UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DA ELÉTRICA
CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO

JUAN FONSECA MAIA DA SILVA

ESTUDO TEÓRICO E EXPERIMENTAL DA DINÂMICA E CONTROLE DE MANIPULADOR ROBÓTICO

TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO
2017

JUAN FONSECA MAIA DA SILVA

ESTUDO TEÓRICO E EXPERIMENTAL DA DINÂMICA E CONTROLE DE MANIPULADOR ROBÓTICO

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina Tcc 2 do Curso de Engenharia de Controle e Automacão da Universidade Tecnologica Federal do Paraná - UTFPR como requisito parcial para obtencão do título de Bacharel em Engenharia de Controle e Automacão.

Orientador: Prof. Dr. Fabian Andres Lara Molina
Coorientador: Prof. Dr. Edson Hideki Koroish

CORNÉLIO PROCÓPIO
2017

Universidade Tecnológica Federal do Paraná
Campus Cornélio Procópio
Departamento Acadêmico de Elétrica
Curso de Engenharia de Controle e Automação

FOLHA DE APROVAÇÃO

Juan Fonseca Maia da Silva

Estudo teórico e experimental da dinâmica e controle de manipulador robótico

Trabalho de conclusão de curso apresentado às 13:00hs do dia 29/11/2017 como requisito parcial para a obtenção do título de Engenheiro de Controle e Automação no programa de Graduação em Engenharia de Controle e Automação da Universidade Tecnológica Federal do Paraná. O candidato foi arguido pela Banca Avaliadora composta pelos professores abaixo assinados. Após deliberação, a Banca Avaliadora considerou o trabalho aprovado.

_____

Prof(a). Dr(a). Fabian Andres Lara Molina - Presidente (Orientador)

_____

Prof(a). Dr(a). Edson Hideki Koroish - (Coorientador)

_____

Prof(a). Dr(a). Luiz Marcelo Chiesse da Silva - (Membro)

_____

Prof(a). Me(a). Marco Antonio Ferreira Finocchio - (Membro)

A folha de aprovação assinada encontra-se na coordenação do curso.

AGRADECIMENTOS

Aos meus pais, pelo amor, incentivo e apoio incondicional.

Aos professores Fabian Andres Lara Molina e Edson Hideki Koroishi que contribuíram muito para que esse trabalho fosse realizado.

A todos os professores do curso, que foram tão importantes na minha vida acadêmica.

Aos amigos e colegas, pelo incentivo e pelo apoio constantes.

RESUMO

FONSECA, Juan Maia da Silva. **Estudo Teórico e Experimental da Dinâmica e Controle de Manipulador Robótico.** 2017. 115 f. Trabalho de Conclusão de Curso (Graduação) – Engenharia de Controle e Automação. Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2017.

Este trabalho tem como objetivo apresentar uma abordagem teórica e experimental de um robô ARM com 3 graus de liberdade. Para isto, as ferramentas desenvolvidas para simulação, desenvolvimento de um protótipo e controle de manipuladores são apresentadas. Baseado nestas ferramentas será feita uma discussão sobre o quão pertinente é utilizar aplicativos computacionais difundidos no âmbito acadêmico e no industrial no análise e controle de manipuladores robóticos. O manipulador estudado neste trabalho é modelado, e sua dinâmica analisada mediante simulação computacional. Adicionalmente, é desenvolvido o projeto do controle de posição de uma junta robótica. Os experimentos comprovarão a eficiência da técnica de controle proposta.

Palavras-chaves: Manipuladores Robóticos, Robotics Toolbox, MATLAB.

ABSTRACT

FONSECA, Juan Maia da Silva. **Theoretical and experimental study of the dynamics and control of robotic manipulator.** 2017. 115f. Work Completion of course (Graduation) - Automation and Control Engineering. Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2017.

This work aims to present a theoretical and experimental approach to an ARM robot with 3 degrees of freedom. Simultaneously it will be presented tools built for simulation and control of manipulators, which will be a discussion of how relevant would be to apply the use of a computer application widespread in the academic, the industrial. a case study where the goal and make a comparison between theory and practice, and verify that the tool developed for simulation could be extended to companies will be done. The developed handler is modeled and its dynamics analyzed through simulations. The experiments will prove the effectiveness of the proposed control technique.

Key-words: Robotic Manipulators, Robotics Toolbox, MATLAB.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Grande parte das pessoas, quando pensam em robô, relaciona-os à humanoides dos filmes de ficção cientifica, sendo apenas uma visão ingênua e fantasiosa criado pela indústria de filmes, como os robôs que pensam como humanos, ou ainda aqueles que escravizam civilizações e invadem a terra. Porem na atualidade esta visão é diferente (STONE, 2004).

Na indústria, os robôs são destinados, por exemplo, para melhor a capacidade de produção pela automação de tarefas de manufatura. Os robôs como aparelho automáticos, geralmente em forma de braço, são capazes de cumprir determinadas tarefas (CRAIG, 2012). O *Robot Institute of América* (Instituto de Robótica da América) designa um robô como "um manipulador multifuncional reprogramável, projetado para mover materiais, peças, ferramentas ou dispositivos especializados através de programas para o modificar o desempenho em uma variedade de tarefas"(STONE, 2004).

Diversas tarefas executadas na indústria demandam de velocidade, precisão e repetição do movimento. Outras tarefas apresentam insalubridade para os operários. Nesse sentido os robôs são introduzidos na indústria para executar a tarefa e preservar a integridade física do trabalhador (CRAIG, 2012).

Em meados de 1945, com o término da segunda guerra mundial, os Estados Unidos impulsionados pela indústria bélica que supria armas para Guerra Fria e por um rápido avanço em sua tecnologia, experimentaram um forte crescimento econômico. Em 1961 a GM(General Motors) instalou o robô INIMATE em sua linha de produção e montagem, dando origem a robótica. Ele pesava 1.800 kg e obedecia a comandos gravados em fitas magnéticas. O robô tinha a função de carga e descarga de peças em altas temperaturas de uma máquina de fundição (SICILIANO; KHATIB, 2008).

Em 1964 foram criados e utilizados na indústria robôs hidráulicos com até 6 graus de liberdade numa fábrica na Noruega. Esse robô foi o primeiro a usar o conceito de coordenadas polares e movimento com trajeto contínua. O robô foi utilizado para pintar peças. Na década de 70 um agricultor britânico criou um robô para soldagem do tipo arco voltaico, o robô foi fruto de uma modificação de um robô pulverizador (SICILIANO; KHATIB, 2008).

Em 1971, durante a restruturação da indústria japonesa, os processos de manufatura foram aperfeiçoados com robôs. O resultado foi corte de custos e melhoria da eficiência. O primeiro robô de nome *Unimate*, foi desenvolvido pela *Kawasaki* em 1972, depois de adquirir o projeto *Unimation* e efetuar melhorias e adaptações em suas funcionalidades com o objetivo de criar um robô de arco-de-solda para trabalhar na linha de montagem de suas motocicletas (SANTOS, 2004).

Em 1973 a *Unimation*, criadora do robô *Unimate* cria WAVE a primeira linguagem de programação para esse manipulador, em 1974 a empresa cria AL, com ambas sendo substituídas pela linguagem VAL alguns anos mais tarde. Ainda em 1974 a *Unimation* cria o seu segundo

manipulador chamado *PUMA-Programmable Universal Machine for Assembly* (Máquina universal programável para montagem), sendo este disponibilizado no mercado em 1978. Ainda nos dias de hoje o manipulador PUMA continua sendo bastante utilizado na indústria (CRAIG, 2012).

No final da década de 70, ainda no Japão foi construído pela Universidade de Yamanashi um robô com grande aplicação industrial: *SCARA-Selective Compliant Assembly Robot Arm* (Braço Robótico para Montagem com Flexibilidade Seletiva). O robô SCARA é geralmente rápido e mais sensível que os sistemas de robôs cartesianos. Sua montagem é simples, porém podem ser mais caros que os robôs cartesianos, e o sistema de controle mais complexo. Atualmente esse tipo de robô é bastante empregado em trabalhos onde demandam precisão e velocidade na operação (CARRARA, 2009).

Os robôs são utilizados numa gama variada de aplicações industriais. As primeira aplicações consistiam em automatizar operações de carga e descarga, porem com o avanço da tecnologia, as aplicações diversificaram-se de simples operação de transporte a operações de montagem e processos (soldagem, pintura, deposição de vedantes, etc.).

O presente trabalho tem como finalidade apresentar um estudo teórico e experimental de um robô ARM com 3 graus de liberdade. Consequentemente, ferramentas computacionais para a simulação e controle e desenvolvimento do manipuladores robóticos são aplicadas no desenvolvimento do trabalho. Ditas ferramentas computacionais utilizam o software MATLAB para simulação, Solid Works para o projeto do protótipo e Arduino IDE para a prototipagem experimental das leis de controle nas juntas do robô. O Objetivo é fazer uma comparação entre a teoria e pratica, e verificar se a ferramenta desenvolvida para simulação poderia ser estendida às empresas, evitando custos em relação a utilização de robôs industriais.

1.1 PROBLEMA

Numa linha de produção a maioria das tarefas são repetitivas, sendo assim que a meados do século passado surgiu um problema: como replicar uma tarefa para outras máquinas, economizando tempo e recursos. O primeiro a procurar uma solução para esse problema foi Sir. Charles Devol, que desenvolveu uma forma de registrar uma sequência de trajetórias a serem seguidas por um robô (GROOVER, 1987).

Um dos grandes problemas do inicio da robótica, foi programar um robô para uma determinada tarefa em toda linha de montagem. Infelizmente falhas podem acontecer, e para evitar esse problema empresas investem quantias substanciais em softwares de simulação. Num software de simulação é possível criar um ambiente virtual para testar o funcionamento de um robô, para somente depois de corrigidos eventuais problemas ele ser aplicado de forma real.

A maioria das universidades de engenharia do mundo utilizam o MATLAB como soft-

ware para análise de controle de sistemas e simulação. Dentro do MATLAB existem as toolboxes (ferramentas) que são funções adicionais daquelas presentes no programa para uma aplicação específica. Estas funções são agregadas mediante a instalação de bibliotecas externas. Uma ferramenta em constante desenvolvimento e com varias funções destinadas á simulação de manipuladores robóticos, chamada *Robotics Toolbox for Matlab* pode ser instalada no MATLAB (CORKE, 2015). Esta toolbox com código fonte aberto, aborda varias questões relativas aos robôs, sendo útil para simulação e controle de manipuladores reais, a partir de experimentos virtuais.

Algumas indústrias, especialmente aquelas que usam o robô PUMA têm implantado a toolbox em sua linha de produção (MATHWORKS, 2011). Nessa abordagem, esse trabalho propõe fazer um estudo mediante simulações computacionais e comparar com experimentos práticos para um robô com três graus de liberdade.

Adicionalmente, o estudo experimental do controle de posição de uma junta robótica também será apresentado com o objetivo de avaliar o desempenho dinâmico da junta de um manipulador utilizando ferramentas para prototipagem de leis de controle.

1.2 MOTIVAÇÃO E JUSTIFICATIVAS

Dentre as principais atribuições de um engenheiro, destaca-se a busca por reduzir e minimizar possíveis falhas em processos produtivos, bem como a busca por novas tecnologias. Com esse proposito se for possível otimizar a utilização de um software, ou seja, e ter acesso a diferentes funcionalidades num mesmo programa, maior será a economia da empresa. Este trabalho se constitui em uma primeira tentativa para desenvolver ferramentas de desenvolvimento utilizadas nas indústrias de manufatura que aplicam manipuladores robóticos nos processos de manufatura. Portanto, é necessário desenvolver ferramentas computacionais para a análise e simulação das leis de controle considerando o modelo dinâmico representativo do manipulador e de seus atuadores. Ademais, faz-se necessário introduzir o uso de simulações com ferramentas intuitivas que podem contribuir no projeto e análise na área da robótica.

1.3 OBJETIVOS

1.3.1 Objetivos Gerais

Realizar um estudo sobre teórico e experimental da cinemática, dinâmica e controle de um manipulador robótico para o projeto de um manipulador robótico com base na simulação computacional e implementação experimental do controle de uma junta robótica.

1.3.2 Objetivos Específicos

- Formular o modelo dinâmico completo do manipulador robótico de três graus de liberdade.
- Simular a dinâmica completa juntamente com o sistema de controle de posição com o auxilio da ferramenta *Robotics Toolbox for Matlab*.
- Projetar e montar um protótipo experimental do manipulador robótico.
- Projetar e implementar o controle experimental de uma junta robótica.

1.4 ESTRUTURA DO TRABALHO

O restante deste trabalho de conclusão de curso está dividido em seis seções. O capítulo 2 discorre sobre a fundamentação teórica. O capítulo 3 apresenta os materiais e métodos, isto é, como será desenvolvido este trabalho. O capítulo 4 apresenta os resultados referentes a construção do protótipo, contemplando a parte mecânica e elétrica, os resultados da simulação com o auxilio do *Simulink* e da toolbox para robótica, e por fim o resultados do controle da junta robótica. Finalmente, no capítulo 5 é identificado os objetivos foram atingidos ou não. Também nesse capítulo é mostrado as limitações da pesquisa e as possíveis contribuições.

2 FUNDAMENTOS TEÓRICOS

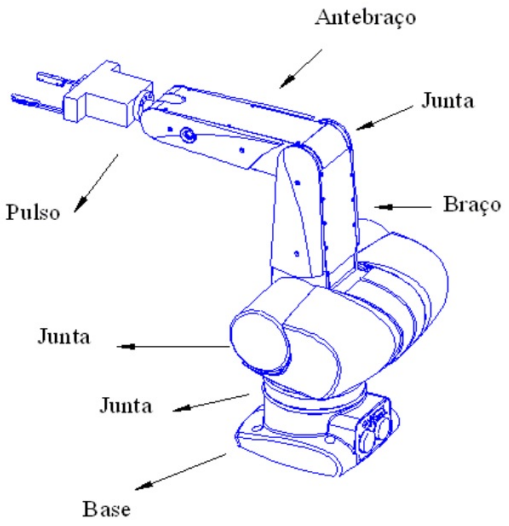
Este capítulo apresentara os conceitos fundamentais relacionados à robótica e aos manipuladores robóticos. Na primeira seção discorre sobre o princípio de funcionamento, e componentes de um robô. Após, a seção da cinemática apresenta a modelagem cinemática para o planejamento dos movimentos e trajetórias do manipulador. A seção sobre a dinâmica apresenta a modelagem do manipulador. A seção controle apresenta as técnicas de controle mais utilizadas no controle de manipuladores, e especificamente apresenta o controle PID utilizando neste trabalho. E por fim na ultima seção será apresentado a ferramenta de simulação e controle robótica utilizada no software MATLAB.

2.1 MANIPULADOR ROBÓTICO

Um manipulador é um conjunto de corpos rígidos ligados por juntas, formando cadeias cinemáticas que definem uma estrutura mecânica. Este é composto por atuadores, que agem sobre as juntas, adicionalmente, a transmissão liga os atuadores à estrutura mecânica. Muitas vezes o termo manipulador e robô são usados com a mesma finalidade, embora, formalmente, tal não seja um consenso no meio cientifico (PIERI, 2002).

A estrutura principal dos manipuladores robóticos antropomórfico é formada por: Estrutura mecânica (Antebraço, pulso, junta, braço e base), atuadores(elétricos, hidráulicos ou pneumáticos), sistema de controle e por um painel de comando (FERREIRA, 2005) como mostrado na Fig. 1.

Figura 1 – Anatomia de um robô angular.



Fonte: (CARRARA, 2009).

2.2 ESTRUTURA MECÂNICA

Esta seção apresenta conceitos sobre base, braço, juntas, órgão terminal, sensores e acionamentos.

A base oferece um suporte ao manipulador. O tipo de base mais utilizada no mundo é a fixa, porem a base pode ser móvel. Os de base fixa operam dentro de um espaço limitado e não podem movimentar-se na base(PIERI, 2002).

O braço é constituído por elos unidos por juntas de movimento rotacional. As informações obtidas pelos sensores são processadas pelo sistema de controle e logo apos o movimento será executado (CARRARA, 2009).

As juntas orientam os elos para as posições que correspondem a uma tarefa á ser realizada (CRAIG, 2012). A junta podem ser do tipo prismática, rotacional, esférica, cilíndrica ou planar. Devido a funcionalidade as juntas rotativas e prismáticas são as mais utilizadas. Por causa da combinação de movimento em linha reta e giro em torno de um eixo contemplar quase todos as trajetórias necessárias para um bom funcionamento do manipulador podem ser realizadas (CARRARA, 2009).

O órgão terminal é conectado ao pulso, também conhecida como garra, tem a função de agarrar objetos de forma segura (ROMANO, 2002).

Os sensores informam ao sistema de controle informações acerca do ambiente que foram inseridos. Na robótica, os sensores são necessários para detecção de posição, orientação, obstáculos e analisar defeitos (ROSÁRIO, 2006);

Os dispositivos de acionamentos é responsável pelo movimento dos elos e da dinâmica dos robôs. Nas linhas de produção acionamentos elétricos são os mais utilizados, visto que apresentam maior precisão e sensibilidade (CARRARA, 2009). O acionamento pode ser direto ou indireto. Ele é direto quando o acionador é adaptado direto na junta. Ele é indireto quando a transmissão de potência é feita com correntes, polias, engrenagens ou correias (SANTOS, 2004).

2.3 CINEMATICA

A cinemática estuda os movimentos dos robôs, isto é a posição no espaço tridimensional, e a velocidade angular(PIERI, 2002).

As principais questões á serem respondidas com relação á cinemática são: a cinemática direta onde a intenção é determinar a posição e orientação do efetuador final em relação a um sistema de referencia, já a forma inversa permite calcular os ângulos das juntas a partir da posição e orientação do mesmo efetuador final (PIERI, 2002).

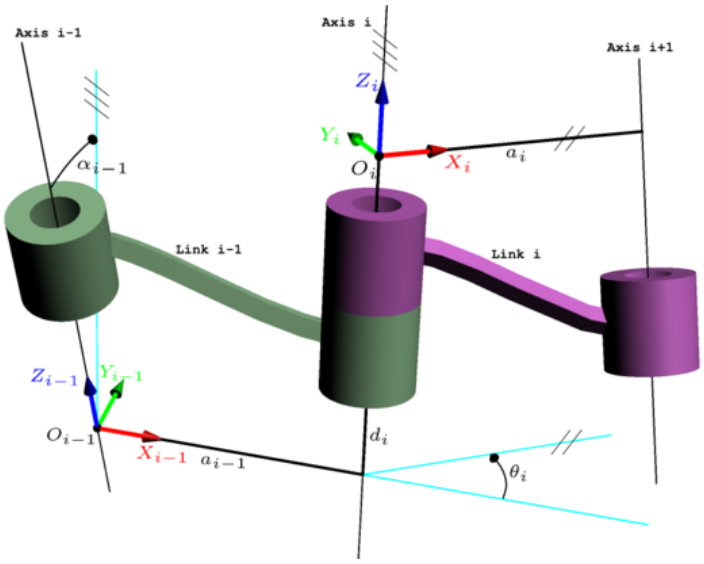
2.3.1 Cinemática Direta

Na cinemática inversa o objetivo é determinar a orientação e posição em relação a base. O resultado será uma equação para cada tipo de robô, de acordo com a sua configuração. Porem é necessário conhecer os parâmetros de cada elemento do robô (SANTOS, 2004). Para um melhor posicionamento do sistema de coordenadas é comum utilizar a notação de *Denavit Hartenberg*, este algoritmo atribui um sistema de coordenada ortonormal, sendo um sistema para cada elo. Para isto constrói-se a matriz de transformação homogênea (SPONG; VIDYASAGAR, 2008).

A representação DH pode ser vista na Figura 2. A representação DH depende de quatro parâmetros, que descrevem o da junta (SANTOS, 2004):

- θ_i: ângulo de junta obtido entre os eixos X_{i-1} e X_i no eixo;
- d_i : distância entre a origem do (i)-ésimo sistema de coordenadas até a intersecção do eixo Z_{i-1} com o X_i, ao longo de Z_{i-1};
- a_{i-1}: distância entre a intersecção dos eixos Z_{i-1} e X_i até a origem do i-ésimo sistema de referência ao longo do eixo X_i;
- α_{i-1}: ângulo entre os eixos Z_{i-1} e Z_i, medidos no eixo X_i (SCHILLING, 1996).

Figura 2 – Notação de Denavit Hartenberg.



Fonte: Adaptada de (CRAIG, 2012).

De posse dos parâmetros DH, e a partir dos sistemas de coordenadas definidos pela notação DH, obtém-se a matriz de transformação homogênea apresentada na equação (2.1):

$$ {}^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i \cos\alpha_{i-1} & \cos\theta_i \cos\alpha_{i-1} & -\sin\alpha_{i-1} & -d_i \sin\alpha_{i-1} \\ \sin\theta_i \sin\alpha_{i-1} & \cos\theta_i \sin\alpha_{i-1} & \cos\alpha_{i-1} & d_i \cos\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1} $$

A matriz pode ser resumida, conforme indicado pela equação (2.2), é formada por uma matriz de rotação R, do tipo 3x3 (responsável por descrever a orientação relativa entre os dois sistemas de coordenadas, definidos como A e B), e um vetor coluna 3x1, cuja função é indicar a origem do sistema de coordenadas B em relação ao sistema de coordenadas A. Da mesma forma, a segunda e terceira coluna representam as posições dos vetores de B em relação á coordenada A. A última linha da matriz será sempre [0 0 0 1] (SCHILLING, 1996).

$$ {}^{A}_{B}T = \begin{bmatrix} {}^{A}_{B}R_{3X3} & {}^{A}O_{B3X1} \\ 0\,0\,0 & 1 \end{bmatrix} \tag{2.2} $$

2.3.2 Cinemática Inversa

Na cinemática inversa buscamos o caminho inverso da cinemática direta, e para resolver esse problema há três métodos principais (PIERI, 2002).

- Método geométrico: Bastante usado quando o robô tem poucos graus de liberdade, bastando apenas equacionar as relações geométricas (PIERI, 2002);
- Matriz de transformação homogênea: É o mais complexo, sendo assim pouco utilizado. A partir da manipulação do modelo cinemático direto obtém-se as relações inversas (PIERI, 2002);
- Desacoplamento cinemático: Trata a posição e orientação final conhecidas, então calcula-se os valores das primeiras variáveis articuladas que atingem esse ponto. As outras variáveis são obtidas com dados de orientações de etapas anteriores(PIERI, 2002).

Dado a simplicidade e eficiência do método geométrico esse será o escolhido para o estudo teórico proposto e posteriores simulações.

2.4 DINÂMICA

O modelo dinâmico do manipulador é de extrema importância para determinar os esforços necessários pelos atuadores para realização do movimento. Para acelerar ou desacelerar um robô afim de completar um movimento é necessário definir um complexo conjunto de esforços que devem ser aplicados aos atuadores. O torque necessário para um movimento depende de parâmetros espaciais e temporais, e o cálculo dos esforços dos efetuadores para controlar o manipulador é feito a partir das equações da dinâmica. A dinâmica do manipulador e importante para o projeto, simulação e uma análise mais profunda dos movimentos dos manipuladores (LATRE, 1988).

A dinâmica lida com equações diferenciais do movimento, onde os manipuladores movimenta-se de acordo com os torques aplicados pelos atuadores (CORKE, 2015).

O modelo dinâmico das n juntas de um manipulador robótico é dado pela equação..., assim:

$$\mathbf{Q} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{F}(\dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) \tag{2.3}$$

Sendo:

$\mathbf{q}$ = vetor com as coordenadas das juntas que descrevem a posição do robô;

$\dot{\mathbf{q}}$ = vetor de velocidade das juntas;

$\ddot{\mathbf{q}}$= vetor com a aceleração das juntas;

$\mathbf{M}$ = matriz N x N da massa do manipulador;

$\mathbf{C}$ = descreve os efeitos da força centrípeta e o torque de Coriolis;

$\mathbf{F}$ = descreve os atritos: viscoso e de Coulomb;

$\mathbf{G}$ = aceleração da gravidade;

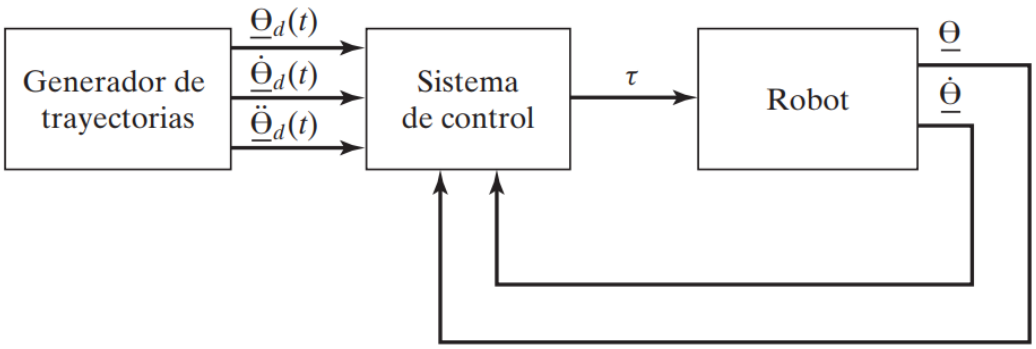
$\mathbf{Q}$ = vetor que descreve as forças associadas aos atuadores nas coordenadas dadas por $\mathbf{q}$.

2.5 CONTROLE

No controle desejamos fazer que as articulações do manipulador sigam trajetórias de posição desejadas, mas os atuadores são comandados em termos de momento de torsão, então devemos usar sistemas de controle para calcular os comandos apropriados para realizar este movimento (CRAIG, 2012).

A Figura 3 mostra a relação entre o gerador de trajetórias e o robô. O robô recebe um vetor de momentos de torsão tau do sistema de controle, enquanto os sensores do manipulador leem os vetores das posições das articulações. As linhas de sinal da Figura 3 levam vetores N x 1 (onde N é o número de articulações do manipulador).

Figura 3 – Diagrama de blocos de um sistema de controle.



Fonte: Adaptado de :(CRAIG, 2012).

Considerando que o algoritmo pudesse implementar um bloco nomeado como “sistema de controle”, uma opção seria usar a equação da dinâmica do robô para calcular os momentos de torsão requeridos para uma determinada trajetória.

Infelizmente a imperfeição do modelo dinâmico e a presença de distúrbios fazem que o uso de uma equação seja algo impraticável em aplicações reais. Geralmente, a única forma de construir um sistema de controle confiável é usando a retroalimentação dos sensores das articulações, como indicado na Figura 3.

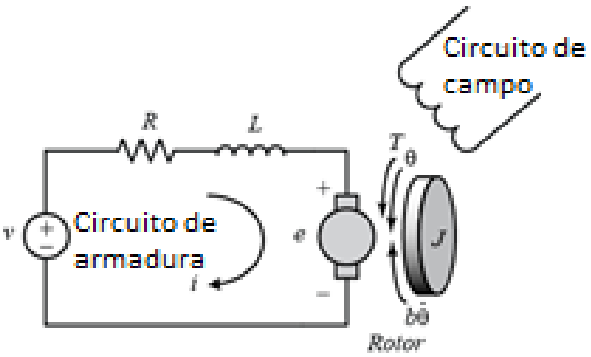
Na retroalimentação um dado importante é o erro de posição, que é a diferença entre a posição desejada e a posição real. Esse sistema de controle é denominado sistema de laço fechado (CRAIG, 2012).

No projeto de um sistema de laço fechado um critério importante é fazer com que o mesmo permaneça estável. Em geral, define-se um sistema como estável se os erros são pequenos ao executar trajetórias desejadas, incluindo a presença de perturbações, já um sistema é tido como instável se os erros são grandes e aumentar ao invés de diminuir.

O sistema da Figura 3 é um problema de controle de MIMO(múltiplas entradas, múltiplas saídas). O método mais adotado atualmente pelos fabricantes de robôs industriais é o controle independente de articulações, onde cada articulação é controlada de forma separada, ou seja tratar cada junta como um sistema SISO(uma entrada, uma saída) (CRAIG, 2012).

Um atuador comum que se encontra em muitos robôs industriais é o motor DC. A fim de gerar um modelo de física do motor, é importante considerar uma versão simplificada do seu funcionamento. A Figura 4 representa um circuito equivalente elétrico da armadura e o diagrama de corpo livre do rotor.

Figura 4 – Modelo motor DC.



Fonte: Autoria Própria.

Para a modelagem será considerado a fonte de tensão V aplicada à armadura do motor como entrada e a velocidade de rotação do eixo como saída, levando em consideração que para uma entrada constante de tensão o motor irá girar a uma rotação correspondente. Para fins de modelagem, o rotor e o eixo são assumidos como rígidos. Assume-se ainda um modelo de atrito viscoso, isto é, que o torque de fricção é proporcional à velocidade angular do eixo.

As seguintes variáveis representam os parâmetros físicos do motor.

- (J) momento de inércia do rotor.
- (B) constante de fricção viscosa do motor.
- (K_e) constante da força eletromotriz.
- (K_t) constante do torque do motor.
- (R) resistência da armadura.
- (L) indutância da armadura.

Em geral, o binário gerado por um motor DC é proporcional à corrente do induzido e à intensidade do campo magnético. Neste trabalho, será considerado que o campo magnético é constante e, portanto, que o torque do motor é proporcional apenas à corrente de indução i por um fator constante K_t, conforme mostrado na equação (2.4). Isto é referido como um motor controlado por corrente de armadura.

$$T = K_t * i \tag{2.4}$$

Nos sistema de controle a velocidade de cálculo é de extrema importância, implicitamente o computador de controle realiza os cálculos da lei de controle em um tempo muito

rápido, de maneira que admite-se que o sistema é continuo no tempo. Essa aproximação é aceitável quando a velocidade de cálculo é muito mais rápida que a frequência natural do sistema que está sendo controlado.

Para que essa aproximação seja válida o conteúdo da frequência de cálculo deve ser limitada como no mínimo o dobro da frequência de rotação do motor que faz o movimento da articulação controlada (CRAIG, 2012).

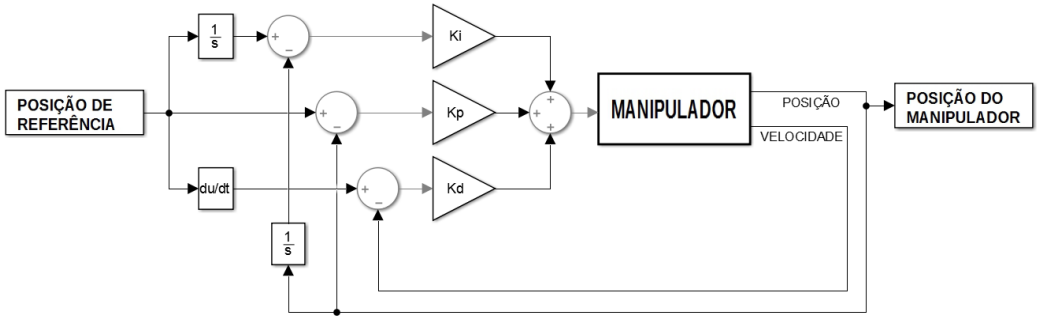
Duas das principais técnicas de controle de trajetória para os manipuladores são: Torque Computado e o PID (Proporcional-Integral-Derivativo). No torque computado determina-se o controle do torque baseado nas equações da dinâmica, por apresentar a necessidade de um sistema preciso de controle de corrente elétrica, esta técnica não é muito utilizada em casos de controles em baixa velocidade. No PID, tem-se o controle isolado para cada junta, sendo assim bastante utilizado por sua facilidade de programação e por oferecer uma precisão satisfatória para muitos casos. No entanto, quando se busca grandes velocidades de operação, a técnica PID não apresenta resultados satisfatórios, já que não leva em consideração os efeitos da dinâmica.

Ele é capaz de eliminar o erro no estado estacionário sem aumentar muito o ganho proporcional, bem como consegue reduzir o efeito eventuais perturbações ocasionadas pelo efeito de acoplamento de outra junta (ROSÁRIO, 2006); Uma desvantagem do PID é que este necessita de um ajuste refinado nos seus parâmetros, além de ser sensível á dinâmica do sistema. Para uma maior eficiência, é necessário o modelo exato da cinemática inversa do manipulador para que seja possível a conversão de trajetória com precisão (CORKE, 2015).

Nesse trabalho será utilizado o controle PID devido a sua facilidade de programação e por oferecer uma precisão satisfatória.

Na Figura 5 ilustra o controle PID para um manipulador com n graus de liberdade. O compensador é resultado da soma das componentes: proporcional, integral e derivativa. O parcela proporcional reduz o tempo de subida até a posição de referência. A parcela integral reduz o erro estacionário e a parcela derivativa melhora a estabilidade em regime permanente (OGATA; MAYA; LEONARDI, 2003).

Figura 5 – Diagrama de blocos de um robô com um compensador PID.



Fonte: Autoria Própria.

O controlador do tipo PID apresenta a seguinte relação:

$$U(s) = K_p E(s) + \frac{K_i}{s} E(s) + K_d s E(s) \tag{2.5}$$

Os parâmetros de controle são: Kp (constante de associada ao termo proporcional), Ki (constante associada ao termo integral e Kd (constante associada ao termo derivativo).A escolha dessas variáveis vão definir o desempenho final do sistema a ser controlado. À escolha destes parâmetros chama-se sintonia do PID (OGATA; MAYA; LEONARDI, 2003).

Em 1942 Ziegler e Nichols propuseram um metodo para sintonia de controladores atraves do ajuste das contastes: Kp, Ki e Kd, baseado na respota do sistema quadno este é submetido a um degrau de referência. É importante frisar que o método de Ziegler e Nichols é apenas uma estimativa, portanto é necessário uma sintonia fina até encontrar o modelo mais adequado (OGATA; MAYA; LEONARDI, 2003).

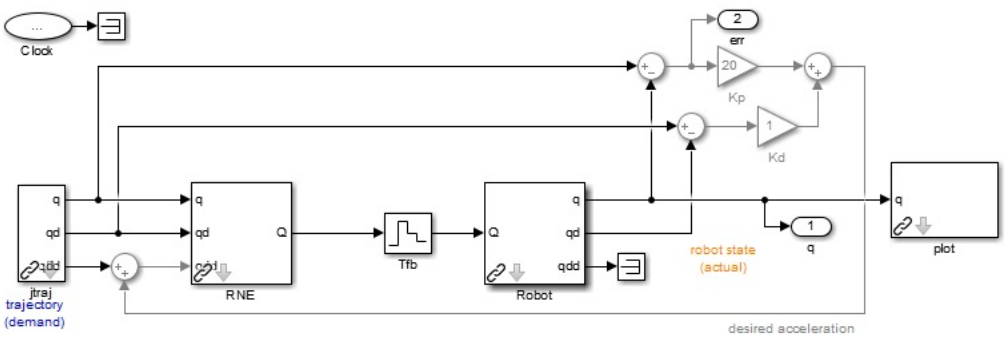
2.6 A FERRAMENTA DE ROBÓTICA DO MATLAB

Nesse trabalho será utilizado para controle e simulação do manipulador a ferramenta *Robotic Toolbox for MATLAB*. Desenvolvida pelo professor Phd. Peter Corke, ela está disponível gratuitamente no endereço: $http://petercorke.com/Robotics_Toolbox.html$.

Para o controle e simulação dos esforços e posição das juntas é necessário o ambiente de simulação Simulink. O ambiente Simulink é bastante utilizado para simulação e análise de sistema dinâmicos. A partir de um diagrama de blocos pode-se simular o modelo. A analise pode ser feita em qualquer ponto do diagrama, bem como fazer mudanças dos parâmetros do sistema e verificar as mudanças (CHAPMAN, 2003).

Na Figura 6 é ilustrado exemplo desse tipo de diagrama. Nesse exemplo o usuário fornece parâmetros de entrada um vetor de posição, um vetor de velocidade e um vetor de aceleração.

Figura 6 – Diagrama de blocos para o controle de torque.



Fonte: Adaptada de (CORKE, 2015).

A toolbox usa um método muito geral de representar a cinemática e a dinâmica dos manipuladores que é a ligação em série ou paralelo de blocos do *Simulink*. O objeto robô pode ser criado pelo usuário para qualquer manipulador. Uma série de exemplos são fornecidos, dentre os quais pode-se citar o robô Rethink, bem como robôs clássicos, como o Puma 560 e o braço de Stanford.

A ferramentas também suporta robôs de base móveis(monociclo, bicicleta), algoritmos de planejamento de trajetória, planejamento cinodinâmico, localização, mapa construção e localização e mapeamento simultâneos. A toolbox também inclui um modelo de *Simulink* detalhado para um robô voador quadrotor.

As vantagens do uso das funções da toolbox são as seguintes:

- O código é maduro e fornece um ponto de comparação para outras implementações dos mesmos algoritmos.
- As rotinas geralmente são escritas de forma direta, o que permite uma compreensão fácil, talvez à custa da eficiência computacional. Se o usuário tiver domínio sobre eficiência computacional, ele sempre poderá reescrever a função para ser mais eficiente, compilar o arquivo M usando o compilador *Matlab* ou criar uma versão MEX.
- Uma vez que o código fonte está disponível, existe um benefício para a compreensão e o ensino.

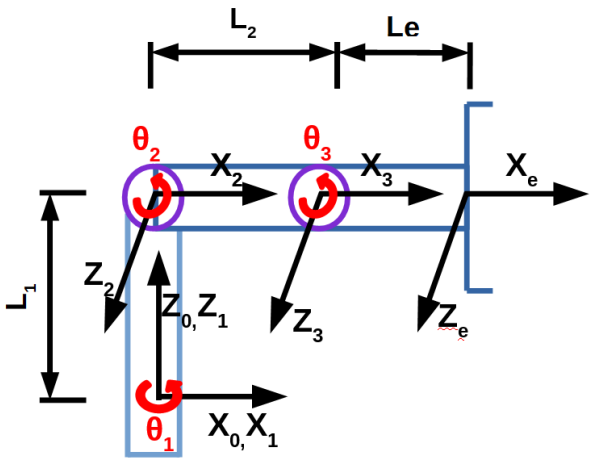
3 MATERIAIS E MÉTODOS

3.1 MODELAGEM DO MANIPULADOR

Nessa seção será estabelecido o conjunto de coordenadas responsáveis pela representação da posição dos elos, bem como equacionar o modelo que relacione a posição e movimento das juntas com a posição do efetuador final.

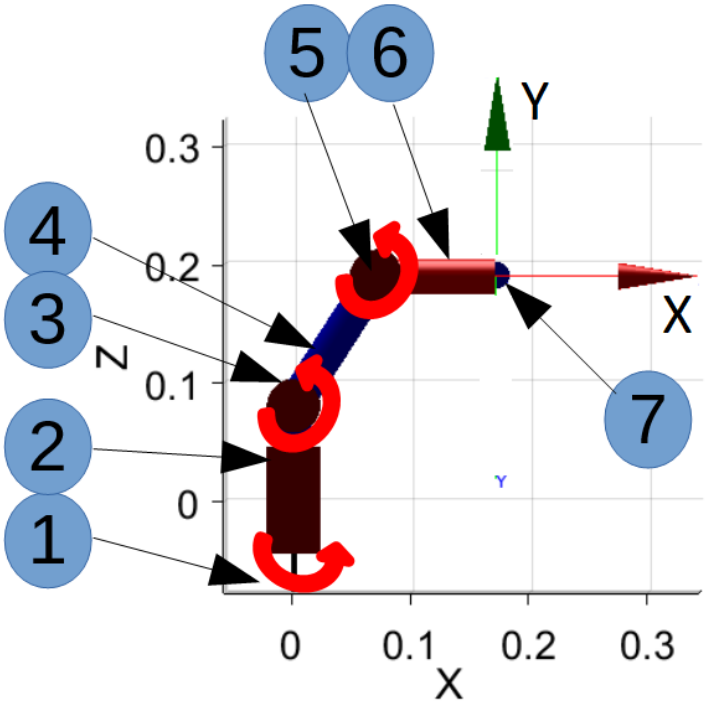
O braço consiste de elos de ligação unidos por juntas de movimento relativo, onde são acoplados os atuadores para realizarem estes movimentos individualmente, dotados de capacidade sensorial, e instruídos por um sistema de controle. O braço é fixado à base por um lado e na outra extremidade é composto por um efetuador final destinada a realizar a tarefa exigida pela aplicação. A orientação do efetuador final nas posições que correspondem à tarefa a ser realizada vai depender com conjunto de posicionamentos angular das juntas. A Figura 7 mostra esquematicamente uma sequência de elos e juntas do braço robótico estudado, e na Figura 8 ilustra o robô com o auxílio da toolbox, sendo essa ilustração algo mais próximo do resultado final do robô.

Figura 7 – Esboço do robô.



Fonte: Autoria Própria.

Figura 8 – Ilustração do robô com auxilio da toolbox.



Fonte: Autoria Própria.

A seguir na Tabela 1 mostra a descrição da Figura 8.

Tabela 1 – Descrição da ilustração da robô

Numero	Descrição
1	Junta 1
2	Elo 1
3	Junta 2
4	Elo 2
5	Junta 3
6	Elo 3
7	Efetuador final

3.1.1 Cinemática Direta

Com o auxílio da toolbox e usando a notação DH, a matriz de transformação homogênea foi criada.

A seguir na Tabela 2 os dados referentes aos parâmetros de Denavit–Hartenberg.

Tabela 2 – Parâmetros de Denavit-Hartenberg

i	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	0	0	θ_1
2	0	90	L_1	θ_2
3	L_2	0	0	θ_3
e	L_e	0	0	0

Aplicando a função *fkine* da toolbox, teremos como retorno a matriz de transformação homogênea.

Sendo $\mathbf{T}$ a posição do efetuador final do robô como uma matriz de transformação homogênea de quatro linhas e quatro colunas para a configuração conjunta $\mathbf{q}$, onde $\mathbf{q}$ é matriz de posição angular da juntas.

Sendo $\mathbf{q}$ uma matriz, as linhas são interpretadas como coordenadas das articulações para uma sequência de pontos ao longo de uma trajetória, e as colunas são os conjuntos de trajetórias. Neste caso, $\mathbf{T}$ é uma matriz de três dimensões (4x4x$\mathbf{N}$) onde $\mathbf{N}$ é o índice ao longo do caminho.

Essa função é de extrema importância, pois logo após de adicionadas as compensações conjuntas é necessário computar a cinemática direta para comparar a trajetória desejada com a trajetória real do manipulador.

$$
{}^{i-1}T_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -d_i\sin\alpha_{i-1} \\ \sin\theta_i\sin\alpha_{i-1} & \cos\theta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} & d_i\cos\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}
$$

Logo a seguir apresenta-se o equacionamento da matriz de transformação homogênea para cada junta. stas matrizes de transformação foram obtidas substituindo os parâmetros da Tabela 2 na matriz homogênea da Eq. (3.1).

$$
{}^{0}T_1 = \begin{bmatrix} C_{\Theta_1} & -S_{\Theta_1} & 0 & 0 \\ S_{\Theta_1} & C_{\Theta_1} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}
$$

$$
{}^{1}T_2 = \begin{bmatrix} C_{\Theta_2} & -S_{\Theta_2} & 0 & 0 \\ 0 & 0 & -1 & -L_1 \\ S_{\Theta_2} & C_{\Theta_2} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}
$$

$$ {}^{2}T_{3} = \begin{bmatrix} C_{\Theta_3} & -S_{\Theta_3} & 0 & L_2 \\ S_{\Theta_3} & C_{\Theta_3} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4} $$

$$ {}^{3}T_{4} = \begin{bmatrix} 1 & 0 & 0 & Le \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.5} $$

Por último, tem-se a matriz de transformação homogênea que relaciona a posição e orientação do efetuador final em relação ao sistema de coordenadas fixo da base, assim ${}^{0}T_{e}$:

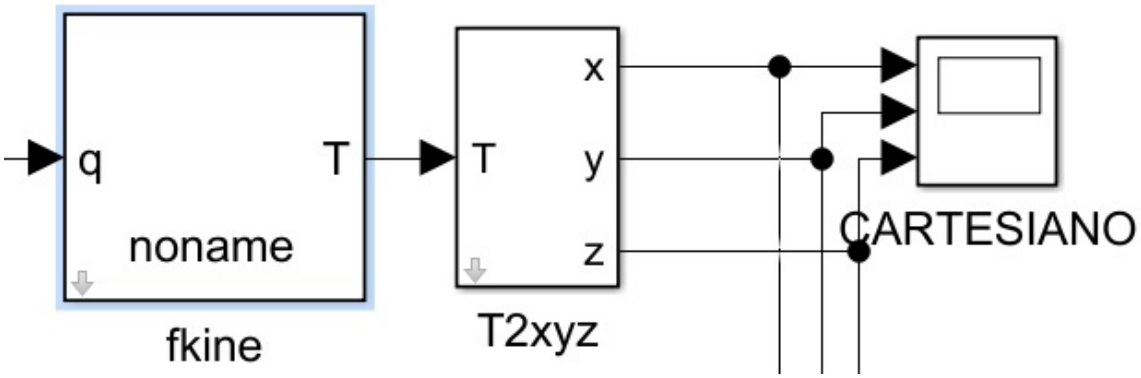
$$ {}^{0}T_{e} = {}^{0}\ T_{1}^{1}T_{2}^{2}T_{3}^{3}T_{4} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{3.6} $$

- $a_{11} = C_{\Theta_1}C_{\Theta_2}C_{\Theta_3} - C_{\Theta_1}S_{\Theta_2}S_{\Theta_3}$
- $a_{12} = -C_{\Theta_1}C_{\Theta_2}S_{\Theta_3} - C_{\Theta_1}S_{\Theta_2}C_{\Theta_3}$
- $a_{13} = S_{\Theta_1}$
- $a_{14} = L_e[C_{\Theta_1}C_{\Theta_2}C_{\Theta_3} - C_{\Theta_1}S_{\Theta_2}S_{\Theta_3}]L_2C_{\Theta_1}C_{\Theta_2} + L_1S_{\Theta_1}$
- $a_{21} = S_{\Theta_1}C_{\Theta_2}C_{\Theta_3} - S_{\Theta_1}S_{\Theta_2}S_{\Theta_3}$
- $a_{22} = -S_{\Theta_1}C_{\Theta_2}S_{\Theta_3} + S_{\Theta_1}S_{\Theta_2}C_{\Theta_3}$
- $a_{23} = -C_{\Theta_1}$
- $a_{24} = L_e[S_{\Theta_1}C_{\Theta_2}C_{\Theta_3} - S_{\Theta_1}S_{\Theta_2}S_{\Theta_3}]L_2S_{\Theta_1}C_{\Theta_2} - L_1C_{\Theta_1}$
- $a_{31} = S_{\Theta_2}C_{\Theta_3} + C_{\Theta_2}S_{\Theta_3}$
- $a_{32} = -S_{\Theta_2}S_{\Theta_3} + C_{\Theta_2}C_{\Theta_3}$
- $a_{33} = 0$
- $a_{34} = L_e[S_{\Theta_2}C_{\Theta_3} + C_{\Theta_2}S_{\Theta_3}] + L_2S_{\Theta_2}$
- $a_{41} = 0$
- $a_{42} = 0$
- $a_{43} = 0$

- $a_{44} = 1$

Logo após com o auxílio da função *T2xyz* pode-se encontrar os pontos no espaço cartesiano; desta forma, a cinemática direta que relaciona a posição do efetuador final em função da posição das juntas é determinada. Logo abaixo na Figura 9 é ilustrado essa interação para o calculo da cinemática direta.

Figura 9 – Cinemática direta.



Fonte: Autoria Própria.

A importância desse conceito está no fato de que a partir de uma coordenada angular dos elos, o operador descobrirá a posição da extremidade do último elo do robô. A movimentação do robô é feita modificando o conjunto de coordenadas das juntas definis por $q = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \end{bmatrix}^T$, desse modo a movimentação do manipulador é feita de forma gradativa.

3.1.2 Cinemática Inversa

O método usado pela toolbox para fazer os cálculos da cinemática inversa é a função *ikine*. Com esta função, o usuário entra com os dados de trajetória no espaço cartesiano e com os parâmetros da geometria do robô. Como resultado a função retorna um vetor **q** das variáveis das juntas para a respectiva trajetória em radianos.

No entanto, a função *ikine*, não apresentou uma resposta numérica satisfatória para o robô analisado neste trabalho. Sendo assim, foi necessário desenvolver um algoritmo para executar a cinemática inversa levando em conta todas as limitações de projeto.

Logo abaixo as equações necessárias para solucionar o problema da cinemática inversa. O software detalhado que foi desenvolvido bem como a geração de trajetória constam na seção de anexos.

O manipulador conta com 3 elos, o elo L_1 é fixo e os elos L_2 e L_e são moveis. Para alcançar uma posição final o efetuador deve posicionar-se de modo que cada junta vai ter um

angulo em relação a origem. Sendo: θ_1 o ângulo da junta 1, θ_2 o angulo da junta 2 e θ_3 o angulo da junta 3.

Obter o modelo cinemático inverso pela forma geométrica consiste em decompor a geometria espacial do manipulador em varias formas de geometria plana.

Sabendo que:

$$\theta_1 = \arctan^2(\frac{p_y}{p_x}) \tag{3.7}$$

$$p_x = \cos(\theta_1)[L_1 + L_2\cos(\theta_2) + L_e\cos(\theta_2 + \theta_3)] \tag{3.8}$$

$$\frac{p_x}{\cos(\theta_1)} - L_1 = L_2\cos(\theta_2) + L_e\cos(\theta_2 + \theta_3) \tag{3.9}$$

$$p_y = \sin(\theta_1)[L_1 + L_2\cos(\theta_2) + L_e\cos(\theta_2 + \theta_3)] \tag{3.10}$$

$$\frac{p_y}{\sin(\theta_1)} - L_1 = L_2\cos(\theta_2) + L_e\cos(\theta_2 + \theta_3) \tag{3.11}$$

Elevando os dois membros ao quadrado tem-se:

$$L_2\cos(\theta_2) + L_e\cos(\theta_2 + \theta_3) = \alpha \tag{3.12}$$

$${L_2}^2\cos(\theta_2)^2 + {L_e}^2\cos^2(\theta_2 + \theta_3) + 2L_1L_2\cos(\theta_2)\cos(\theta_2 + \theta_3) = \alpha^2 \tag{3.13}$$

$$L_2\sin(\theta_2) + L_e\sin(\theta_2 + \theta_3) = p_z \tag{3.14}$$

$$L_2^2\sin(\theta_2)^2 + {L_e}^2\sin^2(\theta_2 + \theta_3) + 2L_1L_2\sin(\theta_2)\sin(\theta_2 + \theta_3) = p_z^2 \tag{3.15}$$

$${L_2}^2 + {L_3}^2 + 2L_2L_e\cos(\theta_3) = \alpha^2 + {p_z}^2 \tag{3.16}$$

Para o ângulo da terceira junta, tem-se:

$$\cos(\theta_3) = \frac{\alpha^2 - {p_z}^2 - {L_2}^2 - -{L_3}^2}{2L_2L_e} \tag{3.17}$$

$$\sin(\theta_3) = \sqrt{1 - \cos^2(\theta_3)} \tag{3.18}$$

$$\theta_3 = \arctan^2(\frac{\sin\theta_3}{\cos\theta_3}) \tag{3.19}$$

Analogamente para o angulo da segunda junta tem-se:

$$k_1 = L_1 + L_2\cos(\theta_2) \tag{3.20}$$

$$k_2 = L_2\sin(\theta_2) \tag{3.21}$$

$$\gamma = \arctan^2(\frac{k_2}{k_1}) \tag{3.22}$$

$$\theta_2 = \arctan^2(\frac{\alpha}{p_z}) - \gamma \tag{3.23}$$

3.1.3 Simulação do Modelo Dinâmico

No modelo dinâmico apresentado pela toolbox as informações referentes aos parâmetros dinâmicos dos elos são apresentadas em forma de matriz.

A seguir na Tabela 3 os parâmetros usados na simulação.

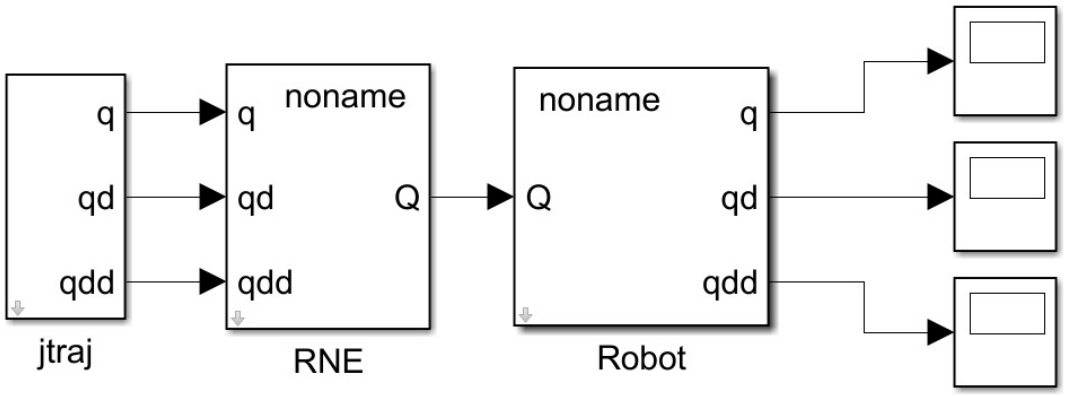
Tabela 3 – Parâmetros usados na simulação dinâmica.

Parâmetro	Junta 1	Junta 2	Junta 3
$\theta_\mathbf{i}$	0	0	0
$\mathbf{d_i}$	0.08	0	0
$\mathbf{a_{i-1}}$	0	0.13	0.1
$\alpha_{\mathbf{i-1}}$	pi	0	0
$\sigma_\mathbf{i}$	0	0	0
m[kg]	0.45	0.5	0.4
rx[m]	0	0.01	0.01
ry[m]	0.03	-0.03	0.03
rz[m]	0.02	0	0
Ixx[kg/m^2]	0	0	0
Iyy[kg/m^2]	0	0	0
Izz[kg/m^2]	0	0	0
Ixy[kg/m^2]	0	0	0
Iyz[kg/m^2]	0	0	0
Ixz[kg/m^2]	0	0	0
Jm[kg/mm^2]	$2 * 10^-7$	$2 * 10^-7$	$2 * 10^-7$
G	50:1	50:1	50:1
Bm[Nm/rad/s]	$9 * 10^-9$	$9 * 10^-9$	$9 * 10^-9$
Tc[Nm]	0	0	0

A formulação recursiva de Newton-Euler calcula a dinâmica inversa do manipulador; Para isto, a função *RNE* é utilizada na toolbox no *Matlab*. Consequentemente, a dinâmica inversa calcula os binários nas juntas necessários para uma determinado posição, velocidade e aceleração das juntas. A recursão direta propaga informações da cinemática tais como velocidades angulares, acelerações angulares, acelerações lineares da base de referência inercial para o efetuador final. A recursão de Newton-Euler retorna as forças e os momentos exercidos em cada ligação do manipulador.

A dinâmica direta do manipulador, isto é, a posição angular, velocidade e aceleração das juntas depois de aplicada uma determinada força pelo atuador é feita pela função *Robot*. A Figura 10 mostra as variáveis envolvidas no cálculo da dinâmica para um sistema.

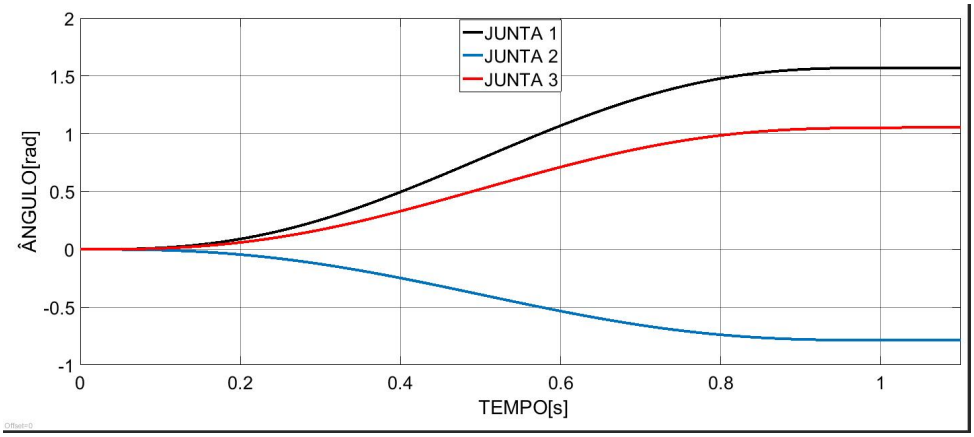
Figura 10 – Funções do modelo dinâmico.



Fonte: Autoria Própria.

A Figura 11 ilustra a posição angular das juntas para uma trajetória de $q_0 = [0\ 0\ 0]$ até $q_f = [\pi/2\ \ -\pi/4\ \pi/3]$ no decorrer de 1 segundo. Logo após, outras figuras ilustrarão o torque, aceleração e velocidade.

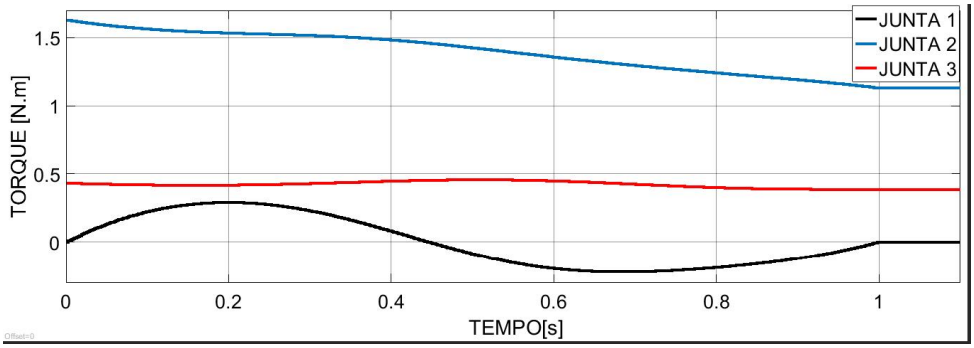
Figura 11 – Posição angular das juntas.



Fonte: Autoria Própria.

O torque em função do tempo para a trajetória, para cada uma das juntas pode ser observado na Figura 12.

Figura 12 – Torque em função do tempo.



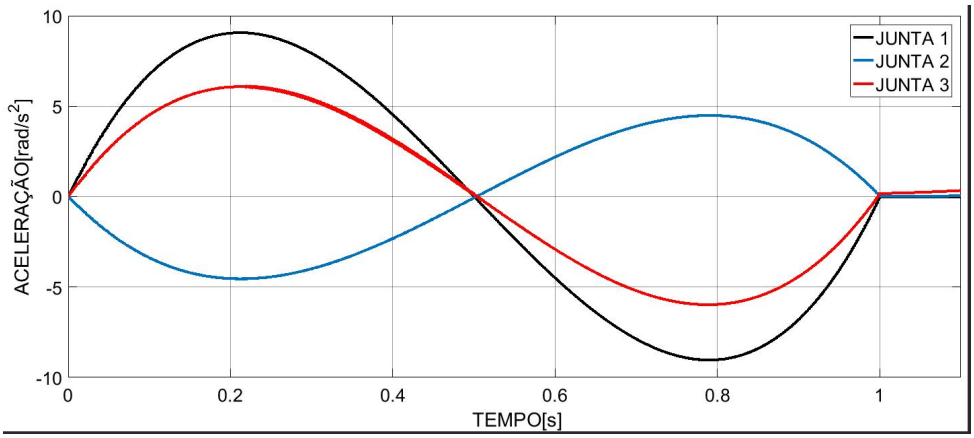
Fonte: Autoria Própria.

Na Figura 12 a junta 1 ilustrada pela linha preta é aquela que requer um torque menor, pois essa junta apenas sofre os esforços causados pelo robô em torno do seu eixo. A junta 2 ilustrada em azul é aquela que requer um maior torque do motor, pois a junta sofre com os esforços causados pela inercia do robô quando o mesmo sai do repouso e para segurar o peso do segundo e terceiro elo. A junta 3 ilustrada em vermelho não requer tanto torque, pois essa apenas vai segurar o peso do segundo elo.

A aceleração angular em função do tempo para a trajetória, para cada uma das juntas pode ser observado na Figura 13. Na ilustração, nota-se que a amplitude a aceleração varia de acordo com o deslocamento angular da junta, ou seja, quanto maior o deslocamento, maior será a sua aceleração. Sendo que na trajetória tem-se a aceleração máxima em 1/4 do tempo da

trajetória, aceleração igual a zero na metade do tempo e aceleração minima em 3/4 do tempo da trajetória.

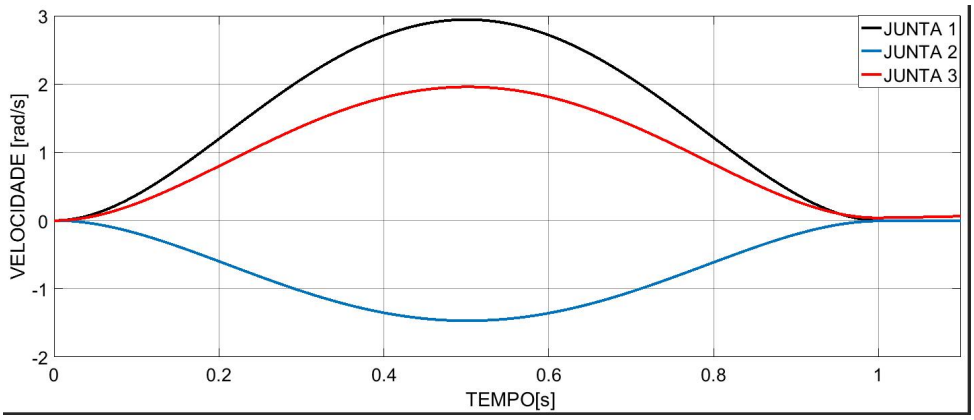
Figura 13 – Aceleração em função do tempo.



Fonte: Autoria Própria.

A velocidade angular em função do tempo para a trajetória, para cada uma das juntas pode ser observado na Figura 14. Assim como a aceleração, a amplitude máxima da velocidade depende da sua trajetória, e sempre o valor máximo da velocidade da junta será na metade do tempo da trajetória.

Figura 14 – Velocidade em função do tempo.



Fonte: Autoria Própria.

3.2 SIMULAÇÃO COMPUTACIONAL

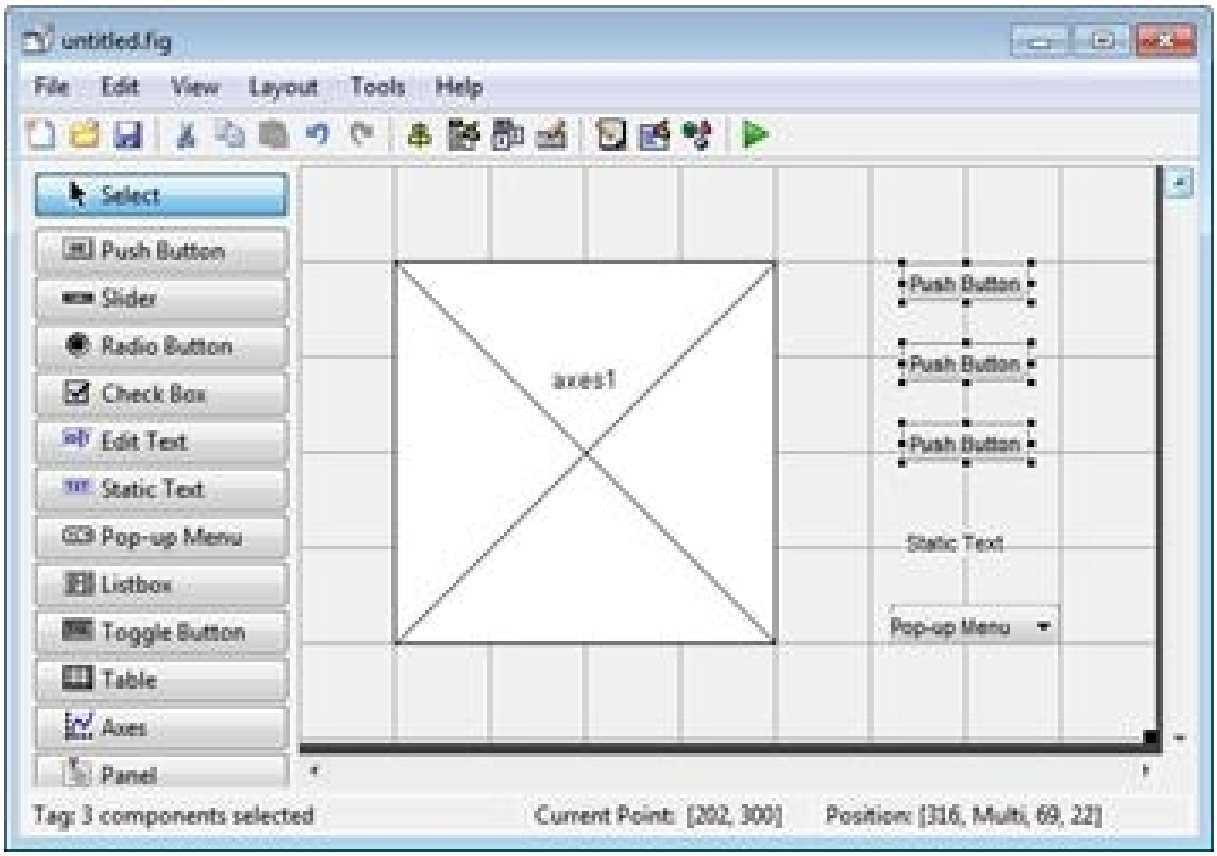
Com a toolbox do *Matlab* será feira a simulação da trajetória percorrida e especificada pelas coordenadas Cartesianas do efetuador final. Adicionalmente, a dinâmica também será si-

mulada para esta trajetória.

Para isto, foi desenvolvida uma GUI (também conhecidas como interfaces gráficas de usuário ou UIs) . As GUIs fornecem o controle de aplicativos de software, eliminando a necessidade de aprender a linguagem de programação ou digitar comandos para executar o aplicativo.O aplicativo GUI automatizam uma tarefa ou cálculo. A GUI normalmente contém controles como menus, barras de ferramentas, botões e controles deslizantes.

Usando o GUIDE Layout Editor foi desenvolvido um código no *MATLAB* para a construção da interfase de usuário conforme apresentado na Figura 15.

Figura 15 – GUIDE MATLAB.



Fonte: Mathworks.

Alguns dos elementos importantes para elaboração deste projeto serão listados abaixo, com uma pequena descrição de seu funcionamento.

- Radio Button: Botão de seleção, normalmente utilizado em conjunto com outros Radio Buttons. O evento se dá ao selecionar o botão.
- Edit Text: Caixa de texto editável que pode aceitar uma ou mais linhas de entrada e mostrar um texto inicial. O evento se dá ao pressionar a tecla Enter com o cursor na caixa de texto, dentre outras formas.

- Static Text: Caixa de texto não editável, normalmente não necessita de um callback. Exibe uma string estática.
- Pop-Up Menu: Menu que pode ser expandido para englobar diversas opções de forma compacta. O evento se dá ao selecionar uma das opções.
- Axes: Elemento que permite exibir gráficos e imagens.
- Panel: Elemento que cria um painel com bordas e título, tornando a GUI mais organizada e fácil de compreender. Normalmente não necessita de um callback.

O desenvolvimento da *GUIDE* levou em conta a facilidade de uso pelo usuário final, com isso buscou-se algo simples, funcional e uma organização eficaz dos elementos de interface.

Os elementos da guide podem ser resumidos em:

- Entrada de dados para definição da trajetória a ser seguida.
- Simulação do sistema em malha fechada.
- Visualização dos gráficos que vão dar informações valiosas para o usuário final.

Para que os componentes acima fossem incorporados na GUI projetada, optou-se por criar um layout com três partes distintas: Definição da trajetória, Simulação e visualização de gráficos. A divisão em três partes permite organizar a explicação e uso *GUIDE* de maneira clara e objetiva evitando que as informações exibidas ao usuário fiquem congestionadas, devido à falta de espaço na tela ou mal uso. Os objetivos a serem alcançados no desenvolvimento da *GUIDE* foram definidos para garantir que a mesma operasse de maneira mais eficiente levando em conta as interações com o usuário que estão resumidas abaixo:

- Permitir que os dados a serem definidos pelo usuário sejam carregados a partir de dados digitados diretamente na *GUIDE*.
- Automatizar as tarefas que devem ser em sua maioria executadas pela *GUIDE*.
- Mostrar ao usuário, por meio de informações no *comand windows*, quando ocorrer um problema no cálculo da trajetória.
- Permitir que o usuário ajuste o tempo de simulação e a escala dos gráficos.
- Seleção simples de quais gráficos devem ser apresentados na tela.

Para todo o desenvolvimento descrito da GUIDE utilizou-se o software *MATLAB R2016b*. A release *2016b* foi a escolhida, pois era a que tinha compatibilidade com a ultima versão da toolbox. No anexo E apresenta-se a forma que o usuário deverá proceder para usar a GUIDE.

3.3 ANÁLISE PREVIA PARA O DESENVOLVIMENTO DO PROTÓTIPO

Esta seção apresentará uma discussão sobre a construção da estrutura mecânica do robô e o seu sistema de controle. Com este proposito, os softwares de CAD *Solidworks* 2016 e *Eagle* foram utilizados no projetos apresentados. Adicionalmente, o sistema no quadro elétrico e suas ligações serão apresentados, sendo estas de extrema importância para uma maior organização da ligação dos componentes que atuam no sistema controle e posição. E por fim será apresentada uma visão geral do protótipo.

Torna-se fundamental avaliar diferentes produtos para as diferentes partes que irão compor os subsistemas do projeto em questão, levando em conta as características dos mesmos e os requisitos do produto.

Assim, foram analisadas as possibilidades para as quatro principais partes do braço manipulador: Prototipagem mecânica e eletrônica; motor elétrico; IDE e linguagem de programação.

3.3.1 Placa microcontroladora do robô

Na tabla 4 apresenta-se uma análise comparativa entre as diferentes placas para prototipagem de sistemas de controle disponíveis no mercado.

Tabela 4 – Comparação de preços de plataformas de desenvolvimento.

Nome	***Arduino***	**Beaglebone**	**Raspberry PI**
Modelo	Mega2560	Rev B	Model B+
Preço	R$ 80,00	R$ 400,00	R$ 200,00
Processador	ATmega2560	ARM Cortex - A8	ARM Cortex – A6
Freq. Clock (max.)MHz	16	800	400
RAM	8 KB	512 MB	512 MB
GPIO	70	65	40

Com base nesta análise comparativa, a Plataforma *Arduino* Mega se torna suficiente no desenvolvimento da placa de controle. devido ao seu baixo custo e elevado desempenho em relação as outras placas.

A primeira versão do braço robótico utilizará *encoders* magnéticos, sendo desnecessária a utilização de microcontroladores com alta frequência de *clock*.

No projeto o codigo será embarcado, evitando assim um atraso do sinal no processamento ocasionado pelo baixo poder de processamento do sistema quando integrado com o *simulink*.

3.3.2 Acionamento das Juntas

O que se necessita é de um motor que receba como sinal de entrada um ângulo de referência e que o eixo do motor se mova para a referência recebida, ou seja, um servo-motor cumpre com o requisito, mas devido a sua pouquíssima precisão ele não será utilizado. Um motor que cumpre todos os requisitos com eficiência é um motor DC com um encoder magnético. Tal dispositivo nada mais é que um motor de corrente contínua com um gerador de pulsos. No mercado existem modelos muito complexos e caros, sendo necessário um grande investimento. A desvantagem desse acionamento é preço e complexidade de controle. Sendo necessário um grande investimento e um usuário com pleno conhecimento no assunto. Na Tabela 5 apresenta-se o motor utilizado no projeto do robô.

Tabela 5 – Informações motor DC

GEAR MOTOR: GM37-545S-50-21D	
TENSÃO	DC 12V
VELOCIDADE SEM CARGA	110 RPM 0.3A
MÁXIMA EFICIÊNCIA	55 N.M, 93 RPM
MÁXIMA POTENCIA	1.55 N.M, 60 RPM
STALL CORRENTE	7A
RESOLUÇÃO	550 PPR
REDUÇÃO	50:1

3.3.3 Interface de Programação

A interface entre o mecanismo robótico e o usuário é feito através da interface de programação, que é a área onde o usuário comandará os motores através de comandos e funções pré-definidos. Essa interface foi construída com a IDE *Arduino*, a qual tem como linguagem de programação C++. Um dos maiores motivos que levou à escolha desse ambiente integrado de desenvolvimento foi a experiência adquirida durante o curso com o uso desta ferramenta, que apresenta grandes vantagens, como:

- Linguagem de alto nível orientada a objeto.
- Integração com a API do *Windows*, o que permite a criação de programas que exploram ao máximo os recursos do sistema operacional.
- Compilador que gera arquivos executáveis nativos, ou seja, em código de máquina, tornando-o extremamente rápido e com proteção do código fonte.
- A IDE *Arduino* pode ser ampliada e personalizada com a adição de componentes e ferramentas criadas utilizando-se a linguagem C++.

3.3.3.1 Integração dos Recursos Tecnológicos

A implementação computacional do projeto, com ênfase na criação da interface de programação e a geração de trajetória do órgão terminal é usada para enviar comandos ao *Arduino* de forma imediata, podendo ou não movimentar o robô, bem como mostrar informações na forma de gráficos em tempo real. O interpretador será construído na IDE do *Arduino* e nele gravado. A geração de trajetória é uma função criada na *Guide* do *MatLab*.

3.4 PARTE MECÂNICA

A seguir serão apresentados aspectos relativos à instalação dos atuadores e dimensionamento das peças que compõem o mecanismo do robô.

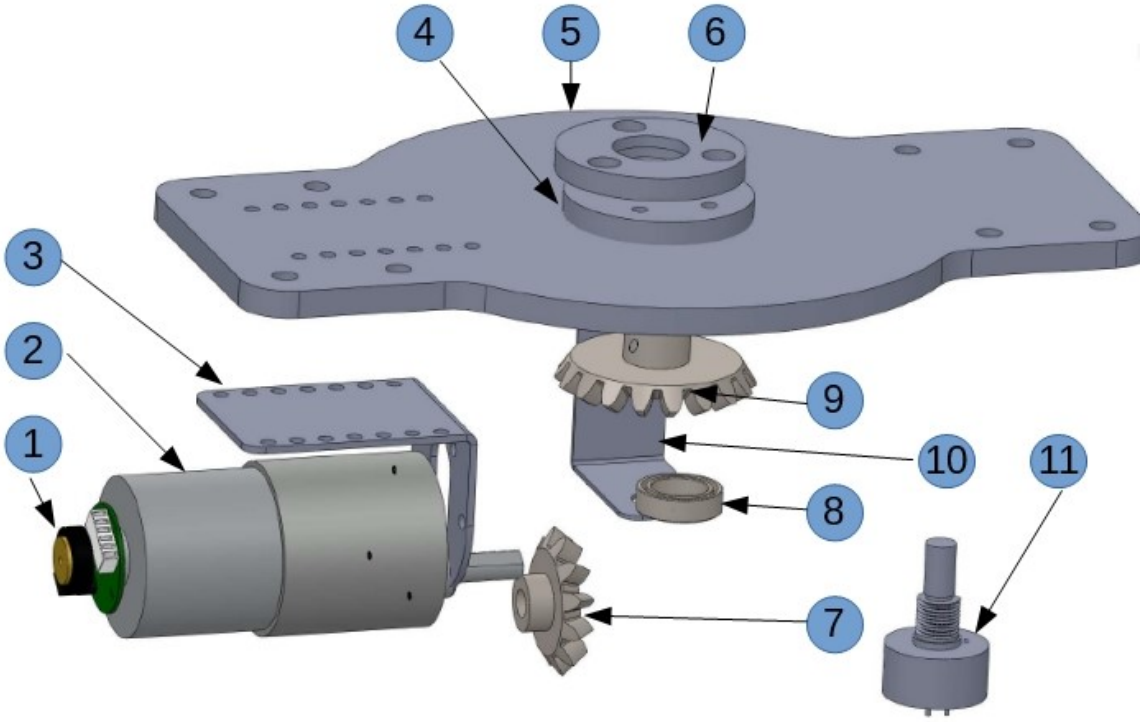
O bom funcionamento do sistema depende diretamente de uma correta fixação dos componentes a base do robô. Uma movimentação indevida de componentes do mecanismo pode, além de atrapalhar o funcionamento, danificar os componentes eletrônicos e mecânicos contidos no robô.

A seguir apresenta-se o projeto mecânico dos elos e a montagem final do mecanismo do robô.

3.4.1 Elos

Na estrutura escolhida para a montagem do robô, os esforços relativos ao peso dos componentes se concentram na base. O arranjo diferencial de engrenagens é composto por um pinhão e uma coroa, que junto com o motor 1 são responsáveis pela movimentação do robô em torno de sua base. A base tem a função de ser um ponto de apoio para estrutura, para fins de estabilidade do robô. A seguir na Figura 16 uma vista expandida dos componentes do elo 1 e na Tabela 6 uma descrição dos componentes.

Figura 16 – Elo 1.



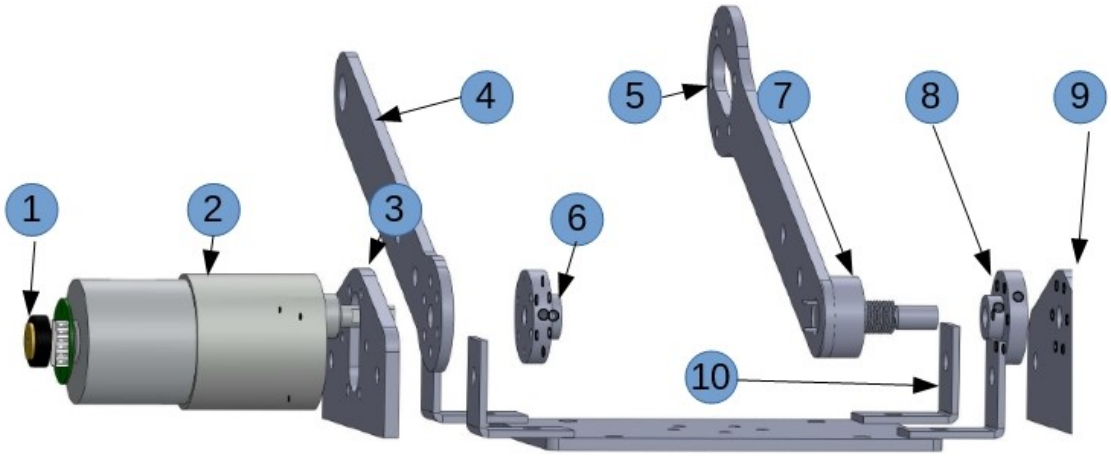
Fonte: Autoria Própria.

Tabela 6 – Nomenclatura das partes do elo 1

Peça	Nome
1	Encoder 1
2	Motor 1
3	Base motor
4	Base de revolução
5	Base do robô
6	Tampa de revolução
7	Pinhão
8	Base do potenciômetro
9	Coroa
10	Cantoneira do potenciômetro
11	potenciômetro

Na montagem do robô os esforços relativos ao peso do elo 2 e elo 3 se encontram principalmente nas quatro cantoneiras de apoio. O motor 2 e as hastes acopladas ao rotor são responsáveis pela movimentação do elo 2. A seguir na Figura 17 uma vista expandida dos componentes do elo 1 e na Tabela 7 uma descrição dos componentes.

Figura 17 – Elo 2.



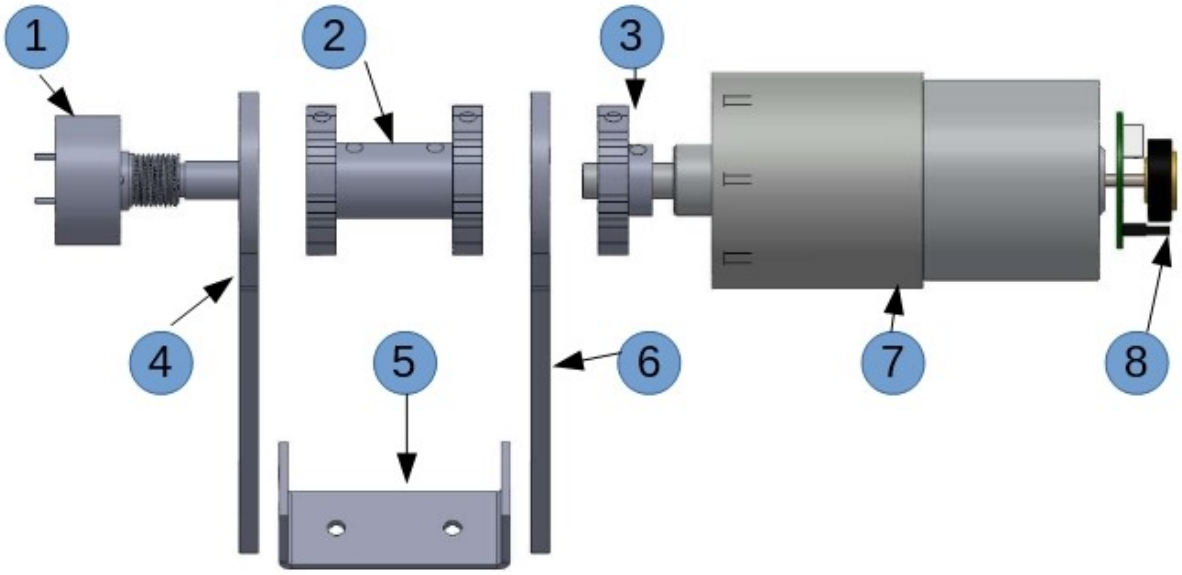
Fonte: Autoria Própria.

Tabela 7 – Nomenclatura das partes do elo 2

Peça	Nome
1	Encoder 2
2	Motor 2
3	Apoio lado esquerdo
4	Lado esquerdo do elo
5	Lado direito do elo
6	Acoplamento motor
7	Potenciômetro
8	Acoplamento do Potenciômetro
9	Apoio lado direito
10	Cantoneira

Na montagem do robô os esforços relativos ao peso do elo 3 se encontram principalmente no conjunto de acoplamentos. O motor 3 e as hastes acopladas ao rotor são responsáveis pela movimentação do elo 3. A seguir na Figura 18 uma vista expandida dos componentes do elo 1 e na Tabela 8 uma descrição dos componentes.

Figura 18 – Elo 3.



Fonte: Autoria Própria.

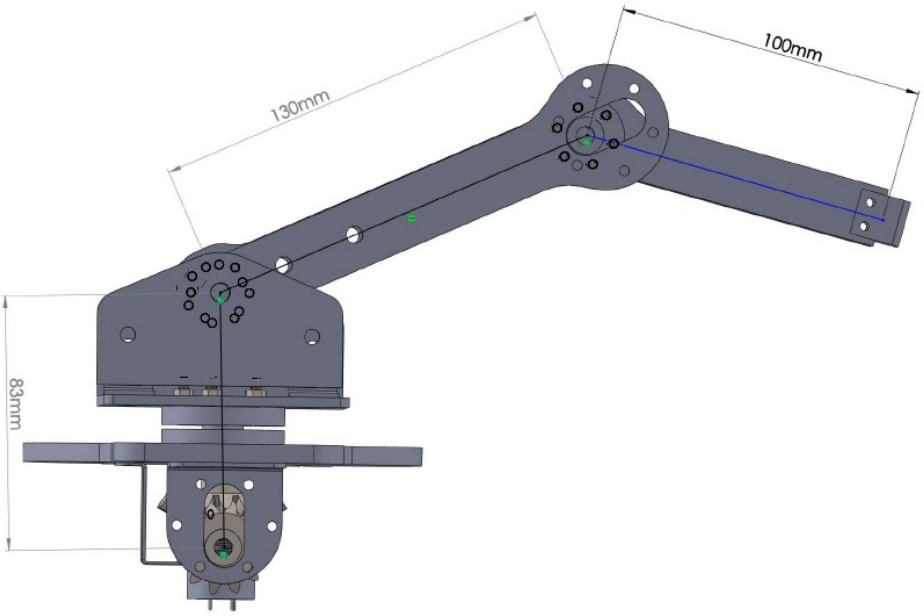
Tabela 8 – Nomenclatura das partes do elo 3

Peça	Nome
1	Potenciômetro
2	Acoplamento duplo
3	Acoplamento motor
4	Lado esquerdo elo
5	Base para garra
6	Lado direito elo
7	Motor 3
8	Encoder 3

3.4.2 Montagem Final

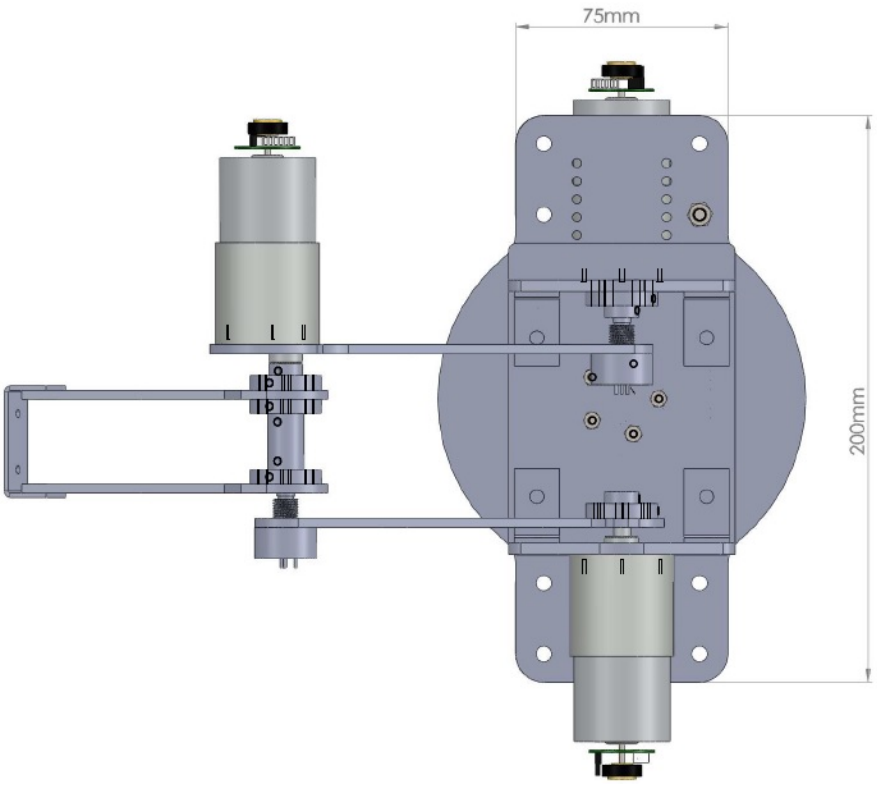
Uma vez feitas as considerações necessárias em relação à estrutura do robô dividida em três partes, pode-se apresentar a montagem final do protótipo, modelada no *Solidworks*. A Figura 19 ilustra a vista lateral do robô, a Figura 20 ilustra a vista superior do robô e por ultimo a Figura 21 ilustra a vista isométrica do robô.

Figura 19 – Robô completo vista lateral



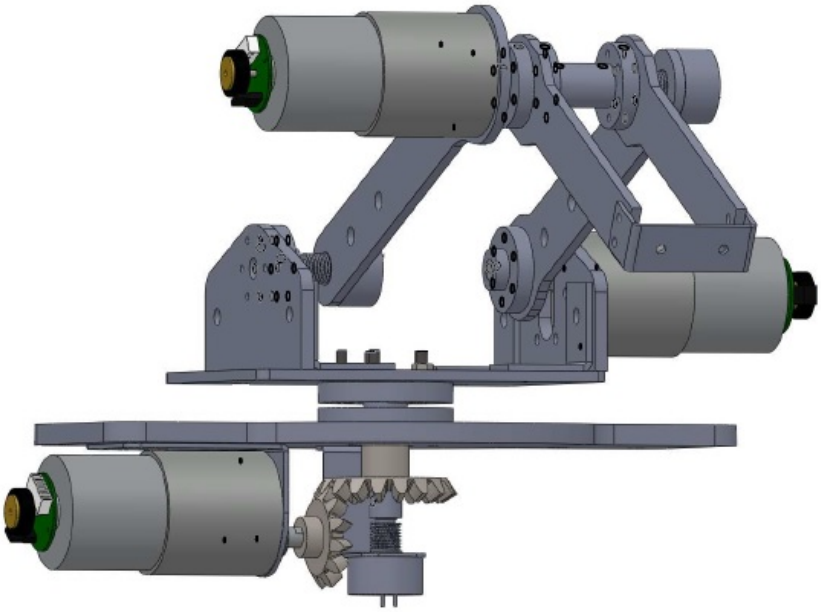
Fonte: Autoria Própria.

Figura 20 – Robô completo vista superior.



Fonte: Autoria Própria.

Figura 21 – Robô completo vista isométrica.



Fonte: Autoria Própria.

3.4.3 Considerações Finais

Nesta seção foram apresentados os aspectos relativos à construção de um protótipo para o robô. É importante ressaltar que todas as modelagens apresentadas foram produzidas com o intuito de comporem um protótipo para testes, e de fabricação barata. Para a fabricação em série seria necessária à análise da durabilidade, preço de revenda, otimização de medidas, e outros aspectos que influenciam na fabricação de um produto final.

3.5 SISTEMAS ELETRÔNICOS

Uma vez que o design mecânico esteja disponível, e sua força e mobilidade, é necessário projetar um estágio que permita que os atuadores se comuniquem com um computador para seu controle e outro estágio para fornecer a força necessária para dar movimento ao sistema. Ambos os estágios devem funcionar em coordenação para o posicionamento corretamente cada junta, de acordo com uma determinada configuração especial desejada.

O primeiro estágio é projetar a eletrônica de potência, que será responsável por fornecer a energia necessária aos atuadores do sistema robótico para posicionar cada uma das juntas. Esta etapa deve ser projetada para poder fornecer energia, mesmo em casos críticos, onde os valores

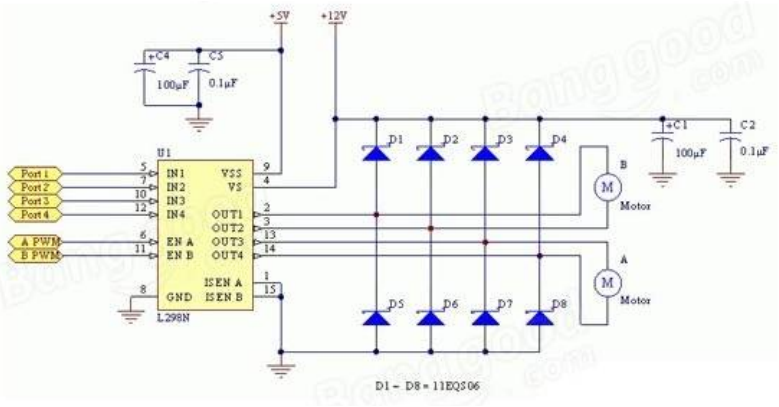
exigidos são muito grande; Por exemplo, ao iniciar o movimento de uma junta e é necessário superar a inércia inerente a cada parte do sistema.

A segunda etapa será responsável por fornecer controle ao sistema para ser capaz de determinar a posição a ser adotada pelas juntas. Nessa etapa sistemas de aquisição de dados vão influenciar na velocidade de processamento.

3.5.1 Eletrônica de Potência

É necessário que a fase de controle seja capaz de manipular e controlar os sinais do sistema robótico para assegurar a convergência do robô aos pontos desejados. No entanto, é muito importante ter um estágio capaz de fornecer energia suficiente para todo o sistema, enquanto controla o mesmo. Sem esse sistema o motor não vai conseguir atingir a posição desejada. Os sinais de controle do motor (PWMs) são introduzidos no estágio de potência, o que proporcionará a energia necessária aos atuadores para girar e parar.

Figura 22 – Chip L298N.



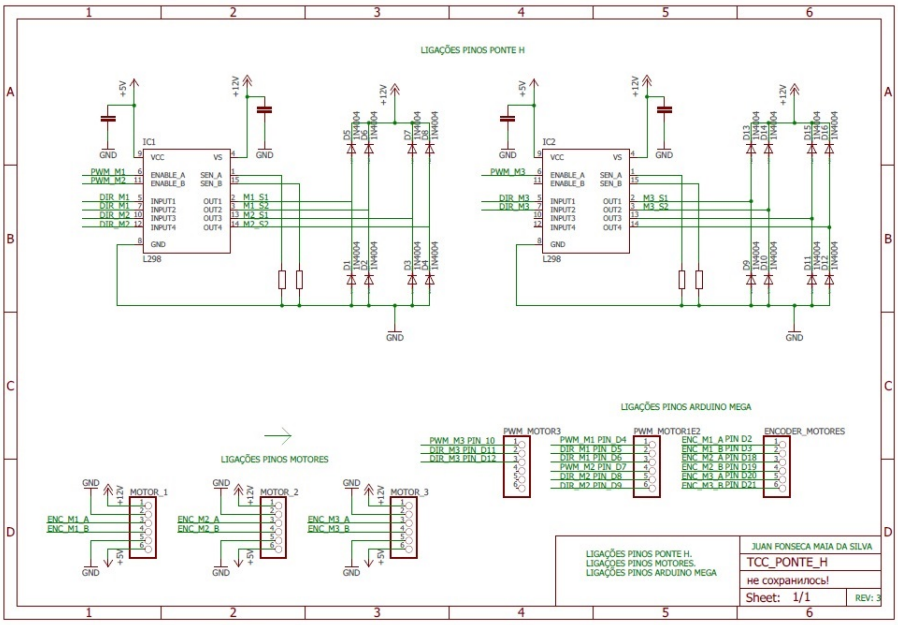
Fonte: Autoria Própria.

O estágio de potência projetado foi baseado nos requisitos atuais do chip L298N, que contém 2 pontes H por chip. O esquema de ligação do chip pode ser visto na Figura 22. A corrente máxima que suporta cada ponte no modo intermitente é 3A, em modo contínuo (80% ON, 20% OFF) suporta 2.5A e na operação constante até 2A.

Os níveis de controle e potência funcionam juntos para monitorar e mover as juntas do sistema. Para conseguir movimentos e velocidades de juntas precisas, um bom controlador deve ser programado para responder a distúrbios externos e colocar o sistema na posição desejada no menor tempo possível e com o menor número de oscilações.

A seguir a Figura 23 ilustra o esquema de ligação entre o chip L298 e as portas do *Arduino* Mega.

Figura 23 – Ligações de L298 e *Arduino*.



Fonte: Autoria Própria.

3.5.2 Eletrônica de controle

Para o controle de uma junta do sistema robotizado as variáveis são as seguintes:

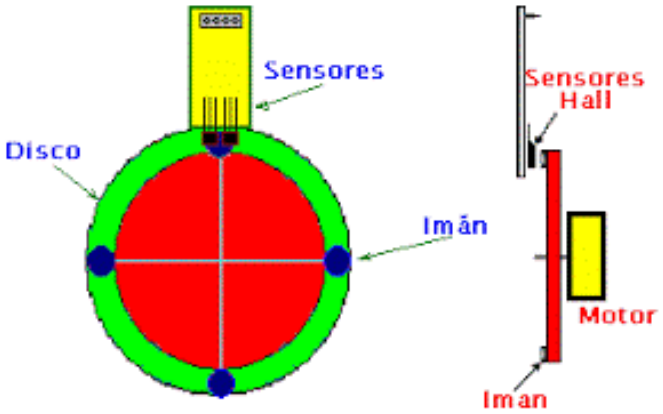
- Fonte de alimentação 12VDC para motores.
- Fonte de alimentação 5VDC para circuitos integrados e *encoders* de posição.
- 2 sinais de entrada digitais para detecção de posição.
- 2 sinais de saída digitais para a posição PWM e controle de rotação.
- 1 sinal de saída digital para intensidade do PWM.

O sistema completo possui 3 motores, o que torna necessário ter 6 sinais de entrada digital, 6 sinais de saída digital e 3 sinais de saída digital para PWM. O sistema pode ser estendido para 4 motores, apenas acrescentando mais uma fiação para um quarto motor.

O transdutor do *encoder* é um sensor de efeito Hall. Um sensor de efeito Hall é capaz de detectar um campo magnético. Ao detectar a passagem de dentes de engrenagem ou um ímã, o sensor de efeito Hall pode gerar uma saída de trem de pulso semelhante à produzida por um codificador óptico. Além disso, o uso de sensores de efeito Hall vai gerar uma saída em quadratura para determinar a direção do movimento.

O motor em particular utilizado no experimento pode ser conduzido por uma fonte de corrente continue de 12V. O codificador fornece 44 contagens por revolução (se contar as bordas ascendentes e descendentes). O motor também inclui uma caixa de velocidades de modo que o sensor de efeito Hall gera 2200 contagens por revolução no eixo de saída da caixa de redução. A configuração do motor com encoder e sua conexão com a placa *Arduino* é mostrada na Figura 24.

Figura 24 – Sensor efeito hall.



Fonte: Adaptado de (PROYECTOS. . . ,).

Para manipular esses sinais, um Arduino *Mega* (ver Figura 25) foi utilizado, onde todos os sinais de entrada e saída necessários estão conectados.

Figura 25 – *Arduino* Mega.

Fonte: Autoria Própria.

O procedimento de controle consiste em ler os sinais dos codificadores e, por meio de Um algoritmo de controle, determine a posição e a velocidade das articulações bem como tomar as decisões de movimento e/ou mudança de rotação. O programa de controle gera sinais necessários para manipular o sistema e posicionar as articulações para que o sistema possa alcançar uma posição específica. Este algoritmo de controle está programado para interagir com o sistema robótico em tempo real e exibir as informações através do monitor. A programação está desenvolvida de forma embarcada e permite incluir qualquer esquema de controle, seja o controle convencional de PID e/ou inteligentes (lógica difusa, redes neurais,Linear, etc.). Os sinais de controle gerados passam do microcontrolador para o estágio de potência. Esses sinais são modificados ou amplificados para ser enviado mais tarde para os motores e produzir o movimento necessário para alcançar configuração desejada.

O esquema de controle implementado foi o controle PID. O software *Simulink* é capaz de simular a programação para qualquer esquema de controle, mas em virtude de uma baixa frequência de aquisição de dados o controle de sistemas rápidos como motores DC é inviável.

O processo de controle segue o esquema mostrado na Figura 26.

Figura 26 – Processo de Controle.

Fonte: Autoria Própria.

1. O sinal de posição vem do codificador óptico presente em cada motor.
2. O sinal é capturado pelo Arduino *Mega*.

3. O sinal está dentro do microcontrolador e é passado para o computador para gerar gráficos com a informação de posição.

4. O microcontrolador analisa o sinal recebido e produz um novo sinal de saída de controle.

5. O sinal de controle é passado para o estágio de potência para ser amplificado e enviado para os atuadores do sistema.

3.6 IMPLEMENTAÇÃO DO CONTROLADOR

Fazer o controle de um robô é basicamente regular o fluxo de corrente no motor conforme a necessidade do sistema, ou seja torná-lo estável para seguir a referência com um erro aceitável, entendendo como aceitável um erro que seja o menor possível. No controle em malha fechada, informações sobre como a saída de controle está evoluindo são utilizadas para determinar o sinal de controle que

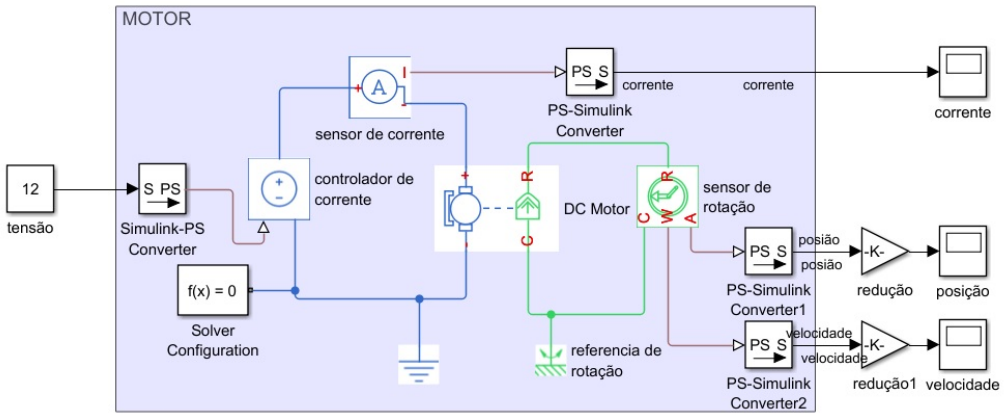
Antes de dar início ao controle do dispositivo, é importante ter os parâmetros do motor DC, bem como a modelagem do mesmo para fazer a validação do motor DC. Na Tabela 9 segue os dados do motor DC.

Tabela 9 – Informações motor DC

GEAR MOTOR: GM37-545S-50-21D	
Tensão	DC 12V
Velocidade sem carga	110 RPM 0.3A
Máxima eficiência	0.55 N.M, 93 RPM
Máxima potencia	1.55 N.M, 60 RPM
Stall corrente	7A
Resolução	550 PPR
Redução	1:50
Resistência de armadura $[\Omega]$	1.5
Indutância de armadura [H]	0.006
Constante de torque [N.m/A]	0.0022
Inércia do motor $[kg.m^2]$	$2e-7$
Atrito viscoso do motor $[kg.m^2/s]$	$9e-8$

A modelagem do motor pode ser vista na Figura 27.

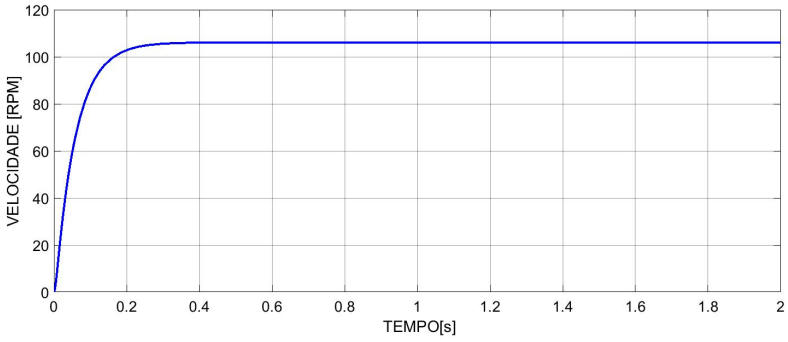
Figura 27 – Modelagem motor DC.



Fonte: Autoria Própria.

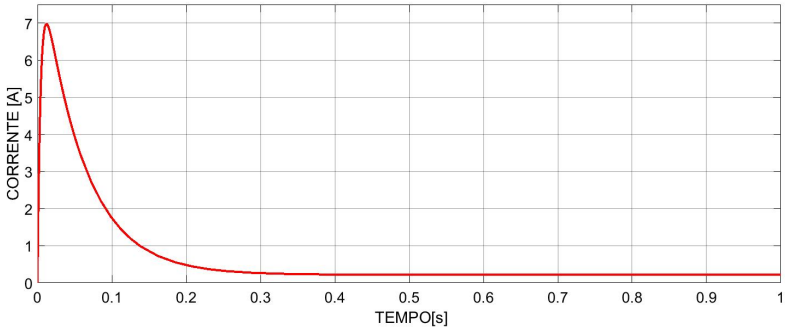
De posse dos parâmetros e da modelagem, a Figura 28 nos mostra que a velocidade de simulação é muito próxima a velocidade informada pelo fabricante, e a Figura 29 nos informa que a corrente é bem próxima. Sendo assim pode-se dar prosseguimento com o controle do sistema, uma vez que os parâmetros corretos são de extrema importância.

Figura 28 – Simulação velocidade motor DC.



Fonte: Autoria Própria.

Figura 29 – Simulação corrente motor DC.



Fonte: Autoria Própria.

O próximo passo a seguir é a implementação de um compensador levando em consideração a dinâmica da junta.

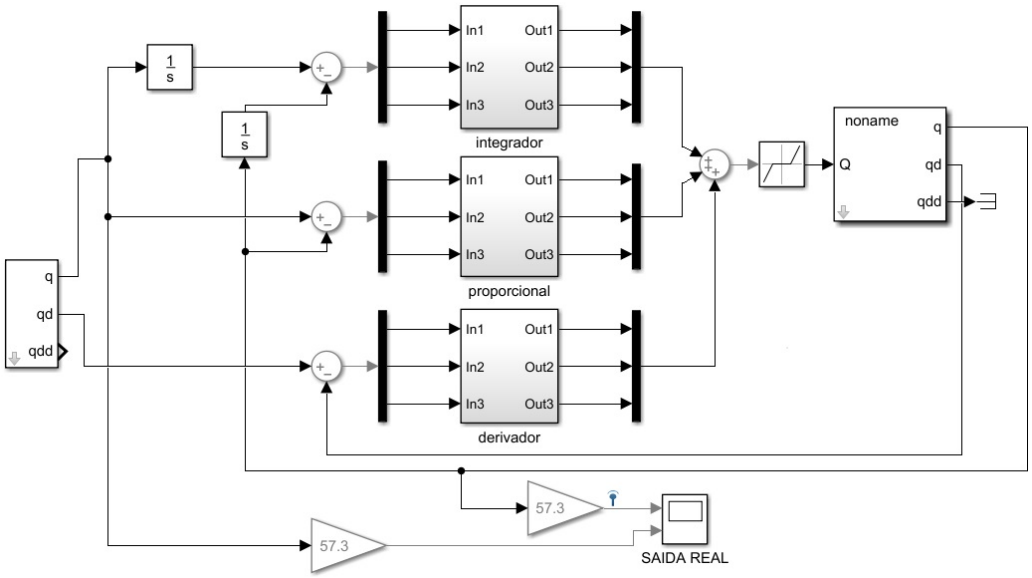
Cada junta será controlada por um sistema SISO(single input- single output). Essa abordagem foi escolhida pois consegue um bom desempenho e tem um menor custo computacional. Por outro lado um erro em uma junta do robô vai prejudicar a estratégia de controle, especialmente um erro na posição da ponta do último elo.

Para os ajustes dos parâmetros do controlador foi usado o segundo método de Ziegler e Nichols, pois o sistema da dinâmica da junta apresenta uma dinâmica oscilatória.

O primeiro passo é definir $K_i = 0$ e $K_d = 0$. Utilizando somente a ação de controle proporcional, aumentar de 0 ao valor critico , no qual a saída exibe uma oscilação sustentada pela primeira vez. Portanto, o ganho crítico e o correspondente período são determinados experimentalmente.

O método foi implementado em conjunto com a toolbox. Foi criado um robô virtual, e os passos foram seguidos inserindo os valores no diagrama de blocos do *MatLab Simulink*. Vale ressaltar que a trajetória escolhida para projetar o controle foi um degrau de 90 graus. A seguir na Figura 30 é apresentado o diagrama de blocos de simulação em malha fechada.

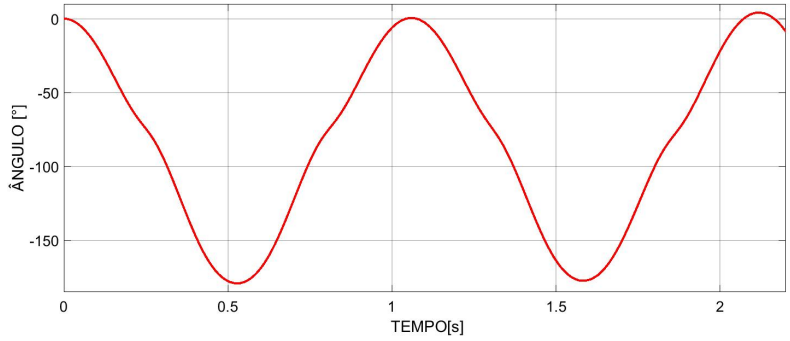
Figura 30 – Diagrama de blocos para ajuste de ganhos do controlador.



Fonte: Autoria Própria.

Como visto na Figura 31 durante a simulação do sistema foi possível encontrar um período crítico de 1.15s para um ganho crítico de 10. De posse desse parâmetros a sintonia do controlador é ajustada de acordo com a Tabela 10.

Figura 31 – Oscilação do sistema para ganho crítico de 10.



Fonte: Autoria Própria.

Tabela 10 – Regra de sintonia de Ziegler e Nichols.

Controlador	K_p	K_i	K_d
P	0.5 K_cr		
PI	0.45 K_cr	1.2/P_cr	
PID	0.6 K_cr	2/P_cr	0.125 P_cr

De acordo com a tabela, os valores do controlador para serem ajustados são: $K_p = 6; K_i = 1.05; K_d = 0.15$.

A nova resposta dinâmica da junta 2 com o PID pode ser vista na linha azul da Figura 32.

Nota-se que a sintonia do controlador PID obtém um desempenho satisfatório, pois o sistema atingiu a referência desejada com pouco sobressinal e com pouca oscilação. No entanto para um robô é importante melhorar a resposta do sistema antes da implementação real.

Aumentando o valor da constante K_p deixou o sistema um pouco mais rápido, mas ocasionou um sobressinal. Dependendo da aplicação o sobressinal é aceitável, mas nesse caso um sobressinal iria prejudicar a vida útil do motor DC, sendo assim deve-se modificar os ganhos K_i e K_d para melhorar ainda mais o sistema.

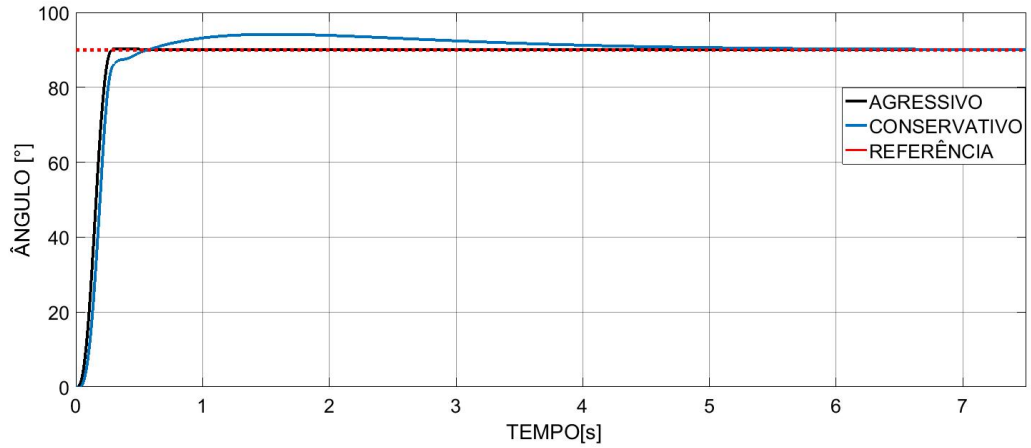
Com o aumento de K_i, o erro em regime permanente foi anulado de forma mais rápida, e com o aumento da constante derivativa diminuí a oscilação durante o tempo de subida.

Fazendo uma sintonia mais refinada do sistema tem-se como resultado os ganhos $K_p = 50; K_i = 60; K_d = 2$. Com ganhos altos o sistema vai chegar na referência de forma rápida, mas surge um problema que é a brusca aceleração do sistema e a necessidade de um freio-motor para que a perturbação ocasionada pelo acoplamento das juntas não aumente o erro. Para solucionar esse problema e assim evitar problemas no motor quando o erro de posição em cada junta for maior que x graus os ganhos serão conservativos($K_p = 6; K_i = 1.05; K_d = 0.15$), e quando o erro for pequeno os ganhos serão agressivos($K_p = 50; K_i = 60; K_d = 2$).

É pertinente salientar que pode ocorrer uma mudança nos parâmetros de acordo com a tarefa que o manipulador deverá executar.

Logo abaixo na Figura 32 é notado a importância e a eficácia dessa abordagem na sintonia do controlador.

Figura 32 – Simulação controle.



Fonte: Autoria Própria.

Nessa simulação foi analisado apenas a junta 2 que é a que vai forçar mais o motor. As juntas 1 e 2 foram utilizadas os mesmo ganhos do controlador.

4 RESULTADOS EXPERIMENTAIS

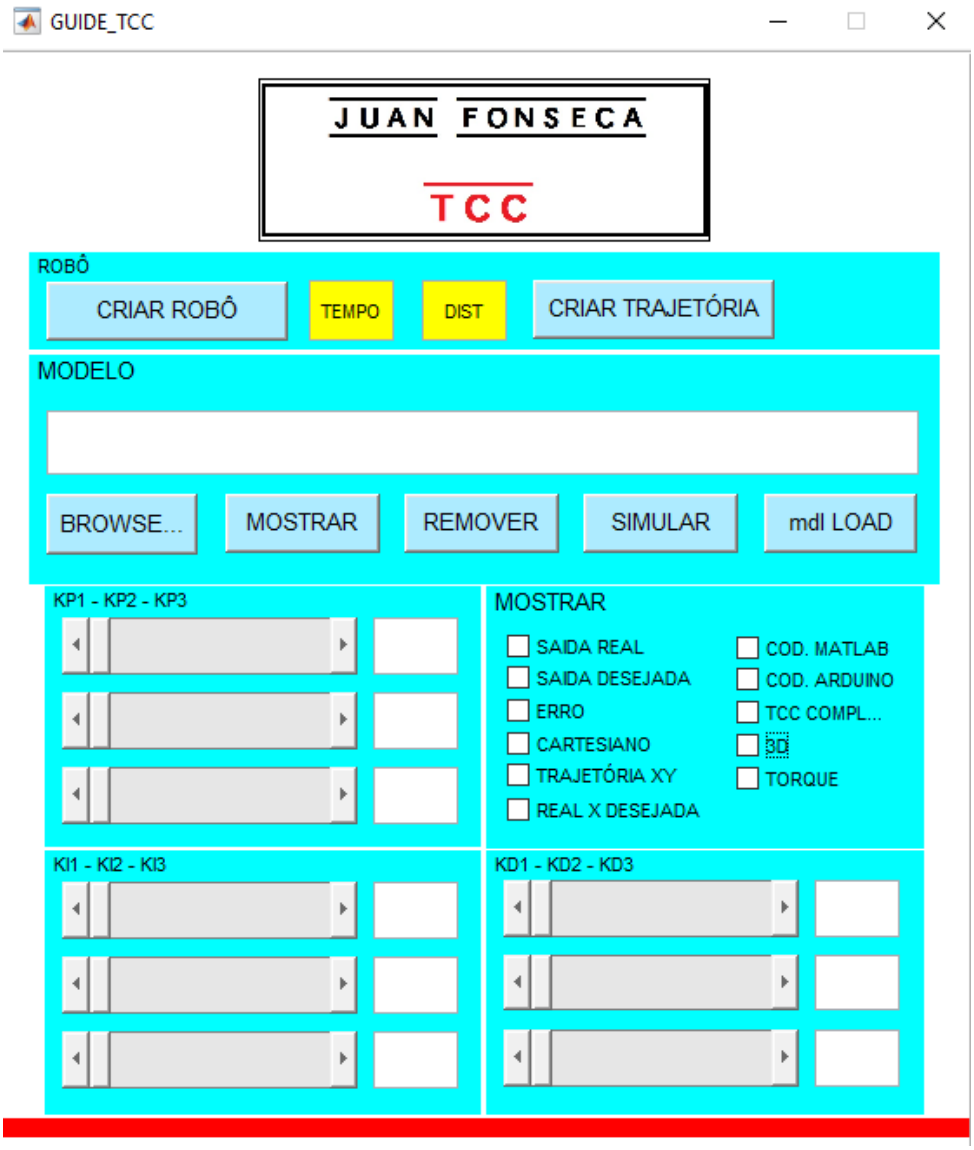
Neste capítulo apresentam-se os resultados experimentais sobre a simulação, construção do protótipo, a parte elétrica e controle de posição do robô.

4.1 RESULTADOS DA SIMULAÇÃO

Este seção apresenta os resultados relativos à interface gráfica da *GUIDE*. Os detalhes relativos programação e ao controle de manipuladores robóticos estão no anexos C e D. Os resultados da simulação serão apresentados em forma de figuras e comentários.

A interface final da guide de acordo as funcionalidades é ilustrada na Figura 33.

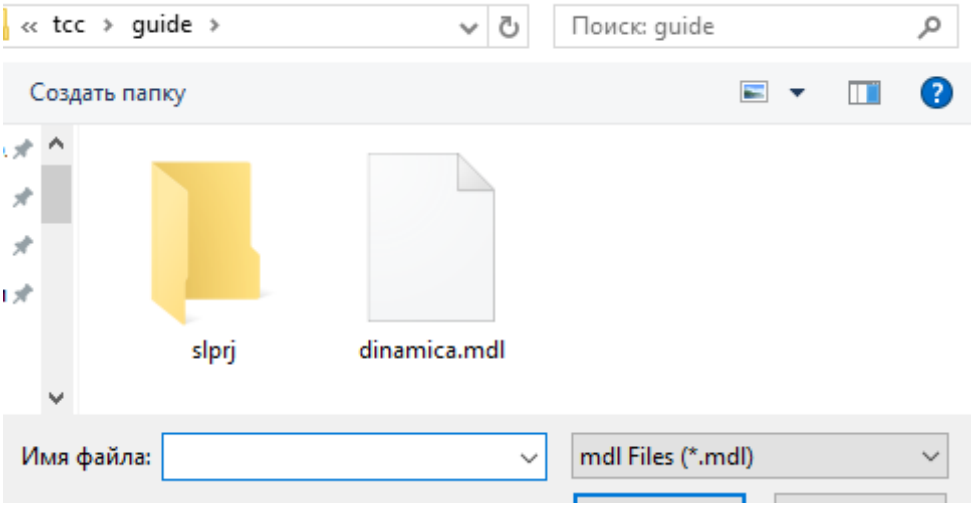
Figura 33 – Interface GUIDE.



Fonte: Autoria Própria.

Ao clicar no botão *BROWSER* abrirá uma janela para pesquisar endereço do modelo à ser simulado.

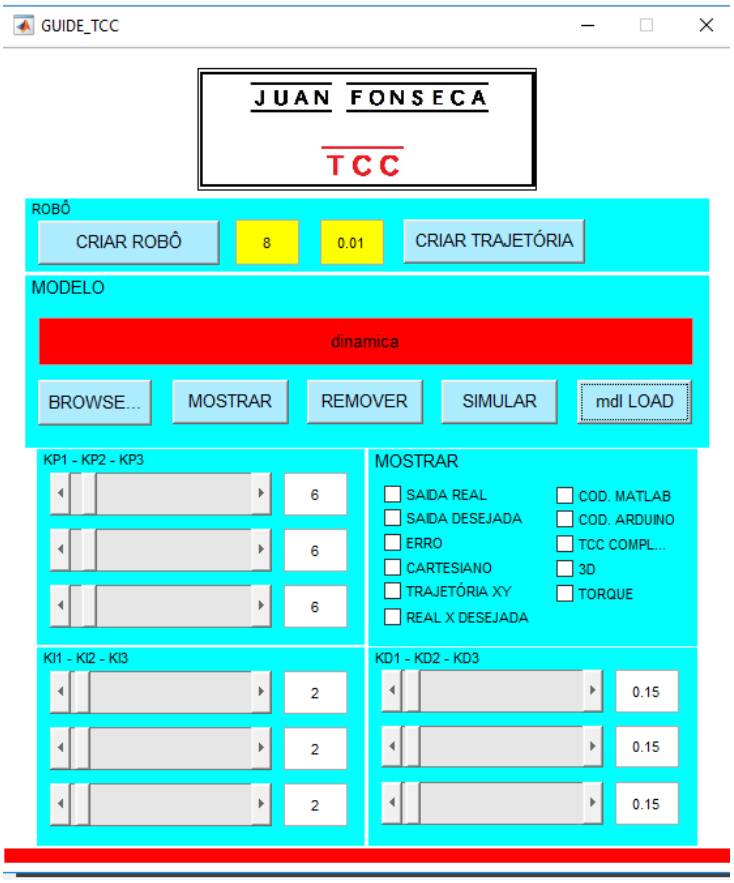
Figura 34 – Funcionalidade do botão para pesquisar endereço do modelo.



Fonte: Autoria Própria.

Quando o usuário clicar no botão *mdl LOAD* os dados dos ganhos PID serão informados ao usuário. A Figura 35 ilustra isso.

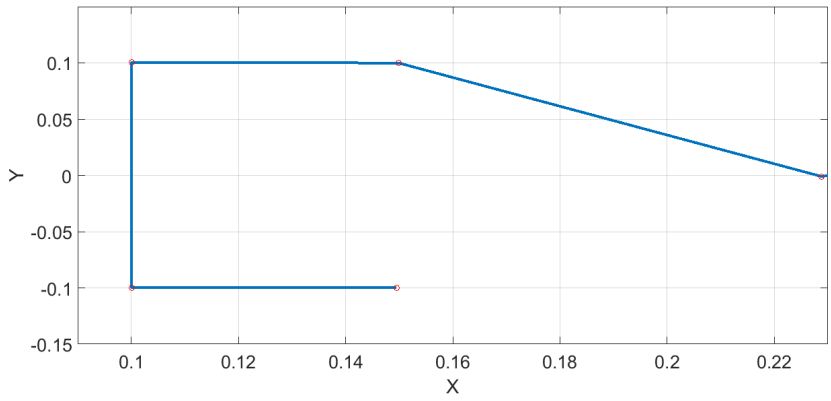
Figura 35 – Funcionalidade do botão para carregar os dados do modelo.



Fonte: Autoria Própria.

A Figura 36 tem a informação da trajetória criada pelo usuário, essa trajetória sera simulada no *MatLab SImulink* quando o usuário clicar no botão *SIMULAR*.

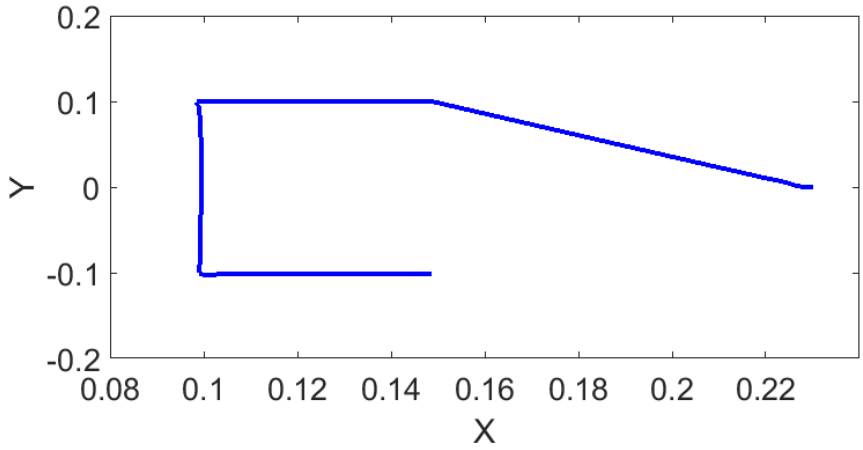
Figura 36 – Trajetória criada pelo usuário.



Fonte: Autoria Própria.

A Figura 37 ilustra a trajetória do efetuador final do manipulador. Essa informação é importante para verificar a semelhança entre trajetória criada pelo usuário e trajetória real do sistema.

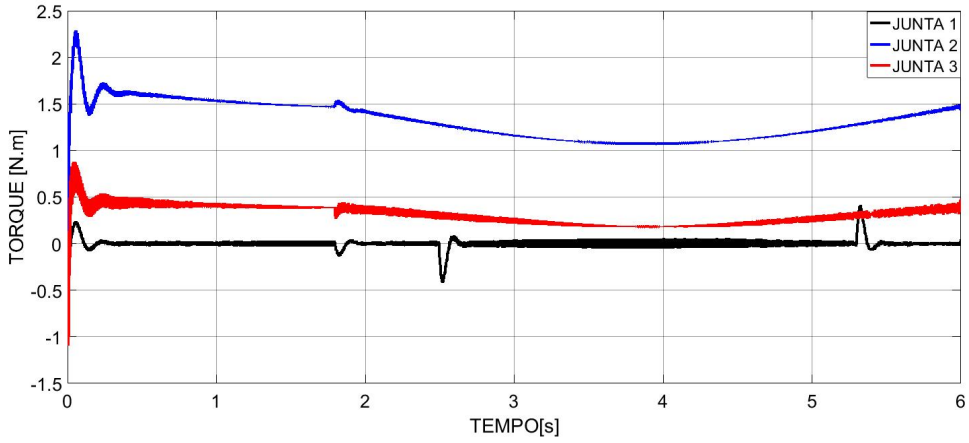
Figura 37 – Trajetória real simulada.



Fonte: Autoria Própria.

Dentre os vários dados que a *GUIDE* facilita a visualização uma importante é o torque do motor na sua trajetória completa, a Figura 38 ilustra essa funcionalidade.

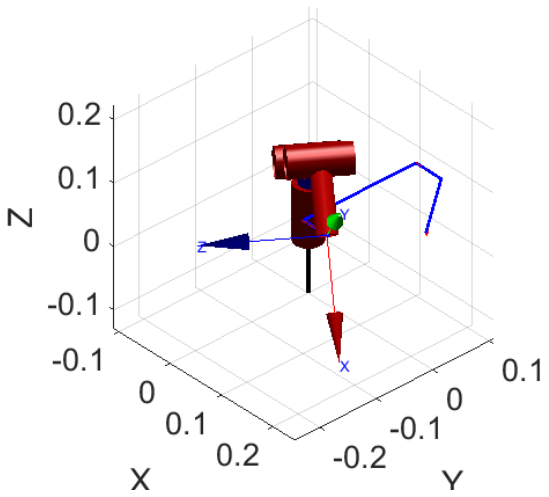
Figura 38 – Torque simulado no motor.



Fonte: Autoria Própria.

A Figura 39 mostra a trajetória 3d do motor. Em linha azul podemos ver a posição final do manipulador desejada e em vermelho a trajetória real.

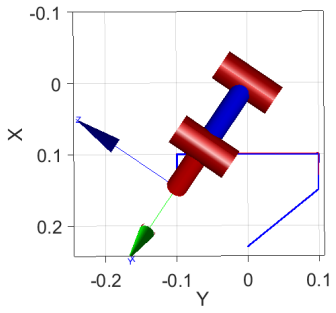
Figura 39 – Trajetória 3D do robô na vista isométrica.



Fonte: Autoria Própria.

A Figura 40 mostra a mesma trajetória da figura anterior, mas de uma perspectiva diferente. Essa imagem nos mostra uma funcionalidade importante na projeção em 3D, que é uma maior proximidade da realidade.

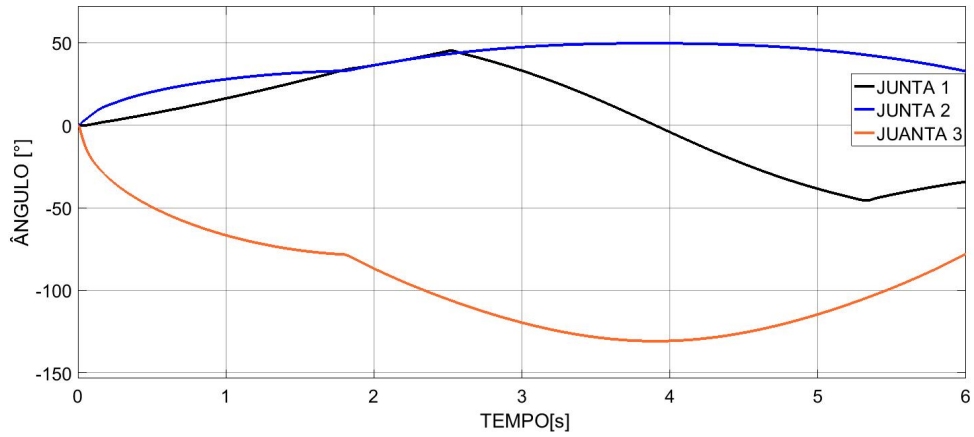
Figura 40 – Trajetória 3D do robô na vista superior.



Fonte: Autoria Própria.

A Figura 41 ilustra a posição angular das 3 juntas no tempo. O sinal preto é referente posição real da junta 1. O sinal azul é ferente posição real da junta 2. O sinal vermelho escuro é ferente posição real da junta 3.

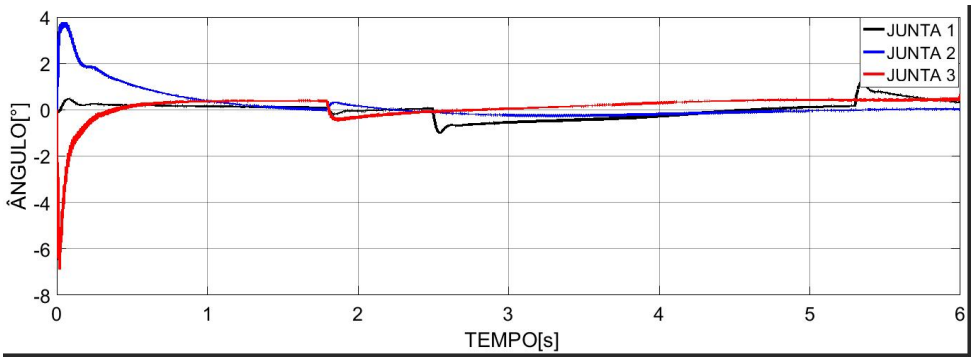
Figura 41 – Posição angular das juntas.



Fonte: Autoria Própria.

A seguir na Figura 56 é ilustrado o erro, ou seja a diferença entre trajetória de referência e trajetória real.

Figura 42 – Erro da trajetória simulada das juntas.



Fonte: Autoria Própria.

4.2 RESULTADOS DA CONSTRUÇÃO DO PROTÓTIPO

A seguir serão apresentados aspectos relativos instalação atuadores, assim como as dimensão das peças que compõem todo o mecanismo do robô. O sistema no quadro elétrico e suas ligações serão apresentados, sendo esses de extrema importância para uma maior organização da ligação dos componentes que atuam no controle e posição. E por fim será apresentada uma visão geral do sistema completo montado.

4.2.1 Parte Mecânica

O processo de impressão em 3D funciona de modo que o plástico sai do cartucho e entra pela máquina até chegar a uma extremidade onde é aquecido a 130 graus celcius.

Apesar de ser o modo mais econômico e de fácil acesso, a impressão em 3D não mostrou ter uma alta qualidade.

A seguir algumas considerações sobre o resultado da impressão das peças quando a resistência, acabamento superficial, custo e velocidade:

- Resistência: as peças impressas em 3D mostraram que não são resistentes como as peças produzidas de forma profissional, por exemplo usinando alumínio. A técnica camada-a-camada que por um lado é uma vantagem também foi a maior fraqueza. A moldagem por usinagem, a peça é bem resistente, já que o material possui uma estrutura relativamente consistente e homogênea. Na impressão 3D, as camadas, mas elas não "grudam" bem. Quando exposto a esforços de compressão e tração a peça parece forte, mas quando exposto a esforços de cisalhamento, flexão e torção as peças se desmontaram facilmente.

- Acabamento da superfície: O resultado esperado da impressão em plástico era de algo brilhante e liso. No entanto o acabamento fosco, cheio de linhas irregulares por todas as camadas e algumas descontinuidades foram algo presentes na maioria das peças. Uma forma de resolver esse problema era pós-processar as peças, mas isso iria envolver trabalho e produtos químicos como acetona, e removedor de detalhes.

- Custos: o custo varia de acordo com o material usado, então coisas grandes são caras, e coisas pequenas são baratas. É isso. Não tem nada a ver com complexidade, e nada a ver com o número de peças. Também não há economia de escala. Assim, a produção de peças de reposição era um desperdício de tempo.

- Velocidade: Para processos de produção, a impressão leva horas, até dias. Uma solução para isso era admitir mais grossas, mas com isso iria aumentar a deterioração da qualidade do acabamento da superfície. A limitação das propriedades químicas de materiais envolvidos também foi um problema. Há uma taxa máxima para aplicá-los no objeto.

  A seguir as imagens da peças que foram impressas em 3D.

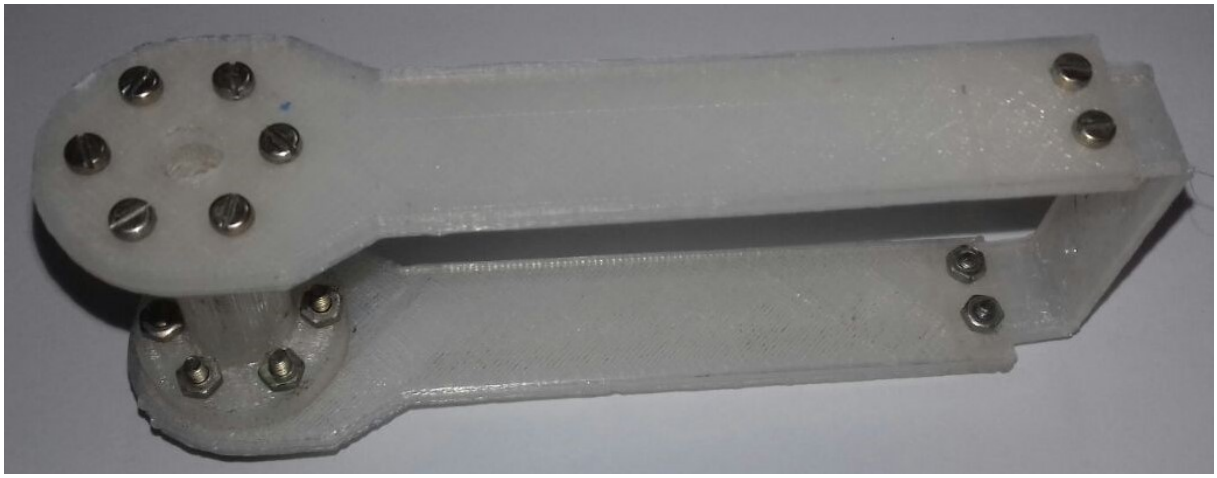
Figura 43 – Impressão em 3D do apoio do elo 2.

Fonte: Autoria Própria.

Figura 44 – Impressão em 3D do elo 1.

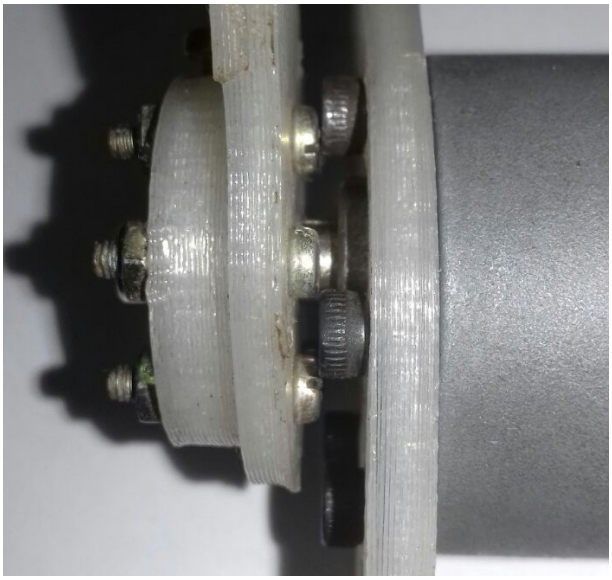
Fonte: Autoria Própria.

Figura 45 – Impressão em 3D do elo 3.



Fonte: Autoria Própria.

Figura 46 – Detalhe de espaço no acoplamento de elos.



Fonte: Autoria Própria.

Figura 47 – Impressão em 3D dos elos 2 e 3.

Fonte: Autoria Própria.

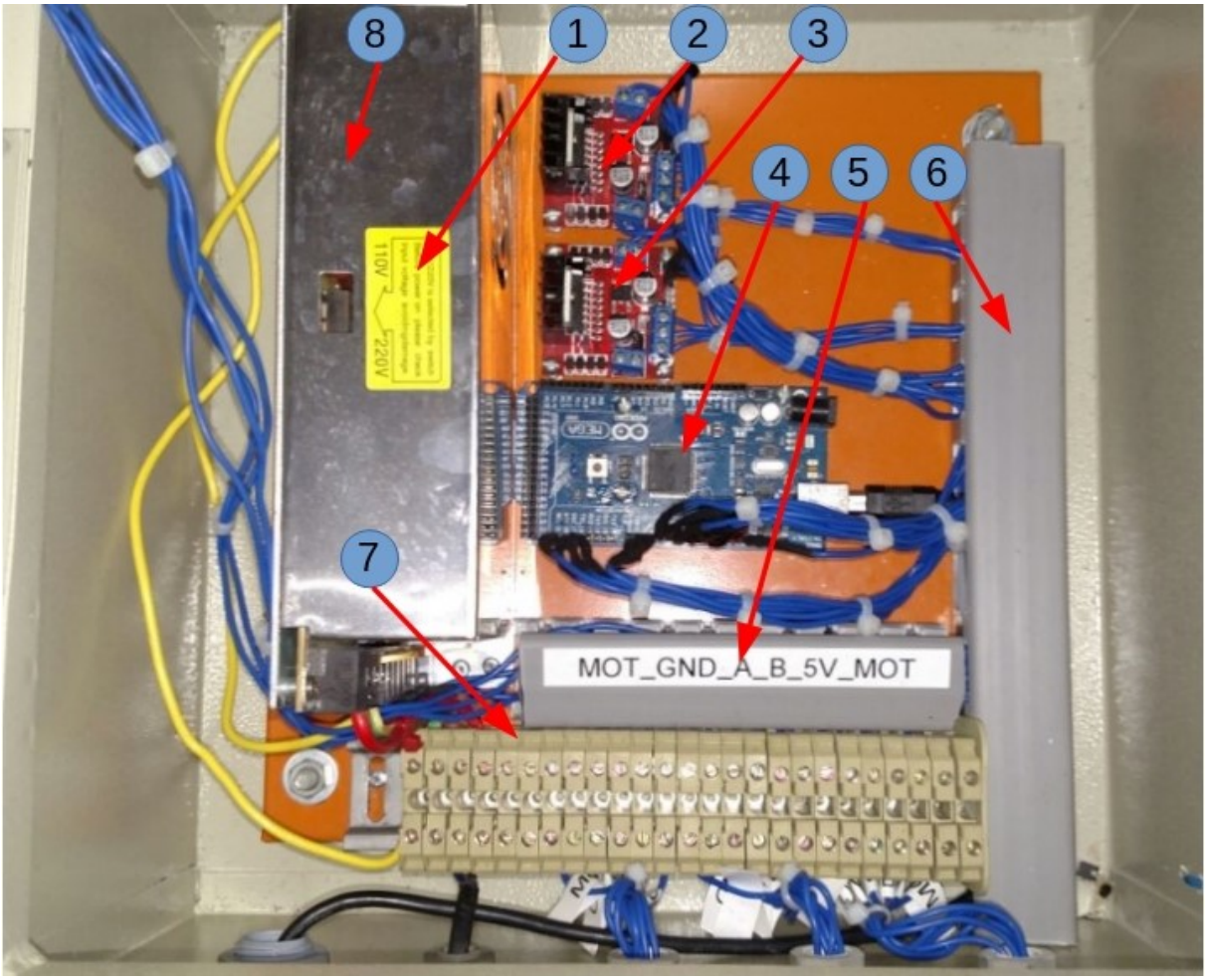
Figura 48 – Impressão em 3D do pinhão do motor.

Fonte: Autoria Própria.

4.2.2 Parte Elétrica

Para acomodação do sistema eletrônico foi utilizado um quadro elétrico, sendo esse um recurso extremamente importante, pois tem a função de proteger os dispositivos que vão receber os sinais dos atuadores e computador, bem como proteger e distribuir sinais de comando em vários circuitos elétricos individuais ou comuns(vide anexo) para alimentação dos atuadores do robô. A seguir na Figura 11 temos a enumeração e logo apos a descrição da ligação.

Figura 49 – Ligação eletrônica no quadro elétrico.



Fonte: Autoria Própria.

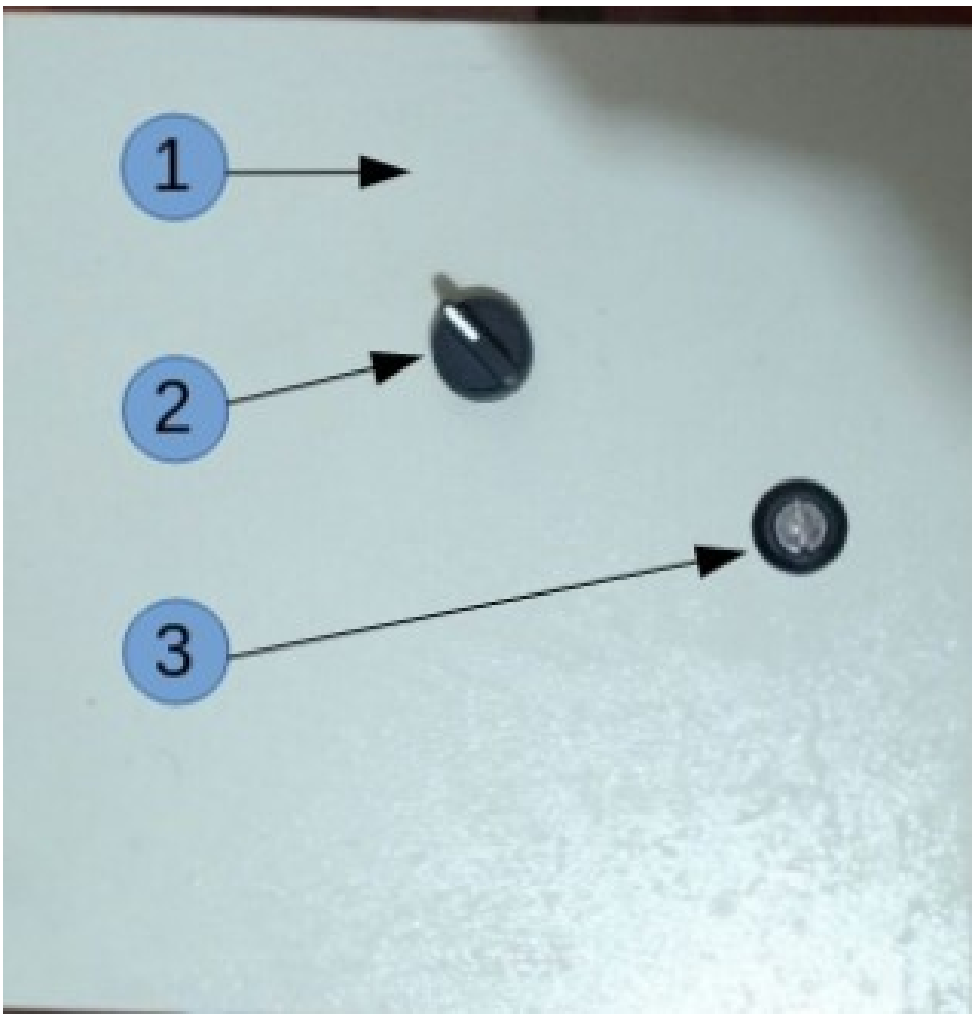
Tabela 11 – Descrição quadro elétrico

Numero	Nome
1	Chave seleção tensão entrada
2	Ponte H com chip L298N
3	Ponte H com chip L298N
4	Arduino Mega
5	Identificação da posição dos fios
6	Canaleta de PVC
7	Born Saque
8	Fonte 12v 20A

Os circuitos elétricos são divididos para uma melhor distribuição das cargas elétricas e garantia da durabilidade de cabos, microcontrolador, fontes e isolação, devendo todos os componentes serem interligados, compatíveis e seletivos.

Como um sistema de segurança manual foi instalada uma chave tipo de duas posições normalmente aberta. A finalidade dessa chave é dar a possibilidade de fazer um corte na energia enviada para o sistema de controle(Arduino) e para o chip L298N. A seguir na Figura 50 uma foto da chave que é posicionada na frente quadro elétrico e a seguir uma breve descrição.

Figura 50 – Parte frontal quadro elétrico.



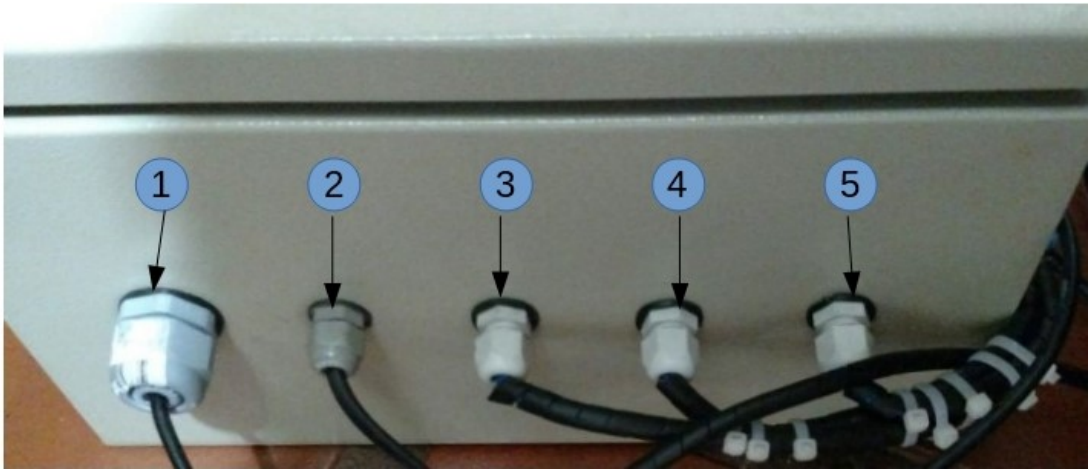
Fonte: Autoria Própria.

Tabela 12 – Descrição parte frontal do quadro elétrico

Número	Nome
1	Quadro elétrico
2	Chave liga-desliga
3	Fechadura da porta

Dando continuidade a descrição da parte externa do quadro elétrico na Figura 51 cinco saídas com prensa, sendo 4 de 1"1/4"e outra de 1", na Tabela 13 temos a descrição da montagem.

Figura 51 – Saídas de cabos do quadro elétrico.



Fonte: Autoria Própria.

Tabela 13 – Descrição das saídas de cabos do quadro elétrico.

Número	Nome
1	Entrada 110V/220V
2	USB
3	Cabo motor 1
4	Cabo motor 2
5	Cabo motor 3

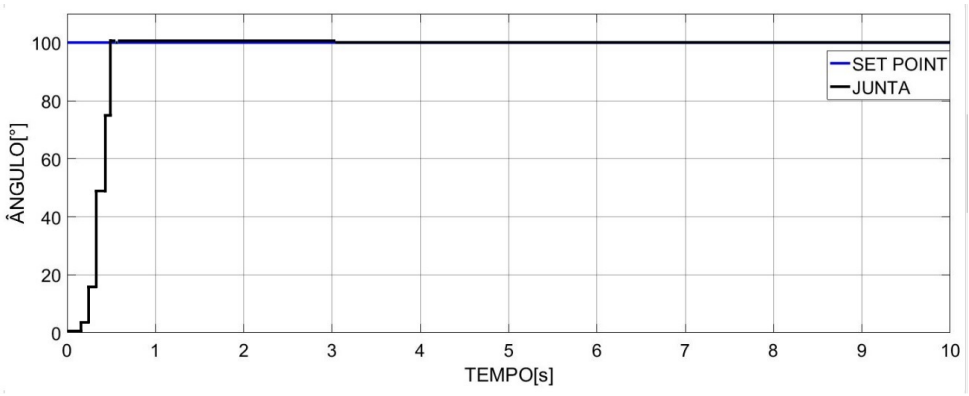
Logo após o detalhamento dos sinais foi feita a montagem da parte elétrica usando somente com materiais capazes de resistir esforços mecânicos, elétricos e térmicos, bem como aos efeitos da umidade, que provavelmente serão encontrados em serviço normal. A proteção contra corrosão foi assegurada pelo uso de materiais apropriados e pela aplicação de camadas protetoras na superfície exposta, levando em conta as condições de uso e manutenção. Os dispositivos e os circuitos foram dispostos de maneira que facilitem a sua operação e manutenção e, ao mesmo tempo, que assegure o grau necessário de segurança.

4.3 RESULTADOS DO CONTROLE

Para o controle de posição dos motores, é importante a integração com os sensores. O papel da simulação é estudar previamente o comportamento do motor e desse modo adequar os ganhos para o motor da melhor forma possível. Com os parâmetros do motor foi possível fazer uma simulação condizente com a realidade e assim aplicar o resultado no experimento. O microcontrolador *Arduino* mostrou-se eficaz no gerenciamento do controle, apesar de não suportar muitos dados de trajetória. Contudo, esse detalhe não vai interferir a trajetória se ela for

pequena. Para um teste inicial no experimento, foi criado um programa que aplica uma entrada desejada como referência para a junta 3 e foram realizadas comparações entre a referência e valor real. Na Figura 52, é aplicado um degrau de 100 graus como referência. O sinal em azul mostra o sinal de referencia, enquanto o sinal preto ilustra o movimento do motor buscando a referência. O tempo necessário para o posicionamento para um degrau de 100 graus graus no redutor, acoplado ao motor, com relação de 50:1 é de 3 segundos. Quando o valor de referência é igual ao valor real os altos ganhos do motor fazem o mesmo funcionar com um freio-motor, evitando assim que o efeito de acoplamento ou perturbações mudem o posicionamento.

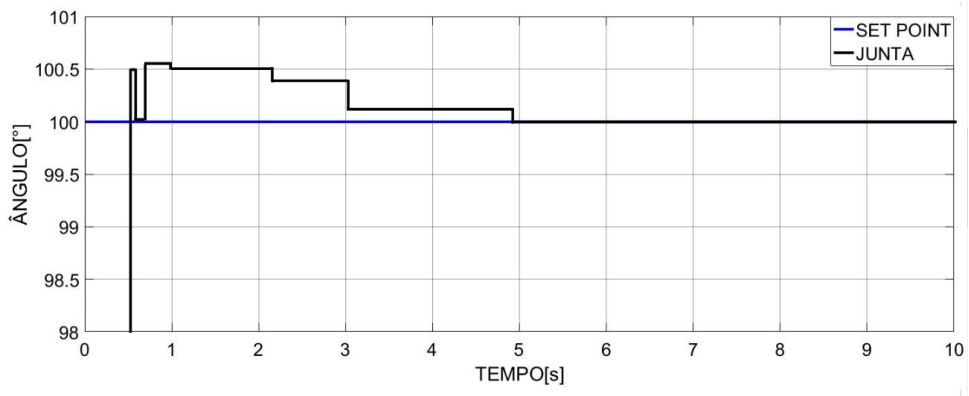
Figura 52 – Resposta para degrau de 100 graus.



Fonte: Autoria Própria.

Na Figura 53 uma ampliação da simulação.

Figura 53 – Resposta ampliada para degrau de 100 graus.



Fonte: Autoria Própria.

Observando a resposta da junta com o controlador PID nota-se que o método foi satisfatório, pois apresentou uma resposta estável, conseguindo atingir o set point desejado em 3

Tabela 14 – Comparação entre simulação e experimento.

Descrição	Simulação	Experimento
Tempo de acomodação [s]	0.28	0.5
Tempo de erro zero [s]	1.15	5
Máximo de ultrapassagem [s]	0.33	0.5
Tempo de pico [s]	0.3	0.6
Tempo de subida [s]	0.28	0.5

segundos e com arranques e freadas suaves, bem como uma resposta estável em regime permanente.

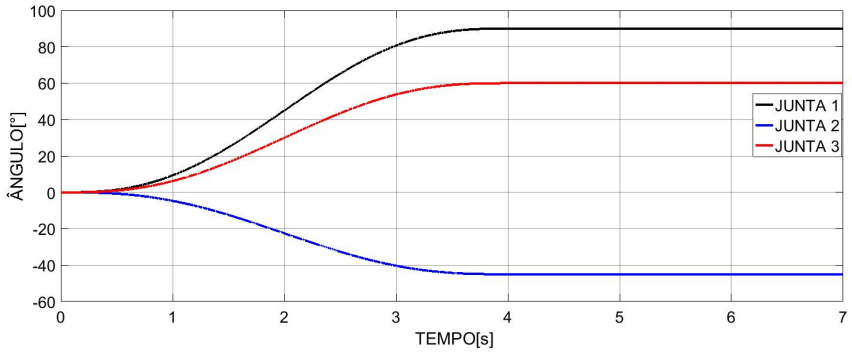
Na Tabela 14 temos a comparação entre o simulado e o experimento.

Comparando-se as respostas do experimento, com as respostas de simulação nota-se uma curva bastante semelhante, exceto na máxima ultrapassagem que na simulação chega a 0.5% e em nenhum dos casos apresentou erro estacionário.

No caso prático, as respostas mais lentas observadas podem ser explicadas pois o modelo de simulação não considera o tempo de processamento do programa e atrasos na transmissão de dados.

A seguir a Figura 55 ilustra uma trajetória de $q_0 = [0\ 0\ 0]$ até $q_f = [\pi/2\ \ -\pi/4\ \pi/3]$ no decorrer de 7 segundos.

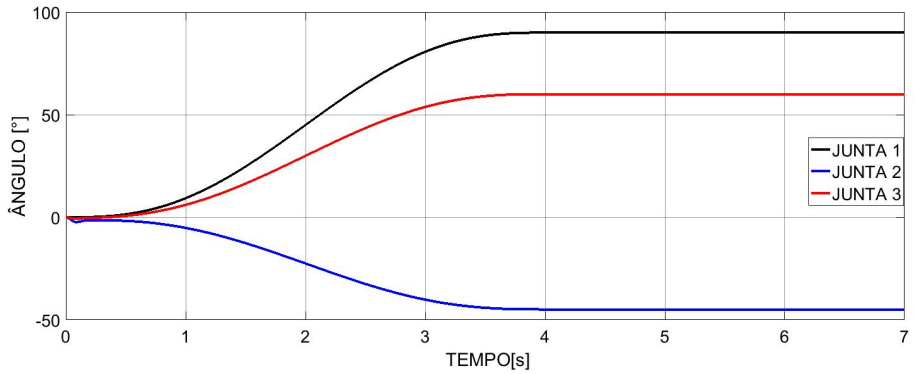
Figura 54 – Trajetória desejada para as 3 juntas.



Fonte: Autoria Própria.

Na Figura 55 nota-se ver o resultado da resposta da junta 1, junta 2 e junta 3 para a trajetória de referência da Figura 54.

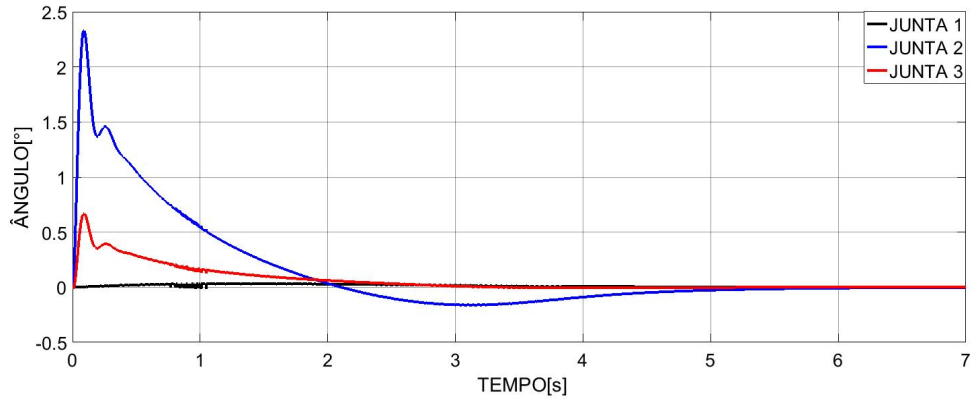
Figura 55 – Trajetória real para as 3 juntas.



Fonte: Autoria Própria.

A seguir na Figura 56 é ilustrado o erro, ou seja a diferença entre trajetória de referência e trajetória real. A junta 1 é aquela que tem o menor erro, a junta 3 tem um pequeno erro no começo de sua trajetória, a junta 3 em virtude dos altos carregamentos que ela deve suportar é aquela que tem o maior erro. Apesar do erro ser pequeno, o gráfico do erro nos mostra que é recomendável a troca do motor 2 por um de maior torque, para diminuir o erro no inicio da trajetória.

Figura 56 – Erro de trajetória para as 3 juntas.



Fonte: Autoria Própria.

Analisando os gráficos acima nota-se claramente a grande semelhança entre a trajetória de referência e trajetória real. Essa semelhança entre as respostas deve-se principalmente a acuracidade do projeto e modelagem do manipulador robótico e também o uso da toolbox *Robotic Toolbox for MatLab* com a correta parametrização para a simulação.

O controlador sintonizado pelo método da sensibilidade limite (Ziegler e Nichols) com e com auxílio da toolbox teve uma resposta adequada, ou seja, sempre que a variável de entrada

do controlador (set point) é modificada a variável de saída do controlador (posição angular do motor) é corrigida visando modificar a posição do efetuador final solicitada no processo.

5 CONSIDERAÇÕES FINAIS

Este projeto de um braço robótico antropomórfico nasceu como uma proposta para fortalecer a linha de pesquisa em robótica e aumentar o conhecimento que permite o desenvolvimento e a melhoria de sistemas que auxiliam na execução de tarefas na indústria. É importante enfatizar que o projeto de sistemas robotizados é um tema de pesquisa de muitos anos, e em que todos os dias designers estão adquirindo experiências que podem melhorar o protótipo.

5.1 OBJETIVOS ABRANGIDOS

Neste trabalho de pesquisa, foram alcançados os seguintes objetivos:

- Foi projetado de um braço robótico antropomórfico com 3 graus de liberdade.
- A simulação de movimento foi realizada tanto para o modelo do *MatLab* quando para o protótipo robótico.
- Foi realizada uma análise estrutural de todo o sistema robótico.
- Foi realizada uma análise dinâmica dos três graus do braço robótico.
- Cinemática direta e inversa foram encontradas para o sistema projetado.

5.2 RESULTADOS DA PARTE MECÂNICA

No que diz respeito ao desenho mecânico do braço robótico, o objetivo de projetar um sistema com 3 graus de liberdade: A seleção de componentes foi complicada, pois suas dimensões não são simples de encontrar nos insumos do mercado interno com as características necessárias(Rolamentos, parafusos, polias, redutores, correias, etc.), por esta razão a seleção fora do mercado interno de algumas das peças de protótipo.

Uma vez que o braço robótico antropomórfico possui 3 graus de liberdade e a distribuição, localização e orientação de seus atuadores, é capaz de flexionar e posicionar-se qualquer posição dentro do seu espaço de trabalho, permitindo uma ampla gama de movimento, limitado apenas pela característica do antropomorfismo, uma vez que o braço robótico projetado é capaz de alcançar posições que um braço humano não pode alcançar por limitações físicas. Algumas especificações e restrições foram consideradas para o projeto, tais como: Dimensões, atuadores, redução, carga máxima, sistema de transmissão e redução, entre outras. O projeto final do braço possui uma massa aproximada de 1.5 kg, com capacidade manuseio de 0,5 kg. No seu efetor

final. O efetor final é uma mão robótica antropomórfica de 3 grau de liberdade, onde 3 atuadores fazem o robô ser colocada e orientada em qualquer posição, à semelhança da mão humana, para pegar e manipular objetos no espaço de trabalho. O motivo para submetê-los a 3 graus de liberdade é porque ele economiza o custo dos motores e reduz o peso total do sistema, sem prejudicar a movimentação em todas as direções.

O filamento de plástico ABS foi selecionado como material de trabalho por causa de seu bom desempenho mecânico e facilidade de compra, mas demostrou ter pouca resistência para aplicações como essa, pois os filamentos não resistiram aos testes.

A impressão em 3D precisa crescer em áreas como mercado de protótipos, sem dúvida essa ferramenta é de grande importância na construção de um protótipo e tem feitos admiráveis. Mas quando levamos em consideração a qualidades das peças, as peças impressas tem um longo caminho e não há vantagem de custo, as pesquisas em geral vão sempre escolher o produto feito em massa.

5.3 RESULTADOS DA PARTE DE SIMULAÇÃO

Ao realizar a análise estrutural estática usando o *SolidWorks* nas partes críticas do ombro, cotovelo e pulso, observa-se que o robô consegue efetuar movimentos em todas as direções. Com o modelo dinâmico do robô, realizado pela *toolbox*, encontrou o torque necessário que os atuadores tiveram que ter em cada um dos elos. Este par nos permitiu selecionar os redutores, pois eles têm alto torque de saída e os sistemas antropomórficos são adequados o uso por causa da dimensão e peso.

Em termos de cinemática envolvida no sistema robótico, a cinemática direita do braço, que serve para posicionar o efetor final, e os parâmetros de rotação das 3 articulações do robô a *toolbox* mostrou capaz de resolver com satisfação esse problema. Com cinemática direta e inversa, verificou-se que é possível posicionar o robô em qualquer lugar no espaço de trabalho do braço robótico.

Com as simulações feitas nesse trabalho com o controle do manipulador, a *toolbox* mostrou que é perfeitamente aplicável no ambiente industrial, desde que os projetistas e pessoas que vão operar diretamente a ferramenta tenha um conhecimento aprofundado da ferramentas e do software Matlab, e que haja tempo suficiente e poder computacional para estas simulações. Um outro fator que pode interferir negativamente é a falta de parâmetros da parte mecânica do robô e dos atuadores envolvidos.

5.4 RESULTADOS DA PARTE ELETRÔNICA

Com o detalhamento dos sinais foi possível uma maior organização na montagem do circuito elétrico. O *Arduino Mega* mostrou-se eficaz na manipulação dos sinais e processamento. O software que foi embarcado foi um problema, pois teve que lidar com a pouca quantidade de memória interna do *Arduino Mega*, algo que deve atrapalhar mais ainda o sistema caso o controlador implementado seja muito complexo. Na fase de potência o sistema teve um bom desempenho, pois não teve sobreaquecimento, apenas apresentando um pequeno aquecimento quando inserido perturbações no motor. Ambos os estágios funcionaram de forma harmoniosa e eficaz. .

5.5 CONTRIBUIÇÕES

As contribuições deste trabalho de pesquisa são as seguintes:

- O desenho de um braço robótico antropomórfico de 3 gdl que aumenta o conhecimento deste tipo de sistemas, na área de mecatrônica na UTFPR.
- Um sistema de potência foi projetado pra uso diversos no controle de motores DC.
- Todo o sistema robótico é deixado em planos, para que as futuras gerações possam realizar a fabricação deste.

5.6 LIMITAÇÕES NO DESENVOLVIMENTO

As Limitações no desenvolvimento deste trabalho foram:

- O orçamento limitado não permitiu a fabricação do sistema robótico.
- Outra limitação foi o tempo, porque o desenvolvimento integral de um braço robótico envolve o conhecimento de diferentes disciplinas e a participação de especialistas em cada um delas. O design sendo um processo iterativo, o tempo foi limitado e com ênfase no projeto até a obtenção de um protótipo funcional.
- Outra limitação é que, porque o braço antropomórfico possui 3 graus de liberdade, apenas dois deles estão em acoplamento direto e a base é uma engrenagem lateral.
- Como na base a transmissão do movimento é por meio de engrenagem, pode haver uma discrepância entre os valores medidos pelo sensor e o valor real da posição da junta. O erro estimado foi de aproximadamente 5 graus.

5.7 SUGESTÕES PARA TRABALHOS FUTUROS

O design de um protótipo é um processo iterativo que busca a melhoria da mesmo. Devido a isso, é importante enfatizar que os resultados obtidos com este primeiro braço robótico é suscetível a melhorias. Não obstante o conhecimento gerado tem sido muito importante e servirá para futuras versões do braço robótico. Os trabalhos futuros relacionados a este trabalho foram classificado em cinco áreas:

Na área de instrumentação:

- Incorporar uma garra com sensores tácteis para obter as informações de força necessárias para que a mão do robô pode segurar e manipular objetos diferentes sem quebrá-los.

Na área mecânica:

- Diminuir o volume do braço robótico completo procurando componentes menores Dimensão.
- Trabalhar com outros tipos de materiais, por exemplo, polímeros ou alumínio, o que diminuirá o peso, além de modificar a fricção entre juntas.

Na área eletrônica:

- Design de um painel de controle para o controle do braço robótico.
- Implementar o sistema embarcado num microcontrolador mais veloz e com maior memória.

Controle:

- Realizar um controle multivariável do robô para verificar velocidades e trajetórias.
- Testar diferentes técnicas de controle.

Informática:

- Desenvolvimento de uma interface de usuário gráfica.
- Implementar uma linguagem de programação e sistemas operacionais para controle em Braço robótico em tempo real.

REFERÊNCIAS

CARRARA, V. Apostila de robótica. **Universidade Braz Cubas, Área de Ciências Exatas Engenharia Mecânica, Engenharia de Controle e Automação**, p. 13–27, 2009.

CHAPMAN, S. J. **Programação em MATLAB para engenheiros**. [S.l.: s.n.], 2003.

CORKE, P. I. Robotics toolbox for matlab. In: IEEE. [S.l.], 2015.

CRAIG, J. J. **Robótica**. 3. ed. [S.l.]: Pearson Education do Brasil, 2012.

DAWSON, F. L. L. D. M. **Robot manipulator control: theory and practice**. [S.l.]: CRC Press, 2003.

FERREIRA, J. C. E. Robótica industrial. **Universidade Federal de Santa Catarina**, p. Capítulo 5, 2005.

GROOVER, M. Automation, production system, and computer integrated manufacturing, 1987. In: **IIE Integrated Systems Conference, Nashville, Tenn**. [S.l.: s.n.], 1987.

LATRE, L. G. Modelagem e controle de posição de robôs. **Revista da SBA: Sociedade Brasileira de Automática**, v. 2, 1988.

OGATA, K.; MAYA, P. Á.; LEONARDI, F. **Engenharia de controle moderno**. [S.l.]: Prentice Hall, 2003.

PIERI, E. R. d. Curso de robótica móvel. **UFSC–Universidade Federal de Santa Catarina**, 2002.

PROYECTOS de robótica. <https://sites.google.com/site/proyectosroboticos/>. Acessado em 18/04/2017.

ROMANO, V. F. **Robótica industrial: aplicação na indústria de manufatufatura e de processos**. [S.l.]: Edgard Blucher, 2002.

ROSÁRIO, J. M. **Princípios de mecatrônica**. [S.l.]: Pearson Prentice Hall, 2006.

SANTOS, V. M. F. **Robótica industrial**. 1. ed. [S.l.]: Editora Universidade de Aveiro, 2004.

SCHILLING, R. J. **Fundamentals of robotics: analysis and control**. [S.l.]: Simon & Schuster Trade, 1996.

SICILIANO, B.; KHATIB, O. **Springer handbook of robotics**. [S.l.]: Springer Science & Business Media, 2008.

SPONG, M. W.; VIDYASAGAR, M. **Robot dynamics and control**. [S.l.]: John Wiley & Sons, 2008.

STONE, W. L. **Robotics and Automation handbook**. [S.l.: s.n.], 2004.

ANEXO A – CINEMÁTICA DO MANIPULADOR ROBÓTICO COM TRÊS GRAUS DE LIBERDADE

O manipulador conta com 3 elos, o elo $1(L_1)$ é fixo e os elos L_2 e L_3 são moveis. Para alcançar uma posição final o efetuador deve posicionar-se de modo que cada junta vai ter um angulo em relação a origem. Sendo: θ_1 o ângulo da junta 1, θ_2 o angulo da junta 2 e θ_3 o angulo da junta 3. As equações que descrevem a cinemática de movimento direto do manipulador são as seguintes:

$$p_x = L_2\cos(\theta_1)\cos(\theta_2) + L_2\cos(\theta_1)\cos(\theta_2+\theta_3)$$
$$p_y = L_2\sin(\theta_1)\cos(\theta_2) + L_2\cos(\theta_1)\cos(\theta_2+\theta_3)$$
$$p_z = L_1 + L_2\sin(\theta_2) + L_3\sin(\theta_2+\theta_3)$$

Obter o modelo cinemático inverso pela forma geométrica consiste em decompor a geometria espacial do manipulador em varias formas de geometria plana.

$$^{i-1}A_i = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

$$\begin{bmatrix} p \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{A.2}$$

Sabendo que:

$$\theta_1 = \arctan^2(\frac{p_y}{p_x}) \tag{A.3}$$

$$p_x = \cos(\theta_1)[L_1 + L_2\cos(\theta_2) + L_3\cos(\theta_2+\theta_3)] \tag{A.4}$$

$$\frac{p_x}{\cos(\theta_1)} - L_1 = L_2\cos(\theta_2) + L_3\cos(\theta_2+\theta_3) \tag{A.5}$$

$$p_y = \sin(\theta_1)[L_1 + L_2\cos(\theta_2) + L_3\cos(\theta_2+\theta_3)] \tag{A.6}$$

$$\frac{p_y}{\sin(\theta_1)} - L_1 = L_2\cos(\theta_2) + L_3\cos(\theta_2+\theta_3) \tag{A.7}$$

Elevando os dois membros ao quadrado temos:

$$L_2\cos(\theta_2) + L_3\cos(\theta_2+\theta_3) = \alpha \tag{A.8}$$

$$L_2{}^2\cos(\theta_2)^2 + L_3{}^2\cos^2(\theta_2+\theta_3) + 2L_1L_2\cos(\theta_2)\cos(\theta_2+\theta_3) = \alpha^2 \tag{A.9}$$

$$L_2\sin(\theta_2) + L_3\sin(\theta_2+\theta_3) = p_z \tag{A.10}$$

$$L_2^2\sin(\theta_2)^2 + L_3^2\sin^2(\theta_2+\theta_3) + 2L_1L_2\sin(\theta_2)\sin(\theta_2+\theta_3) = p_z^2 \tag{A.11}$$

$$L_2^2 + L3^2 + 2L_2L_3\cos(\theta_3) = \alpha^2 + p_z^2 \tag{A.12}$$

Para o ângulo da terceira junta, temos:

$$\cos(\theta_3) = \frac{\alpha^2 - p_z^2 - L2^2 - -L3^2}{2L_2L_3} \tag{A.13}$$

$$\sin(\theta_3) = \sqrt{1-\cos^2(\theta_3)} \tag{A.14}$$

$$\theta_3 = \arctan^2(\frac{\sin\theta_3}{\cos\theta_3}) \tag{A.15}$$

Analogamente para o angulo da segunda junta temos:

$$k_1 = L_1 + L_2\cos(\theta_2) \tag{A.16}$$

$$k_2 = L_2\sin(\theta_2) \tag{A.17}$$

$$\gamma = \arctan^2(\frac{k_2}{k_1}) \tag{A.18}$$

$$\theta_2 = \arctan^2(\frac{\alpha}{p_z}) - \gamma \tag{A.19}$$

ANEXO B – CONCEITOS DA DINÂMICA DO MANIPULADOR ROBÓTICO

Seja a Figura 57 a representação da força centrípeta.

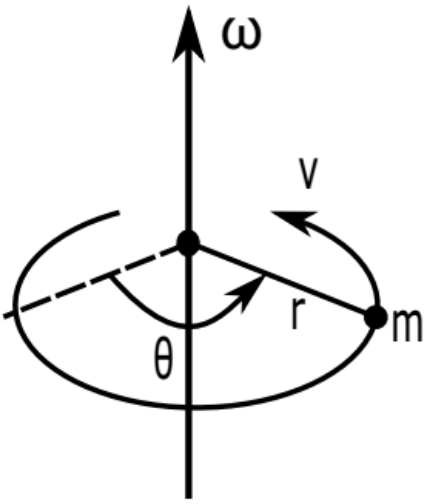


Figura 57 – Força Centrípeta.
Fonte: Adapatada de (DAWSON, 2003).

Onde m é a massa em movimento ao redor de um ponto com raio r, θ é sua posição, v é sua velocidade linear e ω é sua velocidade angular.

A força centrípeta é dada por:

$$F_{cent} = \frac{mv^2}{r} \tag{B.1}$$

A velocidade linear v está apresentada na Equação (B.2).

$$v = \omega r \tag{B.2}$$

Sendo a velocidade angular igual a $\omega = \dot{\theta}$ e considerando a Equação (B.2), a força centrípeta pode ser expressa como:

$$F_{cent} = m\dot{\theta}^2 r \tag{B.3}$$

A força de Coriolis está representada na Figura 58.

A partir da Figura 58, utilizando a regra da mão da direita, verifica-se que a força de Coriolis atua no corpo de massa m desviando-o para a direita com uma velocidade linear v. Então, a força de Coriolis é dada por:

$$F_{cor} = -2m\omega \times v \tag{B.4}$$

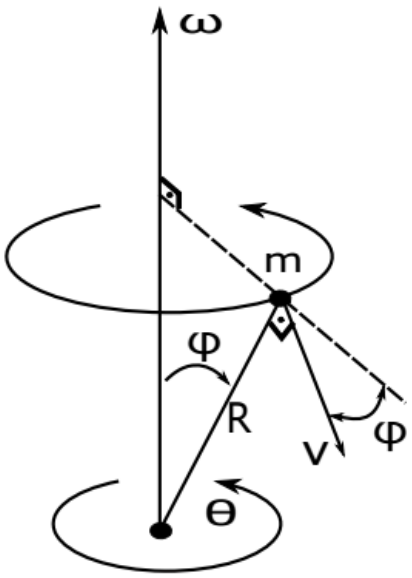


Figura 58 – Força de Coriolis.
Fonte: Adapatada de (DAWSON, 2003).

Como ω é igual a $\dot{\theta}$ e a velocidade linear v é igual a $R\dot{\phi}$, a força Centrípeta pode ser expressa como:

$$F_{cor} = -2mR\dot{\theta}\dot{\phi}cos(\phi) \tag{B.5}$$

Considerando a massa m da Figura 57, sua energia cinética é calculada de acordo com a Equação (B.6).

$$K = \frac{1}{2}mv^2 \tag{B.6}$$

E a energia cinética rotacional é dada por:

$$K_{rot} = \frac{1}{2}I\omega^2 \tag{B.7}$$

Onde I é o momento de inércia.

Considerando m como uma massa pontual, o momento de inércia é expresso como:

$$I = mr^2 \tag{B.8}$$

Então, a Equação (B.7) é reescrita como segue:

$$K_{rot} = \frac{1}{2}mr^2\dot{\theta}^2 \tag{B.9}$$

Também considerando a Figura 57, a energia potencial da massa m é dada como apresenta a Equação (B.10).

$$P = mgh \tag{B.10}$$

Sendo h a altura no campo gravitacional e g a aceleração da gravidade.

A quantidade de movimento de um corpo com massa m e velocidade linear v é calculada de acordo com a Equação (B.11).

$$p = mv \tag{B.11}$$

Já a quantidade de movimento angular é calculado de acordo com a Equação (B.12).

$$P_{ang} = r \times p \tag{B.12}$$

Sendo r a distância da massa m até a origem.

Por fim, o torque de uma força F, considerando a mesma distância r da origem é expresso como indica a Equação (B.13).

$$N = r \times F \tag{B.13}$$

ANEXO C – PROGRAMA ARDUINO TRAJETORIA

```
#include <PID_v1.h>
#include "TimerOne.h"
#include <aJSON.h>

int AnguloRef[3]; // ANGULO DE REFERENCIA
int pulses[3]; // PULSOS ENCODER
int SaidaPWM[3];

// DEFINICAO PINOS ENCODER A B
int encoderBA = 3;
int encoderBB = 2;
int encoderOA = 20;
int encoderOB = 21;
int encoderCA = 18;
int encoderCB = 19;
int juan = 0;
int i =0;
int mattraj[501][3] ={{0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0},
{0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0},{0,0,0},{1,-1,1},
{1,-1,1},{1,-1,1},{1,-1,1},{1,-1,1},{2,-2,2},{2,-2,2},{2,-2,2},
{2,-2,2},{3,-3,3},{3,-3,3},{4,-4,4},{4,-4,4},{5,-5,5},{5,-5,5},
{6,-6,6},{7,-7,7},{7,-7,7},{8,-8,8},{9,-9,9},{9,-9,9},{10,-10,10},
{11,-11,11},{12,-12,12},{13,-13,13},{14,-14,14},{15,-15,15},
{16,-16,16},{17,-17,17},{19,-19,19},{20,-20,20},{21,-21,21},
{23,-23,23},{24,-24,24},{26,-26,26},{27,-27,27},{29,-29,29},
{30,-30,30},{32,-32,32},{34,-34,34},{35,-35,35},{37,-37,37},
{39,-39,39},{41,-41,41},{43,-43,43},{45,-45,45},{47,-47,47},
{49,-49,49},{51,-51,51},{54,-54,54},{56,-56,56},{58,-58,58},
{61,-61,61},{63,-63,63},{65,-65,65},{68,-68,68},{70,-70,70},
{73,-73,73},{76,-76,76},{78,-78,78},{81,-81,81},{84,-84,84},
{87,-87,87},{90,-90,90},{93,-93,93},{96,-96,96},{99,-99,99},{102,-
102,102},{105,-105,105},{108,-108,108},{111,-111,111},{114,-114,114},
{118,-118,118},{121,-121,121},{124,-124,124},{128,-128,128},{131,-
131,131},{135,-135,135},{138,-138,138},{142,-142,142},{145,-145,145},
{149,-149,149},{152,-152,152},{156,-156,156},{160,-160,160},{163,-
163,163},{167,-167,167},{171,-171,171},{175,-175,175},{178,-178,178},
{182,-182,182},{186,-186,186},{190,-190,190},{194,-194,194},{198,-
198,198},{202,-202,202},{206,-206,206},{210,-210,210},{214,-214,214},
{218,-218,218},{222,-222,222},{226,-226,226},{230,-230,230},{234,-
234,234},{238,-238,238},{242,-242,242},{246,-246,246},{250,-250,250},
{254,-254,254},{259,-259,259},{263,-263,263},{267,-267,267},{271,-
271,271},{275,-275,275},{279,-279,279},{283,-283,283},{287,-287,287},
{292,-292,292},{296,-296,296},{300,-300,300},{304,-304,304},{308,-
308,308},{312,-312,312},{316,-316,316},{320,-320,320},{324,-324,324},
```

```
{328,-328,328},{332,-332,332},{336,-336,336},{340,-340,340},{344,-
344,344},{348,-348,348},{352,-352,352},{356,-356,356},{360,-360,360},
{364,-364,364},{368,-368,368},{372,-372,372},{375,-375,375},{379,-
379,379},{383,-383,383},{387,-387,387},{390,-390,390},{394,-394,394},
{398,-398,398},{401,-401,401},{405,-405,405},{408,-408,408},{412,-
412,412},{415,-415,415},{419,-419,419},{422,-422,422},{426,-426,426},
{429,-429,429},{432,-432,432},{436,-436,436},{439,-439,439},{442,-
442,442},{445,-445,445},{448,-448,448},{451,-451,451},{454,-454,454},
{457,-457,457},{460,-460,460},{463,-463,463},{466,-466,466},{469,-
469,469},{472,-472,472},{474,-474,474},{477,-477,477},{480,-480,480},
{482,-482,482},{485,-485,485},{487,-487,487},{490,-490,490},{492,-
492,492},{494,-494,494},{496,-496,496},{499,-499,499},{501,-501,501},
{503,-503,503},{505,-505,505},{507,-507,507},{509,-509,509},{511,-
511,511},{513,-513,513},{515,-515,515},{516,-516,516},{518,-518,518},
{520,-520,520},{521,-521,521},{523,-523,523},{524,-524,524},{526,-
526,526},{527,-527,527},{529,-529,529},{530,-530,530},{531,-531,531},
{533,-533,533},{534,-534,534},{535,-535,535},{536,-536,536},{537,-
537,537},{538,-538,538},{539,-539,539},{540,-540,540},{541,-541,541},
{541,-541,541},{542,-542,542},{543,-543,543},{544,-544,544},{544,-
544,544},{545,-545,545},{545,-545,545},{546,-546,546},{546,-546,546},
{547,-547,547},{547,-547,547},{548,-548,548},{548,-548,548},{548,-
548,548},{548,-548,548},{549,-549,549},{549,-549,549},{549,-549,549},
{549,-549,549},{549,-549,549},{550,-550,550},{550,-550,550},{550,-
550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
```

```
93 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
95 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
97 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
99 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
101 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
103 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
105 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
107 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
109 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
111 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
113 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
115 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
117 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
119 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
121 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
123 {550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-
   550,550},{550,-550,550},{550,-550,550},{550,-550,550},{550,-550,550},
125 {550,-550,550},{550,-550,550},{550,-550,550}};


127
   // definicao pinos PWM
129 #define enB  10
   #define in1B  12
131 #define in2B  11
   #define enO  4
133 #define in1O  6
   #define in2O  5
135 #define enC  7
   #define in1C  8
137 #define in2C  9

139 // ANGUO MAX E MIN
   #define Bmin -4400
```

```
#define Bmax 4400
#define BmaxGraus 4400

#define Omin -4400
#define Omax 4400
#define OmaxGraus 4400

#define Cmin -4400
#define Cmax 4400
#define CmaxGraus 4400

// TEVE PULSO ?
int BpulsesChanged = 0;
int OpulsesChanged = 0;
int CpulsesChanged = 0;
int vmax = 200;
const int sampleRate = 1; // Variable that determines how fast our PID loop runs
double Setpoint[3], Input[3], Output[3]; //These are just variables for storingvalues
double aggKp=10, aggKi=6, aggKd=1;
double consKp=50, consKi=50, consKd=20;
PID myPID0(&Input[0], &Output[0], &Setpoint[0], consKp, consKi, consKd, DIRECT);
PID myPID1(&Input[1], &Output[1], &Setpoint[1], consKp, consKi, consKd, DIRECT);
PID myPID2(&Input[2], &Output[2], &Setpoint[2], consKp, consKi, consKd, DIRECT);

void setup(){
  Serial.begin(57600);

  Timer1.initialize(50000);
  Timer1.attachInterrupt(imprime);
  // PINOS ENCODER COMO ENTRADA
  pinMode(encoderBA, INPUT);
  pinMode(encoderBB, INPUT);
  pinMode(encoderOA, INPUT);
  pinMode(encoderOB, INPUT);
  pinMode(encoderCA, INPUT);
  pinMode(encoderCB, INPUT);
  // INTERRUPCAO ENCODER
  attachInterrupt(0, BA_CHANGE, CHANGE);
  attachInterrupt(1, BB_CHANGE, CHANGE);
  attachInterrupt(2, OA_CHANGE, CHANGE);
  attachInterrupt(3, OB_CHANGE, CHANGE);
  attachInterrupt(4, CA_CHANGE, CHANGE);
  attachInterrupt(5, CB_CHANGE, CHANGE);
```

```
185   myPID0.SetMode(AUTOMATIC); //Turn on the PID loop
      myPID0.SetSampleTime(sampleRate); //Sets the sample rate
187   myPID0.SetOutputLimits(-255, 255);
      myPID1.SetMode(AUTOMATIC); //Turn on the PID loop
189   myPID1.SetSampleTime(sampleRate); //Sets the sample rate
      myPID1.SetOutputLimits(-255, 255);
191   myPID2.SetMode(AUTOMATIC); //Turn on the PID loop
      myPID2.SetSampleTime(sampleRate); //Sets the sample rate
193   myPID2.SetOutputLimits(-255, 255);

195   AnguloRef[0] = 0;
      AnguloRef[1] = 0;
197   AnguloRef[2] = 0;
}//setup
199
void loop(){
201   //ZERAR OS PULSOS CHANGE
      zerapulso();
203   //PID E PWM CADA MOTOR
      Input[0] = pulses[0];
205   Setpoint[0] = AnguloRef[0];
//    double gap0 = abs(Setpoint[0]-Input[0]); //distance away from setpoint
207 //    if(gap0<10)
//    {  //we're close to setpoint, use conservative tuning parameters
209 //      myPID0.SetTunings(consKp, consKi, consKd);
//    }
211 //    else
//    {
213 //       myPID0.SetTunings(aggKp, aggKi, aggKd);
//    }
215   myPID0.Compute();
      pwm0();
217   Input[1] = pulses[1];
      Setpoint[1] = AnguloRef[1];
219 //    double gap1 = abs(Setpoint[1]-Input[1]); //distance away from setpoint
//    if(gap1<10)
221 //    {  //we're close to setpoint, use conservative tuning parameters
//      myPID1.SetTunings(consKp, consKi, consKd);
223 //    }
//    else
225 //    {
//       myPID1.SetTunings(aggKp, aggKi, aggKd);
227 //    }
      myPID1.Compute();
229   pwm1();
      Input[2] = pulses[2];
231   Setpoint[2] = AnguloRef[2];
```

```
//  double gap2 = abs(Setpoint[2]-Input[2]); //distance away from setpoint
//  if(gap2>10)
//  {  //we're close to setpoint, use conservative tuning parameters
//    myPID2.SetTunings(consKp, consKi, consKd);
//  }
//  else
//  {
//     myPID2.SetTunings(aggKp, aggKi, aggKd);
//  }
  myPID2.Compute();
  pwm2();

}

void zerapulso(){
   if (BpulsesChanged != 0) {
    BpulsesChanged = 0;
  }
  if (OpulsesChanged != 0) {
    OpulsesChanged = 0;
  }
  if (CpulsesChanged != 0) {
    CpulsesChanged = 0;
  }
  }
void imprime(){
  Serial.print(pulses[0]);
  Serial.print(",");
  Serial.print(mattraj[i][0]);
  Serial.print(pulses[1]);
  Serial.print(",");
  Serial.print(mattraj[i][1]);
  Serial.print(pulses[2]);
  Serial.print(",");
  Serial.print(mattraj[i][2]);
  Serial.print("\n");
juan++;
if(juan==1){
  juan =0;

  if(i<501){
              AnguloRef[0]=mattraj[i][0];
              AnguloRef[1]=mattraj[i][1];
              AnguloRef[2]=mattraj[i][2];
              i++;
    }
  }
}
```

```
void pwm0(){
281   SaidaPWM[0] = abs(Output[0]);
      SaidaPWM[0] = constrain(SaidaPWM[0], 0, vmax);
283
        analogWrite(enB, SaidaPWM[0]);
285     if (Output[0] < 0 ) {
          digitalWrite(in1B, HIGH);
287       digitalWrite(in2B, LOW);
        }
289     if (Output[0] > 0 ) {
          digitalWrite(in1B, LOW);
291       digitalWrite(in2B, HIGH);
        }
293   }
      void pwm1(){
295   SaidaPWM[1] = abs(Output[1]);
      SaidaPWM[1] = constrain(SaidaPWM[1], 0, vmax);
297
        analogWrite(enO, SaidaPWM[1]);
299     if (Output[1] < 0 ) {
          digitalWrite(in1O, HIGH);
301       digitalWrite(in2O, LOW);
        }
303     if (Output[1] > 0 ) {
          digitalWrite(in1O, LOW);
305       digitalWrite(in2O, HIGH);
        }
307   }
      void pwm2(){
309   SaidaPWM[2] = abs(Output[2]);
      SaidaPWM[2] = constrain(SaidaPWM[2], 0, vmax);
311
        analogWrite(enC, SaidaPWM[2]);
313     if (Output[2] < 0 ) {
          digitalWrite(in1C, HIGH);
315       digitalWrite(in2C, LOW);
        }
317     if (Output[2] > 0 ) {
          digitalWrite(in1C, LOW);
319       digitalWrite(in2C, HIGH);
        }
321   }

323 // ATUALIZACAO DOS PULSOS BASE
    void BA_CHANGE(){
325   if( digitalRead(encoderBB) == 0 ) {
        if ( digitalRead(encoderBA) == 0 ) {
327       // A fell, B is low
```

```
      pulses[0]--; // moving reverse
    } else {
      // A rose, B is low
      pulses[0]++; // moving forward
    }
 } else {
    if ( digitalRead(encoderBA) == 0 ) {
      // A fell, B is high
      pulses[0]++; // moving forward
    } else {
      // A rose, B is high
      pulses[0]--; // moving reverse
    }
  }
  // tell the loop that the pulses have changed
  BpulsesChanged = 1;
}

void BB_CHANGE(){
  if ( digitalRead(encoderBA) == 0 ) {
    if ( digitalRead(encoderBB) == 0 ) {
      // B fell, A is low
      pulses[0]++; // moving forward
    } else {
      // B rose, A is low
      pulses[0]--; // moving reverse
    }
 } else {
    if ( digitalRead(encoderBB) == 0 ) {
      // B fell, A is high
      pulses[0]--; // moving reverse
    } else {
      // B rose, A is high
      pulses[0]++; // moving forward
    }
  }
 // tell the loop that the pulses have changed
  BpulsesChanged = 1;
}

// ATUALIZACAO DOS PULSOS OMBRO
void OA_CHANGE(){
  if( digitalRead(encoderOB) == 0 ) {
    if ( digitalRead(encoderOA) == 0 ) {
      // A fell, B is low
      pulses[1]--; // moving reverse
    } else {
      // A rose, B is low
```

```
      pulses[1]++; // moving forward
    }
  } else {
    if ( digitalRead(encoderOA) == 0 ) {
      // A fell, B is high
      pulses[1]++; // moving forward
    } else {
      // A rose, B is high
      pulses[1]--; // moving reverse
    }
  }
  // tell the loop that the pulses have changed
  OpulsesChanged = 1;
}

void OB_CHANGE(){
  if ( digitalRead(encoderOA) == 0 ) {
    if ( digitalRead(encoderOB) == 0 ) {
      // B fell, A is low
      pulses[1]++; // moving forward
    } else {
      // B rose, A is low
      pulses[1]--; // moving reverse
    }
  } else {
    if ( digitalRead(encoderOB) == 0 ) {
      // B fell, A is high
      pulses[1]--; // moving reverse
    } else {
      // B rose, A is high
      pulses[1]++; // moving forward
    }
  }
  // tell the loop that the pulses have changed
  OpulsesChanged = 1;
}

// ATUALIZACAO DOS PULSOS COTOVELO
void CA_CHANGE(){
  if( digitalRead(encoderCB) == 0 ) {
    if ( digitalRead(encoderCA) == 0 ) {
      // A fell, B is low
      pulses[2]--; // moving reverse
    } else {
      // A rose, B is low
      pulses[2]++; // moving forward
    }
  } else {
```

```
    if ( digitalRead(encoderCA) == 0 ) {
      // A fell, B is high
      pulses[2]++; // moving forward
    } else {
      // A rose, B is high
      pulses[2]--; // moving reverse
    }
  }
  // tell the loop that the pulses have changed
  CpulsesChanged = 1;
}

void CB_CHANGE(){
  if ( digitalRead(encoderCA) == 0 ) {
    if ( digitalRead(encoderCB) == 0 ) {
      // B fell, A is low
      pulses[2]++; // moving forward
    } else {
      // B rose, A is low
      pulses[2]--; // moving reverse
    }
 } else {
    if ( digitalRead(encoderCB) == 0 ) {
      // B fell, A is high
      pulses[2]--; // moving reverse
    } else {
      // B rose, A is high
      pulses[2]++; // moving forward
    }
  }
 // tell the loop that the pulses have changed
  CpulsesChanged = 1;
}
```

ANEXO D – CODIGO MATLAB

MATLAB CODE

```
function varargout = GUIDE_TCC(varargin)
% GUIDE_TCC MATLAB code for GUIDE_TCC.fig
%      GUIDE_TCC, by itself, creates a new GUIDE_TCC or raises the existing
%      singleton*.
%
%      H = GUIDE_TCC returns the handle to a new GUIDE_TCC or the handle to
%      the existing singleton*.
%
%      GUIDE_TCC('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUIDE_TCC.M with the given input
    arguments.
%
%      GUIDE_TCC('Property','Value',...) creates a new GUIDE_TCC or raises
    the
%      existing singleton*.  Starting from the left, property value pairs
    are
%      applied to the GUI before GUIDE_TCC_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
    application
%      stop.  All inputs are passed to GUIDE_TCC_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help GUIDE_TCC

% Last Modified by GUIDE v2.5 09-Sep-2017 19:10:55

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @GUIDE_TCC_OpeningFcn, ...
                   'gui_OutputFcn',  @GUIDE_TCC_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
```

```
38
if nargout
40      [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
42      gui_mainfcn(gui_State, varargin{:});
end
44

46  % End initialization code - DO NOT EDIT

48

    % --- Executes just before GUIDE_TCC is made visible.
50  function GUIDE_TCC_OpeningFcn(hObject, eventdata, handles, varargin)
    % This function has no output args, see OutputFcn.
52  % hObject    handle to figure
    % eventdata  reserved - to be defined in a future version of MATLAB
54  % handles    structure with handles and user data (see GUIDATA)
    % varargin   command line arguments to GUIDE_TCC (see VARARGIN)
56
    % Choose default command line output for GUIDE_TCC
58  handles.output = hObject;

60  % Update handles structure
    guidata(hObject, handles);
62
    myImage = imread('titulo.jpg');
64  axes(handles.axes1);
    imshow(myImage);
66
    % UIWAIT makes GUIDE_TCC wait for user response (see UIRESUME)
68  % uiwait(handles.figure1);


70

    % --- Outputs from this function are returned to the command line.
72  function varargout = GUIDE_TCC_OutputFcn(hObject, eventdata, handles)
    % varargout  cell array for returning output args (see VARARGOUT);
74  % hObject    handle to figure
    % eventdata  reserved - to be defined in a future version of MATLAB
76  % handles    structure with handles and user data (see GUIDATA)

78  % Get default command line output from handles structure
    varargout{1} = handles.output;
80


82  % --- Executes on button press in pushbutton_run.
    function pushbutton_run_Callback(hObject, eventdata, handles)
84  % hObject    handle to pushbutton_run (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
%flag=get(hObject,'string');
%if strcmp(flag,'SIMULAR')==1
set_param(handles.modelname,'SimulationCommand','Start');
%set(hObject,'string','PARAR');
%else
%set_param(handles.modelname,'SimulationCommand','Stop');
%set(hObject,'string','SIMULAR');
%end
guidata(hObject,handles);

function edit_simfile_Callback(hObject, eventdata, handles)
% hObject    handle to edit_simfile (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_simfile as text
%        str2double(get(hObject,'String')) returns contents of edit_simfile
    as a double


% --- Executes during object creation, after setting all properties.
function edit_simfile_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton_simbrowse.
function pushbutton_simbrowse_Callback(hObject, eventdata, handles)
[input_file,pathname] = uigetfile( ...
       {'*mdl','mdl Files (*.mdl)';...
        '*.*', 'All Files (*.*)'}, ...
        'Select files', ...
        'MultiSelect', 'on');
if pathname == 0
    return
end
%gets the current data file names inside the listbox
inputfile= fullfile(pathname,input_file);
current_folder=strcat(cd,'\');
mdlname=strrep(inputfile,current_folder,'');
mdlname=strrep(mdlname,'.mdl','');
%updates the gui to display all filenames in the listbox
set(handles.edit_simfile,'String',mdlname);
guidata(hObject, handles);
```

```
% ---- Executes on button press in pushbutton_simremove.
function pushbutton_simremove_Callback(hObject, eventdata, handles)
modelname=get(handles.edit_simfile,'String');
save_system(modelname);
close_system(modelname);
handles=guidata(hObject);
set(handles.edit_simfile,'String','');
set(handles.edit_simfile,'BackgroundColor', [1 1 1]);
guidata(hObject, handles);


% ---- Executes on button press in pushbutton_loadmodel.
function pushbutton_loadmodel_Callback(hObject, eventdata, handles)
modelname=get(handles.edit_simfile,'string');
set(handles.edit_simfile,'BackgroundColor', [1 0 0]);
if isempty(modelname)
    errordlg('Voce deve carregar o modelo!!!');
end
checkload=~isempty(find_system('type','block_diagram','name',modelname));
if checkload==0
    try
    load_system(modelname);
    catch
    end
end

block_scope_s_real = sprintf('%s/SAIDA REAL',modelname);
block_scope_s_desejada = sprintf('%s/SAIDA DESEJADA',modelname);
block_scope_erro = sprintf('%s/ERRO',modelname);
block_scope_cartesiano = sprintf('%s/CARTESIANO',modelname);
block_scope_trajetoriaxy = sprintf('%s/TRAJETORIA XY',modelname);
block_scope_realdesejada = sprintf('%s/REAL X DESEJADA',modelname);
block_scope_torque = sprintf('%s/TORQUE',modelname);

block_KI1=sprintf('%s/integrador/KI1',modelname);
block_KI2=sprintf('%s/integrador/KI2',modelname);
block_KI3=sprintf('%s/integrador/KI3',modelname);

block_KP1=sprintf('%s/proporcional/KP1',modelname);
block_KP2=sprintf('%s/proporcional/KP2',modelname);
block_KP3=sprintf('%s/proporcional/KP3',modelname);

block_KD1=sprintf('%s/derivador/KD1',modelname);
block_KD2=sprintf('%s/derivador/KD2',modelname);
block_KD3=sprintf('%s/derivador/KD3',modelname);


KP1=get_param(block_KP1,'Gain');
```

```
KP2=get_param(block_KP2,'Gain');
KP3=get_param(block_KP3,'Gain');

KI1=get_param(block_KI1,'Gain');
KI2=get_param(block_KI2,'Gain');
KI3=get_param(block_KI3,'Gain');

KD1=get_param(block_KD1,'Gain');
KD2=get_param(block_KD2,'Gain');
KD3=get_param(block_KD3,'Gain');


set(handles.slider_KP1,'value',str2double(KP1));
set(handles.slider_KP2,'value',str2double(KP2));
set(handles.slider_KP3,'value',str2double(KP3));

set(handles.slider_KI1,'value',str2double(KI1));
set(handles.slider_KI2,'value',str2double(KI2));
set(handles.slider_KI3,'value',str2double(KI3));

set(handles.slider_KD1,'value',str2double(KD1));
set(handles.slider_KD2,'value',str2double(KD2));
set(handles.slider_KD3,'value',str2double(KD3));

set(handles.edit_KP1,'string',num2str(KP1));
set(handles.edit_KP2,'string',num2str(KP2));
set(handles.edit_KP3,'string',num2str(KP3));

set(handles.edit_KI1,'string',num2str(KI1));
set(handles.edit_KI2,'string',num2str(KI2));
set(handles.edit_KI3,'string',num2str(KI3));

set(handles.edit_KD1,'string',num2str(KD1));
set(handles.edit_KD2,'string',num2str(KD2));
set(handles.edit_KD3,'string',num2str(KD3));

handles.modelname=modelname;

handles.block_scope_s_real = block_scope_s_real;
handles.block_scope_s_desejada = block_scope_s_desejada;
handles.block_scope_erro = block_scope_erro;
handles.block_scope_cartesiano = block_scope_cartesiano;
handles.block_scope_trajetoriaxy = block_scope_trajetoriaxy;
handles.block_scope_realdesejada = block_scope_realdesejada;
handles.block_scope_torque = block_scope_torque;

handles.block_KP1=block_KP1;
handles.block_KP2=block_KP2;
```

```
handles.block_KP3=block_KP3;

handles.block_KI1=block_KI1;
handles.block_KI2=block_KI2;
handles.block_KI3=block_KI3;

handles.block_KD1=block_KD1;
handles.block_KD2=block_KD2;
handles.block_KD3=block_KD3;
guidata(hObject,handles)

function edit_KD3_Callback(hObject, eventdata, handles)
handles=guidata(hObject);
val=get(hObject,'string');
set(handles.slider_KD3,'value',str2double(val));
set_param(handles.block_KD3,'Gain',val);
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function edit_KD3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on slider movement.
function slider_KD3_Callback(hObject, eventdata, handles)
val=get(hObject,'value');
set(handles.edit_KD3,'string',num2str(val));
set_param(handles.block_KD3,'Gain',num2str(val));
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function slider_KD3_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

function edit_KD2_Callback(hObject, eventdata, handles)
handles=guidata(hObject);
val=get(hObject,'string');
set(handles.slider_KD2,'value',str2double(val));
set_param(handles.block_KD2,'Gain',val);
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function edit_KD2_CreateFcn(hObject, eventdata, handles)
```

```
274 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
276 end

278 % --- Executes on slider movement.
    function slider_KD2_Callback(hObject, eventdata, handles)
280 val=get(hObject,'value');
    set(handles.edit_KD2,'string',num2str(val));
282 set_param(handles.block_KD2,'Gain',num2str(val));
    guidata(hObject,handles);
284
    % --- Executes during object creation, after setting all properties.
286 function slider_KD2_CreateFcn(hObject, eventdata, handles)
    if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
288     set(hObject,'BackgroundColor',[.9 .9 .9]);
    end
290
    function edit_KD1_Callback(hObject, eventdata, handles)
292 handles=guidata(hObject);
    val=get(hObject,'string');
294 set(handles.slider_KD1,'value',str2double(val));
    set_param(handles.block_KD1,'Gain',val);
296 guidata(hObject,handles);

298 % --- Executes during object creation, after setting all properties.
    function edit_KD1_CreateFcn(hObject, eventdata, handles)
300 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
302 end


304
    % --- Executes on slider movement.
306 function slider_KD1_Callback(hObject, eventdata, handles)
    val=get(hObject,'value');
308 set(handles.edit_KD1,'string',num2str(val));
    set_param(handles.block_KD1,'Gain',num2str(val));
310 guidata(hObject,handles);

312 % --- Executes during object creation, after setting all properties.
    function slider_KD1_CreateFcn(hObject, eventdata, handles)
314 if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor',[.9 .9 .9]);
316 end
```

```
318
320 function edit_KI3_Callback(hObject, eventdata, handles)
    handles=guidata(hObject);
322 val=get(hObject,'string');
    set(handles.slider_KI3,'value',str2double(val));
324 set_param(handles.block_KI3,'Gain',val);
    guidata(hObject,handles);
326
    % --- Executes during object creation, after setting all properties.
328 function edit_KI3_CreateFcn(hObject, eventdata, handles)
    if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
330     set(hObject,'BackgroundColor','white');
    end
332

334 % --- Executes on slider movement.
    function slider_KI3_Callback(hObject, eventdata, handles)
336 val=get(hObject,'value');
    set(handles.edit_KI3,'string',num2str(val));
338 set_param(handles.block_KI3,'Gain',num2str(val));
    guidata(hObject,handles);
340
    % --- Executes during object creation, after setting all properties.
342 function slider_KI3_CreateFcn(hObject, eventdata, handles)
    if isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
344     set(hObject,'BackgroundColor',[.9 .9 .9]);
    end
346
    function edit_KI2_Callback(hObject, eventdata, handles)
348 handles=guidata(hObject);
    val=get(hObject,'string');
350 set(handles.slider_KI2,'value',str2double(val));
    set_param(handles.block_KI2,'Gain',val);
352 guidata(hObject,handles);

354 % --- Executes during object creation, after setting all properties.
    function edit_KI2_CreateFcn(hObject, eventdata, handles)
356 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
358 end


360
    % --- Executes on slider movement.
362 function slider_KI2_Callback(hObject, eventdata, handles)
```

```
val=get(hObject,'value');
set(handles.edit_KI2,'string',num2str(val));
set_param(handles.block_KI2,'Gain',num2str(val));
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function slider_KI2_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



function edit_KI1_Callback(hObject, eventdata, handles)
handles=guidata(hObject);
val=get(hObject,'string');
set(handles.slider_KI1,'value',str2double(val));
set_param(handles.block_KI1,'Gain',val);
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function edit_KI1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on slider movement.
function slider_KI1_Callback(hObject, eventdata, handles)
val=get(hObject,'value');
set(handles.edit_KI1,'string',num2str(val));
set_param(handles.block_KI1,'Gain',num2str(val));
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function slider_KI1_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on slider movement.
function slider_KP1_Callback(hObject, eventdata, handles)
val=get(hObject,'value');
set(handles.edit_KP1,'string',num2str(val));
set_param(handles.block_KP1,'Gain',num2str(val));
```

```
408 guidata(hObject,handles);

410 % --- Executes during object creation, after setting all properties.
    function slider_KP1_CreateFcn(hObject, eventdata, handles)
412 if isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor',[.9 .9 .9]);
414 end

416 function edit_KP1_Callback(hObject, eventdata, handles)
    handles=guidata(hObject);
418 val=get(hObject,'string');
    set(handles.slider_KP1,'value',str2double(val));
420 set_param(handles.block_KP1,'Gain',val);
    guidata(hObject,handles);
422
    % --- Executes during object creation, after setting all properties.
424 function edit_KP1_CreateFcn(hObject, eventdata, handles)
    if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
426     set(hObject,'BackgroundColor','white');
    end
428


430 % --- Executes on slider movement.
    function slider_KP2_Callback(hObject, eventdata, handles)
432 val=get(hObject,'value');
    set(handles.edit_KP2,'string',num2str(val));
434 set_param(handles.block_KP2,'Gain',num2str(val));
    guidata(hObject,handles);
436
    % --- Executes during object creation, after setting all properties.
438 function slider_KP2_CreateFcn(hObject, eventdata, handles)
    if isequal(get(hObject,'BackgroundColor'), get(0,'
        defaultUicontrolBackgroundColor'))
440     set(hObject,'BackgroundColor',[.9 .9 .9]);
    end
442
    function edit_KP2_Callback(hObject, eventdata, handles)
444 handles=guidata(hObject);
    val=get(hObject,'string');
446 set(handles.slider_KP2,'value',str2double(val));
    set_param(handles.block_KP2,'Gain',val);
448 guidata(hObject,handles);

450 % --- Executes during object creation, after setting all properties.
    function edit_KP2_CreateFcn(hObject, eventdata, handles)
```

```
452 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
454 end

456 % --- Executes on slider movement.
    function slider_KP3_Callback(hObject, eventdata, handles)
458 val=get(hObject,'value');
    set(handles.edit_KP3,'string',num2str(val));
460 set_param(handles.block_KP3,'Gain',num2str(val));
    guidata(hObject,handles);
462
    % --- Executes during object creation, after setting all properties.
464 function slider_KP3_CreateFcn(hObject, eventdata, handles)
    if isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
466     set(hObject,'BackgroundColor',[.9 .9 .9]);
    end
468
    function edit_KP3_Callback(hObject, eventdata, handles)
470 handles=guidata(hObject);
    val=get(hObject,'string');
472 set(handles.slider_KP3,'value',str2double(val));
    set_param(handles.block_KP3,'Gain',val);
474 guidata(hObject,handles);

476 % --- Executes during object creation, after setting all properties.
    function edit_KP3_CreateFcn(hObject, eventdata, handles)
478 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
        set(hObject,'BackgroundColor','white');
480 end

482 % --- Executes on button press in checkbox_S_REAL.
    function checkbox_S_REAL_Callback(hObject, eventdata, handles)
484 a = get(hObject, 'Value');
    if (a == 1)
486     open_system(handles.block_scope_s_real);
    else
488     close_system(handles.block_scope_s_real);
    end
490
    % --- Executes on button press in checkbox_S_DESEJADA.
492 function checkbox_S_DESEJADA_Callback(hObject, eventdata, handles)
    a = get(hObject, 'Value');
494 if (a == 1)
        open_system(handles.block_scope_s_desejada);
496 else
```

```
    close_system(handles.block_scope_s_desejada);
end

% --- Executes on button press in checkbox_ERRO.
function checkbox_ERRO_Callback(hObject, eventdata, handles)
a = get(hObject, 'Value');
if (a == 1)
    open_system(handles.block_scope_erro);
else
    close_system(handles.block_scope_erro);
end

% --- Executes on button press in pushbutton_mostra.
function pushbutton_mostra_Callback(hObject, eventdata, handles)
modelname=get(handles.edit_simfile,'String');
open_system(modelname);

% --- Executes on button press in checkbox_T_CART.
function checkbox_T_CART_Callback(hObject, eventdata, handles)
a = get(hObject, 'Value');
if (a == 1)
    open_system(handles.block_scope_cartesiano);
else
    close_system(handles.block_scope_cartesiano);
end

% --- Executes on button press in checkbox_TRAJXY.
function checkbox_TRAJXY_Callback(hObject, eventdata, handles)
a = get(hObject, 'Value');
if (a == 1)
    open_system(handles.block_scope_trajetoriaxy);
else
    close_system(handles.block_scope_trajetoriaxy);
end

% --- Executes on button press in checkbox_T_RD.
function checkbox_T_RD_Callback(hObject, eventdata, handles)
a = get(hObject, 'Value');
if (a == 1)
    open_system(handles.block_scope_realdesejada);
else
    close_system(handles.block_scope_realdesejada);
end
% Hint: get(hObject,'Value') returns toggle state of checkbox_T_RD

% --- Executes on button press in checkbox15.
function checkbox15_Callback(hObject, eventdata, handles)

```

```
% --- Executes on button press in checkbox16.
function checkbox16_Callback(hObject, eventdata, handles)

% --- Executes on button press in checkbox17.
function checkbox17_Callback(hObject, eventdata, handles)

% --- Executes on button press in pushbuttontraj.
function pushbuttontraj_Callback(hObject, eventdata, handles)

set(handles.edit29,'BackgroundColor', [1 1 1]);
global tempo_simulacao;
tempo_simulacao = get(handles.edit29,'String');
tempo_simulacao = str2double(tempo_simulacao);
assignin ('base','tempo_simulacao',tempo_simulacao);
%%
button = 1;
 %eixos
figure(2)
axis([0 0.23 -0.23 0.23])
grid on
hold on

global rx;
global ry;
global rz;

rx = []; %vetor para receber coordenadas de x
ry = []; %vetor para receber coordenadas de y
rz = []; %vetor para receber coordenadas de x

%vai fazer a trajetoria come?ar da posi?? inicial
rx(1) = 0.23;
ry(1) = 0;
rz(1) = 0.08;
i = 2;

while(button ==1) % enquantos clicamos com botao esquerdo do mause
rz(i) = 0.07;
[x,y,button] = ginput(1); %obtemos a coordenada de um ponto da tela
plot3(x,y,rz,'ro'); %mostramos o ponto
%terminamos de ler com um clique no botao direito
rx(i) = x;
ry(i) = y;
i = i+1;
end

x = rx;
y = ry;
```

```
dz = rz;

plot(x,y);

%correcao da altura em z
theta = [0 0 0];
d = [0.08 0 0];
l = [0 0.13 0.1];
alpha = [-pi/2 0 0];
t=[0:1:1]';

z = dz - d(1);

%pontos em x espaco de trabalho
NP = length(x); %numero de pontos
DIST = get(handles.edit30,'String');
DIST = str2double(DIST); %distancia entre cada ponto

%TRAJETORIA
for i=1:1:(NP-1)
    distancia(i) = sqrt((x(i)-x(i+1))^2+(y(i)-y(i+1))^2+(z(i)-z(i+1))^2); %
    distancia entre cada ponto informado
    NPS(i) = distancia(i)/DIST;  %numero de pontos-1 por cada
    NPS(i) = round(NPS(i));
    xd(i) = (x(i+1)-x(i))/(NPS(i)); % distancia entre pontos no eixo x
    yd(i) = (y(i+1)-y(i))/(NPS(i)); % distancia entre pontos no eixo y
    zd(i) = (z(i+1)-z(i))/(NPS(i)); % distancia entre pontos no eixo z


    linhasyez=length(yd);
    pos{1,i}=[]; %cria vetor da trajetoria em x
    pos{2,i}=[]; %cria vetor da trajetoria em y
    pos{3,i}=[]; %cria vetor da trajetoria em z
  for j=1:1:NPS(i)
        pos{1,i}=horzcat(pos{1,i},((x(i)-xd(i))+xd(1,linhasyez)*j));
        pos{2,i}=horzcat(pos{2,i},((y(i)-yd(i))+yd(1,linhasyez)*j));
        pos{3,i}=horzcat(pos{3,i},((z(i)-zd(i))+zd(1,linhasyez)*j));
    end

%     pos{1,i}(1,NPS(i))=x(i+1); % ultimo valor de cada trajeto recebe o
    valor valor final desejado
%     pos{2,i}(1,length(pos{1,i}))=y(i+1); % ultimo valor de cada trajeto
    recebe o valor valor final desejado
%     pos{3,i}(1,length(pos{1,i}))=z(i+1); % ultimo valor de cada trajeto
    recebe o valor valor final desejado
end

CP = [];
```

```
trajx = [];
trajy = [];
trajz = [];

%separa variaveis de trajetoria
for i=1:1:(NP-1)
trajx = horzcat(trajx,pos{1,i});
trajy = horzcat(trajy,pos{2,i});
trajz = horzcat(trajz,pos{3,i});
end
NPN = length(trajx);

rad = pi/180;
graus = 180/pi;

%cinematica
    for i=1:1:NPN
    th0(i) = atan2(trajy(i),trajx(i));
    ix(i) = (trajx(i)^2 + trajy(i)^2)^0.5;
%     #stuff for calculating th2
    r_2(i) = ix(i)^2 + (-trajz(i))^2;
    l_sq = l(2)^2 + l(3)^2;
    term2 = (r_2(i) - l_sq)/(2*l(2)*l(3));
    term1 = ((1 - term2^2)^0.5)*-1;
%     #calculate th2
    th2(i) = atan2(term1, term2);
%     #optional line. Comment this one out if you
%     #notice any problems
    th2(i) = -1*th2(i);
%    x =x y=-z z=y

%     #Stuff for calculating th2
    k1 = l(2) + l(3)*cos(th2(i));
    k2 = l(3)*sin(th2(i));
    r  = (k1^2 + k2^2)^0.5;
    gamma = atan2(k2,k1);
%     #calculate th1
    th1(i) = atan2(-trajz(i),ix(i)) - gamma;
    end

for i=1:1:NPN
    P{i} = [th0(i) th1(i) th2(i)]
end

CP = [];
for i=1:1:(NPN)
CP=vertcat(CP,P{i});
end
```

```
686 for i=1:1:(NPN-1)
        [q{i},qd{i},qdd{i}]=jtraj(P{i},P{i+1}, t);
688 end

690 CQ = [];
    for i=1:1:(NPN-1)
692 CQ = vertcat(CQ,q{i});
    end
694
    CQQ = [];
696 for i=1:1:(NPN-1)
    CQQ = vertcat(CQQ,qd{i});
698 end

700 CQQQ = [];
    for i=1:1:(NPN-1)
702 CQQQ = vertcat(CQQQ,qdd{i});
    end
704
    % figure(2)
706 % %unimos cada par de pontos
    % robot.plot([0,0,0]);
708 % hold on
    % plot3(rx,ry,rz);
710
    %vetor tempo
712 npt = length(CQ); % numero de pontos da trajetoria
    perio = tempo_simulacao/npt; % periodo de amostragem
714 i = 0:perio:(tempo_simulacao-perio);
    assignin ('base','perio',perio);
716 t = i;

718 %adequa matriz para simulink com tempo
    t = t';
720 CQ = horzcat(t,CQ);
    CQQ = horzcat(t,CQQ);
722 CQQQ = horzcat(t,CQQQ);

724 %mandar pra works
    assignin('base','CQ',CQ);
726 assignin('base','CQQ',CQQ);
    assignin('base','CQQQ',CQQQ);
728 %%

730 function edit27_Callback(hObject, eventdata, handles)

732 % --- Executes during object creation, after setting all properties.
```

```
function edit27_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit_tempo_Callback(hObject, eventdata, handles)

% --- Executes on button press in pushbuttonrobo.
function pushbuttonrobo_Callback(hObject, eventdata, handles)
%%
clear all;
% close all;
clc

%constantes do compensador
conskp = 5;
conski = 2;
conskd = 1;
aggkp = 50;
aggki = 63;
aggkd = 2;

%setar parametros nos blocos PID
set_param('dinamica/integrador/KI1','Gain','2');
set_param('dinamica/integrador/KI2','Gain','2');
set_param('dinamica/integrador/KI3','Gain','2');
set_param('dinamica/proporcional/KP1','Gain','6');
set_param('dinamica/proporcional/KP2','Gain','6');
set_param('dinamica/proporcional/KP3','Gain','6');
set_param('dinamica/derivador/KD1','Gain','1');
set_param('dinamica/derivador/KD2','Gain','1');
set_param('dinamica/derivador/KD3','Gain','1');

%parametros motor
%parametros motor
motor_R = 1.5;
motor_L= 0.006;
motor_K = 22/10000;
motor_J = 2/10000000;
motor_B = 9/100000000;

%parametros DH
theta = [0 0 0];
d = [0.08 0 0];
l = [0 0.13 0.1];
alpha = [-pi/2 0 0];
```

```
%criar elos
% theta | D | l | alpha | sigma | m | rx ry rz | Ixx Iyy Izz Ixy Iyz Ixz |
    Jm | G | B | Tc[0 0]
 L(1)=Link([theta(1) d(1) l(1) alpha(1) 0 0.45 0.00 0.03 0.02 0 0 0 0 0 0 0
     motor_J 1/50 motor_B 0 0]);
 L(2)=Link([theta(2) d(2) l(2) alpha(2) 0 0.5 0.01 -0.03 0.00 0 0 0 0 0 0 0
     motor_J 1/50 motor_B 0 0]);
 L(3)=Link([theta(3) d(3) l(3) alpha(3) 0 0.4 0.01 0.03 0.00 0 0 0 0 0 0 0
    motor_J 1/50 motor_B 0 0]);

 %criar robo
global robot;
robot = SerialLink(L);
assignin('base','robot',robot);
t=[0:1:1]';
%%

function edit29_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit29_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in checkbox18.
function checkbox18_Callback(hObject, eventdata, handles)
a = get(hObject, 'Value');
% robot = evalin('base','robot');
global robot;
global rx;
global ry;
global rz;
xr = evalin('base','xr');
yr = evalin('base','yr');
zr = evalin('base','zr');
qreal = evalin('base','qreal');

if (a == 1)
     figure(2);
     realcart = horzcat(xr,yr,zr);
     for i=1:1:length(qreal)-1
     robot.plot(qreal(i,:));
     hold on;
     plot2(realcart(i:i+1,:),'r');
```

```
    end
    robot.plot(qreal(length(qreal),:));
    hold on
    figure(2);
    plot3(rx,ry,rz,'b');
else
   close(figure(2));
end


function checkbox19_Callback(hObject, eventdata, handles)

a = get(hObject, 'Value');
if (a == 1)
    open_system(handles.block_scope_torque);
else
    close_system(handles.block_scope_torque);
end


function edit30_Callback(hObject, eventdata, handles)

function edit30_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
    defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

anexo/GUIDE_TCC.m

ANEXO E – DESENVOLVIMENTO DA GUIDE

Para um melhor entendimento da interface a explicação será feita apresentando um passo a passo da forma que o usuário deve proceder na utilização do software, sendo uma imagem da interface enumerada e logo a seguir a descrição por partes do software numa tabela. Vale ressaltar que a enumeração das partes na imagem estão de acordo com a tabela, sendo essa enumeração aplicada com a finalidade de obter uma maior harmonia no posicionamento dos números na imagem. O passo a passo do funcionamento também é enumerado, mas esse não está em conformidade com a imagem, exceto na figura 59.

De uma forma simples a interatividade entre homem e interface de geração de trajetória deve seguir os seguintes passos:

6. Usuário deve clicar no botão para criar um robô com 3 graus de liberdade
7. Usuário deve digitar um número inteiro referente ao tempo de simulação da trajetória
8. Usuário deve digitar um número real que vai informar o passo da trajetória
9. Usuário deve clicar no botão e informar os pontos da trajetória, sendo o ultimo passo com o botão direito do mouse

A seguir na figura 59 temos a interface responsável por auxiliar o usuário final na trajetória, e logo a seguir a 15 com a descrição da imagem.

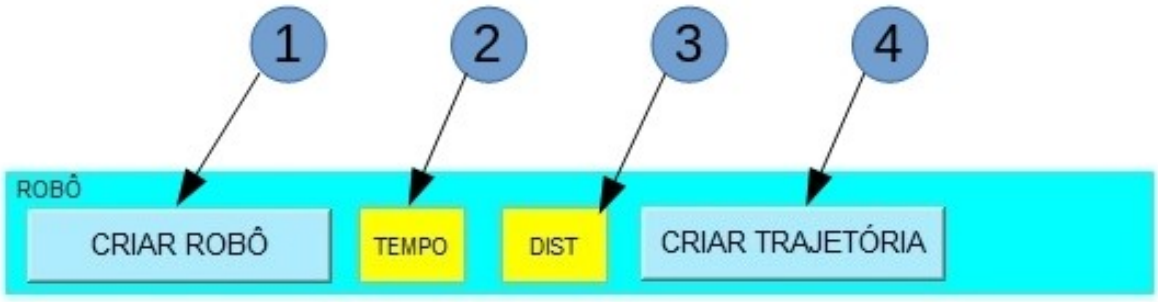


Figura 59 – Trajetória com guide.
Fonte: Autoria própria.

Tabela 15 – Nomenclatura das partes da guide trajetória

Parte	Descrição
1	Botão que cria dados do robô na memória.
2	Tempo de simulação.
3	Distancia de passo em metros da trajetória.
4	Botão que vai pedir os pontos da trajetória.

Já interatividade entre homem e interface de simulação deve seguir os seguintes passos:

10. Usuário deve clicar no botão *BROWSE...* para indicar o endereço do modelo a ser simulado, é indicado que o modelo esteja na mesma pasta do arquivo da *GUIDE*
11. Usuário deve clicar no botão *mdl LOAD* para carregar os dados referentes aos ganhos do controlador.
12. Usuário deve clicar no botão *SIMULAR* para simular o modelo
13. Caso queira ver o modelo o usuário deve clicar no botão *MOSTRAR*
14. Caso queira salvar e fechar o modelo o usuário deve clicar no botão *REMOVER*

A seguir na figura 60 temos a interface responsável por auxiliar o usuário final na simulação, e logo a seguir a 16 com a descrição da imagem.

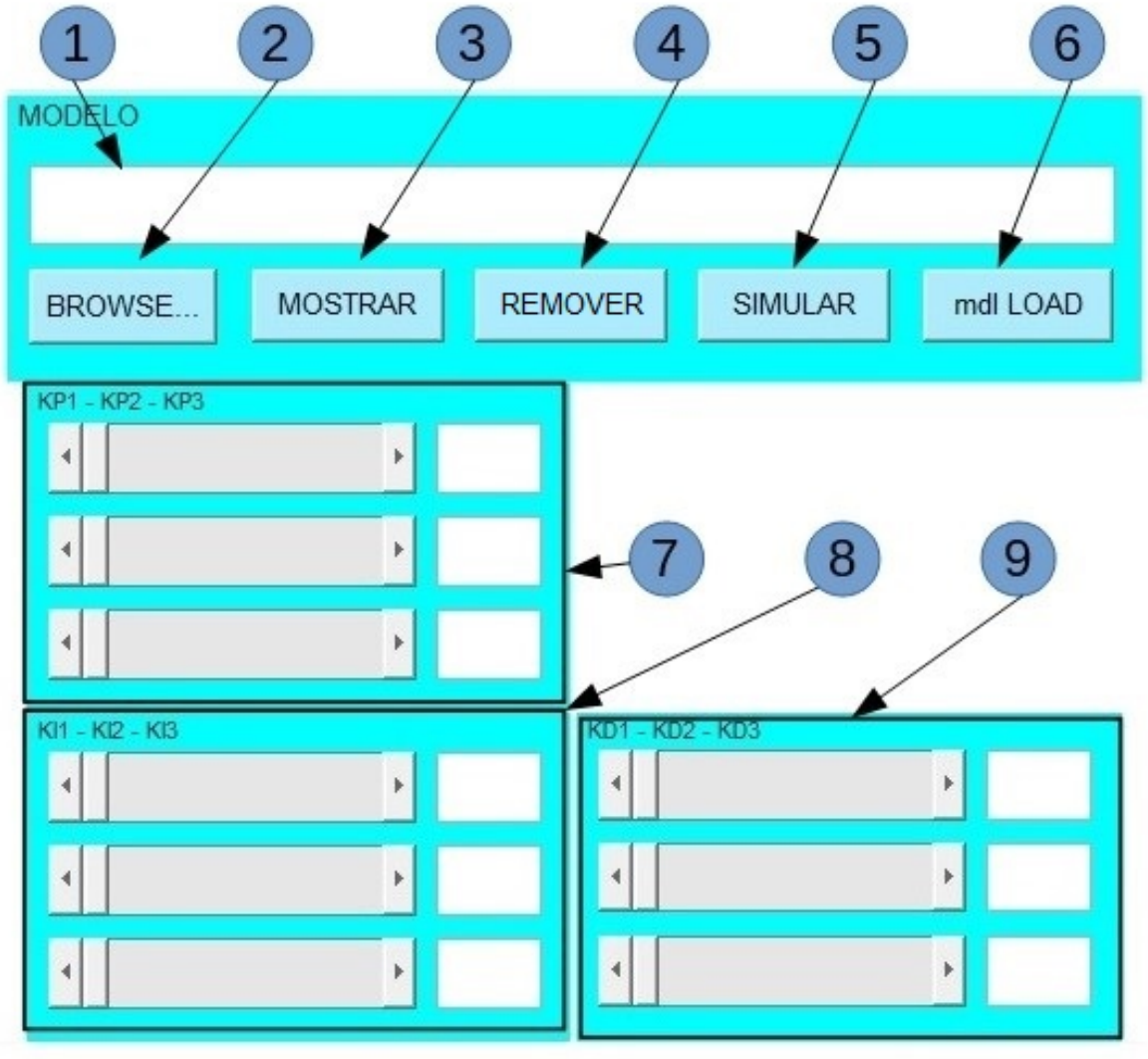


Figura 60 – Simulação com guide.
Fonte: Autoria própria.

Tabela 16 – Nomenclatura das partes da guide simulação

Parte	Descrição
1	Nome do modelo a ser simulado
2	Botão para pesquisar endereço do modelo
3	Botão para mostrar o modelo
4	Botão para fechar o modelo
5	Botão para simular o modelo
6	Botão para carregar dados do modelo
7	Informações sobre ganhos Kp de cada elo
8	Informações sobre ganhos Ki de cada elo
9	Informações sobre ganhos Kd de cada elo

Por fim a interatividade entre homem e interface de exibição de gráficos deve seguir os seguintes passos:

15. Usuário deve marcar a checkbox referente a informação ou gráfico que deseja visualizar

16. Para fechar o gráfico o usuário deve desmarcar a caixa refente ao gráfico que esta em exibição. Esta opção é valida apenas para o *SCOPE* do *simulink*

A seguir na figura 61 temos a interface responsável por auxiliar o usuário final na simulação, e logo a seguir a tabela 17 com a descrição da imagem.

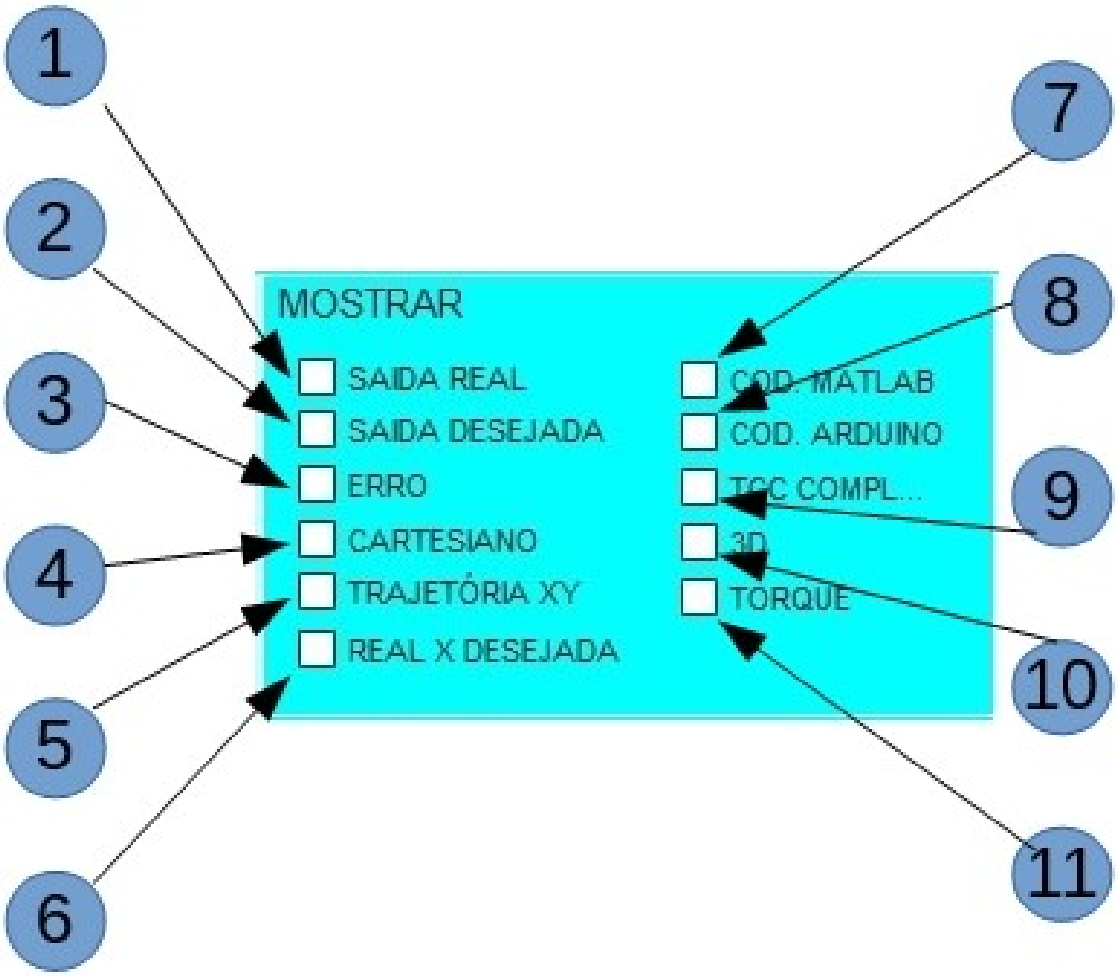


Figura 61 – Gráficos com guide.
Fonte: Autoria própria.

Tabela 17 – Nomenclatura das partes da guide mostrar

Parte	Descrição
1	Checkbox mostrar saída real
2	Checkbox mostrar saída desejada
3	Checkbox mostrar erro de trajetória
4	Checkbox mostrar trajetória no espaço cartesiano
5	Checkbox mostrar trajetória XY
6	Checkbox mostrar Trajetória real x desejada
7	Checkbox mostrar pdf de codigo MATLAB
8	Checkbox mostrar pdf de codigo *Arduino*
9	Checkbox mostrar pdf de TCC completo
10	Checkbox mostrar trajetória 3D
11	Checkbox mostrar torque nas juntas