UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

VINÍCIUS FREITAS RODRIGUES

# SISTEMA LINUX EMBARCADO PARA CÁLCULO E EXECUÇÃO DE TRAJETÓRIAS DE MANIPULADORES ROBÓTICOS

UBERLÂNDIA, MG
2023

Vinícius Freitas Rodrigues

# Sistema Linux Embarcado para Cálculo e Execução de Trajetórias de Manipuladores Robóticos

Projeto de Trabalho de Conclusão de Curso da Engenharia de Controle e Automação da Universidade Federal de Uberlândia - UFU - Campus Santa Mônica, como requisito para a obtenção do título de Graduação em Engenharia de Controle e Automação.

Universidade Federal de Uberlândia - UFU
Faculdade de Engenharia Elétrica - FEELT

Orientador Prof. Éder Alves de Moura

Uberlândia, MG
2023

# Agradecimentos

Agradeço aos meus pais, José Humberto e Luciana, por tudo que fiz até aqui. Cada sonho e oportunidade que tive são reflexo da dedicação deles comigo. Cada projeto que fiz foi graças ao esforço deles. Eles são a prova de que amor e atenção fazem a diferença no futuro de uma pessoa.

Agradeço aos meus amigos que me acompanharam durante essa longa (e bem longa) graduação.

Agradeço, por fim, aos professores da Universidade Federal de Uberlândia pelo ensino e por acreditarem no meu potencial, me oferecendo muitas oportunidades. Em especial, deixo meu agradecimento ao professor Éder Alves de Moura, por me orientar neste e em diversos outros trabalhos.

*" [...]*
*Coração de estudante*
*Há que se cuidar da vida*
*Há que se cuidar do mundo*
*Tomar conta da amizade*
*Alegria e muito sonho*
*Espalhados no caminho*
*Verdes planta e sentimento*
*Folhas, coração, juventude e fé."*
*(Milton Nascimento / Wagner Tiso)*

# Resumo

A modelagem de um manipulador robótico utiliza diversos recursos matemáticos e diferentes algoritmos são aplicados para planejar a trajetória dessas máquinas. Todo esse volume de cálculos para planejamento de trajetórias e o controle de robôs industriais requer o uso de várias ferramentas computacionais. Nesse contexto, cada fabricante de robô tem seus próprios métodos e sistemas que, em grande parte, são protegidos por licenças de uso e modificação bem restritas, devido à questões comerciais. Entretanto, em oposição à esta cultura, o movimento de software livre têm disponibilizado diversas ferramentas de natureza *open-source*, produzidas e mantidas pela comunidade acadêmica, individual ou de grupos, e que permitem a modelagem, simulação e implementação de sistemas robóticos de grande complexidade e desempenho. A *toolbox* para MATLAB criada por Peter Corke, surge no intuito de se modelar robôs com o *know-how* teórico sobre robótica lecionado no ambiente acadêmico. Tal ferramenta permite ampliar a complexidade da modelagem vista nos livros e artigos mais tradicionais. Nesse contexto, este trabalho propõe a integração da *toolbox* para MATLAB a um SBC (*Single Board Computer*) com Linux embarcado para controle de um manipulador robótico. Para tal projeto, portanto, foram desenvolvidos códigos em C++ para o SBC, *scripts* para MATLAB, uma biblioteca para interface com PCA9685 e um hardware dedicado. Como resultado disso, o produto final deste trabalho é um sistema análogo ao de um robô industrial, que permite ao usuário ensinar, calcular, simular e executar trajetórias para um manipulador robótico.

**Palavras-chaves**: Robótica; Planejamento de Trajetória; Linux Embarcado; MATLAB

# Abstract

The modeling of a robotic manipulator uses several mathematical resources and different algorithms are applied to plan the trajectory of these machines. All this volume of calculations for planning trajectories and controlling industrial robots requires the use of several computational tools. In this context, each robot manufacturer has its own methods and systems that, to a large extent, are protected by very restricted use and modification licenses, due to commercial motivations. However, in opposition to this culture, the free software movement has made available several open-source tools, produced and maintained by the academic community, individual or groups, and which allow the modeling, simulation and implementation of highly complex robotic systems. A toolbox for MATLAB, created by Peter Corke, appears with the intention of modeling robots using the theorical know-how about robotics lectured on colleges. Such tool allows to expand the complexity of the models seen in the more traditional books and papers. In this context, this work proposes the integration of the MATLAB toolbox to a SBC (Single Board Computer) with embedded Linux to control a robotic manipulator. For this project, therefore, were developed codes in C++ for the SBC, scripts for MATLAB, a library to interface with PCA9685 and a dedicated hardware. As result of this, emerges a system analog to an industrial robot, which allows the user to teach, calculate simulate and execute trajectories for a robotic manipulator.

**Key-words**: Robotics; Trajectory Planning; Embedded Linux; MATLAB.

# Lista de ilustrações

# Lista de tabelas

# Lista de abreviaturas e siglas

| | |
|---|---|
| DOF | Degrees of Freedom |
| FTP | File Transfer Protocol |
| GUI | Graphical User Interface |
| $I^2C$ | Inter-Integrated Circuit |
| IHM | Interface Homem-Máquina |
| LSPB | Linear Segment with Parabolic Blend |
| PWM | Pulse Width Modulation |
| ROS2 | Robot Operating System 2 |
| RTB | Robotics Toolbox |
| SCARA | Selective Compliance Assembly Robot Arm |
| VSFTPD | Very Secure File Transfer Protocol Daemon |

# Lista de símbolos

| | |
|---|---|
| $\{F\}$ | *Frame* que define um sistema de cordenadas |
| ${}^{A}T_{B}$ | Transformação homogênia que representa o frame $\{B\}$ em relação ao frame $\{A\}$ |
| $\mathbb{R}$ | Conjunto dos números reais |
| $\mathbb{R}^2$ | Conjunto de todos os pontos de um espaço Euclidiano 2D |
| $\mathbb{R}^3$ | Conjunto de todos os pontos de um espaço Euclidiano 3D |
| $\mathbb{S}$ | Conjunto de todos os ângulos em um círculo $[0, 2\pi)$ |
| $\mathcal{T}$ | do inglês, *task space* |
| $\mathcal{C}$ | do inglês, *configuration space* |
| $SE(n)$ | Grupo Euclidiano (do inglês, *special Euclidean group*). É o conjunto de poses em $n$ dimensões. |
| $\theta$ | Ângulo articular |
| $\dot{\theta}$ | Velocidade angular |
| $\ddot{\theta}$ | Aceleração angular |
| $\alpha$ | Ângulo de torção da notação de Denavit-Hartenberg |
| $a$ | Comprimento de elos da notação de Denavit-Hartenberg |
| $d$ | Comprimento do segmento de reta normal-comum entre eixos $z$ da notação de Denavit-Hartenberg |
| $t$ | Tempo |

# Sumário

# 1 Introdução

Este trabalho apresenta um estudo acerca da modelagem matemática de robôs industriais e das estratégias para planejamento de trajetórias. Tal modelo matemático e algoritmos são concebidos a partir da RTB (*Robotics Toolbox*) para MATLAB. Além disso, um manipulador robótico, análogo a um robô industrial, é desenvolvido usando Linux embarcado como plataforma integrada aos recursos da RTB.

## 1.1 Justificativas

O desenvolvimento de manipuladores robóticos robustos, tais como os industriais, exige uma série de conhecimentos das ciências exatas, tais como álgebra, cinemática, controle e matemática computacional. Nesse sentido, para o desenvolvimento de tais máquinas faz-se necessário um conjunto de ferramentas que permitam a computação de cálculos extensos e complexos em curtos intervalos de tempo.

Em se tratando da programação de robôs industriais, há ferramentas disponibilizadas pelos fabricantes que descomplicam a tarefa de se criar trajetórias. Entretanto, as empresas deste setor não permitem a divulgação dos métodos e algoritmos usados em suas soluções. Também não são fornecidos os detalhes envolvidos na solução tecnológica dos problemas de controle coordenado de manipuladores robóticos.

Tanto as plataformas comerciais, como o MATLAB, quanto aquelas *open-source*, como o Linux, podem ser usadas para o processo da modelagem e controle de manipuladores. A primeira, amplamente difundida no setor acadêmico, possui uma *toolbox* que suporta toda cinemática e dinâmica de manipuladores com descrições matriciais (CORKE, 1996). Por outro lado, uma *robot CPU* para controle e atuação baseada em Linux, seria um sistema permissivo para o estudo acerca dos projetos de robôs industriais. A possibilidade de um controlador de robôs usando Linux embarcado garantiria acesso à todas camadas do sistema, diferente do que é praticado pelos fabricantes de robôs comerciais.

Sendo assim, este trabalho tem como incentivo a possibilidade de desenvolver em plataformas abertas e conhecidas os desdobramentos matemáticos para cálculos de trajetórias em robôs industriais. Cálculos esses, que em máquinas comerciais, ocorrem ocultas sob linguagens *script*, hardware e sistemas operacionais proprietários. Além disso, o trabalho se compromete em desenvolver o *software* para aplicação dos algoritmos e projetar um protótipo para validação de conceitos. Estudos assim podem reduzir a assimetria do *know-how* acerca da modelagem manipuladores robóticos, que atualmente é formado de conhecimentos particulares para cada fabricante de robô.

## 1.2 Objetivos

A proposta deste trabalho é desenvolver a integração de ferramentas computacionais para o planejamento, cálculo e execução da trajetória de um manipulador robótico, que tem como controlador um SBC (*single board computer*) com Linux embarcado.

### 1.2.1 Objetivos Específicos

- Parametrização, na notação de Denavit-Hartemberg para um robô de 5 DOF (*degrees of freedom*) e estudo da matemática associada à robótica em seu estado da arte;
- Utilizar a *toolbox* para Matlab de Peter Corke (CORKE, c2023) em cálculos de cinemática (direta e inversa) e geração de trajetórias de um manipulador robótico com 5 DOF;
- Desenvolver camada de abstração de hardware que permita comunicação do Linux embarcado com *chip* gerador de PWM multicanal;
- Implementação de interface entre o Linux embarcado e computador com Windows 10, através de *teach pendant* virtual feito no MATLAB;
- Desenvolver hardware que permita aplicação de tal ambiente integrado, o qual inclui: manipulador robótico controlado por servomotores, gerador de PWM, SBC com Linux embarcado e computador com Windows 10;
- Disponibilização de códigos fonte sob as licenças abertas LGPL e GPL; e
- Prototipar um manipulador robótico com o conjunto de tecnologias estudadas;

# 2 Referencial Teórico

Este capítulo apresentará o referencial teórico utilizado no projeto do manipulador robótico e da geração de trajetória.

A base para o projeto de um manipulador robótico abrange fundamentos de matemática e física. Por esse motivo, os fundamentos sobre a representação matemática de um manipulador robótico serão discutidos na Seção 2.1. Além disso a cinemática de manipuladores e os principais algoritmos para geração de trajetórias serão brevemente analisados nas seções 2.2 e 2.3, respectivamente. A Seção 2.4, por sua vez, dará um panorama dos elementos que compõem um manipulador robótico como os utilizados na indústria. Finalmente, a Seção 2.5 discute alguns recursos usados para o desenvolvimento de aplicações no Linux embarcado.

## 2.1 Representação de um Robô

Usando matrizes é possível estabelecer métodos para descrever objetos, posições orientações e movimentos. Esses elementos de álgebra podem ser estendidos em notações diversas e aplicados para a descrição matemática de mecanismos, tais como robôs.

### 2.1.1 *Workspace, Task Space e Configuration Space*

O envelope de trabalho de um robô, ou *workspace*, é o espaço que contém todos os pontos que a ferramenta do robô alcança. Cada tipo de robô possui um envelope diferente, à depender da quantidade de articulações e como elas estão configuradas. Os robôs SCARA (*Selective Compliance Assembly Robot Arm*), por exemplo, possuem um *workspace* simples de topologia cilíndrica. Esse envelope está contido no espaço euclidiano tridimensional $\mathbb{R}^3$. O envelope de trabalho de um robô SCARA é descrito na Figura 1, em um artigo disponibilizado pela fabricante japonesa FANUC, com a finalidade de ilustrar a capacidade de trabalho de uma família de robôs (FANUC Ltd, 2018).

*Task space* $\mathcal{T}$ é o espaço necessário para uma tarefa que o robô realizará em uma aplicação. Ele é um espaço independente de quaisquer características do robô, e como o próprio nome diz, é específico da tarefa (CORKE, 2017).

Um robô SCARA que trabalha no *pick-and-place* de componentes em placas de circuito impresso, por exemplo, tem um *task space* $\mathcal{T} \subset \mathbb{R}^2$. Ou seja, esse *task-space* é uma superfície plana.

*Configuration space* $\mathcal{C}$ é um domínio especificado pelo número de graus de liberdade

de um sistema (CORKE, 2017). Ainda no exemplo do robô SCARA, com seus 4 graus de liberdade (3 rotativos, 1 linear), o *configuration space* seria $\mathcal{C} \subset \mathbb{S} \times \mathbb{S} \times \mathbb{R} \times \mathbb{S}$.

É usual que a dimensão do espaço $\mathcal{T}$ seja menor que a dimensão de $\mathcal{C}$ para evitar que dimensões do *task space* sejam inacessíveis ao manipulador robótico.

Figura 1 – Envelope de trabalho de um robô SCARA FANUC SR Series.

Fonte: FANUC Ltd (2018)

### 2.1.2 Notação de Denavit-Hartenberg

A notação de Denavit-Hartenberg surge em 1955 (DENAVIT; HARTENBERG, 1955) com a intenção de facilitar a representação de cadeias cinemáticas com múltiplas articulações, bem antes dos primeiros robôs comerciais. A proposta era de que as juntas (prismáticas ou rotativas) de mecanismos poderiam ser representadas por um conjunto de 4 parâmetros, que simbolizam, matematicamente, quatro operações matriciais simples. Essas transformações homogêneas são duas rotações e duas translações.

São dois parâmetros relativos ao elo, os quais são o comprimento do elo $a_n$ e o ângulo de torção $\alpha_n$. Os outros dois parâmetros são relativos às juntas: a distância de um elo com o próximo ao longo do eixo da junta $d_n$, e o ângulo da junta $\theta_n$.

A partir do sistema de referência de uma junta $x_n - z_n$, são aplicados 4 movimentos para se chegar ao *frame* seguinte $x_{n+1} - z_{n+1}$. Desses 4 movimentos, deriva-se a seguinte matriz de transformação homogênea, a qual relaciona o frame $n+1$ com a referência $n$ da notação de Denavit-Hartemberg:

$$
{}^{n}T_{n+1} = \begin{bmatrix} \cos\theta_{n+1} & -\sin\theta_{n+1}\cos\alpha_{n+1} & \sin\theta_{n+1}\sin\alpha_{n+1} & a_{n+1}\cos\theta_{n+1} \\ \sin\theta_{n+1} & \cos\theta_{n+1}\cos\alpha_{n+1} & -\cos\theta_{n+1}\sin\alpha_{n+1} & a_{n+1}\sin\theta_{n+1} \\ 0 & \sin\alpha_{n+1} & \cos\alpha_{n+1} & d_{n+1} \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.1}
$$

Conseguindo relacionar o frame de uma articulação de um robô com a anterior, logo é possível relacionar o referencial da base (denotada por $R$) de um robô qualquer com o referencial da sua própria mão (denotado por $H$). Tal relação será o produto das transformações homogêneas de cada junta (NIKU, 2017):

$$
{}^{R}T_{H} = {}^{R}T_{1}\,{}^{1}T_{2}\,{}^{2}T_{3}\cdots{}^{n}T_{n+1} = A_1A_2A_3\cdots A_{n+1}. \tag{2.2}
$$

O modelo de um robô pode, portanto, ser resumido como uma tabela de parâmetros $\theta$, $d$, $a$ e $\alpha$ como ilustrado na Tabela 1.

Tabela 1 – Tabela de parâmetros do modelo de Denavit-Hartemberg.

| # | $\theta$ | **d** | **a** | $\alpha$ |
|---|---|---|---|---|
| 0-1 | $\theta_1$ | $d_1$ | $a_1$ | $\alpha_1$ |
| 1-2 | $\theta_2$ | $d_2$ | $a_2$ | $\alpha_2$ |
| ⋮ | | | | |
| n-(n+1) | $\theta_{n+1}$ | $d_{n+1}$ | $a_{n+1}$ | $\alpha_{n+1}$ |

### 2.1.3 Posições degeneradas (singularidades)

A degeneração, singularidade, ou ainda *Gimbal lock*, ocorre quando o robô perde um ou mais graus de liberdade e não pode mais operar adequadamente.

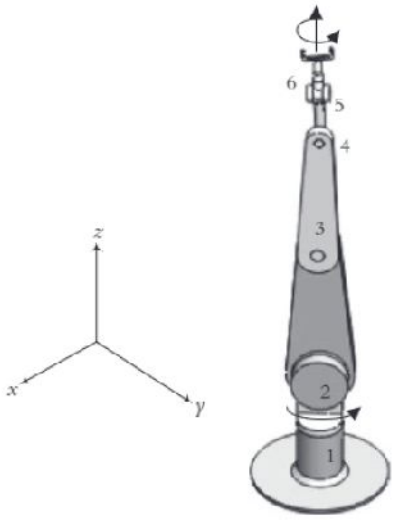
Tal condição pode ocorrer tanto quando o manipulador atinge seus limites físicos e não pode alcançar uma configuração, quanto em alinhamentos de eixos $z$ de articulações distintas (NIKU, 2017). A Figura 2 ilustra uma posição de singularidade, em que os eixos $z$ das articulações 1 e 6 estão alinhados e o robô, portanto, perde um grau de liberdade.

O problema das singularidades vão desde a impossibilidade de cruzá-las, até o surgimento de altas velocidades articulares ao se passar próximo delas. Devido a tais reações indesejadas, é muito importante identificar as posições degeneradas de um manipulador antes de se traçar trajetórias no espaço cartesiano (BONEV, 2019).

## 2.2 Cinemática de Manipuladores Robóticos

O intuito de se estudar cinemática na robótica é determinar posições e velocidades no robô. A analise cinemática é capaz de determinar o conjunto de variáveis do robô

Figura 2 – Exemplo de robô com 6 DOFs em posição degenerada.




Fonte: Niku (2017)

em seu *configuration-space*, que representam suas articulações, e como isso é refletido no movimento da ferramenta no espaço euclidiano tridimensional.

### 2.2.1 Cinemática direta

A cinemática direta é o conjunto de equações, específicas para um dado robô, que permite calcular a posição e orientação da ferramenta a partir de seus valores articulares. Uma vez desenvolvidas as equações cinemáticas diretas, consegue-se calcular onde o robô estará em qualquer instante, para dado conjunto de entradas articulares (NIKU, 2017).

### 2.2.2 Cinemática inversa

Supondo-se que é exigido colocar a mão do manipulador em um local e orientação desejados, é necessário saber quais os ângulos articulares e medidas dos elos que tornarão isso possível (NIKU, 2017). Para isso, desenvolve-se um outro conjunto de relações conhecidas como equações cinemáticas inversas.

Essas equações são as mais importantes, visto que o controlador do robô usará elas para calcular os valores articulares que deixam o manipulador robótico nas posições e orientações desejadas.

## 2.3 Algoritmos para Planejamento de Trajetórias

Formalmente, define-se percurso como uma série de configurações que um robô faz para ir de um lugar ao outro. Nessa definição não importa o tempo tomado ou as velocidades aplicadas.

Por outro lado, existe o conceito de trajetória, que define qual o momento e de que forma cada parte do percurso será atingida. Para a delineação dessa trajetória existem duas principais abordagens: o planejamento pelo espaço articular e o planejamento pelo espaço cartesiano.

### 2.3.1 Planejamento de trajetórias no espaço articular

Existem várias técnicas para o planejamento de trajetórias de um manipulador robótico no espaço articular. Entretanto apenas duas delas serão estudadas e aplicadas para este trabalho: o planejamento de trajetórias por polinômios de 5ª ordem e o LSPB (do inglês, *Linear Segment with Parabolic Blend*).

O uso de um polinômio de quinta ordem é vantajoso pois com ele é possível definir até 6 condições de contorno para a solução do problema, sendo estes: as posições, velocidades e acelerações iniciais e finais do trajeto (NIKU, 2017). Dessa maneira as funções polinomiais da trajetória podem ser definidas como:

$$\theta(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4 + + c_5 t^5, \tag{2.3}$$

$$\dot{\theta}(t) = c_1 + 2c_2 t + 3c_3 t^2 + 4c_4 t^3 + 5c_5 t^4, \tag{2.4}$$

$$\ddot{\theta}(t) = 2c_2 + 6c_3 t + 12c_4 t^2 + 20c_5 t^3. \tag{2.5}$$

Outro método para planejamento no espaço articular é o LSPB. Nessa técnica híbrida, são usados segmentos lineares de velocidade e segmentos polinomiais durante aceleração e desaceleração. Nesse caso, nas transições, a função que define $\theta(t)$ será uma parábola, a velocidade angular uma reta e a aceleração um valor constante.

$$\theta(t) = c_0 + c_1 t + \frac{1}{2} c_2 t^2, \tag{2.6}$$

$$\dot{\theta}(t) = c_1 + c_2 t, \tag{2.7}$$

$$\ddot{\theta}(t) = c_2. \tag{2.8}$$

Para o segmento linear, ou velocidade de cruzeiro, a velocidade será constante. Tal velocidade deve ser selecionada de acordo com a capacidade física dos atuadores.

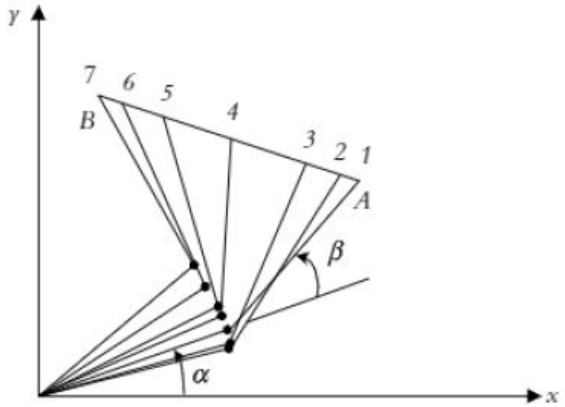
É importante perceber que, apesar de dar toda suavidade possível ao movimento, o planejamento pelo espaço articular não permite atender requisitos específicos de movimentação para a ferramenta. Por exemplo, no caso de haver a necessidade que a ferramenta transite em velocidade linear constante para aplicação de uma cola ou soldagem.

Além disso, pode ser necessário que a ferramenta siga um caminho com formato específico no espaço, como uma reta ou um círculo, por exemplo. Por isso, justifica-se o uso do planejamento de rotas no espaço cartesiano, apesar da robustez apresentada nas técnicas por polinômios de quinta ordem e LSPB.

### 2.3.2 Planejamento de trajetórias no espaço cartesiano

O planejamento de trajetória usando técnicas no espaço articular apenas podem prever o comportamento de cada articulação. Entretanto, o comportamento de posição, velocidade e aceleração do atuador final são imprevisíveis em tal cenário. Nesse contexto surge o planejamento usando o espaço cartesiano.

Figura 3 – Exemplo de trajetória linear no espaço cartesiano. O robô contém 2 DOFs




Fonte: Niku (2017)

A trajetória no espaço cartesiano se resume em traçar a posição e orientação da mão do robô em uma série de pontos, os quais relacionam o *frame* do atuador final com o sistema de referência cartesiano. Nesse algoritmo, os ângulos articulares devem ser calculados repetidamente por meio das equações cinemáticas inversas do robô (NIKU, 2017).

Trajetórias no espaço cartesiano podem fazer a ferramenta do robô seguir um caminho arbitrário, seja ele retilíneo como na Figura 3, circular ou com formas mais complexas. Além disso é possível determinar requisitos de velocidade e aceleração exercidos na mão do braço robótico.

O planejamento no espaço cartesiano pode utilizar de diferentes recursos matemáticos para gerar suas trajetórias tais como interpolação linear e interpolação linear

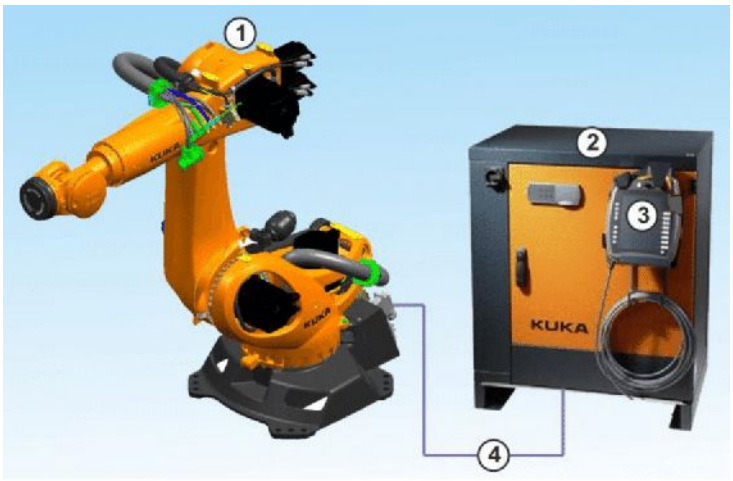
esférica (CORKE, 2017), e perfis de velocidade trapezoidais (CORKE, 2020), por exemplo. Além dessas modalidades, existem diversaspesquisas que propõem técnicas variadas para geração de trajetórias cartesianas, tais como utilizando curvas de Bézier (XU et al., 2014) ou técnicas de otimização (SINGH; BANGA, 2021).

## 2.4 Elementos de um Robô Industrial Comercial

O termo robô industrial se refere ao sistema mecatrônico, servo-atuado, e ordenado por um controlador lógico que normalmente é empregado para realizar variadas tarefas em ambiente industrial. Como descrito em (GADALETA, 2018), e ilustrado na Figura 4, esse sistema é composto por:

- manipulador (estrutura mecânica e atuadores);
- gabinete de controle (unidade de controle, *servo-drivers* e eletrônicos auxiliares); e
- dispositivo periférico para interface homem-máquina.

Figura 4 – Arquitetura básica para um robô industrial comercial que contém: 1) manipulador robótico, 2) gabinete, 3) *teach-pendant* e 4) cabeamento.



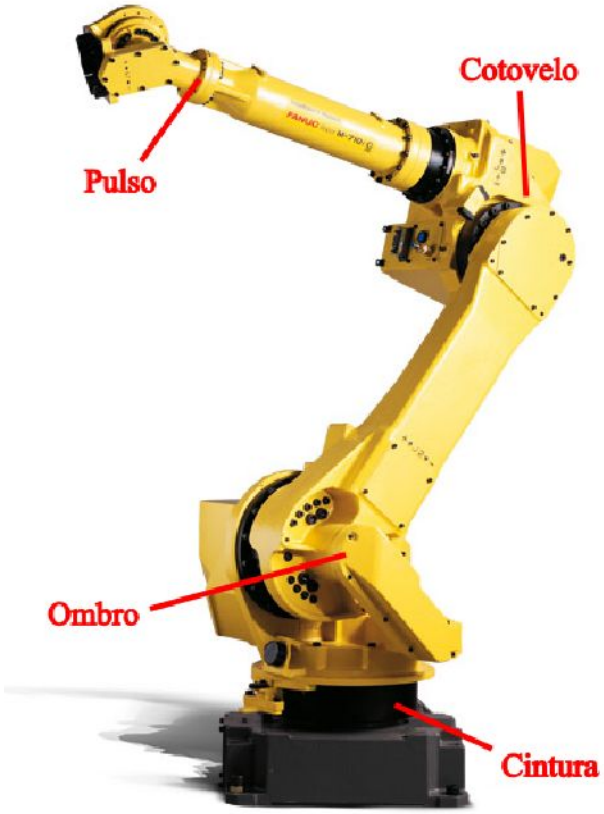

Fonte: Gadaleta (2018)

Dentro do gabinete, estão dispostos os *servo-drivers*, cada um responsável pelo controle de uma articulação. Além disso também há no gabinete a unidade de controle central do robô, interfaces de entrada e saída, dispositivos de segurança e eletrônicos auxiliares tais como fontes de alimentação.

Um dispositivo provido de Interface Homem Máquina (IHM), tradicionalmente chamado *teach-pendant*, é presente conectado ao gabinete. Tal IHM permite ao usuário mover o manipulador, alterar configurações, executar programas, assim como programar o robô.

Em se tratando do manipulador, é comum que manipuladores industriais sejam classificados como robôs antropomórficos. Tal denominação se deve ao fato de as articulações do robô serem dispostas de forma semelhante à de um braço humano. Por isso, é usual adotar uma nomenclatura que associa cada articulação de um robô industrial com uma parte do corpo humano. Um caso em que tal nomenclatura é adotada é nos casos de manipuladores articulados com 6 graus de liberdade. Nesse caso as articulações serão nomeadas como demonstrado na Figura 5.

Figura 5 – Articulações de um manipulador robótico antropomórfico.




Fonte: FANUC Ltd (2023)

Nesses robôs é comum que três articulações estejam associadas ao posicionamento da ferramenta no espaço: a cintura, o ombro e o cotovelo. Os demais graus de liberdade, responsáveis por orientar a ferramenta no espaço são atingidos pela articulação do pulso. Por isso, o pulso de um robô antropomórfico de 6 graus de liberdade é uma articulação esférica.

## 2.5 Recursos para o Desenvolvimento em Linux Embarcado

O uso de sistema operacional Linux é justificável por constituir uma base versátil onde inúmeras aplicações de *software* podem ser implementadas. O uso de linguagens de programação diversas, interfaces gráficas, conectividade de rede e ambiente multi-tarefa tornam o Linux atrativo para aplicações embarcadas com maior grau de complexidade. Dentro desse amplo número de recursos, é possível destacar aqueles que são úteis para o desenvolvimento do manipulador robótico proposto neste trabalho.

### 2.5.1 Protocolo $I^2C$ em Sistemas Linux

O *Inter-Integrated Circuit* ($I^2C$) é um protocolo de comunicação serial, desenvolvido pela Philips Semiconductors, e que se baseia em um simples barramento à 2 fios para comunicação entre circuitos integrados (NXP Semiconductors, 2021). As únicas duas linhas necessárias para esse barramento são: *Serial Data* (SDA) e *Serial Clock* (SCL).

Como a linha de dados pode ser usada para recepção e envio de sinais, diz-se que o protocolo é, portanto, bidirecional. Além disso, como a linha de dados é única, o envio e recepção de dados não podem ser realizados simultaneamente e o protocolo se classifica como *half-duplex*.

No Linux, uma das formas de gerenciar a comunicação $I^2C$ é por meio do sistema `sysfs`. Isso significa que os barramentos e dispositivos no sistema operacional são representados por arquivos organizados no diretório `/sys/class/i2c-dev/`. Isso permite que aplicações que utilizem comunicação $I^2C$ sejam implementadas a partir de operações de arquivo (NAZARI, 2023), abstraindo a interação de um programa com o *device driver* do sistema operacional.

É possível abstrair ainda mais a interação com os *drivers* para $I^2C$ do sistema operacional por meio das linguagens de programação. Uma forma de fazer isso é encapsulando as interações do `sysfs` com orientação à objetos. Um exemplo é a biblioteca `I2CDevice` que traduz as interações comuns com o barramento em métodos de uma classe. Essa abordagem, além de simplificada, é flexível, pois diferentes dispositivos que usam $I^2C$ podem ser representados por meio de classes que herdam o template `I2CDevice` (MOLLOY, 2016). Essa última abordagem, orientada à objetos e implementada em C++, é a adotada para este trabalho.

### 2.5.2 Servidor FTP em Sistemas Linux

O protocolo de transferência de arquivos, ou no inglês *File Transfer Protocol* (FTP), é um mecanismo padrão para transferência de arquivos entre *hosts* por meio de *sockets*

TCP/IP (FOROUZAN, 2007). O protocolo usa, tradicionalmente, as portas 21 para conexão de controle e 20 para conexão de dados.

A rede possui o tradicional modelo cliente servidor, em que o programa do usuário, do lado do cliente, faz leitura e escrita de arquivos em um servidor FTP remoto. A conexão na porta de controle fica aberta durante toda a sessão, enquanto a porta de dados é constantemente aberta e fechada sempre que há uma operação de escrita ou leitura de arquivos.

Uma das melhores formas de se implementar um servidor FTP no Linux é por meio do *Very Secure File Transfer Protocol Daemon* (vsftpd). O vsftpd é um servidor FTP de código aberto para uso em sistemas *UNIX-like*, incluindo o Linux. O servidor permite uma ampla configurabilidade, como possibilidade de configurar acesso e permissões para clientes. Por se tratar de um *daemon* ele roda constantemente como um processo e independente das aplicações do usuário.

### 2.5.3 Compilador g++ e *GNU Make*

O *GNU Compiler Collection* é um conjunto de compiladores integrados com a capacidade de compilar programas em C, C++, Objective-C, Ada, Fortran, Java, ou Treelang (Free Software Foundation, ). O termo ”G++”é utilizado para compilar códigos na linguagem C++, e usado para enfatizar a linguagem do código a ser compilado pelo GCC. O G++ compila o código diretamente da linguagem C++, sem criar uma versão em C intermediária para o programa.

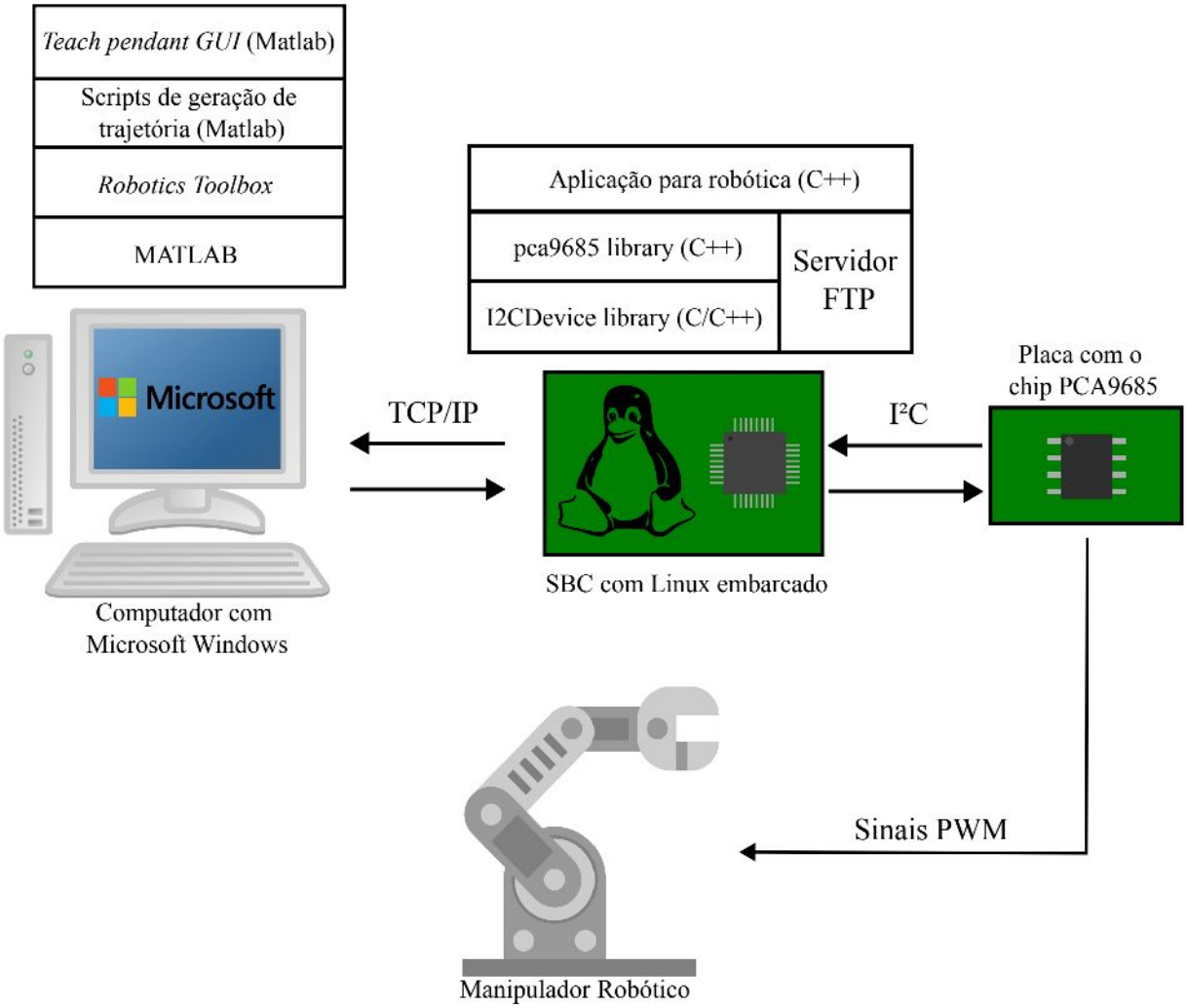
Já o *GNU Make* é um utilitário usado para otimizar o processo de compilação. Ele determina quais porções do código devem ser recompiladas antes de executar comandos para compilação (Free Software Foundation, 2023). Para usar o `make`, deve ser criado um arquivo chamado `makefile`, o qual descreve as relações entre os diversos arquivos do código fonte de um programa e executa apenas os comandos para o compilador atualizar o que realmente for necessário. Simplificadamente, esse `makefile` é um *script* com uma sintaxe característica, com rotinas chamadas receitas. Uma vez que um `makefile` foi elaborado, é possível recompilar um programa chamando o comando `make` no *shell*.

# 3 Metodologia

A multidisciplinaridade de um projeto na área da robótica requer diferentes frentes de desenvolvimento. Em conformidade com esse cenário, o projeto do manipulador robótico deste trabalho será dividido em três esferas: o chip PCA9685 e sua interface com o Linux, o hardware para o manipulador e o MATLAB.

A arquitetura do projeto pode ser descrita como uma integração de ambientes para planejamento de trajetória em um escopo matemático e sua execução em um ambiente real. A execução compreende desde o processamento dos sinais de controle até a plataforma de hardware para o acionamento adequado dos atuadores. Essa arquitetura é descrita simplificadamente de acordo com a Figura 6.

Figura 6 – Arquitetura integrada para planejamento e execução de trajetória de um manipulador robótico.




Fonte: autoria própria (2023)

Além disso, arquitetura visa criar elementos paralelos aos de um robô industrial comercial. Portanto, existem elementos que têm funções análogas àquelas de *servo-drivers*, *teach-pendants*, unidades de processamento central robótica e manipuladores antropomórficos.

Na Seção 3.1 será abordado o chip PCA9685, discutindo-se suas características, funcionamento e vantagens. A Seção 3.2 fala sobre o ambiente do MATLAB e da RTB (*Robotics Toolbox*). A Seção 3.3 analisa a melhor forma de se construir o robô com auxílio do MATLAB. A Seção 3.4 explicará como o próprio MATLAB, além de performar os cálculos, pode também servir de interface gráfica para a programação de tarefas de um manipulador robótico. Por fim, a Seção 3.5 demonstra a solução integrada.

## 3.1 Geração de Sinais de Controle com PCA9685

Como o sistema operacional Linux embarcado no SBC não possui características de tempo real, a geração de sinais de controle pode ser prejudicada. No caso do manipulador a ser desenvolvido, um PWM (*Pulse Width Modulation*) gerado por software no Linux poderia ter seu *duty cycle* instável ao longo do tempo. Como efeito colateral final, os servomotores controlados por tal sinal apresentariam vibração constante e irregular, fenômeno este conhecido como *jittering*.

Então, a solução é usar um dispositivo intermediário para geração de sinais de controle: o PCA9685. Trata-se de um chip com a capacidade de gerar um sinal de PWM estável com 12 bits de precisão (NXP Semiconductors, 2015). Ele se comunica com o SBC pelo protocolo $I^2C$.

Foi definido previamente à implementação da interface com o *chip* que deveria ser feito de forma a ser portável entre diferentes SBCs e distribuições de Linux embarcado. Além disso, tal biblioteca deveria ter também a capacidade de ser expandida para um pacote para ROS2 (*Robot Operating System*) em trabalhos futuros.

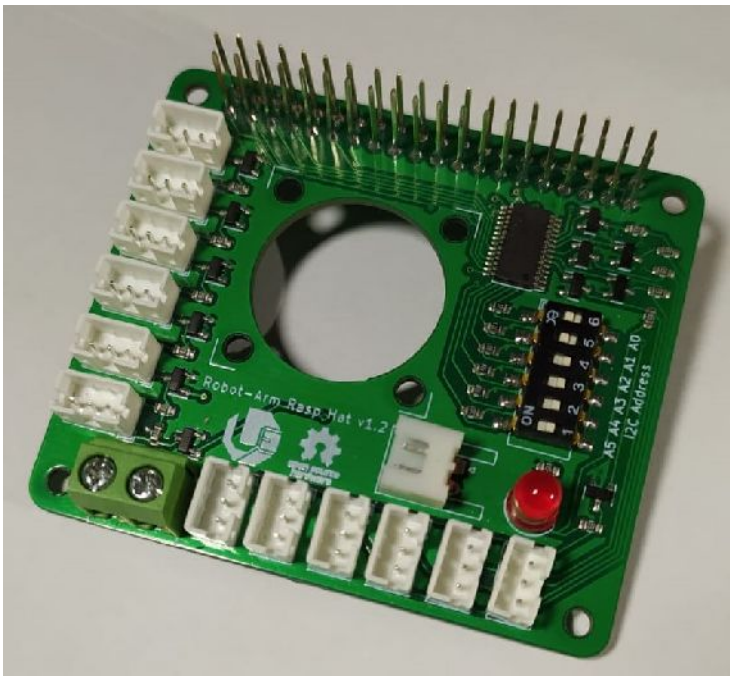
Sendo assim, a melhor alternativa foi a de implementar uma biblioteca em C++ para a camada de abstração de hardware com o PCA9685. Essa biblioteca encapsula o *template* I2CDevice, uma classe genérica para comunicação por protocolo $I^2C$. O métodos criados para a biblioteca incluem funções para configurar as funcionalidades do *chip* e controlar o ciclo de trabalho dos múltiplos sinais de saída.

Além de criar sinais de PWM com a temporização precisa e estável, para controle de motores também é necessário compatibilizar o nível tensão do sinal. Desse modo, para que os sinais sejam entregues ao servo motor com o nível de tensão adequado, um circuito de saída baseado em transístores N-MOS foi incluído na solução. Tanto o *chip* os circuitos *drivers* de saída foram integrados pelo desenvolvimento de uma placa de circuito impresso.

Além dos *drivers* para adequação de tensão, a placa possui interface de hardware para configuração do endereço do *slave* $I^2C$, conectores para servomotores e uma barra de pinos compatível com o SBC Raspberry Pi. O esquemático completo da placa é exposto no Anexo A.

Tanto o desenvolvimento da biblioteca em C++ quanto da placa de circuito impresso, assim como o resultado dessa solução para geração de PWM são melhor descritos em Rodrigues (2023).

Figura 7 – Hardware desenvolvido para controle de servomotores com Raspberry Pi.




Fonte: autoria própria (2023)

A biblioteca possui um exemplo de aplicação voltado para teste de servomotores controlados por PWM. Esse programa é usado para analisar a atuação de servo motores com base na geração de pulsos de larguras específicas. O código serve também para determinar os limites angulares do servomecanismo e qual a largura de pulso correspondente à tal atuação. Tal análise é necessária pois raramente tais dados são fornecidos pelos fabricantes de servomotores, principalmente aqueles mais simples e de baixo custo, controlados por PWM.

Para o motor da *Tower Pro*, modelo MG996R, selecionado para atuação no manipulador robótico desse trabalho, o fabricante apenas forneceu a faixa angular e a frequência de operação do sinal.

Foi determinado, para o angulo de $0^o$, um sinal de controle com largura de pulso de 1ms. Sendo assim, foi obtido empiricamente que, para o robô alcançar o ângulo de $120^o$ proposto pelo *datasheet*, a largura de pulso deveria ter um valor de 2.2ms.

Considerando que a relação entre a largura de pulso do sinal de PWM e o angulo de saída do servomotor é linear, pode-se determinar a relação entre o *duty cycle* do sinal e ângulo desejado no eixo do motor. Esse *duty cycle* será expresso na forma de um número inteiro de 12 bits que deve ser escrito nos registradores do PCA9685 de forma que ele gere o sinal com a largura de pulso desejada. Outro fator a ser considerado é a frequencia do sinal de controle do motor, nesse caso, de $50Hz$. Assim, foi determinada a seguinte relação:

$$\frac{120-0}{\theta-0} = \frac{(2.2-1)(4096 \cdot 50/1000)}{n-(4096 \cdot 50/1000)}$$

que permite determinar o valor inteiro $n$ a ser escrito nos registradores de um canal do *chip* para gerar um ângulo $\theta$ desejado. De outro modo, tem-se:

$$n = \frac{1.2(4096 \cdot 50/1000)}{120} \cdot \theta + (4096 \cdot 50/1000) \tag{3.1}$$

Com a Equação 3.1 foi possível calcular, em bits, os ciclos de trabalho para o motor em posição mínima, máxima e neutra, assim como implementar uma macro para conversão de ângulos recebidos em graus para *duty cycle* em bits.

Figura 8 – Limites angulares e expressão de conversão de ângulo para bits no código.

```
#define MG996R_MIN_POSITION   1*(4096*50/1000)
#define MG996R_MID_POSITION   1.6*(4096*50/1000)
#define MG996R_MAX_POSITION   2.2*(4096*50/1000)
#define MG996R_RAD_TO_PWM(a) (round((1.2(4096*50/1000)/120)*a + (4096*50/1000)))
#define MG996R_MIN_STEP       30000 /* microsseconds */
```

Fonte: autoria própria (2023)

Além disso, foi também obtido experimentalmente que o período mínimo em que o servomotor poderia receber comandos sucessivos era de 30 [ms]. Abaixo desse período o circuito no motor não reconhece alterações na mudança da largura de pulso do sinal. As definições dos limites do motor e a macro para cálculo do ciclo de trabalho em bits foram implementados em C++, como ilustrado na Figura 8.

## 3.2 MATLAB e a *Robotics Toolbox*

Para o processamento matemático computacional relacionado à cinemática e trajetória, foi usado o MATLAB e a *toolbox* RTB. Além dos recursos matemáticos também são utilizadas ferramentas gráficas para simulação do manipulador e visualização das variáveis físicas do robô.

### 3.2.1 Modelagem do manipulador

A frente de desenvolvimento relacionada à aplicação da parte matemática foi desenvolvida usando a RTB para MATLAB. A *toolbox* será para a modelagem do robô e cálculos de trajetórias. O pacote permite ainda simulações da dinâmica do sistema e integração com algoritmos de controle.

Para modelar o robô a partir da *toolbox*, foi necessário previamente elaborar um modelo para o robô de acordo com a metodologia de Denavit-Hartemberg. Uma vez desenvolvido o modelo, ele é usado para criar instâncias do tipo `Link`. Esses objetos recebem como parâmetros o conjunto de quatro variáveis usadas na transformação homogênea entre uma junta e outra do robô.

Para o robô adotado neste trabalho, foi modelado o conjunto de parâmetros segundo a notação Denavit-Hatermberg, conforme o apresentado na Tabela 1.

Tabela 2 – Tabela de parâmetros para modelo de Denavit-Hartemberg do robô de 5 DoF.

| **Transformação** | $\theta$ | **d** | **a** | $\alpha$ |
|---|---|---|---|---|
| 0-1 | $\theta_1^*$ | $L_1$ | 0 | $\frac{\pi}{2}$ |
| 1-2 | $\theta_2^*$ | 0 | $L_2$ | 0 |
| 2-3 | $\theta_3^*$ | 0 | $L_3$ | 0 |
| 3-4 | $\theta_4^*$ | 0 | $L_4$ | -$\frac{\pi}{2}$ |
| 4-5 | $\theta_5^*$ | $L_5$ | 0 | 0 |

Para um modelo mais coerente com o sistema real, além do modelo de Denavit-Hartemberg, é necessário considerar a limitação de cada articulação. Em todas as articulações do robô construído são usados servomotores modelo MG996R, do fabricante *Tower-Pro*. Tal motor tem uma rotação máxima de 120° (Tower Pro, c2023). Essa limitação pode ser adicionada a um objeto da classe `Link` associando-se valores ao atributo `qlim`.

Um último parâmetro para uma modelagem que visa otimizar o workspace do robô, é a criação de *offsets* angulares. Tais atributos devem somar um ângulo constante ao valor de uma junta, de forma que quando a controladora ordena que tal articulação vá para a inclinação de 0° elá irá, na verdade, para uma inclinação de $0 + \phi$. Adicionar tais *offsets* faz com que o robô consiga maior volume de atuação dentro de uma localidade desejada do espaço euclidiano.

O objeto que representa o robô apenas é criado com uma instância da classe `SerialLink`, que recebe como parâmetro um vetor de objetos de elos (`Links`). O trecho de código que cria um modelo matemático do robô no MATLAB é demonstrado na Figura 9.

É possível usar um método para plotar a representação matemática do robô. Com tal método, a *toolbox* cria uma representação gráfica tridimensional simplificada do manipulador. Apesar da simplificação visual, o `plot` é uma representação concisa do modelo matemático do robô. Esse modelo é exibido na Figura 10.

Figura 9 – Trecho de código que modela matematicamente o robô no MATLAB.

```
L1 = 82;
L2 = 104;
L3 = 98;
L4 = 28;
L5 = 50;

E(1) = Link([0 L1 0  pi/2], 'standard');
E(2) = Link([0 0  L2 0],    'standard');
E(3) = Link([0 0  L3 0],    'standard');
E(4) = Link([0 0  L4  -pi/2],'standard');
E(5) = Link([0 L5 0  0],    'standard');

E(1).offset = pi/6;
E(2).offset = pi/6;
E(4).offset = -pi/2;

E(1).qlim = [0 120*pi/180];
E(2).qlim = [0 120*pi/180];
E(3).qlim = [0 120*pi/180];
E(4).qlim = [0 120*pi/180];
E(5).qlim = [0 120*pi/180];

robot = SerialLink(E, 'name', 'robo 5dof');
```

Fonte: autoria própria (2023)

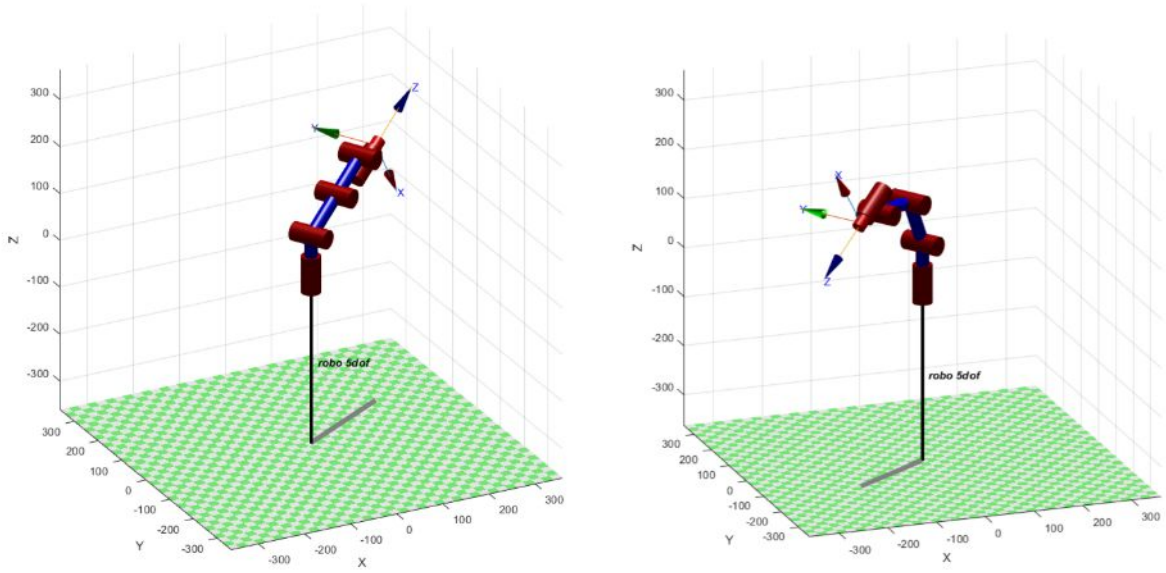
### 3.2.2 Algoritmos de trajetória no MATLAB

Os algoritmos, descritos matematicamente na Seção 2, podem ser usados no MATLAB por meio de funções inseridas no RTB. Essas funções permitem a criação de trajetórias no espaço articular (polinomial de 5ª ordem e LSPB) ou no espaço cartesiano (linear).

As trajetórias em espaço articular independem do uso de cinemática inversa e pode ser calculada apenas informando valores articulares iniciais e finais e um vetor de tempo. Este vetor de tempo deve conter todos os *frames* no qual o algoritmo calcula os valores articulares. Para gerar uma trajetória usando a estratégia de polinômios de 5ª ordem usa-se a função `jtraj`, e para LSPB usa-se a função `mtraj` (Figura 11).

Para a geração de uma trajetória no espaço cartesiano, a estratégia é substancialmente mais complexa. Em uma breve síntese, o processo se resume em criar um vetor de matrizes de transformações homogêneas, através do método `ctraj`, e calcular a cinemática inversa do robô para cada uma dessas transformações. As matrizes de transformação homogênea inicial e final são obtidas pela cinemática direta do robô em posições predeterminadas. A cinemática direta pode ser calculada pelo método `fkine`.

Figura 10 – Representação gráfica simplificada do robô em: A) sua posição zero e B) uma posição arbitrária.




A B

Fonte: autoria própria (2023)

Figura 11 – Arquitetura integrada para planejamento e execução de trajetória de um manipulador robótico.

```
q0 = [0 0 0 0 0];
q1 = [0.2618 2.0944 1.633628 0.439823 1.0472];
qf = [2.0944 0.9110619 1.528908 1.67552 1.0472];
t = 0:0.03:5;

[Q1, Qd1, Qdd1] = jtraj(q0, q1, t); % polinomial
[Q2, Qd2, Qdd2] =  mtraj(@lspb, q1, qf, t); % LSPB
```

Fonte: autoria própria (2023)

Para o cálculo da cinemática inversa, o problema apresenta impedimentos quanto ao uso da *toolbox*. O pacote permite duas formas de cálculo para cinemática inversa: uma solução numérica e outra analítica.

Usando a solução numérica, para o robô projetado, o algoritmo nem sempre converge, fazendo com que seja impossível determinar os ângulos articulares para determinadas poses do robô. É provável que tal situação ocorra por se tratar de um robô sub-atuado, isto é, que não tem graus de liberdade o suficiente para se posicionar em 3 eixos cartesianos e se orientar em três eixos de rotação em um espaço tridimensional. A *toolbox* também conta com uma solução analítica, que entretanto é restrita ao modelo padrão de robôs

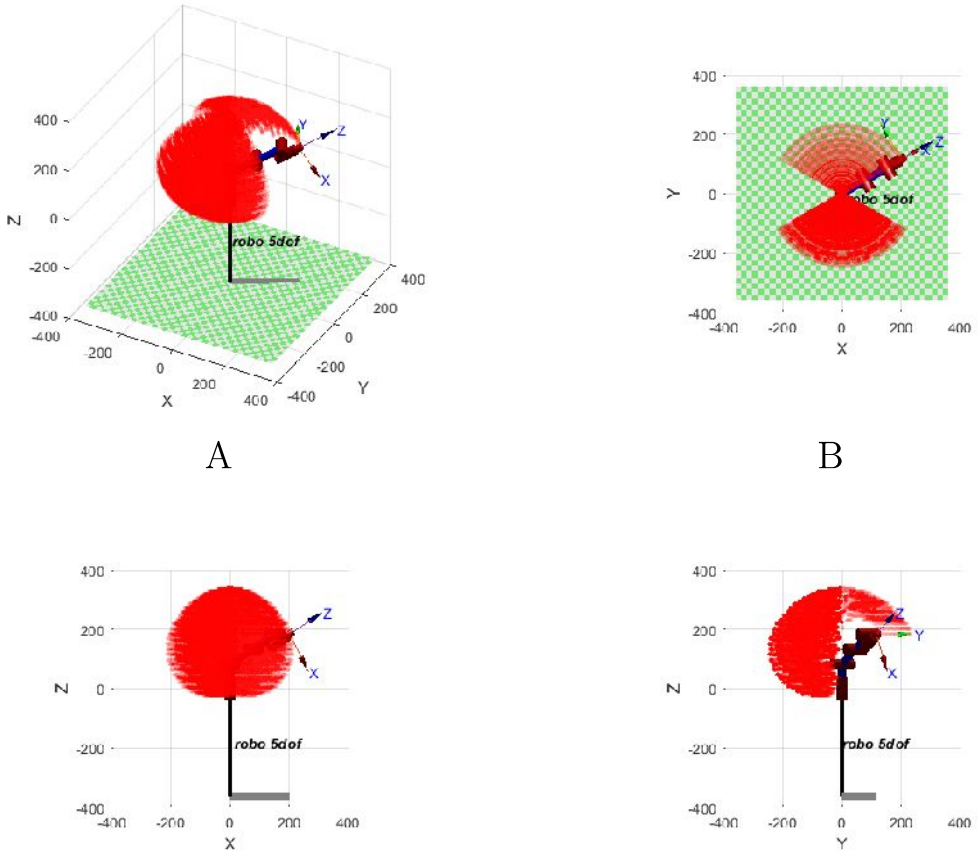
antropomórficos industriais. Robôs que se enquadram nessa categoria são manipuladores robóticos antropomórficos com 6 graus de liberdade e que apresentam articulação esférica em seu pulso. Tal método também é incapaz de resolver a cinemática inversa do robô projetado.

Dessa forma foi necessário o desenvolvimento da cinemática inversa analítica para o robô de 5 DoF. Tal solução é deduzida conforme demonstrado no Anexo B, a partir do modelo de Denavit-Hartemberg do robô e de sua cinemática direta.

## 3.3 Construção do Manipulador Robótico

O robô foi construído de forma que o envelope de trabalho fosse o maior possível, considerando que seu *task-space* fosse alguma superfície plana e horizontal posicionada na área frontal de trabalho do robô.

Figura 12 – Envelope de trabalho do robô construído em: A) ângulo arbitrário, B) visão superior, C) visão frontal e D) visão lateral




Fonte: autoria própria (2023)

Para analisar quais *offsets* angulares permitiam maior manipulabilidade sobre esse workspace, foi implementado um *script* (Anexo C) que plota de maneira gráfica o envelope de trabalho do robô. Esse envelope é representado na simulação como uma nuvem vermelha, que corresponde à todo o volume que a ferramenta do robô alcança.

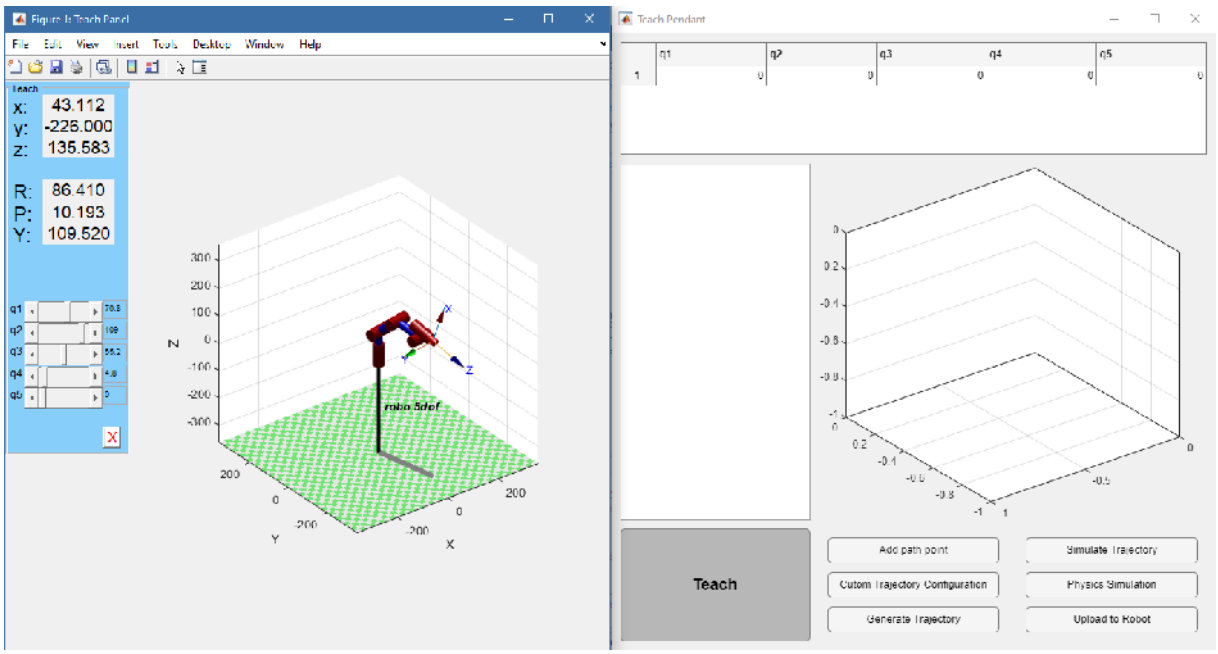
Uma configuração satisfatória foi que possui *offsets* de 60° nas juntas da cintura e do ombro e um *offset* de -90° no ângulo responsável pelo movimento de arfagem do pulso. Esse resultado pode ser visualizado na Figura 12. Nas visões frontal e lateral é possível perceber que superfícies horizontais posicionadas de frente para o robô possuem grande área.

Uma vez definidos os *offsets*, construiu-se, então a estrutura mecânica do manipulador de forma que o *duty cycle* mínimo colocasse os elos com *offsets* nas inclinações escolhidas.

## 3.4 Elaboração de um *Teach Pendant* gráfico com MATLAB

A GUI (do inglês, *Graphical User Interface*) é implementada no computador com sistema Microsoft Windows e MATLAB. O *teach pendant* do robô é uma aplicação baseada em duas janelas. Ambas as janelas são figuras geradas pelo script `teach_pendant.m` feito no MATLAB. Essa interface é apresentada na Figura 13.

Figura 13 – Interface gráfica do usuário implementada para controlar, ensinar e simular o manipulador robótico.




A B

Fonte: autoria própria (2023)

Apesar de ser um *software* voltado para cálculo numérico, também é possível usar

o MATLAB para desenvolvimento de aplicações gráficas. Uma das formas de se fazer isso, e também a mais simples, é desenvolver interfaces baseadas em `uifigures`. Nesse método, uma interface interativa é criada programando-se no próprio *script* de extensão `.m` de tal forma que toda aplicação gráfica seja construída em uma janela de figura (MathWorks®, c2023).

A aplicação gráfica baseada em `uifigures` é construída sobre estruturas chamadas containers. Similar à programação orientada à objetos, os containers são uma abstração que representam classes de objetos que vão desde a própria `uifigure` até elementos gráficos como botões, gráficos, caixas de entrada de texto, tabelas e *checkboxes*.

A primeira das janelas, 13.A, é uma `uifigure` que contém botões com todas as funcionalidades necessárias para ensinar trajetórias ao manipulador. Além disso, existem funcionalidades para visualização, como um `plot` tridimensional que permite o usuário ver o delineado da trajetória no espaço. Outras formas de visualização são a simulação gráfica que cria um vídeo com base no método `plot` da RTB, e uma simulação física que plota as grandezas cinemáticas e dinâmicas das juntas ao longo do tempo.

A segunda janela, 13.B, é responsável por movimentar o robô usando a interface gráfica provida pelo método `teach` da RTB. A função `teach` é um método da classe ETS3 (*Elementary transform sequence in 3D*), que engloba objetos matemáticos representados no Grupo Euclidiano tridimensional SE(3) (CORKE, 2020). Apenas objetos que herdam a classe SE(3), tais os da classe `SerialLink`, podem ser plotados com o método `teach`. Entretanto, o container `uifigure` espera que outros containers gráficos, tais como `uibutton`, sejam inclusos na janela. Dessa forma a interface gráfica fornecida pelo `teach` não pode ser incluída na `uifigure` e, por isso, a interface possui duas janelas.

## 3.5 Aplicação Integrada para Programação, Cálculos e Execução de Trajetórias

Assimilando todas as diferentes frentes de desenvolvimento é possível construir uma aplicação integrada para ensinar, planejar e executar trajetórias em manipuladores robóticos, tal como no esquema ilustrado na Figura 6.

Durante o processo de ensinar trajetórias ao robô, é implementado um *socket* TCP/IP para troca de dados entre o *teach pendant* e a Raspberry Pi. Nessa implementação, o computador com MATLAB assume o papel de cliente e a aplicação da Raspberry Pi de servidor. Tanto o MATLAB quanto a linguagem C++ possuem recursos nativos para implementação. Cada novo ângulo na janela de movimentação do MATLAB é enviado em tempo real para a Raspberry que repassa comandos ao PCA9685, para movimentação dos motores.

Figura 14 – *Task* responsável pela movimentação do robô em tempo real via *GUI*.

```
void teaching_proccess(pca9685 &servo_driver, std::mutex &driver_mutex){
  std::vector< float > joints;
  std::string aux;

  std::cout << "Starting RPi Server." << std::endl;
  SocketServer server(54321);
  std::cout << "Listening for a connection..." << std::endl;
  server.listen();

  while(true){
    std::string rec = server.receive(54);
    unsigned int last_space_pos = 0;

    for(int i=0; i<rec.length(); i++){
      if(rec.at(i) == ' '){
        if(i - last_space_pos > 1){
          joints.push_back( std::stof(aux) );
          aux.clear();
        }
        last_space_pos = i;
      }
      else{
        aux.push_back(rec.at(i));
      }
    }
    joints.push_back( std::stof(aux) );
    int ch = 0;

    driver_mutex.lock();
    for(std::vector<float>::iterator it=joints.begin(); it!=joints.end(); it++){
      std::cout << *it << " ";
      servo_driver.set_pwm_duty_cycle((pca9685::CHANNEL)ch, MG996R_DEG_TO_PWM(*it));
      ch++;
    }
    driver_mutex.unlock();

    aux.clear();
    joints.clear();
    std::cout << "\n\n";
    rec.clear();
    std::this_thread::sleep_for( std::chrono::milliseconds(30) );
  }
}
```

Fonte: autoria própria (2023)

Além de ser capaz de receber dados em tempo real, a Raspberry deve ser capaz de realizar trajetórias autônomas assim que a interface do usuário fizer o *upload* do arquivo. Para a gestão dos arquivos de trajetórias foi usado o *daemon* para servidores FTP vsftpd. Para que o sistema identificasse a adição ou modificação de arquivos no diretório do servidor FTP, foi utilizada o `inotify`. Trata-se de um recurso do kernel do Linux que permite aplicações requisitarem o monitoramento de diretórios ou arquivos do sistema. Quando alguma mudança ocorre nos arquivos monitorados a aplicação é notificada (LOVE,

2005). No contexto da aplicação do robô, assim que o `inotify` notificasse a aplicação sobre alteração no arquivo de trajetória do servidor FTP, a trajetória autônoma se inicia.

Figura 15 – *Task* responsável pelo servidor FTP e pela movimentação autônoma.

```
void ftp_proccess(pca9685 &servo_driver, std::mutex &driver_mutex){
  std::string path_to_be_watched = "/home/vini-rasp/FTP/trajectory_ftp_server";
  signal(SIGINT, sig_handler);

  fd = inotify_init();

  if(fcntl(fd, F_SETFL, O_NONBLOCK) < 0)
    exit(2);

  wd = inotify_add_watch(fd, path_to_be_watched.c_str(), IN_MODIFY | IN_CREATE);
  if(wd==-1)
    std::cout << "Could not watch " << path_to_be_watched << std::endl;
  else
    std::cout << "Watching " << path_to_be_watched << std::endl;

  while(true){
    int i=0, length;
    char buffer[BUF_LEN];

    /* reading the buffer */
    length = read(fd, buffer, BUF_LEN);

    /* processing the events which has ocurred */
    while(i<length){
      struct inotify_event *event = (struct inotify_event*) &buffer[i];

      if(event->len){
        if((event->mask & IN_CREATE)||(event->mask & IN_MODIFY)){
          std::cout << "The file " << event->name << " was created or modified" << std::endl;
          driver_mutex.lock();
          execute_file_trajectory(path_to_be_watched, servo_driver);
          driver_mutex.unlock();
        }
      }
      i += EVENT_SIZE + event->len;
    }
    std::this_thread::sleep_for( std::chrono::seconds(3) );
  } /* while(true) */
} /* ftp process */
```

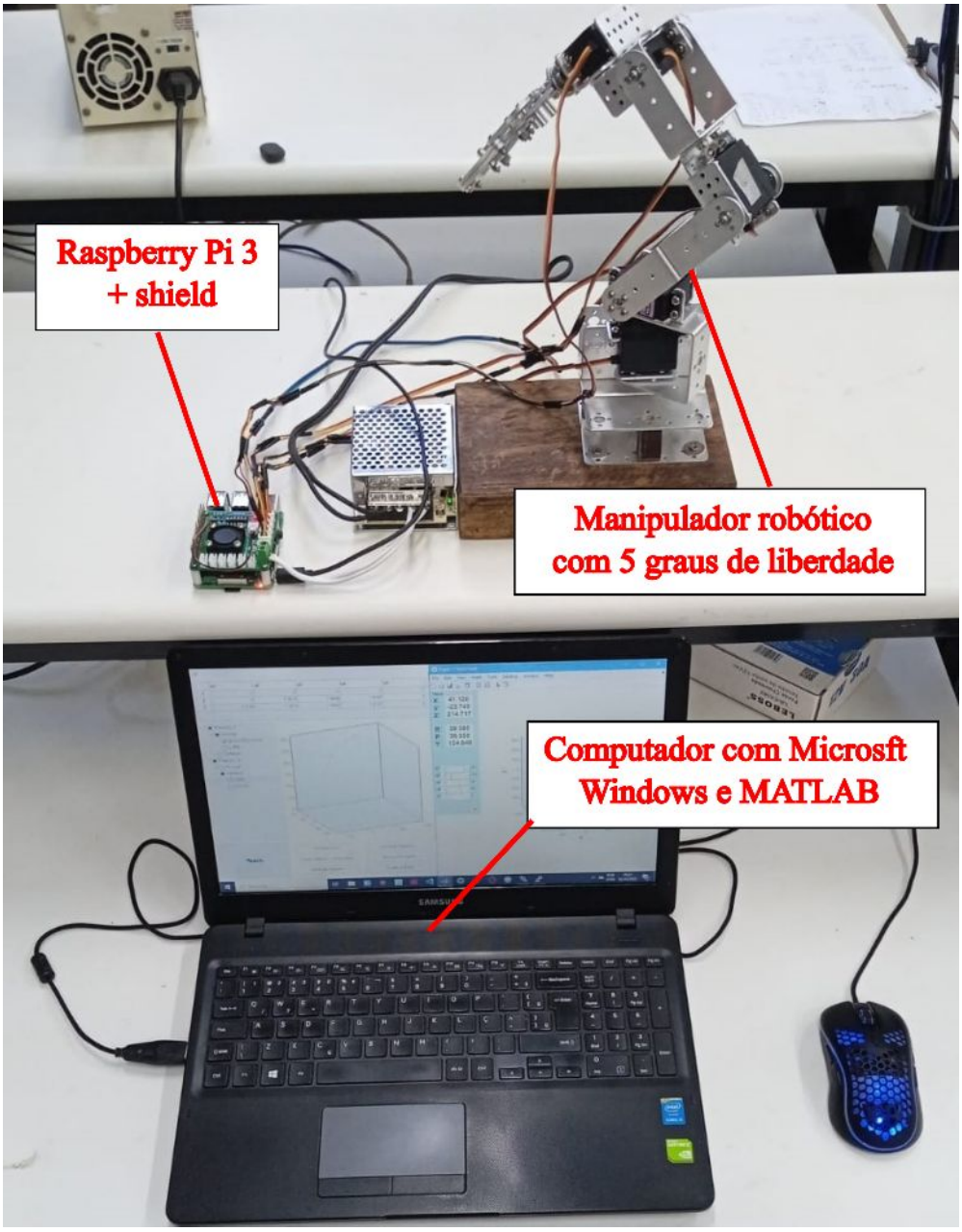
Fonte: autoria própria (2023)

Para que a aplicação tenha a capacidade de lidar com ambas as tarefas, de movimentação controlada pelo usuário e de monitoramento do servidor FTP, foram criadas duas *threads*. A primeira controla o robô pela interface TCP/IP. A segunda espera por notificações após *uploads* de arquivos de trajetória e executa os movimentos autônomos.

A tarefa responsável pela movimentação do braço via interface gráfica, exibida na Figura 14, inicialmente, espera pela conexão do cliente. Uma vez que a conexão é bem sucedida a tarefa entra em um laço infinito. Esse laço captura cada novo pacote do MATLAB, os quais contém um valor angular para cada articulação do braço. Esse pacote é então decodificado e salvo em um vetor de *floats* chamado *joints*. Uma vez salvo na

variável de tipo adequado, o código faz a escrita dos valores no barramento $I^2C$ para que o PCA9685 possa movimentar os servomotores. É importante destacar o uso de um *mutex*, que bloqueia o uso do objeto *servo_driver* por outras *tasks* do código.

Figura 16 – Montagem real do sistema proposto para ensinar, calcular e executar trajetórias.




Fonte: autoria própria (2023)

A tarefa voltada ao monitoramento do servidor FTP também é estruturada em um laço infinito. O primeiro passo executado por essa tarefa, antes de entrar no *loop*, é a criação de um *watch descriptor* (`wd`). Esse elemento irá criar um *link* entre um evento disparado pelo subsistema `inotify` e um descritor de arquivo (`fd`) criado como variável global no código. Ele é configurado de forma a disparar notificações para casos de criação

ou modificação do arquivos no servidor FTP. A implementação em C++ é mostrada na Figura 15.

Durante o laço, os eventos do descritor de arquivos são lidos e armazenados em um *buffer*. Para modificações ou criação do arquivo de trajetória armazenado no servidor FTP, a tarefa chama uma função que lê o arquivo `.csv` e escreve cada *frame* da trajetória para o driver do motor. Da mesma forma que na *task* de movimentação via *GUI*, essa tarefa também usa um mutex para bloquear o acesso ao `servo_driver`. Enquanto todo o arquivo de trajetória não for executado no robô, é impossível movimentar o manipulador pela interface gráfica.

Com o código de integração completo, foi possível montar fisicamente o sistema esquematizado na Figura 6. Essa montagem é exibida na Figura 16. Com tal arranjo é possível executar todo procedimento para criação, simulação e execução de trajetórias proposto por este trabalho.

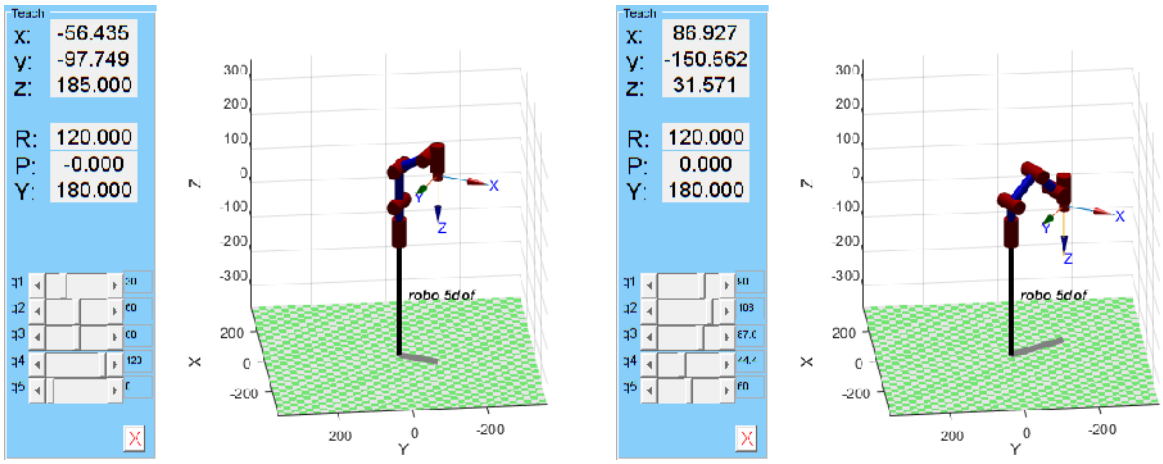
# 4 Resultados e Discussões

Antes mesmo da execução de uma trajetória gerada pela aplicação destinada a ensinar o robô, já é possível testar todos seus recursos gráficos. Dentre esses recursos estão a janela interativa de movimentação do robô, a visualização da prévia das trajetórias e os recursos de simulação.

Uma vez aberta a GUI é possível pressionar o botão *teach*, e assim que a conexão TCP/IP com a Raspberry Pi for bem sucedida, tanto o robô quanto sua representação gráfica podem ser movimentados. Em um teste, para exemplificar o uso da plataforma, o robô foi posicionado em pontos específicos. Ao ser posicionado em cada um dos pontos o botão *"Add path point"* foi pressionado na GUI para adicionar pontos ao percurso. A Figura 17 representa os dois pontos selecionados para o percurso na janela de movimentação.

Na barra lateral esquerda, aparece um menu que possibilitava a seleção do tipo de trajetória para cada trecho do percurso. O primeiro trecho corresponde à movimentação da posição zero do robô até o primeiro ponto adicionado ao percurso. Enquanto isso, a segunda trajetória é o caminho entre o primeiro ponto e o segundo ponto.

Figura 17 – Pontos selecionados para o percurso do manipulador robótico em que: A) é o ponto final da trajetória polinomial e B) ponto final da trajetória LSPB.



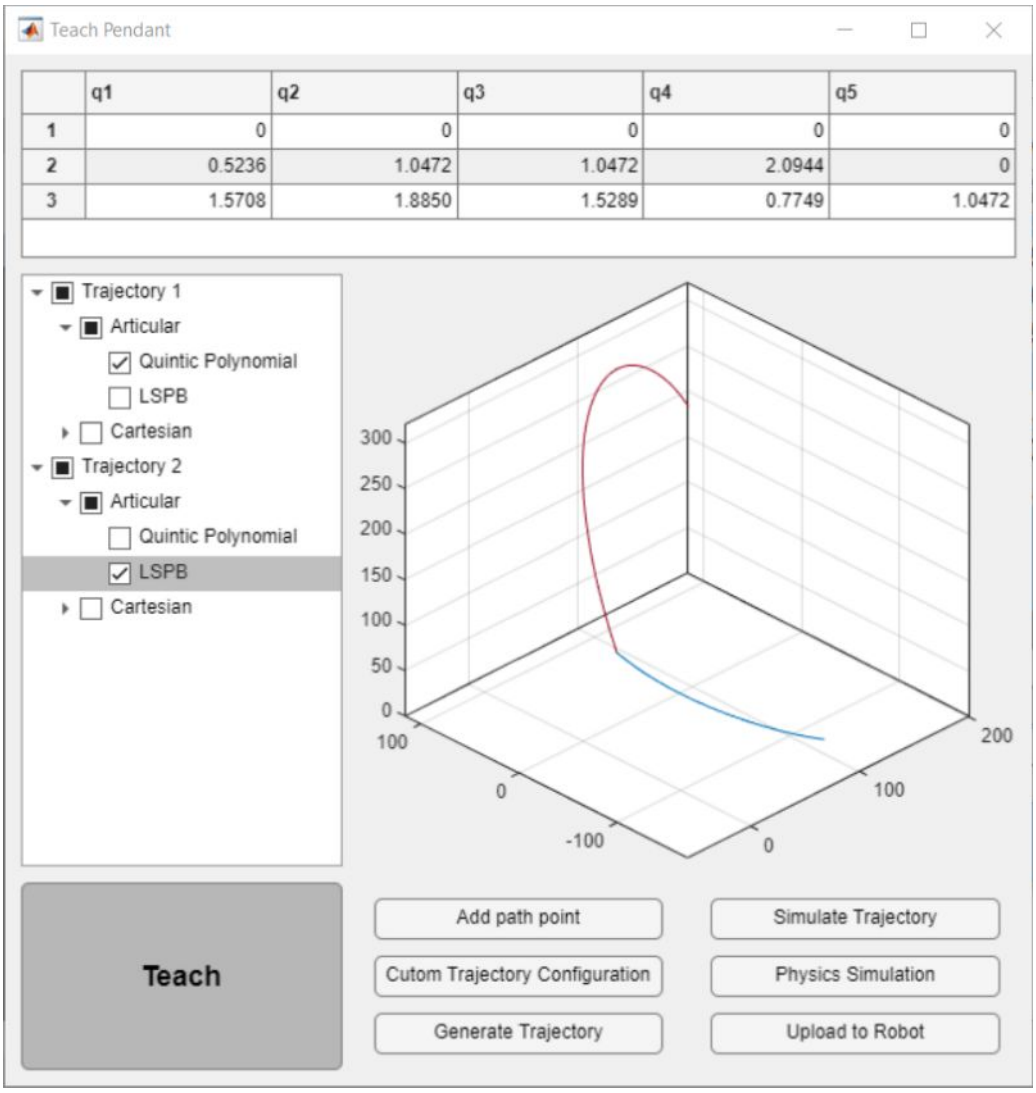

A B

Fonte: autoria própria (2023)

Para a primeira demonstração foram selecionadas as duas modalidades de algoritmo para criar trajetórias no espaço articular. Da posição zero ao primeiro ponto do percurso é usada uma trajetória criada pelo algoritmo de polinômios de quinta ordem. Para a segunda, é usado o algoritmo LSPB.

O botão *"Generate trajectory"* aplica os algoritmos selecionados para cada trecho

Figura 18 – *GUI* configurada e exibição de prévia da trajetória.



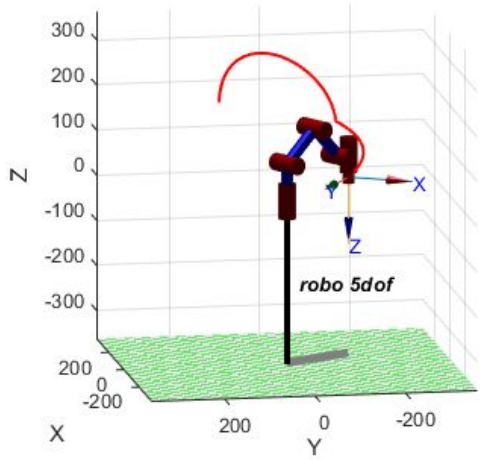

Fonte: autoria própria (2023)

e cria uma única trajetória, a qual é armazenada em um arquivo csv no computador com Microsoft Windows. Além disso também é plotada a prévia da trajetória na própria janela da interface do usuário. A GUI configurada para o exemplo descrito e a a prévia da trajetória gerada são ilustradas na Figura 18.

Uma vez criado o arquivo que armazena a trajetória, é possível usar os recursos de simulação. O primeiro deles é a simulação gráfica. Ela é iniciada pelo botão *"Simulate trajectory"*. Nesse tipo de simulação é aberta uma janela que mostra uma animação da representação tridimensional do robô executando a trajetória. Além disso, conforme o robô avança, traça-se no espaço a curva da trajetória por onde a ferramenta do robô passou. Um frame da simulação gráfica da trajetória exemplo é exibida na Figura 19.

Por fim, antes de uma execução real da trajetória é possível analisar o comportamento das grandezas cinemáticas articulares ao longo da trajetória. Esse recurso é obtido pressionando-se o botão *"Physics simulation"*. Esse botão, em trabalhos futuros poderá ser

Figura 19 – Simulação gráfica da trajetória.




Fonte: autoria própria (2023)

usado também para demais simulações da física do manipulador, como obtenção de curvas de conjugado exercido pelas juntas do robô.

Ao pressionar o botão *"Physics Simulation"*, uma janela com três gráficos em função do tempo é aberta. O primeiro conjunto de curvas exibe o ângulo em cada uma das juntas do braço robótico. O segundo gráfico exibe a velocidade angular em cada uma das juntas. Finalmente, o terceiro mostra a aceleração angular das articulações.

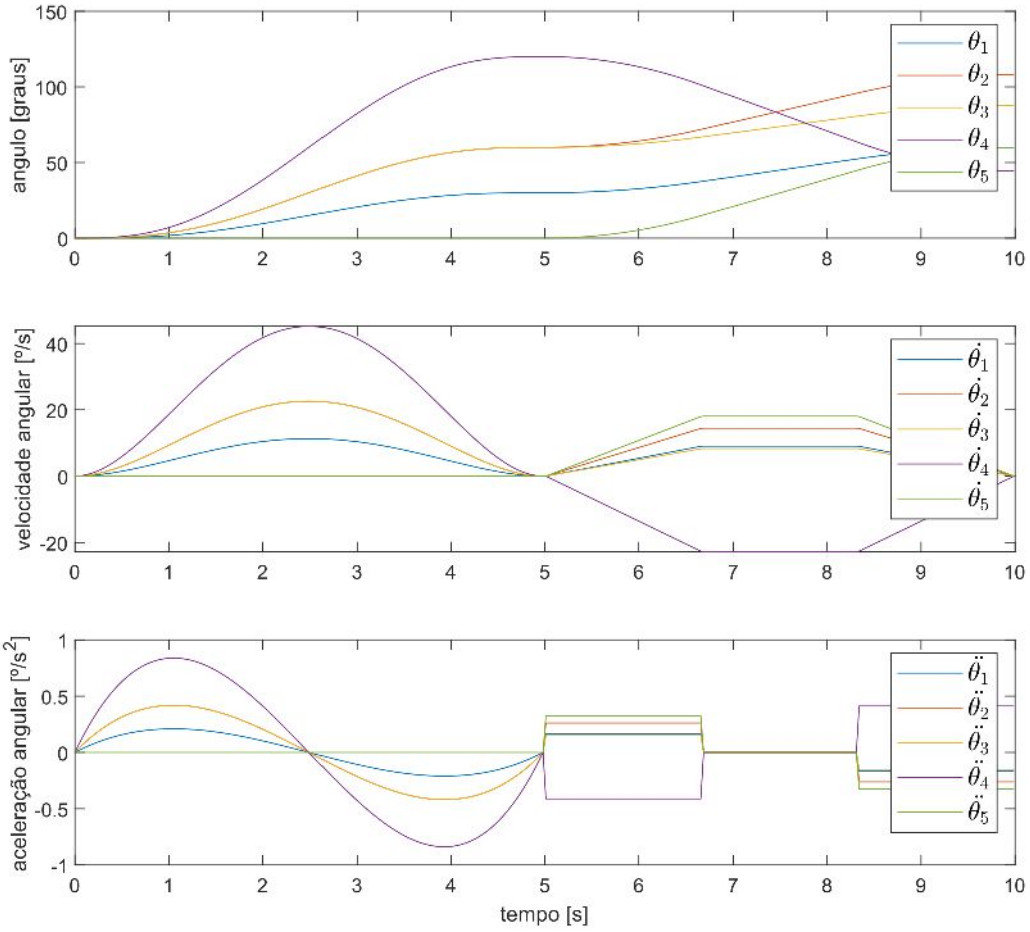
Como esperado, para a trajetória por polinômios de quinta ordem o perfil de todas as curvas foi suave. Esse padrão se repete para o ângulo, velocidade angular e aceleração angular. Isso é visualizado na primeira metade da trajetória apresentada na Figura 20, que equivale à porção da trajetória da posição zero ao primeiro ponto do percurso.

Durante a segunda metade do gráfico, após 5 segundos, teve-se início o trecho que utilizou LSPB. Para essa trajetória a velocidade angular apresentou o esperado perfil trapezoidal. A variação do angulo é vista como uma concatenação de parábolas ao redor de uma reta crescente. Já a aceleração angular assume valores constantes durante as rampas de velocidade, e valor nulo durante o período de velocidade constante.

O botão *"Upload to robot"*, quando pressionado, envia o arquivo de trajetória para o servidor FTP na Raspberry Pi. O *daemon* altera o estado da variável que monitora o sistema no diretório do servidor de arquivos. Nesse momento a *thread* responsável por monitorar os arquivos do servidor FTP, começa a executar a trajetória. O objeto que controla o chip PCA9685 fica bloqueado pelo mutex e a interface não mais é capaz de movimentar o robô até que toda trajetória seja realizada.

Para esse primeiro teste, no entanto, constou-se apenas que o servidor recebeu o arquivo de trajetória, como o esperado, por meio de *logs* no terminal da Raspberry Pi. Apesar do robô se movimentar, é difícil verificar visualmente com precisão se a ferramenta do robô segue com fidelidade as trajetórias no espaço gerada pelos algoritmos polinomial e LSPB. Pela análise visual, no entanto, é possível notar ligeiras e recorrentes vibrações da estrutura. Esse comportamento indesejado é consequência de grau de tolerância ruim das partes mecânicas dos atuadores.

Figura 20 – Simulação de grandezas angulares para trajetória no espaço articular do tipo polinomial e LSPB.




Fonte: autoria própria (2023)

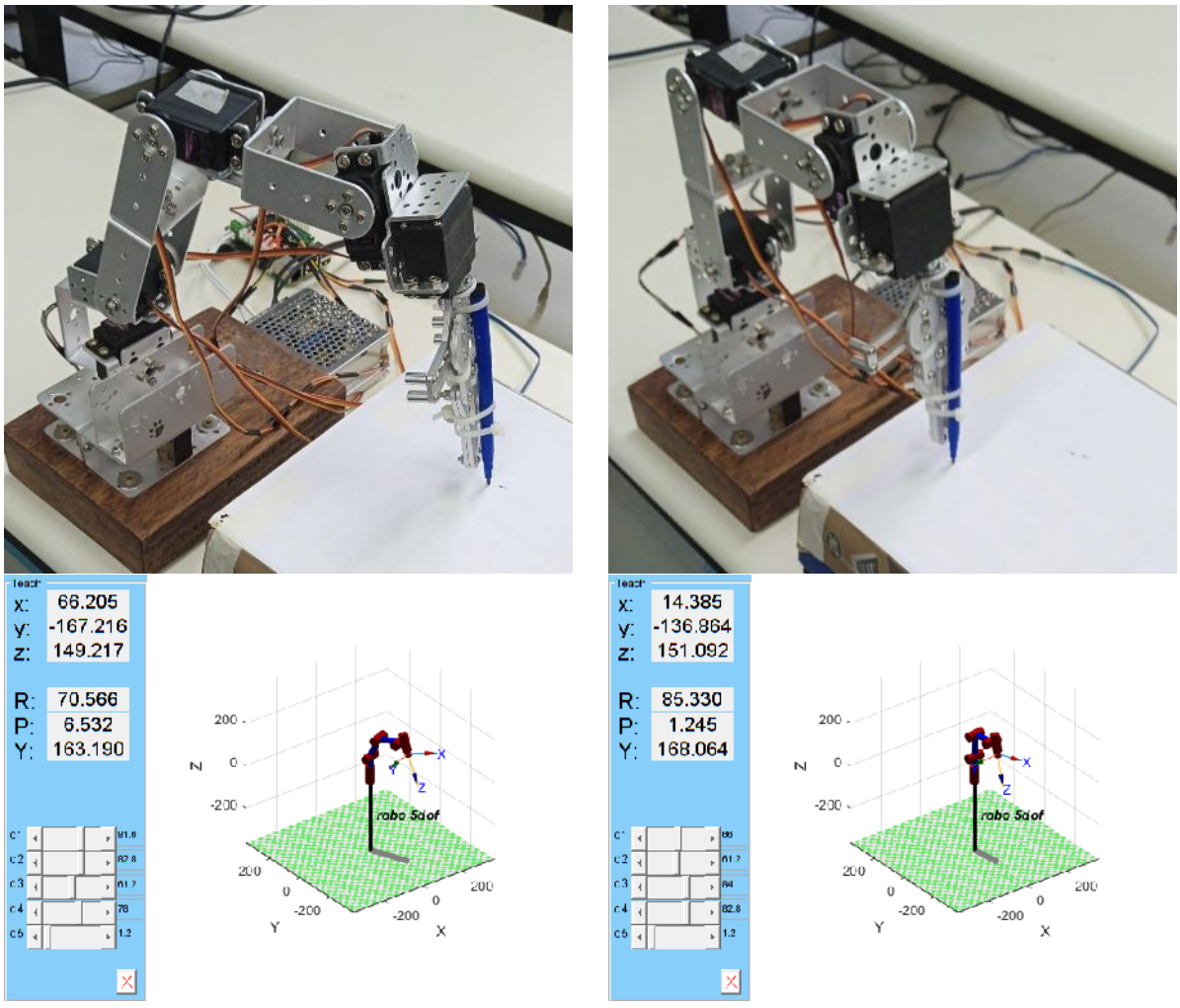
Apesar da integração se provar funcional, ela ainda não exprime qual o nível de acurácia do processo de execução da trajetória. Uma melhor forma de visualizar o quanto as trajetórias são fidedignas aos modelos simulados é pelo uso de trajetórias no espaço cartesiano. Isso porque com esse tipo de trajetória a ferramenta de robô segue uma curva no espaço com formato previamente conhecido.

Nesse sentido, o teste seguinte utiliza uma trajetória linear no espaço cartesiano. A

Figura 21 mostra as duas poses, na interface gráfica e no robô real, que representam o ponto de início e final da trajetória linear no espaço cartesiano.

Sendo assim, o robô sai de sua posição zero e vai até o primeiro ponto ilustrado na Figura 21.A por uma trajetória que usa o algoritmo LSPB. A trajetória que vai do ponto descrito em 21.A até 21.B descreve uma linha reta sobre a folha de papel. Para visualizar a trajetória da ferramenta no robô real, uma caneta hidrocor foi acoplada ao efetuador final.

Figura 21 – Pontos selecionados para: A) início e B) fim da trajetória linear.

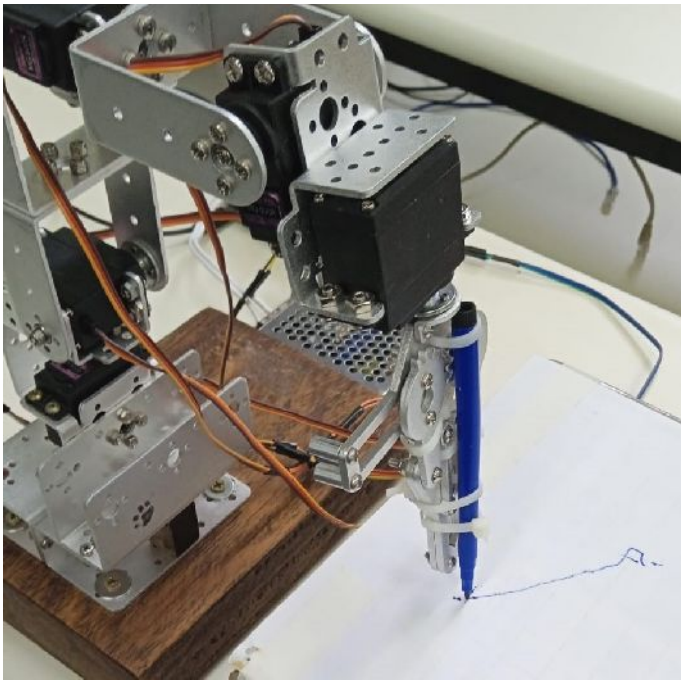



Fonte: autoria própria (2023)

Os resultados da trajetória real desempenhada pelo robô é demonstrada na Figura 22. Comparada à trajetória simulada, na Figura 23, a linha traçada possui ligeiros desvios causados pela vibração da estrutura durante sua movimentação. Esses movimentos irregulares, ocorrem por causa de lacunas entre peças móveis das caixas de engrenagem dos servomotores. Portanto, o erro visualizado é reflexo da folga presente na caixa de redução dos servomotores. Esse efeito poderia ser amenizado pelo uso de atuadores com maior firmeza articular, tais como servomotores que usam caixa de engrenagens planetária. Além

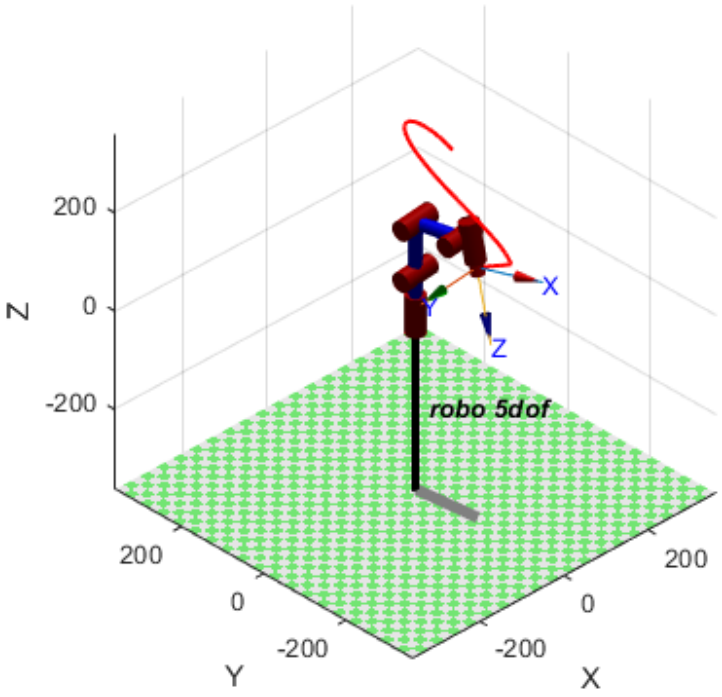
disso o projeto de uma estrutura que posiciona menos peso perto da ferramenta, poderia deixar o controle angular mais estável.

Figura 22 – Trajetória efetuada pela ferramenta do manipulador real.



Fonte: autoria própria (2023)

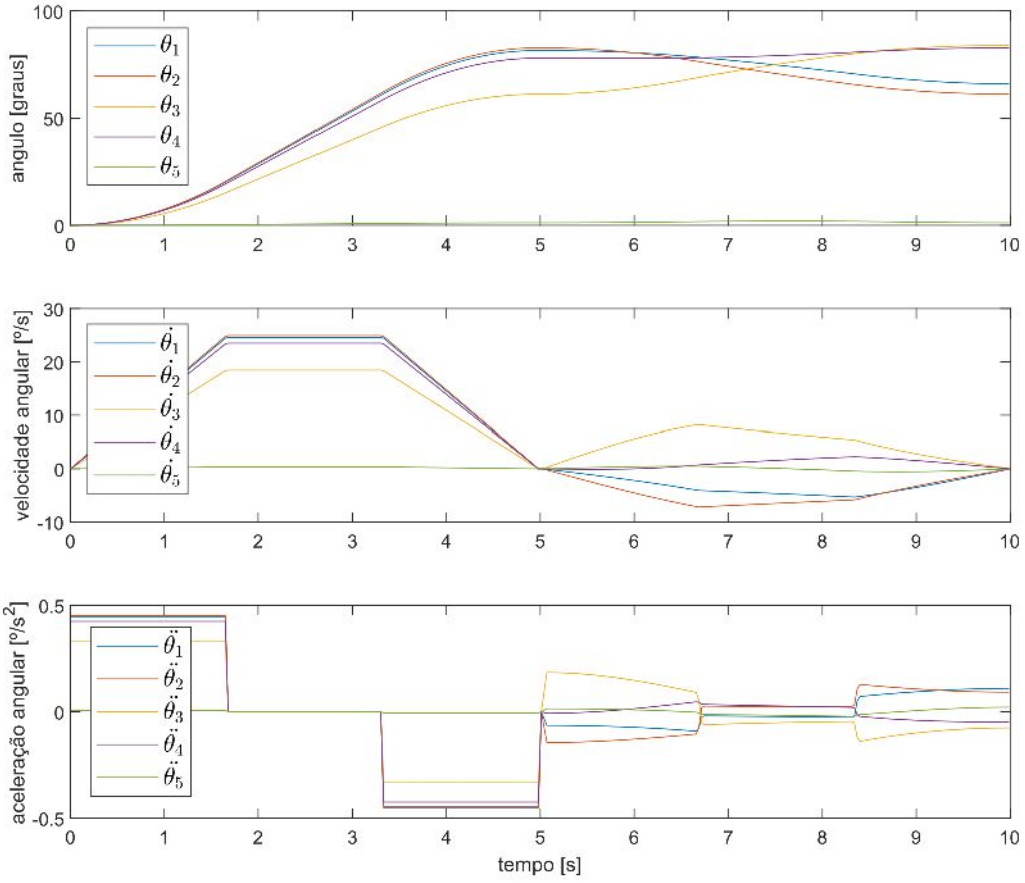
Figura 23 – Trajetória efetuada pela ferramenta do manipulador simulado.




Fonte: autoria própria (2023)

A Figura 24 mostra os gráficos das grandezas angulares do manipulador ao longo das duas trajetórias: a LSPB e a linear. Como o algoritmo traça a trajetória no espaço cartesiano, os perfis de velocidade e aceleração angular são desconhecidos até que a cinemática inversa seja aplicada. Nos algoritmos com tal metodologia, os perfis das curvas de ângulo, velocidade angular e aceleração angular são incertos e irregulares. Em alguns casos, essa imprevisibilidade da resposta angular pode criar trajetórias em que os atuadores sejam exigidos além de sua capacidade física em determinados trechos. Existe também o perigo que a trajetória no espaço cartesiano passe por alguma posição de singularidade, gerando picos de velocidade e aceleração em uma ou mais juntas.

Figura 24 – Simulacão de grandezas angulares para trajetórias LSPB e linear.




Fonte: autoria própria (2023)

# 5 Conclusão

Uma vez que o propósito deste trabalho era a emulação de um sistema de robótica industrial, como o descrito em Gadaleta (2018), usando um SBC com sistema Linux para o controle do robô, o objetivo central deste trabalho foi concretizado. Com a interface foi possível movimentar o robô de maneira intuitiva e configurá-lo para a realização de um variado número de trajetórias. Essas trajetórias planejadas e visualizadas na interface homem-máquina foram reproduzidas em um sistema real satisfatoriamente. Tudo isso, foi elaborado de forma que o robô pudesse ser programado ativamente para realização de tarefas autônomas mais triviais.

Além disso, o trabalho também foi relevante para o estudo de modelagem de robôs com o auxílio de plataformas plenamente difundidas no setor acadêmico, tais como o MATLAB e o Linux. Associando-se à metodologia algébrica mais tradicional, o emprego do MATLAB foi adequado para o estudo da robótica em seu estado da arte, com aplicação de cinemática e algoritmos de criação de trajetória. Uma vez parametrizado o robô com a notação de Denavit-Hartemberg, foi possível simular e analisar o comportamento do manipulador robótico para os algoritmos polinomial de 5ª ordem, LSPB e cartesiano linear.

Outro ponto positivo para o estudo da robótica é a contribuição deixada pelo trabalho para desenvolvimento de robôs controlados por sistemas Linux, tal como a biblioteca para o PCA9685. A biblioteca foi capaz criar uma camada de abstração para a interface com o chip gerador de sinais de controle, de forma que uma nova solução para geração de sinais de PWM estáveis em sistemas Linux foi proposta durante o trabalho. Complementando o *firmware* desenvolvido, foi projetado um *hardware* dedicado ao acionamento de servomotores, tais como os usados no robô prototipado. O trabalho acerca do *chip* PCA9685 pode ser retomado futuramente visando expandir a biblioteca para um pacote compatível com o ROS2.

O maior obstáculo de um projeto que visa emular robôs industriais provou ser a inadequação de componentes eletromecânicos de baixo custo para máquinas que se baseiam em movimentos precisos. A folga das caixas de redução dos servomotores bem como sua estreita banda de larguras de pulso decodificáveis prejudicou drasticamente a execução das trajetórias geradas pelos algoritmos estudados. Tais imperfeições mecânicas e eletrônicas fizeram com que a saída angular tivesse baixa precisão e que vibrações excessivas pudessem ocorrer durante a execução das trajetórias. Mesmo elevando o número de *frames* durante a aplicação do algoritmo de trajetórias e usando *drivers* eletrônicos de alta resolução (como PCA9685), não foi possível reproduzir a trajetória no sistema real de maneira fidedigna.

Por fim, para trabalhos futuros, a principal sugestão para aprimorar o sistema é

alterar o *design* mecânico a fim de criar uma estrutura de maior firmeza articular. Essas modificações incluem o uso de atuadores com melhor grau de tolerância e um projeto mecânico que repense a distribuição de massa na estrutura. Vale também ressaltar que um *teach pendant* voltado para um robô real possui uma série de recursos e configurações além daquelas fornecidas pela interface gráfica desenvolvida com o MATLAB. Nesse quesito, um dos trabalhos a serem desenvolvidos futuramente é conclusão da interface gráfica, adicionando mais recursos similares aos de *teach pendant* industrial.

# Referências Bibliográficas

BONEV, I. What are singularities in a six-axis robot arm? 2019. Disponível em: <https://www.mecademic.com/en/what-are-singularities-in-a-six-axis-robot-arm>. Acesso em: 28 de fevereiro de 2023. Citado na página 16.

CORKE, P. A robotics toolbox for matlab. *IEEE Robotics & Automation Magazine*, p. 24–32, 1996. Citado na página 12.

CORKE, P. *Robotics, Vision and Control - Fundamental Algorithms in MATLAB*. 2. ed. Berlim: Springer, 2017. Citado 3 vezes nas páginas 14, 15 e 20.

CORKE, P. *Robotics Toolbox for MATLAB®*. Release 10. [S.l.], 2020. 462 p. Citado 2 vezes nas páginas 20 e 33.

CORKE, P. Robotics toolbox. c2023. Disponível em: <https://petercorke.com/toolboxes/robotics-toolbox/>. Acesso em: 07 de março de 2023. Citado na página 13.

DENAVIT, J.; HARTENBERG, R. S. A kinematic notation for lower-pair mechanisms based on matrices. *ASME Journal of Applied Mechanics*, p. 215–221, 1955. Citado na página 15.

FANUC Ltd. How to know when a scara robot is the right choice for your application. 2018. Disponível em: <https://www.fanucamerica.com/news-resources/articles/how-to-know-when-a-scara-robot-is-the-right-choice-for-your-application>. Acesso em: 28 de fevereiro de 2023. Citado 2 vezes nas páginas 14 e 15.

FANUC Ltd. *FANUC M-710iC/70*. 2023. Disponível em: <https://www.fanucamerica.com/products/robots/series/m-710/m-710ic-70>. Citado na página 21.

FOROUZAN, B. A. *Data Communications and Networking*. 4th ed.. ed. New Delhi: McGraw Hill,, 2007. Citado na página 23.

Free Software Foundation. *GCC, the GNU Compiler Collection*. Disponível em: <https://gcc.gnu.org>. Acesso em: 31 de setembro de 2023. Citado na página 23.

Free Software Foundation. *GNU Make*. [S.l.], 2023. Version 4.1.1. Disponível em: <https://www.gnu.org/software/make/manual/make.html>. Acesso em: 31 de setembro de 2023. Citado na página 23.

GADALETA, M. *Engineering Methods and Tools for Energy-Efficient Industrial Robotics*. 36-37 p. Tese (Doutorado), 2018. Citado 2 vezes nas páginas 20 e 45.

LOVE, R. Kernel korner - intro to inotify. 2005. Disponível em: <https://www.linuxjournal.com/article/8478>. Acesso em: 07 de março de 2023. Citado na página 35.

MathWorks®. Develop uifigure-based apps. c2023. Disponível em: <https://www.mathworks.com/help/matlab/develop-apps-using-the-uifigure-function.html?s_tid=CRUX_lftnav>. Citado na página 33.

MOLLOY, D. *Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux*. Wiley, 2016. ISBN 978-1-119-1868-1. Disponível em: <http://www.exploringrpi.com/>. Citado na página 22.

NAZARI, S. Understanding i2c communication in linux: A beginner's guide with sample code for raspberry pi. 2023. Disponível em: <https://www.linkedin.com/pulse/understanding-i2c-communication-linux-beginners-guide-soheil-nazari/>. Citado na página 22.

NIKU, S. B. *Introdução à Robótica: Análise, Controle e Aplicações*. 2. ed. Rio de Janeiro: LTC, 2017. Citado 4 vezes nas páginas 16, 17, 18 e 19.

NXP Semiconductors. *16-channel, 12-bit PWM Fm+ I2C-bus LED controller*. [S.l.], 2015. Rev. 4. Citado na página 25.

NXP Semiconductors. *I2C-bus specification and user manual*. [S.l.], 2021. Rev. 7. Citado na página 22.

RODRIGUES, V. F. Jitter-free and portable hardware pwm solution for embedded linux devices. *XXI Conferência de Estudos em Engenharia Elétrica*, 2023. Citado na página 26.

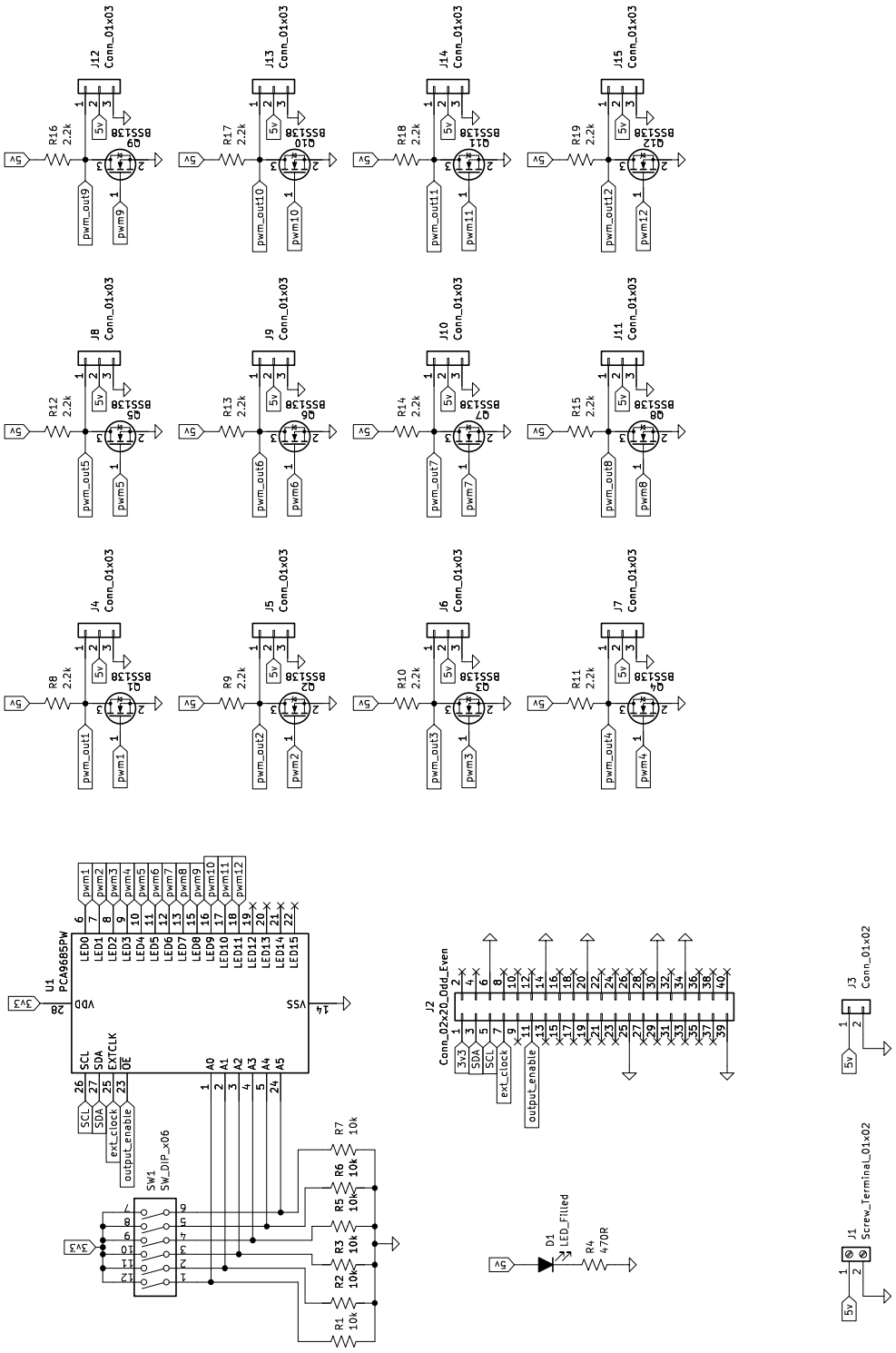
SINGH, G.; BANGA, V. K. Optimal trajectory planning analysis of robot manipulator using pso. *Resarch Square*, 2021. Citado na página 20.

Tower Pro. *MG996R High Torque Metal Gear Dual Ball Bearing Servo*. [S.l.], c2023. Citado na página 28.

XU, Z. et al. Trajectory planning with bezier curve in cartesian space for industrial gluing robot. *Lecture Notes in Computer Science*, p. 146–154, 2014. Citado na página 20.

# ANEXO A – Esquemático da *shield* para geração de sinais com PCA9685

Este anexo apresenta o circuito completo da placa para controle de servomotores.

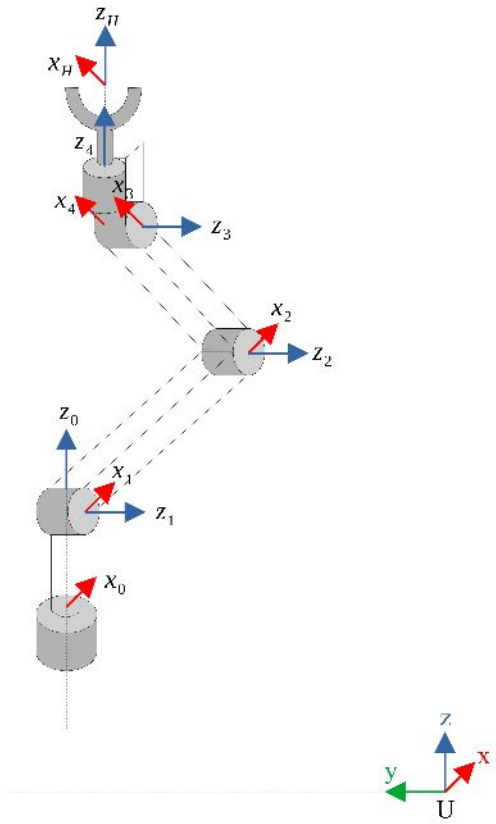


# ANEXO B – Cinemática Inversa Analítica para o Robô de 5 *DoF*

Para a dedução analítica da cinemática inversa do manipulador robótico, é necessário, primeiramente representá-lo de acordo com a notação de Denavit-Hartenberg e deduzir a cinemática direta. Para a representação matemática dos *frames* do robô, usa-se a Equação 2.1.

Como as equações cinemáticas podem ficar extensas, termos como $cos\theta_1$ serão abreviados para apenas $C_1$. Além disso expressões como $(\theta_2 + \theta_3 + \theta_4)$ serão reduzidas para $\theta_{234}$. Nesse sentido, funções trigonométricas com somas de arcos como $\cos(\theta_2 + \theta_3)$, será, em sua forma compactada, somente $C_{23}$.

Figura 25 – Representação do braço robótico de 5 *DoF* com os sistemas de referência de cada articulação.




Fonte: autoria própria (2023)

A matriz de transformação homogenia ${}^0T_1$ que representa a relação entre o *frame* {0} e o *frame* {1}, ou simplesmente $A_1$, de acordo com a fórmula 2.1 será:

$$A_1 = \begin{bmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{B.1}$$

De forma análoga, para a representação de $A_2$, ou ${}^1T_2$, também se dá pela aplicação da relação de Denavit-Hartenberg:

$$A_2 = \begin{bmatrix} C_2 & -S_2 & 0 & l_2C_2 \\ S_2 & C_2 & 0 & l_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{B.2}$$

Semelhantemente, todas as transformações restantes $A_3$, $A_4$, $A_5$ são deduzidas pela Equação 2.1:

$$A_3 = \begin{bmatrix} C_3 & -S_3 & 0 & l_3C_3 \\ S_3 & C_3 & 0 & l_3S_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{B.3}$$

$$A_4 = \begin{bmatrix} C_4 & 0 & -S_4 & l_4C_4 \\ S_4 & 0 & C_4 & l_4S_4 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{B.4}$$

$$A_5 = \begin{bmatrix} C_5 & -S_5 & 0 & 0 \\ S_5 & C_5 & 0 & 0 \\ 0 & 0 & 1 & l_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{B.5}$$

Por fim, para obter a cinemática direta basta multiplicar as transformações, como demonstrado na Equação 2.2:

$${}^0T_5 = \begin{bmatrix} C_{234}C_1C_5 - S_1S_5 & -C_5S_1 - C_{234}C_1S_5 & -S_{234}C_1 & C_1[L_3C_{13} + L_2C_2 + L_4C_{234} - L_5S_{234}] \\ C_1S_5 + C_{234}C_5S_1 & C_1C_5 - C_{234}S_1S_5 & -S_{234}S_1 & S_1[L_3C_{13} + L_2C_2 + L_4C_{234} - L_5S_{234}] \\ S_{234}C_5 & -S_{234}S_5 & C_{234} & L_1 + L_3S_{23} + L_2S_2 + L_5C_{234} + L_4S_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{B.6}$$

Para uma dada configuração articular, pode-se dizer que a ferramenta do manipulador possui uma posição e orientação tal que a transformação homogênea ${}^0T_5$ pode ser representada pela seguinte matriz:

$$ {}^0T_5 = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{B.7} $$

Durante o prosseguimento da dedução será convencionado que a Equação B.7 será o *LHS* (*Left-Hand Side*), e a Equação B.6 o *RHS* (*Right-Hand Side*).

O desenvolvimento da cinemática inversa, compreende encontrar o valor articular para um dado *frame*. Tendo-se um valor da ferramenta desejado (LHS), e a equação de cinemática direta (RHS), é possível deduzir alguns valores igualando-se tais termos,

$$ [LHS] = [RHS]. $$

Fazendo-se a razão dos elementos $a_{14}$ e $a_{24}$ das matrizes $[LHS]$ e $[RHS]$,

$$ \frac{p_y}{p_x} = \frac{S_1[L_3C_{23} + L_2C_2 + L_4C_{234} - L_5C_{234}]}{C_1[L_3C_{23} + L_2C_2 + L_4C_{234} - L_5C_{234}]}, $$

$$ \frac{p_y}{p_x} = \frac{\cos\theta_1}{\sin\theta_1}, $$

$$ \tan\theta_1 = \frac{p_y}{p_x}, $$

$$ \theta_1 = \tan^{-1}\left(\frac{p_y}{p_x}\right). \tag{B.8} $$

A Equação B.8 define o ângulo da "cintura"do manipulador robótico dado um *frame* desejado na ferramenta.

O próximo ângulo capaz de ser obtido da igualdade entre $[LHS]$ e $[RHS]$ é o ângulo $\theta_5$. Ele surge da razão dos elementos $a_{31}$ e $a_{32}$ das matrizes:

$$ \frac{o_z}{n_z} = \frac{-S_{234}S_5}{S_{234}C_5}. $$

Como seno é uma função par, pode-se remover o sinal da expressão.

$$ \frac{o_z}{n_z} = \frac{\sin\theta_5}{\cos\theta_5}, $$

$$ \tan\theta_5 = \frac{o_z}{n_z}, $$

$$\theta_5 = \tan^{-1}\left(\frac{o_z}{n_z}\right). \tag{B.9}$$

A partir desse ponto, não é possível obter mais ângulos articulares utilizando a igualdade entre $[LHS]$ e $[RHS]$. Para isso, serão necessárias manipulações algébricas que desacoplem ângulos no termo $[RHS]$. Uma alteração possível é multiplicar as matrizes definidas nas Equações B.7 e B.6 por $A_1^{-1}$. Isso irá remover senos e cossenos do ângulo $\theta_1$ que multipliquem outros termos angulares no $[RHS]$. Sendo assim,

$$A_1^{-1}[LHS] = \begin{bmatrix} n_xC_1 + n_yS_1 & o_xC_1 + o_yS_1 & a_xC_1 + a_yS_1 & p_xC_1 + p_yS_1 \\ n_z & o_z & a_z & p_z - L_1 \\ n_xS_1 - n_yC_1 & o_xS_1 - o_yC_1 & a_xS_1 - a_yC_1 & p_xS_1 - p_yC_1 \\ n_zL_1 & o_zL_1 & a_zL1 & p_zL_1 + 1 \end{bmatrix},$$

$$A_1^{-1}[RHS] = \begin{bmatrix} C_{234}C_5 & -C_{234}S_5 & -S_{234} & L_3C_{23} + L_2C_2 + L_4C_{234} - L_5S_{234} \\ S_{234}C_5 & -S_{234}S_5 & -C_{234} & L_3S_{23} + L_2S_2 + L_5C_{234} + L_4S_{234} \\ -S_5 & -C_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$A_1^{-1}[LHS] = A_1^{-1}[RHS] = A_2A_3A_4A_5. \tag{B.10}$$

Com essa manipulação foi possível remover multiplicações entre senos e cossenos que acoplavam variáveis articulares. Os elementos $a_{13}$ e $a_{23}$ da Igualdade B.10 permitem isolar o somatório de ângulos $\theta_{234}$:

$$\frac{-\sin\theta_{234}}{\cos\theta_{234}} = \frac{a_x\cos\theta_1 + a_y\sin\theta_1}{a_z},$$

$$\theta_{234} = \tan^{-1}\left(\frac{a_x\cos\theta_1 + a_y\sin\theta_1}{a_z}\right). \tag{B.11}$$

Esse somatório será útil para o cálculo de uma variável articular assim que mais dois outros ângulos sejam encontrados. Ele também será necessário porque os ângulos seguintes possuem em sua fórmula termos como o seno e o cosseno de $\theta_{234}$.

Dos elementos $a_{14}$ dos termos $A_1^{-1}[LHS]$ e $A_1^{-1}[RHS]$, consegue-se extrair a seguinte equação:

$$p_xC_1 + p_yS_1 = L_3C_{23} + L_2C_2 + L_4C_{234} - L_5S_{234},$$
$$p_xC_1 + p_yS_1 - L_4C_{234} + L_5S_{234} = L_3C_{23} + L_2C_2,$$

$$(p_xC_1 + p_yS_1 - L_4C_{234} + L_5S_{234})^2 = L_3^2C_{23}^2 + 2L_3C_{23}L_2C_2 + L_2^2C_2^2. \tag{B.12}$$

Da mesma maneira, os elementos $a_{24}$ dos termos $A_1^{-1}[LHS]$ e $A_1^{-1}[RHS]$ dão à equação:

$$pz - L_1 = L_3S_{23} + L_2S_2 + L_5C_{234} + L_4S_{234},$$
$$pz - L_1 - L_5C_{234} - L_4S_{234} = L_3S_{23} + L_2S_2,$$

$$(pz - L_1 - L_5C_{234} - L_4S_{234})^2 = L_3^2S_{23}^2 + 2L_3S_{23}L_2S_2 + L_2^2S_2^2. \tag{B.13}$$

Somando as Equações B.12 e B.13, obtém-se a expressão:

$$(pz - L_1 - L_5C_{234} - L_4S_{234})^2 + (p_xC_1 + p_yS_1 - L_4C_{234} + L_5S_{234})^2 = L_3^2S_{23}^2 + 2L_3S_{23}L_2S_2$$
$$+L_2^2S_2^2 + L_3^2C_{23}^2 + 2L_3C_{23}L_2C_2 + L_2^2C_2^2. \tag{B.14}$$

A fim de reduzir ainda mais o tamanho das expressões durante o desenvolvimento algébrico, o termo do lado esquerdo da Igualdade B.14 será chamado apenas de $A$. No lado direito da igualdade, os termos relativos à comprimento de elos podem ser colocados em evidência:

$$A = L_3^2(C_{23}^2 + S_{23}^2) + L_2^2(C_2^2 + S_2^2) + 2L_2L_3(C_{23}C_2 + S_{23}S_2).$$

Nos termos $C_{23}^2 + S_{23}^2$ e $C_2^2 + S_2^2$ é possível aplicar a relação fundamental da trigonometria,

$$A = L_3^2 + L_2^2 + 2L_2L_3(C_{23}C_2 + S_{23}S_2).$$

Expandindo a abreviação aplicada nos senos e cossenos, fica evidente que o termo $C_{23}C_2 + S_{23}S_2$ é equivalente à relação trigonométrica do cosseno da subtração de dois arcos:

$$A = L_3^2 + L_2^2 + 2L_2L_3(\cos(\theta_2 + \theta_3)\cos\theta_2 + \sin(\theta_2 + \theta_3)\sin\theta_2),$$
$$A = L_3^2 + L_2^2 + 2L_2L_3\cos(\theta_2 + \theta_3 - \theta_2),$$
$$A = L_3^2 + L_2^2 + 2L_2L_3\cos\theta_3,$$
$$\cos\theta_3 = \frac{A - L_3^2 - L_2^2}{2L_2L_3}.$$

Expandindo novamente o termo $A$, conforme a Igualdade B.14, chagamos à equação que representa o cosseno de $\theta_3$:

$$C_3 = \frac{(pz - L_1 - L_5C_{234} - L_4S_{234})^2 + (p_xC_1 + p_yS_1 - L_4C_{234} + L_5S_{234})^2 - L_3^2 - L_2^2}{2L_2L_3}. \tag{B.15}$$

Pela relação fundamental da trigonometria o seno de $\theta_3$ também pode ser dado por:

$$S_3 = \sqrt{1 - C_3}. \tag{B.16}$$

Finalmente, chega-se no valor do ângulo articular $\theta_3$ a partir do cálculo do arco tangente com os valores de B.16 e B.15:

$$\theta_3 = \tan^{-1}\left(\frac{\sin\theta_3}{\cos\theta_3}\right). \tag{B.17}$$

Para a dedução do $\theta_2$ também são manipulados os termos $a_{14}$ e $a_{24}$ das matrizes $A^{-1}[LHS]$ e $A^{-1}[RHS]$. O primeiro passo é desenvolver as somas de arco do seno e do cosseno de $(\theta_2 + \theta_3)$ na equação formada pelos elementos $a_{14}$:

$$\begin{aligned}
p_xC_1 + p_yS_1 &= L_3C_{23} + L_2C_2 + L_4C_{234} - L_5S_{234},\\
p_xC_1 + p_yS_1 &= L_3(C_2C_3 - S_2S_3) + L_2C_2 + L_4C_{234} - L5S_{234},\\
p_xC_1 + p_yS_1 &= (L_3C_3)C_2 - (L_3S_3)S_2 + L_2C_2 + L_4C_{234} - L5S_{234},
\end{aligned}$$

$$(L_3C_3 + L_2)C_2 - (L_3S_3)S_2 = p_xC_1 + p_yS_1 - L_4C_{234} + L_5S_{234}. \tag{B.18}$$

Pode-se fazer o mesmo com os elementos $a_{24}$ das matrizes:

$$\begin{aligned}
pz - L_1 &= L_3S_{23} + L_2S_2 + L_5C_{234} + L_4S_{234},\\
pz - L_1 &= L_3(S_2C_3 + S_3C_2) + L_2S_2 + L_5C_{234} + L_4S_{234},\\
pz - L_1 &= (L_3C_3)S_2 + (L_3S_3)C_2 + L_2S_2 + L_5C_{234} + L_4S_{234},
\end{aligned}$$

$$(L_3C_3)C_2 + (L_3S_3 + L_2)S_2 = pz - L_1 - L_4S_{234} - L_5C_{234}. \tag{B.19}$$

As Equações B.18 e B.19 formam um sistema de equações nas quais o seno $S_2$ e o cosseno $C_3$ são as variáveis. Isolando-se a variável $C_2$ da Equação B.18:

$$C_2 = \frac{p_xC_1 + p_yS_1 - L_4C_{234} + L_5S_{234} + (L_3C_3)S_2}{L_3C_3 + L_2}.$$

Substituindo $C_2$ na equação B.19:

$$(L_3S_3)\left(\frac{p_xC_1+p_yS_1-L_4C_{234}+L_5S_{234}+(L_3C_3)S_2}{L_3C_3+L_2}\right)+(L_3C_3+L_2)S_2=p_z-L_1-L_4S_{234}-L_5C_{234},$$

$$(L_3S_3)(p_xC_1+p_yS_1-L_4C_{234}+L_5S_{234})+(L_3S_3)^2S_2+(L_3C_3+L_2)^2S_2=$$
$$(pz-L_1-L_2S_2-L_5C_{234}-L_4S_{234})(L_3C_3+L_2),$$

$$\begin{aligned}[(L_3S_3)^2+(L_3C_3+L_2)^2]S_2=(pz-L_1-L_2S_2-L_5C_{234}-L_4S_{234})(L_3C_3+L_2)\\ -(L_3S_3)(p_xC_1+p_yS_1-L_4C_{234}+L_5S_{234}).\end{aligned} \tag{B.20}$$

Isolando a variável $S_2$ na equação B.18:

$$S_2=\frac{-(p_xC_1+p_yS_1-L_4C_{234}+L_5S_{234})+(L_3C_3+L_2)C_2}{(L_3S_3)}.$$

Substituindo $S_2$ em B.19:

$$(L_3S_3)C_2+(L_3C_3+L_2)\left(\frac{-(p_xC_1+p_yS_1-L_4C_{234}+L_5S_{234})+(L_3C_3+L_2)C_2}{(L_3S_3)}\right)=p_z-L_1-L_4S_{234}-L_5C_{234},$$

$$(L_3S_3)^2C_2-(L_3C_3+L_2)(p_xC_1+p_yS_1-L_4C_{234}+L_5S_{234})+(L_3C_3+L_2)^2C_2$$
$$=(pz-L_1-L_4S234-L_5C_234)(L_3S_3),$$

$$\begin{aligned}[(L_3S_3)^2+(L_3C_3+L_2)^2]C_2=(p_z-L_1-L_4S_{234}-L_5C_{234})(L_3S_3)\\ +(L_3C_3+L_2)(p_xC_1+p_yS_1-L_4C_{234}+L_5S_{234}).\end{aligned} \tag{B.21}$$

Fazendo a razão entre as Equações B.20 e B.21 obtemos a tangente de $\theta_2$. Logo, pode-se calcular o arco tangente desse termo para chegar à expressão que representa o ângulo $\theta_2$:

$$\theta_2=\tan^{-1}\left(\frac{(pz-L_1-L_5C_{234}-L_4S_{234})(L_3C_3+L_2)-(L_3S_3)(p_xC_1+p_yS_1-L_4C_{234}+L_5S_{234})}{(p_z-L_1-L_4S_{234}-L_5C_{234})(L_3S_3)+(L_3C_3+L_2)(p_xC_1+p_yS_1-L_4C_{234}+L_5S_{234})}\right). \tag{B.22}$$

Com os valores de $\theta_2$, $\theta_3$ e $\theta_{234}$ já conhecidos, calcula-se o valor de $\theta_4$,

$$\theta_4=\theta_{234}-\theta_2-\theta_4. \tag{B.23}$$

Essas equações ainda não consideram os *offsets* angulares nos ângulos $\theta_n$. *Offsets* foram adicionados às articulações da cintura, ombro e pulso do robô de forma a obter um *workspace* maior e com mais manipulabilidade. Além de um maior volume na área de trabalho, essa escolha de *offset* permitiu ao robô manipular com mais flexibilidade objetos colocados em um plano horizontal abaixo de sua base. Esses *offsets* devem ser subtraídos dos valores encontrados pelas equações de cinemática inversa. Para a cintura do robô foi adicionado um *offset* de $30^{\text{o}}$, logo:

$$\theta_1' = \theta_1 - \frac{\pi}{6}. \tag{B.24}$$

Para o ombro do manipulador também foi adicionado um *offset* de $30^{\text{o}}$:

$$\theta_2' = \theta_2 - \frac{\pi}{6}. \tag{B.25}$$

Para a articulação responsável pelo movimento de arfagem do pulso, foi adicionado um *offset* de -$90^{\text{o}}$:

$$\theta_4' = \theta_4 + \frac{\pi}{2}. \tag{B.26}$$

# ANEXO C – Script para visualização do envelope de trabalho do robô.

O *script* apresentado abaixo simula graficamente o envelope de trabalho do robô a partir de uma nuvem de pontos. É possível, dessa forma, mensurar os impactos no *workspace* do robô para dadas limitações dos atuadores ou *offsets* angulares adicionados às articulações.

```
clear all; close all;
clc;

L1 = 82;
L2 = 104;
L3 = 98;
L4 = 28;
L5 = 50;

E(1) = Link([0 L1 0  pi/2], 'standard');
E(2) = Link([0 0  L2 0],    'standard');
E(3) = Link([0 0  L3 0],    'standard');
E(4) = Link([0 0  L4  -pi/2],'standard');
E(5) = Link([0 L5 0  0],    'standard');

E(1).offset = pi/6;
E(2).offset = pi/6;
E(4).offset = -pi/2;

E(1).qlim = [0 120*pi/180];
E(2).qlim = [0 120*pi/180];
E(3).qlim = [0 120*pi/180];
E(4).qlim = [0 120*pi/180];
E(5).qlim = [0 120*pi/180];


robot = SerialLink(E, 'name', 'robo 5dof');

i = 0;
conv = pi/180; % cte para converter rad para graus

```

```
for q1=0 : 2*conv : 120*conv          % limites da junta 1
  for q2=0*conv : 20*conv : 120*conv % limites da junta 2
    for q3=0: 20*conv: 120*conv       % ... da junta 3
      for q4=0*conv : 20*conv : 120*conv % ... da junta 4
        for q5=0 : 20*conv : 120*conv   % ... da junta 5
            T01 = trotz(q1+pi/6)*transl(0,0,L1)*trotx(pi/2);
            %T12 = trotz(q2)*transl(L2,0,0); %[without offset]
            T12 = trotz(q2+pi/6)*transl(L2,0,0); %[with offset]
            T23 = trotz(q3)*transl(L3,0,0);
            T34 = trotz(q4-pi/2)*transl(L4,0,0)*trotx(-pi/2);
            T45 = trotz(q5)*transl(0,0,L5);
            T05 = T01*T12*T23*T34*T45;
            i = i+1;
            p = T05(1:3,4); % pega apenas a translacao
            p1(i) = p(1);    % transl em x
            p2(i) = p(2);    % ... em y
            p3(i) = p(3);    % ... em z
        end
      end
    end
  end
end

figure(1)
scatter1 = scatter3(p1,p2,p3,'r','filled');
scatter1.MarkerFaceAlpha = .2;
scatter1.SizeData = 10;
hold on
robot.teach()
```