TRABALHO DE GRADUAÇÃO

DESENVOLVIMENTO DE SOFTWARE DE SIMULAÇÃO E CONTROLE DE UM ROBÔ MANIPULADOR

Victor Hugo Marques Vieira

Brasília, Novembro de 2021

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

DESENVOLVIMENTO DE SOFTWARE DE SIMULAÇÃO E CONTROLE DE UM ROBÔ MANIPULADOR

Por,

Victor Hugo Marques Vieira

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro de Controle e Automação.

Banca Examinadora

Prof. Walter de Britto Vidal Filho, UnB/ ENM (Orientador) _____

Prof. Guilherme Caribé de Carvalho _____

Prof. José Maurício Santos Torres da Motta _____

Brasília, Novembro de 2021

FICHA CATALOGRÁFICA

VIEIRA, Victor Hugo Marques

DESENVOLVIMENTO DE SOFTWARE DE SIMULAÇÃO E CONTROLE DE UM ROBÔ MANIPULADOR.

[Brasília, Distrito Federal], 2021.

xii, 98p, 297 mm, (FT/UnB, Engenheiro, Controle e Automação, 2021). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1.Simulação	2.Robô Manipulador
3.Controle Robôs	4.Robôs Industriais
I. Mecatrônica/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

VIEIRA, Victor Hugo Maques, (2021). Desenvolvimento de software de Simulação e controle de um Robô Manipulador. Trabalho de Graduação em Engenharia de Controle e Automação, Faculdade de Tecnologia, Universidade de Brasília, Brasília – DF, 98p.

CESSÃO DE DIREITOS

AUTOR: Victor Hugo Marques Vieira

DESENVOLVIMENTO DE SOFTWARE DE SIMULAÇÃO E CONTROLE DE UM ROBÔ MANIPULADOR: Levantamento de dados a cerca de softwares de simulação e controle de robôs manipuladores, assim como desenvolvimento de software próprio.

GRAU: Engenheiro ANO: 2021

_____

VICTOR HUGO MARQUES VIEIRA
QD. 03, CONJ. A, SBIS – Núcleo Bandeirante.
71736-301 Brasília – DF – Brasil.

AGRADECIMENTOS

Poucos acontecimentos me deixaram tão feliz e ansioso quanto ao futuro, como o resultado do vestibular que me iniciou neste caminho: feliz pois era uma recompensa sobre todo o esforço gerado até aquele momento, e ansioso pois sabia que o esforço viria a ser ainda maior. Quero agradecer a todas as pessoas que estiveram ao meu lado durantes estes longos anos. A minha mãe Eliana que mostrou muita força encorajando um filho como eu. Aos meus tios, que foram o modelo que me mostrou ser possível tal jornada. A minha sobrinha Lara, que deu esperança de um futuro melhor.

Quero agradecer ainda aos amigos que conquistei durante esta jornada. Aos bons amigos formados no SAMU. Aos professores que muitas vezes apostaram na minha capacidade, mais do que eu mesmo apostaria.

Victor Hugo Marques Vieira

RESUMO

Realizou-se uma pesquisa das principais ferramentas relacionadas a simulação, e quando disponível, controle de robôs. Levantou-se características importantes para um software deste segmento, como auxílio para desenvolvimento do software objetivo deste trabalho. Para o desenvolvimento do software deste trabalho, escolheu-se um robô SCARA para ser base da apresentação gráfica. Levantou-se sobre ele as propriedades relacionadas a sua cinemática, e sua construção gráfica. Uma análise sobre as ferramentas passíveis para o desenvolvimento do software foi realizada e, levando em consideração os recursos disponíveis optou-se pelo desenvolvimento de todo o programa em Python, assim como a construção gráfica do robô desenvolvida no FreeCAD. Para o controle da simulação, pesquisou-se a respeito das linguagens de programação de robôs de forma offline, chegando a uma pseudolinguagem utilizado para o controle do robô, baseada na linguagem AML da IBM, dada as simplificações necessárias. O software final desenvolvido é capaz de simular a movimentação do robô de trabalho, utilizando uma interface com usuário, onde é capaz escrever o programa de simulação e observar a simulação do mesmo. Para o módulo de controle desenvolvido, são adaptados os sinais de posicionamento do robô simulado, para o controle do robô real, gerando a paridade de posicionamento necessária entre simulação e controle. Testes indicaram uma boa confiabilidade entre a simulação do sistema, e o controle do robô representado por um sistema de testes utilizados. O software desenvolvido atende aos requisitos iniciais, e é passível da sua utilização para a simulação e controle de robôs reais.

Palavras-chave: simulação, robô manipulador, controle robôs, robôs Industriais.

ABSTRACT

A survey of the main tools related to simulation, and when available, robot control was carried out. Important characteristics for a software of this segment were raised, as an aid for the development of the objective software of this work. For the software development of this work, a SCARA robot was chosen to be the basis of the graphic presentation. The properties related to its kinematics and its graphic construction were raised over it. An analysis of the possible tools for the software development was carried out and, taking into account the available resources, it was decided to develop the entire program in Python, as well as the graphic construction of the robot developed in FreeCAD. To control the simulation, research was carried out on offline programming languages for robots, arriving at a pseudolanguage used for robot control, based on the IBM AML language, given the necessary simplifications. The final software developed is capable of simulating the movement of the work robot, using a user interface, where it is able to write the simulation program and observe its simulation. For the control module developed, the positioning signals of the simulated robot are adapted for the control of the real robot, generating the necessary positioning parity between simulation and control. Tests indicated good reliability between system simulation and robot control represented by a test system used. The developed software meets the initial requirements, and can be used for the simulation and control of real robots.

Keywords: simulation, robotic manipulator, robot control, industrial robot.

SUMÁRIO

LISTA DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

J1	Junta 1 do robô.	[mm]
J2	Junta 2 do robô.	[mm]
J3	Junta 2 do robô.	[mm]
L1	Elo 1 do robô.	[mm]
L2	Elo 2 do robô.	[mm]
L3	Elo 3 do robô.	[mm]
L4	Elo 4 do robô.	[mm]
L5	Variável de saída da altura do robô.	[mm]
M1	Motor 1 do robô.	[kgfm]
M2	Motor 2 do robô.	[kgfm]
M3	Motor 3 do robô.	[kgfm]

Símbolos Gregos

α	Ganho que relaciona simulação e controle.	[rad/rad]
θ_1	Angulo de saída do atuador 1 do robô.	[rad]
θ_2	Angulo de saída do atuador 2 do robô.	[rad]
θ_R	Angulo do robô real.	[rad]
θ_S	Angulo do robô simulado.	[rad]

Siglas

GUI	Graphical User Interface

1 INTRODUÇÃO

A utilização de robôs manipuladores é uma ferramenta consolidada em várias indústrias. Atividades dos mais variados segmentos de produção podem ser beneficiados pela utilização de robôs manipuladores, usufruindo da sua capacidade de aperfeiçoar os recursos disponíveis.

Esta característica se dá pelo alto grau de competitividade do segmento industrial, e pela constante necessidade de diminuir a necessidade dos recursos disponíveis como forma de diferencial de mercado. Ações como a diminuição de perda, diminuição do tempo de produção por peça, maior velocidade de produção da linha, possibilidade de montagem de produtos mais complexos, são objetivos comuns para a aplicação de robôs manipuladores na indústria.

A aplicação deste recurso pode criar sistemas complexos, em que o funcionamento e adaptações tomam muito tempo e outros recursos. Ainda no contexto industrial é necessária muitas vezes a parada da linha de produção para adaptações do processo produtivo, ou das funções específicas do robô. Essa parada, mesmo que programada, implica em grandes custos financeiros, o que não é aceitável dado à necessidade já colocada de constante aumento de eficiência.

Uma solução aplicada para contornar, ou diminuir, os efeitos desta parada de linha, é a utilização de softwares de simulação da linha de produção. Com o auxílio de uma simulação, é possível estabelecer todos os parâmetros necessários para a atuação do robô na linha de produção, diminuindo o tempo de instalação da alteração em si. Mesmo que haja limitações quanto à aplicação desta ferramenta, a mesma tem-se tornado de grande relevância neste contexto [SOUSA, et all; 2018].

Diversas características da programação de um robô podem sofrer impacto positivo da aplicação de um simulador. Tais como:

- Otimização de trajetórias.
- Determinar o melhor ângulo de aproximação de um efetuador terminal.
- Estimar tempo de trabalho para processos complexos.
- Determinar parâmetros de resposta para situações de sinistro.

Este último item recebe ainda uma complexidade maior, pois apesar de ser altamente benéfico à simulação de cenários de acidentes, e determinação de parâmetros para situações não desejadas, há uma dificuldade em determinar o ambiente destas situações. Por outro lado, cenários já estabelecidos conseguem ser simulados sem necessariamente aplicar o risco da utilização no ambiente real.

Durante as pesquisas iniciais deste trabalho observou-se, entretanto, que apesar de ter disponíveis diversos softwares livres que simulam diferentes robôs, há poucos (ou nenhum) com a finalidade de simulação e controle de um robô real. A integração destas duas funções é benéfica, pois cria uma plataforma que possibilita a aplicação do ciclo completo de desenvolvimento de um processo robotizado: da determinação das trajetórias até o controle final do mesmo.

1.1 Contexto e Motivação

Dada a importância da utilização de simuladores no contexto descrito, é comum que junto aos softwares de controle de robôs comerciais, serem recebidos software de simulação. Há uma limitação na disponibilidade de softwares livres com esta capacidade, limitando a utilização de robôs não comerciais em aplicações práticas.

Dois pontos importantes a serem desenvolvidos neste trabalho:

- Desenvolver uma ferramenta que possibilite a simulação e o controle, simultâneo, de um robô manipulador.
- Desenvolver um software livre para uso direto ou adaptado de pequenas indústrias em sua cadeia produção.

No primeiro deve-se definir um modelo físico-matemático do robô de trabalho, e utilizar métodos de representação digital para simularmos, de forma preditivo, qual é a resposta do robô em um cenário real.

Para o segundo ponto, busca-se a integração entre a capacidade de simulação do robô de trabalho, com sua aplicação direta na sua tarefa de manufatura. Queremos desenvolver uma plataforma em que a simulação e controle tornem o software uma poderosa ferramenta de produtividade no conceito da programação de robôs manipuladores.

Estas duas finalidades são representadas neste trabalho, pois dois diferentes módulos com finalização específica, mas com fácil integração para realização do objetivo geral. Isso pois mesmo que a integração de ambos os módulos seja o objetivo principal, a confiabilidade do software também pode ser mensurado pelo funcionamento dos mesmos de forma individual.

1.2 Metodologia de Trabalho

O desenvolvimento deste trabalho se iniciou com uma revisão do estado da arte, referente a simuladores, seja para, robôs manipuladores tal como o escopo aqui apresentado, até simulações de equipamentos semelhantes. Os passos utilizados para o desenvolvimento apresentado podem ser classificados em:

Identificação do problema: identificação da problemática envolvida na aplicação de robôs com escopo industrial. Considerações a respeito desta dificuldade em simulações voltadas a aplicações industriais.

- Investigação: com o problema bem descrito, investigou-se a respeito das soluções de mercado já existentes. Os prós e contras associados a cada modelo são levantados, sendo um passo inicial para se estimar o diferencial necessário para a solução proposta.
- Idealização do modelo: com as necessidades do software de simulação bem descritas, pensou-se um modelo que as atenda, assim como ferramentas de construção capazes de tornar sua construção viável.
- Implementação do modelo: o modelo anteriormente idealizado é implementado com as ferramentas escolhidas, sendo levadas em consideração as adaptações necessárias.
- Validação: sucedida a implementação do modelo, testa-se o mesmo como solução possível para o problema inicialmente identificado.
- Conclusão: revisão do caminho percorrido e dos resultados.

2 CONCEITOS SOBRE ROBÓTICA

Neste capítulo são abordados os princípios necessários para o bom entendimento dos conceitos de robótica industrial, assim como a sua simulação e controle. São definidos pontos necessários para o bom funcionamento do software desenvolvido, assim como um levantamento de softwares similares.

A análise dos simulares análogos ao do presente trabalho, é apresentado para a sistematização das funções requeridas. Tanto para sua finalidade de otimização do processo de programação *off-line* de um robô, como as funções necessárias para sua simulação com maior confiabilidade.

2.1 Conceitos de robótica

A robótica surge como produto dos estudos relacionados a mecânica, eletrônica e computação, tendo aplicação específica de teoria de controle, microeletrônica, teoria de produção e outros. Este produto denominado robô, pode ter as mais diversas classificações, que podem vir de sua configuração física, quanto aplicação, quanto à cadeia cinemática, quanto aos tipos de atuadores presentes[CARRARA, 2015].

O escopo deste trabalho é voltado para robôs com aplicações industriais, ou seja, sistemas desenvolvidos com a finalidade realizar processos na industrial. Segundo Valdemir Carrara [CARRARA, 2015]: “Na terminologia aqui empregada, um robô será composto de um circuito eletrônico computadorizado de controle e um mecanismo articulado denominado manipulador”.

Neste contexto, um robô manipulador pode ser resumido ainda como um conjunto de elos, ligados por juntas móveis que se movimentam pelo controle de atuadores. Tem-se portanto uma extremidade fixa em uma superfície e uma extremidade móvel para realização de tarefas.

2.1.1 Anatomia e classificação de robôs industriais

Um robô manipulador comumente é associado a um braço robótico, pois sua estrutura é semelhante ao braço humano. O robô possui uma base fixa onde comumente se tem o ponto

de referência do sistema de coordenadas do robô, sendo esta base conectada por meio de juntas controláveis a elos de ligação. Na extremidade móvel do robô, têm-se um efetuador terminal, responsável pela finalização da atividade pretendida pelo robô. A figura 1 apresenta um esquemático desta estrutura:

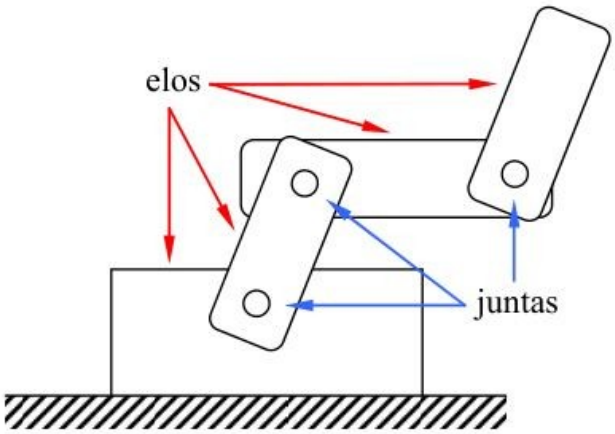

Figura 1: Esquema de notação de elso e juntas[CARRARA, 2015]

O efetuador terminal utilizado no robô comumente é adaptável, a depender da atividade empregada pelo robô, variando-se entre pinças, garras, ferramentas, pontos de solda, pistola para pintura, copos de sucção e outros. Atualmente o desenvolvimento de efetuadores de alto desempenho para funções específicas é uma importante área de pesquisa. A figura a seguir exemplifica um efetuador terminal com um conjunto de copos de sucção, utilizados reposicionamento de produtos em uma linha de produção:

Figura 2: Robô ABB IRB 120 com efetuador terminal de copos de sucção em uma linha de produção da Loreal[CARDWELL, BELANGER ; 2020]

As juntas dos robôs também apresentar grande variação, condicionada a sua aplicação específica, podendo ser uma junta linear, rotacional, esférica, cilíndrica, planar ou de parafuso. Estas se movimentar em uma, duas ou três direções, e a partir desta movimentação se defini os graus de liberdade desta junta e do robô em si[CARRARA, 2015].

Pela facilidade da aplicação e controle de motores em robôs manipuladores, juntas rotativas são as mais comuns nesta área de aplicação. De fato, há ainda uma subclassificação para as juntas deste tipo, baseada na relação de posicionamento de seus elos e relação ao movimento exercido pela junta[CARRARA, 2015]:

- Junta rotativa torcional (T): o elo anterior e posterior a junta mantém a mesma direção após o movimento, mudando o ângulo entre eles normal ao movimento.
- Junta rotativa rotacional (R): o eixo de rotação da junta é perpendicular ao elo anterior e posterior a junta.
- Junta rotativa revolvente (V): os elos são perpendicular, e a junta é o intermediário desta rotação.

A junta prismática é outra importante para robôs manipuladores. Isso pois movimentações lineares com este tipo de junta são de simples funcionamento e controle, sendo utilizado sempre que possível. A figura 3 apresenta um esquemático da junta linear e das juntas rotacionais, por serem as mais frequentes:

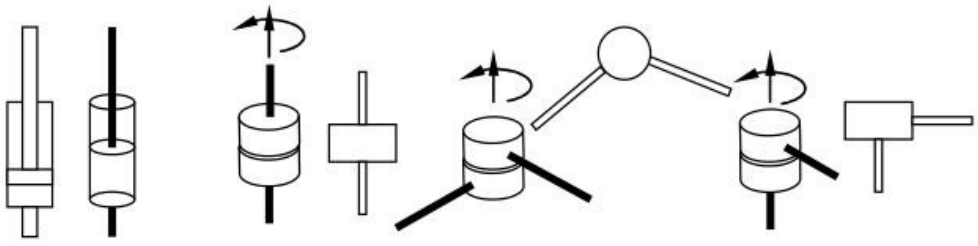

Figura 3: Esquemático das juntas mais frequentes[CARRARA, 2015]

A partir dessas definições de juntas, há à possibilidade de se tipicar os diferentes tipos de robôs industriais. Esta disponibilidade é o que define o espaço que o robô pode atuar, sendo este espaço definido como volume de trabalho, e a partir deste se tem a principal classificação de configuração dos robôs industriais[ROSÁRIO, 2010]:

- Configuração Cartesiana (LLL): robô constituído por três juntas lineares, geralmente presas em eixos deslizantes. A descrição do volume de trabalho é dada preferencialmente em coordenadas cartesianas.
- Configuração Cilíndrica (LVL): robô possui duas juntas lineares separadas por uma junta revolvente, construindo assim um volume de trabalho cilíndrico. Coordenadas cilíndricas são utilizadas para descrever este volume de trabalho.
- Configuração Articulada (TRR): a primeira junta deste robô é torcional, possibilitando que o braço seja direcionado em qualquer direção. As juntas rotacionais que constroem o braço em si tornam possível que o braço chegue aos pontos de seu volume de trabalho de vários ângulos possíveis, sendo preferível em ambientes de trabalho complexos. Seu volume de trabalho é o espaço intermediário entre duas esferas.
- Configuração Esférica (TRL): para este robô é possível construir um volume de trabalho esférico, com a limitação do menor tamanho apresentável para a junta linear (L).
- Configuração SCARA (RRL): o robô utilizado para o desenvolvimento deste trabalho, possui duas juntas rotativas a partir de sua base, e uma terceira junta linear, utilizada para aproximação do efetuador terminal a um ponto de trabalho.

Estas classificações podem ser utilizadas de modo genérico, mas é possível que robôs com esta mesma disposição de juntas sejam utilizados para a construção de um volume de trabalho diferente. A ordem de precedência destas juntas também é determinante para este volume de trabalho.

Nesta classificação tem-se uma separação entre os graus de liberdade do braço do robô, e o punho. Isso pois o punho pode ser alterado a depender do efetuador terminal utilizado, o que não necessariamente muda a configuração física do robô. Entretanto o punho também possui uma classificação baseada no tipo de junta que ele representa, ou seja, em seus graus de liberdade.

2.1.2 Ferramentas de análise da robótica

O conjunto de pontos que um robô é capaz de alcançar utilizando seus atuadores é dito como seu volume de trabalho. Este volume é o limitante para atuação final do robô em sua função na indústria. Utiliza-se portante de métodos matemáticos para relacionar as variáveis de controle dos atuadores do robô, para o posicionamento do seu efetuador terminal em seus pontos de trabalho. Duas análises são importantes para iniciar o entendimento da movimentação da extremidade móvel deste robô manipulador: cinemática e dinâmica[FERREIRA, 1999].

A cinemática é responsável por uma análise da movimentação do robô, sendo principalmente a descrição de sua extremidade móvel. Usualmente uma associação matemática, entre um ponto de referência fixa, como a base fixa do robô, até o punho do robô, que é uma referência móvel. O robô pode ser interpretado deste modo, como um sistema de transformação de posicionamento. Esta relação é dependente do posicionamento das juntas do robô, sendo estes valores denominados variáveis de junta.

Dois tipos de análise cinemática podem ser realizados, a cinemática direta e a cinemática inversa[FERREIRA, 1999]:

- Cinemática direta: a relação geométrica entre as variáveis de junta e os elos do robô, levam de forma única até um ponto específico do volume de trabalho. As variáveis de junta são definidos, e procura-se a localização da extremidade móvel do robô.
- Cinemática inversa: com um ponto específico de trabalho bem determinado, procura-se uma relação de variáveis de junta que levem a extremidade móvel do robô a este ponto. A depender da configuração do robô, pode não haver solução única, como em robôs articulados por exemplo.

A utilização destas duas análises são utilizadas intercaladamente para a construção de trajetória de trabalho do robô, determinando tanto seus pontos de interesse como o posicionamento dos atuadores para chegar a estes pontos.

A dinâmica do robô, relaciona suas variáveis cinemáticas, como posicionamento e velocidade, as forças realizadas pela extremidade móvel. Consequentemente, há uma associação para com as forças e binários realizados pelos efetuadores nas juntas do robô, para possibilitar a cinemática esperada[FERREIRA, 1999].

De modo análogo à cinemática, a dinâmica também pode ser direta ou inversa, a depender das necessidades esperadas pelo controlador. Pela presença de efeitos reais, como atrito, folga das juntas, deformação dos elos e juntas decorrentes de altas cargas, há uma complexificação da análise[FERREIRA, 1999].

2.2 Simuladores

Um dos diferenciais competitivos em uma indústria, é a eficiência com que a mesma consegue realizar seus processos de produção. Uma eficiência maior possibilita que se oferte produtos a preços inferiores, ou se pratique uma taxa de lucro superior, sendo ambos os cenários atraentes do ponto de vista econômico.

Na indústria, a implantação da automação de processos pode se tornar complexa. O uso de diversos equipamentos como motores, cilindros, equipamentos robóticos, esteiras, sensores e etc, fazem com que esta automação tenha um desenvolvimento pleno dificultado. Sua implantação como um todo, demanda muito tempo esforço, assim como qualquer alteração proveniente de erros também se torna custosa, diminuindo a eficiência buscada para o projeto. Tais erros podem ser dos mais variados tipos, causando danos a operadores do sistema, ou aos equipamentos em si. Uma forma de lidar com esta complexidade, é a construção de simuladores dos equipamentos reais, para que seja possível se antecipar aos eventos diversos [PONTES, 2012].

Estes simuladores criam projeções do comportamento do equipamento, a partir de uma construção virtual de suas características físicas. Para isso é utilizado de modelos que descrevem a realidade em um “mundo virtual”, já que também há à necessidade de se construir a características de contorno do ambiente de trabalho deste equipamento [CARRASQUINHO, 2015].

Neste ambiente virtual é possível se avaliar, com custos mínimos, qualquer alteração na linha de produção. Cenários de sinistro podem ser experimentados sem risco algum, sendo possível pensar uma resposta cabível, e da mesma forma, testar a nova resiliência do sistema após esta adaptação. Acidentes provocados por falha humana também podem ser simulados, levando em consideração as limitações do sistema, ou capacidade de criar todos os cenários possíveis.

No âmbito deste trabalho, é analisado de forma preferencial simuladores voltados para robôs manipuladores. Estes aplicam todo o conceito de análise para robôs voltados para manufatura industrial, logo se tem vários softwares voltados para diferentes robôs, com aplicação em diferentes indústrias. A interação do robô com os objetos de seu trabalho é tão relevante quanto a própria trajetória do robô, sendo uma das especificações que difere a aplicação da simulação[CARRASQUINHO, 2015].

Para o desenvolvimento do simulador e controlador proposto neste trabalho, foi analisado alguns softwares com finalidade parecida. A aplicação específica destes podem ser diferente da aplicação aqui proposta, entretanto algumas das características destes, são relevantes para o escopo deste trabalho, sendo estas discutidas posteriormente. Alguns destes softwares são de uso pago, sendo aqui analisado somente a versão de testes do mesmo.

2.2.1 RoboAnalyzer

Software de simulação desenvolvido no Departamento de Engenharia Mecânica (*Mechanical Engineering Department*) do Instituto Indiano de Tecnologia de Delhi (*Indian Institute of Technology Delhi*). Este software possui a finalidade de auxiliar a geração de trajetórias de diversas configurações de robôs, para finalidades educacionais, industriais, desenvolvimento, etc [SAHA, Subir Kumar].

Este software tem um histórico de desenvolvimento iniciado em 1996, e desde lá, várias funcionalidades foram implementadas, e seus modelos matemáticos otimizados. Atualmente é possível acompanhar e gerar trajetórias para robôs com até 7 eixos, que podem ser escolhidos entre eixos lineares e rotacionais. Utilizando o robô construído com estes eixo, o software permite mudar suas características físicas utilizando a notação de DH, para elos fixos e ângulos não variáveis. A imagem 4 mostra a interface deste software para um robô rotacional com dois eixos.

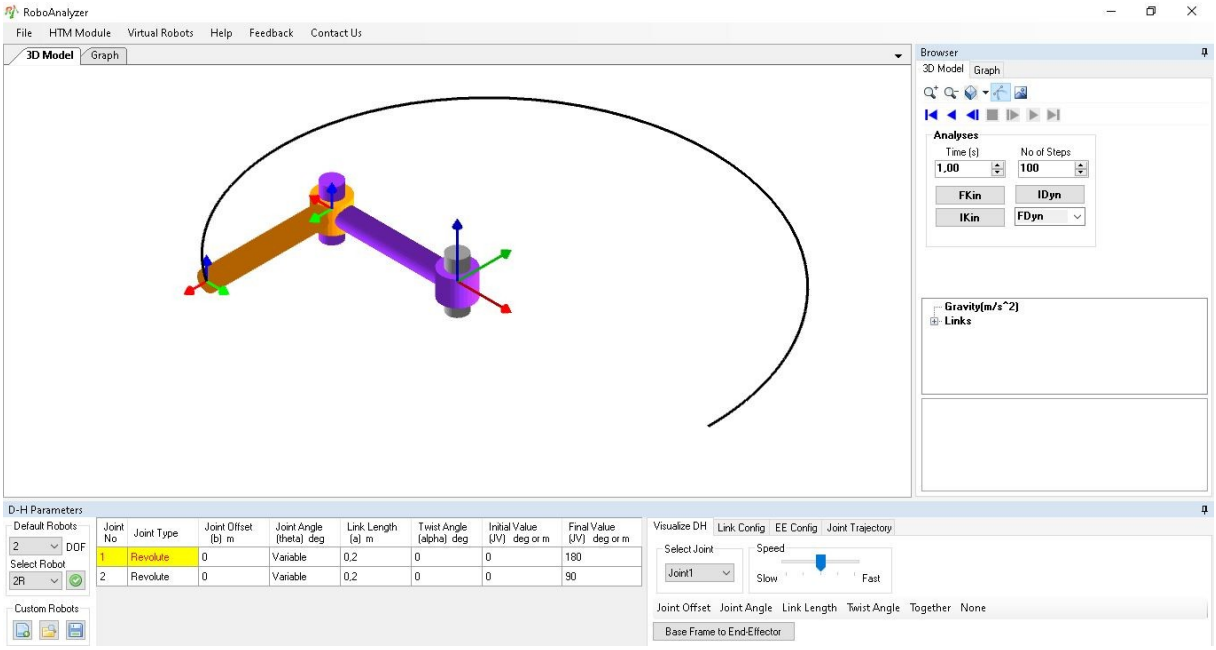

Figura 4: Interface de simulação do Robot Analyzer para robô com dois eixos rotacionais

Duas características importantes destes simulador, é a possibilidade de se controlar a velocidade de movimentação de seus atuadores por meio do tempo, e os passos dados pelos atuadores. Essas funções possibilitam uma maior integração entre a simulação visualizada no computador, e a movimentação real do robô.

Há ainda a possibilidade de se analisar de forma gráfica, o posicionamento dos elos em relação aos eixos de referência durante movimentação, assim como o valor de mudança dos atuadores em relação ao tempo. Essas propriedades possibilitam criar trajetórias complexas no robô real. A figura 5 mostra os valores dos atuadores observados para a movimentação apresentada na figura 4.

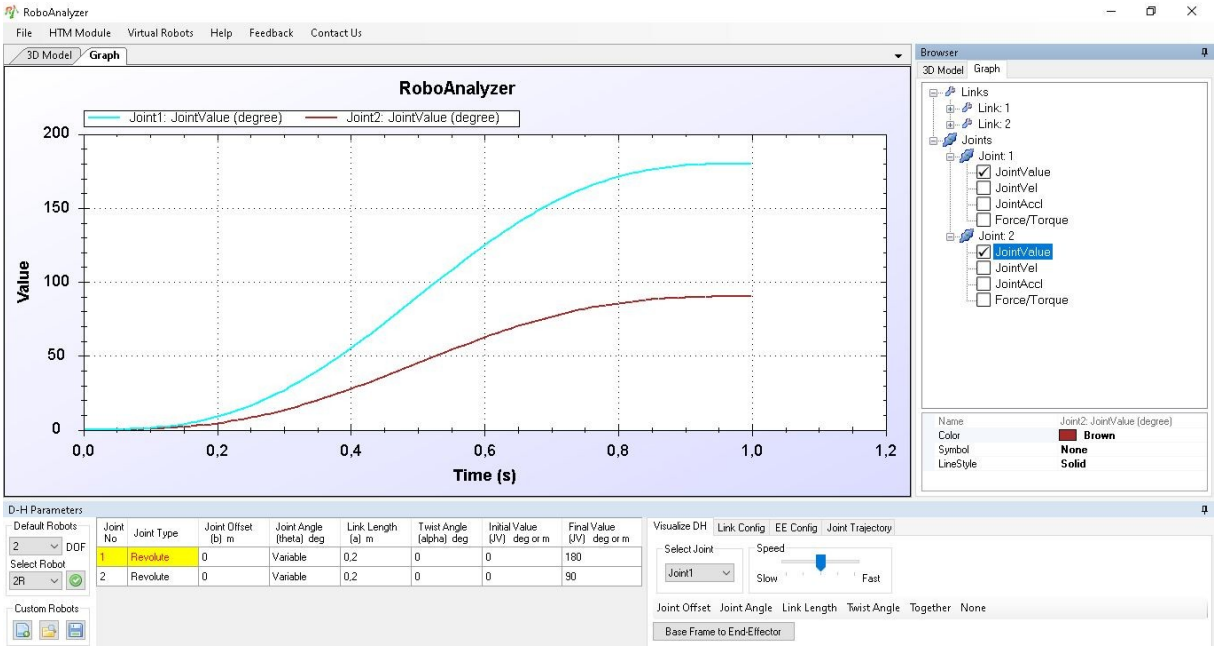

Figura 5: Valores dos atuadores para um tempo de trajetória

Cabe salientar que para este software específico é possível o acréscimo de mais juntas ou elos a depender da congruência do sistema requerido. Desta forma o software pode simular diferentes robôs, a depender da capacidade do usuário em definir estes modelos.

2.2.2 RoboSim

Software desenvolvido pela estadunidense UC Davis C-Stem, com finalidade de simulação e controle de robôs para aplicação educacional. Permite-se tanto a simulação dos funcionamento do robô utilizado e calculo de trajetória, como o controle de robôs de Lego (*Lego Mindstorms*). Software inicialmente desenvolvido C/C++ realizou uma migração de sua plataforma para Java buscando uma maior compatibilidade com múltiplas plataformas [C-STEM, 2020].

A figura a seguir (Figura 6) apresenta um exemplo da interface utilizada pelo programa para as finalidades dispostas:

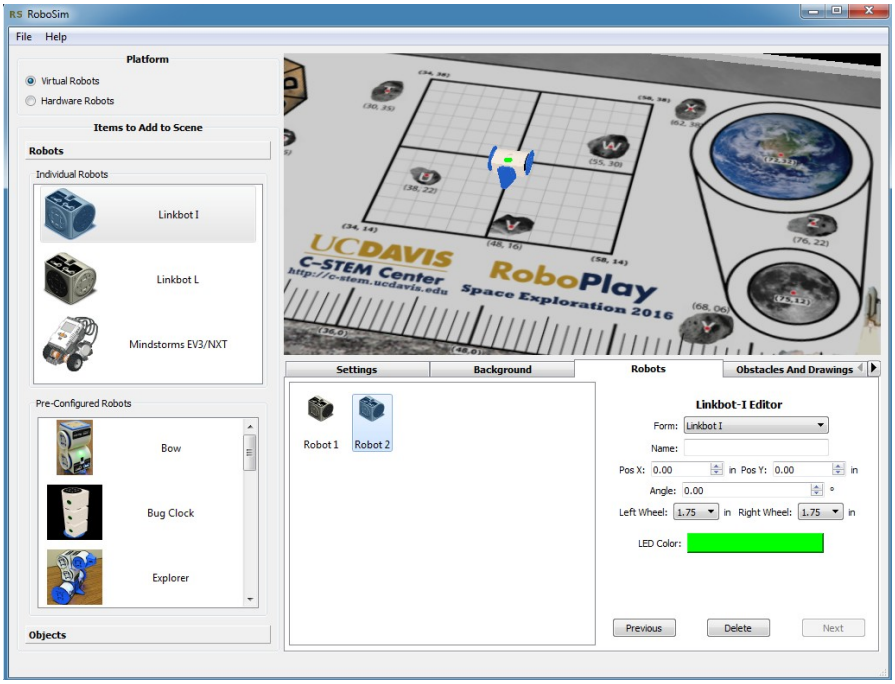

Figura 6: Interface do RoboSim[C-STEM, 2020]

A figura 7 apresenta um exemplo de trajetória complexa criada com dois robôs:

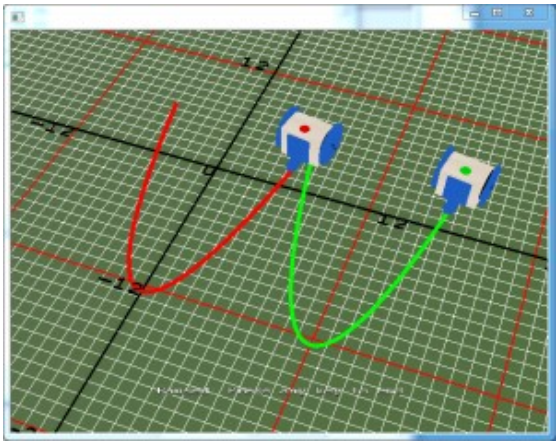

Figura 7: Exemplo de trajetória criada com RoboSim[C-STEM, 2020]

2.2.3 RobotStudio

A plataforma mais completa atualmente para a simulação de robôs, sendo desenvolvida pela líder de mercado neste seguimento ABB (Asea Brown Boveri)

[CARRASQUINHO, 2015], com a finalidade de auxiliar na programação e controle dos robôs desenvolvidas pela mesma. Todas as configurações e funcionamento de equipamento são análogos ao esperado do equipamento real, possibilitando uma programação *offline* da linha com maior eficiência[ABB, 2008].

O software possibilita uma criação de ambiente complexa, de forma a representar o ambiente de trabalho do robô, além da criação de objetos análogos com o de trabalho do mesmo. Há ainda a função de otimização de trabalho, que otimiza o funcionamento da programação imputada no mesmo.

A linguagem utilizada para a programação do robô é a RAPID. Linguagem de alto nível também desenvolvida pela ABB e introduzida para o S4 *control system,* no ano de 1994, por isso tendo muito tempo de mercado[ABB, 2008]. A figura 8 apresenta a interface do programa para o robô IRB 120.

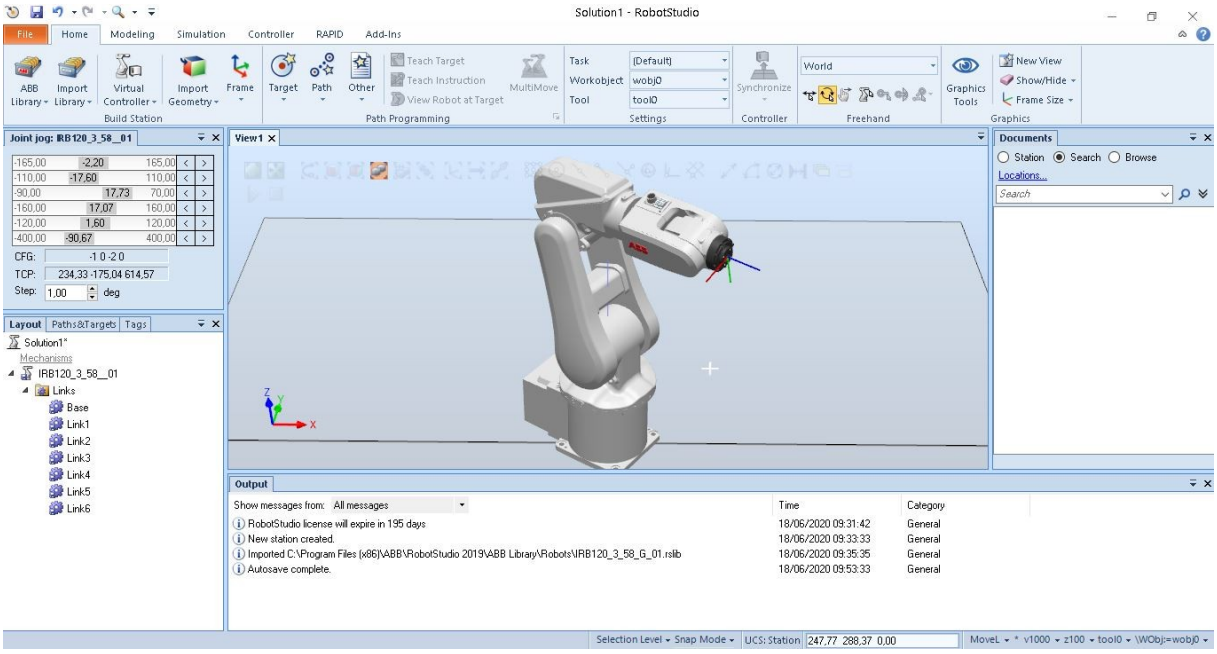

Figura 8: Interface do RobotStudio com o robô IRB 120

2.2.4 RoKiSim

O RoKiSim (*Robot Kinematic Simulator*) é um software de simulação de robôs desenvolvido no Laboratório de Controle e Simulação da ÉTS (*École of Technologie*

Supérieure). Este software possibilita apresentar em visão 3D todas as configurações de trabalho dos robôs de sua biblioteca, por meio do controle direto das variáveis de junta[PARALLEMIC, 2015].

Além de vários pontos de visão do robô, é possível a implementação de um arquivo simulação desenvolvido no Pspice, para realizar a movimentação do robô em questão. Este simulador deixou de ser atualizado em 2015[PARALLEMIC, 2015], porém ainda tem funcionalidades interessantes para aplicações simples, ou para auxiliar no aprendizado de robótica industrial. A interface deste programa é apresentada na figura 9, com o robô ABB IRB 120:

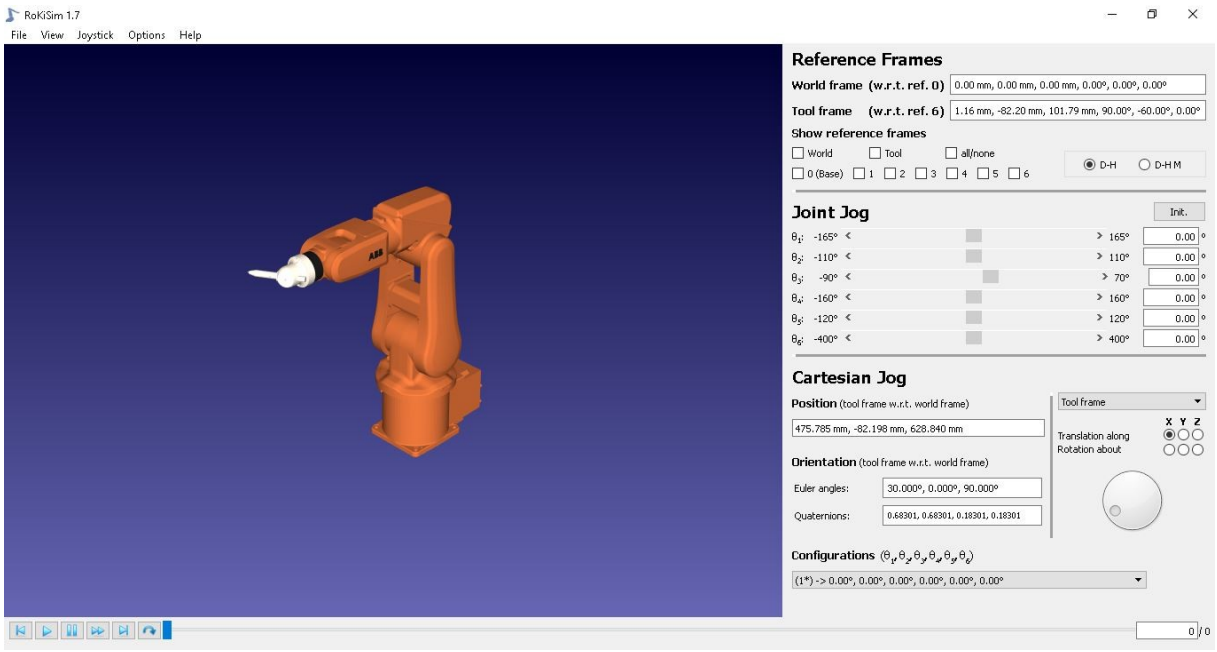

Figura 9: Interface do Simulador RoKiSim para o robô ABB IRB 120

2.3 Ferramentas desejáveis de simulação

Para a análise dos softwares disponíveis no mercado, foram levantadas algumas funcionalidades para a comparação do objetivo almejado para o software aqui desenvolvido. Estas funcionalidades são descritas, para sua fácil identificação:

- Simulação pontos: capacidade de simular a movimentação de movimentação entre pontos.

- Simulação de trabalho: capacidade de simular a movimentação para vários pontos, gerando diferentes trajetórias, não necessariamente conectadas entre si.
- Programabilidade: possibilidade de fácil programação da simulação, mantendo sua eficiência e confiança nesse processo. Para que a plataforma seja aplicada da forma idealizada, é necessário que a programação do robô seja desenvolvida e forma eficiente.
- Controle: possibilidade de controle do robô físico pelo software, ou por comunicação com um software terceiro.

A tabela seguir apresenta um resumo das análises levantadas para diferentes softwares de simulação robótica. Cabe salientar que alguns destes softwares possuem uso livre limitado, sendo analisado somente sua versão de testes:

Tabela 1: Análise das funcionalidades dos softwares disponíveis

Software	**Simulação Simples**	**Simulação Complexa**	**Programabilidade**	**Controle**
RoboAnalyzer [SAHA, Subir Kumar]	SIM	NÃO	NÃO	NÃO
RoboSim [C-STEM, 2020]	SIM	SIM	NÃO	NÃO
RobotStudio [ABB, 2008]	SIM	SIM	SIM	SIM
RokiSim [PARALLEMIC, 2015]	SIM	SIM	SIM	NÃO

Observa-se com a Tabela 1 que, apesar de existir diversos softwares bem estruturados para a simulação de robôs nas mais diversas funcionalidades, não se tem softwares livres com intuito de unificação e simulação. O software com a funcionalidade de Controle (RobotStudio), desenvolvido e comercializado pela ABB (Asea Brown Boveri) [ABB, 2008] disponibiliza a possibilidade de controle e simulação almejada. Limito, entretanto, a integração com robôs da própria empresa com seus softwares.

Esta análise evidencia a necessidade de um software livre com as funcionalidades descritas: uma plataforma de programação *off-line* de robôs industriais, em que haja a funcionalidade de simulação dos mesmos, a fim de melhorar a produtividade e assertividade deste processo.

Estes softwares apresentem opções não avaliadas, porém de grande importância para o software aqui desenvolvido. Para a maior generalidade da aplicação da simulação trabalhada, é interessante a possibilidade de simular diferentes efetuadores terminais. A possibilidade observada em exportar as trajetórias criadas em formatos externos pode facilitar a comparação entre diferentes trajetórias, facilitando a otimização neste requisito.

3 ANÁLISE DO ROBÔ DE TRABALHO

Nesta seção é realizada uma análise completa das propriedades do robô utilizado como base para a montagem deste simulador. Trata-se de um robô manipulador com configuração SCARA construído por Otavio Mello, sendo desenvolvido com objetivo de auxiliar no ensino de robótica industrial [MELLO, 2016]. Na Figura 10 a seguir têm-se o modelo apresentado enquanto a Figura 11 apresenta o robô real:

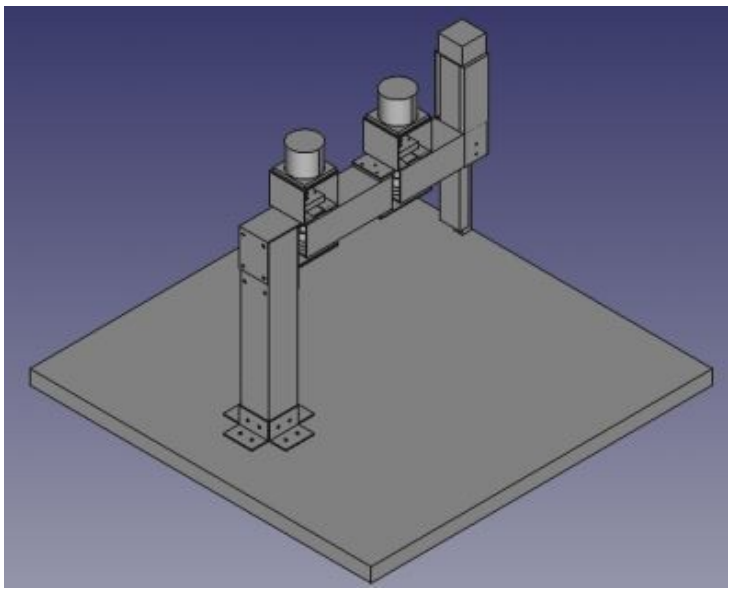

Figura 10: Modelo construído do robô utilizado de base para simulação [MELLO, 2016]

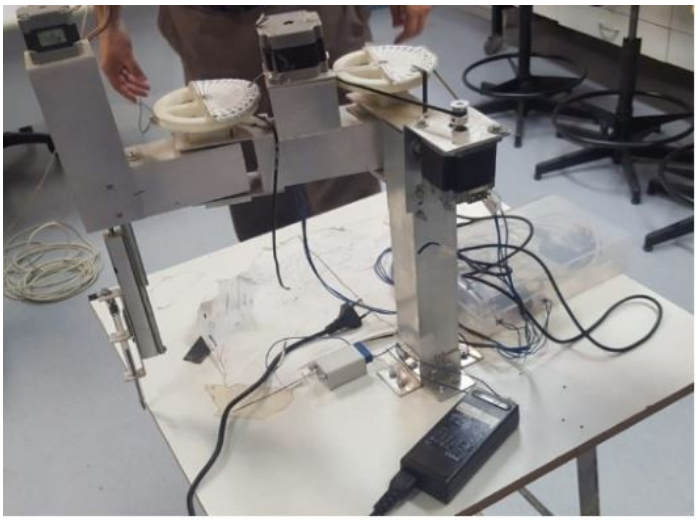

Figura 11: Construção do robô educacional 1

O trabalho desenvolvido por Mello apresenta as grandezas físicas do robô, entretanto, optou-se por utilizar os valores próprios medidos para o mesmo. Isso pois além das grandezas em si, é necessário ter certeza do sistema de referências utilizado no trabalho original, o que poderia ser um empecilho futuramente. Com isso esquematizou-se as grandezas relevantes de medição do robô na Figura 12:

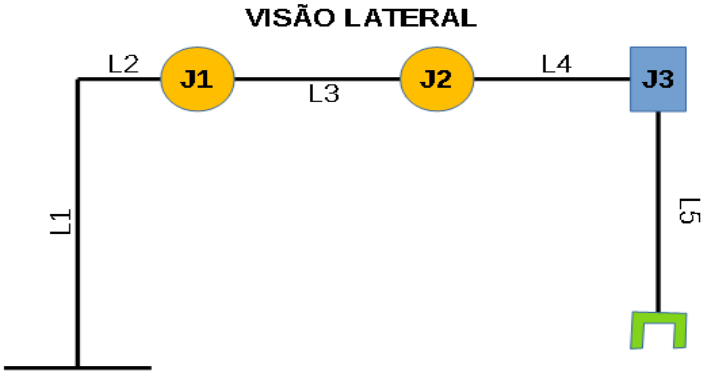

Figura 12: Esquemático das grandezas medidas para o robô

Os valores medidos para essa referência são apresentados na Tabela 2:

Tabela 2: Grandezas medidas para o robô

Grandeza	Valor [mm]
L1	280
L2	65
L3	174
L4	140

Estas grandezas foram medidas a partir da base do robô, levando em consideração um sistema de referência com zero dentro do tubo *metalon* do segmento **L1.** As juntas **J1** a **J2**, rotacionam seus respectivos elos criando uma movimentação do efetuador terminal, enquanto a junta **J3** é responsável por setar uma altura para este mesmo efetuador. A seguir apresenta um esquemático simplificado da visão superior do robô, e as variáveis de controle associadas a cada junta.

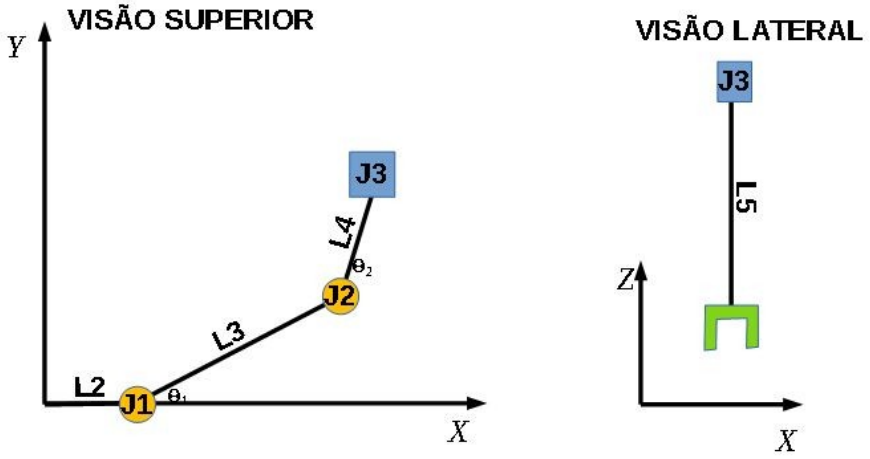

Figura 13: Representação das variáveis controladas do sistema

Alguns aspectos quanto a construção do robô são essenciais para seu completo entendimento. A sua construção faz com que as juntas $\mathbf{J_1}$ e $\mathbf{J_2}$ limitam a rotação em 90º e -90º, com a própria estrutura de fixação dos motores que representam as juntas, apresentados na Figura 10, funcionem como limitadores de fim de curso.

O mesmo é válido para a altura do efetuador terminal, em que não é possível ir às alturas maiores que a altura da base de fixação do elo $\mathbf{L_5}$, ou ainda mais baixo que a base de fixação do robô.

Para a simulação deste robô, é necessário que se conheça a descrição matemática de seu funcionamento, para tanto a seção a seguir apresenta uma análise da cinemática associada ao robô.

3.1 Análise cinemática

Nesta seção é analisado as propriedades matemáticas utilizadas para controle do robô. Para a descrição dos pontos de interesse no volume de trabalho do robô, dois tipos de análise são relevantes: análise cinemática direta e análise cinemática inversa. Ambas são utilizadas para a construção dos algoritmos de descrição da simulação, assim como para o controle do robô real futuramente.

Como apresentado na Figura 13, tem-se um sistema de referência cartesiano localizado na base do robô, no início do elo $\mathbf{L_1}$. Devido à simplicidade do robô de trabalho, somente uma análise geométrica é suficiente para o completo entendimento, não sendo necessárias análises mais complexas como a de Denavit-Hartenberg [CARRARA, 2015].

3.1.1 Cinemática Direta

A cinemática direta é uma análise matemática que associa diretamente as variáveis de controle do robô, ao posicionamento do efetuador terminal. A partir de uma relação geométrica das variáveis de controle ($\mathbf{\Theta_1}$, $\mathbf{\Theta_2}$, $\mathbf{L_5}$), chega-se a uma posição no espaço (**X, Y, Z**). O elo $\mathbf{L_2}$ é uma transformação de posicionamento constante, logo podemos definir um vetor e posição $\mathbf{P_0}$**:**

$$P_0 = \begin{bmatrix} L_2 \\ 0 \\ 0 \end{bmatrix} \tag{1}$$

Para o primeiro elo do robô, o vetor de posição gerado $\mathbf{P_1}$, partindo da junta $\mathbf{J_1}$, as componentes do posicionamento analisando a Figura 14 são:

$$P_1 = L_3 . \begin{bmatrix} \cos(\theta_1) \\ sen(\theta_1) \\ 0 \end{bmatrix} \tag{2}$$

De forma análoga, para o segundo elo do robô, partindo da junta **J_2,** temos um vetor posição **P_2** que associa a posição do vetor **P_1** para a posição do efetuador terminal. Como este posicionamento é dependente do primeiro ângulo, tem-se uma soma de das variáveis de controle angular:

$$P_2 = L_4 . \begin{bmatrix} \cos(\theta_1 + \theta_2) \\ sen(\theta_1 + \theta_2) \\ 0 \end{bmatrix} \tag{3}$$

A partir da junta **J_3**, temos uma relação de soma para a altura final do efetuador terminal. Tal altura, assumindo que os elos **L_2**, **L_3** e **L_4** estão alinhados sem variação no eixo-Z, dependerá da variação de **L_5** e da altura inicial dada por **L_1**.

$$P_3 = \begin{bmatrix} 0 \\ 0 \\ 280 - L_5 \end{bmatrix} \tag{4}$$

Somando os vetores de posição construídos, podemos associar o ponto zero do sistema de referência a posição do efetuador terminal, dada as variáveis de controle implementadas.

$$P = P_0 + P_1 + P_2 + P_3 \tag{5}$$

$$P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} L_2 + L_3 . \cos(\theta_1) + L_4 . \cos(\theta_1 + \theta_2) \\ L_3 . \sin(\theta_1) + L_4 . \sin(\theta_1 + \theta_2) \\ 280 - L_5 \end{bmatrix} \tag{6}$$

Para os valores reais apresentados na Tabela 2, tem-se a equação a seguir, utilizada que será utilizada para a simulação do robô.

$$P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 65 + 174 . \cos(\theta_1) + 140 . \cos(\theta_1 + \theta_2) \\ 174 . \sin(\theta_1) + 140 . \sin(\theta_1 + \theta_2) \\ 280 - L_5 \end{bmatrix} \tag{7}$$

3.1.2 Cinemática Inversa

A cinemática inversa, de forma contraria a apresentada na seção anterior, procura-se associar uma entrada de posicionamento (**X**, **Y** e **Z**), para variáveis de controle ($\mathbf{\Theta_1}$, $\mathbf{\Theta_2}$, $\mathbf{L_5}$) que leve a estes pontos específicos. Podendo escolher um ponto específico que se deseja posicionar o efetuador terminal, e descobrir um par de variáveis de controle que as consigam. Analisando a figura seguir, temos as referências geométricas importantes entre o efetuador terminal e a base da junta $\mathbf{J_2}$.

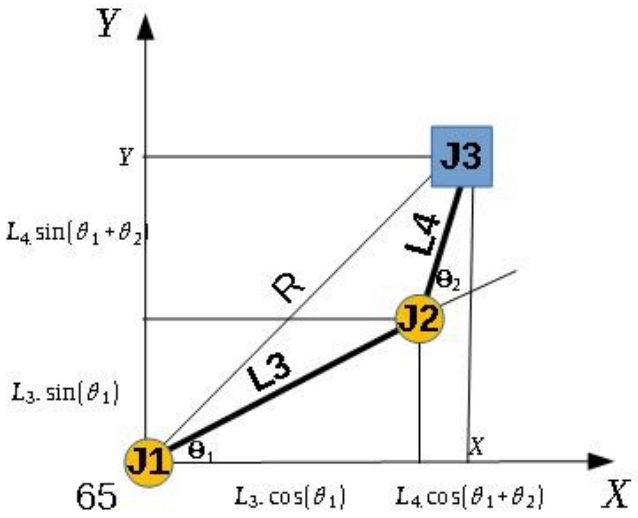

Figura 14: Relações geométricas do robô

Vemos que a partir do triângulo exterior gerado pela base, até o efetuador terminal, têm-se:

$$R^2 = X^2 + Y^2 \tag{8}$$

Porém os valores de **X** e **Y** foram definidos na seção anterior, chegando á:

$$\begin{aligned} X^2+Y^2 &= (L_3.\cos(\theta_1)+L_4.\cos(\theta_1+\theta_2))^2+(L_3.\sin(\theta_1)+L_4.\sin(\theta_1+\theta_2))^2 \\ X^2+Y^2 &= (L_3)^2+(L_4)^2+2.L_3.L_4.[\cos(\theta_1).\cos(\theta_1+\theta_2)+sen(\theta_1).sen(\theta_1+\theta_2)] \end{aligned} \tag{9}$$

Da soma de produtos de senos e cossenos, vemos que a relação pode ser simplificada à:

$$X^2+Y^2=(L_3)^2+(L_4)^2+2.L_3.L_4.\cos(\theta_2) \tag{10}$$

Isolando a variável de controle θ_2:

$$\theta_2=\pm\arccos\left(\frac{X^2+Y^2-(L_3)^2-(L_4)^2}{2.L_3.L_4}\right) \tag{11}$$

Tanto a solução positiva, como a solução negativa da equação anterior satisfazem o posicionamento do efetuador terminal, diferenciando quanto a direção do elo **L_3**. De forma prática, deve-se levar em consideração a limitação para ângulos maiores que 90º. Isso pois no limite destes ângulos, para atender a um ponto no semiplano positivo, caso escolhida a solução negativa, θ_1 precisaria ser um valor maior que 90º para chegar ao ponto final. A prova matemática não é discorrida, pois é um aspecto simples de se observar na Figura 14. Desta a variável de controle é descrita da forma:

$$\left\{\begin{array}{ll}\theta_2=+\arccos\left(\frac{X^2+Y^2-(L_3)^2-(L_4)^2}{2.L_3.L_4}\right) & ;Se\ Y\geq 0\\ \theta_2=-\arccos\left(\frac{X^2+Y^2-(L_3)^2-(L_4)^2}{2.L_3.L_4}\right) & ;Se\ Y<0\end{array}\right\} \tag{12}$$

A segunda variável de controle θ_1, é analisada observando na Figura 14 que para α:

$$\tan(\alpha)=\frac{Y}{X} \tag{13}$$

Para o ângulo intermediário β, têm-se [CARRARA, 2015]:

$$\tan(\beta)=\frac{L_4.sen(\theta_2)}{L_3+L_4.\cos(\theta_2)} \tag{14}$$

Vendo ainda que é válida a relação:

$$\theta_1=\alpha-\beta \tag{15}$$

Aplicando a tangente de θ_1, têm-se:

$$\tan(\theta_1) = \tan(\alpha - \beta) \tag{16}$$

Pela lei das tangentes [CARRARA, 2015]:

$$\tan(\alpha - \beta) = \frac{\tan(\alpha) - \tan(\beta)}{1 + \tan(\alpha).\tan(\beta)} \tag{17}$$

Com as equações 13 e 14, chega-se a relação para o ângulo requerido:

$$\tan(\theta_1) = \frac{Y.(L_3 + L_4.\cos(\theta_2)) - X.(L_4.sen(\theta_2))}{X.(L_3 + L_4.\cos(\theta_2)) + Y.(L_4.sen(\theta_2))} \tag{18}$$

$$\theta_1 = \arctan\left(\frac{Y.(L_3 + L_4.\cos(\theta_2)) - X.(L_4.sen(\theta_2))}{X.(L_3 + L_4.\cos(\theta_2)) + Y.(L_4.sen(\theta_2))}\right) \tag{19}$$

A altura é facilmente invertível por:

$$L_5 = 280 - Z \tag{20}$$

Deve-se salientar quanta a limitação física em termos de altura implícita nessa equação. A variável controlada $\mathbf{L_5}$ não pode passar da altura da construção de sua base, logo há um limite para $\mathbf{Z} = 280$, ou $\mathbf{L_5} = 0$. De forma análoga, para o esquemático apresentado na Figura 13 e Figura 14 observa-se uma limitação inferior em $\mathbf{Z} = 0$ e $\mathbf{L_5} = 280$, quando o efetuador terminal esta encostado no chão. As relações de cinemática inversa se reduzem à:

$$Q = \begin{bmatrix} \theta_1 \\ \theta_2 \\ L_5 \end{bmatrix} = \begin{bmatrix} \arctan\left(\frac{Y.(L_3 + L_4.\cos(\theta_2)) - X.(L_4.sen(\theta_2))}{X.(L_3 + L_4.\cos(\theta_2)) + Y.(L_4.sen(\theta_2))}\right) \\ \pm\arccos\left(\frac{X^2 + Y^2 - (L_3)^2 - (L_4)^2}{2.L_3.L_4}\right) \\ 280 - Z \end{bmatrix} \tag{21}$$

Sendo a escolha de θ_2 dada pela equação 12.

3.2 Estrutura de controle do robô de trabalho

Os atuadores do robô de trabalho são caracterizados por 3 motores de passo, cada qual responsável pela movimentação de um elo do robô. Estes atuadores são descritos a seguir:

- Motor1 ($\mathbf{J_1}$): motor de passo unipolar/bipolar. Sua atuação esta condicionada a uma redução de 7.5 vezes.
- Motor2 ($\mathbf{J_2}$): motor de passo unipolar/bipolar Sua atuação esta condicionada a uma redução de 7.5 vezes.
- Motor3 ($\mathbf{J_3}$): motor de passo unipolar/bipolar. Sua construção não tem redução, atuando de forma linear por meio de um fuso com comprimento do tamanho da variação do elo.

Os três motores são do mesmo modelo, sendo cabível o acréscimo da informação da relação de atuarem com 1.6744 graus para cada passo dado pelo motor. Esta informação é necessária para definir-se a variação miníma de grau para suas respectivas variáveis de saída.

Estes atuadores são acionados por três placas L298[ST ELECTRONICS, 2021] que atuam com a função de ponte H. As ondas de pulsos utilizadas para a atuação do robô são geradas um microcontrolador Arduino Uno Rev3[ATMEL, 2021]. Sua programação gera ondas iniciais para que os atuadores não se movimentem por forças externas, e os pulsos gerados são gerenciados para que a movimentação dos motores sejam de forma sequencial.

4 CONTROLE DO ROBÔ

Foi determinado anteriormente que o controle do robô, a ser desenvolvido posteriormente, seria dado de forma off-grid, ou seja, poderia determinar o controle do robô sem sua retirada de suas funções, como forma de otimizar o processo de controle em uma aplicação industrial real. Para tanto um método de controle que realize tal iteração não dependente da visualização do robô.

A utilização de uma linguagem *off-line* de robôs possibilita este cenário, além dos objetivos discutidos no Capítulo 1.1. A combinação desta linguagem de programação com a simulação do cenário de movimentação do robô, tem o potencial de tornar a aplicação do robô mais eficiente.

Como o desenvolvimento de robôs para aplicação industrial não é concentrado, ou seja, o mercado é composto por várias empresas, durante as últimas décadas diversas linguagens de robô foram desenvolvidas para aplicação industrial, sendo que cada pode ter uma aplicação específica, ou propriedade intelectual reservada.

A tabela a seguir mostra algumas das linguagens de programação utilizadas e suas empresas de desenvolvimento[SOUSA, et all; 2018]:

Tabela 3: Exemplos de Linguagens de programação[SOUSA, et all; 2018](ADAPTADO)

Nome	Significado	Empresa	Data
AL	Assembly Language	Stanford University	1974-1983
AML	A Manufacturing Language	IBM	1972-1982
AUTOPASS	AUTOmated Parts Assembly System	IBM	1972
IRL	Intuitive Robot Language	Microbo	1978-1980
LM	Language of Manipulation	University of Grenobol	1973-1983
LPR	Langage de Programmation de Robot	Renault - ACMA	1978-1983
VAL	Victor's Assembly Language	Stanford University	1974-1978

A linguagem de programação utilizada neste trabalho, deve ser tal que atenda a todas as possíveis movimentações apresentadas para o robô de trabalho apresentado no Capítulo 3, além de antevir a possibilidade de inserção de controles mais complexos no futuro.

A linguagem AML, apesar de não ser utilizada atualmente pela IBM foi um diferencial nos anos 70, além de ter sido base de desenvolvimento para outras linguagens como AML/2, AML/E, AML/V e AML/X foram desenvolvidas a partir dela, dando o tom de sua relevância. A linguagem aqui desenvolvida será por isto, uma adaptação simplificada desta linguagem, com as funções necessárias para a simulação, e futuramente, o controle do robô de trabalho.

Para a utilização desta linguagem, é necessário o desenvolvimento de um compilador que interaja entre ela e a simulação, ou com o próprio robô quando se almejar o controle do mesmo.

4.1 Linguagem de programação adaptada

A linguagem de programação AML é completa, de modo que todas as funções de uma linguagem de programação usual são observadas nela, com adição de funções utilizadas para o controle do robô em questão.

A linguagem AML completa possui entre suas possibilidades[TAYLOR, 1982]:

- Declaração de variáveis (globais e locais):
- Expressões matemáticas.
- Expressões lógicas.
- Determinação de sub-rotinas.
- Criação de ciclos e decisões condicionais.
- Comandos de robôs e sensores.

Estas funções possibilitam a construções de programas completos, com a maior eficiência possível, entretanto para a simplificação deste desenvolvimento inicial, optou-se por desenvolver somente as funções de comando para o robô. As possibilidades para a simulação ficam limitadas, entretanto o desenvolvimento da estrutura de compilação e interpretação do robô neste primeiro momento é mais importante, que as funções em si.

Como o robô de trabalho ainda não possui sensores eletrônicos que auxiliem o seu controle, o desenvolvimento da interpretação dos mesmos fica condicionada a esse adendo futuro no robô. Como inicialmente almejamos somente a simulação do robô, esta não fica limitada pela adição da simulação dos sensores.

Observando as funções de comando de robôs, apresentadas por Taylor em *"AML: A Manufacturing Language"* [TAYLOR, 1982], abstraiu-se alguns comandos que são essenciais para o controle do robô simulado, e simplificou algumas de suas aplicações para a simplificação das funções completas apresentadas anteriormente:

Tabela 4: Instruções da linguagem de programação do robô

Instrução	Descrição
move Θ_1 Θ_2 L_5	Move o braço para a posição (Θ_1,Θ_2,L_5) por meio da cinemática direta.
pmove X Y Z	Move o braço para o ponto (X,Y,Z) no espaço por meio da cinemática direta.
lmove X Y Z	Move o braço para o ponto (X,Y,Z) no espaço por meio da cinemática direta em uma trajetória retilínea.
down	Move o braço para o ponto Z mais baixo possível.
up	Move o efetuador terminal para o ponto Z mais alto.
zmove Z	Move o efetuador terminal para uma altura Z específica, sem alterar os posicionamentos em X e Y.
home	Move o efetuador terminal para o ponto inicial de trabalho dado pelas variáveis de controle ($\Theta_1 = 0$, $\Theta_2 = 0$, $L_5 = 5$).

Cada comando interno deste, deve ser interpretado por um compilador com o sinal que deverá controlar o robô. No caso específico da simulação, para que a mesma ocorra de forma independente do controle do robô, é necessário ainda um interpretador desta linguagem, que associa o código objeto a movimentação do robô, de forma análoga ao realizado pelo robô real.

4.2 Sinais de controles

A descrição objetiva de como a programação para controle do robô acontece é relevante para definirmos como será gerado os sinais de controle dos atuadores. Como apresentado no Capítulo 3, o robô de trabalho possui seus atuadores caracterizados por motores de passo. Tais motores se deslocam um ângulo específico para cada sinal de passo recebido[BRITES, SANTOS; 2008].

O software então deverá gerar uma quantidade equivalente de passos para o robô, condizente com qualquer alteração de posicionamento apresentada na simulação. Será gerado portanto pulsos na plataforma do software aqui desenvolvido, que encaminhado para o controlador do robô, converterá em rotação radial. Isso para garantir a confiabilidade de simulação de controle simultâneo.

Escolheu-se valores binários a serem comunicadores de forma serial entre o software e o controlador do robô. Como a escolha de diferentes sinalizadores não alterariam a eficiência do

software desenvolvido, escolheu-se sinalizadores de fácil interpretação para o processo de programação. Sinalizadores apresentados na tabela a seguir:

Tabela 5: Tabela de códigos para acionamentos dos atuadores do robô

CÓDIGO NUMÉRICO	CÓDIGO ASCII	DESCRIÇÃO
65 / 66	A / B	Acionamento Horário/Anti-horário
67 / 68	C / D	Acionamento Horário/Anti-horário
69 / 70	E / F	Acionamento Horário/Anti-horário

A simultaneidade dos eventos de simulação e controle é diretamente afetada pela velocidade de comunicação entre o software e o controlador do robô de forma serial. Desta forma é necessário que sejam setados parâmetros que otimizem essa relação. A comunicação serial deve ser a mais rápido possível, maior frequência de atualização, diminuindo o tempo de delay neste quesito.

A princípio os mesmos marcadores de movimentação utilizados para a simulação podem ser utilizados para a movimentação do robô real. Entretanto, este princípio gera um problema de paridade entre as movimentações simuladas e realizadas, dado que o motor de passo somente é capaz de se mover em quantidades discretas de ângulos, enquanto na simulação eu consigo representar qualquer angulação (mesmo que somente visualmente).

5 CONCEITOS E FERRAMENTAS PARA SIMULAÇÃO

O conhecimento e a informação são uns dos artigos mais valiosos da economia, seja em uma análise macro da mesma, ou micro para setores individuais. Esta afirmação é válida não somente agora, como ao longo da história humana. A revolução industrial, via de exemplo, foi possibilitada pela criação, décadas antes, de “escolas técnicas” que sistematizaram o conhecimento adquirido até então, possibilitando sua aplicação em grande escala [GAVIRA, 2003].

O conhecimento está associado a aprendizagem, e a possibilidade de se entender o funcionamento de um fenômeno. Seja este um fenômeno novo ao qual não se entende por completo suas motivações, seja um fenômeno de descrição já conhecida, mas que há uma possibilidade de se aprender mais sobre ele.

Dois conceitos também relacionados ao conhecimento são imprescindíveis de se entender, para delimitar por completo a importância do conhecimento: dados e informação. Dados são registro realizados a cerca de um evento, mas que não há atribuição de significado para os mesmos. Informação por outro lado, é um conjunto de dados dirigidos, em que se pode abstrair significado, julgamento e interpretação. De forma análoga, ao se atribuir significado amplo a um fenômeno ou processo, pode-se afirmar aquisição de conhecimento a respeito do mesmo[GAVIRA, 2003].

Olhando no âmbito industrial a quantidade de conhecimento relacionado aos fatores de produção também é um fator primordial. A alta competitividade deste setor, faz com que pequenas diferenças de produtividade, matéria-prima, tempo, desgaste e outros, sejam um diferencial competitivo extremamente relevante. Como conhecimento a cerca destes recursos é um limitante para sua otimização, métodos eficientes de aquisição de conhecimento são demandados.

Uma ferramenta de solução possível para essa problemática é a simulação computacional. A ferramenta computacional torna possível, que a observação dos eventos associados a um sistema seja feita de forma rápida e barata. Desta forma a utilização de simuladores esta limitada somente pelo seu entendimento a cerca do sistema em específico simulado.

5.1 Modelos

Dado a diferença de paradigma entre o funcionamento de um sistema real, e uma representação computacional deste sistema, a validade de nossa simulação será tão grande quanto a semelhança do seu modelo ao sistema real. A imagem a seguir representa a extrapolação de informações de entrada e saída para um sistema real genérico, e um modelo do mesmo:

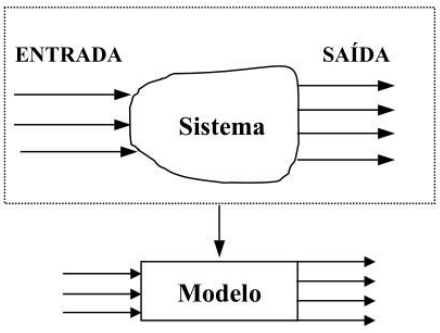

Figura 15: Processo simples de modelagem[GAVIRA, 2003]

Os modelos podem ser diferenciados quanto ao modo de desenvolvimento do mesmo, e com isso têm-se diferentes tipos de acréscimo de conhecimento a cerca do sistema. Entre os principais têm-se[GAVIRA, 2003]:

- Modelo físico: modelos físicos criam uma representação que se parece com o sistema representado. Podem ser considerados réplicas deste sistema, e de seu funcionamento se assim necessário.
- Modelo matemático: neste sistema a representação do sistema se dá por meio de informação lógica e quantitativa, sendo mais fácil a manipulação destas variáveis para experimentação e acréscimo de conhecimento interpolado para o sistema real.

Neste trabalho será desenvolvido um modelo matemático, pela maior facilidade de se modificar as variáveis de interesse no sistema. A base matemática do robô já desenvolvida no capítulo 3, torna possível a criação de seu modelo matemático. O modelo matemático terá ainda uma construção gráfica que facilite sua operação, além de facilitar a observação das relações existentes entre o robô e o modelo matemático construído.

5.2 Interface com Usuário

Utilizar uma simulação como forma de adquirir conhecimento no contexto da programação de robôs, depende da observação de um usuário capacitado. Desta forma se torna necessário além da construção da simulação, discutido a frente, a construção de uma interface com usuário que transforma a simulação em uma ferramenta de aprendizado assistido.

Desta forma levantou-se algumas possibilidades para o desenvolvimento desta interface:

- OpenGL e C/C++: A biblioteca de computação gráfico possibilita além da construção gráfica descrita no próximo subcapítulo, como a construção de toda a interface gráfica

com o usuário. Softwares desenvolvidos na linguagem C/C++ tendem a ter um desempenho alto, que é um fator requerido para este tipo de sistema. Entretanto a complexidade de desenvolvimento nesta linguagem pode ser maior que o necessário.

- Tkinter e Python: Tkinter é uma biblioteca desenvolvida para interpretador python, para a criação de GUI (*Graphics User Interface*). Possibilita uma portabilidade simplificada para os principais Sistemas Operacionais, além de ser de fácil aprendizado e desenvolvimento. A escolha da linguagem python pode diminuir a eficiência do sistema como um todo, entretanto torna mais fácil o desenvolvimento e o acréscimo de novas ferramentas ao software futuramente.

Como colocado, potencialmente a escolha da utilização do OpenGL pode garantir uma eficiência maior do sistema, como respostas mais rápida ou um menor requisição de recursos computacionais. Por experiências anteriores, no desenvolvimento de sistemas semelhantes, observou-se que a simplificação do desenvolvimento é um recurso mais importante para garantir a funcionalidade do software como um todo, que o potencial ganho de eficiência.

Com essas características definidas, optou-se por desenvolver o software em python, com auxílio da biblioteca tkinter para a GUI. A simplificação de desenvolvimento se torna o fator diferencial, dito que há uma limitação para o tempo de desenvolvimento do software, além que almeja-se futuramente acrescentar funções de controle do robô.

Uma solução intermediária entre as apresentadas, seria utilizar a biblioteca PyOpenGL. Uma biblioteca em python que utiliza as funções de OpenGL de forma cruzada, entretanto é uma biblioteca que eu não conhecia ao momento, e não havia como afirmar sobre a viabilidade de seu funcionamento. Mantendo a escolha de desenvolver o software em python, com a interface com o usuário possibilitada pela biblioteca Tkinter.

5.3 Construção Gráfica

A construção gráfica do modelo matemático da simulação pode acontecer de várias formas. Assim como a Figura 10 é uma representação gráfica do robô real, queremos realizar uma construção gráfica, que possibilite representar também as características dinâmicas do robô.

Dois tipos de representação gráfica comum são aplicáveis a esta situação em específico: sistema computacional CAD e motores de renderização gráfica.

5.3.1 Sistema computacional CAD

Os CAD's, *Computed Aided Design*, fornecem ferramentas para a representação gráfica de objetos sólidos, por meio da utilização de figuras abstratas no plano, ou objetos no espaço, como triângulos, quadrados, esferas, ou cubos. Sua utilização tem por foco a precisão matemática da representação do objeto real. Entretanto não é comum que este modelo venha junto de ferramentas de renderização, que possibilita a representação do movimento[CARRARA, 2015].

Alguns programas computacionais:

- AutoCAD: programa desenvolvido pela *Autodesk* é o líder de mercado para esse tipo de computação. Possui aplicações industriais, mecânicas, construção civil etc. Não possui um foco em específico mas é utilizado para várias aplicações.
- SolidWorks: sistema desenvolvido pela *Dassault Systèmes*, possui vastas ferramentas de design 3D, com aplicação industrial, mecânica, e até algumas ferramentas de animação. Grande biblioteca de modelos já criados, que auxiliam novos desenvolvimentos, porém na versão paga.
- *Blender*: software grátis desenvolvido pela alemã *Blender Foundation*, possui características tanto de modelagem, renderização e animação.
- FreeCAD: sistema de design 3D de software livre, desenvolvido pela FreeCAD fórum. Possui diversas ferramentas que propiciam a construção industrial e mecânica, mas que podem ser estendidas a outras aplicações.

Dada a simplicidade inicial da aplicação, criar um modelo 3D do robô de trabalho, todas as ferramentas são passíveis de aplicação quanto aos requisitos técnicos. A escolha se deu então pela disponibilidade do software e conhecimento prédio para o desenvolvimento. Escolheu-se o desenvolvimento do modelo do robô no FreeCAD. Por tratar de um software grátis e sem barreiras de entrada, sua utilização foi priorizada em detrimento dos softwares mais consolidados no mercado.

Mesmo que alguns dos softwares descritos anteriormente apresentem versões grátis, ou versões utilizadas para estudo, é necessário utilizar um software que não tenha limitação de recursos, e que mantenha sua aplicação possível caso haja um aumento na necessidade de complexidade do robô construído.

5.3.2 Motores de renderização gráfica

Para os softwares responsáveis pela exibição das imagens necessárias, dá-se o nome de motores de renderização gráfica, . Os motores de renderização gráfica também constroem imagens, que são uma representação gráfica de um modelo físico. Entretanto, neste caso, há uma ênfase nas características estéticas da representação do modelo, como cores, textura, luminosidade e outras, além de uma facilidade para a programação de movimentos.

Estes motores de renderização atualmente, possuem sua principal aplicação no contexto de criação de jogos. Isso pois, trata-se de uma aplicação com alta demanda de desempenho, além de ser também uma indústria com grande movimentação financeira.

Alguns exemplos de motores de renderização gráfica geral e motores de jogos[CARRARA, 2015]:

- 3DS Max: programa pertencente a *Autodesk*, utilizado para renderização e animação de jogos e filmes. Apesar de ser possível realizar o design, seu foco principal é na animação e renderização. Projeto descontinuado em 2015 mas que serviu de base para vários outros motores desenvolvidos atuantes no mercado.
- *Godot Engine*: motor de jogos desenvolvido no MIT, mas que hoje possui código público. Possui renderização gráfica baseada no OpenGL Es2 e 3, possibilitando construções em 2D, 3D no modo animação, física e colisões.
- *Unreal Engine*: motor de games desenvolvido pela *Epic Games*, está no mercado desde 1998 mas é utilizado em jogos até o momento. Sua linguagem principal de desenvolvimento é C++, mas apresenta possibilidade de portabilidade entre plataformas.
- *Unity Engine*: sistema criado pela *Unity Technologist*, utilizado em vários jogos da empresa que é líder no segmento de jogos.
- OpenGL: trata-se de uma biblioteca utilizado para programação gráfica, como apresentado anteriormente. Apresenta alto desempenho por ser desenvolvido em C e C++, e haver a possibilidade de utilização direta da biblioteca. Entretanto uma aplicação desenvolvida do zero pode apresentar uma complexidade grande, mesmo havendo várias funções de fácil aplicação.

- PyGame: trata-se de uma biblioteca desenvolvida para python com suporte para funções aplicáveis na criação de jogos, ou construções gráficas em geral. Torna possível que programas desenvolvidos em python apresentem construção gráfica 2D ou 3D, tenham suporte para criação de sons, detecção de colisões e renderização de forma geral[HOLZER, 2019].
- Matplotlib: não trata-se de um motor gráfico desenvolvido para jogos como os anteriores, mas comumente aplicado na apresentação de gráficos e dados em geral. Há suporte entretanto para a criação de animações, objetos no plano, objetos no espaço, além de possuir a versatilidade da linguagem python[HUNTER, et. al.; 2021].

Os motores gráficos apresentados acima possuem aplicações muito grandes, com alto desempenho. Deve-se entretanto escolher uma ferramenta que possibilite a implementação da proposta em tempo hábil, com eficiência aplicável, além de ser possível intercambiabilidade entre a simulação os outros sistemas do software: interface com usuário e controle do robô real.

Dito isso escolheu-se a opção menos obvia: utilizar a biblioteca Matplotlib. Como escolheu-se antes que o software como um todo seria desenvolvido em python, escolhe-se uma ferramenta que simplifique esta escolha anterior. Além que a biblioteca Matplotlib já era uma ferramenta conhecida pelo desenvolvedor, mantendo o objetivo de simplificação do desenvolvimento.

A escolha desta ferramenta torna o desenvolvimento simples e rápido, assim como alterações futuras do software, como simulação de robôs diferentes do apresentado no Capítulo 3, ou funções de controle diferentes.

5.4 Ferramentas aplicadas

O software será desenvolvido na linguagem de programação python. Inicialmente toda a interface com o usuário (*front end*) e os processos internos (*back end*) serão desenvolvidos utilizando as bibliotecas padrões da linguagem para isso, além da biblioteca tkinter em específico para a construção gráfica desta interface do programa.

A representação do robô será desenvolvida no FreeCAD, e utilizada uma exportação para representação no Matplotlib, dentro da GUI a ser desenvolvida. Queremos ainda determinar uma forma de conectar o controle da interface com o usuário.

6 DESENVOLVIMENTO DO SOFTWARE

As ferramentas ponderadas anteriormente foram utilizadas para o desenvolvimento do simulador. Neste capítulo são apresentados os resultados obtidos para cada ponto do desenvolvimento do software, assim como discussões quanto a erros e acertos das escolhas de ferramentas realizadas anteriormente.

Utilizou-se para escrever o software o Ambiente de Desenvolvimento Python Spyder, acompanhado de interpretador python 3.7, última versão no momento deste desenvolvimento. Escolha realizada pela praticidade do ambiente, e pela possibilidade de melhores ferramentas quanto a versão escolhida.

Como discutido pelas motivações de desenvolvimento no Capítulo 1.1, o software pretende ser uma plataforma de simulação e controle para um robô SCARA, com possibilidade de análise e desenvolvimento *off-grid*. Optou-se por uma nomeação direta RSAC: *Robot Simulation Analyze and Control.*

6.1 Diagrama de funcionamento do Software

Com o funcionamento necessário para o software estipulado durante todos os capítulos anteriores, desenhou-se um diagrama como representação de seu funcionamento. Este diagrama apresenta os passos internos utilizados pelo programa para o manejo interno de informação, assim como as respostas necessárias do usuário para a realização de uma atividade em específico.

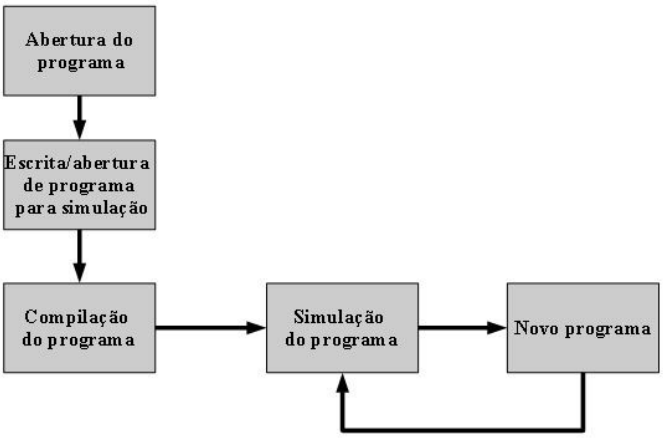

Figura 16: Diagrama de fluxo do software

A programação orientada a objeto utilizada no desenvolvimento da interface do programa (*tkinter*), possibilita que o desenvolvimento em módulos independentes seja facilitada, definindo cada como uma classe do objeto principal. Desta forma alterações relevantes, como acréscimo do modulo de controle, ou alteração do robô de trabalho, sejam facilmente realizados, sem a perda das construções independentes. A Figura 17 mostra a modularização desenvolvida:

Figura 17: Módulos do software

Como a construção gráfica é feita em um módulo único, alterações das dinâmicas do robô discutidas no capítulo 3 também podem ser otimizadas facilmente. Neste trabalho

entretanto uma validação do objeto simulado e o objeto real não é desenvolvido, ficando futuramente dependente para uma utilização eficiente deste simulação.

A confiabilidade deste software é uma dos aspectos mais importante, dado que em sua suposta aplicação procura-se diminuir os erros de programação de robôs reais. Dado a importância desta confiabilidade, desenvolveu-se também um diagrama para respostas de erros durante a simulação. Com isso espera-se que o processo de programação do robô seja mais ágil, e livre de erros imprevisíveis, assim como o desenvolvimento futuro do controle do mesmo.

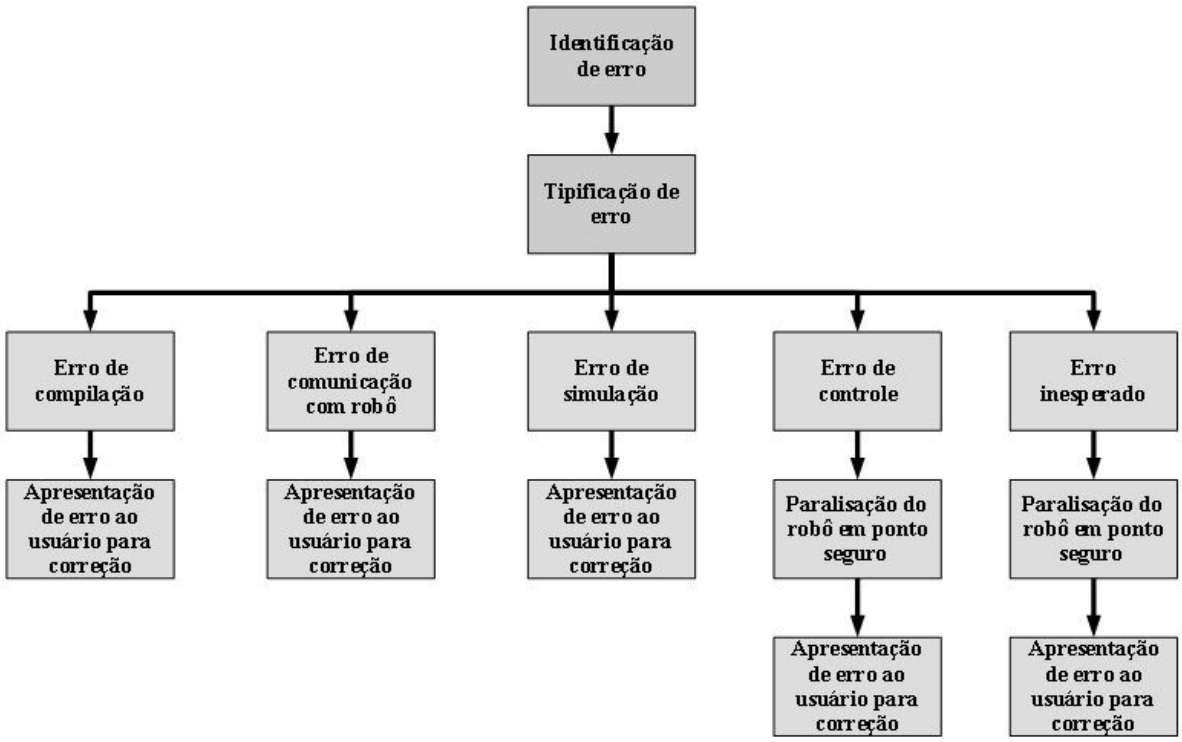

Figura 18: Diagrama de erros

6.2 Requisito de software

Como discutido no capítulo 5, a aplicação de simuladores para se adquirir conhecimento sobre um processo, é condicionada a fidelidade do processo simulado com o processo real. Para que o ganho de produtividade no processo de programação do robô seja relevante, o software de simulação deve garantir fidedignidade das regras do ambiente de trabalho do robô, com o ambiente de trabalho virtual construído.

A associação futura da simulação de controle do robô para o controle real, faz com que a estabilidade do software seja um diferencial importante. Eventuais erros internos do software tem o potencial de criar acidentes com lesões aos operadores, caso haja, ou mesmo danos a linha de produção e o próprio robô controlado.

Além do desenvolvimento ser voltado para a segurança, é necessário que possíveis erros de um cenário não prescrito, também sejam limitados em sua influência na simulação e no controle do robô. Desta forma cria-se uma barreira de proteção ainda maior para a incidência de sinistros no controle do robô de trabalho.

Por se tratar de um software de aplicação técnica específica, a funcionalidade do mesmo para a aplicação proposta é mais importante que o design da interface construída. Busca-se uma interface que suporte todas as funções que se desenvolverá para o controle, mantendo o aspecto de eficiência no processo de programação *off-line* do robô, o que é necessário caso a função do robô em uma planta industrial seja complexa.

A Tabela 7 apresenta os principais requisitos levantados para o desenvolvimento deste software:

Tabela 6: Requisitos de Software

REQUISITO	JUSTIFICATIVA
Fidedignidade com processo real	A aplicação do software depende da qualidade de sua representação do processo real.
Tratamento preciso com erros	Para segurança do software associado a processos reais, é tratamento com erros deve ser conhecido com o usuário.
Limitação da influência de sinistros	Sinistros internos do software não devem causar acidentes no controle do robô.
Interface voltada a produtividade.	A produtividade no processo de programação *off-line* deve ser alta.
Modularização	Alterações no robô de trabalho, mesmo sua mudança total deve ser facilitada.

Ainda no contexto de aplicação técnica, a documentação é essencial para que o usuário entenda a funcionalidade e as funções internos do robô, sendo importante estabelecer a importância do requisito de documentação. Entretanto esta documentação não é desenvolvida até o momento, sendo uma dependência para o futuro.

6.3 Design do Simulador

Com as características de design relacionadas a interface do usuário, construiu-se cada módulo do software. O que também é esperado pois há à necessidade de modularização do software, discutida anteriormente.

O acesso as ferramentas do software são acessadas por meio de um menu de opções. Além de ser um método intuitivo de apresentação de possibilidades, é possível acrescentar opções facilmente neste menu. A Figura 19 apresenta este Menu com visto:

Figura 19: Design do menu construído

De forma complementar, a Figura 20 apresenta um diagrama das opções deste menu:

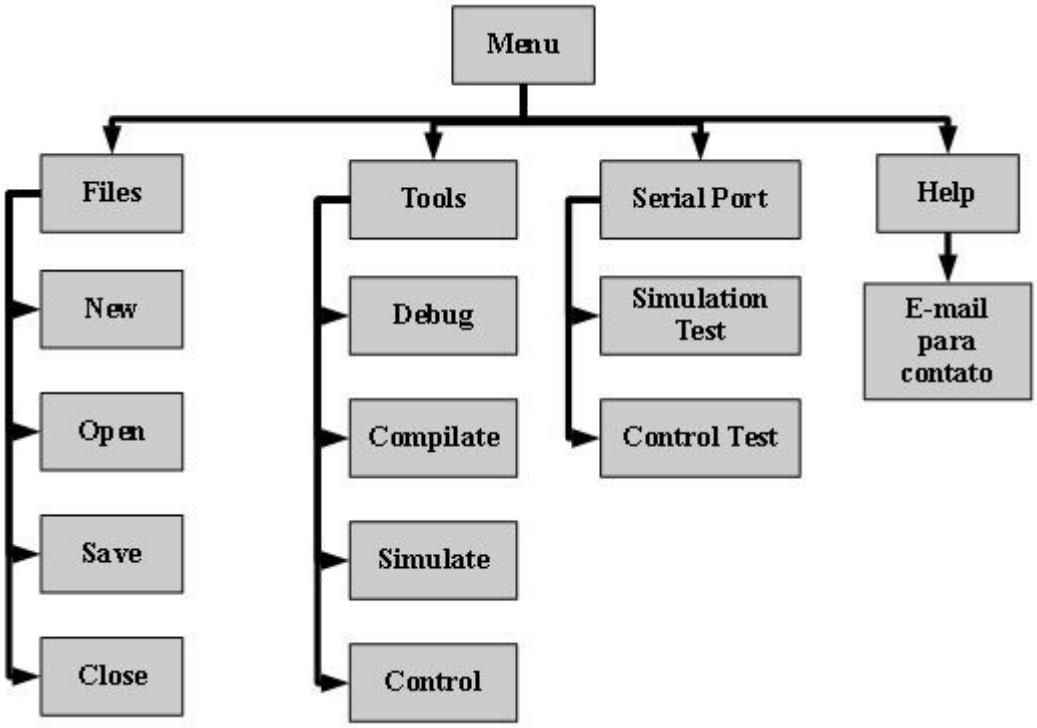

Figura 20: Diagrama para o Menu de funções do software

Para a escrita dos programas desenvolveu-se uma interface com possibilidade de escrita, e edição de texto. A mesma possui várias das funções necessárias para um editor de

texto, facilitando a programação por meio do operador. A figura apresenta esta interface para edição de texto acompanhado de um programa escrita na mesma, na linguagem de programação do nosso robô de trabalho:

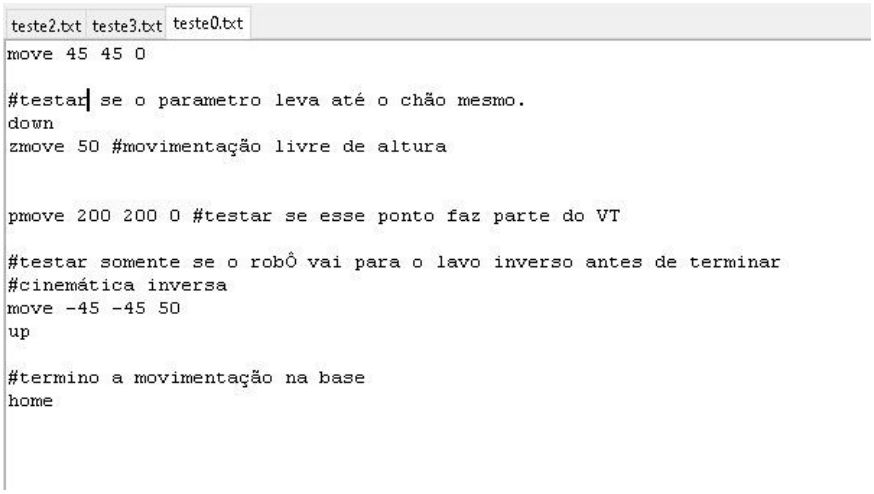

Figura 21: Interface de programação

O ambiente gráfico construído com o Matplotlib foi colocado em um plano cartesiano tridimensional, para facilitar a associação da movimentação do robô com o posicionamento no espaço real. A visualização pode ser aumentada ou diminuída, possuindo ainda possibilidades de correção de perspectiva e rotação.

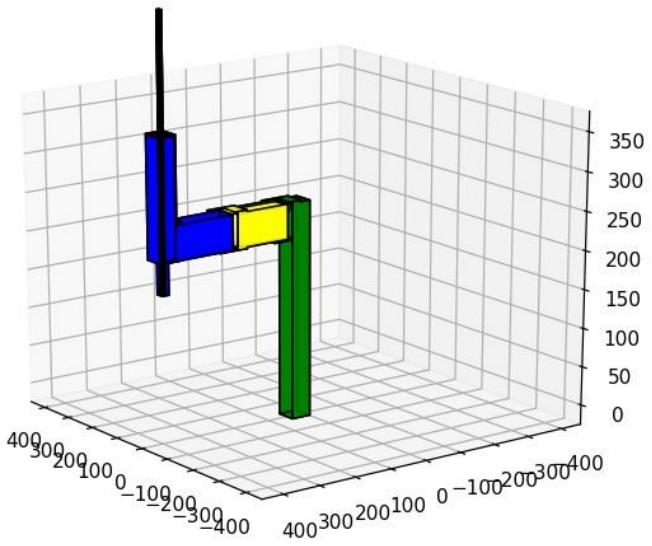

Figura 22: Visão do robô construída na interface gráfica

Com a importância do usuário em acompanhar os erros internos do programa, têm-se um espaço direcionado somente para a apresentação de erros e mensagem ao programador do robô.

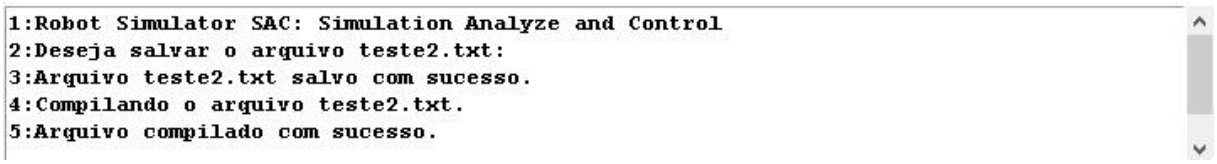

Figura 23: Caixa de erro

Cada módulo desenvolvido separadamente foi apresentado em conjunto para a construção da interface gráfica com o usuário. A concatenação destes módulos não interrompeu a funcionalidade de nenhum dos sistemas, a interdependência de cada módulo ficou ponderada por propriedades de cada Classe.

A Figura 24 apresenta a interface desenvolvida:

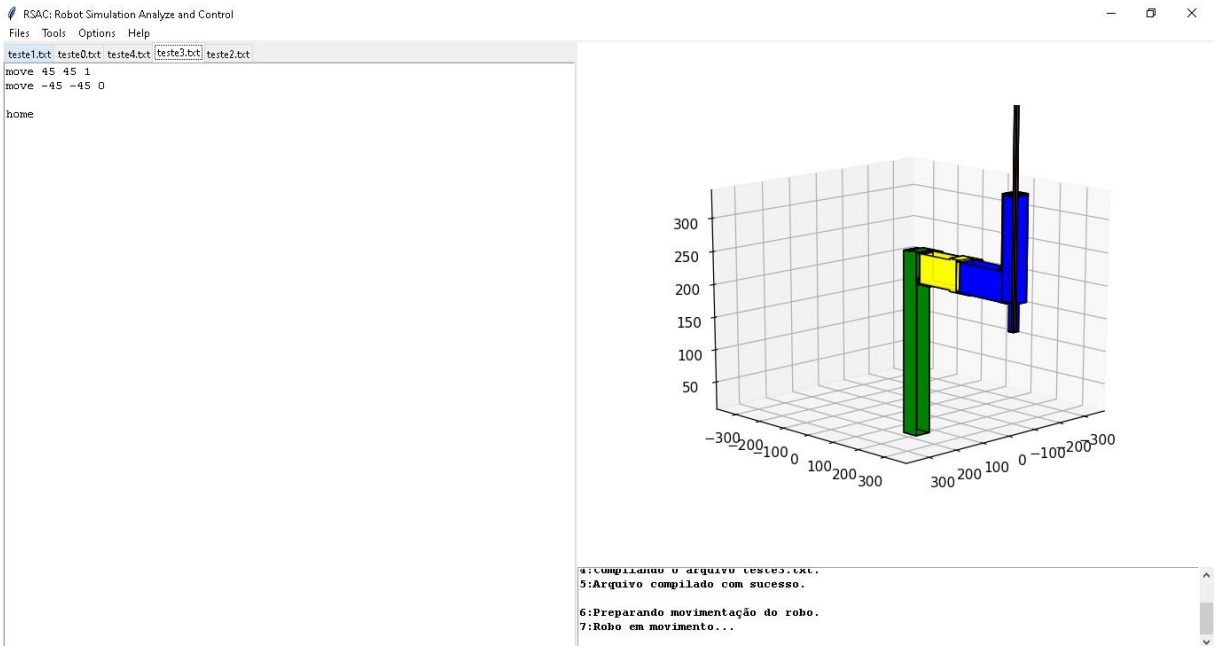

Figura 24: Interface Gráfica desenvolvida

A interface funciona de forma satisfatória com os requisitos almejados, sem problemas relacionados a desempenho ou a dificuldades do uso das funções de simulação. O tratamento dos arquivos gerados na programação do robô, ou abertura de arquivos trabalhados anteriormente ocorre da forma esperada.

Como não há a determinação de um ambiente de trabalho específico até este momento, a construção gráfica é somente do robô em si, sem ferramentas ao redor. Entretanto caso construa-se um ambiente mais complexo, (como esteiras, ferramentas adicionais ou mesmo outros robôs), pode-se acrescentar tal modificação no módulo de construção gráfica.

6.4 Linguagem de programação do robô

A utilização da linguagem de programação do robô construída passe-se por dois pontos:

1. Análise da Linguagem de programação do robô e conversão para pseudocódigo máquina.
2. Interpretação da do pseudocódigo de máquina do robô na linguagem de máquina construída.

Todas as instruções apresentadas na Tabela 5 foram implementadas com sucesso, tendo bons resultados na utilização da mesma como ferramenta de controle da simulação.

Na primeira etapa o código disponibilizado pelo usuário é analisado quanto a possíveis erros:

- Erros sintáxicos: procura-se por erros relacionados a sintaxe do programa, seja pela declaração de instruções não definidas na linguagem de programação do robô ou pela utilização de formatos incorretos para a instrução proposta.
- Erros semânticos: os erros semânticos na compilação do arquivo são associados principalmente a declaração de pontos ou trajetórias não pertencentes ao Volume de Trabalho do robô, logo não têm significado na representação interna do software.

Como dito, criou-se uma codificação intermediária para as instruções da programação do robô. Essa codificação é análoga a transformação a uma montagem de código *assembly* para código de máquina. Optou-se por essa transformação pois facilita a passagem de informação para o interpretador do código de programação do robô real, mesmo que para o robô simulado não se tenha uma diferença de desempenho. Escolheu números em sequência pela simplicidade da escolha, assim como pela utilização de menos bits na comunicação com o robô futuramente. A tabela apresenta as instruções implementas e sua codificação interna do programa:

Tabela 7: Codificação interna das instruções

INSTRUÇÃO	CÓDIGO INTERNO
move	1
pmove	2
up	3
down	4
delay	5
home	6
lmove	7
zmove	8

Apesar de as funções implementadas serem suficientes para diversas atuações do robô, no contexto de eficiência seria necessário a implementação de mais funções que facilitem o processo:

- O controle de velocidades da simulação e controle do robô não é abordado, sendo controlado somente o posicionamento independente da velocidade do robô. Funções relacionadas a este aspecto devem ser desenvolvidas para melhor utilização do software.
- A criação de trajetória linear é coberta pela função **lmove** implementada. Entretanto em rotas circulares, em que somente espera-se que o robô movimente-se ao ponto específico, várias possibilidades não são abrangidas, como:
  - A movimentação intercalada de cada atuador.
  - Requisito de velocidade para que ambos os atuadores terminem uma movimentação ao mesmo tempo.
  - Funções que a movimentação em arco tenha uma curvatura específica, de modo a evitar um obstáculo intermediário entre dois pontos.
- Para a produtividade do processo de programação do robô, funções de análise do cenário de movimentação do robô seriam interessantes. Funções que calculem a trajetória mais rápida de um ponto a outro.

A segunda etapa utilizada para utilização da linguagem desenvolvida é a interpretação da linguagem intermediária construída, por meio do módulo de construção gráfica. Optou-se por fazer com que a interpretação desta linguagem intermediária na representação gráfica do

robô, ocorra de forma independente da compilação em si, para que se mantenha condizente com os requisitos de modularização proposto nos capítulos anteriores.

A utilização desta linguagem intermediária facilita também, esta segunda etapa da utilização da linguagem de programação do robô. Com utilização de códigos intermediários de representação para cada instrução, a transformação da instrução de entrada no módulo gráfico, para a saída gráfica da movimentação do robô fica condicionada somente a uma comparação simples entre inteiro e um vetor de inteiros. Como cada código interno de representação das instruções foi escolhido como uma contagem de naturais (1, 2, 3 etc), é possível associar diretamente uma instrução a sua função de construção gráfica.

Com as discussões levantadas anteriormente construiu-se a figura representa o diagrama do caminho de dados realizado para a compilação de um arquivo no robô:

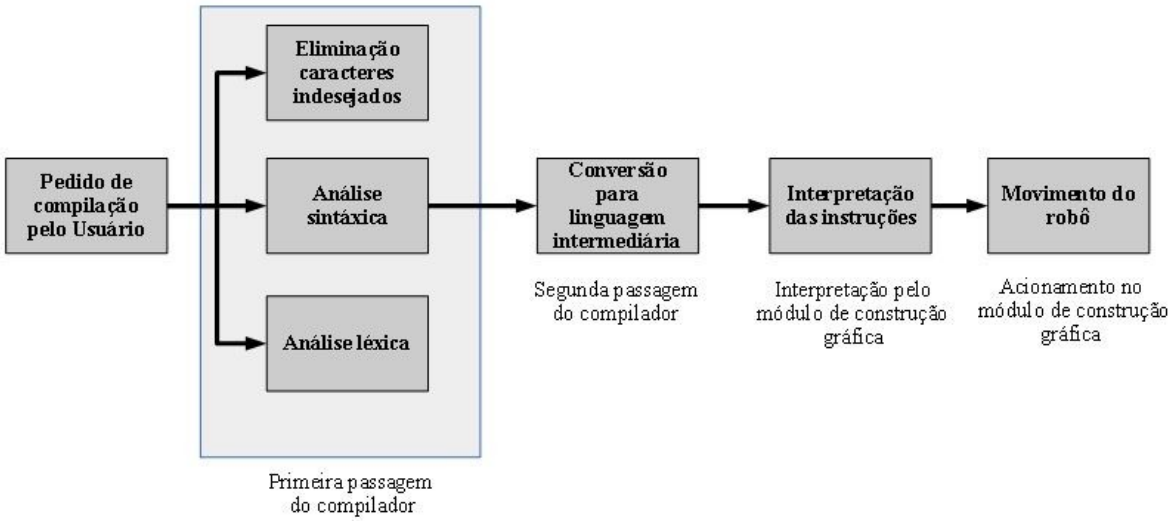

Figura 25: Diagrama do compilador

6.5 Representação Gráfica do Robô

A representação gráfica, como descrito no Capítulo 5.4 foi desenvolvida em dois passos distintos:

1. Criação do modelo CAD do robô no software FreeCAD.

2. Exportação do modelo CAD para apresentação na interface do programa por meio da biblioteca Matplotlib.

Com a medidas apresentadas para o robô no Capítulo 3 assim como na Tabela 2, foi possível construir o modelo CAD para o robô de trabalho. De modo geral a construção do robô é dividida em quatro peças, que vão desde a fixação na base até o efetuador terminal.

Como as medidas internas relacionadas ao atuador **J3** não alteram a relação do posicionamento final, simplificou-se este mecanismo. Sem perda entretanto de confiabilidade na representação gráfica construída e o robô real. A Figura 26 apresenta o modelo CAD construído para o robô, onde esta diferença é observada em comparação com a Figura 10.

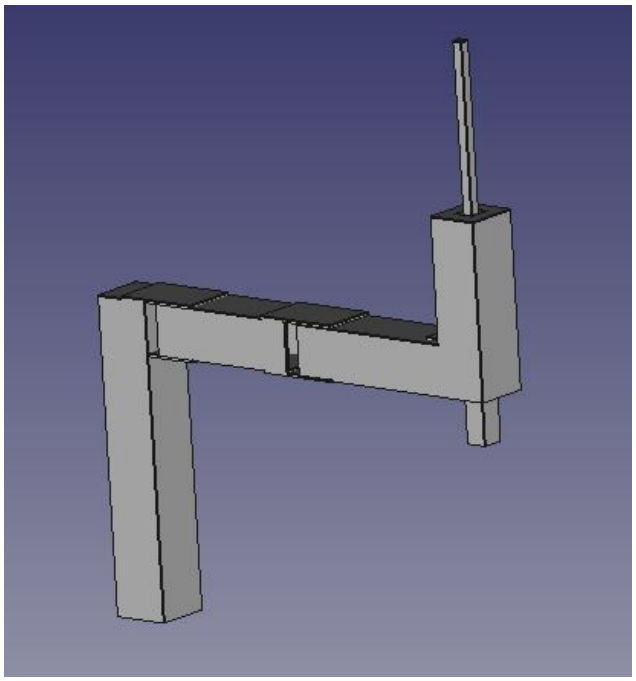

Figura 26: Construção CAD do robô

O FreeCAD possibilita a exportação dos dados relacionados as peças criadas em termos de pontos e superfícies. Exportou-se estes dados em um arquivo de texto, e criou-se um algoritmo que converte estes dados para um compreendido pelo pelo Matplotlib. Para que esta conversão não acontecesse sempre que o software de simulação fosse iniciado, copiou-se estes dados para o arquivo-fonte do software em python.

A Figura 27 e Figura 28 presentas o modelo criado no Matplotlib em diferentes ângulos:

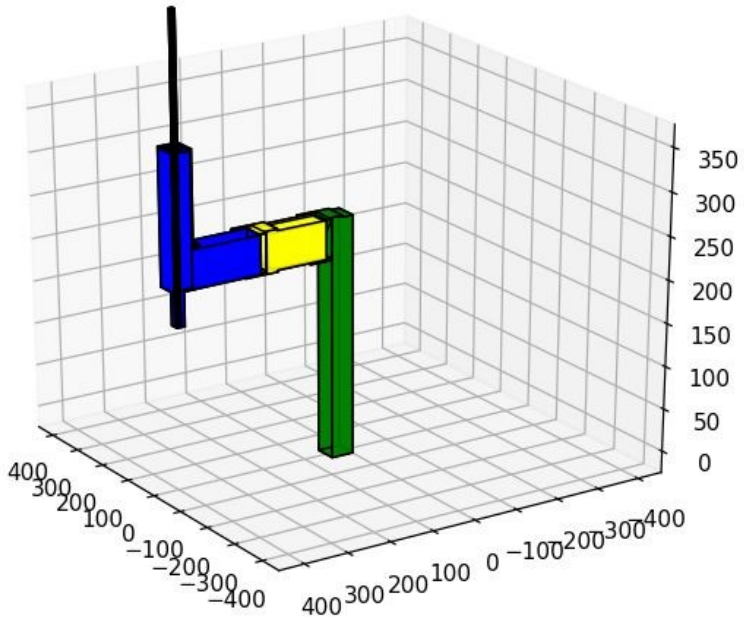

Figura 27: Construção do robô de trabalho no Matplotlib

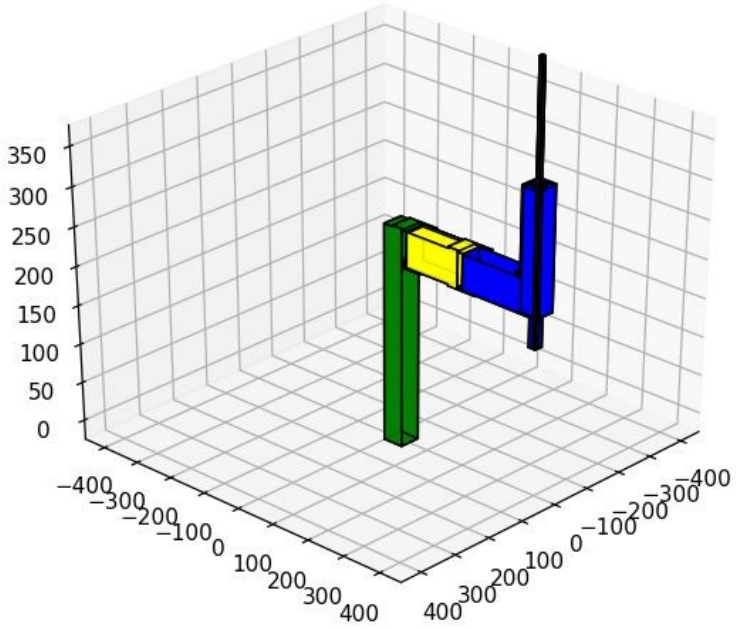

Figura 28: Construção do robô de trabalho no Matplotlib visão rotacionada

Nesta construção modificada do robô, toda a construção do robô fica condicionada somente em termos de pontos que geram superfícies planas. Desta forma, a rotação das partes móveis do robô são facilmente a aplicáveis pela rotação dos pontos de criação do robô. Os

atuadores do robô real possuem uma relação de passo do motor e ângulo de saída na junta do robô, entretanto para simulação neste primeiro momento optou-se por utilizar uma relação fixa de 1/1, ficando a cargo de testes futuros a determinação a relação de passo/ângulo do robô real.

Como analisado no Capítulo 3, a rotação em um determinado ângulo do efetuador terminal pode ser dado pela equação 7, assim como para os atuadores **J1** e **J2** pelas equações 2 e 3 assim como a relação para o atuador **J3** pode ser dado pela equação 4. Utilizando estas equações determinou-se as rotações relacionadas a cada peça da representação do robô.

Um aspecto importante é que caso haja uma correção da peça para o ponto de origem, a aplicação da rotação fica direta, substituindo um passo da multiplicação de rotação, pela soma de um *off-set* fixo para cada peça.

A precedência de cada braço em relação, faz com que toda rotação no atuador **J1** interfira no posicionamento dos atuadores **J2** e **J3**, assim como rotações em **J2** interferem em **J3**. Essa precedência é importante tanto para o funcionamento do robô em seu controle, como para associar possíveis erros entre a movimentação do robô.

6.6 Ferramentas de Desenvolvimento e Software Final

As ferramentas utilizadas tiveram sua escolha fundamentada em requisitos mínimos de aplicação, assim como na simplicidade de desenvolvimento do software final. Como o software foi desenvolvido com os objetivos descritos no Capítulo 1.1, as escolhas foram acertadas.

A escolha do desenvolvimento do código como um todo em python foi um acerto, pois possível o desenvolvimento de várias funções no tempo disponível, e a modulação que torna a manutenção e otimização simplificada.

A escolha da biblioteca Tkinter também configura uma boa escolha para o desenvolvimento da interface com o usuário, dado a simplicidade da construção de estruturas como a Interface de programação apresentado na Figura 21. O fato de haver uma importação automática da construção gráfica realizada com a biblioteca Matplotlib também é um facilitador para a utilização desta ferramenta em específico.

Quanto a construção gráfica, um problema encontrado foi a falha no algoritmo de conversão do CAD para a representação gráfica no software por meio da biblioteca

Matplotlib. Esses erros específicos acontecem em pontos específicos da construção gráfica, de forma que ficam de difícil reconhecimento, como é apresentado nas figuras Figura 28 e Figura 27. Mesmo com este obstáculo para a representação plena do modelo construído, a escolha desta ferramenta até o momento se torna acertada, pois houve uma grande simplificação da visualização do robô construído, como opções de zoom, rotação, e animação.

Não há portanto a necessidade de alterações das ferramentas escolhidas para o desenvolvimento do software final, ficando dependente entretanto a otimização da utilização das ferramentas aplicadas, em pontos específicos de sua utilização.

6.7 Desenvolvimento do Controle

Há uma variação no posicionamento do robô durante uma simulação, de modo que a variação das variáveis de posição do robô na simulação ($\Delta\theta_S$), pode ser convertida em uma variação das variáveis de posição do robô real ($\Delta\theta_R$). Desta forma garantimos parcialmente que a simulação ocorrerá simultaneamente.

$$\Delta\theta_S = \alpha \,.\, \Delta\theta_R \tag{22}$$

Definimos para tanto o valor de α de modo controlar a variação das variáveis de controle do robô real.

Idealmente para garantirmos que a simulação e o controle ocorram de forma simultânea, seria necessários sensores de posicionamento com uma malha fechada de controle para levar o erro entre este processo a zero. Isso pois, mesmo que os atuadores de motores de passo sejam acionados de forma discreta, não há garantia que não ocorrerá eventos de não linearidade no motor real. Eventos de escorregamento, mesmo que pequenos, para programações longas do robô podem ter grandes efeitos de dispersão entre simulação e controle.

Utilizando as equações desenvolvidas no capítulo 3 é possível definir o valor de α, realizar o controle do robô real. Dois pontos são importantes para esta abordagem:

- O valor de alfa não necessariamente associará uma quantidade discreta de passos para a variação angular da simulação, e dada a impossibilidade da realização de passos discretos, isso pode provocar um erro sistemático que deve ser abordado. Os passos

gerados pelo robô criam um fator de correção da soma da diferença da discretização dos passos, de modo que para longas simulações não há acumulo deste erro.

- O software de simulação se mostrou computacionalmente demandante, isto é, demandas grandes valores de memória e processamento. Isso faz com que o controle do tempo de acionamento seja de difícil determinação, o que é extremamente necessário caso queiramos realizar o controle da velocidade do robô. Uma solução possível para este problema, é a paralelização do módulo de controle, facilitando a determinação das pausas do acionamento. Esta solução ainda não foi implementada, devido a uma dificuldade em paralelizar o código criado.

7 TESTES REALIZADOS

Neste capítulo são apresentados um resumo dos testes realizados no software desenvolvido. São necessários diferentes tipos de teste para averiguar o funcionamento do software de desenvolvido, como a simulação, o controle e a simultaneidade deste dois processos.

Devido a não disponibilidade para este tipo de teste do robô utilizado para o desenvolvimento do trabalho até este momento, montou-se um sistema análogo em termos de cinemática direta e inversa, para que os testes necessários fossem documentados. Este sistema possui as mesmas equações trabalhadas no Capítulo 3, variando entretanto o comprimento de seus elos.

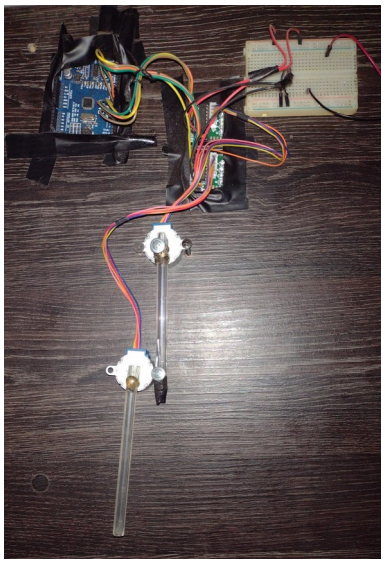

Figura 29: Sistema utilizado para a realização dos testes

Como o acionamento do robô educacional 1 é feito pelos códigos descritos na Tabela 6, podemos acionar um robô diferente simplesmente alterando o fator α descrito no subcapítulo 6.7.

Dado que a alteração não é sistemática, mas pontual, as alterações aplicadas no software vão possibilitar o controle de um do sistema de teste, apesar das diferenças com o robô idealizado.

7.1 Simulação nos limites do volume de trabalho

Neste teste é analisado a capacidade do simulador em mostrar movimentação próxima ao limite do seu volume de trabalho. A programação utilizado no simulador é apresentada a seguir:

```
#PROGRAMA DE TESTE 1
#TESTE DE LIMITES DURANTE SIMULAÇÃO
move 90 90 0 #limites positivos das variáveis de controle
move -90 -90 0 #limites negativos das variáveis de controle
down
move 90 90 160 #intercalação de limites positivo/negativo
move 0 0 160 #intercalção de limite negativo/positivo
home
```

A simulação gerou os limites do volume de trabalho do robô como esperado. Esta limitação indica que o robô não possui os problemas comumente associados aos limites do volume de trabalho do robô. Indicando que a simulação se comporta da forma para a qual ela foi idealizada:

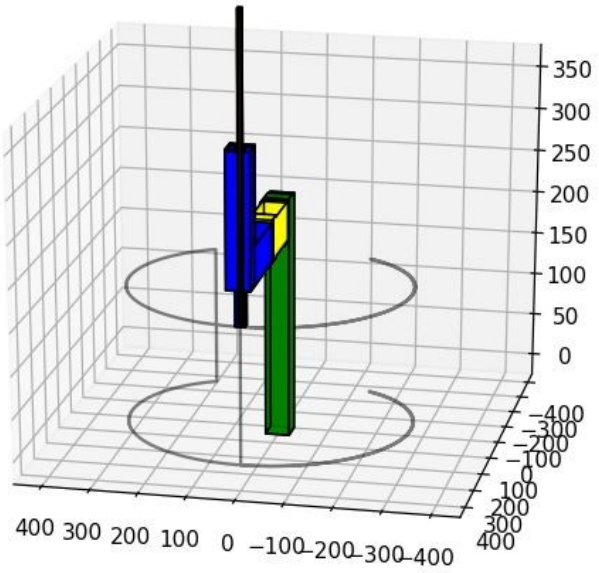

Figura 30: Trajetória pra o teste 1

7.2 Simulação de trajetória composta

Neste teste são realizadas movimentações para um mesmo ponto do volume de trabalho, com diferentes funções de movimentação. Isto é necessário pois procuramos pontos de erro de repetitividade na simulação. A programação utilizada para a simulação é apresentado a seguir:

```
#PROGRAMA DE TESTE 2
#TESTE DE TRAJETÓRIA COMPOSTAS (variação de funções de movimentação)
#teta1 = 45
#teta2 = 70
#indo para o ponto
pmove 63.87 249.92 0
home
#indo para o ponto em linha reta
lmove 63.87 249.92 0
home
#indo para o ponto
pmove 63.87 -249.92 0
home
#indo para o ponto em linha reta
```

lmove 63.87 -249.92 0
home
*#140*cos(45+45)+174*cos(45)+65 = 123.04*
*#140*sin(45+45)+174*sin(45) = 263.04*

Observou-se que simulador mantêm as características para diferentes funções, nos dois semiplanos que o robô atua. Observou-se na formação da trajetória gerada, que a repetibilidade é conseguida, dado que as trajetórias se sobrepõem para as diferentes execuções do programa. Esta característica é esperada, dado que há uma conservabilidade entre as transformações durante a simulação.

Fica evidente que a trajetória levou para o mesmo ponto utilizando as duas movimentações (pmove e lmove). A aproximação utilizada para geração da trajetória linear faz com que a mesma possua níveis de ruído fora o ideal, entretanto mantendo sua característica principal de linearidade. A imagem a seguir apresenta o resultado obtido:

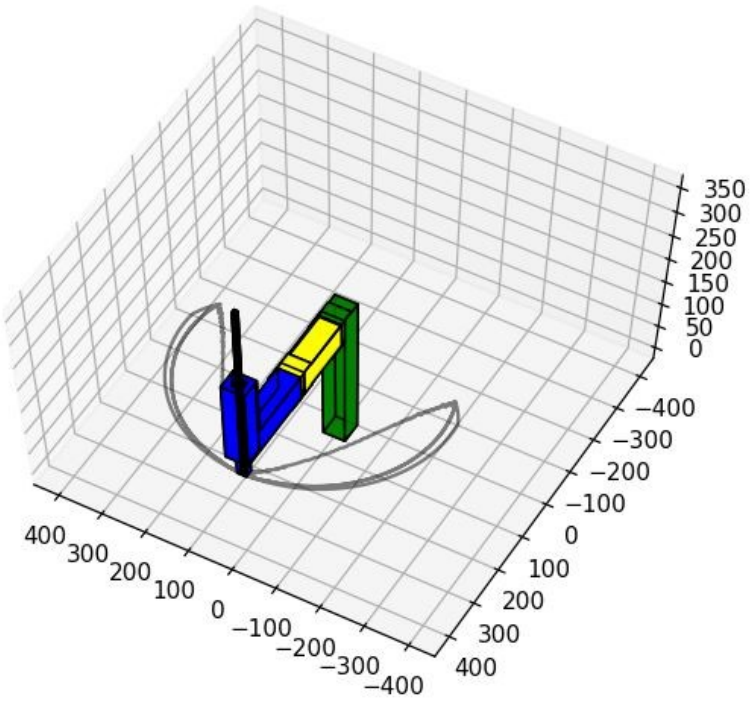

Figura 31: Trajetória gerada para o segundo teste

7.3 Simulação e Controle de rotas no limite do volume de trabalho

Neste teste são observadas a capacidade de simular e controlar o robô nos seus limites do volume de trabalho. O primeiro teste se difere deste, pois queremos isolar a influência da simulação e do controle. O programa utilizado é apresentado a seguir:

```
#PROGRAMA DE TESTE 3
#TESTE DE LIMITES DURANTE SIMULAÇÃO/CONTROLE
#TESTE EQUIVALENTE AO TESTE 1, PORÉM COM  CONTROLE SIMULTÂNEO
move 90 90 0 #limites positivos das variáveis de controle
move -90 -90 0 #limites negativos das variáveis de controle
zmove -260
move 90 -90 -200 #intercalação de limites positivo/negativo
move -90 90 -200 #intercalção de limite negativo/positivo
zmove 0 #análise de variações em Z.
home
```

Dado que definimos os testes somente para o plano de trabalho do robô, e que a trajetória simulada já foi apresentada na Figura 30, na figura a seguir têm-se a trajetória gerada pelo robô para este mesmo programa:

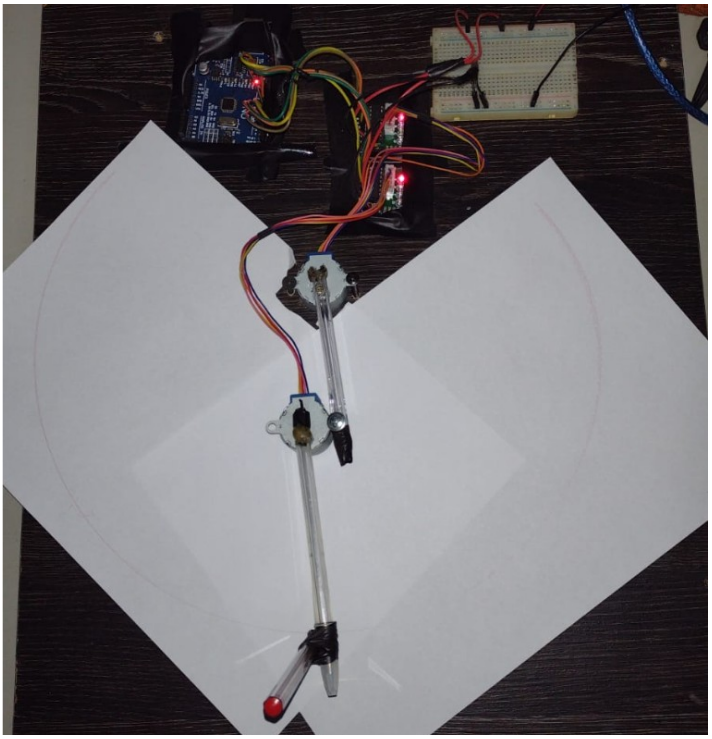

Figura 32: Trajetória criada no teste 3

Observa-se que a figura circular é gerada como esperado, caracterizando a relação esperada entre simulação e controle. Há entretanto os erros associados ao controle de posicionamento de elos reais. A Figura 33 mostra em detalhes uma região da trajetória criada, de modo que podemos observar a influência das não linearidades do robô real. Devido as folgas presentes no sistema de teste, as diferentes movimentações nesta região tiveram uma variação de até 3 mm.

Uma observação também importante para a trajetória apresentada na Figura 33, é a fácil observação das mudanças de passo na trajetória criada. Uma movimentação que primordialmente deveria ser contínua, é marcada pela mudança de passos do sistema de teste.

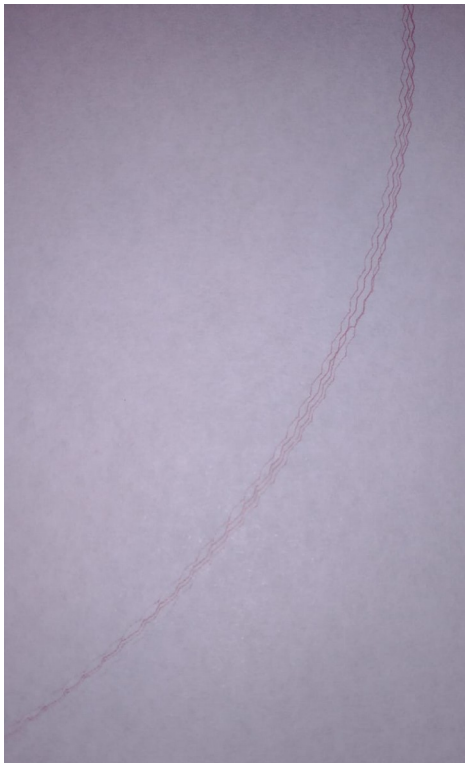

Figura 33: Ampliação da trajetória gerada no teste 3

7.4 Análise da confiabilidade da simulação

Busca-se observar com este teste se há boa confiabilidade da simulação em relação o controle. Desta forma, queremos definir se é possível a programação do robô somente

utilizando a simulação, e os resultados do controle do robô serão encontradas. Segue o programa utilizado:

```
#PROGRAMA DE TESTE 4
#TESTE DE TRAJETÓRIA COMPOSTAS (variação de funções de movimentação)
#teta1 = 45
#teta2 = 70
#indo para o ponto
pmove 63.87 249.92 0
home
#indo para o ponto em linha reta
lmove 63.87 249.92 0
home
```

Utilizando um programa com os mesmos pontos apresentados no teste 2, faz-se nesse ponto que o sistema de teste também receba os sinais do sistema de controle. Como observou-se que a simulação ocorre da forma esperada para os hemisférios positivo e negativo, neste programa há somente a simulação/controle em um hemisfério. A Figura 34 apresenta o resultado obtido:

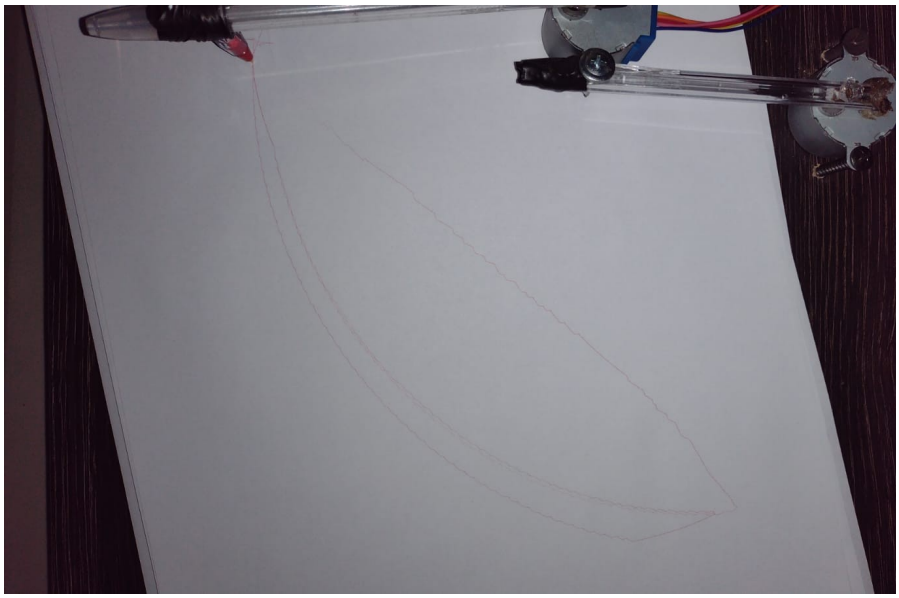

Figura 34: Trajetória gerada para o teste4

Como encontrado no teste anterior 7.3, há uma perda de repetibilidade do sistema de testes de aproximadamente 3 mm, de modo que o resultado obtido esta contido nesta margem de erro. Observou-se entretanto um erro associado ao início e fim da trajetória linear gerada, o que é um forte indício de uma alta histerese do sistema. Para mensurar este erro, traçou-se

uma linha entre os pontos iniciais e finais da movimentação ideal do sistema, representada pelo ponto gerado pela primeira movimentação do programa atual, utilizando a função *pmove*.

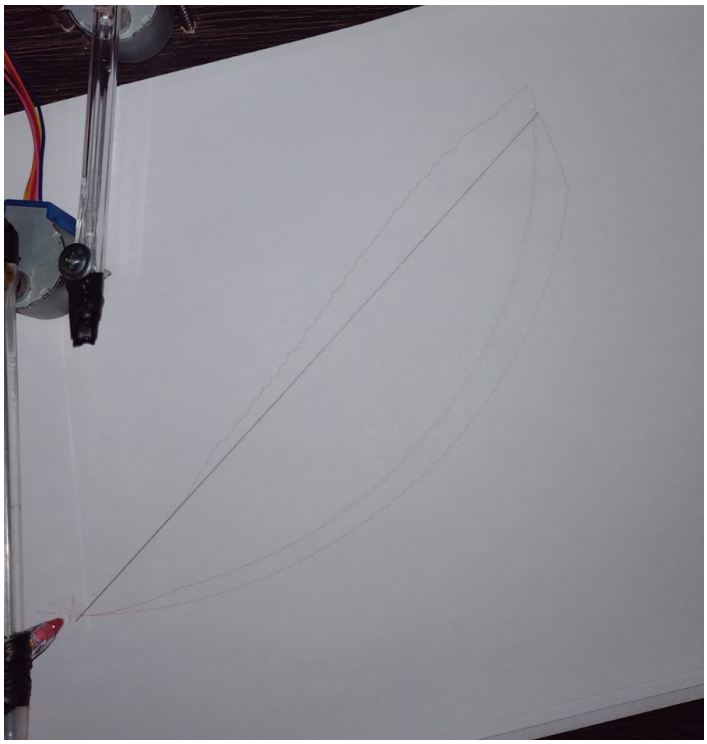

Figura 35: Estimativa visual do erro gerado na trajetória 4

O maior erro observado se encontra próximo ao fim da trajetória, de modo que é possível admitir que há um acumulo do erro. O erro máximo de 6 mm no sistema de testes torna questionável a capacidade do nosso software em controlar o sistema real. Entretanto como o erro associado a criação de trajetórias circulares, menos suscetíveis a erros, nos diz que o erro está associado ao sistema em si, e não ao controlador. Essa afirmação é mantida se observa-se a trajetória gerada no simulador e apresentada na Figura 36, idêntica a trajetória gerada, se eliminarmos o erro na trajetória linear:

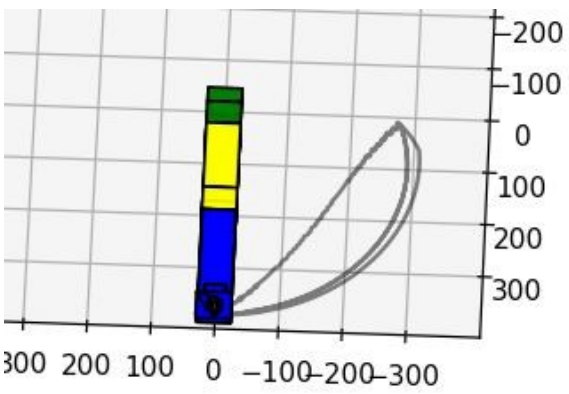

Figura 36: Trajetória simulada para o teste 4

7.5 Criação de trajetória complexa

Com o objetivo de confirmarmos a possibilidade da utilização deste software em aplicações reais, são simuladas as criações de trajetórias complexas. Isto pois é necessário vermos como os erros anterioemente constatados influenciam a repetibilidade do sistema de testes, e a capacidade do robô em si. Segue programa utilizado:

```
#PROGRAMA DE TESTE 5
#TESTE DE TRAJETÓRIA COMPOSTAS (variação de funções de movimentação)
move 45 45 0
home
pmove 94 273 0
home
lmove
move 60 60 0
home
pmove 17 272 0
home
lmove 17 272 0
home
```

Observa-se na Figura 37 que para trajetórias maiores os erros anteriores se mantém, dado que estão associados a não linearidades do sistema de testes em si. Entretanto observa-se que o sistema possui uma boa repetibilidade, indicando que o erro do sistema de testes possui uma direção preferencial, positiva para o atuador 1.

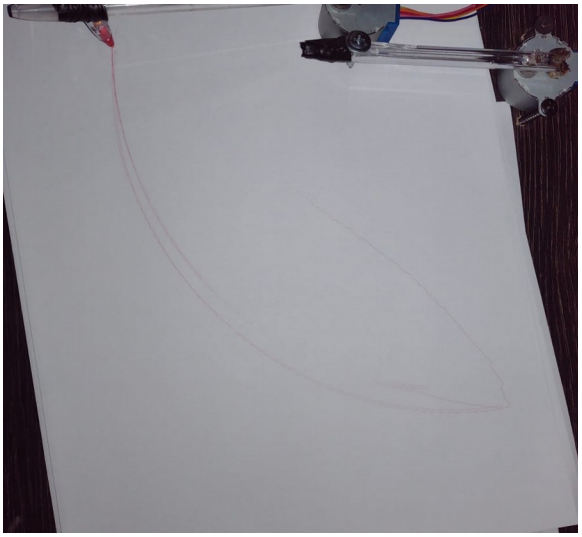

Figura 37: Trajetória gerada para o teste 5

Esta determinação é importante pois o objetivo primordial deste trabalho é o desenvolvimento de um software de simulação e controle de um robô manipulador. Erros associados ao sistema de teste, apesar de serem melhorias obvias, não configuram o escopo deste desenvolvimento.

Dado este contexto, uma boa característica qualitativa do controle realizado é a relação entre a simulação desta trajetória com a simulação. Comparando portanto a Figura 37 e a Figura 38 vemos sua semelhança, abstraindo as linearidades já descritas.

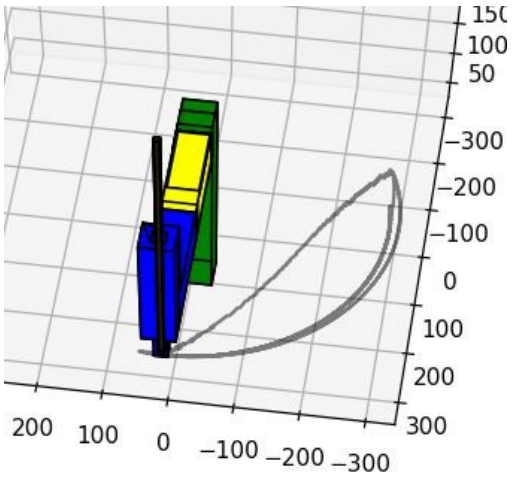

Figura 38: Trajetória simulada para o teste 5

8 CONCLUSÃO

Com as premissas propostas desenvolveu-se um software com capacidade para simulação e programação *off-line* do robô de trabalho proposto. Essa programação é feita pela linguagem de programação discutida e desenvolvida durante o trabalho, direcionando uma codificação das instruções para um construtor gráfico que realiza a movimentação do robô simulado.

As ferramentas propostas foram assertivas na solução do problema a qual elas foram escolhidas. Seja pela simplicidade do desenvolvimento nestas ferramentas específicas, ou pela possibilidade de integração entre as ferramentas escolhidas. Em termos de capacidade de processamento, nenhuma ferramenta obteve baixo desempenho na entrega da solução que se propunha.

O compilador proposto e construído atende aos requisitos de programação do robô, dada as devidas limitações das instruções propostas. Espera-se entretanto que o acréscimo de outras instruções aumentem a produtividade da programação do robô de trabalho, sendo um ponto futuro de desenvolvimento deste escrito. Assim como o acréscimo de análises de trajetória que otimizem a movimentação do robô em seu ambiente de trabalho.

A utilização do par FreeCAD e Matplotlib foi suficiente para a construção gráfica do robô proposto na interface com o usuário. Assim como a função de animação facilitou o desenvolvimento da movimentação do robô, além de acrescentar os métodos de visualização do mesmo em um ambiente gráfico. Observou-se entretanto, erros associados ao algoritmo de exportação dos dados gráficos do robô, ocasionando em erros em sua construção. Não chegou-se a uma solução para estes erros, em parte aos mesmos ocorrerem em pontos de complexificação da estrutura do robô, sendo a solução deste impasse uma proposta de melhoria.

Durante a programação do robô observou-se uma dificuldade em associar o volume de trabalho simulado do robô, ao volume de trabalho do robô real. Isto se deve em parte a falta de referências entre estes espaços, dado que única referência visual entre ambos é a própria base do robô. Uma solução seria o acréscimo junto ao robô real um sistema de referência, talvez um papel milimetrado, que auxilie a programação *off-line* do robô, dado que os pontos são facilmente alcançáveis no simulador.

Ainda na associação entre o volume de trabalho simulado e o real, uma solução para descoberta de pontos não específicos (como por exemplo, pontos associados de forma visual), o controle do robô simulado por meio ferramentas mecânicas pode facilitar a associação visual na simulação. Uma proposta portanto pode ser o controle com uma ferramenta específica, como um *touchpad* ou *joystick*, o que dependeria da disponibilidade destas ferramentas, ou a utilização dos próprios meios de entrada do computador que roda o software (mouse e teclado), para este controle direto do robô. Sendo mais uma possibilidade de desenvolvimento futuro.

Testes com o software finalizado transparecem as características de funcionamento apresentadas anteriormente, tanto para seu funcionamento ou ausência de funcionalidades. Apesar de o software ser capaz de simular e controlar separadamente o robô manipulador proposto, para seu funcionamento como plataforma dedicada são necessárias otimizações no manejo do processamento de tempo real. Além de uma carência de funções embutidas para análise das trajetórias criadas, ou mesmo otimização das mesmas.

9 BIBLIOGRAFIA

Asea Brown Boveri Robotics. ABB. Operating manual RobotStudio. 2008. Disponível em: https://library.e.abb.com/public/244a8a5c10ef8875c1257b4b0052193c/3HAC032104-001_revD_en.pdf. Acesso em: Novembro de 2020.

ATMEL. “ATmega 328P: 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash”. 2021. Disponível em: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf. Acesso em: Outubro de 2021.

CARRARA, Valdemir. Introdução à Robótica Industrial. Instituto Nacional de Pesquisas Espaciais: São José dos Campos, 2015. Disponível em: http://urlib.net/8JMKD3MGP3W34P/3K5JPL8. Acesso em: Novembro de 2020.

CARRASQUINHO, Felipe Correa. Ferramenta de Simulação para Robô Industrial. 2015. 81p. Dissertação (Mestrado Engenharia Eletrotécnica e de Computadores) – Faculdade de Ciências e Tecnologias, Universidade Nova de Lisboa. Lisboa, Portugal, 2015. Disponível em: https://run.unl.pt/handle/10362/16561. Acesso em: Novembro de 2020.

FERREIRA, Nuno Miguel Fonseca. Simulação Dinâmica e Controle de Robôs Industriais. 1999. 193p. Dissertação (Mestre em Engenharia Eletrotécnica e de Computadores)- Faculdade de Engenharia, Universidade do Porto, Porto, Portugal, 1999. Disponível em: https://repositorio-aberto.up.pt/bitstream/10216/11453/2/Texto%20integral.pdf. Acesso em: Novembro de 2020.

FILGUEIRAS, Yuri de Amorim Gaspar. Simulação e Controle de um Manipulador Robótico. 2019. 81p. Trabalho Conclusão de Curso (Engenharia Elétrica) – Centro de Tecnologia, Universidade Federal do Ceará, Fortaleza, Ceará, 2018. Disponível em: http://repositorio.ufc.br/handle/riufc/49931. Acesso em: Novembro de 2020.

GAVIRA, Muriel de Oliveira. Simulação Computacional como uma Ferramenta de Aquisição de Conhecimento. 2003. 163p. Dissertação (Mestre em Engenharia de Produção) - Escola de Engenharia de São Carlos, Universidade de São Paulo. São Carlos, São Paulo, 2003. Disponível em: https://teses.usp.br/teses/disponiveis/18/18140/tde-20052003-004345/publico/Gavira1.pdf. Acesso em: Novembro de 2020.

HENRIQUES, R.V.B., DUTRA, M.S., ROSARIO, J.M., PEREIRA, C.E., LAGES, W.F., COSTA, A.H.R. (Org). ROBÓTICA INDUSTRIAL: Aplicação na Indústria de Manufatura e de Processos. Capítulo 6. 1.ed. São Paulo: Edgard Blücher, 2002. Disponível em: http://www.fem.unicamp.br/~hermini/Robotica/livro/cap.6.pdf. Acesso em: Novembro de 2020.

HOLZER, Raphael.PyGame Tutorial Documentation. 2019. Disponível em: https://buildmedia.readthedocs.org/media/pdf/pygame/latest/pygame.pdf. Acesso em: Novembro de 2021.

HOSS, Alessandro; HOUNSELL, Marcelo da Silva; LEAL, André Bittencourt. Um ambiente de programação para um simulador de robô manipulador. 2014. Universidade do Estado de Santa Catarina (UDESC), Joinville, Santa Catarina, 2014. Disponível em: https://larva.joinville.udesc.br/portal/uploads/publicacoes/VirBot4u.pdf. Acesso em: Novembro de 2020.

HUNTER, John; et all. Matplotlib: visualization with python. 2021. Disponível em: https://matplotlib.org/. Acesso em: Novembro de 2021.

Mark Cardwell. Quick Makeover: L'Oréal Canada needed to set up a packaging line in Montreal for hair coloring products. ABB Robotics, Estudo de Caso. Disponível em: http://search.abb.com/library/Download.aspx?DocumentID=9AKK105152A9809&LanguageCode=en&DocumentPartId=&Action=Launch. Acesso em: Novembro de 2020.

MELLO, O. P. M.. Desenvolvimento de um Robô Manipulador SCARA.2016. 122p. Trabalho de Graduação (Engenharia de Controle e Automação) - Faculdade de Tecnologia, Universidade de Brasília, Brasília, Distrito Federal, 2016. Disponível em: https://bdm.unb.br/bitstream/10483/14934/1/2016_OtavioPellicanoMoreiraDeMello_tcc.pdf. Acesso em: Novembro de 2020.

PARALLEMIC. RoKiSim 1.7: Robot Kinematics Simulator, 2015. Disponível em: https://www.parallemic.org/RoKiSim.html. Acesso em: Novembro de 2020.

PONTES, Heráclito Lopes Jaguaribe. Desenvolvimento de um Ambiente de simulação da Manufatura baseado em Features e Realidade Virtual. 2012. 301 f. Tese (Doutor em Engenharia Mecânica) – Escola de Engenharia de São Carlos, Universidade de São Paulo. São Carlos, São Paulo, 2012. Disponível em: http://www.teses.usp.br/teses/disponiveis/18/18145/tde-10122012-111828/. Acesso em Novembro de 2020.

ROCHA, Edrysson Sebastien Araujo Análise de desempenho e controle de um robô SCARA. 2016. 67p. Trabalho de Graduação (Engenharia de Controle e Automação), Faculdade de Tecnologia, Universidade de Brasília, Brasília, Distrito Federal, 123p. Disponível em: https://bdm.unb.br/bitstream/10483/17132/1/2016_EdryssonSebastienRocha_tcc.pdf. Acesso em: Novembro de 2020.

SAHA, Subir Kumar. Robo Analyzer: User Manual. 2012. Disponível em: http://www.roboanalyzer.com/uploads/2/5/8/8/2588919/roboanalyzerusermanual.pdf. Acesso em: Novembro de 2020.

SOUSA, Artur Farias; GARCIA, Marco Aurélio; CRUZ, Sabrina Melo de; SANTOS, Richard Vieira dos Santos. ROBÓTICA: Conceitualização e Linguagens de Programação. Revista Conexão Eletrônica – Três Lagoas, MS.Volume 15, Nº 1, 10p, 2018. Disponível em: http://revistaconexao.aems.edu.br/wp-content/plugins/download-attachments/includes/download.php?id=1882. Acesso em: Novembro de 2020.

ST ELECTRONICS. "L298: Dual Full Bridge Driver", 2021. Disponível em: https://www.filipeflop.com/img/files/download/Datasheet_L298_Ponte_H.pdf. Acesso em: Setembro de 2021.

UC Davis C-STEM Center. RoboSim Manual. 2020. Disponível em: https://c-stem.ucdavis.edu/studio/robosim/. Acesso em: Novembro de 2020.

TAYLOR, R. H.; SUMMERS, P. D.; MEYER, J. M. AML: A Manufacturing Language.Watson Research Center Yorktown Heights New York, EUA, 1982. Disponível em: https://journals.sagepub.com/doi/abs/10.1177/027836498200100302. Acesso em: Novembro de 2020.

10 ANEXO

10.1 Função principal do software

```
# -*- coding: utf-8 -*-
"""
@author: Victor Hugo Marques
Modelo 17
"""
#IMPORTAR ARQUIVOS E FONTES
from tkinter import *
from tkinter import filedialog, ttk
from tkinter import scrolledtext

import os

import MenuCreator.MenuFunctions

import Graphics.GraphicsBase

import Compilador.CompilateBase

class Janela:
    def __init__(self, instancia_de_Tk):
        instancia_de_Tk.title('RSAC: Robot Simulation Analyze and Control')
        instancia_de_Tk.geometry('1366x768')
        instancia_de_Tk.state('zoomed')

        principal=Menu(instancia_de_Tk)
        #Itens para FILE
        Files=Menu(principal)
        Files.add_command(label="New",command=self.New)
        Files.add_command(label="Open",command=self.Open)
```

```
Files.add_command(label="Save",command=self.Save)
Files.add_command(label="Close",command=instancia_de_Tk.destroy)
principal.add_cascade(label="Files",menu=Files)

#Itens para TOOLS
Tools=Menu(principal)
Tools.add_command(label="Debug",command=self.Debug)
Tools.add_command(label="Compilate",command=self.Compilate)
Tools.add_command(label="Simulate",command=self.Simulate)
Tools.add_command(label="Control",command=self.Control)
principal.add_cascade(label="Tools",menu=Tools)

#Itens para OPTIONS
Options=Menu(principal)
Options.add_command(label="Serial Port",command=self.SerialPort)
Options.add_command(label="Simulation Test",command=self.SimulationTest)
Options.add_command(label="Control Test",command=self.ControlTest)
principal.add_cascade(label="Options",menu=Options)

#Item para HELP
principal.add_command(label="Help",command=self.Help)

#SETANDO AS CONFIGURAÇÕES DE MENU
instancia_de_Tk.configure(menu=principal)

###########################################################################
#CAIXA DE TEXTO
self.tabs = {'ky': 0}
#Keep a record of the open tabs in a list.
self.tab_list = []
self.notebook = ttk.Notebook(instancia_de_Tk)
self.notebook.pack(side = LEFT,anchor = "w" ,expand = True, fill= 'both')
```

```
    ##########################################################################
    #CAIXA PARA DEMONSTRAÇÃO DE ERRO
    self.ErroBox = MenuCreator.MenuFunctions.ErroBoxHandle(instancia_de_Tk)

    ##########################################################################
    #VISÃO DO ROBÔ
    self.graphicsVar = Graphics.GraphicsBase.GraphicsFunctions(instancia_de_Tk)
    ##########################################################################
    #objeto interna de compilação
    self.CompilateText = Compilador.CompilateBase.CompilateFunction(instancia_de_Tk )
##############################################################################
#DEFINIÇÕES DO ITEM FILE
def New(self):
    textvalue = ""

    if self.tabs['ky'] < 20:
        self.tabs['ky'] += 1
        self.add_tab('Document' + str(self.tabs['ky']), textvalue)
    #seleciono para visão o arquivo novo
    self.notebook.select(self.notebook.index('end')-1)

def add_tab(self, name, textvalue):
    tab = MenuCreator.MenuFunctions.Tab(self.notebook, name , textvalue)
    self.notebook.add(tab, text=name)
    self.tab_list.append(tab)

def get_tab(self):
    #pega a tab da tablist
    tab = self.tab_list[self.notebook.index('current')]
    return tab

def Open(self):
    file = open(filedialog.askopenfilename(), 'r+')
```

```
            self.text_value = file.read()
            title = os.path.basename(file.name)
            self.add_tab(title, self.text_value ) #crio uma nova tab para o arquivo aberto
            self.notebook.select( self.notebook.index('end')-1)

            file.close()

        def Save(self) :
            tab_to_save = self.get_tab()

            self.tab_list[self.notebook.index('current')].name = tab_to_save.save_tab()
                                        self.notebook.tab( self.notebook.index('current') , text =
self.tab_list[self.notebook.index('current')].name )



        ##########################################################################
        #DEFINIÇÕES DO ITEM TOOLS
        def Debug(self):
            print("Debug")

        def Compilate(self):
            #salvo para garantir que nao tera mudanças
            #imprimo mensagem de salve
                                    self.ErroBox.ErroTextValue = ("Deseja salvar o arquivo
"+self.tab_list[self.notebook.index('current')].name+":")
            self.ErroBox.ErroBoxAtualiza()
            self.Save()

            #imprimo mensagem de compilação
              self.ErroBox.ErroTextValue = ("Arquivo "+self.tab_list[self.notebook.index('current')].name+" salvo
com sucesso.")
            self.ErroBox.ErroBoxAtualiza()
```

```
            #compilando arquivo
            #mensagem de compilação
                                        self.ErroBox.ErroTextValue = ("Compilando o arquivo
"+self.tab_list[self.notebook.index('current')].name+ ".")
            self.ErroBox.ErroBoxAtualiza()

            #declaro novo objeto ou atualizo para conter o esperado da compilação
            self.CompilateText.InicioCompilacao(self.tab_list[self.notebook.index('current')].name)

            #imprimo mensagem de compilação
            self.ErroBox.ErroTextValue = ("Arquivo compilado com sucesso.\n")
            self.ErroBox.ErroBoxAtualiza()


        def Simulate(self) :
            self.ErroBox.ErroTextValue = "Preparando movimentação do robo."
            self.ErroBox.ErroBoxAtualiza()

            self.ErroBox.ErroTextValue = "Robo em movimento..."
            self.ErroBox.ErroBoxAtualiza()

            self.graphicsVar.move_handle_funtion(self.CompilateText.list_lines_comp)

            if self.graphicsVar.ani_state == False:
                self.ErroBox.ErroTextValue = "Movimento finalizado.\n"
                self.ErroBox.ErroBoxAtualiza()


        def Control(self):
            #metodologia de impressão de erro
             self.ErroBox.ErroTextValue = "Teste de funcionamento do Controle que sera desenvolvido no TCC2.\
n"
```

```
            self.ErroBox.ErroBoxAtualiza()


        ##########################################################################
        #DEFINIÇÕES DO ITEM OPTIONS
        def SerialPort(self):
            print("Serial Port")

        def SimulationTest(self):
            print("SimulationTest")

        def ControlTest(self):
            print("ControlTest")
            self.tab_list[0].Tab.textWidget()

        ##########################################################################
        def Help(self) :
             self.ErroBox.ErroTextValue = ("You can read the manual contacting:\n  victorh_delete@hotmail.com\
n")
            self.ErroBox.ErroBoxAtualiza()


    raiz = Tk()
    Janela(raiz)
    raiz.mainloop()
    #top.mainloop()
```

10.2 Cinemática Inversa

```
import numpy as np
#função para cálculo da cinematica direta baseado no meu robo de trabalho
def cinematica_direta (t1, t2):
    x = 140*np.cos((t1+t2)*(np.pi/180)) + 174*np.cos((t1)*(np.pi/180))
    y = 140*np.sin((t1+t2)*(np.pi/180)) + 174*np.sin((t1)*(np.pi/180))
    return x,y
```

```
#função para cálculo da cinemática inversa do meu robô de trabalho.
def cinematica_inversa(x, y):
    #eu utilizei o método geométrico já que o robô é bem simples.
    #calculo do segundo ângulo (ainda em radianos)
    t2 = np.arccos((pow(x,2)+ pow(y,2) - pow(174,2) - pow(140,2)) / (2*140*174))

    #caso seja semiplano negativo de y uso teta2 negativo
    if y<0:
        t2 = -abs(t2)

    #calculo um valor possível para teta1 e testo para saber se é válido
    num = y*(174+140*np.cos(t2) )  - x*140*np.sin(t2)
    den = x*(174+140*np.cos(t2) )  + y*140*np.sin(t2)

    #temp1 = pow(x,2) + pow(y,2) + pow(174,2)-pow(140,2)
    #temp2 = pow(x,2) + pow(y,2) - pow(174,2)-pow(140,2)

    #num = y*(temp1)- x*pow( 4*pow(140,2)*pow(174,2) - pow(temp2,2), 0.5)
    #den = x*(temp1)+ y*pow( 4*pow(140,2)*pow(174,2) - pow(temp2,2), 0.5)

    t1 = np.arctan(num/den) #aproximação para evitar divisão por zero.

    #passo para graus
    t1 = t1*(180/np.pi)
    t2 = t2*(180/np.pi)

    return t1,t2
```

10.3 Construção gráfica

```
from tkinter import *
```

```
import numpy as np

from matplotlib.figure import Figure

from matplotlib.backend_bases import key_press_handler

import matplotlib.animation as animation
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection, Line3DCollection
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg, NavigationToolbar2Tk )

import serial
import time

import VisualData.Peca1_dados
import VisualData.Peca2_dados
import VisualData.Peca3_dados
import VisualData.Peca4_dados


class GraphicsFunctions:
    def __init__(self, instancia_de_Tk):
        #angulo de variação da movimentação, via de regra é 1 mas acrescenta-se dessa forma para pode renderizar em variações
        self.lista_instrucoes = []

        #maiores de angulo
        vel = 0.5
        self.delta1 = 1.0*vel
        self.delta2 = 1.0*vel
        self.delta3 = 1*2

        self.trajetoriaX = []
```

```
self.trajetoriaY = []
self.trajetoriaZ = []

self.traj_list = [[0, 400, 165]]

#variável que define a posição atual das juntas do robô
self.teta1_atual = 0.0
self.teta2_atual = 0.0
self.altura3_atual = 0.0

#variável que define a posição futura das juntas do robô
self.teta1_novo = 0.0
self.teta2_novo = 0.0
self.altura3_novo = 0.0

self.ref_ponto1 = np.array([0.0,65.0,280.0])
self.ref_ponto2 = np.array([0.0,230.0,280.0])

##############################################

self.fig = Figure(figsize=(8,5), dpi=110)

self.canvas_robo = FigureCanvasTkAgg(self.fig, master = instancia_de_Tk)  # A tk.DrawingArea.
self.canvas_robo.draw()

self.ax = self.fig.add_subplot(111, projection="3d")

self.ax.add_collection3d(Poly3DCollection(peca4_verts,
 facecolors='orange', linewidths=1, edgecolors='k', alpha=.2))

self.ax.add_collection3d(Poly3DCollection(peca3_verts,
 facecolors='blue', linewidths=1, edgecolors='k', alpha=.9))
```

```
self.ax.add_collection3d(Poly3DCollection(peca2_verts,
 facecolors='yellow', linewidths=1, edgecolors='k', alpha=.9))

self.ax.add_collection3d(Poly3DCollection(peca1_verts,
 facecolors='green', linewidths=1, edgecolors='k', alpha=.9))

"""
########################
#rotacionando eixos para testes
#movento o braço 1 em relação ao ponto fixo [0,65,280] self.ref_ponto1
for i in range(len(peca1_verts)):
    for j in range(len(peca1_verts[i])):
        peca1_verts[i][j] = peca1_verts[i][j] - self.ref_ponto1
        peca1_verts[i][j] = self.rotacionando(peca1_verts[i][j], -90)
        peca1_verts[i][j] = peca1_verts[i][j] + self.ref_ponto1

for i in range(len(peca2_verts)):
    for j in range(len(peca2_verts[i])):
        peca2_verts[i][j] = peca2_verts[i][j] - self.ref_ponto1
        peca2_verts[i][j] = self.rotacionando(peca2_verts[i][j], -90)
        peca2_verts[i][j] = peca2_verts[i][j] + self.ref_ponto1

#movento o braço 1 em relação ao ponto fixo [0,65,280]
for i in range(len(peca3_verts)):
    for j in range(len(peca3_verts[i])):
        peca3_verts[i][j] = peca3_verts[i][j] - self.ref_ponto1


        peca3_verts[i][j] = self.rotacionando(peca3_verts[i][j], -90)


        peca3_verts[i][j] = peca3_verts[i][j] + self.ref_ponto1

#movento o braço 1 em relação ao ponto fixo [0,65,280]
for i in range(len(peca4_verts)):
```

```
            for j in range(len(peca4_verts[i])):
                peca4_verts[i][j] = peca4_verts[i][j] - self.ref_ponto1
                peca4_verts[i][j] = self.rotacionando(peca4_verts[i][j], -90)
                peca4_verts[i][j] = peca4_verts[i][j] + self.ref_ponto1

        ######################
        """


        self.canvas_robo.get_tk_widget().pack(side=TOP , fill=BOTH, expand=1)

        self.ani = animation.FuncAnimation(self.fig, self.animate, interval=2)

        self.ani.event_source.stop() #movimentação inicia parada

        self.ani_state = False

        #######self.porta_serial = serial.Serial('COM6', 9600)

    def moving_stop(self):
        self.ani.event_source.stop()
        self.ani_state = False

    def moving_start(self):
        self.ani.event_source.start()
        self.ani_state = True

    #def move_handle_funtion(self, move_kind, new_point):
    def move_handle_funtion(self, lista_instrucoes):

        self.lista_instrucoes = lista_instrucoes

        self.InterpretaInit()
```

```
        #rearranjo o novos pontos de movimentação
        #inicio a movimentação
        self.moving_start()


##########################################################################

    #função para animar
    def animate(self, i ):
        #eixos a serem movimentados
        movimento1 = 0
        movimento2 = 0
        movimento3 = 0

        time_delay = 0.001

        temp = [[]]

        #se eu estiver com movimento acionado para 1
        if self.teta1_novo != self.teta1_atual:
            if self.teta1_novo > self.teta1_atual:
                self.teta1_atual += abs(self.delta1) #adiciono a varação minima de teta1
                self.delta1 = abs(self.delta1)       #a variação é positiva

            if self.teta1_novo < self.teta1_atual:
                self.teta1_atual -= abs(self.delta1) #retiro a varação minima de teta1
                self.delta1 = (-1)*abs(self.delta1)       #a variação é negativa
            movimento1 = 1

        if self.teta2_novo != self.teta2_atual:
            if self.teta2_novo > self.teta2_atual:
                self.teta2_atual += abs(self.delta2) #adiciono a varação minima de teta2
```

```
        self.delta2 = abs(self.delta2)       #a variação é positiva

    if self.teta2_novo < self.teta2_atual:
        self.teta2_atual -= abs(self.delta2) #retiro a varação minima de teta2
        self.delta2 = (-1)*abs(self.delta2)       #a variação é negativa
    movimento2 = 1

if self.altura3_novo != self.altura3_atual:
    if self.altura3_novo > self.altura3_atual:
        self.altura3_atual += abs(self.delta3) #adiciono a varação minima de altura3
        self.delta3 = abs(self.delta3)       #a variação é positiva

    if self.altura3_novo < self.altura3_atual:
        self.altura3_atual -= abs(self.delta3) #retiro a varação minima de altura3
        self.delta3 = (-1)*abs(self.delta3)       #a variação é negativa
    movimento3 = 1

#se acabar minhas instruções a serem executadas
if len(self.lista_instrucoes) == 0:
    self.moving_stop()

#analiso se tem movimento da instrução anterior
if movimento1 == 0 and movimento2 == 0 and movimento3 == 0 and len(self.lista_instrucoes) != 0:
    #excluo o primeiro item de instrução
    self.lista_instrucoes = self.lista_instrucoes[1:len(self.lista_instrucoes)]

    #se não houver mais instruções eu paro a movimentação
    if len(self.lista_instrucoes) != 0:
        self.InterpretaInit()



    if len(self.lista_instrucoes) == 0:
        self.moving_stop()
```

```
                    #self.trajetoriaX.append(140*np.cos((self.teta1_atual + self.teta2_atual+90)*(np.pi/180))
+174*np.cos((self.teta2_atual+90)*(np.pi/180))) #valor de x
                    #self.trajetoriaY.append(140*np.sin((self.teta1_atual + self.teta2_atual+90)*(np.pi/180))
+174*np.sin((self.teta2_atual+90))**(np.pi/180)+65) #valor de y
        #self.trajetoriaZ.append(165.43-self.altura3_atual) #valor de z

        #limpa a tela
        self.ax.clear()

        #define limites para o ambiente de trabalho
         self.ax.scatter3D(0,-400,0, facecolors = "black", alpha = 0.0) ###importante usar um volume quadrado
para não deformar a visão durante movimento
        self.ax.scatter3D(400,400,350, facecolors = "black", alpha = 0.0)
        self.ax.scatter3D(-400,400,350, facecolors = "black", alpha = 0.0)

        #self.ax.plot([0,400,0], [0,0,0],color='black')#linewidths=1, edgecolors='k', alpha=.9
        #self.ax.plot([0,-400,0], [0,0,0],color='black')

        #impressão de ponto de referencia para movimentação dos elos
        #self.ax.scatter3D(200+self.ref_ponto2[0], self.ref_ponto2[1], self.ref_ponto2[2], facecolors = "black" )

        if movimento1 == 1 :
            #movento o braço 1 em relação ao ponto fixo [0,65,280] self.ref_ponto1
            for i in range(len(peca2_verts)):
                for j in range(len(peca2_verts[i])):
                    peca2_verts[i][j] = peca2_verts[i][j] - self.ref_ponto1
                    peca2_verts[i][j] = self.rotacionando(peca2_verts[i][j], self.delta1)
                    peca2_verts[i][j] = peca2_verts[i][j] + self.ref_ponto1

            #movento o braço 1 em relação ao ponto fixo [0,65,280]
            for i in range(len(peca3_verts)):
                for j in range(len(peca3_verts[i])):
```

```
        peca3_verts[i][j] = peca3_verts[i][j] - self.ref_ponto1

        peca3_verts[i][j] = self.rotacionando(peca3_verts[i][j], self.delta1)

        peca3_verts[i][j] = peca3_verts[i][j] + self.ref_ponto1

#movento o braço 1 em relação ao ponto fixo [0,65,280]
for i in range(len(peca4_verts)):
    for j in range(len(peca4_verts[i])):
        peca4_verts[i][j] = peca4_verts[i][j] - self.ref_ponto1
        peca4_verts[i][j] = self.rotacionando(peca4_verts[i][j], self.delta1)
        peca4_verts[i][j] = peca4_verts[i][j] + self.ref_ponto1

#corrigindo posicionamento do ponto de referencia
self.ref_ponto2 = self.ref_ponto2 - self.ref_ponto1
self.ref_ponto2 = self.rotacionando(self.ref_ponto2 , self.delta1) #rotacionando ponto de referencia 2
self.ref_ponto2 = self.ref_ponto2 + self.ref_ponto1

#corrigindo posicionamento do ponto de trajetoria
temp1 = self.traj_list[-1] - self.ref_ponto1
temp1 = self.rotacionando(temp1, self.delta1) #rotacionando ponto de referencia 2
temp1 = temp1 + self.ref_ponto1
self.traj_list.append(temp1)
self.trajetoriaX.append(self.traj_list[-1][0])
self.trajetoriaY.append(self.traj_list[-1][1])
self.trajetoriaZ.append(self.traj_list[-1][2])

#######if self.delta1 > 0:
#######    for i in range(5):
#######        self.porta_serial.write(b"B")
#######        time.sleep(time_delay)

#######if self.delta1 < 0:
```

```
        #######   for i in range(5):
        #######       self.porta_serial.write(b"A")
        #######       time.sleep(time_delay)

#############################################################################################
    #movimento do braço 2 em relação ao ponto fixo [0, 230, 280]
    if movimento2 == 1 :
        #movento o braço 1 em relação ao ponto fixo [0, 230, 280]
        for i in range(len(peca3_verts)):
            for j in range(len(peca3_verts[i])):
                peca3_verts[i][j] = peca3_verts[i][j] - self.ref_ponto2
                peca3_verts[i][j] = self.rotacionando(peca3_verts[i][j], self.delta2)
                peca3_verts[i][j] = peca3_verts[i][j] + self.ref_ponto2

        #movento o braço 1 em relação ao ponto fixo [0,65,280]
        for i in range(len(peca4_verts)):
            for j in range(len(peca4_verts[i])):
                peca4_verts[i][j] = peca4_verts[i][j] - self.ref_ponto2
                peca4_verts[i][j] = self.rotacionando(peca4_verts[i][j], self.delta2)
                peca4_verts[i][j] = peca4_verts[i][j] + self.ref_ponto2

        temp1 = self.traj_list[-1] - self.ref_ponto2
        temp1 = self.rotacionando(temp1, self.delta2) #rotacionando ponto de referencia 2
        temp1 = temp1 + self.ref_ponto2
        self.traj_list.append(temp1)
        self.trajetoriaX.append(self.traj_list[-1][0])
        self.trajetoriaY.append(self.traj_list[-1][1])
        self.trajetoriaZ.append(self.traj_list[-1][2])

        #######if self.delta2 > 0:
        #######   for i in range(5):
        #######       self.porta_serial.write(b"D")
        #######       time.sleep(time_delay)
```

```
        #######if self.delta2 < 0:
        #######   for i in range(5):
        #######       self.porta_serial.write(b"C")
        #######       time.sleep(time_delay)

##############################################################################################
        #movimento do braço 3
        if movimento3 == 1:
            #movento o braço 1 em relação ao ponto fixo [0,65,280]
            for i in range(len(peca4_verts)):
                for j in range(len(peca4_verts[i])):
                    peca4_verts[i][j] = peca4_verts[i][j] - [0, 0, self.delta3]

            temp1 = self.traj_list[-1] - self.ref_ponto1
            temp1 = self.rotacionando(temp1, self.delta3) #rotacionando ponto de referencia 2
            temp1 = temp + self.ref_ponto1
            self.traj_list.append(temp1)
            self.trajetoriaX.append(self.traj_list[-1][0])
            self.trajetoriaY.append(self.traj_list[-1][1])
            self.trajetoriaZ.append(self.traj_list[-1][2])

        #imprimo as listas de trajetórias atualizada
        self.ax.plot(self.trajetoriaX, self.trajetoriaY, self.trajetoriaZ, alpha=.5, color='black')

        #imprimindo os vertices na tela
        self.ax.add_collection3d(Poly3DCollection(peca4_verts,
         facecolors='orange', linewidths=1, edgecolors='k', alpha=.2))

        self.ax.add_collection3d(Poly3DCollection(peca3_verts,
         facecolors='blue', linewidths=1, edgecolors='k', alpha=.9))

        self.ax.add_collection3d(Poly3DCollection(peca2_verts,
```

```
    facecolors='yellow', linewidths=1, edgecolors='k', alpha=.9))

    self.ax.add_collection3d(Poly3DCollection(peca1_verts,
    facecolors='green', linewidths=1, edgecolors='k', alpha=.9))


########################
def rotacionando(self, ponto_rot, new_angulo):
    ang = new_angulo

    #matriz de rotação, quando for mudar tem que generalizar para o robô.
    matriz_rot = np.array([[np.cos(ang*(np.pi/180)) , -np.sin(ang*(np.pi/180)) ,0   , 0],
                    [np.sin(ang*(np.pi/180)) , np.cos(ang*(np.pi/180))  ,0   , 0],
                    [0                  , 0                  ,1   , 0],
                    [0                  , 0                  ,0   , 1]])

    new_ponto = np.zeros((8,4))

    #multiplicando o ponto pela matriz de rotação
    new_ponto = self.multi_matriz(ponto_rot, matriz_rot)

    return new_ponto[0:3]

#multiplica um vetor por uma matriz 3x3
def multi_matriz(self, vetor, matriz):
    new_vetor = np.zeros(4)

    #multiplicação de matriz, fiz com só um loop para se precisar mudar em um único índice
    for i in [0,1,2,3]:
        new_vetor[i] = vetor[0]*matriz[i][0] +vetor[1]*matriz[i][1] + vetor[2]*matriz[i][2] +1*matriz[i][3]

    return new_vetor
```

```
#####################################3#################################################

def InterpretaInit(self):

    #Função move
    if self.lista_instrucoes[0][0] == 1 :
        #uso angulo direto
        self.teta1_novo = self.lista_instrucoes[0][1]
        self.teta2_novo = self.lista_instrucoes[0][2]
        self.altura3_novo = self.lista_instrucoes[0][3]
        print("teta1novo", self.teta1_novo)
        print("teta2novo", self.teta2_novo)

    #função pmove
    if self.lista_instrucoes[0][0] == 2:
        #funcao que transforma a posição em teta para o robo
        self.Cinematica_inversa()
        #a altura não muda nessa posição
        self.altura3_novo = self.lista_instrucoes[0][3]
        print("teta1novo", self.teta1_novo)
        print("teta2novo", self.teta2_novo)

    #função up
    if self.lista_instrucoes[0][0] == 3:
        self.teta1_novo = self.teta1_atual
        self.teta2_novo = self.teta2_atual
        self.altura3_novo =  0 #altura máxima


    #função down
    if self.lista_instrucoes[0][0] == 4:
        #ponto mais baixo do
        self.teta1_novo = self.teta1_atual
```

```
        self.teta2_novo = self.teta2_atual
        self.altura3_novo =  160 #altura máxima

    #função delay
    if self.lista_instrucoes[0][0] == 5:
        print('delay')

    #função home
    if self.lista_instrucoes[0][0] == 6:
        #volta para o estado inicial, chamar no final do programa.
        self.teta1_novo = 0
        self.teta2_novo = 0
        self.altura3_novo =  0

    if self.lista_instrucoes[0][0] == 7:
        print('lmove')
                                    Xold = 140*np.cos((self.teta1_atual + self.teta2_atual)*(np.pi/180))
+174*np.cos((self.teta2_atual)*(np.pi/180))+65 #valor de x
                                    Yold = 140*np.sin((self.teta1_atual + self.teta2_atual)*(np.pi/180))
+174*np.sin((self.teta2_atual))**(np.pi/180) #valor de y
        Xnew = self.lista_instrucoes[0][1]
        Ynew = self.lista_instrucoes[0][2]
        Zold = self.altura3_atual
        Znew = self.lista_instrucoes[0][3]
        lista = [] #lista vazia para substituir reorganizar o movelo antigo

        #Xold = abs(Xold)


        delta = (Xnew-Xold)/100
        gain = (Ynew-Yold)/(Xnew-Xold) #the gain in rect

        num_passes = round(( Xnew - Xold )/delta)
```

```
        #linha reta em z
        gainZ = ( Znew-Zold )/num_passes
        print("kkkkkkkkkkkkkkk")
        print("num_passes" ,num_passes)
        print("teta1", self.teta1_atual)
        print("teta2", self.teta2_atual)
        print("Xold", Xold)
        print("Xnew", Xnew)
        print("Yold", Yold)
        print("Ynew", Yold)
        print("gain", gain)

        for i in range(int(100)):
            Xold += delta
            Yold += delta*gain
            Zold += gainZ

            lista.append([2, (Xold), (Yold), Zold ])


        self.lista_instrucoes = lista + self.lista_instrucoes[1:]
        print("minha lista", self.lista_instrucoes

    #função zmove
    if self.lista_instrucoes[0][0] == 8:
        self.teta1_novo = self.teta1_atual
        self.teta2_novo = self.teta2_atual
        self.altura3_novo =  self.lista_instrucoes[0][1] #altura máxima


def Cinematica_inversa(self):
    x = self.lista_instrucoes[0][1]
```

```
        y = self.lista_instrucoes[0][2]

        #consertando para origem
        x = x-65

        #assumindo compilação correta e o ponto esta na área de trabalho
        #primeiro o segundo angulo
        t2n = np.arccos((pow(x,2)+ pow(y,2) - pow(174,2) - pow(140,2)) / (2*140*174))

        #k = 0

        if y<0:
            t2n = abs(t2n)
            #k = 1
            y = abs(y)

        #calculo um valor possível para teta1 e testo para saber se é válido
        num = y*(174+140*np.cos(t2n) )  - x*140*np.sin(t2n)
        den = x*(174+140*np.cos(t2n) )  + y*140*np.sin(t2n)

        #soma de offset para evitar divisão por zero. A resposta sai correta mas é bom evitar.
        t1n = np.arctan(num/(den+0.0001))

        self.teta1_novo = round(t1n*(180/np.pi), 0)
        self.teta2_novo = round(t2n*(180/np.pi), 0)

peca1_verts = VisualData.Peca1_dados.peca1()
peca2_verts = VisualData.Peca2_dados.peca2()
peca3_verts = VisualData.Peca3_dados.peca3()
peca4_verts = VisualData.Peca4_dados.peca4()
```