Universidade Federal de São Carlos

Projeto do robô cartesiano no eixo Z

Relatório Final Disciplina de Controle e Servomecanismo

Professor Orides Morandin Jr.

Amanda Oliveira, RA: 727331
Fernando Miguelão, RA:727336
Gustavo de Souza, RA:626082
Matheus Kirstus, RA: 628310
Rafael Silva, RA: 595047
Vinicius Oliveira, RA: 628263

2019

Sumário

Introdução **4**

Análise de sensibilidade da função de transferência **5**
Função de Transferência 5
Testes com as variáveis de inércia e atrito 7

Simulação do PID e análise das variáveis Kp, Ki, e Kd **8**
PID e suas constantes 8
Criação do diagrama de blocos no SciLab 8
Avaliação do Kp, Kd e Ki 8

Projeto mecânico **10**
Componentes e Montagem 10
Melhorias 10

Projeto eletrônico **12**
Arquitetura do projeto eletrônico 12
Circuito de contagem do Encoder 12
Driver de potência do Motor 14
Sensores 14
Motor 16
Acoplador Ótico 16
Arduino 17
Fonte 27

Definição de parâmetros de desempenho **19**
Descrição dos parâmetros de desempenho 19
Sensibilidade dos parâmetros ao PID 20

Definição e execução do conjunto de testes **21**
Mecânica 21
Eletrônica 21
Software 22

Projeto de software **23**
Arquitetura de Software 23
Software de Controle do Contador 26
Software Controlador PID 30
Software de Controle do Motor 30
Software de Tratamento de Interrupções 30

Parte específica - RTOS **32**
Introdução ao RTOS 32
Facilidades e dificuldades na utilização de RTOS 32
Etapas do projeto 32
Modificações na arquitetura eletrônica 32

Conclusão **33**
Referências **33**

Lista de figuras

Figura 1 - Esquema eletromecânico do motor DC.

Figura 2 - Diagrama de blocos da função de transferência.

Figura 3 - Resultados dos testes. A variável avaliada é a Inércia.

Figura 4 - Resultado dos testes. A variável avaliada é o Atrito.

Figura 5 - O PID no SciLab.

Figura 6 - Efeito do Kp no sinal.

Figura 7 - Efeito do Ki no sinal. Kp foi mantido constante.

Figura 8 - Efeito do Kd no sinal. Kp e Ki foram mantidos constantes.

Figura 9 - Arquitetura eletrônica do projeto.

Figura 10 - Os pinos do CI 74193.

Figura 11 - O circuito do contador.

Figura 12 - A ponte H.

Figura 13 - O encoder.

Figura 14 - O sensor indutivo.

Figura 15 - O motor.

Figura 16 - Arduino Mega 2560.

Figura 17 - Os parâmetros de desempenho de um sinal.

Figura 18 - Módulos do sistema.

Figura 19 - Módulos do sistema com maiores detalhes.

Figura 20 - Diagrama de classes do sistema.

Figura 21 - Sequência de execução e fluxo de dados.

Figura 22 - Camadas do sistema.

Figura 23 - Diagrama esquemático do software de controle do contador.

Lista de tabelas

Tabela 1 - Alterações dos parâmetros do sinal mediante variação de Kp, Ki e Kd.

1. Introdução

O projeto proposto durante a execução desta matéria foi o projeto mecânico e eletrônico do eixo vertical de um robô cartesiano, juntamente com o desenvolvimento do software de ajuste de posição da carga utilizando controlador proporcional integral derivativo. Esse projeto foi planejado de modo que os resultados obtidos devem ser futuramente acrescidos de elementos necessários para o controle de posição nas coordenadas X e Y, a fim de completar o desenvolvimento do robô cartesiano. Como produto *standalone,* o funcionamento desse projeto se assemelha ao de um pequeno elevador de precisão que pode elevar ou descender objetos de pequenas massas.

Nos tópicos a seguir, será detalhado os pormenores do projeto, desde a sua parte mecânica e até a eletrônica e de software.

2. Análise de sensibilidade da função de transferência

Para que seja possível realizar o controle preciso do deslocamento no eixo é necessário entender o funcionamento de um motor, tanto em seus aspectos elétricos quanto mecânicos.

O motor utilizado para atuar no sistema é um motor de corrente contínua com armadura simples. Como é de se esperar de um componente mecânico, ao tentar aplicar uma força para movê-lo, encontraremos um momento de inércia e um atrito se opondo ao movimento. No caso o movimento aplicado será o de rotação, e temos T representando o torque aplicado, J representando o momento de inércia no rotor da máquina e B representando o atrito viscoso entre o motor e o eixo. Esses dois valores contribuem para a inércia do sistema com a seguinte componente:

$$\tau_i(t) = B\omega(t) + J\frac{d\omega(t)}{dt}$$

Já na parte elétrica temos o circuito da armadura, composto por uma resistência e uma indutância em série, denotados por R_a e L_a respectivamente, e uma tensão proporcional à velocidade angular do motor, denotada por e_b na Figura 1, vista a seguir. Quando uma tensão e_a é aplicada no circuito da armadura, é gerada uma corrente i_a neste circuito, e o campo magnético gerado resulta na rotação do motor. Na figura abaixo podemos ver todas as variáveis envolvidas nesse sistema eletromecânico.

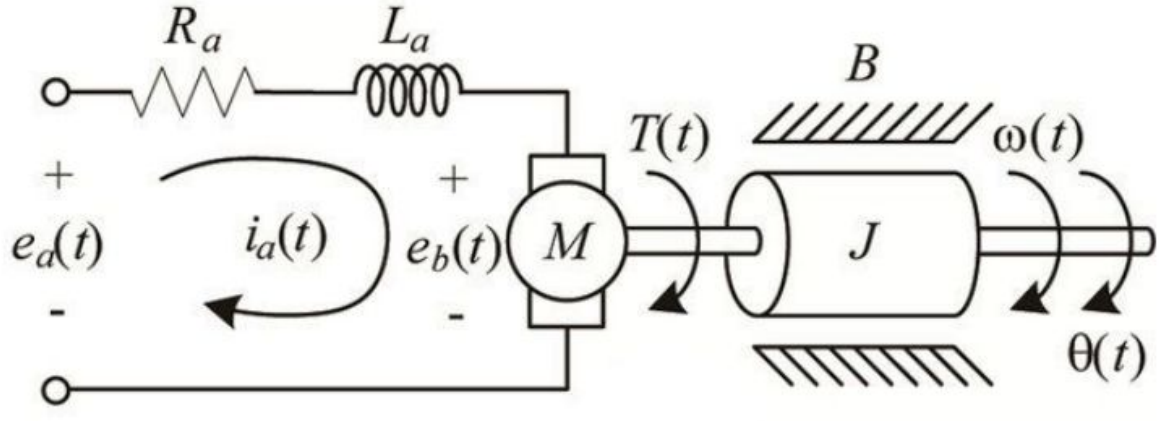

Figura 1 - Esquema eletromecânico do motor DC.

2.1. Função de Transferência

A partir do entendimento básico dos parâmetros envolvidos no sistema é possível obter a função de transferência realizando manipulações matemáticas.

Aplicando a Lei de Faraday e a Lei das Tensões de Kirchoff na malha da figura acima, obtemos as seguintes expressões:

$$e_a(t) = R_a \cdot i_a(t) + L_a \frac{di_a(t)}{dt} + e_b(t)$$

$$e_b(t) = K_{a'} \cdot \omega(t)\Phi$$

E considerando o fluxo magnético constante, temos que:

$$e_b(t) = K_2 \cdot \omega(t)$$

Onde eb é chamada força eletromotriz e K1 é uma constante que engloba o fluxo magnético. Para manipular os parâmetros mecânicos nas mesmas equações que os parâmetros elétricos, é útil lembrar da relação que surge quando a armadura é percorrida por uma corrente, gerando um campo magnético. As linhas desse campo magnético são cortadas pelo motor, o que faz aparecer um torque, descrito na equação abaixo, onde K1 é a constante de torque do motor:

$$\tau_m(t) = K_1 \cdot i_a(t)$$

Usando a equação de torque descrita anteriormente, e partindo da relação que a soma dos torques do sistema é igual ao torque gerado pelo motor, temos também a seguinte equação, onde τ_d é o torque devido à carga:

$$\tau_m(t) = \tau_d(t) + B\omega(t) + J\frac{d\omega(t)}{dt}$$

À partir das quatro equações obtidas para o sistema eletromecânico, podemos modelar a função de transferência do motor, a fim de controlar posição e velocidade. Na Figura 2 encontra-se o diagrama de blocos da função de transferência obtida ao considerar todas as equações no domínio de frequência, após aplicar a Transformada de Laplace.

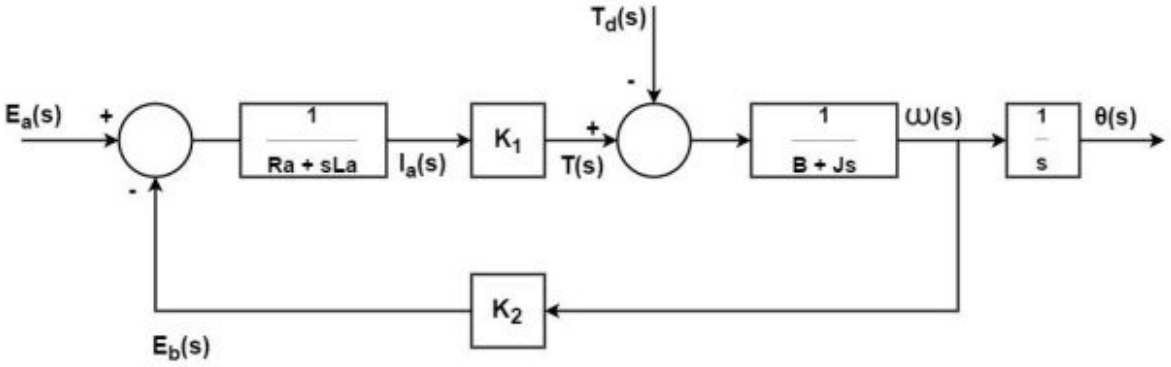

Figura 2 - Diagrama de blocos da função de transferência.

Cada bloco da figura acima representa uma etapa da função de transferência, em ordem: Circuito elétrico da armadura, torque gerado no motor pela corrente ia, parte mecânica e características do motor, e por último a obtenção da posição angular em radianos, também no domínio de frequência.

2.2. Testes com as variáveis de inércia e atrito

Na Figura 3 abaixo, temos os resultados dos testes com a variável de inércia(J). É possível perceber que ao aumentar essa variável, o tempo de resposta também aumenta de forma diretamente proporcional à essa inércia. Ao dobrarmos J, o tempo de resposta também será dobrado.

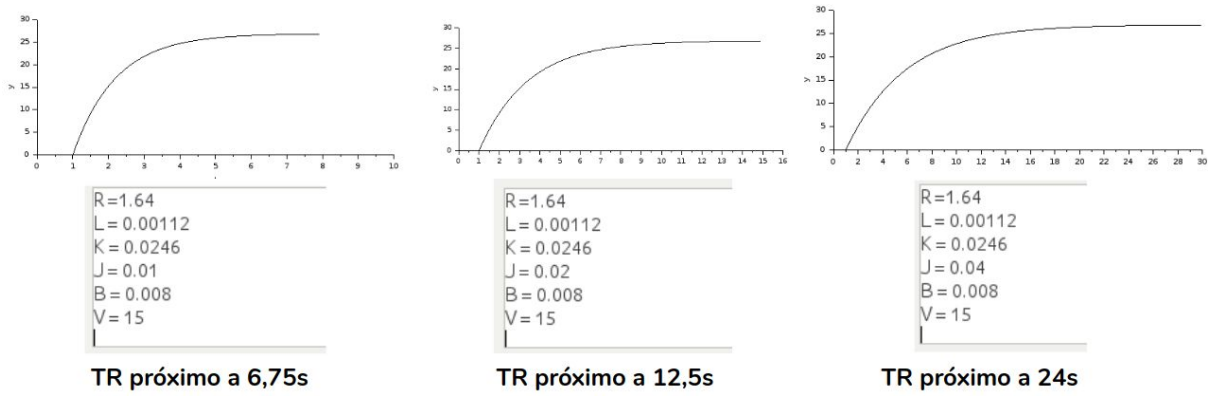

Figura 3 - Resultados dos testes. A variável avaliada é a Inércia.

Os testes foram repetidos, mas agora com a variável de atrito(B) sendo o foco. Seus resultados estão na Figura 4, a seguir.É possível perceber que com essa variável, a relação entre B e o valor final de regime são inversamente proporcionais. Ao dobrarmos B, o valor final de regime cai pela metade.

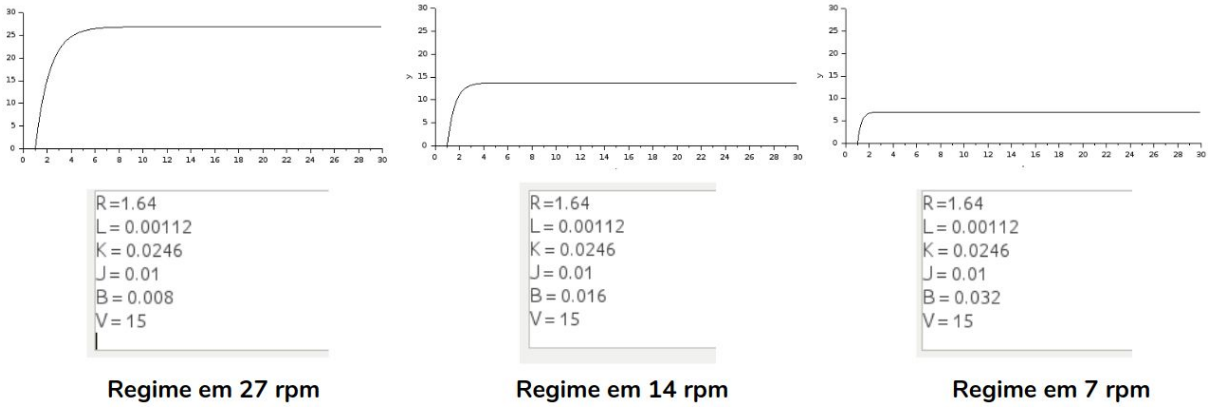

Figura 4 - Resultado dos testes. A variável avaliada é o Atrito.

3. Simulação do PID e análise das variáveis Kp, Ki, e Kd

3.1. PID e suas constantes

Para um melhor entendimento do projeto, foram feitas análises das variáveis Kp, Ki e Kd do PID para verificar como elas influenciam no sinal de saída dos sistema e nos parâmetros de desempenho dele.

3.2. Criação do diagrama de blocos no SciLab

Para ser feita essa análise, foi utilizado o programa SciLab, onde foi adicionado um bloco de PID na estrutura do circuito, mostrado abaixo na Figura 5.

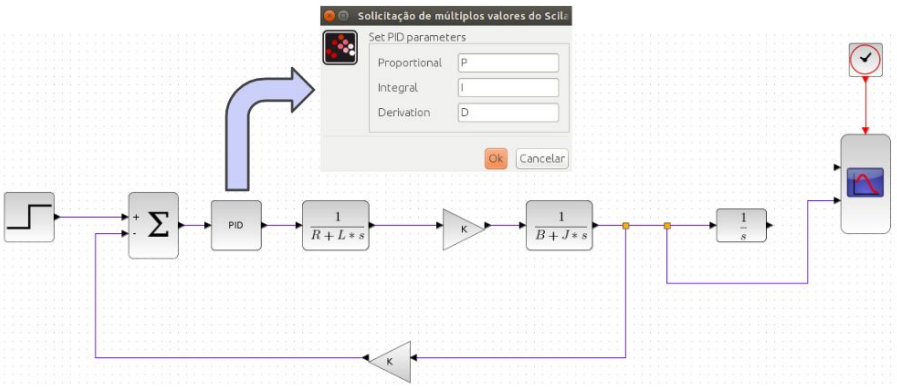

Figura 5 - O PID no SciLab.

Para entender como cada variável afeta o sistema, foram feitos experimentos parecidos em relação aos parâmetros de desempenho. Como em um motor real os componentes mecânicos influenciam muito mais do que os componentes elétricos, a curva de saída do sistema comporta-se como um sistema de primeira ordem, não acontecendo um sobressinal.

Para ser possível analisar a influência, foi necessário modificar os valores de R e L para que ficassem próximos dos valores de B e J, "forçando" o sistema a se comportar como um sistema de segunda ordem.

3.3. Avaliação de Kp, Kd e Ki

Nas próximas figuras, Figuras 6, 7 e 8, temos as alterações do sinal de acordo com a variação das variáveis Kp, Kd e Ki.

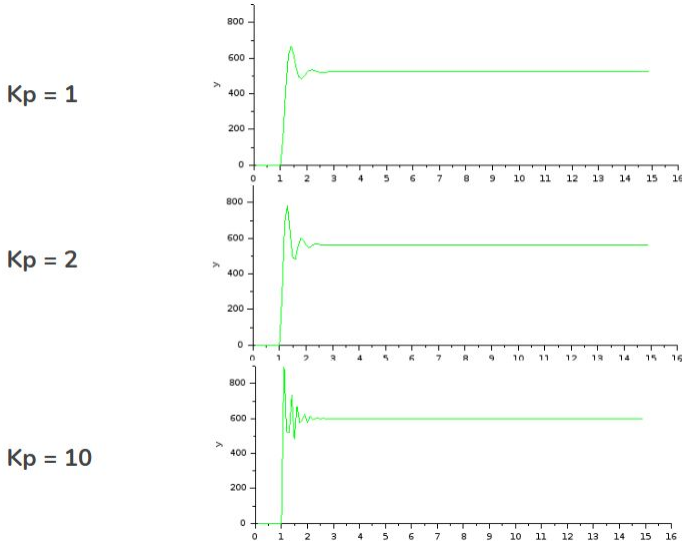

Figura 6 - Efeito do Kp no sinal.

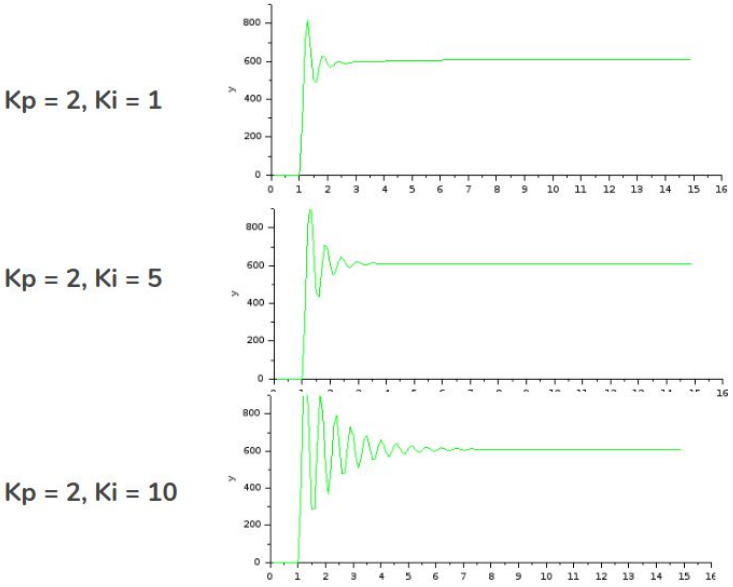

Figura 7 - Efeito do Ki no sinal. Kp foi mantido constante.

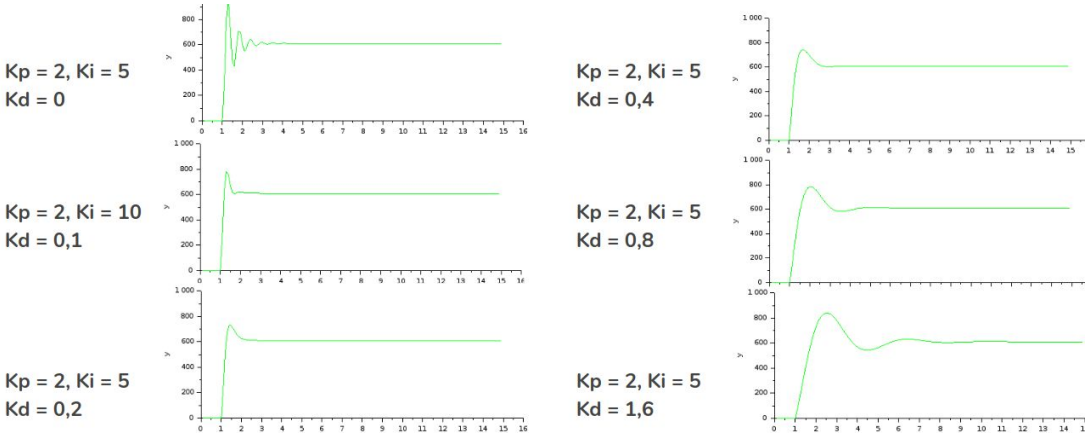

Figura 8 - Efeito do Kd no sinal. Kp e Ki foram mantidos constantes.

4. Projeto mecânico

4.1. Componentes e montagem

O projeto,mecanicamente falando,consiste em um eixo vertical com uma barra acoplada e uma base fixada na barra através de uma porca.

A barra é rotacionada pelo giro do motor que encontra-se acima dela.A cada uma volta completa a barra movimenta a base acoplada nela 4mm verticalmente.

4.2. Melhorias

4.2.1. Mancal Linear

O mancal linear é o responsável pela movimentação da base do elevador no eixo vertical. É um componente que, quando de qualidade, possibilita uma movimentação precisa devido ao baixo coeficiente de atrito.

Como melhoria do projeto,uma possível sugestão é trocar o mancal atual por um mancal linear com trilho.

4.2.2. Fuso de Esferas

É um dos componente do projeto mais essenciais para o seu funcionamento. Isso se dá pelo fato de que ele é o responsável por transformar o movimento rotacional do motor no movimento vertical do elevador.

Um fuso de esferas de qualidade provém um baixo torque de partida,prevenindo eventuais travamentos no movimento do elevador e necessitando de pouca manutenção.

4.2.3. Acoplamento de Eixo

O acoplamento de eixo tem como função unir e melhorar a sincronia entre o motor e o mancal linear.

Além de sincronizar com uma maior precisão o movimento do motor, esse componente também serve para prevenir que o motor danifique a si próprio(quando usa uma força muito grande) e para absorver eventuais choques.

4.2.4. Base e plataforma robustas

É possível notar no projeto atualmente que apenas uma pequena área do elevador está afixada na placa de madeira que atua como base

de tudo.Isso pode causar instabilidades no mecanismo inteiro ao serem feitos teste com pesos maiores, devido à instabilidade. Para melhorar isso, é necessário fixar uma maior área do elevador nessa base de madeira.

Além disso, é possível notar que a plataforma do elevador não possui forma nenhuma de segurar um objeto em cima dela, causando instabilidades devido ao livre movimento do objeto durante a subida ou descida do elevador. Para resolver isso, basta a utilização de uma grade(cerca) para delimitar uma área "segura" na plataforma, onde os objetos estariam presos.

5. Projeto eletrônico

Essa seção explicará o projeto eletrônico visando cumprir as funcionalidades que um robô cartesiano exige. Basicamente, se falará sobre a arquitetura eletrônica utilizada e os componentes que compõem esta, buscando um melhor entendimento do projeto.

5.1. Arquitetura do projeto eletrônico

A seguir, na Figura 9, há um esquemático contendo a arquitetura eletrônica utilizada no projeto. Basicamente, o projeto está centrado no bloco "Controladora/Arduino". Este bloco recebe as informações dos sensores, realiza os cálculos do algoritmo PID, que por sua vez, irá atuar na tensão de alimentação do motor com o objetivo de controlar a posição do robô cartesiano.

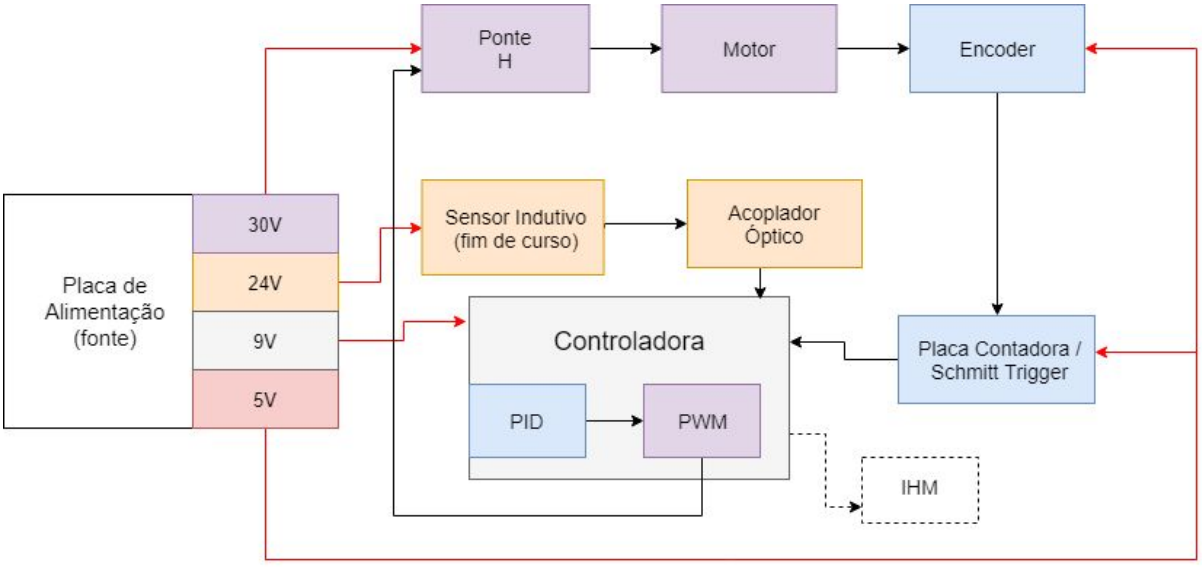

Figura 9 - Arquitetura eletrônica do projeto.

No decorrer deste documento serão explicados com mais detalhes cada um dos elementos descritos no esquemático.

5.2. Circuito de contagem do Encoder

O circuito de contagem do Encoder tem como objetivo receber pulsos do encoder, contá-los e passar a informação para a arquitetura de software.

Para realizar a contagem foram utilizados dois CIs 74193. O CI é um contador binário de 4 bits, TTL, com frequência máxima de entrada de 35MHz. Estes

contadores servirão para contar a quantidade de pulsos gerados pela fase A do encoder. A Figura 10 ilustra os pinos do CI usado.

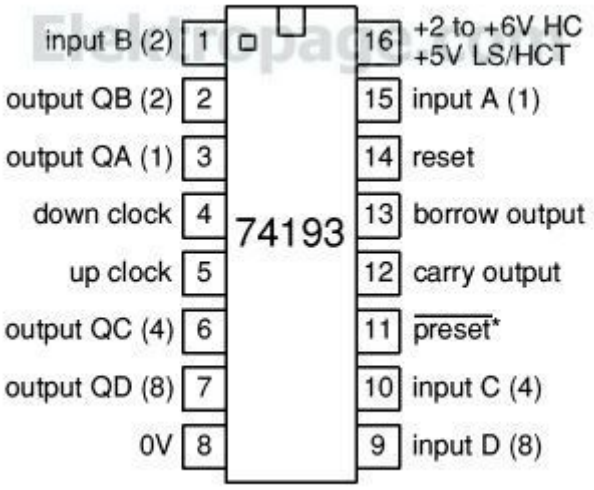

Figura 10 - Os pinos do CI 74193.

Inicialmente, montamos o circuito do contador. Era necessário um contador de oito bits e, por isso, juntamos dois contadores de quatro bits. Os contadores foram montados em uma placa furada, e as conexões foram feitas usando a técnica wire wrapping, seguindo o datasheet. A Figura 11 mostra o resultado final da montagem do circuito.

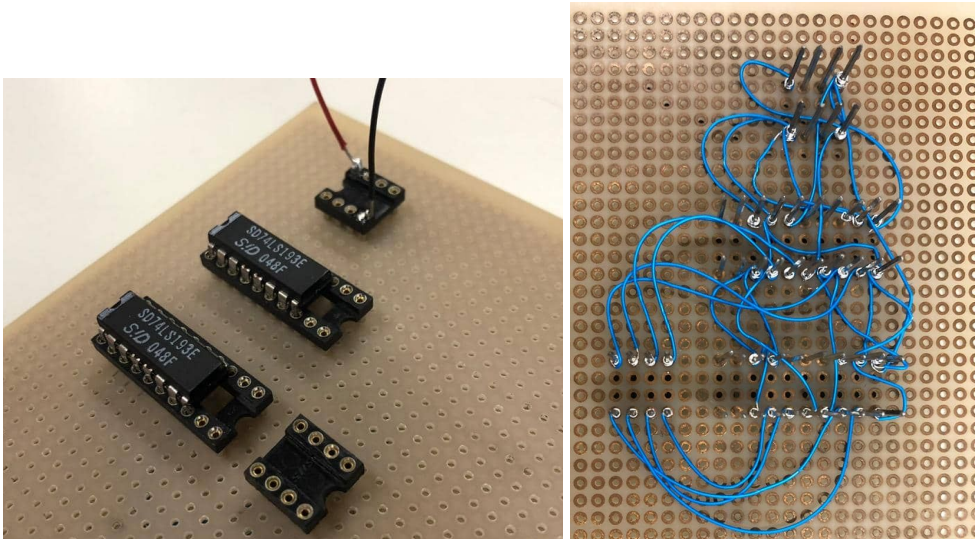

Figura 11 - O circuito do contador.

A saída do circuito de contagem é ligada ao Arduino, e o valor de contagem é utilizado para saber a posição do eixo. Com as informações de contagem é possível calcular o algoritmo de PID.

5.3. Driver de potência do Motor (Ponte H)

O bloco “Ponte H” é composto por uma shield que contém o CI L298n e componentes auxiliares. O CI presente na shield é uma ponte H.

A ponte H permite o controle da tensão de alimentação do motor e consequentemente da sua velocidade, além de permitir o controle do sentido de rotação do motor. O driver utilizado é ilustrado na Figura 12, a seguir.

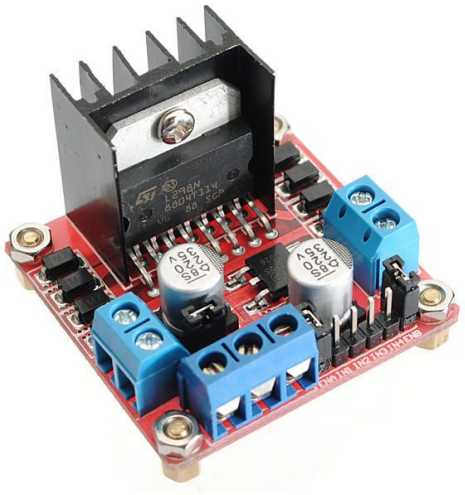

Figura 12 - A ponte H.

Essa shield tem como tensão de operação de 4-35V servindo para alimentar o motor e tensão lógica de 5V para a entrada de sinal de controle PWM.

No sistema o driver serve de interface entre o Arduino e a fonte de alimentação de 30V. Basicamente, o Arduino gera um sinal PWM, que é usado pela ponte H, para em conjunto com a fonte de alimentação gerar a tensão de alimentação correta do motor.

5.4. Sensores

Na figura não existe um bloco “Sensores”, mas a há presença de dois blocos que compreende os sensores utilizados. No caso, são os blocos: “Sensor indutivo” e “Encoder”. Esses blocos são responsáveis por produzir as informações necessárias para o cálculo de posição, PID e para determinar os finais de cursos do sistema.

5.4.1. Encoder

O bloco “Encoder” é composto por um encoder GP1A35RV, exibido na Figura 13. O encoder é constituído por um fotointerruptor acompanhado por um disco

ranhurado, acoplado ao eixo do motor. As ranhuras presente no disco interferem no fotointerruptor, modificando o sinal desde, gerando uma onda quadrada.

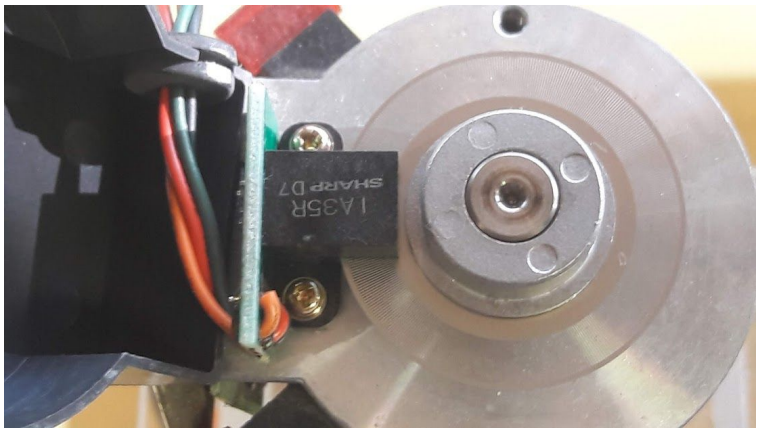

Figura 13 - O encoder.

O GP1A35RV é alimentado com 5V, tem um disco com 400 ranhuras, frequência máxima de operação de 12kHz e duas saídas TTL - fase A e fase B, onde a fase B está 90° grau defasada em relação a fase A.

5.4.2. Sensor Indutivo

O bloco “Sensor indutivo” é constituído por 3 sensores indutivos que servem para localizar o sistema a respeito dos seus limites (extremidades) e determinar onde é o “zero”. Abaixo, na Figura 14, há uma representação do tipo de sensor usado no projeto.

Um sensor indutivo basicamente detecta a presença de materiais magnéticos, modificando seu sinal caso detecte a presença destes.

Os sensores utilizados são da fabricante Balluff, de uso industrial, com tensão de 24V. O sinal lógico gerado demanda um tratamento para poder ser utilizado no Arduino. Esse tratamento será tratado no bloco “Acoplador Óptico”.

Figura 14 - O sensor indutivo.

5.5. Motor

O bloco “Motor” é composto somente pelo motor. No caso, o projeto possui um motor DC 30V 1,1A com o encoder já instalado em sua carcaça, o que facilita muito a construção do projeto, pois não há a necessidade de instalação do encoder.

A saída de alimentação do motor está ligada a saída da ponte H, permitindo que haja a corrente e tensão necessárias para o controle do motor. A Figura 15 contém o motor utilizado. Ele é visto como o elemento metálico, acoplado a estrutura.

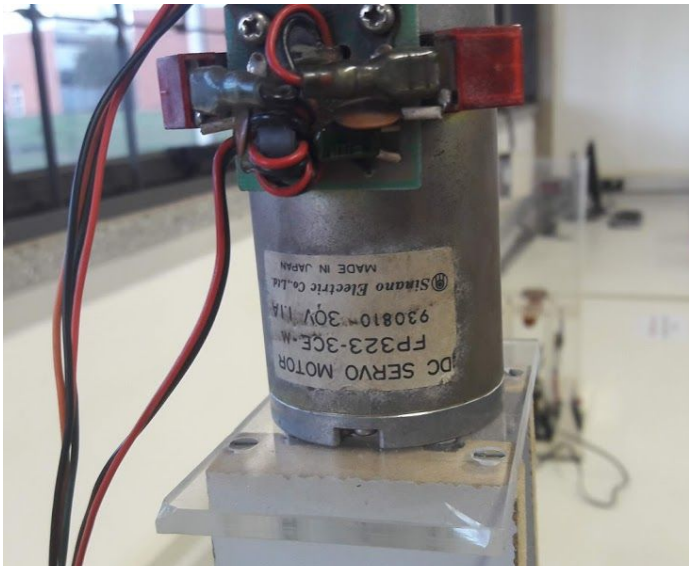

Figura 15 - O motor.

5.6. Acoplador Ótico

O bloco “Acoplador Óptico” é constituído por 3 circuitos contendo o CI 4N25, que são acopladores ópticos.

Um acoplador ótico basicamente é composto por um circuito encapsulado que contém um fotoemissor e um fotorreceptor - led e fototransistor respectivamente, isolados eletricamente entre eles.

Um acoplador óptico segue o mesmo princípio de funcionamento já explicado no encoder, onde a transmissão do sinal é feita a partir da luz. Dependendo como o circuito é montado, caso o fotoemissor esteja ativo o fotorreceptor também ficará ativo.

Uma aplicação interessante do acoplador óptico é como interface entre elementos com tensões diferentes ou para a proteção galvânica.

No projeto os acopladores ópticos são usados como interfaces entre os sensores indutivos, que trabalham com tensão de 24V e o Arduino que trabalha com 5V.

5.7. Arduino

O bloco “Placa controladora” é composto por um Arduino Mega, como o da Figura 16, e futuramente por uma Raspberry Pi. O Arduino contém as interfaces de entrada e saída dos sinais usados para o controle do sistema.

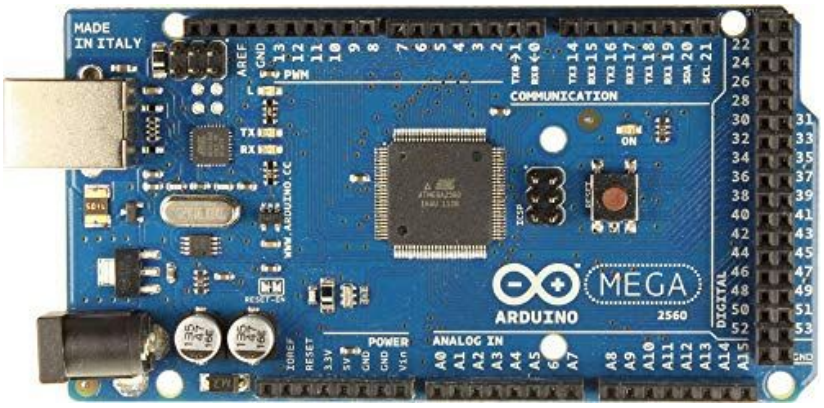

Figura 16 - Arduino Mega 2560.

No estado atual o Arduino Mega tem em suas entradas digitais os pinos de saída da placa contadora - os pinos de saída dos contadores e os pinos de reset/clear destes para acertar o processo de contagem - e os sinais de saída dos sensores indutivos.

O Arduino permite rodar o algoritmo de PID e contém saídas digitais que podem ser usadas para gerar um sinal PWM (necessário, pois esse sinal que será usado pela ponte H para alimentar corretamente o motor).

O uso do Arduino é interessante nesse caso devido ao seu baixo custo e facilidade para realizar a prova de conceito, porém numa aplicação prática seu uso não é recomendado devido às suas limitações de memória, arquitetura limitada e falta de garantia para uso em ambiente industrial.

5.8. Fonte

O bloco “Placa de alimentação” atualmente se resume a uma fonte de bancada que contém 3 saídas, onde duas delas são reguláveis com tensão entre 0-30V e uma saída 5V.

Uma das saídas reguláveis é utilizada para alimentar a shield ponte H, que por sua vez alimenta o motor. A saída 5V pode ser utilizada para alimentar o encoder (o Arduino pode realizar essa tarefa).

Uma evolução interessante para o sistema é a confecção de uma “placa de alimentação” que contenha as saídas de alimentação para cada parte do sistema com as tensões específicas.

6. Definição de parâmetros de desempenho

6.1. Descrição dos parâmetros de desempenho

- **Valor final (Vf):** Valor para o qual a resposta tende, assíntota da função de controle.
- **Tempo de subida (Tr):** tempo necessário para o sinal atingir um valor próximo do valor final, geralmente esse valor próximo estabelecido é definido num intervalo entre 90% e 100% do valor final;
- **Tempo de acomodação (Ts)**: tempo necessário para garantir que o sinal está definitivamente dentro de uma margem ao redor do Valor final. Geralmente essa margem é de ±5% ao redor do valor final;
- **Tempo de pico (Tp):** tempo que a resposta leva até atingir seu valor máximo. Esse parâmetro só é utilizado caso haja sobressinal.
- **Overshoot / Sobressinal (S)**: diferença entre o valor máximo de pico obtido na subida do sinal e o valor final;
- **Período das oscilações amortecidas (Ta):** período das oscilações amortecidas que a resposta apresenta em torno do valor final. Esse parâmetro só é utilizado caso existirem oscilações periódicas independentes do amortecimento da amplitude.
- **Erro em regime**: diferença entre o sinal e o valor final de referência quando o sinal entrar em regime permanente, definido como o módulo da diferença entre o sinal de referência e o valor mais atual função de controle, após o tempo de acomodação.

Na Figura 17, abaixo, estão ilustrados todos os parâmetros descritos acima:

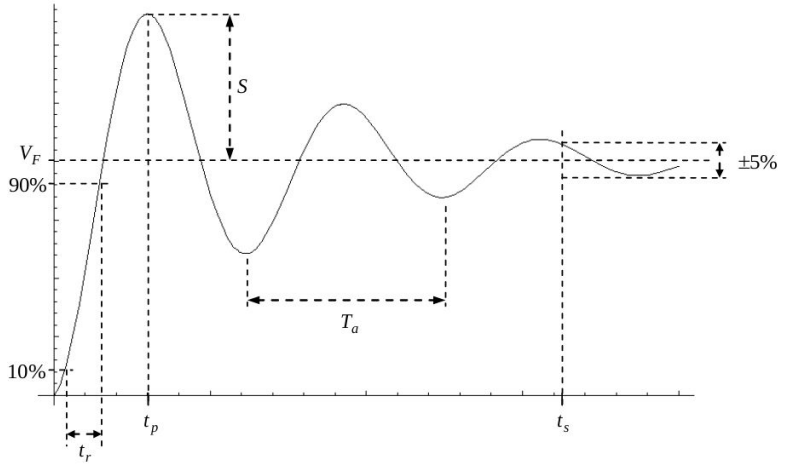

Figura 17 - Os parâmetros de desempenho de um sinal.

Portanto os parâmetros de desempenho escolhidos para o sistema de controle atual são os que sempre irão ocorrer, que são: tempo de resposta, overshooting e erro em regime

6.2. Sensibilidade dos parâmetros ao PID

A seguir, temos a tabela da sensibilidade dos parâmetros em relação às variáveis do PID:

Variável	Tempo de Resposta	Sobressinal	Tempo de acomodação	Erro em regime
Kp	Diminui exponencialmente	Aumenta	Pouca mudança	Diminui exponencialmente
Ki	Diminui	Aumenta	Aumenta exponencialmente	Diminui
Kd	Pouca mudança	Diminui exponencialmente	Diminui	Não afeta

Tabela 1 - Alterações dos parâmetros do sinal mediante variação de Kp, Ki e Kd.

Para os parâmetros que serão considerados no projeto, vemos que o Kp influencia muito para reduzir dois deles, o tempo de resposta e o erro em regime. Como descrito na seção 3.2, o comportamento do motor real não gera sobressinal, então um bom dimensionamento do Kp será suficiente para essa aplicação.

7. Definição e execução do conjunto de testes

7.1. Mecânica

Os testes da parte mecânica foram realizados com todos os componentes estruturais e com aqueles responsáveis ao movimento, sendo estes a plataforma estrutural e o motor.

O motor foi testado com objetos de variadas massas, com o intuito de se obter uma faixa operacional segura. A parte estrutural também foi testada com diferentes massas, para garantir a integridade física do projeto em diversas situações.

O sensor de indutância também entrou nessa rodada de testes, sendo medidas as distâncias mínima e máxima que o sensor apresenta alguma resposta à metais ferromagnéticos.

7.2. Eletrônica

7.2.1. Portas do Arduino

Os testes nas portas do Arduino foram feitos a partir de um programa simples, cujo objetivo era ler e escrever valores nos pinos. As portas que geram sinais PWM também foram testadas para verificar que a funcionalidade estava operante.

Os resultados obtidos foram que todas as portas do Arduino estavam em conformidade com as especificações do dispositivo e também do projeto.

7.2.2. Ponte H

A ponte H foi testada a fim de verificar seu funcionamento, e inicialmente foi verificado que a mesma estava danificada. Testes posteriores confirmaram que ela estava inoperante por estar queimada. Uma solicitação de troca foi feita.

Após receber a nova ponte H, os mesmos testes foram feitos, e verificou-se que ela estava funcionando. Com isso, os testes de PWM puderam ser realizados. Para isso, um sinal PWM foi enviado usando o gerador de sinais, e a saída foi lida no osciloscópio. O resultado foi satisfatório. Posteriormente, um sinal PWM foi enviado usando o Arduino, e verificado no osciloscópio. Novamente, o resultado foi satisfatório.

7.2.3. Acopladores ópticos

Os acopladores ópticos foram testados separadamente do sensor indutivo, utilizando multímetro e gerador de tensões. Foi verificado que o seu funcionamento está correto, passando o sinal lógico enquanto isola os os dois circuitos ao qual está conectado.

7.3. Software

Os testes de software foram feitos a partir de programas simples para verificar a consistência do processamento do Arduino. Foi feito um programa para verificar a capacidade de escrita e modificação do sinal PWM, e também um para testar a leitura do contador. Ambos os testes retornaram resultados satisfatórios.

8. Projeto de software

8.1. Arquitetura de Software

A arquitetura de software definida para o sistema de controle foi pensada de modo que haja um baixo acoplamento e alta coesão entre os componentes. Desse modo, foram definidos os módulos necessários para o funcionamento do sistema, que estão representados na Figura 18 abaixo em altíssimo nível.

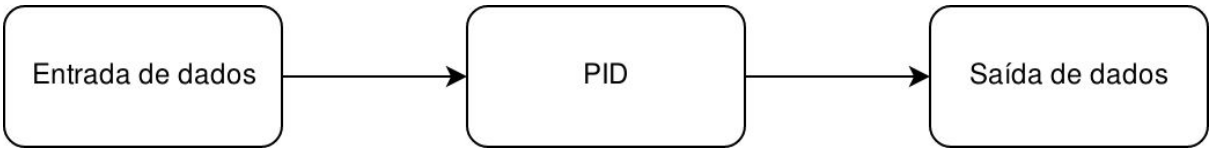

Figura 18 - Módulos do sistema.

No diagrama acima vemos o funcionamento básico do sistema em três módulos, o primeiro recebe os dados do contador referente à posição da plataforma, o segundo realiza os cálculos do PID, e o terceiro envia os resultados do cálculo para o driver do motor.

Esse funcionamento está melhor detalhado explicitando os blocos de tratamento de dados e especificando a sua interface com os componentes de entrada e saída do sistema. No diagrama abaixo, Figura 19, temos esse nível um pouco maior de detalhamento.

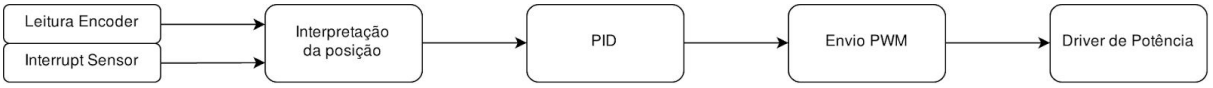

Figura 19 - Módulos do sistema com maiores detalhes.

Com esses módulos e interfaces bem definidos, foi possível criar o diagrama de classes do sistema, constando todas as funções e variáveis necessárias para o funcionamento completo do sistema. As funções do fluxo principal do sistema são privadas, pois são chamadas periodicamente à partir de um loop, enquanto que todas as funções de interrupção são públicas pois elas são chamadas à partir de eventos externos. Todas as variáveis são privadas, pois devem ser manipuladas somente dentro de certos módulos, e acessadas por outros. O diagrama de classes contendo essa informação é mostrado na Figura 20 a seguir.

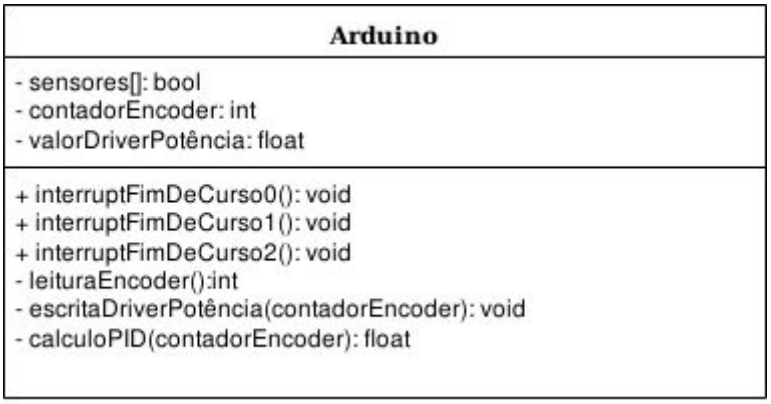

Figura 20 - Diagrama de classes do sistema.

O funcionamento específico de cada uma dessas funções será detalhado em seu subtópico específico, dentro do tópico “8. Projeto de Software”.

8.1.1. Sequência de execução

Para especificar em um nível mais baixo os detalhes de implementação e documentar a sequência de execução das funções do sistema e o fluxo de dados envolvidos foi criado um diagrama de sequência. No diagrama abaixo, Figura 21, podemos ver as funções principais sendo chamadas periodicamente à partir do loop principal, e as funções de tratamento de interrupção sendo chamadas de forma assíncrona à partir do *handler* de interrupções do arduino.

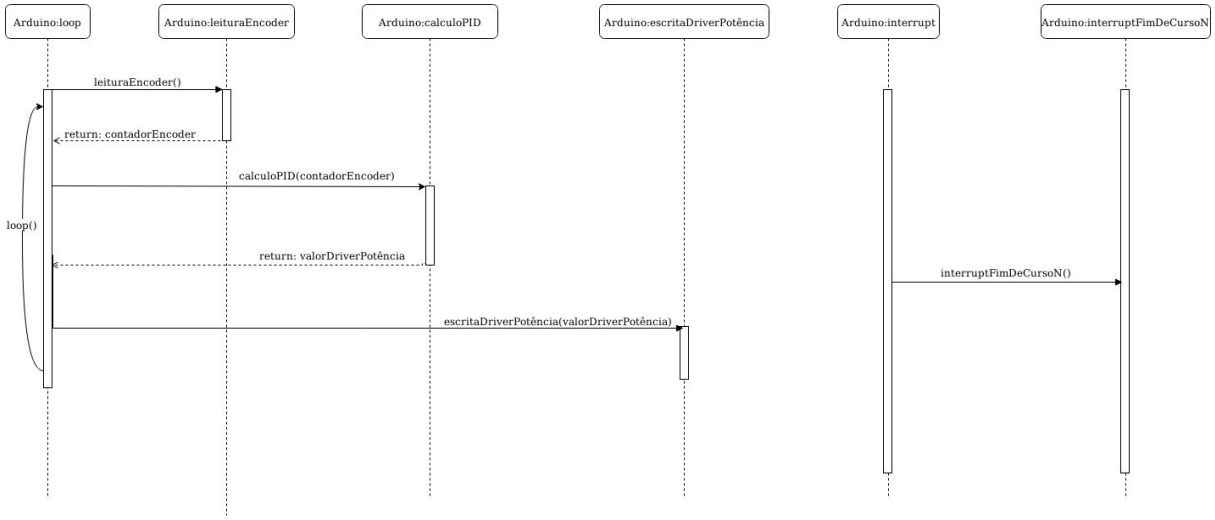

Figura 21 - Sequência de execução e fluxo de dados.

É importante observar no diagrama acima que somente as funções de leitura do contador e de cálculo do PID possuem valores de retorno com dados para seguir a execução do programa. Na versão atual do sistema não existe nenhum feedback sobre o envio dos valores ao driver de potência do motor além da leitura periódica, portanto é importante que o tempo entre cada iteração desse loop não seja nem

muito curto e nem muito longo, de modo que seriam causados problemas e imprecisões no resultado do controle.

Outro ponto importante sobre o fluxo de execução é notar que as interrupções dos sensores de posição ocorrem de maneira assíncrona e possuem prioridade sobre os outros módulos do sistema. No caso de uma interrupção de fim de curso ocorrer, em qualquer momento da execução, a iteração atual é invalidada e a ação atual do sistema será conforme a rotina de tratamento de interrupção.

8.1.2. Alternativas e versões futuras

Durante o projeto do sistema foram consideradas diversas opções válidas, porém todas iriam implicar em mudanças em componentes de hardware que já estão prontos ou necessitar de componentes adicionais e mais horas de trabalho, o que não se encaixou no cronograma. Nesse subtópico temos alternativas que foram descartadas da versão final devido aos motivos supracitados.

Inicialmente a proposta era um sistema de software com arquitetura de camadas, sendo executado em um Arduino e em uma RaspberryPi concorrentemente. O funcionamento do sistema em alto nível seria exatamente o mesmo, porém seria um sistema mais robusto, escalável e com uma garantia de execução em intervalos de tempo mais curtos.

Com o intuito de detalhar melhor o funcionamento do sistema e as responsabilidade de cada um dos computadores envolvidos foi criado um diagrama de blocos, separando as camadas do sistema em três: a interface com o hardware, o Arduino e a Raspberry Pi. Na Figura 22 abaixo se encontra esse diagrama em camadas.

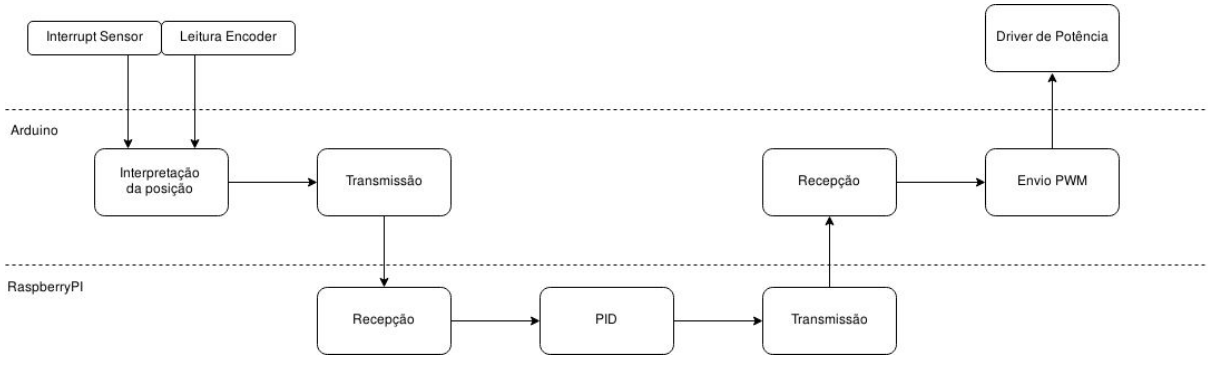

Figura 22 - Camadas do sistema.

Nessa representação em alto nível, os únicos blocos adicionais são referentes aos módulos de interface entre o Arduino e a Raspberry Pi, para transmissão e

recepção de dados. O ponto do fluxo de execução definido para ser a interface entre os dois computadores foi escolhido de modo que o Arduino faça somente a interface com os componentes de hardware, devido ao seu número de pinos de IO e protocolos de transmissão de dados aos quais tem suporte, enquanto a raspberry pi executa somente a parte crítica do sistema, o cálculo do PID, devido à sua maior capacidade de processamento.

8.2. Software de Controle do Contador

O software que controla o contador foi projetado de forma a coletar os dados que virão do contador em intervalos de tempos predefinidos. Isso garante que a verificação de posição, atributo derivado das contagens, seja consistente e reduza o custo computacional de monitorar uma entrada de dados constantemente. Há também a vantagem de deixar o tratamento de ruídos para ser feito via hardware e não via software, deixando mais recursos disponíveis para o programa geral.

O software capta o valor do contador, converte ele, verifica a diferença entre a contagem atual e a anterior para estimar quanto o robô andou, e depois calcula a velocidade, como visto no diagrama da Figura 23 a seguir.

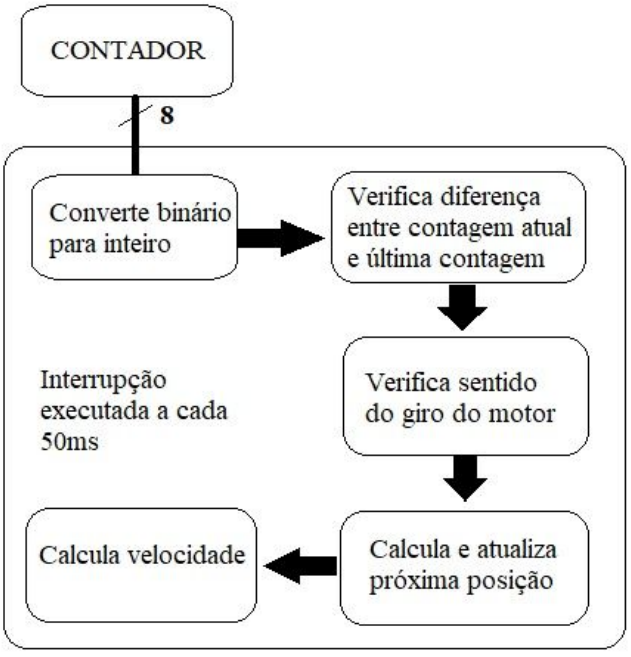

Figura 23 - Diagrama esquemático do software de controle do contador.

8.2.1. A Biblioteca de tempo

Para fazer o controle do contador via interrupção por tempo foi usada a biblioteca ‘TimerOne.h’. Essa biblioteca serve para realizar chamadas de interrupção em intervalo de tempos definidos inicialmente.

A biblioteca possui onze funções principais: initialize, start, stop, restart, read, setPeriod, pwm, setPwmDuty, disablePwm, attachInterrupt e detachInterrupt. Para esse controlador só usamos duas: initialize e attachInterrupt. A initialize configura o temporizador para funcionar a partir de um período, em microssegundos, passado como parâmetro. A attachInterrupt configura uma interrupção para ser chamada sempre que o temporizador acuar overflow. A interrupção é definida por uma função passada como parâmetro para a attachInterrupt.

Para o controlador projetado foi definido o valor do intervalo de tempo de 50000μs (50ms), configurando assim o temporizador com a função initialize. Depois foi criada uma função chamada leEncoder para ser a interrupção.

8.2.2. Avaliação de tempo de execução

O tempo definido para chamar a função de interrupção deve ser avaliado com cuidado. Uma função que demore para executar tanto tempo quanto o definido para chamá-la causará um problema de starvation no sistema, isto é, somente ela irá ser executada, não dando tempo de execução para as outras funções. O tempo com que o contador irá dar uma contagem de 8-bits completa deve ser considerado também, para que a leitura não fique defasada em relação ao contador, lendo valores incorretos por estar atrasada na condição de corrida.

Para isso, é preciso saber quanto tempo a função de interrupção leva para executar, quanto tempo o software geral leva para executar, e quanto tempo leva para o contador dar uma contagem completa. O fator mais limitante aqui é o contador, uma vez que ele é controlado pelo encoder, que é controlado pelo motor, que tem limitações físicas mais restritivas que as elétricas do circuito de contagem e também do software e hardware do Arduino.

Com o motor rodando em uma velocidade angular de 750 RPM, o encoder estará gerando uma onda quadrada de aproximadamente 5kHz, que é pouco menos que a metade de sua capacidade, e uma faixa segura de trabalho. Essa frequência faz com que a contagem total do contador seja de 51,2ms, mais do que suficiente

para a função que está sendo chamada a 50ms ser mais rápida. Velocidades abaixo disso só deixarão a margem cada vez mais segura para os 50ms.

8.2.3. Tratamento da saída do contador

O contador, então, irá gerar um valor de contagem novo a cada pulso do encoder. Esse valor de 8 bits irá vir dos circuitos de contagem e entrar pelas portas do Arduino de forma paralela. Para trabalhar com esses valores no software, foi preciso converter esse valor paralelo binário para uma forma inteira. Para isso, uma função chamada binToInt foi desenvolvida. Ela lê, um a um, os valores que entram nos pinos 46 a 53 inclusive, realiza um deslocamento para a esquerda a cada valor, e vai acumulando o valor em uma soma. O valor final, após todos os deslocamentos e somas, será o decimal a ser usado como contagem.

8.2.4. Distâncias

Para conseguir transformar uma contagem em distância, foi preciso saber as especificações físicas do encoder e da montagem do motor com o eixo que sustenta o suporte.

A primeira informação obtida foi o número de marcações do encoder, a fim de conseguir precisar qual a taxa de contagem que teríamos com o giro do motor. O valor de 400 marcações foi encontrado olhando no datasheet do encoder, e foi verificado um valor muito próximo disso usando uma câmera digital. A divergência pode ter sido ocasionada por erro humano de contagem. Depois, foi realizada uma medição de deslocamento vertical do suporte a partir de uma revolução completa do motor. O resultado encontrado foi de 4mm a cada volta. Com isso pôde-se concluir que cada contagem equivale a 0,01mm.

8.2.5. Cálculo do deslocamento

Como ponto de partida o ponto inicial sendo a contagem 0, é possível ver que a nova posição será obtida a partir da posição atual somada, ou decrementada, de um valor de deslocamento. Esse valor é a diferença entre o valor da contagem atual e a última contagem, que serão convertidas em posição.

A diferença entre as contagens deve levar em conta a possibilidade do contador reiniciar a contagem entre uma leitura e outra, gerando dados incorretos. Para isso, uma verificação foi incluída. Caso a contagem atual seja maior que a

última contagem, então a diferença de contagens será simplesmente a contagem atual menos a última contagem. Caso a contagem atual seja menor que a última contagem, então significa que o contador reiniciou a contagem, e uma outra diferença deve ser levada em conta. Para esse caso, leva-se em conta a diferença entre o limite superior de 256 e o valor da última contagem, e soma-se ao valor da contagem atual (que é a diferença entre a contagem atual e o limite inferior, que é 0). Lembrando que por causa da análise de tempo e definição de intervalo de chamada de interrupção, é impossível uma contagem atual ser maior que a antiga ocorrendo uma volta completa à frente da leitura, isto é, o contador dar mais de uma volta antes da chamada de leitura da contagem.

Após isso, o controlador verifica qual é o sentido do motor, que é feito através da leitura de dois pinos gerados pelo circuito da ponte H. O intuito desta parte do software é transformar a diferença entre contagens em um valor que seja positivo caso o motor esteja em sentido horário, ou negativo caso esteja em sentido anti-horário.

Com todos os dados obtidos, o controlador faz o cálculo da nova posição somando, ou subtraindo, a posição antiga com o equivalente em distância da diferença de contagens, que é obtido multiplicando essa diferença por 0,01mm.

8.2.6. Cálculo da velocidade

Uma vez que o deslocamento já foi obtido, a velocidade pode ser encontrada com uma simples divisão, pois a definição de velocidade é deslocamento dividido por tempo. O deslocamento foi obtido no passo anterior, e o tempo foi definido no início do programa, uma vez que a função é executada a cada intervalo de tempo.

A única modificação que se deve ser feita é a introdução de um fator de correção para a unidade. A unidade do deslocamento está em milímetros (mm), o que é viável para um projeto que não possui mais de 1m de eixo, ou seja, 1000mm. Porém, a unidade do tempo é µs, o que faz gerar uma unidade de velocidade complicada de se trabalhar. Como a unidade mm/s fica mais fácil de se ver o efeito prático, um multiplicador de 1000000 foi adicionado para esse ajuste.

8.3. Software Controlador PID

Para a implementação do controlador PID no Arduino será utilizada uma biblioteca escrita em C responsável por utilizar os valores lidos do contador do encoder para gerar a saída necessária ao motor.

As constantes derivadas das simulações da função de transferência e PID no Simulink e terminadas na simulação previamente realizada serão utilizadas como valores iniciais para o sistema. Tais constantes serão *tunadas* e melhoradas de forma iterativa, observando o comportamento do sistema e incrementando ou decrementando as constantes.

8.4. Software de Controle do Motor

Devido às características elétricas do motor DC utilizado, será necessário a utilização de um driver de potência para poder prover a energia necessária para acionar o motor.

O driver de potência têm como característica “transformar” os 5V do Arduino em 30V no motor DC. Dessa forma, será necessário no Arduino um módulo responsável por fazer o cálculo do duty cycle necessário para o PWM de forma a prover ao motor DC a tensão exata para adquirir a velocidade desejada.

Além disso, serão utilizados dois fios para definir o sentido de rotação dos motores, de forma que com base do sinal do valor de saída do PID (positivo ou negativo), serão gerados os sinais correspondentes a tal sentido de rotação.

8.5. Software de Tratamento de Interrupções

Além do contador do encoder serão utilizados 3 sensores de fim de curso para demarcar o ponto 0 do sistema, assim como os limites superiores e inferiores. De forma a otimizar o tempo de processamento do sistema e garantir a segurança do mesmo, serão utilizadas portas compatíveis com interrupções para a leitura de tais valores.

As interrupções utilizadas serão as de “borda de subida”, de forma que a função de tratamento seja chamada apenas uma vez após acionamento do sensor. Cada sensor terá como função de tratamento uma função própria, na qual uma variável booleana associada ao sensor será setada como *true*.

Desta forma é possível garantir que o tratamento da interrupção será realizada em pouquíssimo tempo, deixando o tratamento das variáveis em si para a função *loop* do Arduino, onde tais interrupções serão utilizadas para garantir a integridade do sistema.

9. Parte específica - RTOS

9.1. Introdução ao RTOS

Um Sistema Operacional em Tempo Real é um sistema operacional otimizado para uso em aplicações embarcadas ou em domínios críticos que necessitem de atuação em tempo real. Seu principal objetivo é garantir uma resposta oportuna e determinista aos eventos. Um evento pode ser externo, como um interruptor de limite sendo atingido, ou interno como um caractere sendo recebido.

A utilização de um sistema operacional em tempo real permite que um aplicativo de software seja gravado como um conjunto de tarefas independentes. Cada tarefa recebe uma prioridade e é responsabilidade do Sistema Operacional em Tempo Real garantir que a tarefa com a maior prioridade capaz de executar seja a tarefa em execução.

9.2. Facilidades e dificuldades na utilização de RTOS

Atualmente nenhum Raspberry Pi possui clock em tempo real, o que seria parte crucial para a implementação de um RTOS. Esse seria um componente necessário que deve ser comprado e acoplado na placa, ocupando 4 pinos da placa.

A interface de rede é implementada como um dispositivo USB, compartilhando o mesmo barramento dos outros componentes USB.

Seria necessário fazer uma compilação cruzada no kernel do linux para aplicar *patches* ao kernel da Raspberry. Existem algumas bibliotecas que nos ajudam a fazer isso.

9.3. Etapas do projeto

Adicionar componente de Real Time Clock

Aplicar patches no kernel do linux para Raspberry Pi e fazer compilação cruzada PREEMPT_RT

9.4. Modificações na arquitetura eletrônica

Necessário adicionar uma interface de 3.3 ⇔ 5V para funcionamento do PWM na ponte H de forma correta, já que a Raspberry Pi funciona com 3.3V. Também é necessário adaptar a interface de contagem para que ocupe menos pinos de entrada/saída, pois ela dispões de poucos, e 4 deles já estarão reservados para o Real Time Clock.

10. Conclusão

Podemos concluir que foi uma experiência proveitosa trabalhar neste projeto, pois possibilitou aprender sobre Controle de uma forma aplicada e ativa. Além disso, foi possível revisitar e utilizar diversos conceitos aprendidos em disciplinas anteriores - como Elétricos, Eletrônicos e Microcontroladores, Engenharia de Software, algo interessante, pois mostra a interdisciplinaridade da área.

Como perspectivas futuras para o projeto, temos:

- Finalizar o projeto na disciplina de SIAI;
- Substituir o Arduino Mega por uma Raspberry Pi. Isto é interessante porque a Raspberry tem um maior poder de processamento, tem um processador ARM -facilitando o porte do código para outras plataformas de hardware e por fim permite implementar algoritmos/funcionalidades mais sofisticados.
- Uso de um RTOS para gerar garantias determinísticas na execução do código.

Referências

https://www.socallinuxexpo.org/sites/default/files/presentations/Steven_Doran_SCALE_13x.pdf

https://freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf

https://freertos.org/Documentation/FreeRTOS_Reference_Manual_V10.0.0.pdf

http://www.valvers.com/open-software/raspberry-pi/step01-bare-metal-programming-in-cpt1/

https://www.get-edi.io/Real-Time-Linux-on-the-Raspberry-Pi/

https://github.com/Forty-Tw0/RaspberryPi-FreeRTOS

https://github.com/jameswalmsley/RaspberryPi-FreeRTOS

https://www.stevebate.net/chibios-rpi/GettingStarted.html .