Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

Desenvolvimento de software de geração de trajetórias para construção de modelos metálicos tipo casca por meio do processo GTAW com adição de arame frio

Bruno Sales de Matos

PROJETO FINAL DE CURSO

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Brasília

2022

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

Desenvolvimento de software de geração de trajetórias para construção de modelos metálicos tipo casca por meio do processo GTAW com adição de arame frio

Bruno Sales de Matos

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Orientador: Prof. Dr. Gulherme Caribé de Carvalho

Brasília
2022

Sales de Matos, Bruno.
S163d Desenvolvimento de software de geração de trajetórias para construção de modelos metálicos tipo casca por meio do processo GTAW com adição de arame frio / Bruno Sales de Matos; orientador Gulherme Caribé de Carvalho. -- Brasília, 2022.
102 p.

Projeto Final de Curso (Engenharia de Controle e Automação) -- Universidade de Brasília, 2022.

1. Robótica Industrial. 2. Manufatura Aditiva. 3. ABB. 4. RAPID. I. Caribé de Carvalho, Gulherme, orient. II. Título

Universidade de Brasília
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

Desenvolvimento de software de geração de trajetórias para construção de modelos metálicos tipo casca por meio do processo GTAW com adição de arame frio

Bruno Sales de Matos

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Trabalho aprovado. Brasília, 23 de Setembro de 2022:

Prof. Dr. Guilherme Caribé de Carvalho, UnB/FT/ENM
Orientador

Prof. Dr. Maksym Ziberov , UnB/FT/ENM
Examinador interno

Prof. Dr. Walter de Britto Vidal Filho , UnB/FT/UNM
Examinador interno

Brasília
2022

Agradecimentos

Primeiramente gostaria de agradecer ao meu falecido pai Devaldo, pois tive o seu apoio até o fim de seus dias e a minha mãe Maria Lúcia por me incentivar nos momentos de desamino e cansaço a nunca desistir.

Sou grato a minha esposa Nicole, pois sem o seu apoio não teria conseguido concluir esse importante passo da minha vida. Sou grato à toda minha família pelo apoio que sempre me deram durante toda a minha vida.

Agradeço ao meu orientador Guilherme Caribé por aceitar me orientar na condução deste presente trabalho. Agradeço à Universidade de Brasília e a todos os professores do meu curso pela elevada qualidade do ensino oferecido.

Resumo

Este trabalho tem como objetivo o desenvolvimento de um software para programação de trajetórias em uma célula de soldagem robotizada de 8 graus de mobilidade para construção de modelos metálicos tipo casca por deposição em camadas por meio do processo GTAW com adição de arame frio (manufatura aditiva). A célula de soldagem é composta por um braço manipulador ABB-2600ID integrado a uma mesa posicionadora ABB IRBP A-250 e uma fonte de soldagem GTAW Fronius MW5000. Existem algumas limitações do robô em relação ao processo, pois, a adição do arame deve ser sempre contrária à direção do movimento e o giro da tocha de solda é limitado. Sendo assim, desenvolveram-se algumas estratégias de movimentação do robô para contornar essas limitações. Desenvolveu-se um software em Python capaz de recalcular os pontos de trajetórias obtidos através de um código em Karel, aplicando-se as estratégias de movimentação e então gerar as coordenadas, tanto da ferramenta acoplada ao punho do robô, uma tocha GTAW, quanto da mesa posicionadora, onde é fixada a base de deposição. Esses pontos são, então, organizados em um programa escrito na linguagem nativa do controlador do robô (RAPID), incluindo os comandos relacionados à fonte de soldagem, de modo a se realizar a deposição respeitando os requisitos do processo, no tocante à direção de alimentação do material de adição e às limitações de orientação possíveis de serem atingidas pelo robô, visando a construção de modelos metálicos tridimensionais tipo casca por meio de deposição contínua de metal.

Palavras-chave: Robótica Industrial. Manufatura Aditiva. ABB. RAPID.

Abstract

This project aimed at developing a software for programming trajectories in an 8 joint robotic GTAW cell for the construction of 8 joint robotic GTAW cell by means of metal additive manufacturing using the CW-GTAW (Cold Wire Gas Tungsten Arc Welding) process. The welding cell consists of an ABB-2600ID manipulator integrated with an ABB IRBPA-250 positioning table and a GTAW Fronius MW5000 power source. There are some limitations of the robot in relation to the process, since the addition of the wire must always be in the opposite direction of the arc movement (wire is feed in front of the arc) and the rotation of the welding torch is limited. Therefore, some robot movement strategies were developed to overcome these limitations. A software program in Python was developed capable of recalculating the trajectory points obtained through the Karel code, that results from the slicing of the part geometry model, by applying the proposed movement strategies, it generates the new trajectory points, including both the tool coordinates and the corresponding part orientation, provided by the 2-degree-of-freedom positioning table, where the deposition base is fixed.. These trajectory points are organized in a program written in native language (RAPID) of the robot controller, including the commands related to the welding source, in order to produce the deposition respecting the process requirements, in terms of to the direction of material addition and the orientation limitations possible to be reached by the robot, aiming at the construction of three-dimensional metallic shell models by continuous deposition of metal, always in the flat position.

Keywords: Industrial Robotic. Additive manufacturing. ABB. RAPID.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ABB	ASEA BROWN BOVERI	14
E/S	Entrada e Saída	39
FDM	Modelagem por Deposição de Material Fundido	13
GTAW	Gas Tungsten Arc Welding	13
RP	Rapid Prototyping	16
SMD	Shaped Metal Deposition	13
TCP	Tool Center Point	23
WAAM	Wire Arc Additive Manufacturing	14

Sumário

1 Introdução

1.1 Contextualização

Existe uma crescente demanda por novas abordagens sobre o processo de produção, especialmente, sobre a redução de custos. Na era da Indústria 4.0, a robótica industrial é um campo promissor com boas oportunidades de trabalho. Especialistas na área se destacam trabalhando diretamente na automação de tarefas, como exemplo na indústria automotiva. A robótica industrial é um excelente campo de pesquisa, pois cada vez mais os seres humanos estão sendo substituídos por máquinas na realização de tarefas estressantes e repetitivas. De fato, um estudo da Associação Brasileira de Automação mostra que a automação cresceu 8% no Brasil entre 2017 e 2018, destacando-se a Manufatura aditiva entre as áreas relacionadas à automação de processos produtivos. (BLOG IMPACTA, 2018)

> Segundo o site Stratasys (FDM tecnology), as primeiras técnicas de impressão 3D começaram na década de 1980 e foi chamada de tecnologia de prototipagem rápida (RP) devido à sua velocidade e custo-benefício em comparação com outros métodos de prototipagem. O SLA-1, foi o primeiro sistema de prototipagem rápida comercial e foi introduzido em 1987 pela 3D Systems e vendido em 1988. Este sistema usava estereolitografia, uma técnica de manufatura aditiva que usa luz para ligar cadeias de moléculas poliméricas. Também em 1987, Carl Deckard criou o método Selective Laser Sintering (SLS), um método para produzir peças a partir da aglomeração de partículas de material em pó. Em 1989, Scott Crump criou o método por deposição de material fundido (FDM), que possibilitou a produção de peças por extrusão por meio da fusão de polímeros, depositado em camadas sucessivas. A soldagem 3D ou Deposição de Metal Moldado (SMD, do inglês "Shape Metal Deposition") é um caso específico do FDM. Esse método apresenta o problema de transferência de calor para a peça que está sendo fabricada pois a maioria dos metais tem alto ponto de fusão, e por isso pode ser necessário um sistema de refrigeração. (apud. Vasconcellos e Figueiredo (2016))

Tradicionalmente, o SLS (Selective Laser Sintering) é aplicado a materiais poliméricos e vidro e poderia ser aplicado a metais, entretanto neste caso específico foi denominado Fusão Seletiva a Laser (SLM do inglês "Selective Laser Melting"ou DMLS do alemão "Direkt Metall Laser Schmelzen"). (ENGIPRINTERS, 2019)

O processo GTAW é um excelente método para a aplicação em manufatura aditiva, pois é um processo refinado, que oferece acabamento superficial superior, é compatível com uma enorme gama de materiais, não produz respingos e possui excelentes características mecânicas.(INFOSOLDA, 2022)

1.2 Motivação

O software de fatiamento, desenvolvido pelo Rodrigo C. Andrade (ANDRADE, 2013), foi projetado para geração de trajetórias para construção de modelos metálicos tipo casca por meio do processo GMAW, visando as características da célula robótica composta por um robô manipulador ABB IRB 2000 com uma tocha GMAW como órgão terminal e um controlador S3 que a programação offline é na linguagem ARLA.

Nesse presente trabalho utiliza-se uma célula robótica composta por um robô manipulador ABB IRB 2600 com uma tocha GTAW como órgão terminal e uma mesa posicionadora ABB IRBP A-250, ambos controlados por um controlador IRC5, onde a programação offline é feita na linguagem RAPID. Nesse contexto, o processo GTAW tem restrição de direcionamento da alimentação do arame, que deve ser feita na direção contraria ao movimento, pois, diferente do processo GMAW em que o arame também é o eletrodo, o eletrodo de tungstênio não é consumível e a adição de material (arame) é separada. Sendo assim, em uma deposição camada por camada em um contorno fechado a tocha deveria girar várias vezes para manter a adição de material sempre contraria a direção do movimento. Porém o giro da tocha é limitado em 180º e -180º, pois, com ângulos maiores poderia ocorrer a torção dos cabos de conexão da tocha e danifica-los. Então, com o auxílio da mesa posicionadora, é possível limitar aos ângulos aplicados a tocha enquanto a base de deposição, fixada no prato da mesa, gira sem limitação.

1.3 Objetivo

O objetivo desse trabalho é, por meio da célula robótica composta por um robô ABB IRB 2600 com uma tocha GTAW e uma mesa posicionadora IRB-A 250, aplicar o processo WAAM (Wire Arc Additive Manufacturing) Manufatura Aditiva a Arame e Arco, para produzir modelos tridimensionais do tipo casca por meio de um contorno fechado camada por camada. Em outras palavras, o objetivo é adaptar o método de fatiamento desenvolvido no trabalho do Rodrigo C. Andrade (ANDRADE, 2013) a essa nova célula robótica e ao processo GTAW.

Desenvolveu-se um software que obtem as coordenadas cartesianas de movimentação por meio de um código em linguagem Karel. Esse código é obtido a partir de uma das saídas do software de fatiamento. As coordenadas são recalculadas de acordo com a estratégia adotada, respeitando-se as limitações do robô, e, por fim, gera-se um código em linguagem Rapid.

1.4 Composição e estrutura do trabalho

Este documento foi organizado nos seguintes capítulos:

- Introdução (1): Consiste da motivação para o trabalho, os objetivos a serem atingidos por este, e a estrutura de apresentação deste documento.
- Fundamentação Teórica (2): Contém uma revisão bibliográfica dos assuntos necessários para compreensão dos métodos e processos.
- Metodologia (3): Introduz a metodologia utilizada para atingir os objetivos estabelecido, além de expor os materiais (equipamentos e softwares) utilizados.
- Resultados (4): Detalha a metodologia e demonstra os ensaios realizados e análises.
- Conclusão (5): Discute os resultados e possíveis objetivos para uma continuação do trabalho.

2 Fundamentação teórica

2.1 Prototipagem rápida

A Prototipagem Rápida (RP) pode ser definida como um grupo de técnicas usadas para fabricar rapidamente um modelo em escala de uma peça ou montagem, usando dados tridimensionais de projeto assistido por computador (CAD). Tem um uso óbvio como forma de visualização e uso para testes, como um formato de aerofólio é colocado em um túnel de vento por exemplo. Em alguns casos, a peça RP pode ser a parte final, mas normalmente o protótipo não é resistente ou preciso o suficiente. (EFUNDA , INC, 2008)

Existe uma infinidade de metodologias experimentais de RP em desenvolvimento. Nesse trabalho se concentrará no método WAAM - Manufatura Aditiva a Arame e Arco com deposição por GTAW.

2.2 Soldagem

A soldagem é tradicionalmente vista como um processo de união, mas hoje pode ser amplamente aplicado para deposição de material em superfícies com a finalidade de restaurar peças desgastadas, revestimento ou até a manufatura de uma peça inteira. (MARQUES; MODENESI; QUEIROZ, 2009)

Há dois tipos de processos de soldagem: por pressão e por fusão. O processo por pressão inclui processos de soldagem por forjamento, ultrassom, fricção, difusão, explosão, entre outros. Já o processo por fusão pode haver deposição de material ou não e se caracteriza pela formação de uma poça onde há material fundido. A fusão pode ser feita por aquecimento da escoria liquida, arco elétrico, feixe de elétrons, feixe de luz ou chama oxi-acetilênica.(MARQUES; MODENESI; QUEIROZ, 2009)

2.2.1 Soldagem GTAW (gas tungsten arc welding)

A soldagem a arco com eletrodo de tungstênio e proteção gasosa (do inglês, Gas Tungsten Arc Welding – GTAW) é um processo no qual peças metálicas são unidas por aquecimento e fusão por meio de um arco elétrico criado entre um eletrodo de tungstênio não consumível e a peça. Por meio de gases inertes (ou misturas de gases inertes) é feita a proteção da poça de fusão e do arco elétrico contra a contaminação atmosférica e pode ou não haver adição de metal. A Figura 2.1 ilustra o processo GTAW. (MARQUES; MODENESI; QUEIROZ, 2009)

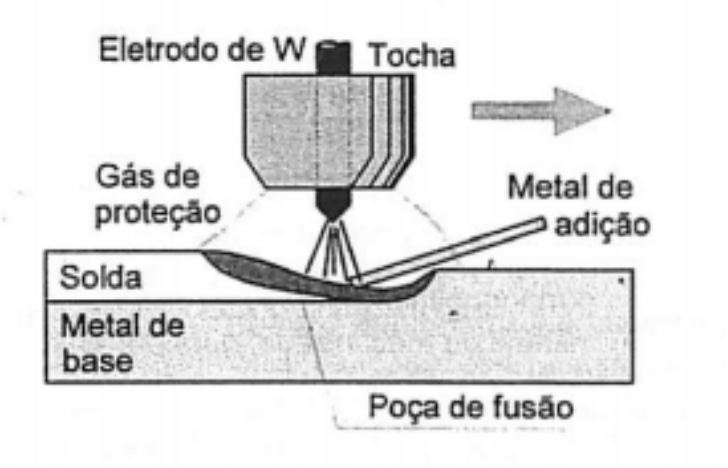


Figura 2.1 – Soldagem GTAW.

Fonte: (MARQUES; MODENESI; QUEIROZ, 2009).

Uma caraterística importante desse processo é o excelente controle da energia transferida para peça, aliado com uma eficiente proteção contra a contaminação. O arco elétrico é bastante estável, suave e produz soldas com bom acabamento.(MARQUES; MODENESI; QUEIROZ, 2009)

O projeto consiste em utilizar o processo de deposição por GTAW com deposição de arame frio. Na Figura 2.2 ilustra um esquema de montagem de uma célula de soldagem.

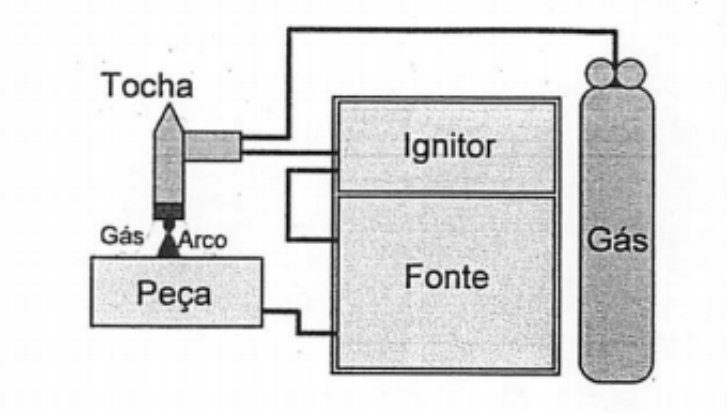


Figura 2.2 – Equipamento para soldagem TIG.

Fonte: (MARQUES; MODENESI; QUEIROZ, 2009).

2.3 Robótica industrial

Os manipuladores robóticos são compostos por membros conectados por juntas, que podem ser rotativas (rotação entre dois membros) ou prismáticas (permitem apenas translação relativa entre dois membros). (SILVA, s.d.)

Robôs manipuladores podem ter diferentes configurações entre os membros e as juntas. A quantidade de graus de liberdade de um manipulador é determinada pelo número de juntas. Para atingir uma determinada coordenada no espaço é necessário no mínimo 3 graus de liberdade. Entretanto é interessante determinar a orientação em que o órgão terminal atinge a coordenada, para isso são necessários 3 graus de liberdades adicionais para orientação. (SILVA, s.d.)

Na Figura 2.3 um exemplo de robô manipulador industrial com 6 graus de liberdade.

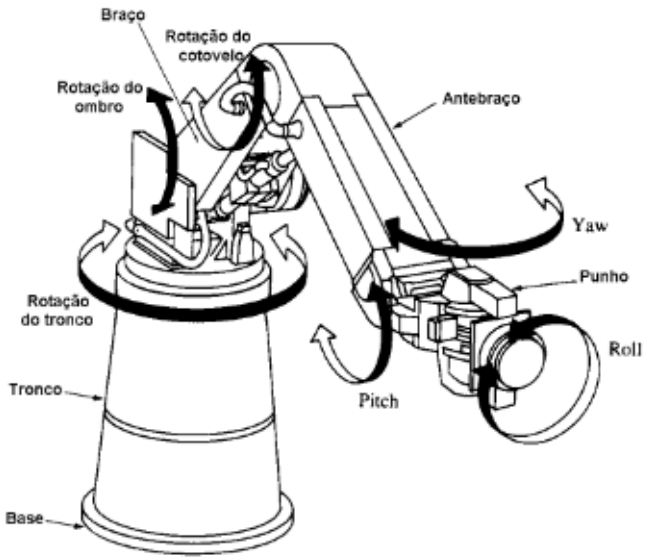


Figura 2.3 – Robô Industrial Típico.

Fonte: (SILVA, s.d.).

Pode-se classificar os robôs industriais em cinco configurações geométricas tradicionais: Articulado (RRR), Esférico (RRP), SCARA (RRP), Cilíndrico (RPP), Cartesiano (PPP), onde R significa junta rotativa e P significa junta prismática. Dentre as configurações o robô IRB 2600 se caracteriza como tipo articulado, pois todas as suas juntas são rotativas. (SILVA, s.d.)

O punho é definido pelo conjunto de juntas entre o último membro do robô e o órgão terminal, de modo a orientação desse último. Essa orientação pode ser obtida a partir de combinações dos ângulos de Euler ou então por quatérnions.

2.3.1 Modelagem Cinemática

2.3.1.1 Sistemas de Coordenadas

Um ponto no espaço pode ser definido em diferentes referenciais, as coordenadas desse ponto dependem da posição do sistema de coordenadas referenciado. Na Figura 2.4 mostra o ponto q visto pelo sistema de coordenadas R e N. (SANTOS, 2004)

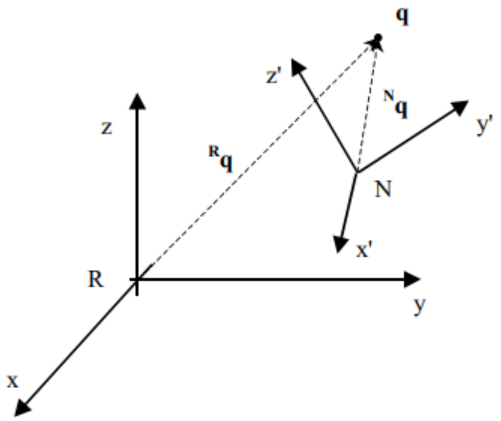


Figura 2.4 – Ponto q visto de referenciais diferentes.

Fonte: (SANTOS, 2004)

A relação entre esses referenciais pode ser descrita através de uma matriz transformação. Essa matriz tem componentes de rotação e de translação, descrita na Figura 2.5 e essa transformação pode ser demostrada na Figura 2.6.

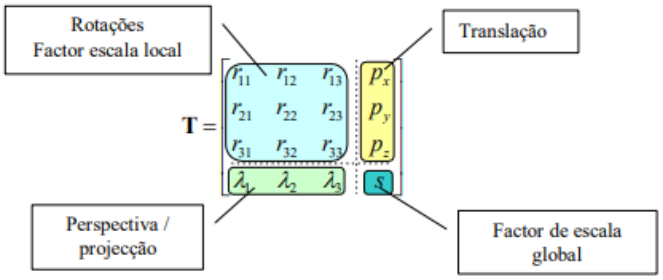


Figura 2.5 – Matriz transformação.

Fonte: (SANTOS, 2004)

.

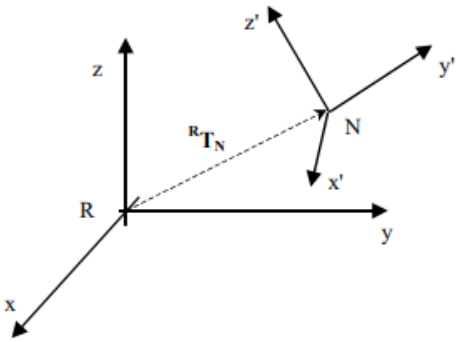


Figura 2.6 – Transformação entre os referenciais.

Fonte: (SANTOS, 2004)

Dessa maneira, de acordo com a Figura 2.4, relaciona-se as coordenadas de um mesmo ponto em referenciais diferentes por meio da Equação 2.1.

$$ {}^{R}q = {}^{R}T_{N}^{N}q \tag{2.1} $$

Em robótica industrial é necessário determinar os sistemas de coordenadas para caracterizar cada elemento do robô. Define-se os sistemas de coordenadas em cada junta, obtendo-se a matriz transformação entre os referenciais. Em um robô Manipulador Articulado (RRR), exemplo da Figura 2.7, a translação de cada junta é determinada pela distância entre a juntas, ou seja, é fixada pelo tamanho de cada membro, então a coordenada do órgão terminal é obtida pelas rotações das juntas. As transformações nessa configuração são de translações fixas e rotações variáveis. (SANTOS, 2004)

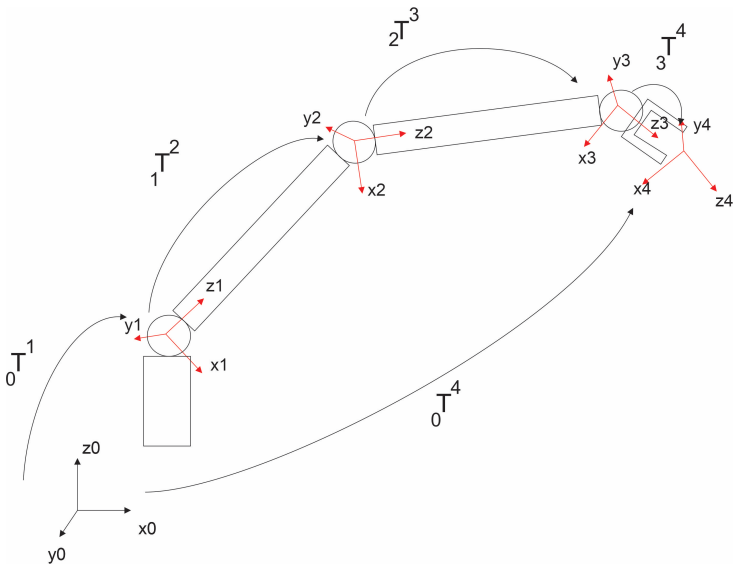


Figura 2.7 – Exemplo de definição do sistema de coordenadas de um manipulador articulado.

2.3.1.2 RPY

Os ângulos de Euler são ângulos rotação em torno de cada um dos três eixos coordenados do sistema cartesiano. Existem várias combinações de rotações usadas e a mais conhecida é a RPY (Roll-Pitch-Yaw) e consiste de rotações sequenciais nos eixos x, y e z, nesta ordem. Essa cominação de rotações é amplamente aplicada a orientações de posicionamento do órgão terminal em robótica. A Figura 2.8 ilustra a definição de roll, pitch e yaw. (SANTOS, 2004)

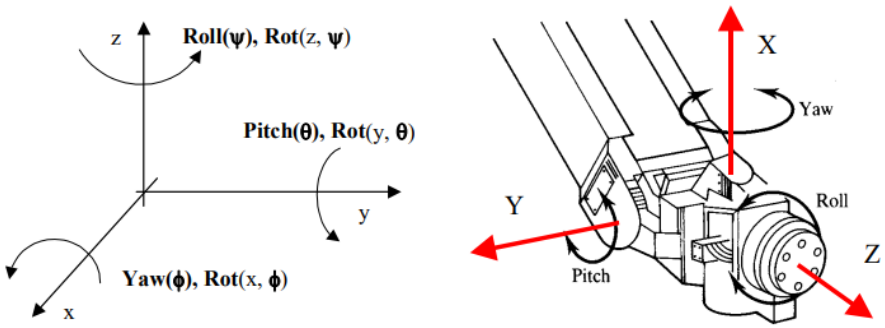


Figura 2.8 – Rotações RPY e correspondência usual no punho de um manipulador

Fonte: (SANTOS, 2004)

2.3.1.3 Quaternion

Um quaternion, semelhante aos números complexos, possui uma parte real e três imaginarias i, j, k. e podem ser escritos na forma $q = q_0 + q_1 i + q_2 j + q_3 k$ ou então na forma vetorial $q = (q_0,q_1,q_2,q_3)$, sendo q_0 a parte real e q_1, q_2, q_3 as parte imaginarias. (BIASI; GATTASS, s.d.)

As propriedades dos quatérnions são generalizações dos complexos. As partes imaginarias são definidas por $i^2 = j^2 = k^2 = -1$. Já a multiplicação de quatérnions é um pouco diferente e se assemelha ao produto vetorial, onde $ij = k$ e $ji = -k$, então por definição (BIASI; GATTASS, s.d.):

$$jk = i \Rightarrow jki = ii \Rightarrow jki = -1 \Rightarrow iijk = -i \Rightarrow -jk = -i \Rightarrow jk = i$$

Um quaternion unitário ($|q| = 1$) pode representar rotações de referenciais em torno de um vetor unitário no espaço euclidiano tridimensional. A Equação 2.2 descreve a rotação. (BIASI; GATTASS, s.d.)

$$q = \cos\frac{\theta}{2}, \vec{n}\sin\frac{\theta}{2} \tag{2.2}$$

Pode-se descrever uma determinada rotação como uma composição de rotações, simplesmente multiplicando os quatérnions correspondentes. Como exemplo, pode-se descrever as rotações do tipo Roll - Pitch - Yaw em quatérnions, bastando multiplicar os quatérnions correspondentes à rotação de cada eixo coordenado, de acordo com a Equação 2.3. (BIASI; GATTASS, s.d.)

$$q = (\cos\frac{\theta_x}{2}, (1,0,0)\sin\frac{\theta_x}{2})(\cos\frac{\theta_y}{2}, (0,1,0)\sin\frac{\theta_y}{2})(\cos\frac{\theta_z}{2}, (0,0,1)\sin\frac{\theta_z}{2}) \tag{2.3}$$

2.4 RAPID

RAPID é uma linguagem de programação de movimentação para robôs industriais da ABB. O RAPID substitui a linguagem de programação anterior ARLA . (ABB, 2010a)

Abaixo exibe-se um exemplo simples de programa em RAPID cujo objetivo é calcular a área de um retângulo e escrever o resultado na tela do FlexPedant:

```
1  MODULE MainModule
2      VAR num length;
3      VAR num width;
4      VAR num area;
5      PROC main()
6          length := 10;
7          width := 5;
8          area := length * width;
9          TPWrite "The area of the rectangle is " \Num:=area;
10     END PROC
11 ENDMODULE
```

2.4.1 Variáveis

2.4.1.1 Definição de variáveis

Há 3 formas de definir variáveis no código RAPID: VAR, CONST, PERS.

VAR (variable) – são variáveis manipuláveis que após o fim de uma chamada de função o valor da variável não é mais acessível;

CONST (constant) – são variáveis constantes, ou seja, são protegidas de manipulação;

PERS (persistent variable) – são variáveis manipuláveis que conservam o valor após o fim de uma chamada de função.

Obs.: Essas 3 formas não definem os tipos de dados armazenados na variável, é apenas uma abstração para definir o tratamento de uma variável no código.

2.4.1.2 Tipos de variáveis

Seguem-se descrições detalhadas de alguns tipos de dados RAPID, incluindo exemplos de como usá-los.

- **Robtarget** (robot target) – é usado para definir as posições do robô e de eixos externos, sendo declarada da seguinte maneira:

  CONST robtarget p15 := [[600, 500, 225.3], [1, 0, 0, 0], [1, 1,0, 0], [11, 12.3, 9E9, 9E9, 9E9, 9E9]];

Como pode ser visto acima, ela é dividida em 4 grupos de dados, posição, orientação, configuração do robô e eixo externos. A variável p15 acima é descrita da seguinte maneira:

- A posição (x, y e z) do TCP expresso em mm. A posição do robô: x = 600, y = 500 e z = 225,3 mm no sistema de coordenadas do robô;
- A orientação da ferramenta, expressa na forma de um quaternion (q1, q2, q3 e q4). A orientação da ferramenta corresponde à mesma direção que o sistema de coordenadas que o robô tem, quando este está em sua posição zero;
- A configuração do eixo do robô (cf1, cf4, cf6 e cfx). Isso é definido na forma de um quarto de volta (360/4) atual do eixo 1, eixo 4 e eixo 6. O primeiro quarto de volta positivo de 0 a 90 ° é definida como 0. O significado do componente cfx depende do tipo de robô. A configuração do eixo do robô: juntas 1 e 4 na posição 90-180 °, junta 6 na posição 0-90°.
- A posição dos eixos lógicos externos (até 6 eixos), “a” e “b”, expressos em graus ou mm (dependendo do tipo de eixo). Os eixos “c”, “d”, “e”, “f” são indefinidos onde mostram 9E9.

- **Speeddata** - é usado para especificar a velocidade com que o robô e o eixo externo se movem, e é declarado da seguinte maneira:

  CONST speeddata vp1 := [1000, 30, 200, 15];

  Os dados de velocidade vp1 são definidos com as seguintes velocidades:

  - 1000 mm/s para o TCP.
  - 30 graus/s para reorientação da ferramenta.
  - 200 mm/s para eixos externos lineares.
  - 15 graus/s para eixos externos rotativos.

- **Wobjdata** – Work Object Data – Define a posição e orientação do sistema de coordenadas do objeto de trabalho com relação ao sistema de coordenadas global.

  Se os objetos de trabalho forem definidos em uma instrução de posicionamento, a posição será baseada nas coordenadas do objeto de trabalho. As vantagens disso são as seguintes:

  - Se os dados de posição forem inseridos manualmente, como na programação off-line, os valores podem ser tirados de um desenho.
  - Os programas podem ser reutilizados rapidamente após alterações na instalação do robô. Se, por exemplo, o acessório é movido, apenas o sistema de coordenadas do usuário deve ser redefinido. Se for utilizada uma ferramenta estacionária

ou eixos externos coordenados, o objeto de trabalho deve ser definido, pois o caminho e a velocidade estariam relacionados ao objeto de trabalho em vez do TCP. Os dados do objeto de trabalho também podem ser usados para jogging (movimentação manual do robô):

- O robô pode ser movimentado nas direções do objeto de trabalho.
- A posição atual exibida é baseada no sistema de coordenadas do objeto de trabalho.

*PERS wobjdata variablename :=[[**robhold**], [**ufprog**], [**ufmec**], [**uframe** [x,y,z], [q1, q2, q3 ,q4] [**oframe** [x,y,z], [q1, q2, q3 ,q4]]];*

robhold - robot hold

Tipo de dados: bool

Define se o robô na tarefa do programa atual está ou não seguindo o objeto de trabalho (TRUE ou FALSE):

ufprog

Tipo de dados: bool

Define se um sistema de coordenadas de usuário é fixo ou não (TRUE ou FALSE):

• TRUE: Sistema fixo de coordenadas do usuário.

• FALSE: Sistema de coordenadas do usuário móvel, ou seja, eixos externos coordenados são usados. Também para ser usado em um sistema MultiMove em modo coordenado, semicoordenado ou sincronizado.

ufmec

Tipo de dados: string

A unidade mecânica com a qual os movimentos do robô são coordenados. Especificado apenas no caso de sistemas de coordenadas móveis do usuário (ufprog é FALSE). Especifica o nome da unidade mecânica definida nos parâmetros do sistema.

uframe - Coordenadas do usuário

Tipo de dados: posição

O sistema de coordenadas do usuário, ou seja, a posição da superfície de trabalho atual ou fixação:

• A posição da origem do sistema de coordenadas (x, y e z) em mm.

• A rotação do sistema de coordenadas, expressa como um quatérnion (q_1, q_2, q_3, q_4).

Se o robô estiver com a ferramenta, o sistema de coordenadas do usuário é definido no sistema de coordenadas global (no sistema de coordenadas do pulso se uma ferramenta estacionária for usada).

Para sistema de coordenadas de usuário móvel (ufprog é FALSE), o frame de usuário é definido continuamente pelo sistema.

oframe - Coordenadas do objeto

Tipo de dados: posição

O sistema de coordenadas do objeto, ou seja, a posição do objeto de trabalho atual (veja a Figura 2.8):

• A posição da origem do sistema de coordenadas (x, y e z) em mm.

• A rotação do sistema de coordenadas, expressa como um quatérnion (q1, q2, q3, q4).

O sistema de coordenadas do objeto é definido no sistema de coordenadas do usuário.

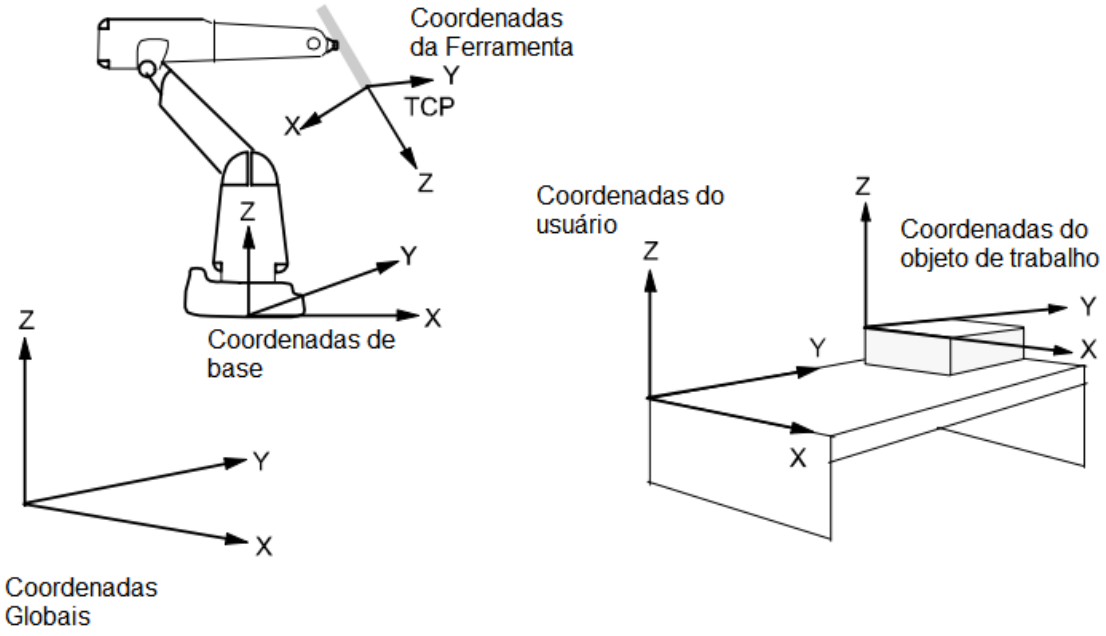


Figura 2.9 – Sistemas de coordenadas do robô.

Fonte: (ABB, 2010a)

Exemplo:

PERS wobjdata wobj2 :=[FALSE, TRUE, , [[300, 600, 200], [1, 0, 0 ,0]], [[0, 200, 30], [1, 0, 0 ,0]]];

O objeto de trabalho no exemplo acima é descrito usando os seguintes valores:

- O robô não está seguindo o objeto de trabalho.
- O sistema de coordenadas do usuário é fixo.
- O sistema de coordenadas do usuário não é girado e as coordenadas são x= 300, y = 600 e z = 200 mm no sistema de coordenadas global.
- O sistema de coordenadas do objeto não é girado e as coordenadas de sua origem são x= 0, y= 200 e z= 30 mm no sistema de coordenadas do usuário.

– *wobj2.oframe.trans.z := 38,3;* – A posição do objeto de trabalho wobj2 é ajustada para 38,3 mm na direção z, com relação ao sistema de coordenadas do usuário.

Um exemplo de programa de movimentação:

```
MODULE MainModule
    TASK PERS tooldata TTW5500_1 := [TRUE,[[32.1099,
       0.389434,387.111], [1,0,0,0]],
       [1,[0,0,100],[1,0,0,0],0,0,0]];

    CONST robtarget p0:=[[1455.38,0.95,1463.94],
       [0.7073,-0.00719,0.708,0.0084],  [0,0,-1,1],
       [9E+09,0,0,9E+09,9E+09,9E+09]];
    CONST robtarget p1:=[[-9.98,-94.47,0.10], [0,0,1,0],
       [-1,0,-1,0] , [9E+09,0.00,0.00,9E+09, 9E+09,9E+09]];
    CONST robtarget p2:=[[-4.75,-94.85,0.23], [0,0,1,0],
       [-1,0,-1,0], [9E+09,0.00,-9.06,9E+09, 9E+09,9E+09]];
    CONST robtarget p3:=[[-4.82,-94.84,0.35],[0,0,1,0],
       [-1,0,-1,0],[9E+09,0.00,-15.02,9E+09, 9E+09,9E+09]];
    CONST robtarget p4:=[[-4.68,-94.85,0.48],[0,0,1,0],
       [-1,0,-1,0],[9E+09,0.00,-21.14,9E+09, 9E+09,9E+09]];

    CONST speeddata vd1 :=[50.00,500,5000,0.00];
    COSNT speeddata vd2 :=[25.01,500,5000,45.25];
    CONST speeddata vd3 :=[0.59,500,5000,30.17];

    PROC main()
        ActUnit STN1;
        MoveJ p1, vd1, z5, TTW5500_1;
        MoveL p2, vd2, z5, TTW5500_1;
        MoveC p3, p4, vd3, z5, TTW5500_1;
    END PROC
END MODULE
```

2.4.2 Instruções de movimentação

Existem três instruções de movimentação, MoveL, MoveC e MoveJ. (ABB, 2010a)

MoveL - interpola uma trajetória linear entre o ponto de atual do TCP do robô e o ponto de destino.

MoveC - interpola uma trajetória circular utilizando o ponto atual do TCP do robô e dois pontos indicados na instrução.

MoveJ - move o TCP do ponto atual ao ponto de destino com velocidades angulares das juntas constantes, ou seja, não será linear necessariamente.

A instrução é escrita no programa da seguinte forma:

MoveL p1, v1000, z30, tool;

- p1 - é a variável de posicionamento do tipo robtarget declarada no campo de dados;
- v1000 - é a variável de velocidade linear do tipo speeddata declarada no campo de dados;
- z30 - variável do tipo zonedata, é usado para especificar como uma posição deve ser finalizada, ou seja quão perto da posição programada os eixos devem estar antes de passar para a próxima posição. O número 30 indica o raio (em mm) da circunferência (ou casca esférica) em volta do ponto;
- tool - variável do tipo tooldata, que indica a posição do TCP com relação a punho do robô.

2.4.3 Mais algumas instruções

A seguir mais algumas instruções que serão pertinentes ao presente trabalho:

Set – Atribui "1" a um sinal de saída digital.

Reset – Atribui "0" a um sinal de saída digital.

Set do15;

Atribuiu-se 1 ao sinal do15.

Reset weldon;

Atribuiu-se 0 ao sinal weldon;

SetAO – é utilizado para atribuir um valor real a um sinal de saída analógico.

SetAO ao2, 5.5;

Atribui-se 5.5 ao sinal ao2.

WaitDO - pausa a execução até que o sinal digital de saída seja atribuído um valor desejado.

WaitDI - pausa a execução até que o sinal digital de entrada seja atribuído um valor desejado.

WaitDO do4, 1;

A execução do programa continua apenas depois o valor do sinal de saída do4 ser 1.

WaitDI grip_status, 0;

A execução do programa continua apenas depois o valor do sinal de entrada grip_status ser 0.

3 Metodologia

De início, analisa-se as saídas do software de fatiamento. Esse software tem duas opções de linguagens de saída: Karel, utilizado para simulação em Workspace LT®, e ARLA (ABB Robot Language) utilizado no SportS3® (software de programação Off-line textual do IRB2000 voltada para controladores S3).

O código na linguagem Karel é composto por dois arquivos: um contendo as instruções de movimentação e outro contendo as posições de trajetória. Essa linguagem se mostrou mais interessante pois a sua estrutura simplifica o algoritmo de leitura dos dados no software desenvolvido nesse trabalho, pois utiliza-se apenas as posições.

No processo GTAW existem algumas condições a serem respeitadas. A alimentação de arame ser majoritariamente contraria ao movimento da tocha é a condição mais notável. Sendo assim, em uma deposição em várias camadas de contorno fechado a tocha deveria realizar movimentos circulares maiores que 360º, acarretando em torções nos cabos de conexão da tocha. Para satisfazer essa condição e evitar danos, por meio da mesa posicionadora IRBP A-250 é possível limitar os ângulos aplicados à tocha enquanto a base de deposição, fixada no prato da mesa, gira sem limitação.

Observando essa solução, a estratégia mais óbvia é a conversão das posições para coordenadas cilíndricas, onde o movimento da tocha está condicionado a um raio e ao incremento das camadas enquanto a base gira na posição angular correspondente. Além disso, é também calculado a orientação da tocha para que a alimentação de arame seja na direção contraria ao o movimento relativo da tocha com relação a base. Define-se o sistema de coordenadas do objeto no centro do prato da mesa posicionadora, assim as posições angulares da mesa são sincronizadas com as posições da tocha.

Contudo, há a possibilidade de ocorrer escorrimento da poça na confecção de peças de superfície inclinada e causar imperfeições na deposição, ou até impossibilitar o processo. Então para minimizar esse efeito, reorienta-se a superfície da base de tal modo que a deposição ocorra na direção da gravidade. A mesa posicionadora possui apenas 2 eixos de rotação, e um desses eixos já é utilizado para resolver o problema de torção da tocha, restando apenas o outro eixo para a reorientação da superfície de deposição.

Na estratégia de coordenadas cilíndricas, não é possível garantir que o eixo de reorientação da superfície esteja paralelo ao eixo de reorientação da mesa. Então para satisfazer essa nova limitação, desenvolve-se uma segunda estratégia: opta-se por restringir o movimento da tocha a uma trajetória paralela ao eixo de reorientação da mesa, com exceção nas transições do giro do prato. Dessa maneira não há necessidade de orientar a tocha (quaternion de orientação), pois ela, agora, é feita pela mesa, então a tocha permanece em uma orientação

fixa.

Portanto, desenvolve-se um software em Python, nomeado KareltoRapid, para realizar a leitura das posições contidas no código Karel, aplicar uma das duas abordagens expostas para recalcular as trajetórias (opcional à escolha do usuário), e então gerar o código em Rapid.

Para validar as novas trajetórias geradas escritas no código em Rapid, modifica-se o software KareltoRapid para não incrementar as camadas (sem incremento no eixo de crescimento), para que as trajetórias permaneçam em um mesmo plano, obtendo-se as curvas de nível do modelo desejado. Fixa-se um lápis no TCP e então realiza-se ensaios de movimentação do robô desenhado essas curvas em uma cartolina fixada na mesa posicionadora. Portanto é possível determinar se o robô está executando a trajetória esperada e mensurar eventuais erros.

3.1 Equipamentos e softwares utilizados

Utilizaram-se um manipulador ABB IRB 2600ID-15/1.85 com uma tocha GTAW TW5500 como órgão terminal, uma mesa posicionadora ABB IRBP A-250, uma fonte de soldagem Fronius MagicWave 5000 Job G/F MV, um controlador ABB IRC5 e o Software de Fatiamento.

3.1.1 Software de fatiamento

O software de fatiamento, desenvolvido pelo Rodrigo C. Andrade (ANDRADE, 2013), é um programa escrito na linguagem Scheme e utiliza modelos computacionais tridimensionais criados no Scheme ACIS Interface Driver Extension (Scheme AIDE) para obtenção dos pontos, ilustrado na Figura 3.10. Esse software gera códigos de movimentação de robôs na linguagem Karel, utilizado para simulação em Workspace LT®, e ARLA (ABB Robot Language) utilizada no SportS3® (software de programação Off-line textual do IRB2000 voltada a controladores S3).

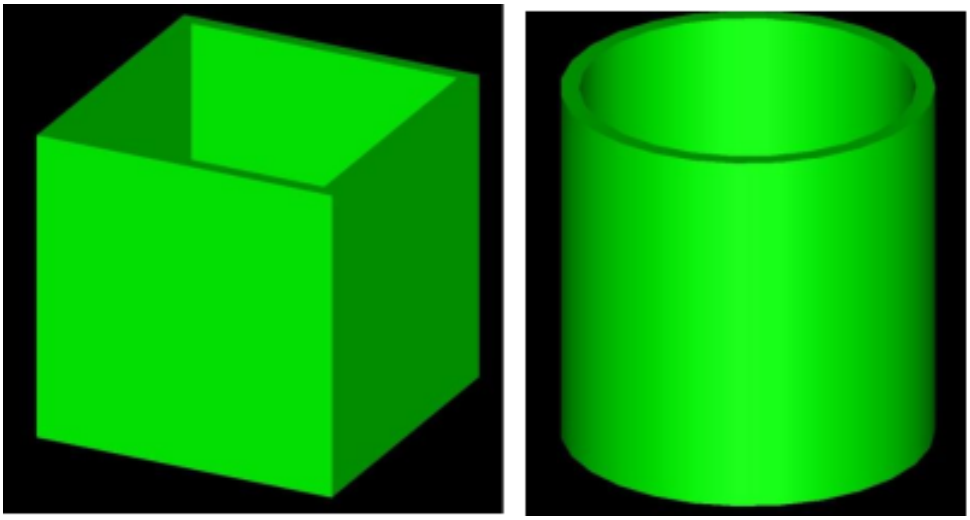


Figura 3.10 – Exemplo de modelos 3D do Scheme AIDE

Fonte: (ANDRADE, 2013)

Os modelos são “fatiados” em vários planos paralelos à base do modelo. A estratégia de fatiamento utilizada consiste em efetuar fatiamentos em planos paralelos, mas mantendo uma movimentação contínua do braço robótico. Ao invés de percorrer o perímetro da intersecção do modelo com cada plano, uma trajetória contínua é interpolada baseada na interseção entre o modelo e planos de fatiamento consecutivos, dessa maneira obtém-se uma trajetória helicoidal (ANDRADE, 2013). A Figura 3.11 ilustra um exemplo de um ciclo de fatiamento.

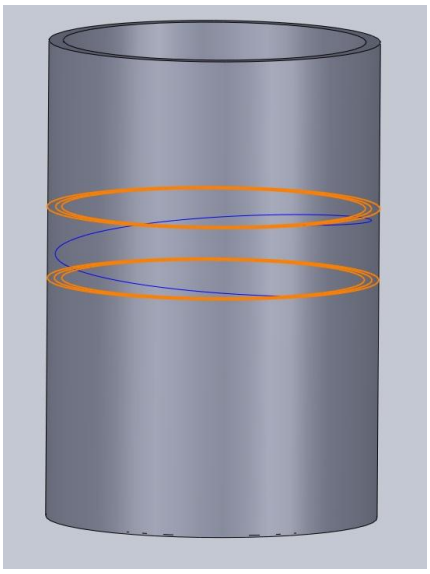


Figura 3.11 – Ilustração de exemplo de um ciclo de fatiamento

Fonte: (ANDRADE, 2013)

3.1.2 Célula robótica IRB 2600ID e IRBP A-250

O IRB 2660ID é um robô manipulador de 6 graus de liberdade, 3 graus para movimentação e 3 graus para orientação da ferramenta acoplada. A tocha GTAW é fixada no punho do robô. A Figura 3.12 apresenta a geometria e as dimensões do robô IRB 2600ID 15/1.85.

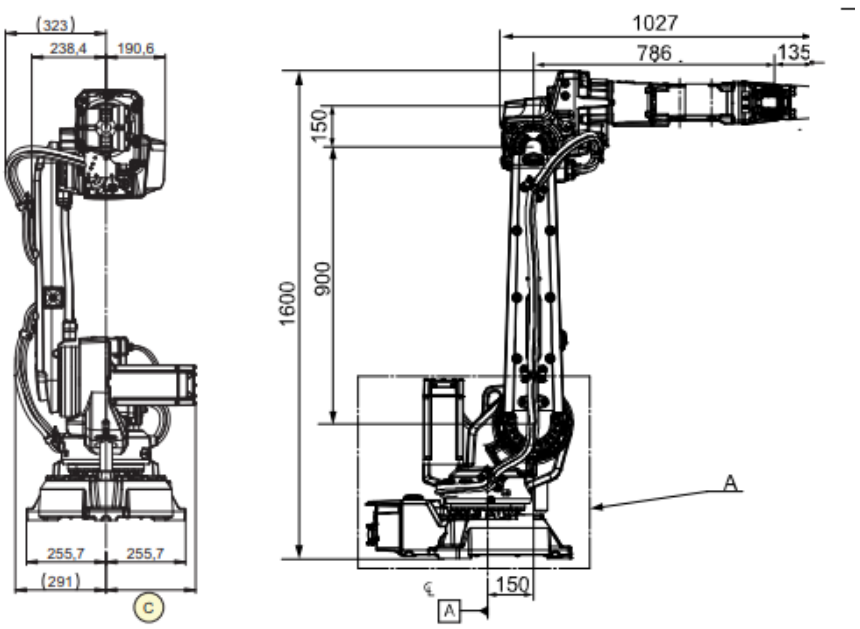


Figura 3.12 – Dimensões do IRB 2600ID 15/1.85

Fonte: (ANDRADE, 2013)

A Figura 3.13 ilustra as dimensões da área de trabalho do robô.

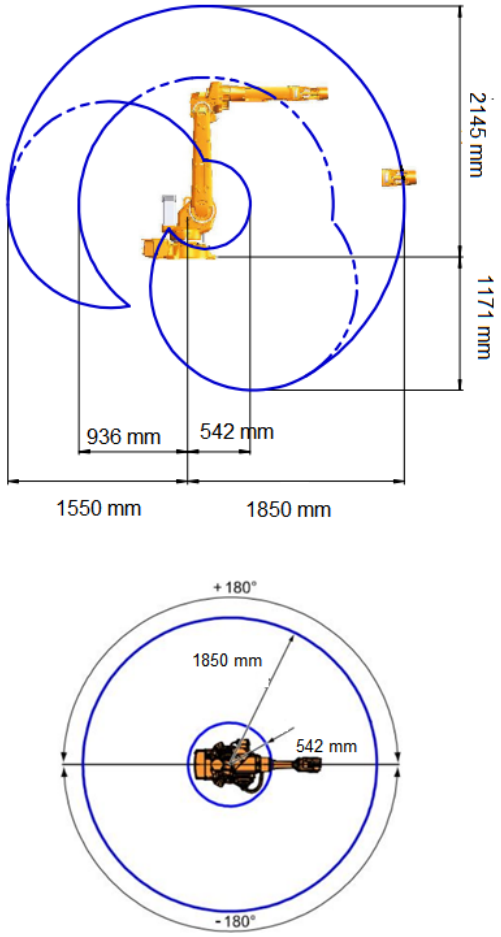


Figura 3.13 – Dimensões da Área de trabalho do Robô IRB 2600

Fonte: (ANDRADE, 2013)

Figura 3.14 – Foto do robô IRB 2600ID

Fonte: (ANDRADE, 2013)

O IRBP A-250 é uma mesa posicionadora com 2 graus de liberdade. É ideal para trabalho em peças que precisam ser giradas em torno de dois eixos para alcançar a posição ideal do processo. Compartilha o mesmo controlador e acionadores de robôs manipuladores e seus movimentos podem ser combinados no mesmo programa. A Figura 3.15 ilustra a geometria e as dimensões da mesa posicionadora IRBP A-250. A Figura 3.16 ilustra a representação tridimensional da mesa posicionadora e os limites de movimentação das juntas.

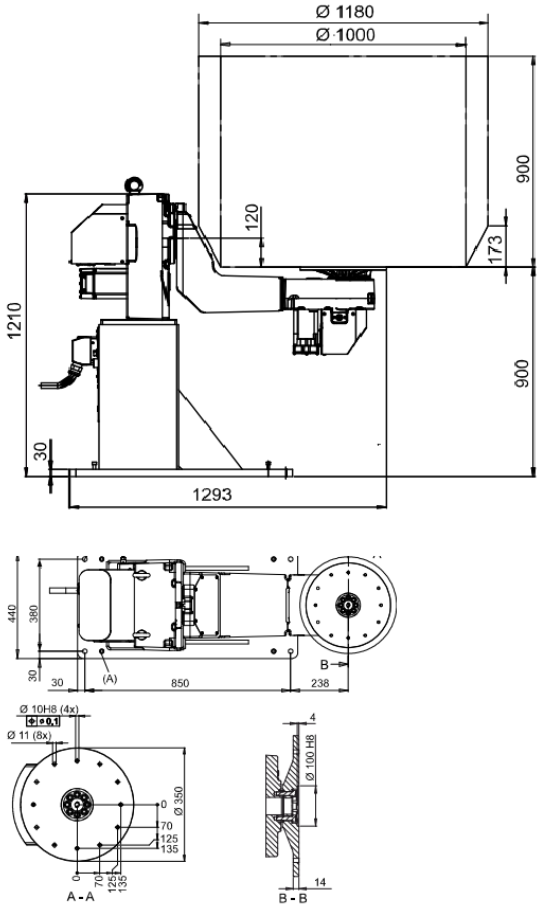


Figura 3.15 – Dimensões do IRBP A-250

Fonte: (ANDRADE, 2013)

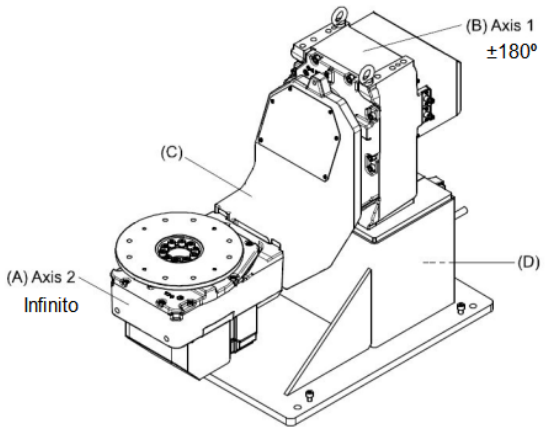


Figura 3.16 – Modelo tridimensional da mesa Posicionadora e limites dos eixos

Fonte: (ANDRADE, 2013)

Figura 3.17 – Foto da mesa posicionadora IRBP A-250

3.1.2.1 Modelagem cinemática da célula robótica

Para garantir a sincronia da trajetória da ferramenta acoplada e os movimentos de orientação da mesa posicionadora é necessário que a transformação que leva o sistema da ferramenta (tool) ao ponto de deposição no objeto (p) seja equacionada para uma matriz identidade, por meio da variação das coordenadas de juntas do robô e das coordenadas de juntas da mesa, conforme mostra a Equação 3.1, que deve ser calculada para todos os pontos da trajetória pretendida. A definição dos pontos a serem programados no conjunto robo-mesa foram obtidas por meio desse equacionamento (subseção 2.3.1). Simplificando, as transformações do robô manipulador até o ponto devem ter o mesmo resultado das transformações da mesa posicionadora até esse mesmo ponto. A Figura 3.18 ilustra as transformações.

$$^{Global}T_0 *^0 T_{tool} *^{tool} T_p = ^{Global} T_{m0} *^{m0} T_{m1} *^{m1} T_{base} *^{base} T_p \tag{3.1}$$

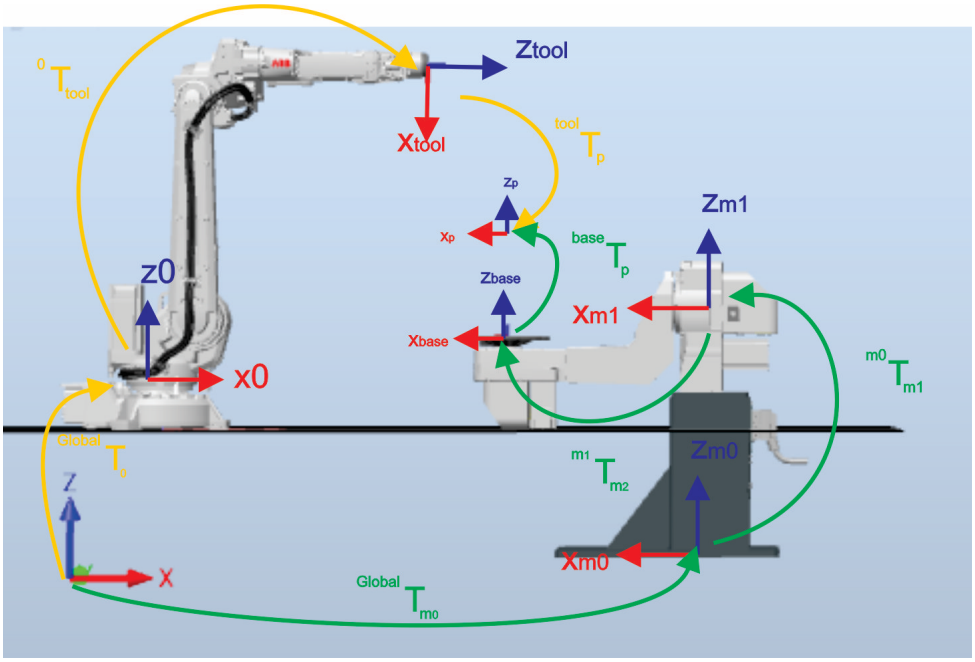


Figura 3.18 – Representação da modelagem cinemática da célula robótica

As transformações do robô manipulador, da base ao punho, já são pré-definidas de fábrica, bastando determinar apenas a transformação do punho à ferramenta acoplada para obter as coordenadas do ponto central da ferramenta (do inglês - Tool Center Point – TCP). A transformação do TCP é obtida através de configuração no FlexPendant do controlador. Define-se algumas posições referentes ao punho do manipulador movendo o referencial desejado da ferramenta manualmente para um mesmo ponto no espaço com orientações diferentes, obtendo assim as coordenadas e orientações do punho (para esse ponto), então o controlador fará automaticamente as interpolações e retornará a transformação correspondente do novo TCP.

A transformação do referencial global ao centro do prato (base de deposição) é definida manualmente por uma variável do tipo **Work Object Data** no programa Rapid (subseção 2.4.1), ou seja, define-se o objeto de trabalho com as coordenadas da base na mesa.

3.1.3 Fonte de soldagem Fronius MagicWave 5000

As fontes de soldagem GTAW MagicWave (MW), são fontes inversoras totalmente digitalizadas e comandadas com microprocessador, assim por meio da conexão LocalNet, todas as suas funcionalidades podem ser comandadas por um controlador externo. A fonte MagicWave se caracteriza por uma estrutura robusta e operação simples.(FONTE..., 2010)

Figura 3.19 – Representação da modelagem cinemática da célula robótica

3.1.4 Controlador IRC5

Figura 3.20 – Controlador IRC5

O IRC5 controla a movimentação do robô IRB 2600ID, da mesa posicionadora IRBP A-250, e, por conexão em uma rede LocalNet, também controla as ações da fonte de soldagem MagicWave 5000, dessa forma é possível, através de um código em RAPID, sincronizar a movimentação do robô com o processo GTAW. A operação é feita pelo dispositivo FlexPendant ou remotamente pelo software RobotStudio.

3.1.4.1 FlexPendant

O FlexPendant é uma interface do controlador com o usuário, (ocasionalmente denominado TPU ou unidade de programação) é uma unidade de operador portátil usada para efetuar muitas das tarefas envolvidas na operação de um sistema de robô: executar programas, manobrar o manipulador, modificar os programas do robô, etc. Na Figura 3.21 apresenta uma foto real do FlexPendant. Ao inicializar o controlador do Robô é mostrado a tela inicial como na Figura 3.22. (ABB, 2010b)

Figura 3.21 – Foto a Unidade FlexPendant

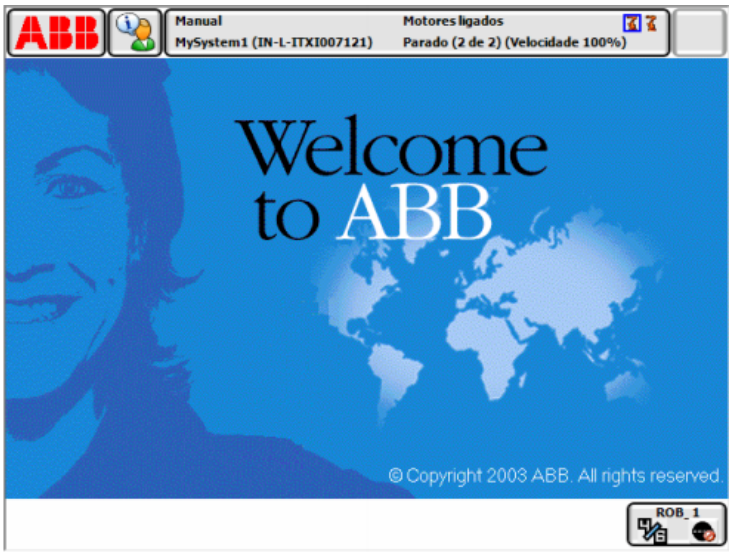


Figura 3.22 – Tela Inicial do FlexPendant

Fonte: (ABB, 2010b)

Os seguintes itens podem ser selecionados no menu ABB:

- **HotEdit**
- **Entradas e Saídas**

- **Manobra**
- **Janela de produção**
- **Editor do programa**
- **Dados do programa**
- **Backup e restauração**
- **Calibração**
- **Painel de controle**
- **Registro de eventos**
- **FlexPendant Explorer**
- **Informações do sistema**
- **etc.**

HotEdit é usado para ajustar as posições programadas, ou seja, posições do tipo robtarget carregadas de um programa podem ser facilmente alteradas (Figura 3.23). (ABB, 2010b)

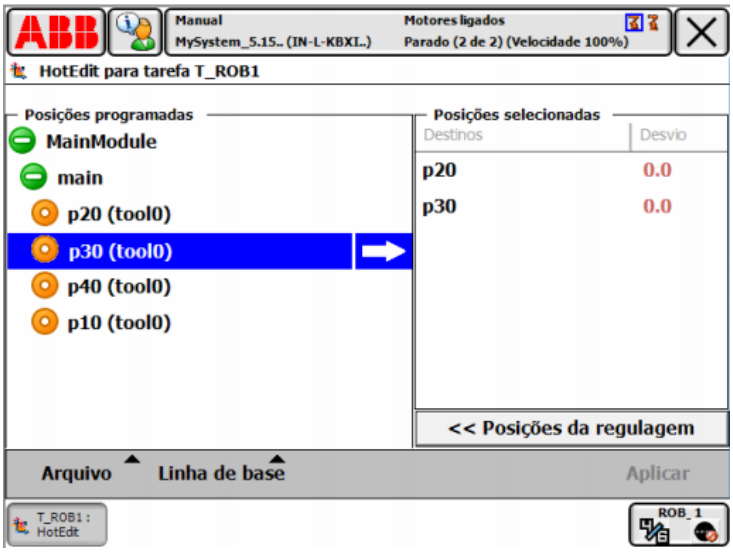


Figura 3.23 – Ilustração da visualização do HotEdit

Fonte: (ABB, 2010b)

Entradas e Saídas, E/S, são sinais usados no sistema do robô. Um sinal E/S é a representação lógica do software:

- Entradas ou saídas localizadas numa unidade E/S fieldbus (barramento de campo) que está conectada a um barramento dentro do sistema do robô (sinal real E/S).
- Um sinal de E/S sem uma representação em qualquer fieldbus da unidade de E/S (sinal virtual E/S).

Entradas ou saídas localizadas numa unidade E/S fieldbus (barramento de campo) que está conectada a um barramento dentro do sistema do robô (sinal real E/S).(ABB, 2010b)

Um sinal de E/S sem uma representação em qualquer fieldbus da unidade de E/S (sinal virtual E/S).

Especificando um sinal E/S, é criada uma representação lógica do sinal real ou virtual. (ABB, 2010b)

É possível visualizar e modificar os valores das entradas e saídas de controle da fonte de soldagem MW 5000 mapeadas no controlador (estas serão esclarecidas mais adiante) (Figura 3.24).

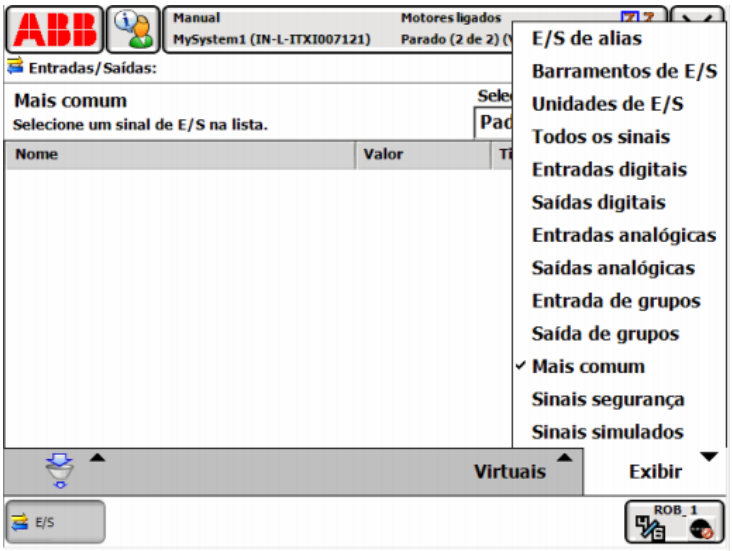


Figura 3.24 – Ilustração Entradas e Saídas

Fonte: (ABB, 2010b)

No menu **Manobrar** é feita a movimentação manual do robô, onde é possível a movimentação de cada junta separadamente com medidas em graus ou movimentação linear do TCP medida em mm no sistema de coordenadas cartesiano (XYZ) e o quarternion (q0, q2, q3, q4) de orientação do TCP. A Figura 3.25 mostra a tela do menu Manobrar. (ABB, 2010b)

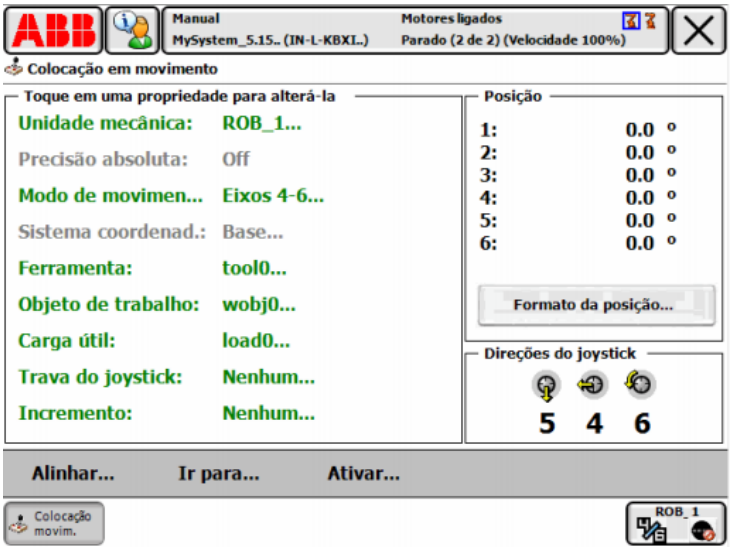


Figura 3.25 – Ilustração do menu Manobrar

Fonte: (ABB, 2010b)

No menu **Dados do Programa** estão todos os dados do programa Rapid carregado, dados do tipo robtarget (posições), speeddata (velocidades), tooldata (definição do TCP), etc. Podem ser visualizados, adicionados ou editados nesse menu (Figura 3.26). (ABB, 2010b)

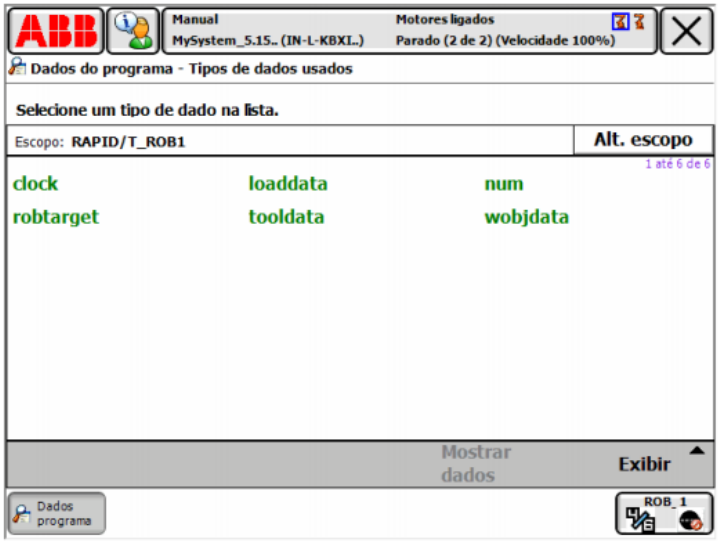


Figura 3.26 – Ilustração de Dados do Programa

Fonte: (ABB, 2010b)

O **Editor do programa** é o local onde se criam ou modificam as instruções do programas em RAPID (Figura 3.27). (ABB, 2010b)

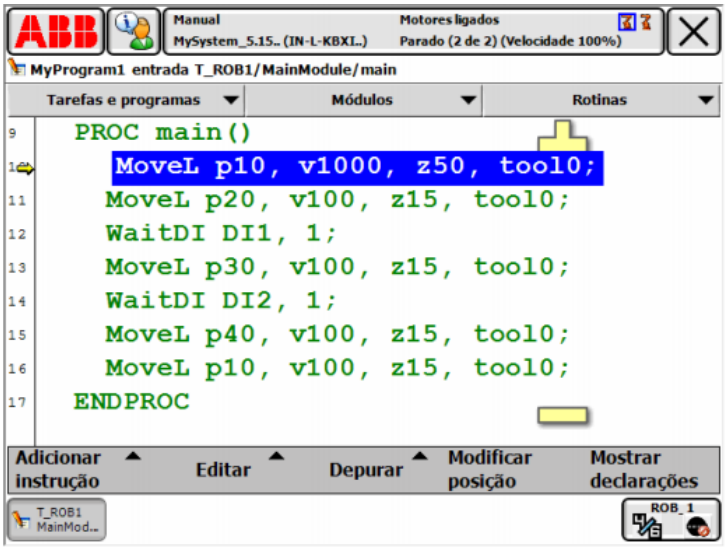


Figura 3.27 – Ilustração do Editor de Programa

Fonte: (ABB, 2010b)

3.1.5 RobotStudio

O RobotStudio é um aplicativo de PC para modelagem, programação offline e simulação de células robóticas. Por meio de comunicação via Ethernet com o controlador é possível o carregamento de programas, configuração do robô e a supervisão da operação remotamente. A Figura 3.28 mostra a tela do RobotStudio.

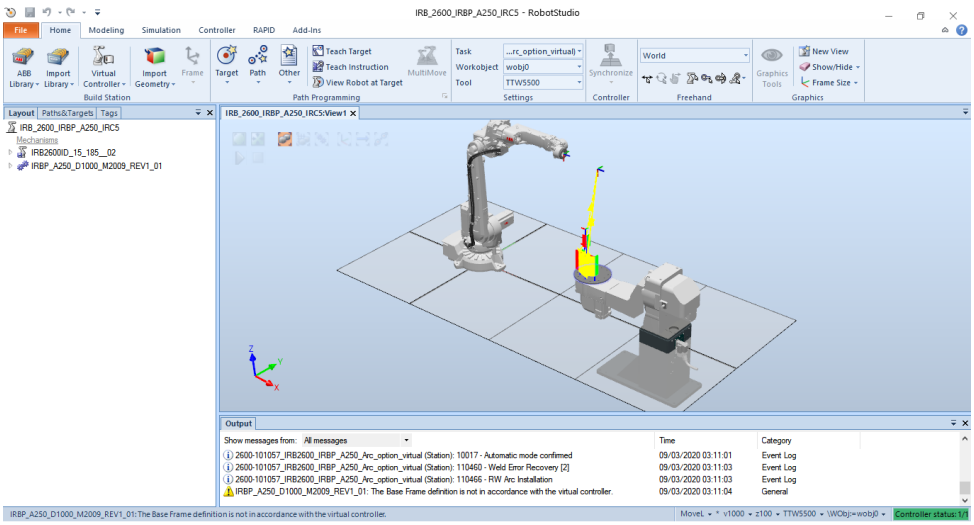


Figura 3.28 – Tela do RobotStudio

4 Resultados

Desenvolveu-se um programa na linguagem Python capaz de ler um programa em linguagem Karel, recalcular os pontos de movimentação de acordo com a estratégia adotada e escrever um programa em Rapid, sendo assim possível a execução no controlador do Robô.

4.1 Execução do Software de Fatiamento

O software de fatiamento contém muitas funcionalidades, porém para o desenvolvimento desse trabalho destaca-se apenas o código de saída em Karel e não se utilizarão as orientações da mesa posicionadora proposta nesse software. Sendo assim a execução do software é da seguinte maneira:

Ao executar o arquivo “Software de Fatiamento.exe” (contido na pasta do software), exibe-se a seguinte tela da Figura 4.29:

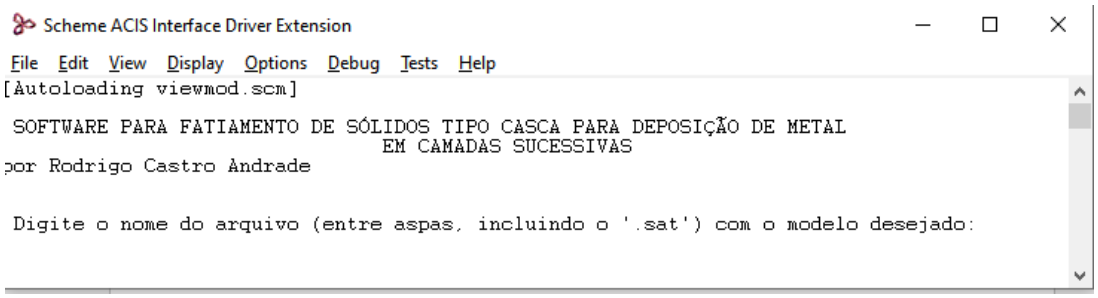


Figura 4.29 – Tela inicial do software de fatiamento

Fonte: (ANDRADE, 2013)

Pede-se para digitar o nome do arquivo do modelo, existente na pasta modelos (existem vários modelos já pré-produzidos), por exemplo utilizar-se-á o cubo. Depois de digitar o nome do arquivo são exibidas duas janelas, uma com o modelo 3D gerado (Figura 4.30) e outra solicitando a distância do fatiamento (Figura 4.31). Essa distância é muito importante pois ela define a altura da camada depositada.

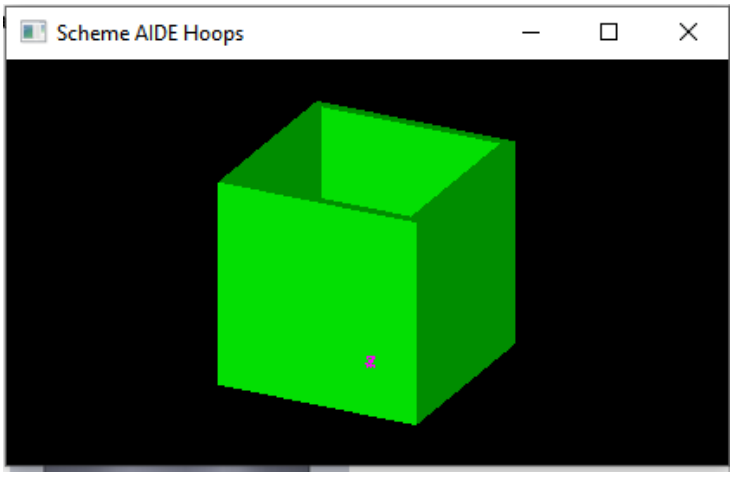


Figura 4.30 – Modelo 3D

Fonte: (ANDRADE, 2013)

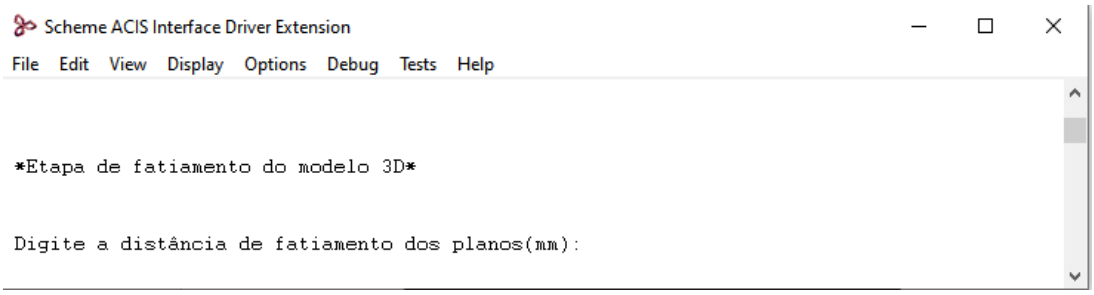


Figura 4.31 – Aquisição da distância entre os planos paralelos

Fonte: (ANDRADE, 2013)

Após essa etapa o software pergunta ao usuário se há mesa posicionadora (Figura 4.32). Como os pontos serão recalculados posteriormente basta clicar em “Não”.

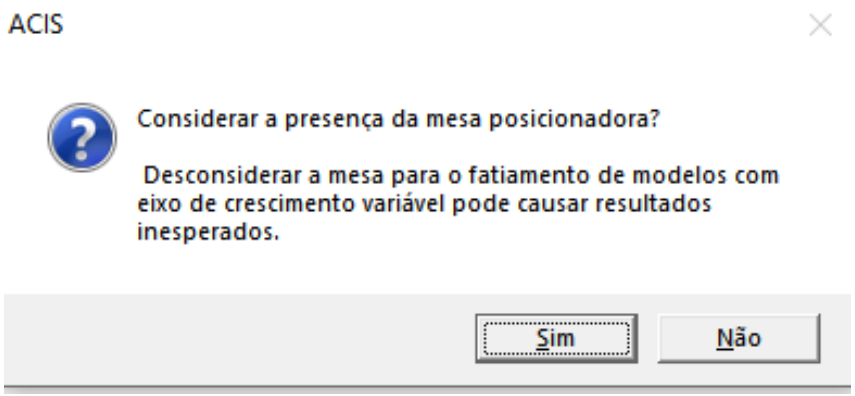


Figura 4.32 – Pergunta sobre a mesa posicionadora

Fonte: (ANDRADE, 2013)

Logo após pede-se para clicar na janela com o modelo 3D (ainda aberta) (Figura 4.33)

.

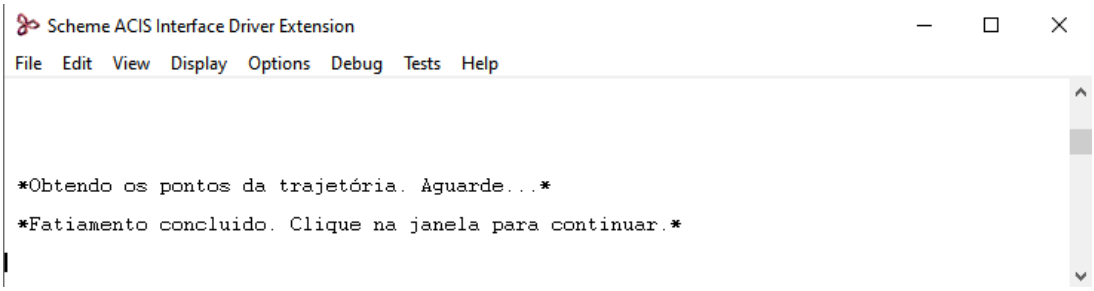


Figura 4.33 – clique sobre a janela

Fonte: (ANDRADE, 2013)

Ao fim do processo de fatiamento, os pontos da trajetória estão armazenados em uma lista. Logo após é exibida uma janela para selecionar a linguagem de saída do programa, “Sim” para WorkspaceLT (Karel) e “Não” para SportS3 (ARLA) (Figura 4.34). Opta-se pelo WorkSpaceLT (Karel).

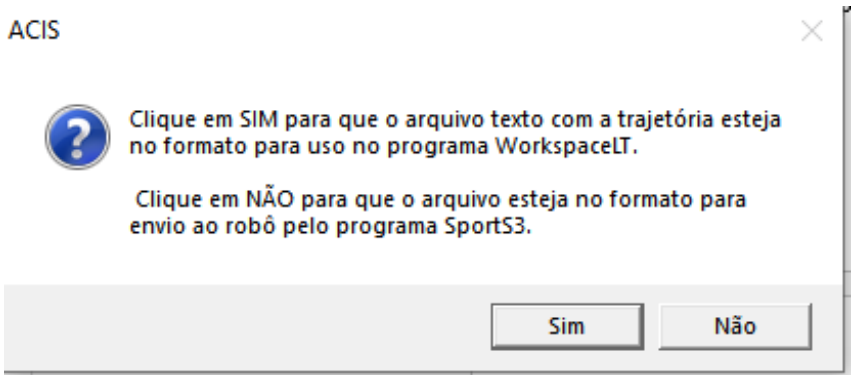


Figura 4.34 – Escolha entre WorkspaceLT ou SportS3

Fonte: (ANDRADE, 2013)

Depois pergunta-se sobre limitar as casas decimais (Figura 4.35), isso não é necessário pois posteriormente no programa em Python já é feita essa limitação. Não limitar pode minimizar eventuais erros de integração e aproximação no recalculo das coordenadas.

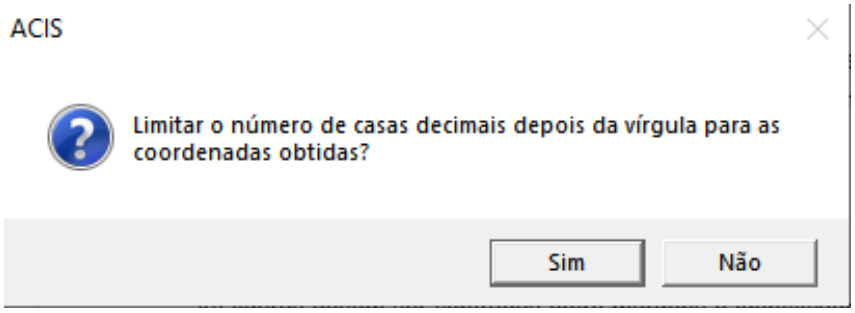


Figura 4.35 – Limitar casas decimais

Fonte: (ANDRADE, 2013)

Os pontos da trajetória são gravados no arquivo texto, e o usuário é questionado se ele deseja visualizar a trajetória obtida pelo processo de fatiamento (Figura 4.36) e então finalizar o programa (Figura 4.37.

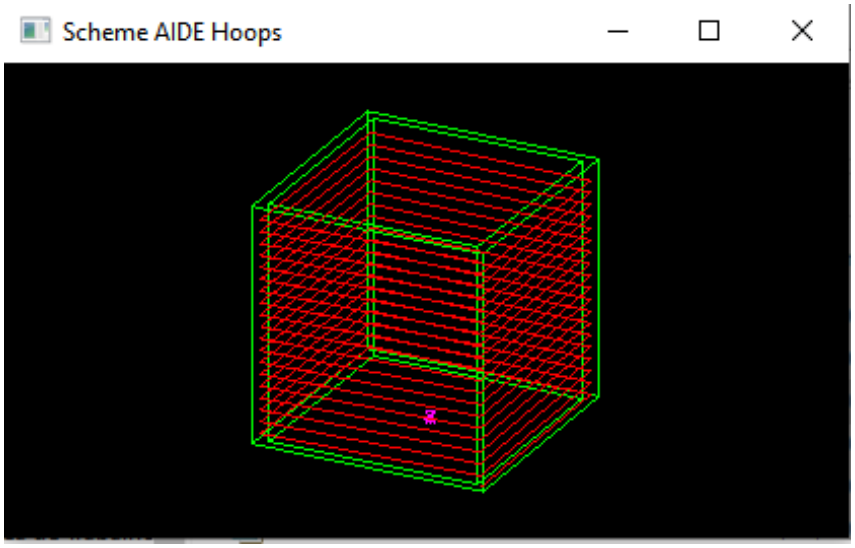


Figura 4.36 – Visualização da trajetória

Fonte: (ANDRADE, 2013)

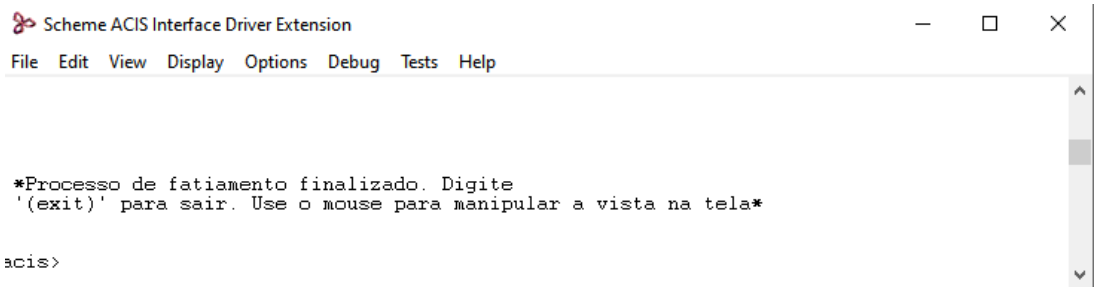


Figura 4.37 – Fim do processo

Fonte: (ANDRADE, 2013)

Após o final do processo são gerados dois arquivos texto com o nome “Track01.kl” e “Track01#.kl”, sendo apenas esse segundo relevante ao programa que será descrito na próxima seção. O primeiro arquivo contém as chamadas de movimentação e o segundo os dados das posições.

4.2 Desenvolvimento do Software KareltoRapid

O programa é dividido em quatro partes: **Leitura, Cálculo, Escrita e Interface Gráfica**. A parte Leitura é responsável pela leitura e tratamento das coordenadas obtidas do programa em Karel, na parte Cálculo são implementadas as estratégias de movimentação e

na parte Escrita, é escrito o arquivo Rapid. Abaixo tem um exemplo do arquivo de dados em Karel:

```
1     TP1 = POS(-95,0,0.1,0,0,0,'LUNT')
2     TP2 = POS(-92.9025,-19.8526,0.2173,0,0,0,'LUNT')
3     TP3 = POS(-90.3024,-29.5039,0.3345,0,0,0,'LUNT')
4     TP4 = POS(-86.7026,-38.8285,0.4518,0,0,0,'LUNT')
```

Como pode ser observado, em cada linha os três primeiros números são coordenadas XYZ e estão separados por vírgula. A cada leitura de uma linha, é feita a separação e armazenamento num vetor de três posições, e esse vetor armazenado em um outro vetor de tamanho variável, criando assim uma lista de posições.

4.2.1 Estratégias de Movimentação

Na deposição por GTAW a adição de arame deve ser majoritariamente na direção contrária ao movimento, como ilustrado na Figura 4.38. Dessa maneira a tocha giraria 360º a cada camada depositada. Porém, o punho do manipulador não pode girar infinitamente devido à restrição imposta pelo fabricante devido à torção das mangueiras. Portanto, visando contornar essa limitação, optou-se em girar o prato da mesa posicionadora (giro em torno do eixo z da mesa), pois não tem limite, e então recalcular todas as posições do manipulador para a sincronização das coordenadas do TCP e da mesa.

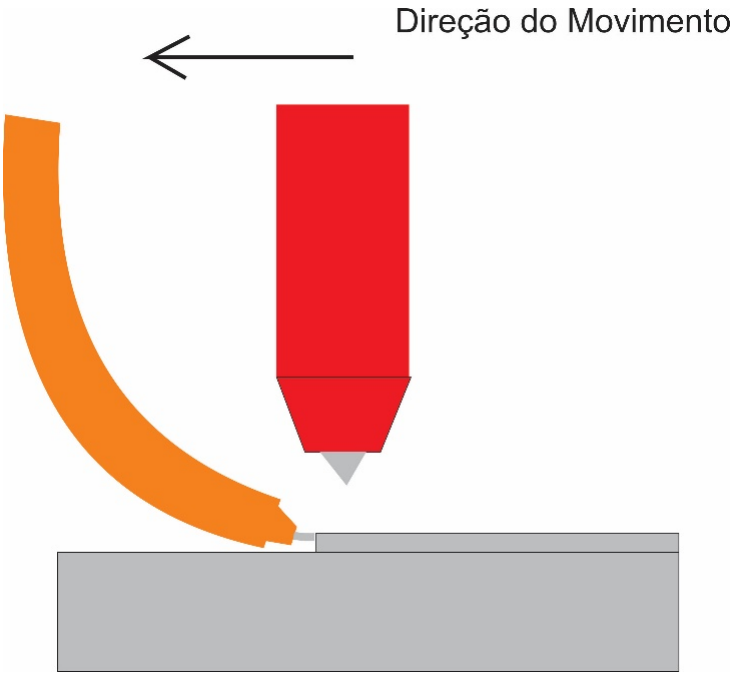


Figura 4.38 – Ilustração de movimentação da tocha GTAW

De início definem-se duas nomenclaturas para a orientação do TCP:

Orientação de movimentação do TCP - define a direção do movimento do TCP;

Orientação de deposição do TCP - define a direção em que está sendo feita a deposição de material.

4.2.1.1 Estratégia sem orientação da mesa

Nessa estratégia o plano de deposição é fixo, ou seja, não há movimento no braço da mesa posicionadora, apenas do prato. Como o movimento dos contornos da peça a fabricar são feitos através do giro da base, então pensou-se em utilizar coordenadas cilíndricas (r,θ,z), dessa maneira o TCP se movimenta em um raio e na direção z do objeto de trabalho, enquanto o giro do prato determina o ângulo θ.

Portanto, o movimento do TCP é limitado a um plano paralelo ao eixo y e z enquanto a mesa gira em um ângulo θ correspondente à coordenada cartesiana. Dessa forma, o programa basicamente converte as coordenadas cartesianas em coordenadas cilíndricas. Além disso, é recalculada a orientação de movimento do TCP (orientação em quaternion), para que a adição do arame esteja na direção contraria ao movimento relativo do TCP com a mesa. Essa estratégia é ilustrada na Figura 4.39.

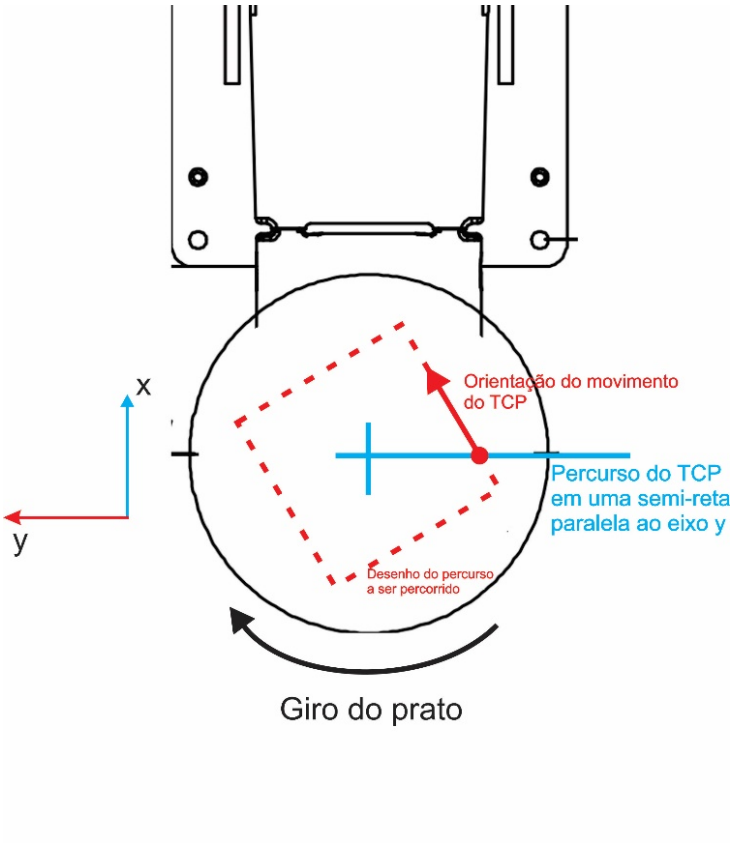


Figura 4.39 – Ilustração da estratégia por coordenadas cilíndricas

O algoritmo é executado da seguinte maneira:

1. Realiza-se um loop para recalcular todas as posições;

2. Dentro do loop adicionam-se pontos intermediários nas trajetórias de tal maneira que a distância entre os pontos seja menor que um limite pré-estabelecido (D_{Max}) e assim poder executar a interpolação de tais pontos. Os pontos adicionais serão necessários para as interpolações das trajetórias.

   Esses pontos são adicionados por meio da parametrização de uma semi-reta entre os pontos. A parametrização é realizada de acordo com a Equação 4.1 (onde i define a iteração no loop):

$$P = P[i-1] + t(P[i] - P[i-1]), t \in [0,1] \tag{4.1}$$

   Determina-se o número de pontos intermediários a serem adicionados na trajetória. Para isso faz-se a divisão entre o modulo da diferença entre o P[i] e P[i-1] e a distância máxima entre os pontos (Equação 4.2). Como não é uma divisão inteira, é feito o arredondamento para cima para satisfazer a condição de distância máxima entre os pontos.

$$n = \frac{|P[i] - P[i-1]|}{D_{max}} \tag{4.2}$$

   Portanto, definem-se os pontos intermediários substituindo t na Equação 4.1 e variando j de 1 até $n-1$. Dessa forma a cada iteração adiciona-se a o ponto P_{int} calculado à lista de coordenadas.

$$t = \frac{j}{n}, j \in [1,n-1] \tag{4.3}$$

$$P_{int} = P[i-1] + \frac{j}{n}(P[i] - P[i-1]), j \in [1,n-1] \tag{4.4}$$

3. Com os pontos intermediários adicionados à lista, esses pontos em coordenadas cartesianas (x, y, z) são convertidos, então, em coordenadas cilíndricas (r, θ, z).

4. A construção das coordenadas do TCP do robô é feita da seguinte maneira: $x_i = 0$, $y_i = r$ e $z_i = z$;

5. Adiciona se o ângulo $\theta + 360n$ na lista de ângulos (em graus) do prato da mesa, sendo "n" o número de voltas da mesa.

6. Calcula-se o quaternion referente a orientação de movimentação do TCP e o adiciona em uma lista.

4.2.1.2 Estratégia com orientação da mesa

No processo de deposição há a possibilidade de ocorrer escorrimento da poça na confecção de peças com a superfície inclinada, causando imperfeições no acabamento da peça ou até impossibilitando a fabricação. Devido a esse problema desenvolveu-se uma segunda estratégia de movimentação. Optou-se a orientar a mesa, girando-a em torno do eixo x, de tal maneira que a superfície que esteja sendo trabalhada fique ortogonal à gravidade, conforme ilustrado na Figura 4.40.

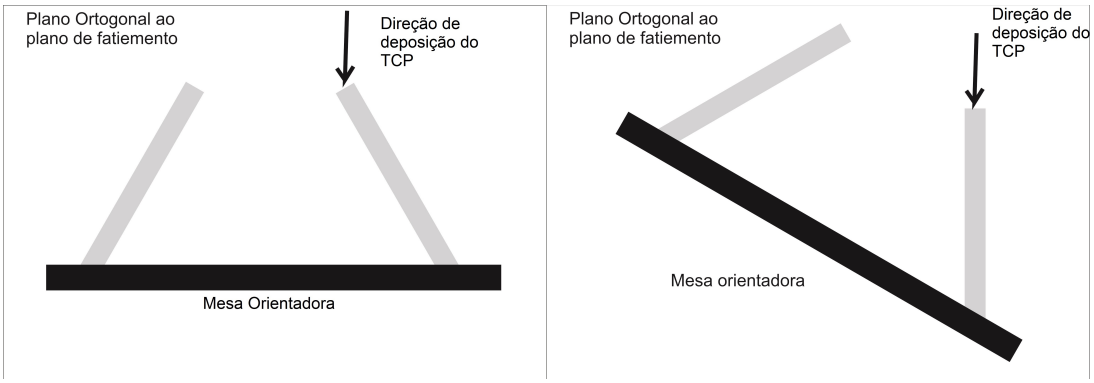


Figura 4.40 – Ilustração da orientação da mesa (vista lateral)

Nessa nova abordagem, tornou-se inviável a utilização das coordenadas cilíndricas, pois seria impossível garantir a ortogonalidade da superfície à direção da gravidade.

Portanto, adota-se uma estratégia diferente: mantem-se a orientação do TCP fixa e realiza-se a orientação da coordenada utilizando a mesa posicionadora. A orientação da mesa é feita apenas no eixo x, então mantém-se a movimentação cartesiana do TCP restrita apenas a movimentos paralelos a esse eixo, exceto quando ocorre o giro da mesa. A projeção da região de atuação do TCP decorrente dessa estratégia é mostrada na Figura 4.41.

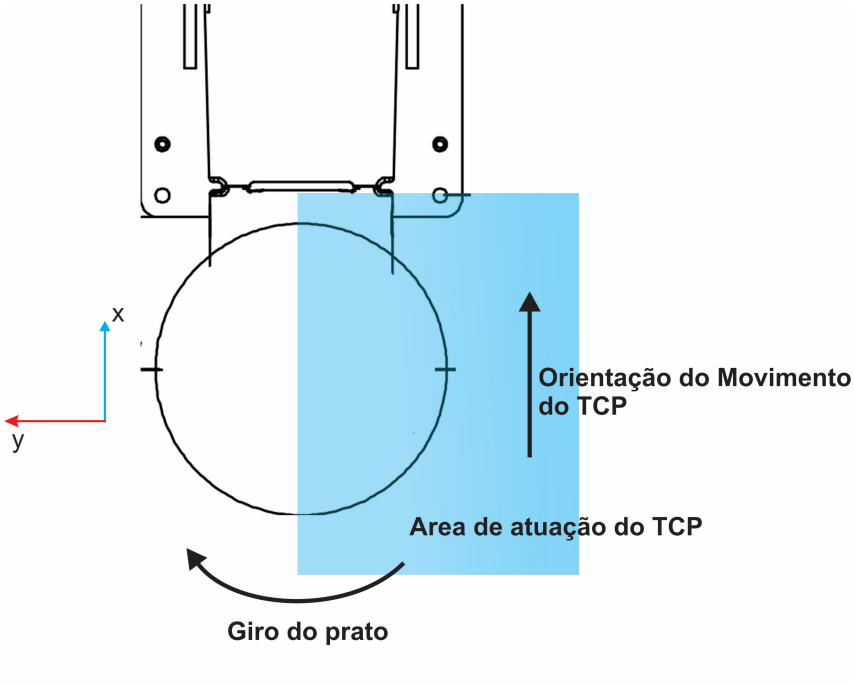


Figura 4.41 – Ilustração da estratégia de movimentação com mesa posicionadora

O algoritmo é executado da seguinte maneira:

1. Realiza-se um loop para recalcular todas as posições;
2. Faz-se a diferença vetorial entre o ponto atual da iteração (P[i]) e ponto anterior (P[i-1]) para obter o vetor deslocamento. (Figura 4.42(a));

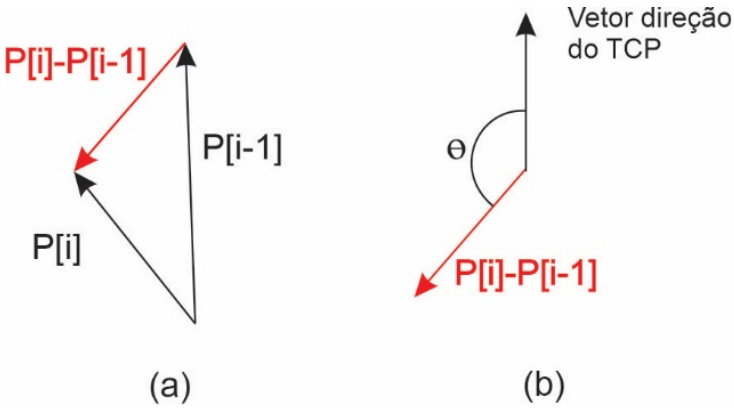


Figura 4.42 – Orientação do vetor velocidade

3. Esse vetor deve estar na mesma direção de movimento do TCP, paralelo ao eixo x, então calcula-se o ângulo θ entre o vetor do TCP desejado e o vetor deslocamento;
4. Rotaciona o ponto P[i] com o ângulo θ calculado e adiciona na nova lista de pontos;
5. Adiciona-se o ângulo $\theta + 360n$ na lista de ângulos (em graus) do prato da mesa, sendo "n" o número de voltas da mesa.

6. O deslocamento angular do prato da mesa pode dessincronizar o ponto do referencial da mesa com o ponto do referencial do TCP, pois a mesa realiza um movimento circular e o TCP linear, além de exigir uma alta aceleração e velocidade do manipulador. O problema é ilustrado na Figura 4.43.

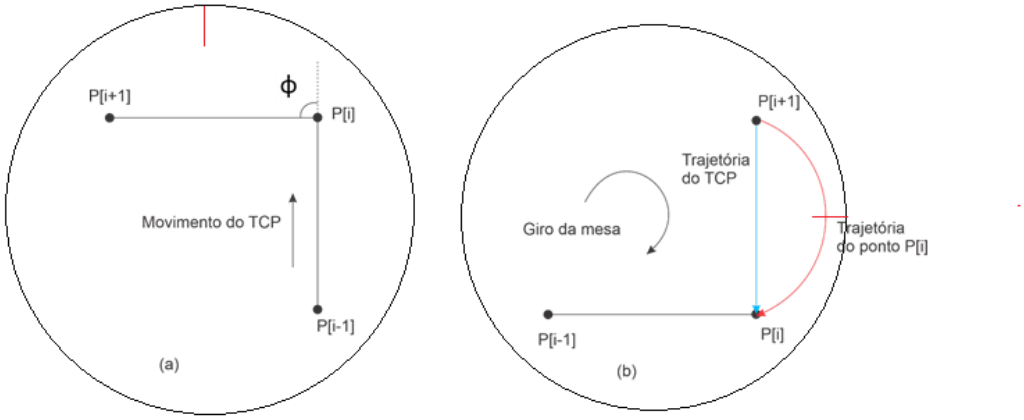


Figura 4.43 – Problema na trajetória

Para solucionar esse problema, primeiro é necessário detectar os pontos em que há deslocamento angular:

- Primeiro calcula-se a diferença vetorial entre o ponto atual (P[i]) e o anterior (P[i-1]) (como descrito anteriormente na Figura 4.42) e a diferença vetorial entre o ponto seguinte P[i+1] e o atual (P[i]), obtendo-se o deslocamento entre os três pontos, como na Figura 4.44(a)
- Compara-se o ângulo (ϕ) entre as duas diferenças, se for maior que 0 (considerando um erro nesse ângulo*), é detectado o deslocamento angular. Figura 4.44(b).

Obs.: *Foi determinado erro de $\phi > 2^o$, ou seja, uma alteração na trajetória menor que 2^o ainda é considerado interpolação linear.

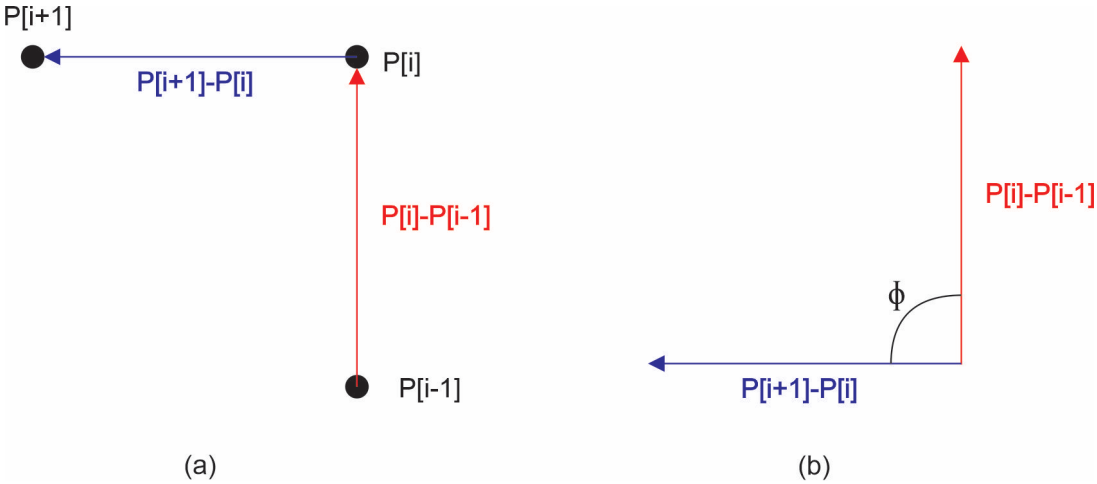


Figura 4.44 – Detecção de deslocamento angular da mesa

Se detectado deslocamento angular, parametriza-se uma semi-reta entre P[i-1] e P[i] com a seguinte equação:

$$P = P[i-1] + t(P[i] - P[i-1]), t \in [0,1] \tag{4.5}$$

Depois são adicionados dois pontos intermediários Pi1 e Pi2 de tal maneira que fiquem próximos ao ponto P[i]. Esses pontos podem ser calculados com as seguintes equações:

$$Pi_1 = P[i-1] + t_1(P[i] - P[i-1]) \tag{4.6}$$

$$Pi_2 = P[i-1] + t_2(P[i] - P[i-1]) \tag{4.7}$$

onde t1 e t2 são definidos de tal maneira que as distâncias entre os pontos P[i],Pi_1 e Pi_2 seja menor que uma distância máxima pré-estabelecida (D_{Max}), assim como foi feito na estratégia de coordenadas cilíndricas (Equação 4.8) . t_1 e t_2 são definidos pela Equação 4.9 e Equação 4.10, respectivamente.

$$n = \frac{|P[i] - P[i-1]|}{D_{max}} \tag{4.8}$$

$$n = \frac{n-2}{n} \tag{4.9}$$

$$n = \frac{n-1}{n} \tag{4.10}$$

Dessa forma os pontos Pi_1 e Pi_2 estão bem próximos de P[i] a uma distância menor que D_{Max}. Então os pontos adicionais são representados na Figura 4.45.

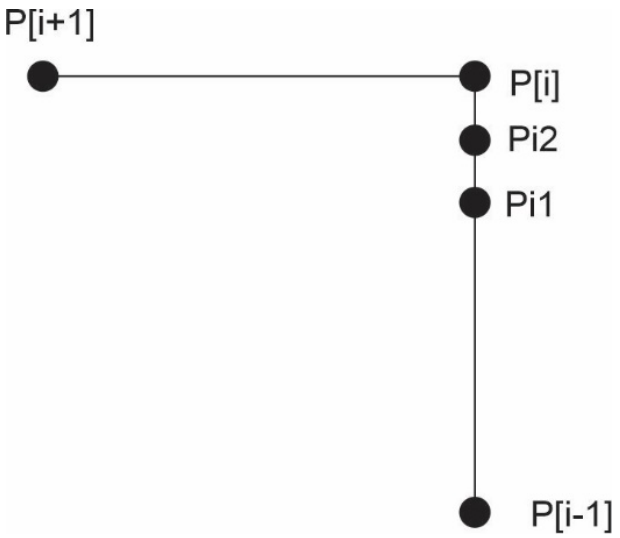


Figura 4.45 – Pontos intermediários (as distancias entre os pontos estão exageradas para melhor compreensão)

Então o TCP se movimenta com interpolação linear do ponto P[i-1] até o Pi1 (Figura 4.46(a)), rotaciona-se o ponto P[i] como o ângulo ϕ calculado na Figura 4.44, rotaciona-se o ponto Pi_2 com o ângulo $\phi/2$, então utiliza-se uma interpolação circular entre o ponto Pi_1 e os pontos Pi_2 e $P[i]$ rotacionados. Dessa forma sincroniza-se a trajetória do prato da mesa posicionadora com a trajetória do TCP (Figura 4.46(b)).

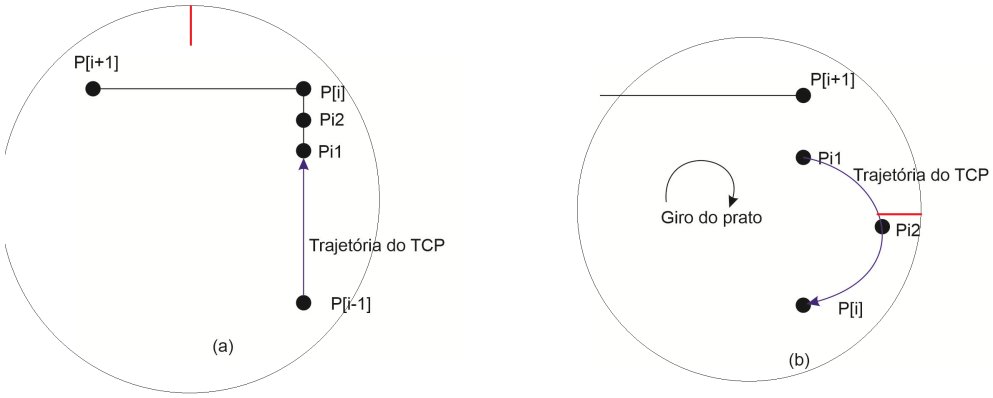


Figura 4.46 – Interpolação circular

Essa solução descrita acima cria alguns erros de trajetória, quanto menor a distância entre os pontos intermediários menor é esse erro, porém implica em maiores velocidades e acelerações no TCP, o que pode ocasionar vibrações indesejadas.

7. Determinação da inclinação da mesa (ângulo α, em torno do eixo x no referencial da mesa).

É possível determinar quantas camadas serão depositadas sem orientação, visto que as camadas mais baixas podem-se considerar o escoamento da poça desprezível, sendo que no mínimo a primeira camada deve ser sem orientação.

A rotação das coordenadas deve ser em relação ao referencial da mesa posicionadora, então o eixo x deve coincidir com a posição da mesa na posição plana.

O algoritmo fica da seguinte maneira:

- Após a primeira ou mais camadas, calcula-se a inclinação do ponto P[i] com relação ao ponto logo abaixo no plano yz e então calcula-se o ângulo $\alpha[i]$.
- Se houver deslocamento angular em torno do eixo x, ou seja, $\alpha[i] - \alpha[i-1] > \epsilon$ (dentro de um erro estabelecido) então:
  - Se a movimentação do TCP for do tipo linear então:
    - Adiciona-se um ponto intermediário Pi_1 exatamente na metade do caminho entre $P[i-1]$ e $P[i]$, utilizando a Equação 4.5 e substituindo $t = 0.5$;
    - Rotaciona Pi_1 entorno do eixo x com ângulo $\alpha[i]/2$ e $P[i]$ com ângulo $\alpha[i]$, adiciona-se $\alpha[i]/2$ e $\alpha[i]$ na lista de ângulos α e define interpolação circular entre os pontos, dessa forma garante a sincronização da mesa com TCP;
  - Se a movimentação do TCP for tipo circular então:
    - Utiliza-se o ponto intermediário Pi_1 já adicionado anteriormente;
    - Assim como anteriormente, rotaciona Pi_1 em torno do eixo x com ângulo $\alpha[i]/2$ e $P[i]$ com ângulo $\alpha[i]$, adiciona-se $\alpha[i]/2$ e $\alpha[i]$ na lista de ângulos α e executa interpolação circular entre os pontos;
  - Se não houver deslocamento angular, então:
    - Apenas se rotaciona o ponto $P[i]$ (e o ponto intermediário no caso de interpolação circular) com o ângulo $\alpha[i]$, e faz-se a interpolação já estabelecida anteriormente.

4.2.1.3 Cálculo da velocidade relativa

No início da execução do programa é solicitado ao usuário a velocidade linear (v_{linear}) do TCP, porém todos os pontos foram modificados, então é necessário também recalcular as velocidades para que a velocidade do TCP relativa ao prato ($v_{relativa}$) permaneça constante. Isso é possível comparando as distâncias dos pontos originais ($d_{original}$) com as distâncias dos pontos recalculados ($d_{recalculada}$), o tempo em que o TCP deveria percorrer as duas distancias deve ser igual, assim se calcula da seguinte maneira:

$$\frac{v_{linear}}{d_{original}} = \frac{v_{relativa}}{d_{recalculada}} = tempo\ de\ trajetoria \tag{4.11}$$

$$v_{relativa} = \frac{d_{recalculada}}{d_{original}} v_{linear} \tag{4.12}$$

Também são calculadas as velocidades angulares da movimentação do prato da seguinte maneira:

$$\omega = \frac{deslocamento\ angular}{tempo\ de\ trajetoria} \tag{4.13}$$

Dessa forma, não somente a trajetória, mas também as velocidades do TCP e do prato estão sincronizadas.

4.2.2 Escrita do Programa em Rapid

No final de cada estratégia obtêm-se algumas listas de dados necessários para a escrita do código em RAPID: uma lista com as posições x,y,z do TCP; uma lista com os ângulos do prato da mesa posicionadora, uma lista com as velocidades lineares do TCP e angulares da mesa; No caso da estratégia sem orientação, uma lista com os quatérnions de orientação do TCP e no caso da estratégia com orientação, uma lista com os ângulos de orientação da superfície da peça.

O código é escrito em um arquivo chamado “MainModule.mod” contido na pasta “proto”. Essa pasta contém também os arquivos “TCP_TTW5500_1.mod”, onde são definidas as coordenadas do TCP, “weld.mod”, onde se encontram as rotinas de soldagem e “proto.pgf”, responsável pelo carregamento do programa RAPID no controlador do robô. Obs.: Os arquivos “TCP_TTW5500_1.mod” e “weld.mod” serão discutidos posteriormente nesse trabalho.

1. Inicia-se a escrita definindo a estrutura do programa e as variáveis básicas como o sistema de coordenadas da ferramenta (tooldata) e do objeto de trabalho (wobjdata).

   As coordenadas do objeto de trabalho é a posição do centro da mesa posicionadora.

   As coordenadas da ferramenta são definidas por meio de uma rotina presente no FlexPendant do controlador.

```
1         MODULE MainModule
2             TASK PERS tooldata TTW5500_1 := [TRUE, [[32.1099,
                  -0.389434, 387.111], [1,0,0,0]] , [1,[0,0,100]
                  , [1,0,0,0],0,0,0]];
```

```
3       PERS wobjdata wobj0 := [ FALSE, TRUE, "", [[0.0000,
           0.0000, 0.0000], [1,0,0,0]], [[0,0,0], [1,0,0,0] ]
           ];
```

2. Faz-se um loop para a escrita das variáveis de posição (robtarget):

   Para o algoritmo sem orientação:

```
1       CONST robtarget P[i] := [x[i], y[i], z[i]], [q0[i],
           q1[i], q2[i], q3[i]], [-1, 0, -1, 0] , [9E+09, 0, θ
           [i], 9E+09, 9E+09, 9E+09]];
```

   Para o algoritmo com orientação:

```
1       CONST robtarget P[i] := [x[i], y[i], z[i]], [0, 0, 1,
           0], [-1, 0, -1, 0] , [9E+09, α[i], θ[i], 9E+09,
           9E+09, 9E+09]];
```

   P[i] - Nome da variável;

   x[i], y[i], z[i] – lista de posições do TCP;

   q0[i], q1[i], q2[i], q3[i] – quaternions de orientação do TCP;

   θ[i] – ângulo de giro do prato;

   α[i] – ângulo de orientação da mesa.

3. Faz-se um loop para a escrita das variáveis de velocidade (speeddata):

```
1       CONST speeddata vd[i] :=[vl[i],500,5000,ω[i]];
```

   Observe que são apenas utilizadas as velocidades vl[i] (velocidade linear do TCP) e ω[i] (velocidade angular da mesa). As outras velocidades estão no valor máximo pois o controlador já limita pela menor velocidade.

4. Escrevem-se as rotinas de soldagem pré-determinadas no arquivo “weld.mod”;

```
1       PROC stand()
2       ...
3       ENDPROC
4       PROC start0()
5       ...
6       ENDPROC
7       PROC stop0()
```

```
8           ...
9           ENDPROC
```

5. Escreve-se a função principal (main). É nessa função onde serão introduzidas as instruções de movimentação e soldagem.

```
1           PROC main()
2               ActUnit STN1;
3               MoveJ p1, vl1, z5, TTW5500_1 \WObj := wobj0;
4               stand;
5               start0;
```

A instrução ActUnit tem a função de ativar os motores da mesa, que está configurada no controlador com o nome “STN1”. É feita a movimentação para o primeiro ponto (p1) para, depois disso, dar início aos procedimentos de soldagem. Na rotina “stand;” definem-se os valores iniciais das variáveis de saída para a fonte de solda. Na rotina “start0;” definem-se os comandos para abertura do arco e início da deposição.

6. É realizado um loop para a adição das instruções de movimentação:

   Para interpolação linear:

```
1               MoveL p[i], vl[i], z5, TTW5500_1 \WObj := wobj0;
```

   Para interpolação circular:

```
1               MoveC p[i], p[i+1], vl[i], z5, TTW5500_1 \WObj :=
                    wobj0;
```

7. Adiciona-se a rotina de parada de soldagem e uma última instrução de movimentação para o afastamento da ferramenta com a peça já fabricada:

```
1               stop0;
2            MoveL p_final, vl_final, z5, TTW5500_1 \WObj :=
                wobj0;
```

8. E por fim a finalização do programa:

```
1               ENDPROC
2           ENDMODULE
```

Por fim, um exemplo de um programa completo:

```
1         MODULE MainModule
2           TASK PERS tooldata TTW5500_1 := [TRUE,
               [[32.1099,-0.389434,387.111] , [1,0,0,0]],
               [1,[0,0,100],[1,0,0,0],0,0,0]];
3           PERS wobjdata wobj0:= [ FALSE,TRUE,"",
               [[0.0000,0.0000,0.0000], [1,0,0,0]], [[0,0,0],
               [1,0,0,0] ] ];
4           CONST robtarget p1 := [[-141.4137,-134.1917,0.1000]
               ,[0,0,1,0], [-1,0,-1,0], [9E+09,-0.00,
              0.00,9E+09,9E+09,9E+09]];
5           CONST robtarget p2 :=
               [[123.3687,-134.1917,2.5628],[0,0,1,0],
               [-1,0,-1,0],
               [9E+09,-0.00,-0.00,9E+09,9E+09,9E+09]];
6           CONST robtarget p3 := [[-3.8557,-183.5819,2.5814],
               [0,0,1,0], [-1,0,-1,0],
               [9E+09,-0.00,-44.25,9E+09,9E+09,9E+09]];
7           CONST robtarget p4 := [[-130.8100,-130.8100,2.6000]
               ,[0,0,1,0], [-1,0,-1,0],
               [9E+09,-0.00,-88.50,9E+09,9E+09,9E+09]];
8           CONST robtarget p5 := [[126.8102,-130.8100,5.0618]
               ,[0,0,1,0], [-1,0,-1,0],
               [9E+09,-0.00,-88.50,9E+09,9E+09,9E+09]];
9         ....
10              VAR speeddata vd1 :=[10.00,500,5000,500.00];
11              VAR speeddata vd2 :=[10.00,500,5000,1.00];
12              VAR speeddata vd3 :=[635.43,500,5000,221.25];
13              VAR speeddata vd4 :=[10.00,500,5000,1.00];
14              VAR speeddata vd5 :=[644.05,500,5000,225.00];
15              ...
16              PROC stand()
17              ...
18              ENDPROC
19              PROC start0()
20              ...
21              ENDPROC
22              PROC stop0()
23              ...
24              ENDPROC
25              PROC main()
26              ActUnit STN1;
27              MoveJ p1, vd1, z5, TTW5500_1 \WObj:=wobj0;
28              stand;
29              start0;
30              MoveL p2, vd2, z5, TTW5500_1 \WObj:=wobj0;
31              MoveC p3, p4, vd3, z5, TTW5500_1 \WObj:=wobj0;
32              MoveL p5, vd4, z5, TTW5500_1 \WObj:=wobj0;
33              ...
34                  stop0;
```

```
35              MoveL p227, v10, z10, TTW5500_1 \WObj:=wobj0;
36            ENDPROC
37          ENDMODULE
```

4.2.3 Estrutura do software

O software é composto por 4 módulos: interface gráfica (GUI.py), módulo sem orientação (coordenadas cilíndricas) (sem orientação.py), módulo com orientação (com orientação.py) e modulo de funções com cálculos recorrentes no algoritmo (calc.py). Foi utilizado o Python 3.10 para a programação e importaram-se algumas bibliotecas extras na programação:

- Numpy – é uma biblioteca que suporta o processamento de vetores e matrizes, juntamente com uma grande coleção de funções matemáticas de alto nível para operar sobre estas matrizes.
- Tkinter – é utilizado para gerar a interface gráfica.
- OS – é utilizado para executar chamadas externas do sistema, como abrir programas externos.
- Shutil – é utilizado para operações de alto nível em arquivos e diretórios.

O modulo de interface gráfica (GUI.py) é o modulo principal, lá estão as funções onde são feitas as configurações e chamadas dos demais módulos. Por tanto é a partir de modulo que será feita a compilação do programa.

É utilizado o software “Pyinstaller” e o “Auto py to exe” para a compilação do programa para que não seja necessário ter instalado o Python na máquina que executará o programa. As seguintes configurações de compilação são necessárias: Localização do arquivo “GUI.py”, será em um diretório, será baseado em janela (o console estará oculto), a localização do ícone do executável (opcional), localizações da pasta do software de fatiamento e da pasta “proto” que contém com alguns módulos em Rapid: parâmetros de soldagem (weld.mod), parâmetros sobre o TCP (TTW_5500_1.mod) e o arquivo de inicialização no controlador do robô (proto.pgr) e definir o nome do executável (Karel to Rapid). As configurações são ilustradas na Figura 4.47.

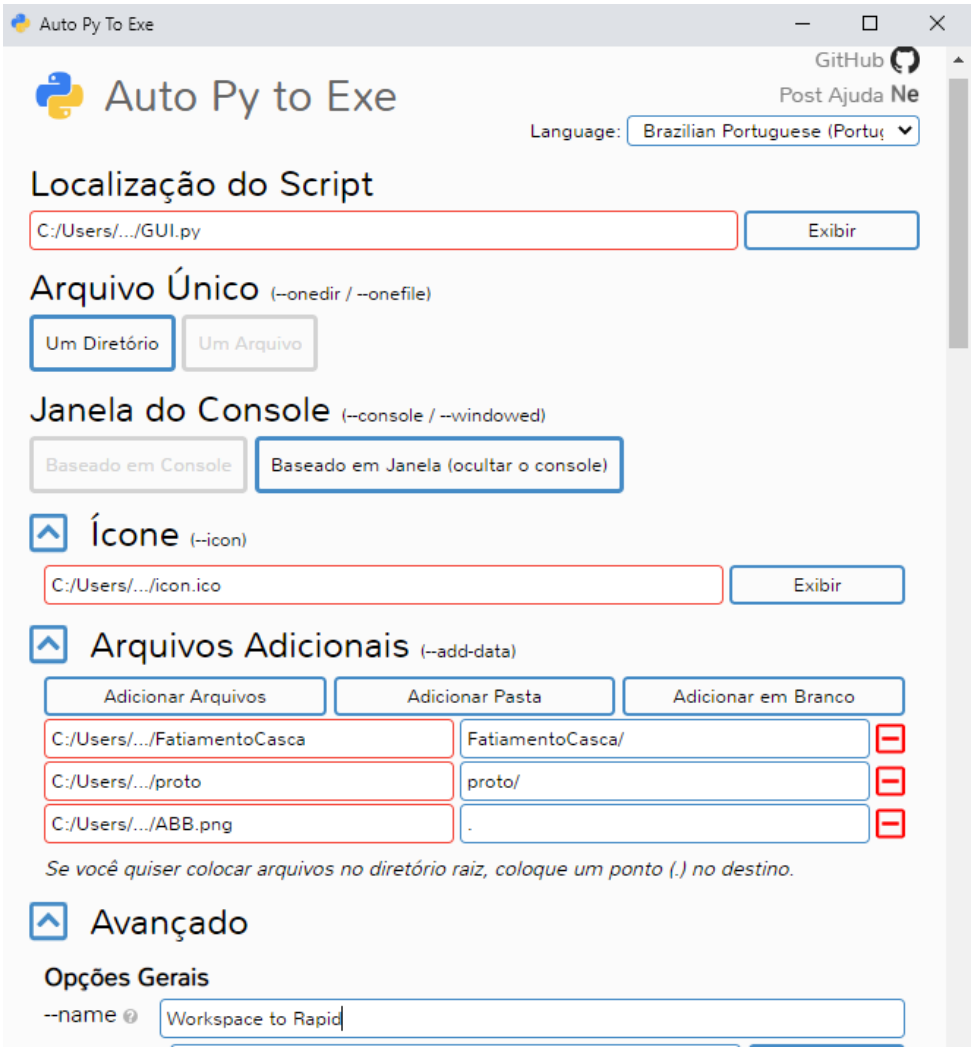


Figura 4.47 – Auto py to exe

4.2.4 Execução do software

Após a compilação é gerada uma pasta contendo todos os arquivos necessários para execução. Para abrir o programa executa-se o arquivo “Karel to Rapid.exe” e aparecerá a tela inicial do programa Figura 4.20.

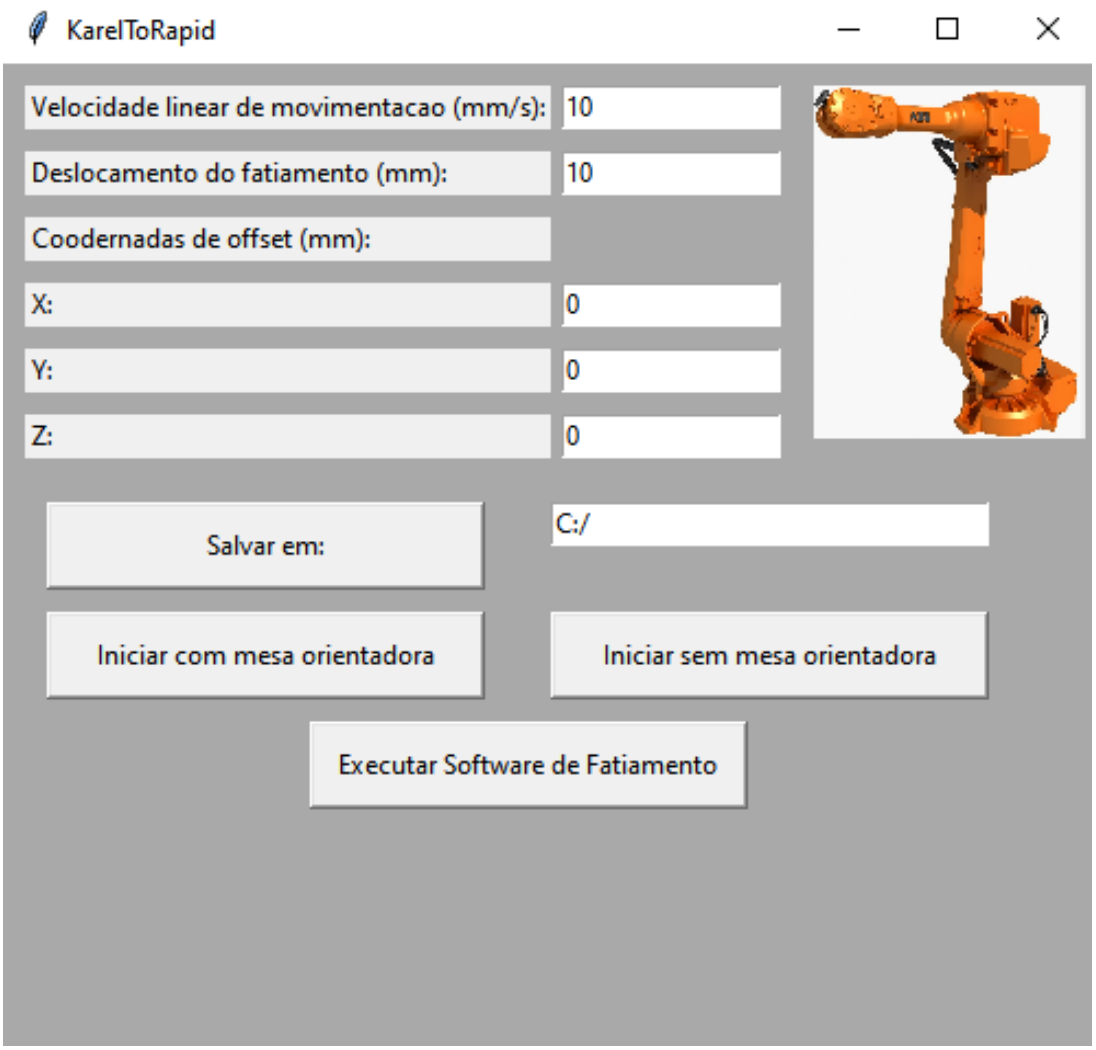


Figura 4.48 – Tela inicial do Karel to Rapid

Caso esteja executando pela primeira vez ou deseja mudar o modelo 3d, primeiro executa-se o software de fatiamento (ANDRADE, 2013), de acordo explicado na seção 4.1. Após a execução é gerado um arquivo chamado “Track01#.kl” na mesma pasta do software de fatiamento e nele estará escrito o programa em Karel.

É necessário informar a velocidade linear do TCP (relativa a base, ou seja, como se não houvesse uma mesa posicionadora) o deslocamento do fatiamento, ou seja, o distanciamento entre as camadas, as coordenadas XYZ de offset da mesa (o deslocamento do sistema de coordenadas do centro da mesa com relação ao sistema de coordenadas do robô) e o diretório onde será salvo o programa em Rapid.

Caso não exista algum dos arquivos necessários no diretório de saída, como os arquivos “weld.mod” ou o “TTW_5500_1.mod”, eles serão copiados para o diretório, caso existam eles serão utilizados para a execução.

Após a execução será gerado um arquivo “MainModule.mod” que contém o programa completo em Rapid.

Por fim, basta copiar a pasta “proto”, contida na pasta de saída, para o controlador do robô através do RobotStudio.

4.3 Configuração das rotinas de deposição

As variáveis de sinais da fonte MW 5000 são mapeadas no controlador do robô partir da rede LocalNet da Fronius, por meio de um acoplador DeviceNet para K-bus Beckhoff BK-5200 e uma interface de comunicação serial KL6021, que juntos convertem os sinais da rede LocalNet (RS485) para a rede DeviceNet, usada no controlador IRC5. (MIRANDA, 2018)

Os sinais que a fonte envia para o controlador são mapeadas como variáveis de entrada, e as que o controlador envia para fonte são mapeadas como variáveis de saída. Tais variáveis podem ser acessadas e manipuladas por meio de programas em RAPID.

O mapeamento dessas variáveis foi feito no trabalho de graduação MIRANDA (2018). A Tabela 4.1 mostra o mapeamento definido no controlador IRC5:

Tabela 4.1 – Sinais de entrada e saída da fonte de solda (do ponto de vista do controlador)

Sinal	**Tipo**	**Endereço no controlador**
doFr1ArcOn	Digital Output	0
doFr1RobotReady	Digital Output	1
doFr1OpMode0	Digital Output	2
doFr1OpMode1	Digital Output	3
doFr1OpMode2	Digital Output	4
doFr1MasterSelectionTwin	Digital Output	5
doFr1GasTest	Digital Output	8
doFr1FeedForward	Digital Output	9
doFr1FeedRetract	Digital Output	10
doFr1ErrorReset	Digital Output	11
doFr1TouchSense	Digital Output	12
doFr1ColdWireDisable	Digital Output	13
aoFr1JobNumber	Analog Output	16-23
doFr1DC_AC	Digital Output	24
doFr1DCn_DCp	Digital Output	25
doFr1CapShaping	Digital Output	26
doFr1PulseDisable	Digital Output	27
doFr1PulseRange0	Digital Output	28
doFr1PulseRange1	Digital Output	29
doFr1PulseRange2	Digital Output	30
doFr1WeldingSim	Digital Output	31
aoFr1Power	Analog Output	32-47
aoFr1BaseCurrent	Analog Output	64-71
aoFr1DutyCycle	Analog Output	72-79
doFr1BaseCurrentDisable	Digital Output	82
doFr1DutyCycleDisable	Digital Output	83
aoFr1WireSpeedWfi	Analog Output	86-95
diFr1ArcStable	Digital Input	0
diFr1ProcessActv	Digital Input	2
diFr1MainCurrent	Digital Input	3
diFr1TorchColisn	Digital Input	4
diFr1WelderReady	Digital Input	5
diFr1CommunicRdy	Digital Input	6
aiFr1ErrorNumber	Analog Input	08-15
diFr1HighFrequenceActive	Digital Input	25
diFr1WireAvailable	Digital Input	27
diFr1PulseHigh	Digital Input	30
aiFr1WeldingVoltage	Analog Input	32-47
aiFr1WeldingCurrent	Analog Input	48-63
aiFr1MotorCurrent	Analog Input	64-71
aiFr1ArcLength	Analog Input	72-79
aiFr1WireFeedSpeed	Analog Input	80-95

Fonte: MIRANDA (2018)

A variáveis com prefixo:

- “diFr1” – *“Digital input”* (entrada digital) são sinais digitais (apenas 1 bit - 1 ou 0) enviados da fonte para o controlador.
- “aiFr1” – *“Analog input”* (entrada analógica) são sinais analógicos (palavra binaria)

enviados da fonte para o controlador.

- “doFr1” – *“Digital output”* (saída digital) são sinais digitais (apenas 1 bit - 1 ou 0) enviados do controlador para a fonte.
- “aoFr1” – *“Analog output”* (saída analógica) são sinais analógicos (palavra binaria) enviados do controlador para a fonte.

É possível comandar a fonte por meio das variáveis do tipo digital output e analog output. No caso das variáveis “doFr1” utilizam-se as funções no RAPID “set” e “reset”, ou seja, atribui-se 1 ou 0 para a variável com essas funções, respectivamente. No caso das variáveis “aoFr1” utiliza-se a função “setAO”, onde se atribui um número real ao sinal.

```
1    ...
2    reset doFr1FeedRetract;
3    setAO aoFr1WireSpeedWfi , 400;
4    set doFr1FeedRetract;
5    ...
```

No modulo “weld.mod” já estão escritas 3 rotinas de soldagem: stand, start0, stop0. Na rotina “stand”, atribuem-se valores às variáveis iniciais com objetivo de configuração do processo, ou seja, é definido o tipo de soldagem (TIG), velocidade de alimentação de arame, corrente do arco elétrico, entre outras. Na rotina “start” definem-se as variáveis para a abertura do arco elétrico e o início do processo de deposição, ao contrário da rotina “stop”, que define a parada do processo. É possível modificar as rotinas para determinado processo antes da execução das estratégias de movimentação para que o software adicione tais rotinas ao código RAPID final.

4.4 Projeto de fabricação de uma base de deposição

O projeto foi pensado para prover uma base plana, livre de cabos e mangueiras, sobre a qual a base de deposição seria fixada, considerando seu centro como referência para construção da peça fatiada, para que a base fique alinhada ao eixo de inclinação da mesa e para haver uma melhor dissipação de calor na deposição. A Figura 4.49 e a Figura 4.50 ilustra o desenho técnico da base a Figura 4.51 ilustra a montagem na mesa posicionadora.

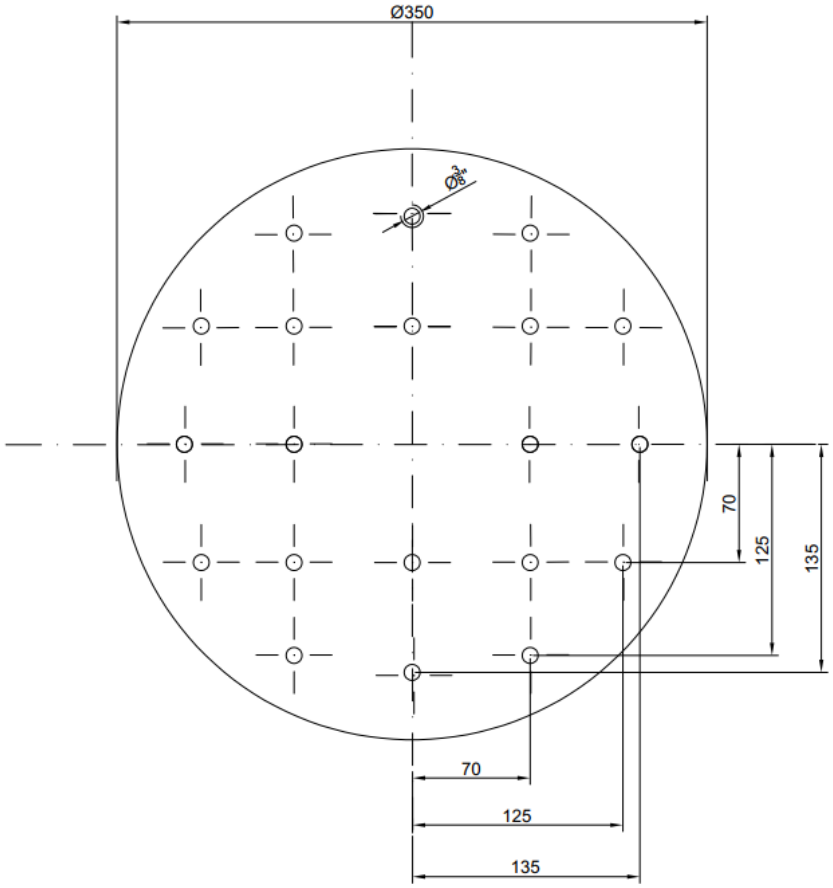


Figura 4.49 – Desenho técnico da base de deposição (vista superior)

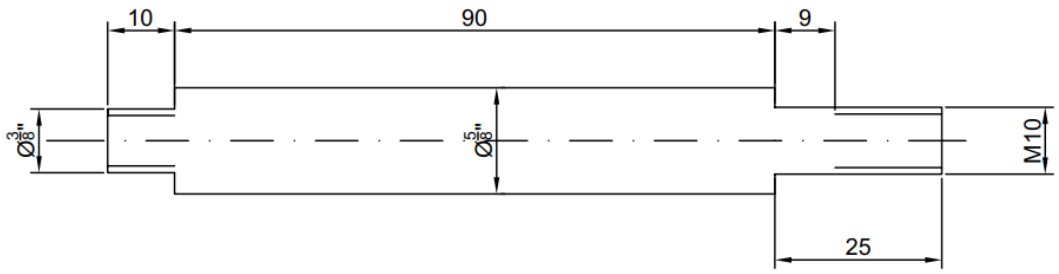


Figura 4.50 – Desenho Técnico dos pinos de suporte da base (4 unidades)

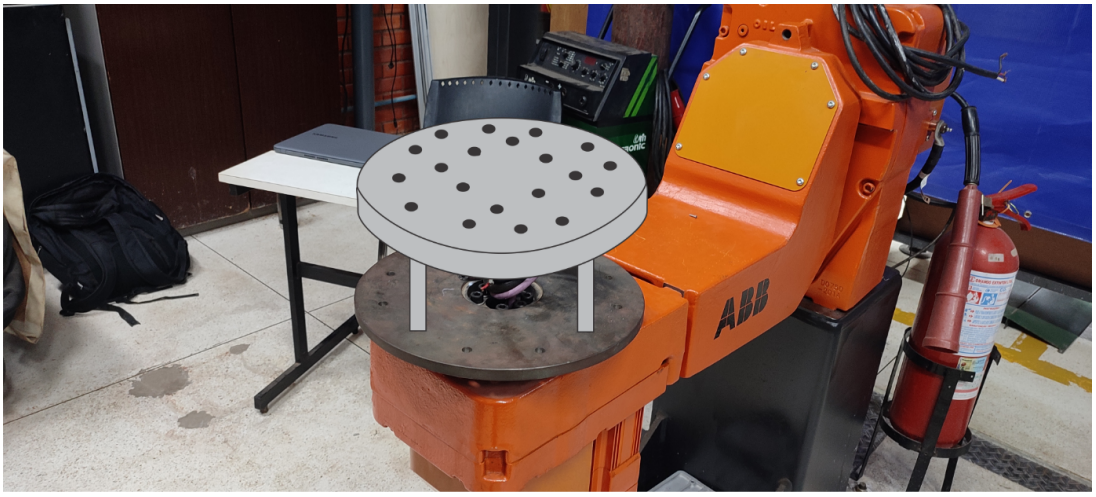


Figura 4.51 – Montagem da base na mesa posicionadora

4.5 Montagem para o ensaio

Motivos alheios impossibilitaram a manufatura da base de deposição (seção 4.4). Então, para validar as trajetórias obtidas pelo software “KarelToRapid”, acopla-se um lápis no local onde é montado o tubo que se passa o arame, improvisa-se uma base de papelão para fixar uma cartolina e modifica-se o código do programa para desabilitar o incremento das camadas (crescimento em z da peça), dessa maneira obtêm-se as curvas de nível das trajetórias desenhadas pelo robô na cartolina. A figura 4.24 apresenta a montagem do TCP e a Figura 4.52 a montagem da base.

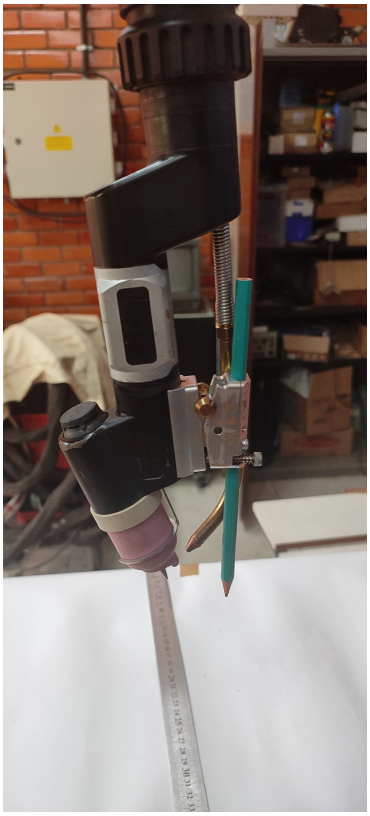


Figura 4.52 – Montagem do TCP

Figura 4.53 – Montagem da base

A Transformação de coordenadas do TCP (tocha GTAW) não coincidem com a transformação de coordenadas da ponta do lápis. Como descrito na subseção 3.1.2.1, com o FlexPendant é possível definir a transformação de coordenadas do punho para o TCP. Define-se algumas posições referentes ao punho do manipulador movendo a ponta do lápis manualmente para uma mesma coordenada no espaço com orientações diferentes, obtêm-se as coordenadas e orientações do punho, o controlador fará automaticamente os cálculos e retornará a transformação correspondente do novo TCP. Assim todas as coordenadas de movimentação no programa Rapid são aplicadas à ponta do lápis.

4.6 Ensaio de movimentação

Utilizaram-se geometrias já inclusas no software de fatiamento para realização de testes de movimentação.

- **Cubo**

Na Figura 4.54 apresenta a trajetória desejada para a manufatura do cubo, obtida por meio do software de fatiamento.

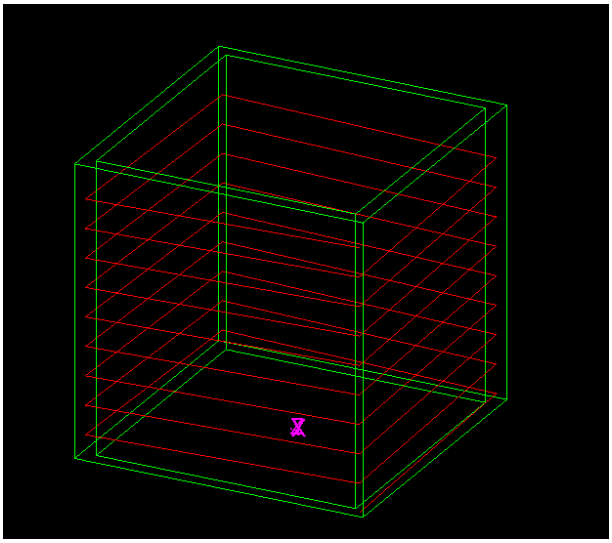


Figura 4.54 – Trajetórias do cubo obtidas pelo software de fatiamento

Fonte: (ANDRADE, 2013)

Como é desconsiderado o crescimento em z, as curvas de nível esperadas são quadrados sobrepostos, ou seja, em cada camada o TCP se move para as mesmas posições.

Na Figura 4.55 apresenta-se as curvas para estratégia “sem orientação” para o cubo. Nessa estratégia o TCP mantém-se limitado ao eixo y (raio em coordenadas cilíndricas), realiza as reorientações necessárias do TCP (quatérnions) enquanto a mesa gira sincronizada, obtendo uma trajetória relativa linear.

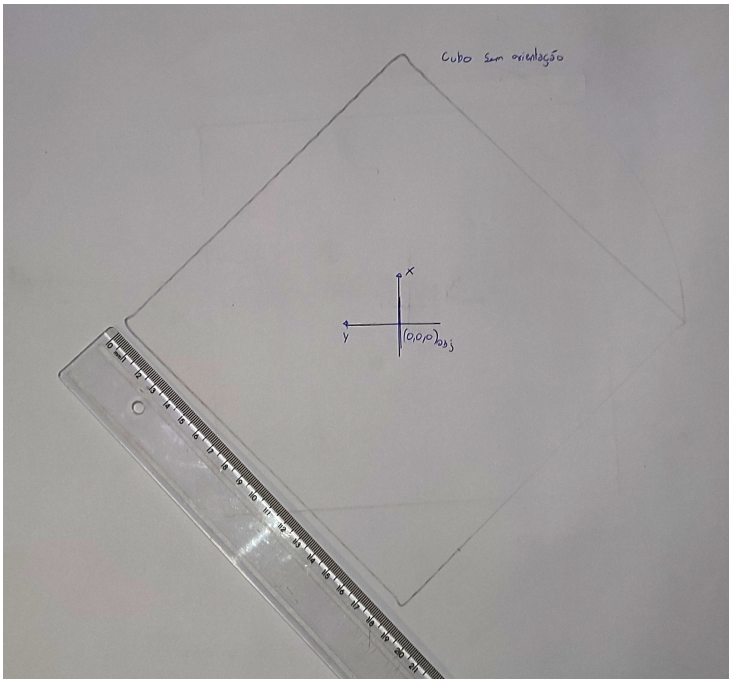


Figura 4.55 – TCurvas do cubo na estratégia “sem orientação”

Na Figura 4.58 apresenta-se as curvas para estratégia “com orientação” para o cubo. Nessa estratégia não é realizada orientação da mesa (giro da mesa em torno do eixo x) pois toda a superfície lateral do cubo já é alinhada à gravidade, então realiza-se apenas o movimento linear até o ponto de quina e o movimento circular para o próximo ponto recalculado, sempre mantendo a direção do movimento paralela ao eixo x.

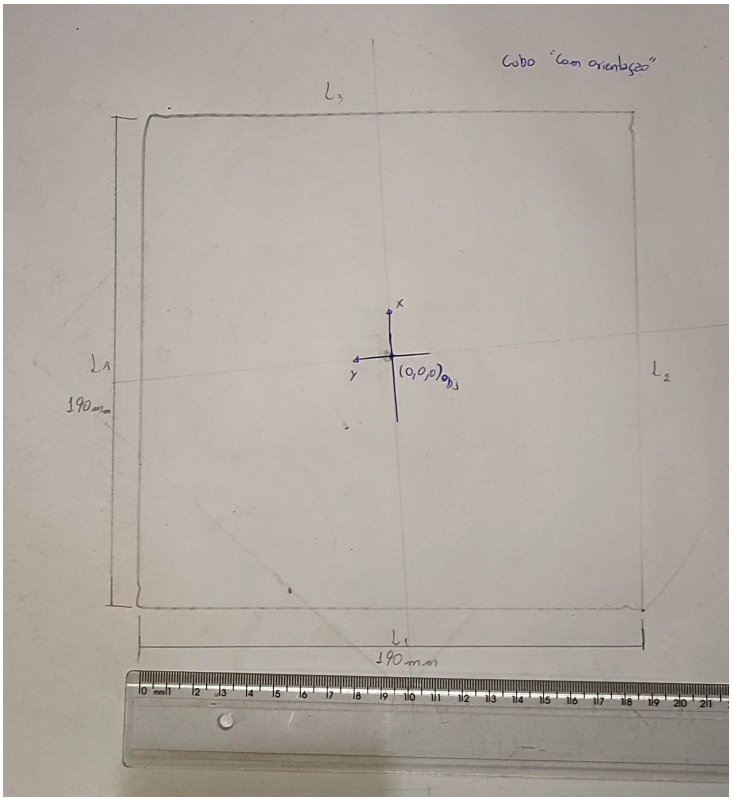


Figura 4.56 – Curvas do cubo na estratégia “com orientação”

- **Piramide**

Na Figura 4.57 apresenta a trajetória desejada para a manufatura do tronco da pirâmide, obtida por meio do software de fatiamento.

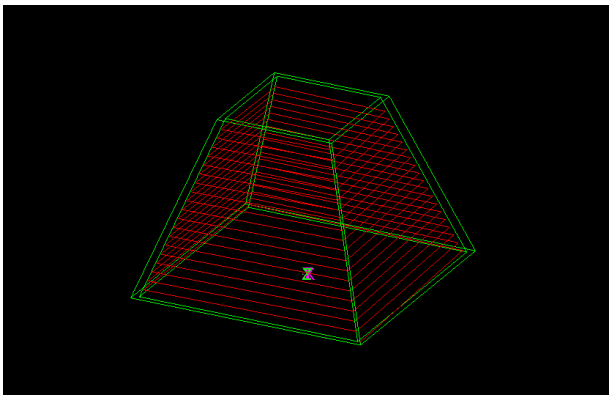


Figura 4.57 – Trajetórias da pirâmide obtidas pelo software de fatiamento

Fonte: (ANDRADE, 2013)

As curvas de níveis esperadas são semelhantes às do cubo, porém os lados dos quadrados vão diminuindo a cada traço, formando assim uma espiral.

Na **??** apresenta-se as curvas utilizando a estratégia “sem orientação” para a pirâmide.

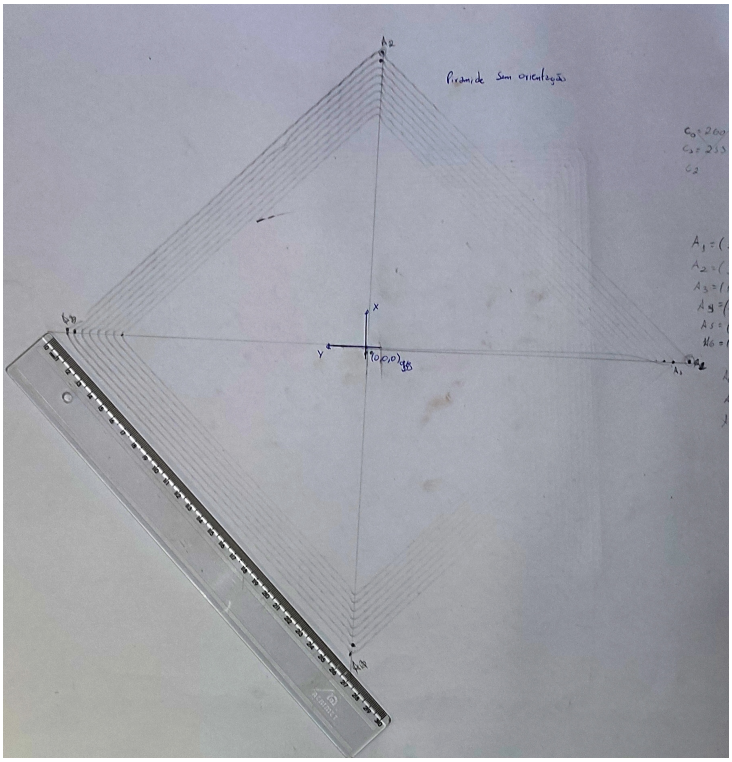


Figura 4.58 – Curvas da pirâmide na estratégia “sem orientação”

Com a estratégia “com orientação” é então observado a orientação da mesa. Mantém-se a direção do movimento sempre paralela ao eixo x, após a primeira camada a mesa orienta de acordo com a inclinação necessária para que a tangente à superfície lateral em construção, no sentido do crescimento camada a camada, permaneça alinhada à direção da gravidade. A Figura 4.59 apresenta as curvas da pirâmide na estratégia “com orientação”.

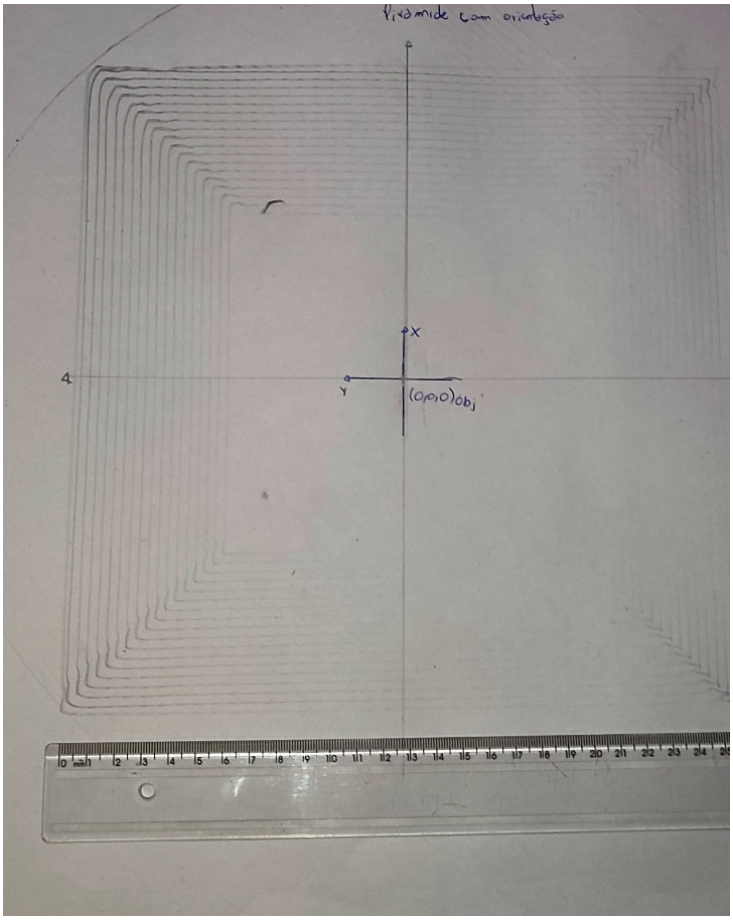


Figura 4.59 – Curvas da pirâmide na estratégia “com orientação”

- **Geometria mista**

Realizaram-se testes utilizando o modelo da Figura 4.32 para visualizar o resultado utilizando geometrias mistas, com trajetórias circulares e lineares. A figura 4.33 apresenta as curvas obtidas pela movimentação do robô.

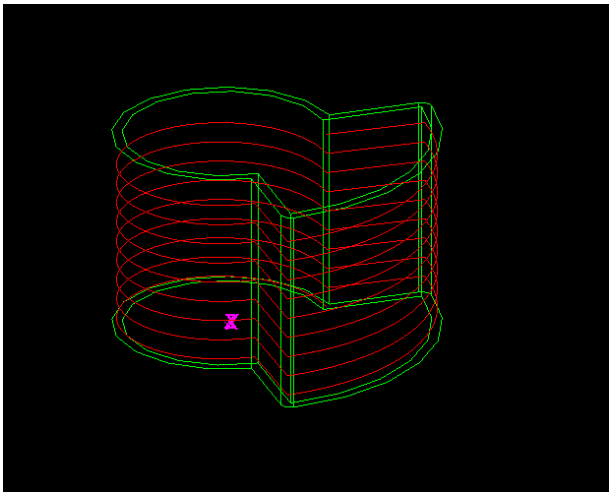


Figura 4.60 – Trajetórias de uma geometria mista

Fonte: (ANDRADE, 2013)

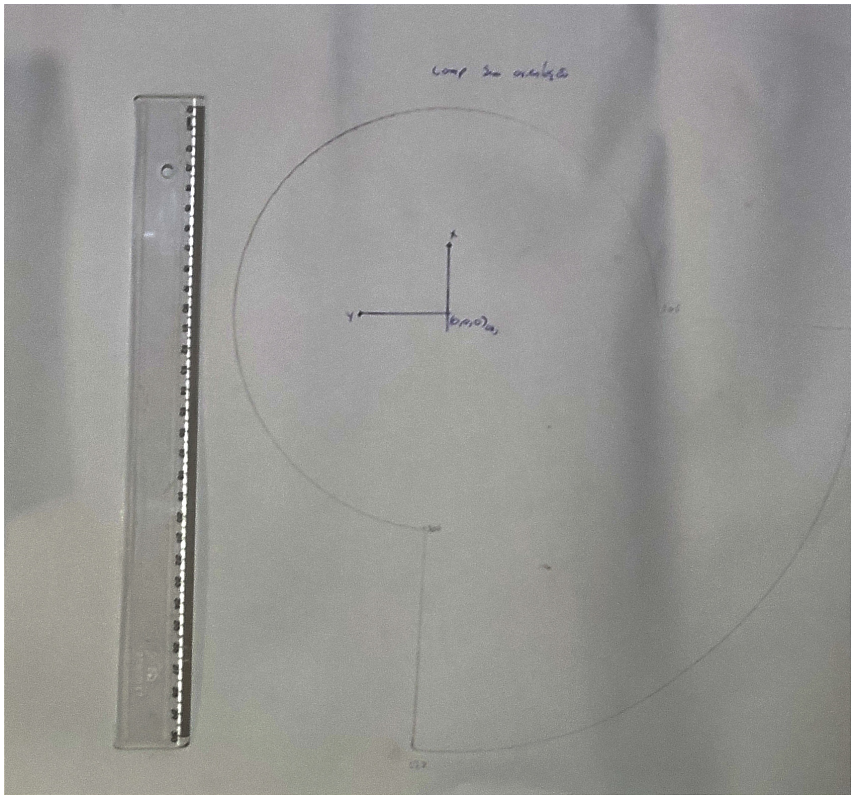


Figura 4.61 – Curvas da pirâmide na estratégia “com orientação”

4.7 Análise das trajetórias

- **Cubo**

Utilizando uma régua com resolução de 0,5 mm, medem-se as coordenadas polares (r,θ) das 4 arestas do quadrado desenhado e comparam-se com as coordenadas correspon-

dentes presente no código em Rapid executado, onde o raio é da coordenada cartesiana Y do TCP e θ o ângulo do prato da mesa posicionadora. Obtém-se a Tabela 4.3:

Tabela 4.2 – SMedidas do cubo na estratégia "sem orientação"

Coordenada medida (±0,5 mm)	**Coordenada no código Rapid** (mm)
(134 , 0º)	(-134.3503 , -0º)
(134 , 90º)	(-134.3503 , -90º)
(134 , 180º)	(-134.3503 , -180º)
(134 , 270º)	(-134.3503 , -270º)

- **Pirâmide**

Semelhante à medida do cubo, medem-se as coordenadas polares das arestas, porém as arestas diminuem a cada volta completa, então medem-se as arestas em 2 voltas do prato (0º a 720º), ou seja, medem-se 9 arestas no contorno.

Tabela 4.3 – Medidas do cubo na estratégia "com orientação"

Coordenada medida (±0,5 mm)	**Coordenada no código Rapid** (mm)
(194, 0º)	(195.0, 0º)
(184, 90º)	(185.0, 90º)
(184, 180º)	(185.0, 180º)
(184, 270º)	(185.0, 270º)
(184, 360º)	(185.0, 360º)
(179, 450º)	(180.0, 450º)
(179, 540º)	(180.0, 540º)
(179, 630º)	(180.0, 630º)
(179, 720º)	(180.0, 720º)

Nos modelos do cubo e da pirâmide constaram-se pequenas distorções nas arestas das trajetórias, aplicando ambas as estratégias. Isso é devido às estratégias adotadas para minimizar os efeitos da mudança brusca na direção do movimento (previsto em 4.2.1.2). Porém ao observar a movimentação do robô nas trajetórias constata-se que essa distorção também pode ser devido à elasticidade do lápis utilizado. As distorções podem ser visualizadas nas Figura 4.62 para o cubo e Figura 4.63 para a pirâmide.

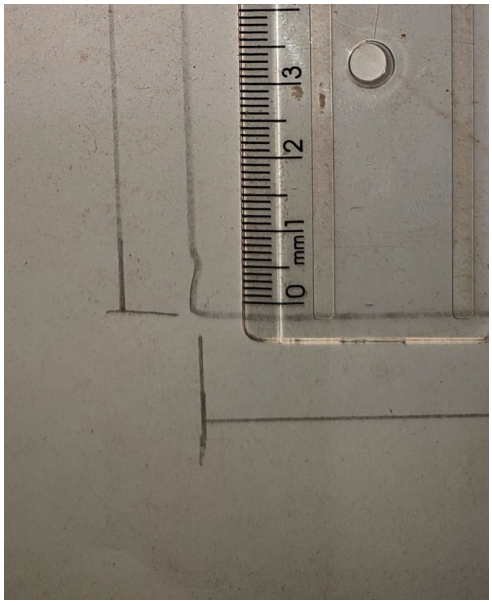


Figura 4.62 – Distorções nas arestas da cubo

Figura 4.63 – Distorções nas arestas da pirâmide

5 Conclusões

No desenvolvimento desse trabalho, foi possível aplicar vários dos conhecimentos adquiridos durante todo o curso. Pois envolveu conhecimentos em álgebra linear, modelagem cinemática robótica, algoritmos de programação, Modelagem 3D, CAD, além de conhecimentos adquiridos das plataformas desenvolvidas pela ABB.

Esse projeto apresenta-se eficiente em cumprir o objetivo de gerar as trajetórias respeitando as limitações do robô IRB 2600. Viu-se que é possível a implementação da mesa posicionadora de forma a reorientar a superfície de deposição e sincronizar essa reorientação com o movimento da tocha, e a adaptação do software de fatiamento para gerar as trajetórias para uma célula robótica diferente da proposta. Logo, o trabalho mostra-se promissor enquanto ponte para o desenvolvimento de futuros projetos de manufatura aditiva com o processo GTAW.

Notou-se a necessidade de um estudo maior para determinação de parâmetros da fonte de soldagem pertinentes ao processo GTAW para a deposição metálica, visto que há trabalhos semelhantes desenvolvidos no mesmo laboratório, porém em GMAW.

5.1 Trabalhos Futuros

Possíveis pontos a serem desenvolvidos em trabalhos futuros:

- Determinação de Parâmetros de soldagem GTAW, além da captura e análise dos dados;
- Manufatura da base de deposição (desenho técnico do projeto na seção 3.4.1);
- Testes com deposição metálica;
- Análise dos erros de trajetória e da deposição em metal;
- Adaptação do software para conversão de código G, além do Karel;

Referências

ANDRADE, R. C. de. **Desenvolvimento de software de fatiamento de sólidos tipo casca e geração de trajetórias para fabricação de peças por deposição de metal em camadas sucessivas utilizando o processo GMAW.** Abr. 2013. Faculdade de Tecnologia, Universidade de Brasília, Brasília. Citado nas pp. 14, 29–34, 43–46, 62, 69, 71, 74.

ASEA BROWN BOVERI. **Introduction to RAPID**: 3HAC029364-001-rev-en. 2010a. P. 1264. Citado nas pp. 22, 25, 26.

ASEA BROWN BOVERI. **Operating manual - Trouble shooting, IRC5**: 3HAC020738-001$_r ev K_e n$. 2010b. P. 466. Disponível em: https://library.e.abb.com/public/e6617595547fc6c2c1257cc5004451bd/Operating%20manual_Trouble%20shooting_3HAC020738-001_revK_en.pdf. Citado nas pp. 38–42.

BIASI, S. C. de; GATTASS, M. **Utilização de quatérnios para representação de rotações em 3D**. Disponível em: http://webserver2.tecgraf.puc-rio.br/~mgattass/fcg/Quaternios.pdf – acesso em 28 Set. 2022. Citado na p. 21.

BLOG IMPACTA. **O que é robótica industrial: entenda tudo sobre a área.** Disponível em: impacta.com.br/blog/o-que-robotica-industrial-entenda-tudo-sobre/ – acesso em 25 set. 2022. 2018. Citado na p. 13.

EFUNDA , INC. **Rapid Prototyping: An Overview**. Disponível em: http://www.efunda.com/processes/rapid_prototyping/intro.cfm – acesso em 23 nov. 2019. 2008. Citado na p. 16.

ENGIPRINTERS. **Os tipos de Tecnologia de Impressão 3D**. Disponível em: https://engiprinters.com.br/os-tipos-de-tecnologia-de-impressao-3d/ – acesso em 05 Out. 2022. 2019. Citado na p. 13.

FRONIUS. **Fonte de solda TIG - MagicWave 4000 / 5000 Job**: 42,0426,0025,PB. 2010. P. 168. Disponível em: https://www.fronius.com/~/downloads/Perfect%20Welding/Operating%20Instructions/42%2C0426%2C0025%2CPB.pdf – acesso em 28 Set. 2022. Citado na p. 36.

INFOSOLDA. **Processo TIG**. Disponível em: https://infosolda.com.br/processo-tig/ – acesso em 25 Set. 2022. 2022. Citado na p. 13.

MARQUES, P. V.; MODENESI, P. J.; QUEIROZ, A. **Soldagem: fundamentos e tecnologia**. 3. ed.: Editora UFMG., 2009. Citado nas pp. 16, 17.

MIRANDA, T. B. D. **Integração do processo de manufatura aditiva por soldagem TIG com adição de arame frio em uma célula robótica de 8 graus de mobilidade.** Abr. 2018. Faculdade de Tecnologia, Universidade de Brasília, Brasília. Citado nas pp. 63, 64.

SANTOS, V. M. F. **Robótica Industrial.** 2004. Universidade de Aveiro. Citado nas pp. 18–21.

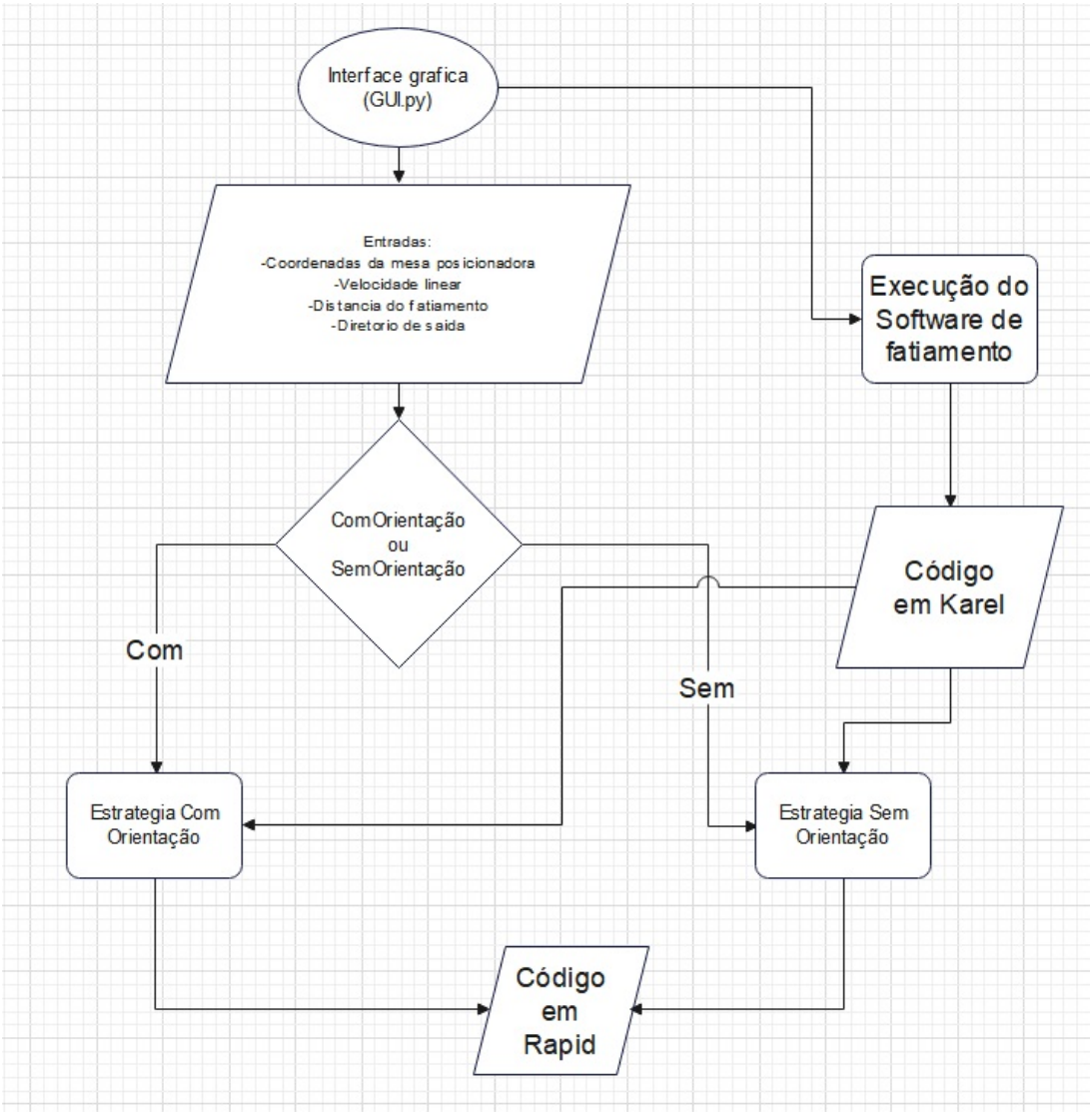
SILVA, R. M. da. **Introdução à dinâmica e ao controle de manipuladores robóticos**: Apostila compilada para uso dos alunos do curso de engenharia de controle e automação da PUCRS. Disponível em: https://docplayer.com.br/18224530-Introducao-a-dinamica-e-ao-controle-de-manipuladores-roboticos.html – acesso em 28 Set. 2022. Citado nas pp. 17, 18.

VASCONCELLOS, A. C. de; FIGUEIREDO, G. R. S. **Concepção, projeto, montagem e controle de uma mesa de soldagem 3D para peças com simetria cilíndrica.** Dez. 2016. Faculdade de Tecnologia, Universidade de Brasília, Brasília. Citado na p. 13.

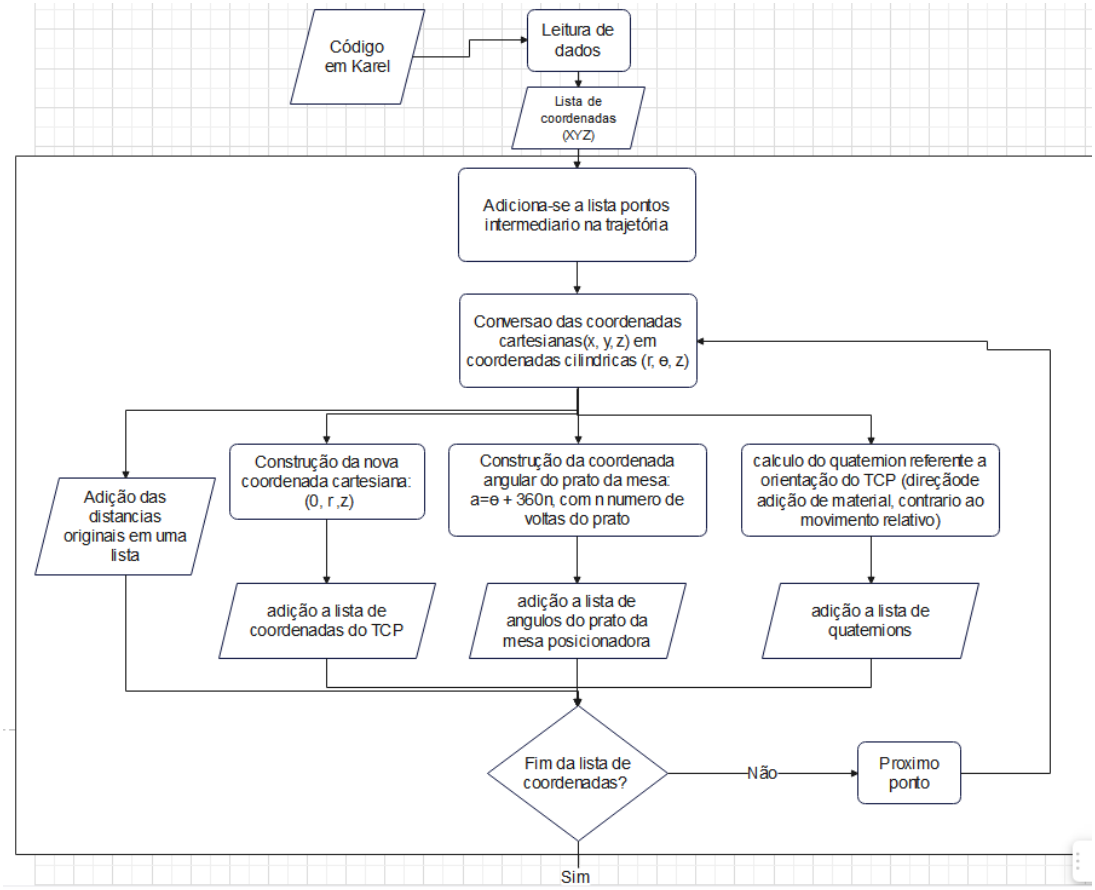
Apêndices

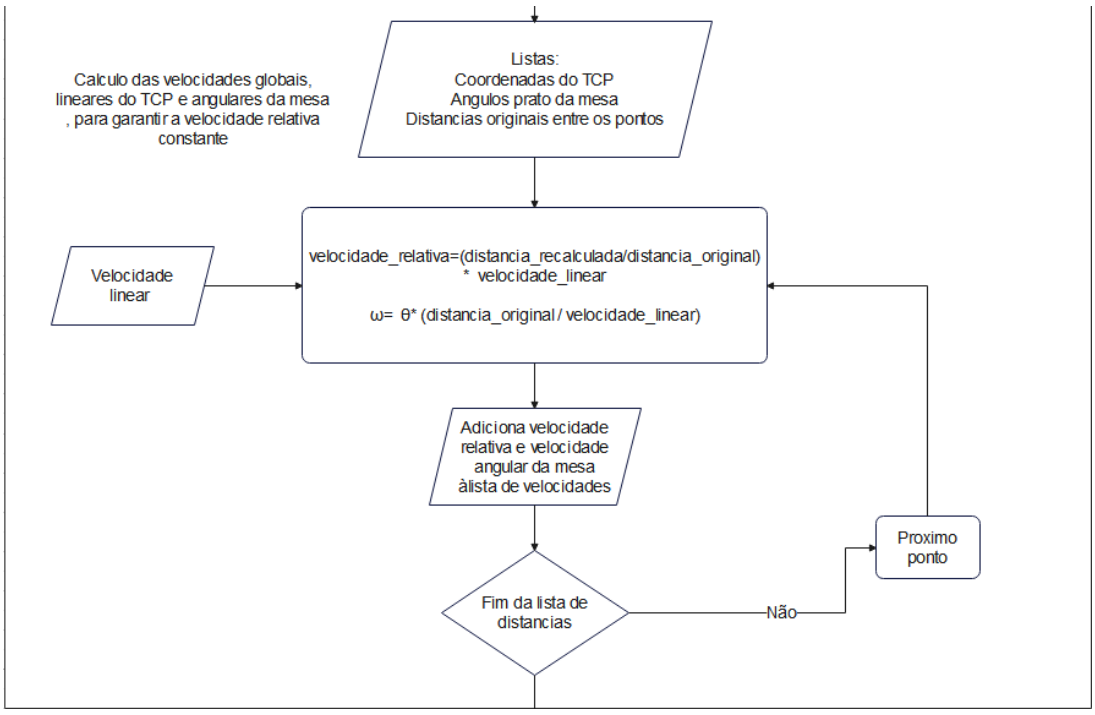
Apêndice A – Fluxograma do algoritmo do software KarelToRapid

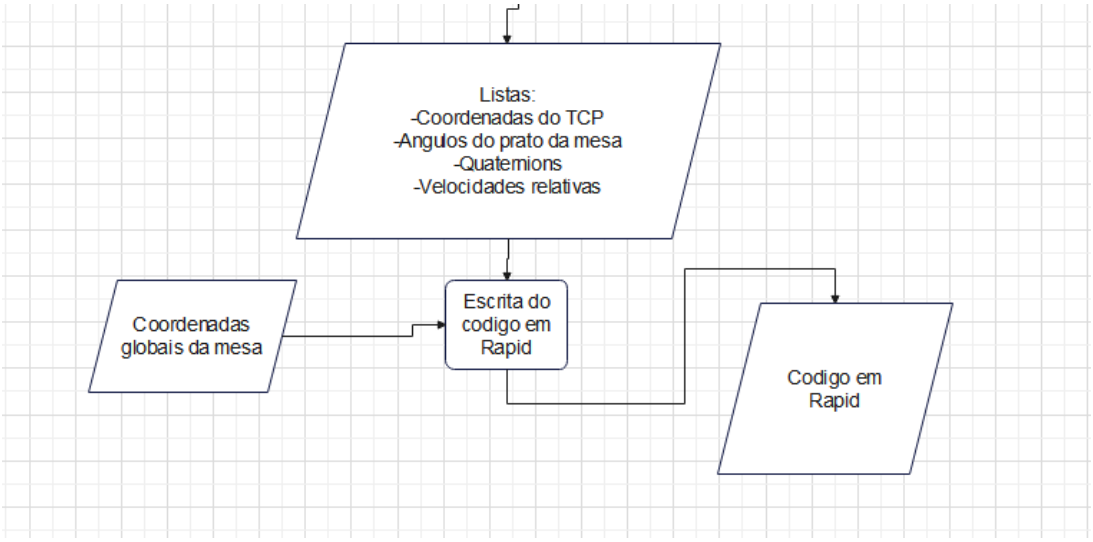
A.1 Fluxograma Geral:



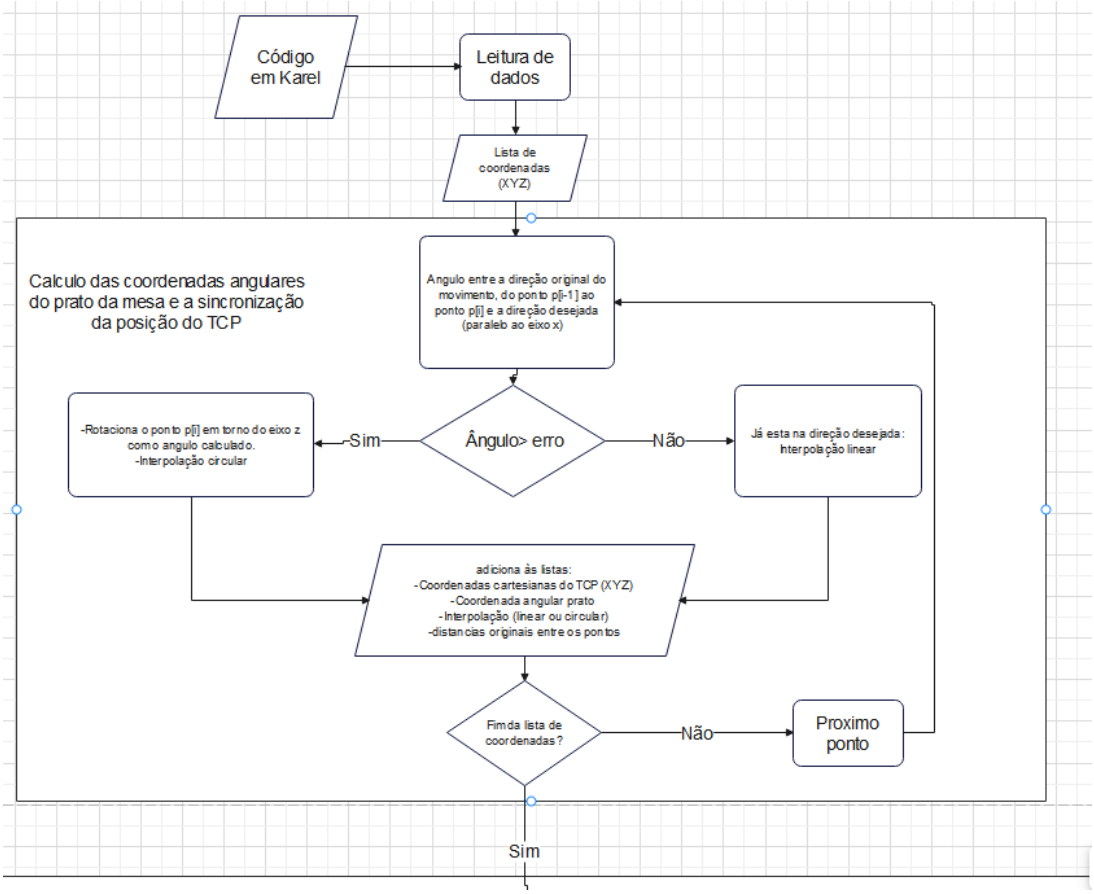
A.2 Fluxograma da Estratégia “Sem Orientação”

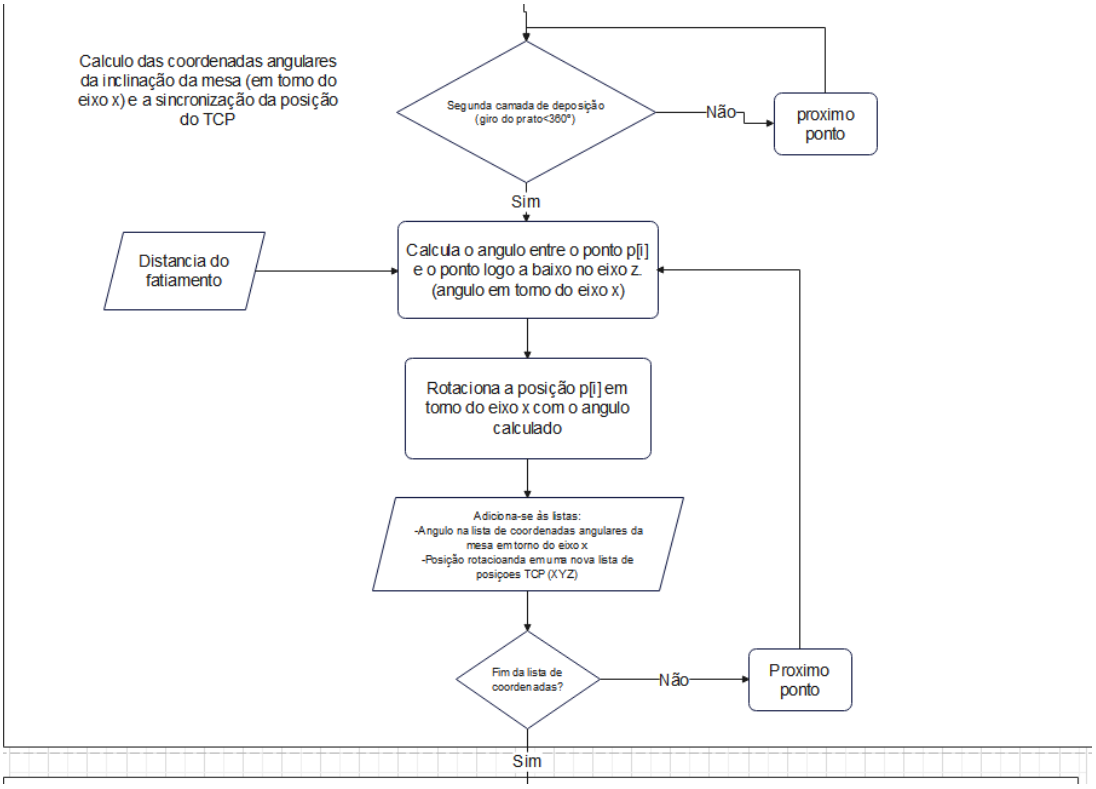


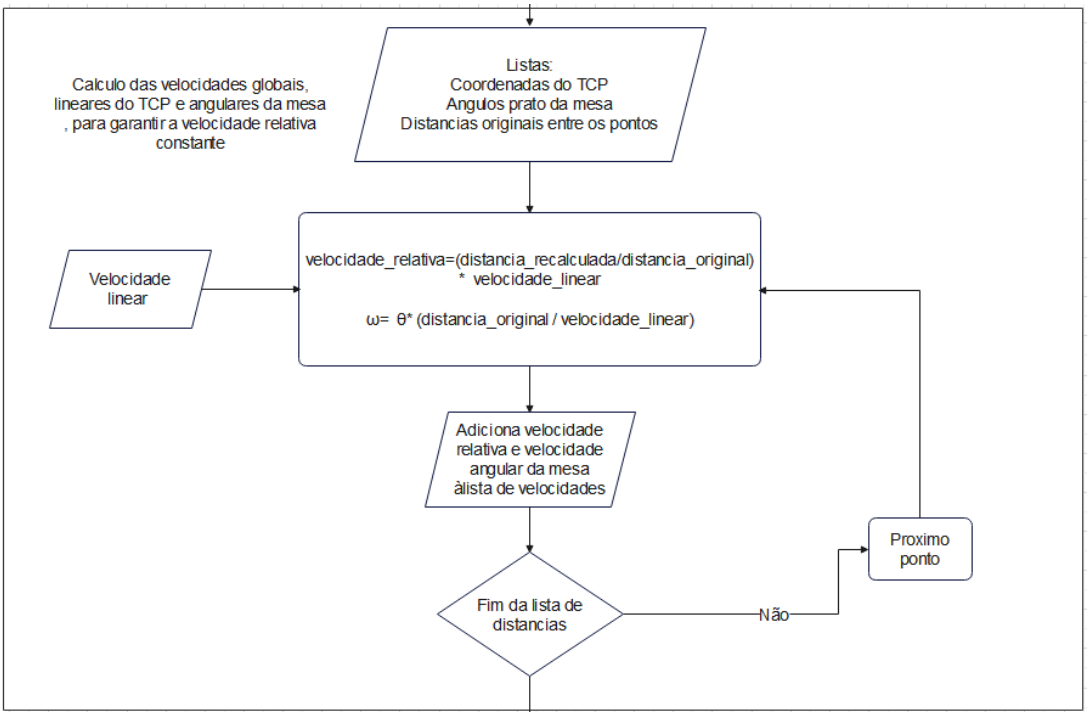


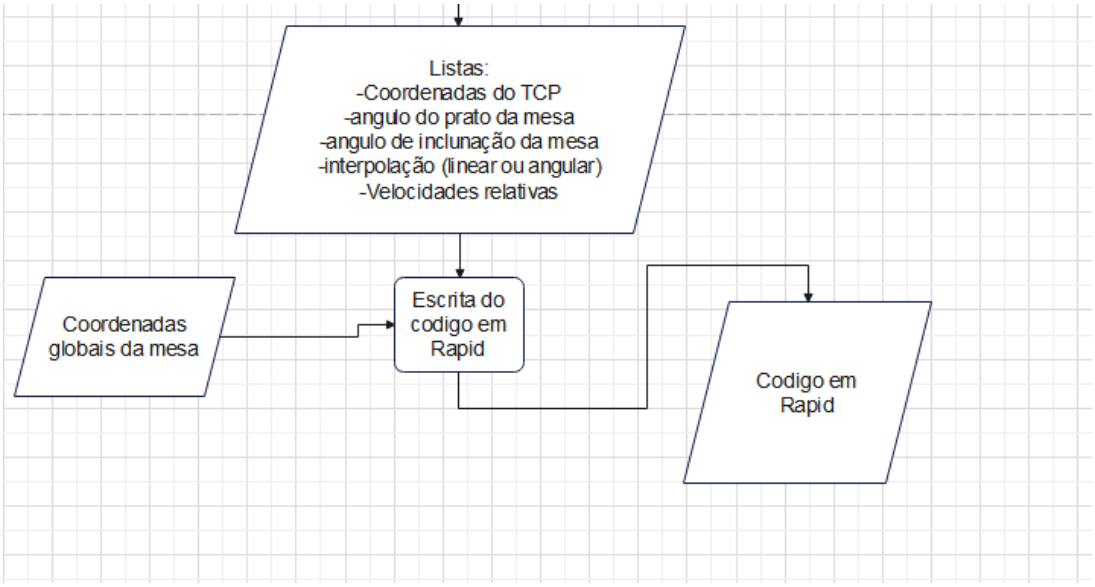


A.3 Fluxograma da Estratégia “Com Orientação”




Calculo das coordenadas angulares da inclinação da mesa (em torno do eixo x) e a sincronização da posição do TCP
Segunda camada de deposição (giro do prato<360°)
Não
proximo ponto
Sim
Distancia do fatiamento
Calcula o angulo entre o ponto p[i] e o ponto logo a baixo no eixo z. (angulo em torno do eixo x)
Rotaciona a posição p[i] em torno do eixo x com o angulo calculado
Adiciona-se às listas:
-Angulo na lista de coordenadas angulares da mesa em torno do eixo x
-Posição rotacioanda em uma nova lista de posiçoes TCP (XYZ)
Fim da lista de coordenadas?
Não
Proximo ponto
Sim


Calculo das velocidades globais, lineares do TCP e angulares da mesa , para garantir a velocidade relativa constante
Listas:
Coordenadas do TCP
Angulos prato da mesa
Distancias originais entre os pontos
Velocidade linear
velocidade_relativa=(distancia_recalculada/distancia_original) * velocidade_linear
ω= θ* (distancia_original / velocidade_linear)
Adiciona velocidade relativa e velocidade angular da mesa àlista de velocidades
Fim da lista de distancias
Não
Proximo ponto


Listas:
-Coordenadas do TCP
-angulo do prato da mesa
-angulo de inclunação da mesa
-interpolação (linear ou angular)
-Velocidades relativas
Coordenadas globais da mesa
Escrita do codigo em Rapid
Codigo em Rapid

Apêndice B – Código em Python da Interface Gráfica (GUI.py).

Código B.1 – Código de Python

```
#Bruno Sales de Matos 13/0070599
#Trabalho de graduacao
#Software de conversao coordenadas de Karel para Rapid
#
#
#------Modulo principal e de Interface Grafica
#
from tkinter import *
import tkinter.filedialog as fdlg
import os

import com_orientacao as co
import sem_orientacao as so



# criação do script para inicializaçao do software de fatiamento
if not os.path.isfile(os.getcwd()+'/Fatiamento.bat'):
    fat=open(os.getcwd()+'/Fatiamento.bat','w')
    fat.write('cd FatiamentoCasca\nstart
       softwaredeFatiamento.exe\nexit')
    fat.close()


#----Execução do modulo com orientação
def executar1():
  vd=v.get()
  dd=d.get()
  xd=x.get()
  yd=y.get()
  zd=z.get()
  directory=diretor.get()
  done.place_forget()
  error.place_forget()
  if((vd !='') and (dd !='') and (xd !='') and (yd !='') and (zd
     !='') and (directory !='')):
    co.work(v.get(),d.get(),x.get(),y.get(),z.get(),diretor.get())

    done.place(x=140,y=400,width=240,height=20)
  else:
    error.place(x=140,y=400,width=240,height=20)

```

```
41 #----Execução do modulo sem orientação
42 def executar2():
43   vd=v.get()
44   dd=d.get()
45   xd=x.get()
46   yd=y.get()
47   zd=z.get()
48   directory=diretor.get()
49   done.place_forget()
50   error.place_forget()
51   if((vd !='') and (xd !='') and (yd !='') and (zd !='') and
        (directory !='')):
52     so.work(v.get(),x.get(),y.get(),z.get(),diretor.get())
53
54     done.place(x=140,y=400,width=240,height=20)
55   else:
56     error.place(x=140,y=400,width=240,height=20)
57
58 #----- Definição do diretorio de saida
59 def diretorio():
60   opcoes = {}                  # as opções são definidas em um
        dicionário
61   #opcoes['defaultextension'] = '.txt'
62   #opcoes['filetypes'] = [('Todos arquivos', '.*'), ('arquivos
        texto', '.txt')]
63   opcoes['initialdir'] = 'C:/'    # será o diretório atual
64   #opcoes['initialfile'] = '' #apresenta todos os arquivos no
        diretorio
65
66   opcoes['title'] = 'Salvar em:'
67
68   dire= fdlg.askdirectory(**opcoes)
69   if dire=='':
70     dire=diretor.get()
71
72   diretor.delete(0,"end")
73   diretor.insert(0,dire)
74
75 # execução do script de inicio do software de fatiamento
76 def fatiamento():
77   os.system('start Fatiamento.bat')
78
79
80 #------------------ Inicio da execução da interface grafica
81 app=Tk()
82 app.title("KarelToRapid")
83 app.geometry("500x450")
84 app.configure(background="#A9A9A9")
85
86
87 # texto para confirmação de execução concluida ou erro
```

```
88  error=Label(app,text="Erro - preencha os campos corretamente",
        foreground="#FF0000", background="#A9A9A9")
89  done=Label(app,text="Processo Concluido", foreground="#008000",
        background="#A9A9A9" )
90
91
92  Label(app,text="Velocidade linear de movimentacao
        (mm/s):",foreground="#000000",anchor=W).place(x=10,y=10,width=240,height=
93  v=Entry(app)
94  v.insert(0,'10')
95  v.place(x=255,y=10,width=100,height=20)
96
97  Label(app,text="Deslocamento do fatiamento
        (mm):",foreground="#000000",anchor=W).place(x=10,y=40,width=240,height=20
98  d=Entry(app)
99  d.insert(0,'10')
100 d.place(x=255,y=40,width=100,height=20)
101
102 # Entrada das coordenadas Offset
103 Label(app,text="Coodernadas de offset
        (mm):",foreground="#000000",anchor=W).place(x=10,y=70,width=240,height=20
104 Label(app,text="X:",foreground="#000000",anchor=W).place(x=10,y=100,width=24
105 Label(app,text="Y:",foreground="#000000",anchor=W).place(x=10,y=130,width=24
106 Label(app,text="Z:",foreground="#000000",anchor=W).place(x=10,y=160,width=24
107
108 x=Entry(app)
109 x.insert(0,'1399.3')
110 x.place(x=255,y=100,width=100,height=20)
111
112 y=Entry(app)
113 y.insert(0,'12.7')
114 y.place(x=255,y=130,width=100,height=20)
115
116 z=Entry(app)
117 z.insert(0,'588')
118 z.place(x=255,y=160,width=100,height=20)
119
120 # Add imagem
121 photo = PhotoImage (file = "ABB.png")
122 Label(app,image=photo,anchor=E).place(x=370,y=10,width=124,height=161)
123
124
125 # Botao de diretorio
126 dire='C:/'
127
128 Button(app,text="Salvar
        em:",command=diretorio).place(x=20,y=200,width=200,height=40)
129 diretor=Entry(app)
130 diretor.place(x=250,y=200,width=200,height=20)
131 diretor.insert(0,dire)
132
133 # Botao de inicio com mesa
```

```
134 Button(app,text="Iniciar com mesa
    orientadora",command=executar1).place(x=20,y=250,width=200,height=40)
135 # botao de inicio sem mesa
136 Button(app,text="Iniciar sem mesa
    orientadora",command=executar2).place(x=250,y=250,width=200,height=40)
137 # botao de inicio do software de fatiamento
138 Button(app,text="Executar Software de
    Fatiamento",command=fatiamento).place(x=140,y=300,width=200,height=40)
139
140 app.mainloop()
```

Apêndice C – Código em Python da implementação sem mesa orientadora – (sem_orientacao.py).

Código C.1 – Código de Python

```
#Bruno Sales de Matos 13/0070599
#Trabalho de graduacao
#Software de conversao coordenadas de Karel para Rapid
#considerando mesa orientadora
#
#
#-----------Modulo sem orientação da mesa------------

import numpy as np
import calc
import os.path
import shutil

def work(v,x_offset,y_offset,z_offset,diretorio):


  v=float(v)
  x_offset=float(x_offset)
  y_offset=float(y_offset)
  z_offset=float(z_offset)

  ####################------------------------ Leitura do arquivo
     em Karel----------------------------#######################
  arquivo=open('FatiamentoCasca/Track01#.kl','r')

    #Inicializacao das variaveis
  linhaf=[]
  diff=np.array([0,0,0])
  pontox=[[]]
  pontoc=[[]]
  pi=np.pi

    #Loop para ler e tratar as coordenadas e adiconar em uma lista
       de arrays
  linha=arquivo.readline()
  while(linha!=''):
      linha=linha.split('(')
      linha[1]=linha[1].split(',')
      linhaf = np.array([  float(linha[1][0]), float(linha[1][1]),
         float(linha[1][2])  ])
```

```
38      pontox.append(linhaf)
39      linha=arquivo.readline()
40
41  arquivo.close()
42
43  ######################----------------Calculo dos pontos
       adicionando a mesa
       posicionadora--------------------################
44
45  ponto2=[np.array([0,0,0])]
46  ponto3=[np.array([0,0,0])]
47
48  dist=[0.0,0.0]      #Array com as distancias entre os
       pontos(utilizado para calculo das velocidades relativas)
49  a1=[0.0,0.0]        #angulo da mesa em z (graus)
50  a2=[0.0,0.0]      #angulo da mesa em x (graus)
51  diff=np.array([0.0,0.0,0.0])
52
53  q=[np.array([0,0,1,0])] #direcao de movientacao da ferramenta em
       quaternions
54
55  fine=4   # distancia maxima entre os pontos para interpolacação
       em mm
56
57    #--------------Addiciona pontos intermediarios para uma melhor
         interpolacao
58  ponto2.append(pontox[1])
59  i=2
60  while i<len(pontox):
61
62    diff=pontox[i]-pontox[i-1]
63
64    if(calc.modulo(diff)>fine):
65      n=calc.modulo(diff)/fine
66      n=int(n)+1
67      j=1
68      while(j<=n):
69        t=float(j/n)
70        p_int=pontox[i-1]+t*(diff)
71        ponto2.append(p_int)
72        j+=1
73    else:
74      ponto2.append(pontox[i])
75    i+=1
76
77
78  #--------armazena as distancias originais entre os pontos para
       fim de calculo posterior das velocidades
79  i=2
80  while i<len(ponto2):
81    diff=ponto2[i]-ponto2[i-1]
82    dist.append(calc.modulo(diff))
```

```
83      i+=1
84
85
86    #---------Converte as coordenadas cartesiana em cilindricas e
         determina a posição angular da mesa
87    vc=calc.xyztocilinder(ponto2[1])
88    a_offset=vc[1]
89    a_offset*=180/pi
90
91    ponto3.append(np.array([0,-vc[0],vc[2]]))
92
93    i=2
94    while i<len(ponto2):
95      vc=calc.xyztocilinder(ponto2[i])
96      ponto3.append(np.array([0,-vc[0],vc[2]]))
97      ang=vc[1]
98      ang=ang*180/pi
99      ang-=a_offset
100     ang=calc.angulo(a1[i-1],ang)
101     a1.append(ang)
102     i+=1
103
104
105   #--------------Obtencao das velocidades linear e angulares da
         mesa relativa da ferramenta    superficie------
106
107   vd=[np.array([0.0,0.0])]                          #vd=[vl,va] onde
         vl-velocidade linear e va-velocidade angular
108   vd.append(np.array([float(v),0.0]))
109   i=2
110   while i<len(dist):
111     vel=(calc.modulo(ponto3[i]-ponto3[i-1])) /(dist[i])
          #vel > velocidade linear do TCP
112     vel*=float(v)
113     if vel<0:
114       vel*=(-1)
115     omega= (a1[i]-a1[i-1])*float(v)/dist[i]
116     if omega<0:
117       omega*=(-1)
118     if vel<1:
119       vel=1
120     if omega<1:
121       omega=1
122     vd.append(np.array([vel,omega]))
123     i+=1
124   vd[1]=np.array([50,50])
125   vd.append(vd[1])
126
127   ##------------obtencao da direcao da ferramenta------------
128   i=1
129   while(i<len(ponto3)-1):
130
```

```
131     diff=ponto2[i+1]-ponto2[i]
132     ang=calc.anguloxy(ponto2[i],np.array([0,-1,0]))
133 
134     diff=calc.rotacionarxy(diff,-ang)
135     az=calc.anguloxy(diff,np.array([1,0,0]))
136 
137     q_int=calc.quaternion(az,pi,0)
138     q.append(q_int)
139     i+=1
140   q.append(q[i-1])
141 
142 
143 
144   #----------------------Escrita do arquivo de
        saida----------------------------
145   if os.path.isdir(diretorio+'/proto/'):
146     if not os.path.isfile(diretorio+'/weld.mod'):
147       shutil.copyfile(os.getcwd()+'/proto/weld.mod',diretorio+'/proto/weld.m
148         # else():
149         #
              shutil.copyfile(diretorio+'/proto/weld.mod',os.getcwd()+'/proto/w
150 
151     if not os.path.isfile(diretorio+'/TCP_TTW5500_1.mod'):
152       shutil.copyfile(os.getcwd()+'/proto/TCP_TTW5500_1.mod',diretorio+'/pro
153 
154     if not os.path.isfile(diretorio+'/proto.pgf'):
155       shutil.copyfile(os.getcwd()+'/proto/proto.pgf',diretorio+'/proto/proto
156 
157     if not os.path.isfile(diretorio+'/TCP_TTW5500_1.mod'):
158       shutil.copyfile(os.getcwd()+'/proto/TCP_TTW5500_1.mod',diretorio+'/pro
159 
160   else:
161     shutil.copytree (os.getcwd()+'/proto/', diretorio+'/proto/')
162 
163   saida=open(diretorio+'/proto/MainModule.MOD','w')
164   i=1
165   saida.write('MODULE MainModule\n')
166   saida.write('\tTASK PERS tooldata
        TTW5500_1:=[TRUE,[[32.1099,-0.389434,387.111],[1,0,0,0]],[1,[0,0,100],[
167   saida.write('\tPERS wobjdata wobj1:=[
        FALSE,TRUE,"",[[%.4f,%.4f,%.4f],[1,0,0,0]],[[0,0,0],[1,0,0,0]
        ] ];\n\n'%(x_offset,y_offset,z_offset))
168   saida.write('\tCONST robtarget
        p0:=[[1455.38,0.95,1463.94],[0.707633,-0.000297139,0.70658,0.00010484],
169 
170   #----Escrita das variaveis do tipo posicao
171 
172   # while i<len(ponto3):
173   while i<2000:
174       saida.write('\tCONST robtarget
            p%d:=[[%.4f,%.4f,%.4f],[%.4f,%.4f,%.4f,%.4f],[-1,0,-1,0],[9E+09,%.4
175       i+=1
```

```
176   i=1
177
178   #---Escrita das variaveis do tipo velocidade
179   # while i<len(vd):
180   while i<2000:
181       saida.write('\tVAR speeddata vd%d
              :=[%.2f,500,5000,%.2f];\n'%(i,vd[i][0],vd[i][1]))
182       i+=1
183
184   #---Escrita das rotinas de soldagem pre escritas no arquivo
         weld.MOD------
185   weld=open(diretorio+'/proto/weld.MOD','r')
186   linha=weld.readline()
187   linha=weld.readline()
188   linha=weld.readline()
189   while(linha!='ENDMODULE'):
190     saida.write(linha)
191     linha=weld.readline()
192   weld.close()
193
194
195   #----Escrita da main, ou seja, as chamadas de movimentação
196   saida.write('\n\n\tPROC main()\n')
197   saida.write('\t\tActUnit STN1;\n')
198
199   saida.write('\t\tMovej p1, vd1, z10, TTW5500_1
         '+r'\WObj:=wobj1;'+'\n') # primeiro movimento do tipo junta
         para evitar singularidades)
200
201
202   saida.write('\t\tstand;\n')
203   saida.write('\t\tstart0;\n\n')
204
205   i=2
206   # while i<len(ponto3):
207   while i<2000:
208       saida.write('\t\tMoveL p%d, vd%d, z10, TTW5500_1
              '%(i,i)+r'\WObj:=wobj1;'+'\n')
209
210       i+=1
211
212   saida.write('\t\tstop0;\n\n')
213
214   saida.write('\t\tMoveJ p0, v100, fine, tool0;\n')
215   saida.write('\tENDPROC\n')
216   saida.write('ENDMODULE')
217
218   saida.close()
```

Apêndice D – Código em Python da implementação com mesa orientadora – (com_orientacao.py).

Apêndice E – Código em Python de funções com cálculos recorrentes – (calc.py).

Código E.1 – Código de Python

```
1
2  #Bruno Sales de Matos 13/0070599
3  #Trabalho de graduacao
4  #Software de conversao coordenadas de Karel para Rapid
5  #considerando mesa orientadora
6  #
7
8  #----------Modulo de funcoes para calculo recorrentes----
9
10 import numpy as np
11
12 pi=np.pi
13
14 #Calcula o angulo entro dois vetores no plano xy
15 def anguloxy(v1,v2):
16
17     a1=np.arccos(v1[0]/(np.sqrt(np.square(v1[0])+np.square(v1[1]))))
18     a2=np.arccos(v2[0]/(np.sqrt(np.square(v2[0])+np.square(v2[1]))))
19     q1=quadrante(v1)
20     q2=quadrante(v2)
21     if q1==3 or q1==4:
22         a1=(2*pi)-a1
23     if q2==3 or q2==4:
24         a2=(2*pi)-a2
25
26     return (a1-a2)
27
28 #Rotacao do vetor (v) no plano xy com relacao ao angulo (a)
29 def rotacionarxy(v,a):
30     vr=np.array([0.1,0.1,0.1])
31     #Rotacao do vetor (v) no plano xy com relacao ao angulo (a)
          obtido
32     if a<pi:
33         a=a-(2*pi)
34     vr[0]=(v[0]*(np.cos(a)))-(v[1]*(np.sin(a)))
35     vr[1]=(v[0]*(np.sin(a)))+(v[1]*(np.cos(a)))
36     vr[2]=v[2]
37     return vr
38
39 #Calcula o angulo entre o vetor plano yz e o vetor gravidade
```

```
40 def anguloyz(v):
41 
42     a=np.arcsin(v[1]/(np.sqrt(np.square(v[1])+np.square(v[2]))))
43 
44     if a>pi:
45         a=a-(2*pi)
46     if a<(-pi):
47         a=(2*pi)-a
48 
49 
50     if a>(pi/4):       #limita o angulo em -45   e 45
51         a=pi/4
52     if a<(-pi/4):
53         a=(-pi/4)
54     return (a)
55 
56 #Rotacao do vetor (v) no plano yz com relacao ao angulo (a)
      obtido, z é o offset do sistema de coodernadas com o eixo da
      mesa
57 def rotacionaryz(v,a,z):
58     vr=np.array([0.1,0.1,0.1])
59 
60     if a>pi:
61         a=a-(2*pi)
62 
63     v[2]=v[2]+z
64 
65     vr[0]=v[0]
66     vr[1]=(v[1]*(np.cos(a)))-(v[2]*(np.sin(a)))
67     vr[2]=(v[1]*(np.sin(a)))+(v[2]*(np.cos(a)))
68     vr[2]-=z
69     return vr
70 
71 #retorna o quadrante do angulo
72 def quadrantet(a):
73     while a>(2*pi):
74         a-=(2*pi)
75     if a<(pi/2):
76         q=1
77     elif a<pi:
78         q=2
79     elif a<(3*pi/4):
80         q=3
81     else:
82         q=4
83 
84     return q
85 
86 #retorna o quadrante do vetor
87 def quadrante(v):
88     if v[0]>=0:
89         if v[1]>=0:
```

```
90              q=1
91          else:
92              q=4
93      else:
94          if v[1]>=0:
95              q=2
96          else:
97              q=3
98      return q
99
100 #converte coordenadas cartesianas para cilindicas
101 def xyztocilinder(v):
102     vc=np.array([(np.sqrt(np.square(v[0])+np.square(v[1]))),anguloxy([0,-1,0
103
104     # q=quadrante(v)
105
106     # if q==3 or q==4:
107     #     vc[1]=2*pi-vc[1];
108
109     return vc
110
111 # retorna o tamanho do vetor (funcao módulo)
112 def modulo(v):
113     vt=(np.sqrt(np.square(v[0])+np.square(v[1])+np.square(v[2])))
114     return vt
115
116 # retorna o a diferença entre os angulos [a1] e [ang]
117 def angulo(a1,ang):
118
119     f=a1/360 #calcula quantas voltas a mesa deu
120     f=int(f)
121     # print(f)
122     a=a1-(360*f)
123     angr=ang-360
124     # print(a1)
125     # print(ang)
126     # print(angr)
127     # print(a)
128     # input()
129     dif1=np.sqrt(np.square(ang-a))
130     dif2=np.sqrt(np.square(angr-a))
131
132     # print(dif1)
133     # print(dif2)
134     if(dif1)<=(dif2):
135         if (ang>0 and a<0):
136             return(angr+((f+1)*360))
137         elif(ang<0 and a>0):
138             return(angr-((f-1)*360))
139         else:
140             return(ang+(f*360))
141     else:
```

```
142         if (angr>0 and a<0):
143             return(ang+((f+1)*360))
144         elif(angr<0 and a>0):
145             return(ang-((f-1)*360))
146         else:
147             return(angr+(f*360))
148 
149 
150 #converte roll (Z), pitch (Y), yaw (X) em quaternion
151 def quaternion(z,y,x):
152     q=np.array([0.0,0.0,0.0,0.0])
153     cz = np.cos(z*0.5)
154     sz = np.sin(z*0.5)
155     cy = np.cos(y*0.5)
156     sy = np.sin(y*0.5)
157     cx = np.cos(x*0.5)
158     sx = np.sin(x*0.5)
159 
160     #Quaternion q
161     q[0]=(cz*cy*cz)+(sx*sy*sz)
162     q[1]=(sx*cy*cz)-(cx*sy*sz)
163     q[2]=(cx*sy*cz)+(sx*cy*sz)
164     q[3]=(cx*cy*sz)-(sx*sy*cz)
165 
166     q=np.array([q[0],q[1],q[2],q[3]])
167     return q
```

Apêndice F – Modulo com rotinas de soldagem – (weld.mod).

unbtex-example/codigos/weld.mod

```
MODULE weld
    !Esse modulo atribui as configuraes iniciais antes de se
       iniciar a soldagem

    PROC stand()
        !set atribui 1 a variavel
        !reset atribui 0 a variavel


        reset doFr1RobotReady; ! O robo nao esta pronto para
           soldagem

        set doFr1OpMode2;   !seleciona tipo de soldagem TIG (110)
        set doFr1OpMode1;
        reset doFr1OpMode0;

        reset doFr1GasTest; !Abre (1) e fecha (0) a valvula de gas

        reset doFr1ColdWireDisable;  !(1) para desabilitar a
           alimentacao de arame e (0) para habilitar


        reset doFr1FeedForward;   !Para a alimenta  de arame
           (caso acionado)
        reset doFr1FeedRetract;
        SetAO aoFr1WireSpeedWfi,400;! Define velocidade da
           alimentacao do arame [0-22000 mm/min]


        reset doFr1WeldingSim; ! Simulacao de solda

        reset doFr1DC_AC; !  AC (1) ou DC(0)
        set doFr1DCn_DCp; ! DC+ (1) ou DC- (0)

        set DoFr1PulseDisable; ! (1) para desabilitar funcao
           Pulsar e (0) para habilitar

        reset DoFr1PulseRange2;  ! Seleciona frequencia de
           pulsacao (vide manual)
        reset DoFr1PulseRange1;
        reset DoFr1PulseRange0;

```

```
36          SetAO aoFr1Power,125; !Define a corrente principal do arco
               eletrico (0 a 500A)
37
38          set doFr1DutyCycleDisable; ! (1)para desabilitar o Duty
               cycle e (0) para habilitar
39          SetAO aoFr1DutyCycle, 255; ![0-255) varia o duty cycle de
               10% a 90%
40
41          SetAO aoFr1BaseCurrent,100;! A corrente de soldadagem
               reduzida para 0 a 100% da corrente principal. [0 a 100]
42
43
44
45          set doFr1ErrorReset; ! Reseta a memoria de erros
46          WaitTime 0.5;
47          reset doFr1ErrorReset;
48
49      ENDPROC
50
51      PROC start0()
52          !set atribui 1 a variavel
53          !reset atribui 0 a varaiavel
54
55
56          !reset doFr1WeldingSim; ! Simulacao de solda
57
58          set doFr1RobotReady; ! O robo esta pronto para soldagem
59
60          set doFr1ErrorReset; ! Reseta a memoria de erros
61          WaitTime 0.2;
62          reset doFr1ErrorReset;
63
64          set doFr1ArcOn;
65
66          WaitDI diFr1ArcStable, 1;
67
68          set doFr1FeedForward;   !Alimenta  de arame (caso
               acionado)
69          WaitTime 0.5;
70
71      ENDPROC
72
73      PROC stop0()
74          !set atribui 1 a variavel
75          !reset atribui 0 a varaiavel
76
77          reset doFr1RobotReady; ! O robo nao esta pronto para
               soldagem
78          reset doFr1FeedForward;   !Alimenta  de arame (caso
               acionado)
79          reset doFr1ArcOn;
80
```

```
81
82      ENDPROC
83
84
85
86  ENDMODULE
```