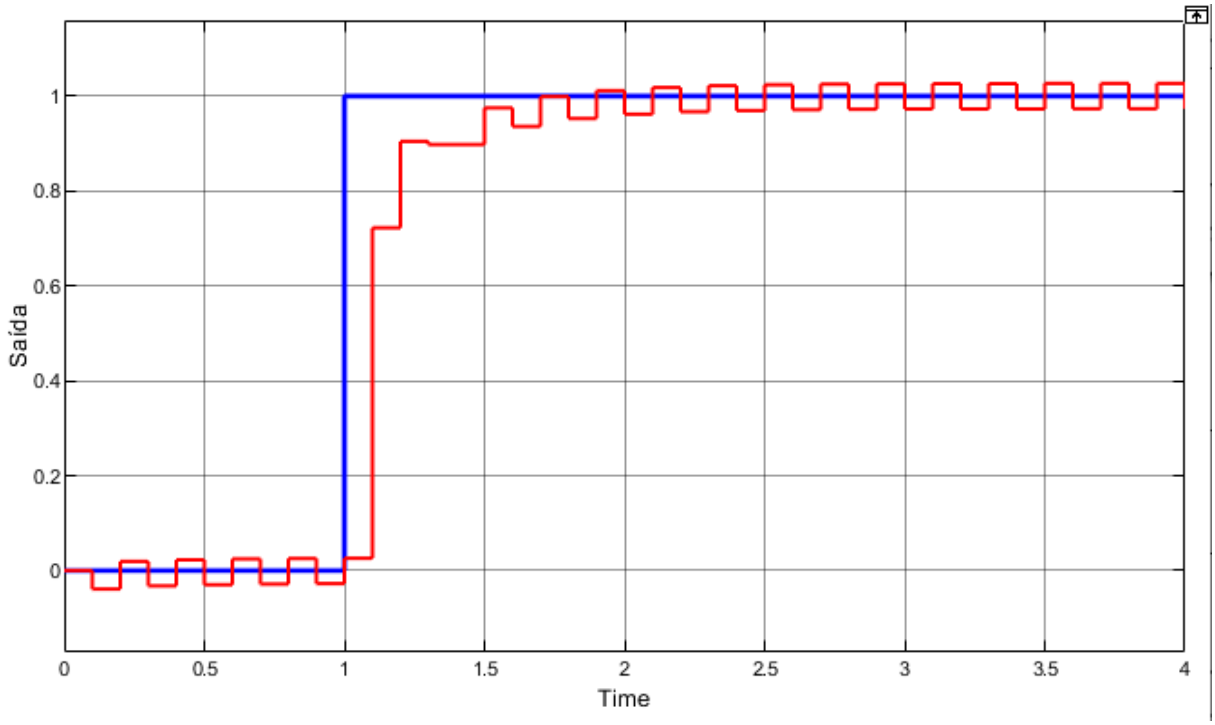


Figura 14: Resposta ao degrau da segunda proposta de controle

O *overshoot* por sua vez fica abaixo de 5 %, e esse requisito também é resolvido.

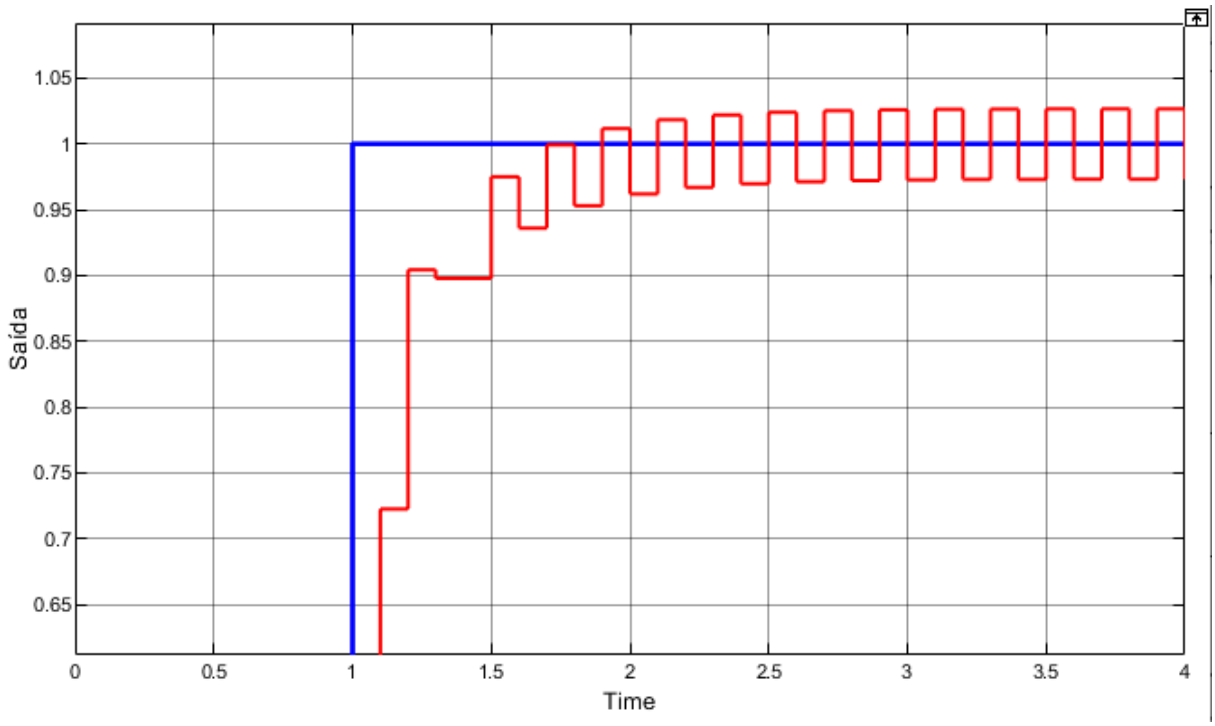


Figura 15: Overshoot da segunda proposta de controle

Por fim, realizou-se um teste de stress no sistema com um sinal square de amplitude 1, o que seria semelhante ao carro do cartucho de um impressora se deslocando durante o processo de impressão.

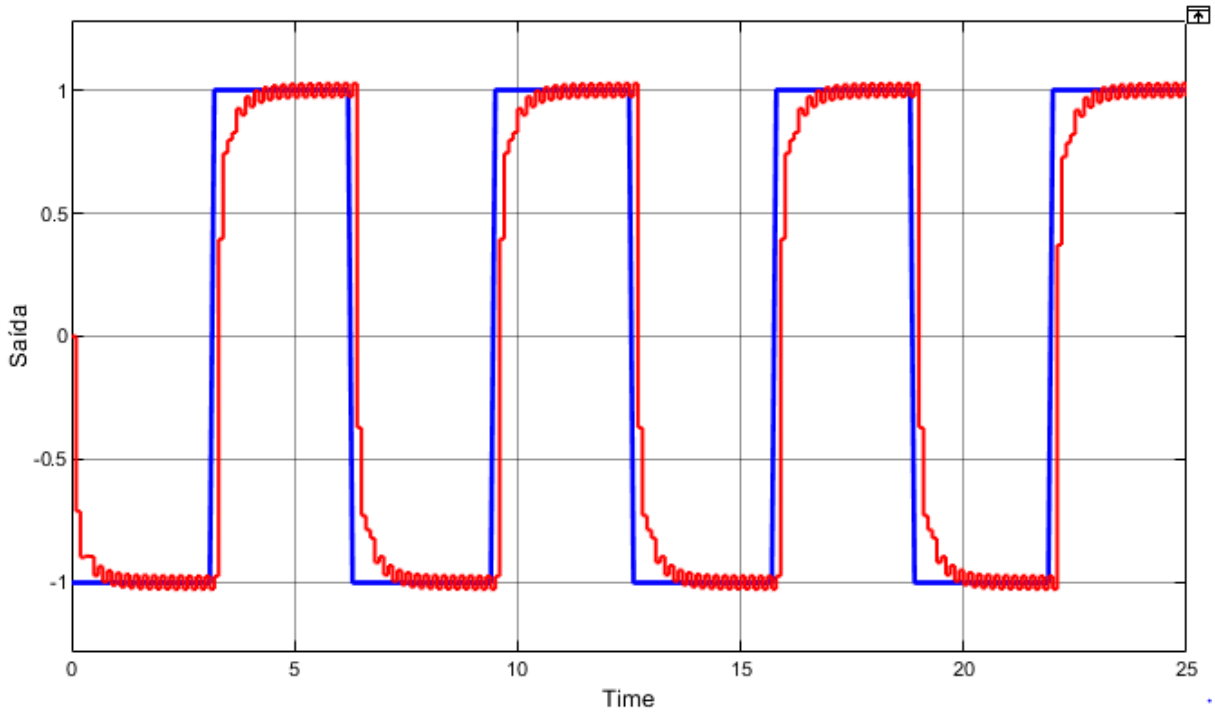


Figura 16: Resposta a squares

A função de transferência do caminho direto do sistema fica com o seguinte formato (11):

$$G(z) = (\frac{0.19422\,z - 0.092392}{z^2 - 1.6576\,z + 0.65762}).(3.46\frac{z - 0.7116}{z - 0.4493}) \tag{11}$$

Desenvolvendo-se algebricamente obtêm-se (12):

$$G(z) = \frac{Y(z)}{U(z)} = \frac{0.672001z^2 - 0.797872z + 0.227482}{z^3 - 2.1069z^2 + 1.40238z - 0.295469} \tag{12}$$

Por fim, a equação das diferenças do sistema é descrita em 13:

$$\begin{aligned} y[k] = 0.672001 \cdot x[k-1] - 0.797872 \cdot x[k-2] + 0.227482 \cdot x[k-3] \\ -2.1069 \cdot y[k-1] + 1.40238 \cdot y[k-2] - 0.295469 \cdot y[k-3] \end{aligned} \tag{13}$$

A equação das diferenças é uma descrição do sistema adequada a ser utilizada em microcontroladores digitais.

3 Conclusão

A implementação de um sistema de controle de posição a partir de Arduino trouxe vários desafios. Preparar um hardware e garantir as mínimas condições de operação em tempo real para os controladores acabou tomando mais tempo que o previsto. Durante o processo alguns motores foram danificados e também houve perda de um encoder, porém tal representou uma excelente oportunidade de aprendizado tanto em como lidar com as características de cada dispositivo usado como também permitir uma compreensão melhor da teoria de controle.

Como abordagem principal, tanto na escolha do tema dos métodos utilizados foi buscado a todo momento a integração dos conhecimentos desenvolvidos da teoria de controle de sistema a tempo discreto em conjunto os campos de programação de microcontroladores, instrumentação digital e programação em tempo real. Esta abordagem trouxe seus ônus por acrescentar etapas no projeto além da simulação pura de um sistema físico idealizado, mas contribuiu bastante no

melhor entendimento do impacto das características físicas e tipos de ruídos associados de cada parte no sistema de controle.

A partir do estudo feito, foi possível projetar um controlador que atendesse os requisitos propostos tanto para o sistema real como simulado. Com as simplificações feitas na planta, reduzindo o frequência da dinâmica do controlador e da planta foi possível simplificar bastante o projeto final, de modo que um ganho proporcional foi o suficiente para atender os requisitos mínimos de desempenho. Em continuidade a este estudo, margens mais rígidas de desempenho podem ser testadas como forma de permitir o estudo de controladores mais complexos.

Embora não apresentado aqui, o estudo feito ao longo deste projeto permitiu verificar experimentalmente de forma qualitativa o comportamento da planta em malha fechada para diversos valores de ganho. Tendo sido possível perceber as regiões com comportamento amortecido, oscilação amortecida e instabilidade. O que contribuiu bastante para a consolidação dos conhecimentos desenvolvidos na matéria.

Referências

[1] N. NISE. *Engenharia de Sistemas de Controle*. Gen | LTC, 2012.

[2] K. Ogata. *Modern Control Engineering*. Instrumentation and controls series. Prentice Hall, 2010.

[3] I. The MathWorks. *Control System Toolbox*. Natick, Massachusetts, United State, 2020.

[4] I. The MathWorks. *Symbolic Math Toolbox*. Natick, Massachusetts, United State, 2020.

Anexos

Matlab

Leitura dados Arduino

```
%% Arduino Test
function x = plotArduino(port,buffersize)

    % Connect to Arduino
    delete(instrfind({'Port'},{port}));
    serialPort = serial(port);
    serialPort.BaudRate = 9600; % High BaudRate cause buffer skipping
    fopen(serialPort);

    x = zeros(buffersize,3);
    i = 1;

    while ((~isempty(serialPort)) && strcmp(serialPort.Status,'open') && i
    ↪  <= buffersize)
        % Read Data
        str = fgetl(serialPort);
        str = strsplit(str, ' ')
        [~,n] = size(str)

        x(i,1) = str2double(str{1});
        x(i,2) = str2double(str{2});
        x(i,3) = str2double(str{3});
        x(i,4) = str2double(str{4});

        % Update Index
        i = i+1;
    end
 end
```

Identificação de Parâmetros

```
%%

% Get path from current file and generate absolute path
file_path = fileparts(mfilename('fullpath'))
data_path = strcat(file_path,"/../../data/")

% Read Exp Data
filename = strcat(data_path,"expdata_20201124_220716.csv")
fileID = fopen(filename,'r')
fgets(fileID)
data = transpose(fscanf(fileID,"%d,%d,%d",[3 Inf]))

deadzone = 125

% Split data into column vectors
t = data(:,1)
y = data(:,2)
u = data(:,3) - (data(:,3) > 0)*deadzone

% Find Period
Ts = mean(diff(data(:,1)))/1000

% Find coefficients
k = 1:2000;
A = [-y(k+1) -y(k) u(k+1) u(k)];
coeff = inv(A'*A)*A'*y(k+2)

% Symbolic Discrete Transfer Function
syms z
num = vpa(poly2sym(coeff(3:4)',z),4);
den = vpa(poly2sym([1 coeff(1:2)'],z),4);
sGz = vpa(num/den,4)

%  Discrete Transfer Function
Gz = tf(coeff(3:4)',[1 coeff(1:2)'],Ts)

Gz2 = zpk(Gz)
```

Arduino

Dado as limitações do Arduino foram produzidos diferentes códigos como forma de avaliar o funcionamento de cada componente de hardware isoladamente.

Teste Encoder

Para avaliar o funcionamento do encoder foi usado o seguinte código

```
/**
 * PID Control
 */

#define encoder0PinA 3
#define encoder0PinB 2
#define PIN_ENCODER 10

// Encoder
volatile long  encoderCount = 0;
volatile bool isRotatingCCW = true;
long  encoderCountTotal = 0;

// Timers (ms)
unsigned long currentTime;
unsigned long lastTime;
const unsigned long periodTime = 300; // For Real Systems You should try
↪  something above 200ms due to the serial communication time

/**
 * Setup Routine
 * - Run once at beginning
 */
void setup() {

    // PID - Set Input PIN
    pinMode(PIN_ENCODER, OUTPUT);


    // Encoder - Set PIN
    pinMode(encoder0PinA, INPUT_PULLUP);
    pinMode(encoder0PinB, INPUT_PULLUP);

    // Attach Interrupt signal to Encoder PIN
    attachInterrupt(digitalPinToInterrupt(encoder0PinA), isrCount,
    ↪  RISING);


    pinMode(LED_BUILTIN, OUTPUT);

    // Serial
    Serial.begin(9600);
}

```

```
43  void loop(){
44      // Control Loop Frequency Time
45      if(millis() >= lastTime + periodTime)
46      {
47        lastTime = millis(); // Reset Timer
48
49        // Read Encoder
50        noInterrupts();
51        encoderCountTotal += encoderCount;
52        encoderCount = 0;
53        interrupts();
54
55        // Serial - Plot Chart
56        Serial.print(millis());
57        Serial.print(" ");
58        Serial.println(encoderCountTotal);
59      }
60  }
61
62  /**
63   * Count Enconder Steps
64   *  - Called by Interruption
65   */
66  void isrCount()
67  {
68    // Detect Rotation Direction
69    isRotatingCCW = digitalRead(encoder0PinB);
70    digitalWrite(LED_BUILTIN,isRotatingCCW);
71    if(isRotatingCCW == HIGH)
72    {
73      encoderCount--;
74    }
75    else
76    {
77      encoderCount++;
78    }
79  }
```

Teste Motor

Para avaliar o funcionamento do motor foi usado o seguinte código

```
#include <elapsedMillis.h>

#define PINLED 13
#define PINMOTOR1 6
#define PINMOTOR2 5
#define PINMOTOR_EN 9

elapsedMillis timerLoop;
const int timeStep = 100; // ms

elapsedMillis timerGlobal;

int speedMotor,speedStep;

void setup() {

  Serial.begin(9600);

  pinMode(PINLED, OUTPUT);
  pinMode(PINMOTOR1, OUTPUT);
  pinMode(PINMOTOR2, OUTPUT);
  pinMode(PINMOTOR_EN, OUTPUT);

  digitalWrite(PINMOTOR_EN,HIGH);
  analogWrite(PINMOTOR1,0);
  analogWrite(PINMOTOR2,0);

  timerGlobal = 0;
}

void loop() {
  if( timerLoop >= timeStep ) // Control Frequency time
  {
    // Reset Timer
    timerLoop = 0;

    speedStep = (speedMotor >=255)?-1:(speedMotor < 10) ?1:speedStep;
    speedMotor +=speedStep;

    analogWrite(PINMOTOR1,0);
    analogWrite(PINMOTOR2,speedMotor);

    //
    Serial.print(timerGlobal);
    Serial.print(" ");
    Serial.println(speedMotor);
  }
}
```

Teste Motor

O código usado para caracterização da zona morta do sistema e avaliação do funcionamento conjunto do motor com o encoder foi usado o seguinte:

```
#include <elapsedMillis.h>

#define PINLED 13
#define PINMOTOR1 6
#define PINMOTOR2 5
#define PINMOTOR_EN 9

#define PINENCODER_A 2
#define PINENCODER_B 3

elapsedMillis timerGlobal;
elapsedMillis timerLoop;
const int timeStep = 60; // ms

// Motor
int speedMotor,speedStep;

// Encoder
volatile long  encoderCount = 0;
volatile bool isRotatingCCW = true;
long  encoderCountTotal = 0;

void setup() {

  Serial.begin(9600);

  pinMode(PINLED, OUTPUT);
  pinMode(PINMOTOR1, OUTPUT);
  pinMode(PINMOTOR2, OUTPUT);
  pinMode(PINMOTOR_EN, OUTPUT);

  digitalWrite(PINMOTOR_EN,HIGH);
  analogWrite(PINMOTOR1,0);
  analogWrite(PINMOTOR2,0);

  // Encoder - Set PIN
  pinMode(PINENCODER_A, INPUT_PULLUP);
  pinMode(PINENCODER_B, INPUT_PULLUP);

  // Attach Interrupt signal to Encoder PIN
  attachInterrupt(digitalPinToInterrupt(PINENCODER_A), isrCount, RISING);

  speedStep = 1;

  timerGlobal = 0;
}

```

```
48  void loop() {
49    if( timerLoop >= timeStep ) // Control Frequency time
50    {
51      // Reset Timer
52      timerLoop = 0;
53
54      // Read Encoder
55      noInterrupts();
56      encoderCountTotal += encoderCount;
57      encoderCount = 0;
58      interrupts();
59
60      // Control Signal
61      if ( speedMotor >= 255 )
62      {
63        speedStep = -1;
64      }
65      else if ( speedMotor <= -255 )
66      {
67        speedStep = 1;
68      }
69      speedMotor += speedStep;
70
71
72      if ( speedMotor > 0)
73      {
74        // Set Motor Speed
75        digitalWrite(PINMOTOR1,LOW);
76        analogWrite(PINMOTOR2,speedMotor);
77      }
78      else if ( speedMotor < 0 )
79      {
80        // Set Motor Speed
81        analogWrite(PINMOTOR1,-speedMotor);
82        digitalWrite(PINMOTOR2,LOW);
83      }
84      else
85      {
86        digitalWrite(PINMOTOR1,LOW);
87        digitalWrite(PINMOTOR2,LOW);
88      }
89
90      // Output
91      Serial.print(timerGlobal);
92      Serial.print(",");
93      Serial.print(encoderCountTotal);
94      Serial.print(",");
95      Serial.println(speedMotor);
96    }
97  }
98
```

```
99
100 /**
101  * Count Enconder Steps
102  *  - Called by Interruption
103  */
104 void isrCount()
105 {
106   // Detect Rotation Direction
107   isRotatingCCW = digitalRead(PINENCODER_B);
108   if(isRotatingCCW == HIGH)
109   {
110     encoderCount--;
111   }
112   else
113   {
114     encoderCount++;
115   }
116 }
```

Identificação da planta

O código usado para identificação dos parâmetros do planta:

```
#include <elapsedMillis.h>

#define PINLED 13
#define PINMOTOR1 6
#define PINMOTOR2 5
#define PINMOTOR_EN 9

#define PINENCODER_A 2
#define PINENCODER_B 3

#define DEADZONE 180

elapsedMillis timerGlobal;
elapsedMillis timerLoop;
elapsedMillis timerMotorOn;
const int timeStep = 100; // ms
int timeMotorOn = 10*timeStep; // ms

bool isMotorON = false;
uint8_t speed = DEADZONE;
uint8_t stepSpeed = 5;
uint8_t counter = 0;

// Encoder
volatile long  encoderCount = 0;
volatile bool isRotatingCCW = true;
long  encoderCountTotal = 0;

void setup() {

  Serial.begin(9600);

  pinMode(PINLED, OUTPUT);
  pinMode(PINMOTOR1, OUTPUT);
  pinMode(PINMOTOR2, OUTPUT);
  pinMode(PINMOTOR_EN, OUTPUT);

  digitalWrite(PINMOTOR_EN,HIGH);
  analogWrite(PINMOTOR1,0);
  analogWrite(PINMOTOR2,0);

  // Encoder - Set PIN
  pinMode(PINENCODER_A, INPUT_PULLUP);
  pinMode(PINENCODER_B, INPUT_PULLUP);

  // Attach Interrupt signal to Encoder PIN
  attachInterrupt(digitalPinToInterrupt(PINENCODER_A), isrCount, RISING);

```

```
49    timerMotorOn = 0;
50    timerGlobal = 0;
51  }
52
53  void loop() {
54    if( timerLoop >= timeStep ) // Control Frequency time
55    {
56      // Reset Timer
57      timerLoop = 0;
58
59      // Read Encoder
60      noInterrupts();
61      encoderCountTotal += encoderCount;
62      encoderCount = 0;
63      interrupts();
64
65      if (timerMotorOn >= timeMotorOn )
66      {
67        timerMotorOn = 0;
68        isMotorON = !isMotorON;
69        counter = (counter + 1) & 15;
70        speed = counter*stepSpeed + DEADZONE;
71        timeMotorOn = (25-counter)*timeStep;
72      }
73
74      if ( isMotorON )
75      {
76        // Set Motor Speed
77        digitalWrite(PINMOTOR2,LOW);
78        analogWrite(PINMOTOR1,speed);
79      }
80      else
81      {
82        digitalWrite(PINMOTOR1,LOW);
83        digitalWrite(PINMOTOR2,LOW);
84      }
85
86      // Output
87      Serial.print(timerGlobal);
88      Serial.print(",");
89      Serial.print(encoderCountTotal);
90      Serial.print(",");
91      Serial.println( speed * isMotorON );
92    }
93  }
94
95
96  /**
97   * Count Enconder Steps
98   *  - Called by Interruption
99   */
```

```
100 void isrCount()
101 {
102   // Detect Rotation Direction
103   isRotatingCCW = digitalRead(PINENCODER_B);
104   if(isRotatingCCW == HIGH)
105   {
106     encoderCount--;
107   }
108   else
109   {
110     encoderCount++;
111   }
112 }
```

Controlador 1

Para implementação do controlador P foi usado o seguinte código

```
/**
 * PID Control - Tinkercad
 */

#define PWM_PIN 9
#define MOTOR_EN1_PIN  6
#define MOTOR_EN2_PIN  5
#define ENCODER_PINA 3
#define ENCODER_PINB 2

#define PIN_INPUT_P A0
#define PIN_INPUT_I A1
#define PIN_INPUT_D A2

#define PIN_INPUT_REF A3

#define ERRO_MIN 5

#define DEADZONE 130

// Encoder
volatile long  encoderCount = 0;
volatile bool isRotatingCCW = true;
long  encoderCountTotal = 0;

// Motor
int motorSpeed = 0;   // Set Motor Speed
bool rotateCCW;       // Flag for Rotation Direction

// Control Internal Constants
int cntrlP = 0;
int cntrlI = 0;
int cntrlD = 0;
int cntrlSignal = 0;
int error,lastError = 0;

// Timers (ms)
unsigned long currentTime;
unsigned long lastTime;
const unsigned long periodTime = 100;

/**
 * Setup Routine
 * - Run once at beginning
 */
void setup() {

    // PID - Set Input PIN
```

```
49      pinMode(PIN_INPUT_P, INPUT_PULLUP);
50      pinMode(PIN_INPUT_I, INPUT_PULLUP);
51            pinMode(PIN_INPUT_D, INPUT_PULLUP);
52
53      // Reference - Set Input PIN
54      pinMode(PIN_INPUT_REF, INPUT_PULLUP);
55
56      // Encoder - Set PIN
57            pinMode(ENCODER_PINA, INPUT_PULLUP);
58            pinMode(ENCODER_PINB, INPUT_PULLUP);
59
60            // Attach Interrupt signal to Encoder PIN
61      attachInterrupt(digitalPinToInterrupt(ENCODER_PINA), isrCount,
        ↪  RISING);
62
63      // Motor
64            pinMode(MOTOR_EN1_PIN, OUTPUT);
65            pinMode(MOTOR_EN2_PIN, OUTPUT);
66
67            // Serial
68      Serial.begin(9600);
69  }
70
71  void loop(){
72      // Control Loop Frequency Time
73            if(millis() >= lastTime + periodTime)
74      {
75        lastTime = millis(); // Reset Timer
76
77        // Read Input (Range: 0-1023)
78        int inputRead = analogRead(PIN_INPUT_REF);
79        int inputP = 10;//analogRead(PIN_INPUT_P);
80        int inputI = 0;//analogRead(PIN_INPUT_I);
81        int inputD = 0;//analogRead(PIN_INPUT_D);
82
83        // Read Encoder
84        noInterrupts();
85        encoderCountTotal += encoderCount;
86        encoderCount = 0;
87        interrupts();
88
89        // PID Control action
90        lastError = error;
91        error = inputRead - encoderCountTotal;
92        cntrlSignal = error >> 2;
93
94        if( cntrlSignal >= ERRO_MIN )
95        {
96          rotateCCW = LOW;
97          motorSpeed = DEADZONE + cntrlSignal;
98        }
```

```
99        else if( cntrlSignal <= -ERRO_MIN)
100       {
101         rotateCCW = HIGH;
102         motorSpeed = DEADZONE -cntrlSignal;
103       }
104       else
105       {
106         rotateCCW = LOW;
107         motorSpeed = 0;
108       }
109
110       motorSpeed = (motorSpeed > 255)?255:motorSpeed;
111
112       // Serial - Plot Chart
113       Serial.print(millis());
114       Serial.print(" ");
115       Serial.print(encoderCountTotal);
116       Serial.print(" ");
117       Serial.print(inputRead);
118       Serial.print(" ");
119       Serial.println(motorSpeed);
120
121       // Control Motor
122       digitalWrite(MOTOR_EN1_PIN, rotateCCW);
123       digitalWrite(MOTOR_EN2_PIN, !rotateCCW);
124       analogWrite(PWM_PIN, motorSpeed); // Range 0-255
125     }
126 }
127
128 /**
129  * Count Enconder Steps
130  *  - Called by Interruption
131  */
132 void isrCount()
133 {
134   // Detect Rotation Direction
135   isRotatingCCW = digitalRead(ENCODER_PINB);
136   if(isRotatingCCW == HIGH)
137   {
138     encoderCount--;
139   }
140   else
141   {
142     encoderCount++;
143   }
144 }
```

Controlador 2

Para implementação do controlador 2 foi usado o seguinte código

```
/**
 * PID Control - Tinkercad
 */

#define PWM_PIN 9
#define MOTOR_EN1_PIN  6
#define MOTOR_EN2_PIN  5
#define ENCODER_PINA 3
#define ENCODER_PINB 2

#define PIN_INPUT_P A0
#define PIN_INPUT_I A1
#define PIN_INPUT_D A2

#define PIN_INPUT_REF A3

#define ERRO_MIN 5

#define DEADZONE 130

// Encoder
volatile long  encoderCount = 0;
volatile bool isRotatingCCW = true;
long  encoderCountTotal = 0;

// Motor
int motorSpeed = 0;   // Set Motor Speed
bool rotateCCW;       // Flag for Rotation Direction

// Control Internal Constants
int cntrlP = 0;
int cntrlI = 0;
int cntrlD = 0;
float cntrlSignal = 0;
int error,lastError = 0;

unsigned long k = 0;
float bufferX[4];
float bufferY[4];

// Timers (ms)
unsigned long currentTime;
unsigned long lastTime;
const unsigned long periodTime = 100;

/**
 * Setup Routine
 * - Run once at beginning
```

```
 */
void setup() {

    // PID - Set Input PIN
    pinMode(PIN_INPUT_P, INPUT_PULLUP);
    pinMode(PIN_INPUT_I, INPUT_PULLUP);
          pinMode(PIN_INPUT_D, INPUT_PULLUP);

    // Reference - Set Input PIN
    pinMode(PIN_INPUT_REF, INPUT_PULLUP);

    // Encoder - Set PIN
          pinMode(ENCODER_PINA, INPUT_PULLUP);
          pinMode(ENCODER_PINB, INPUT_PULLUP);

          // Attach Interrupt signal to Encoder PIN
    attachInterrupt(digitalPinToInterrupt(ENCODER_PINA), isrCount,
    ↪  RISING);

    // Motor
          pinMode(MOTOR_EN1_PIN, OUTPUT);
          pinMode(MOTOR_EN2_PIN, OUTPUT);

    bufferX[3] = 0;
    bufferX[2] = 0;
    bufferX[1] = 0;
    bufferX[0] = 0;
    bufferY[3] = 0;
    bufferY[2] = 0;
    bufferY[1] = 0;
    bufferY[0] = 0;

          // Serial
    Serial.begin(9600);
}

void loop(){
    // Control Loop Frequency Time
          if(millis() >= lastTime + periodTime)
    {
      lastTime = millis(); // Reset Timer

      // Read Encoder
      noInterrupts();
      encoderCountTotal += encoderCount;
      encoderCount = 0;
      interrupts();

      // Read Input (Range: 0-1023)
      int inputRead = analogRead(PIN_INPUT_REF);;

```

```
99        // Rotate Buffer
100       bufferX[3] = bufferX[2];
101       bufferX[2] = bufferX[1];
102       bufferX[1] = bufferX[0];
103       bufferX[0] = inputRead;
104
105       cntrlSignal = 0.672001*bufferX[1] - 0.797872*bufferX[2] + 0.227482 *
          ↪  bufferX[3] -2.1069 * bufferY[1] + 1.40238 * bufferY[2] -
          ↪  0.295469 * bufferY[3];
106
107       // Control Buffer
108       bufferY[3] = bufferY[2];
109       bufferY[2] = bufferY[1];
110       bufferY[1] = bufferY[0];
111       bufferY[0] = cntrlSignal;
112
113       if( cntrlSignal >= ERRO_MIN )
114       {
115         rotateCCW = LOW;
116         motorSpeed = DEADZONE + cntrlSignal;
117       }
118       else if( cntrlSignal <= -ERRO_MIN)
119       {
120         rotateCCW = HIGH;
121         motorSpeed = DEADZONE -cntrlSignal;
122       }
123       else
124       {
125         rotateCCW = LOW;
126         motorSpeed = 0;
127       }
128
129       motorSpeed = (motorSpeed > 255)?255:motorSpeed;
130
131       // Serial - Plot Chart
132       Serial.print(millis());
133       Serial.print(" ");
134       Serial.print(encoderCountTotal);
135       Serial.print(" ");
136       Serial.print(inputRead);
137       Serial.print(" ");
138       Serial.println(motorSpeed);
139
140       // Control Motor
141       digitalWrite(MOTOR_EN1_PIN, rotateCCW);
142       digitalWrite(MOTOR_EN2_PIN, !rotateCCW);
143       analogWrite(PWM_PIN, motorSpeed); // Range 0-255
144     }
145 }
146
147 /**
```

```
148   * Count Enconder Steps
149   *  - Called by Interruption
150   */
151  void isrCount()
152  {
153    // Detect Rotation Direction
154    isRotatingCCW = digitalRead(ENCODER_PINB);
155    if(isRotatingCCW == HIGH)
156    {
157      encoderCount--;
158    }
159    else
160    {
161      encoderCount++;
162    }
163  }
```