UNIVERSIDADE FEDERAL DA BAHIA
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
ENGC75 – INTRODUÇÃO À ROBÓTICA

Daniel Carneiro
Guilherme Prazeres

BRDC
(Braço Robótico Demarcador de Corte)

Salvador - BA - Brasil
2023

SUMÁRIO

1 Introdução

O BRDC (Braço Robótico Demarcador de Corte) é um sistema composto por um braço robótico (para este projeto, foi adotado o UR5) capaz de realizar demarcações, utilizando um piloto ou caneta, de curvas dispostas em arquivos PDF vetorizados. O sistema deve ser capaz de demarcar diversos materiais e possui como entradas o arquivo PDF e a espessura do material, denominada *offset*, que será a distância que o braço robótico deverá compensar em relação a sua base.

Este projeto tem como objetivo a implementação em ambiente simulado da solução proposta, utilizando scripts em Python para realizar o planejamento e execução de movimentos, utilizando o Gazebo para simulação e o ROS juntamente ao framework MoveIt para operar um modelo do UR5.

Para realização do projeto proposto, o grupo subdividiu o problema em 4 sub-etapas:

- Elaboração de scripts em Python para planejamento e execução dos movimentos.
- Configuração do framework MoveIt para realizar a movimentação do braço robótico.
- Implementação de movimentos simples através do MoveIt utilizando os scripts em Python.
- Realização de simulação de execução através do Gazebo.

As etapas foram realizadas de forma sequencial, de forma que caso o grupo encontrasse algum problema em algum passo, pudesse retornar às etapas anteriores para adaptação aos percalços encontrados.

Ao longo do projeto, algumas modificações e simplificações foram adotadas em relação a solução inicialmente proposta, de forma que pudesse ser obtida uma solução ao menos funcional, dentro do prazo de tempo estabelecido para o trabalho.

Para validação do projeto, foram elaborados 3 datasets que foram executados e validados através do Gazebo.

2 Revisão Bibliográfica

2.1 OpenCV

Para elaboração do script planejador, foi utilizada a biblioteca OpenCV para Python, a fim de realizar o processamento das imagens vetorizadas através do arquivo PDF para identificação dos segmentos e dos pontos que os compõem.

Utilizou-se fundamentalmente a função threshold (para binarizar as cores da imagem para garantir a identificação dos caminhos) e a função findContours. Esta utiliza o algoritmo de detecção de contornos de Canny para identificar as bordas na imagem. Em seguida, aplica um algoritmo de preenchimento para encontrar regiões fechadas e formar contornos e por fim retornar uma lista de contornos, onde cada contorno é uma sequência de pontos representando as coordenadas dos pixels ao longo do contorno.

Estas coordenadas eram então convertidas para pontos tridimensionais, através do script planejador, explicado em detalhes no tópico 3.1.1.

2.2 ROS + RViz

O ROS (Robot Operating System) é um framework de código aberto amplamente utilizado na área de robótica. Ele fornece uma plataforma flexível e poderosa para desenvolver sistemas robóticos complexos, facilitando a integração de diferentes componentes de software, como sensores, atuadores e algoritmos de controle.

O ROS foi projetado para ser modular e distribuído, permitindo a comunicação e o compartilhamento de dados entre diferentes nós (unidades de processamento) em um sistema robótico. Esses nós podem ser escritos em várias linguagens de programação, como C++, Python, e podem ser executados em diferentes computadores dentro de uma rede. Isso permite que os desenvolvedores dividam tarefas complexas em unidades menores e independentes, facilitando o desenvolvimento e a depuração de sistemas robóticos.

O ROS também oferece uma série de ferramentas e utilitários que auxiliam no desenvolvimento de robôs. Algumas das ferramentas mais conhecidas incluem o roscore, que é o componente central do ROS que gerencia a comunicação entre os nós, o roslaunch, que permite a inicialização de vários nós com um único comando, e o RViz, que é uma ferramenta de visualização 3D para depuração e análise de dados.

Além disso, o site oficial do ROS fornece uma vasta documentação, tutoriais e exemplos de código para ajudar os usuários a começarem a trabalhar com o ROS e explorar suas funcionalidades. Esses tutoriais abrangem uma ampla gama de tópicos, desde conceitos básicos, como instalação e configuração do ROS, até tópicos mais avançados.

2.3 MoveIt

O MoveIt é um poderoso framework projetado para o Robot Operating System (ROS) que simplifica o planejamento e controle de movimento para robôs manipuladores. Ele fornece uma ampla gama de recursos e funcionalidades que facilitam o desenvolvimento de aplicações robóticas complexas.

Através do MoveIt, é possível realizar tarefas como planejamento de trajetória, controle de movimento, detecção de colisões, cinemática inversa e interação com o ambiente. Ele oferece uma abordagem baseada em APIs e componentes reutilizáveis, o que agiliza o processo de implementação de funcionalidades avançadas em um sistema robótico.

Uma das principais vantagens do MoveIt é sua capacidade de trabalhar com uma variedade de configurações de robôs. Ele suporta diferentes tipos de manipuladores e pode ser adaptado para atender às necessidades específicas de diferentes aplicações. Além disso, o MoveIt é compatível com uma ampla gama de sensores e sistemas de percepção, permitindo uma integração perfeita com recursos adicionais, como visão computacional e reconhecimento de objetos.

O framework também possui ferramentas visuais e de simulação que facilitam o processo de desenvolvimento e depuração. Ele oferece uma interface gráfica intuitiva para visualizar e interagir com o robô, bem como realizar testes de movimento em ambientes virtuais antes de implantar em hardware real.

Para este trabalho, o MoveIt foi essencial simplificar significativamente o processo de planejamento e execução de movimentos utilizando cinemática inversa.

2.4 Gazebo

O Gazebo é um simulador de robôs amplamente utilizado na comunidade de robótica. Ele fornece um ambiente virtual realista onde é possível simular e testar o comportamento de robôs em diversas situações. O Gazebo é conhecido por sua capacidade de simular não apenas

a dinâmica dos robôs, mas também o ambiente ao seu redor, incluindo física, sensores e interações com objetos.

A combinação do Gazebo, ROS e MoveIt oferece um ambiente poderoso e flexível para prototipagem, desenvolvimento e teste de aplicações robóticas. Essa integração permite a realização de experimentos virtuais, análise de desempenho, otimização de algoritmos e a validação de sistemas robóticos antes de sua implementação física.

Para este projeto, o Gazebo é primariamente utilizado para validar os resultados dos movimentos realizados através de sua *engine* física.

3 Solução Proposta

Conforme explicitado na seção introdutória, o projeto foi dividido em 4 sub-etapas, que serão detalhadas nos tópicos a seguir:

3.1 Implementação dos scripts em Python

Foram elaborados 2 scripts em Python para permitir a execução do sistema. O primeiro deles é o script planejador, que atua como um equivalente a um *slicer* (fatiador) para impressoras 3D, que ao invés de realizar o tratamento de arquivos 3D, prepara arquivos de curva em PDF para os movimentos que serão executados pelo robô.

O segundo script será executado junto ao ROS + MoveIt e o Gazebo, a fim de fazer o envio dos comandos de movimentos gerados pelo script anterior.

Nos tópicos a seguir, será explicado em detalhes o funcionamento destes scripts.

3.1.1 Script planejador

O script planejador é responsável por realizar a leitura de arquivo PDF, contendo vetores com o caminho a ser percorrido na cor preta. O arquivo deve estar em escala e para padronizar o projeto, adotou-se a escala padrão de milímetros.

O arquivo então é processado utilizando a biblioteca OpenCV, para identificação dos segmentos e inserção dos pontos de caminho ao longo deles. É utilizado um valor de resolução, referente a um processo de filtragem com base da distância dessa contante. Portanto, a quantidade de pontos e consequentemente a sua fidelidade ao formato original são inversamente proporcionais ao valor atribuído a constante “RESOLUTION”. Para os casos testes, variou-se a resolução para valores como 5, 10, 30, 50, 80 e 100.

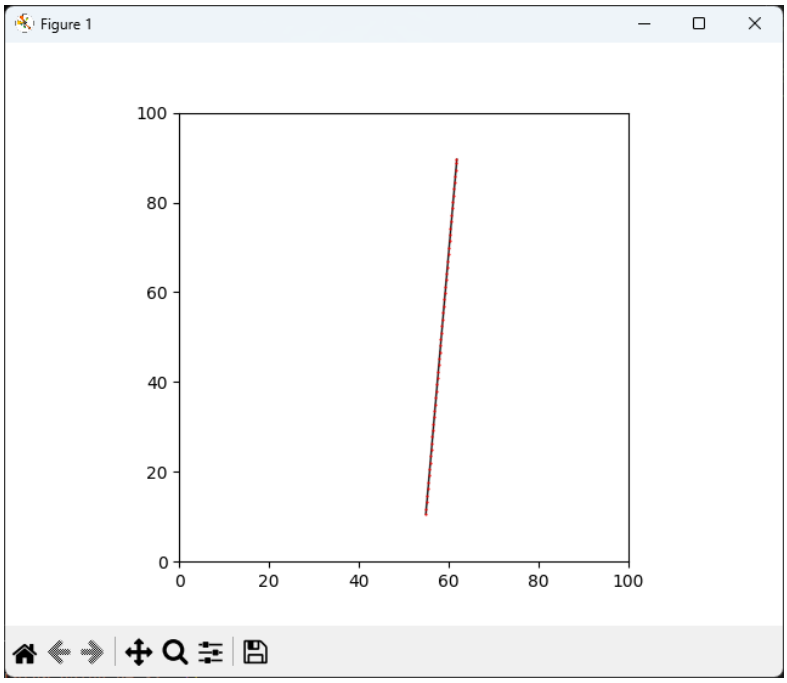

Figura 1- Resultado 2D do script planejador (Dataset 01)

Com os pontos encontrados, realiza-se então a transformação deles em pontos tridimensionais, aplicando-se um valor em Z de acordo com offset configurado na constante. Com as coordenadas tridimensionais, utiliza-se uma função para calcular os movimentos relativos entre os pontos, tendo-se como base o ponto anterior (excetuando-se o primeiro ponto).

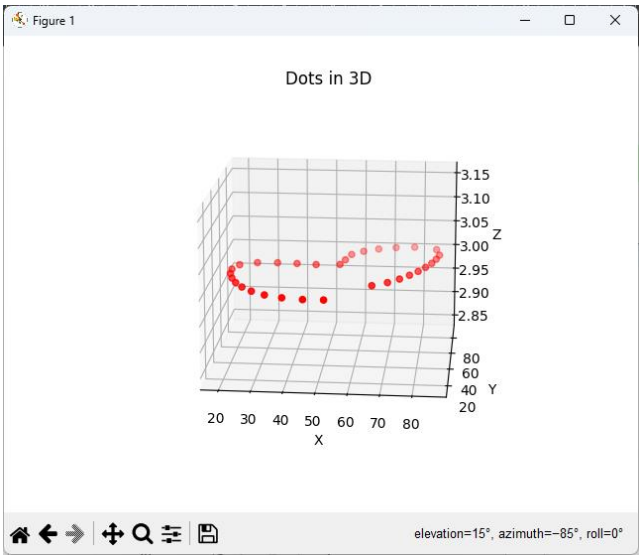

Figura 2- Resultado 3D do script planejador (Dataset 03)

Tanto as coordenadas quanto os movimentos relativos são exportados para arquivos .txt, para que sirvam de input para o script de execução.

Também é possível realizar o *plot* em 2D e em 3D dos pontos obtidos, para verificação do processo dos pontos gerados, principalmente em desenhos curvos.

Foi implementada uma função capaz de analisar os pontos e segmentos gerados e realizar o movimento de Z-Hop (aumento e diminuição em Z) para casos que houvessem mais de um segmento no arquivo. Porém, devido a problemas referentes a movimentos em Z em etapas futuras, optou-se por simplificar o código e remover a função responsável pelo Z-Hop. Caso houvesse mais tempo hábil para execução do trabalho, o grupo provavelmente iria reinserir a função em seu script, com o objetivo de testar datasets mais complexos.

3.1.2 Script de execução

O script de execução é responsável por ler os movimentos dispostos no arquivo .txt gerados pelo script planejador e realizar a seguinte sequência de passos:

1. Mover para uma posição inicial configurada (similar a um *home point*).
2. Mover para o primeiro ponto disposto no arquivo.
3. Mover-se sequencialmente sobre os pontos dispostos no arquivo até encontrar alguma condição de parada utilizando movimentos relativos.

Através da biblioteca “moveit_commander”, foram configurados e inicializados os comandos para manipulação do robô através do script.

A posição home foi definida alterando o valor de rotação das juntas para um valor determinado e utilizando a função da classe do move_group “go”.

Após atingir a posição de home, o script então faz a leitura do arquivo txt gerado pelo manipulador e itera sobre cada linha de pontos de movimentos relativos a partir do primeiro ponto.

A cada ciclo iterativo, o script soma os movimentos em X, Y e Z à pose atual, que utiliza o base_link (eixo 0) como referência. É realizado o planejamento da trajetória através da função compute_cartensian_path e então realizado o movimento.

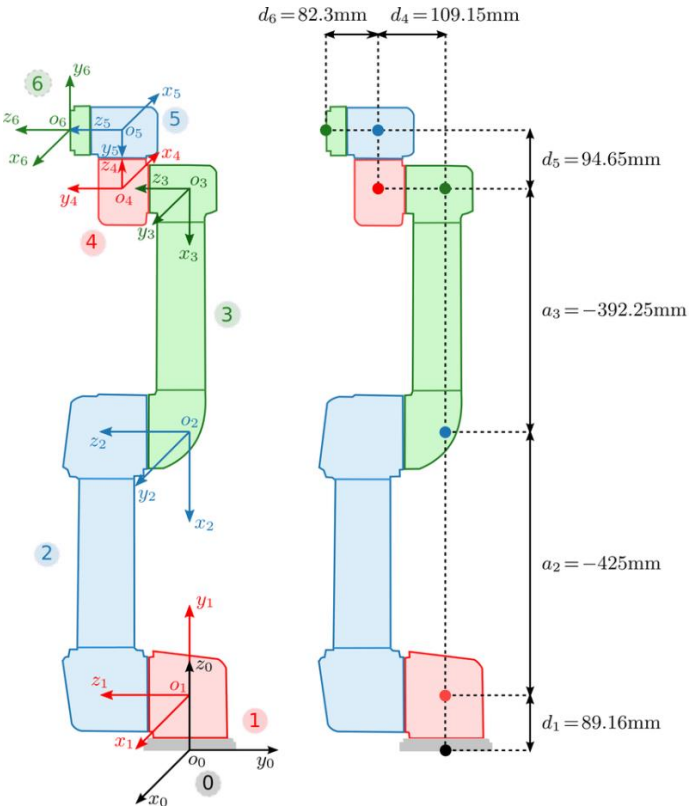

Figura 3 - Medidas de referência UR5

Após cada ciclo iterativo, também é checado se alguma condição de parada foi encontrada. Estas podem ser a execução de todos os pontos ou erros encontrados no processo de planejamento e execução da trajetória. Ou seja, caso o MoveIt não consiga computar alguma das trajetórias entre pontos, o robô para seu percurso de imediato e não prossegue.

3.2 ROS + RViz + MoveIt

A equipe decidiu adotar o ROS como framework de controle para o robô, seguindo a recomendação do professor. Optou-se pela versão NOETIC do ROS devido à ampla disponibilidade de pacotes e tutoriais para essa distribuição específica.

Utilizou-se o pacote da Universal Robotic, fabricante do braço robótico, baixando a versão Melodic, já que não havia uma versão compatível com a versão do ROS utilizada. Felizmente, os arquivos eram compatíveis e pode-se utilizá-la sem problemas.

O RViz foi utilizado para visualizar o robô e as trajetórias planejadas. O MoveIt foi utilizado para calcular a cinemática inversa, planejar rotas e através dele, executar o código em Python para realizar movimentos.

3.3 Gazebo

Com o intuito de simular o funcionamento do braço robótico UR5, utilizou-se o Gazebo para validação dos scripts implementados, utilizando os arquivos URDF obtidos do pacote. Para tal, adotou-se o modelo UR5 disponibilizado pelo fabricante em seu repositório no GitHub. Inicialmente, havia-se a intenção de implementar um atuador no braço para segurar um lápis ou caneta, no entanto, devido a dificuldades encontradas e à inexperiência da equipe, surgiram diversos problemas relacionados aos arquivos URDF. Diante dessa situação, optou-se por manter o braço desprovido de um atuador, uma vez que a presença de um *gripper* não é estritamente necessária para o escopo do projeto.

O ambiente modelado foi bastante simplificado, contando apenas com um braço fixado ao solo e um tampo de mesa flutuante, com o objetivo de simular o material em sua altura final em que se deseja realizar a marcação.

Para simulação, não foram implementados o elemento demarcador (caneta ou afins) no Gazebo, a fim de simplificar a simulação e tendo em vista que o ajuste de altura pode ser facilmente feito através do *offset* final aplicado em Z.

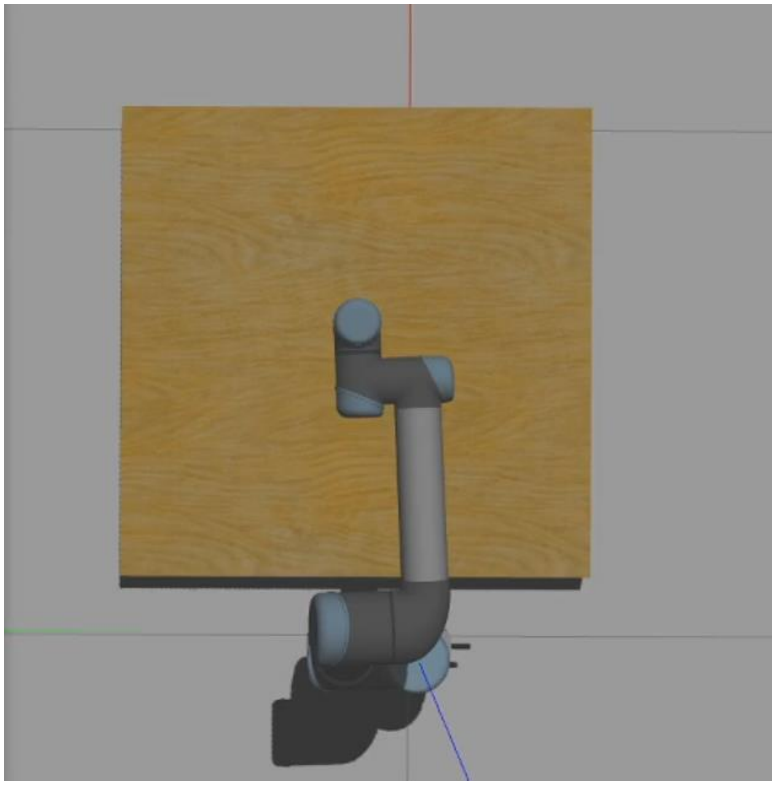

Figura 4 - Ambiente simulado através do Gazebo

3.4 Avaliação da Solução

Para realizar a avaliação da solução proposta, foram elaborados 3 *datasets* para atestar o funcionamento do sistema. O objetivo dos casos de teste era validar a capacidade do sistema de realizar caminhos abertos e fechados, retos ou curvos.

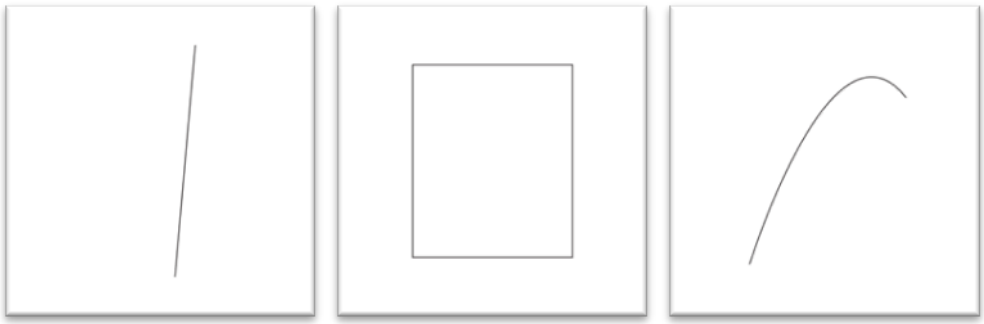

Figura 5 - Datasets propostos

Não foram considerados nos 3 casos iniciais situações como múltiplos traçados, a qual seria necessário implementar a funcionalidade de Z-Hop. Outros 2 *datasets* que simulavam estas situações chegaram a ser elaborados, porém esses casos não puderam ser executados em tempo hábil para entrega deste trabalho, devido aos movimentos indesejados no eixo Z encontrados durante a simulação dos primeiros casos e remoção da função que implementava o Z-Hop do script de planejamento.

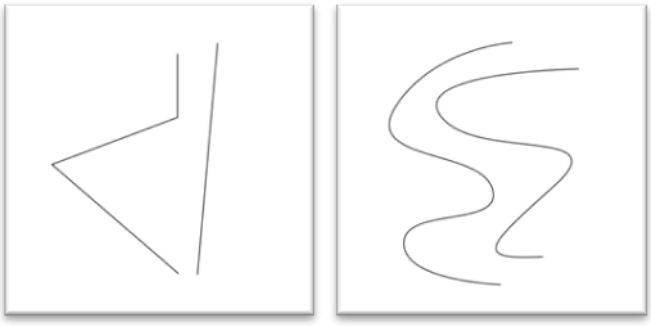

Figura 6 - Datasets que utilizam Z-Hop

Para verificação da corretude dos movimentos, utilizou-se primariamente a validação visual, apesar de que também é possível realizar a validação da operação através das saídas geradas no terminal executando o script, que a cada ciclo de iteração, imprimem na tela o status atual do robô e suas posições.

4 Resultados obtidos

O grupo adotou metas graduais de objetivos a serem alcançados, levando como base a complexidade crescente entre cada um deles.

4.1 Movimentação manual

Planejou-se inicialmente a realização de movimentos simples manualmente através da interface gráfica do MoveIt, realizando a visualização destes através do RViz.

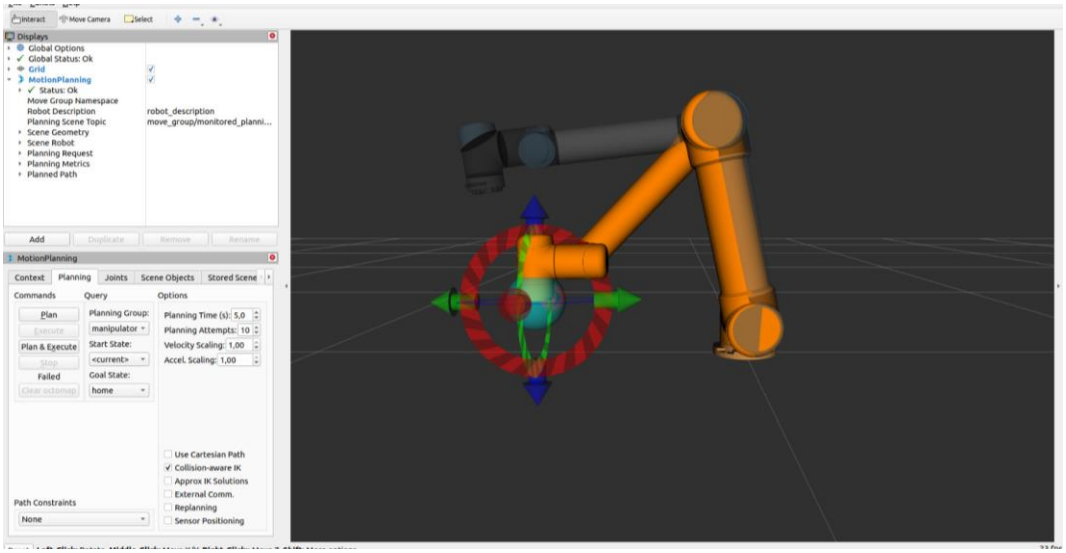

Figura 7- Realização de movimentos manuais via MoveIt

O grupo obteve sucesso em realizar movimentos alterando diretamente as rotações de cada junta, bem como realizar o planejamento e execução de trajetórias utilizando apenas a interface do MoveIt.

4.2 Movimentação via script

Após a familiarização com movimentos manuais, o grupo buscou utilizar scripts em Python simplificados para envio dos comandos. Os scripts iniciais buscavam apenas realizar

movimentações básicas com base em valores inseridos via terminal e foram escritos usando como base os tutoriais básicos disponíveis na documentação do ROS.

Após conseguir realizar a movimentação através de scripts simples, a dupla os usou de inspiração para implementar o script executor e então realizar o planejamento e a execução de movimentos iterativamente de forma autônoma, com base no caminho planejado.

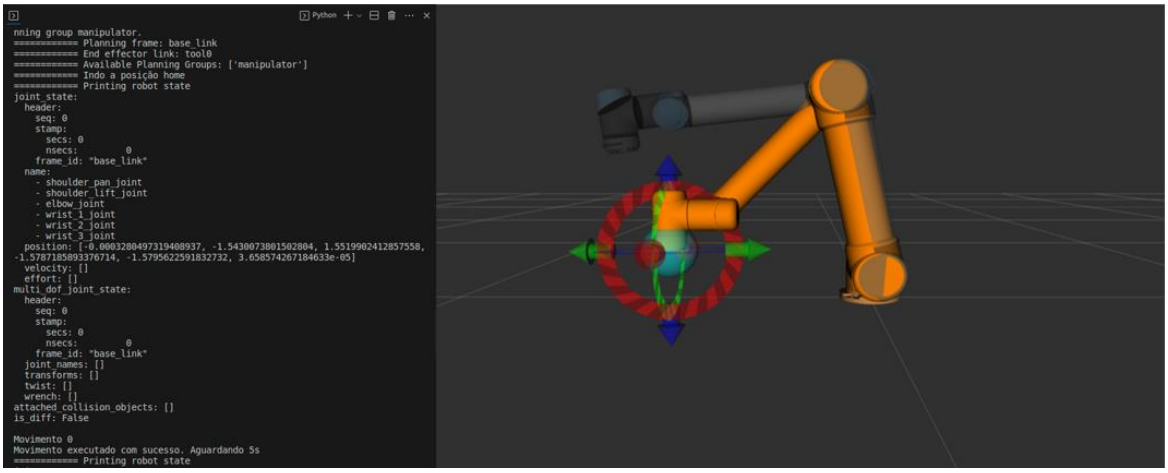

Figura 8- Realização de movimentos via script/terminal

Os resultados apresentados inicialmente no RViz + MoveIt foram satisfatórios, mas apresentaram pequenas divergências quando comparados ao simulado através do Gazebo.

4.3 Simulação através do Gazebo

Após validar a solução através do RViz, buscou-se realizar uma simulação do sistema através do Gazebo. Ao tentar executar o script executor, notou-se que em determinados movimentos realizados no processo iterativo, o frame 6 (localizado na ponta do braço robótico, onde o dispositivo marcador seria posicionado) apresentava um movimento indesejado de rotação e movimentação no eixo Z do sistema de referência (eixo 0 do robô).

O grupo não foi capaz de determinar em tempo hábil quais os motivos por trás deste efeito, mas levantou duas possíveis hipóteses para causa do problema.

Uma delas está em uma possível falha no frame de referência a qual o script está utilizando. Ao realizar a movimentação do robô, o eixo utilizando para movimentos pode estar sofrendo pequenas rotações, as quais as componentes de movimento em X e Y, que deveriam ser as únicas presentes, quando rotacionadas, são responsáveis por uma pequena componente em Z, que gera o efeito indesejado observado.

Apesar de ter sido configurado para utilizar o frame da base como referência, o grupo não tem conhecimento a fundo de como a biblioteca Python responsável pelo controle do robô funciona, podendo o script necessitar de alguma configuração adicional não implementada pela equipe.

Outra possível causa está no processo de modelagem e simulação do Gazebo e sua sensibilidade para pequenos movimentos. Para realizar o processo de movimentação ao longo da curva, o braço realiza uma série de movimentos curtos através dos pontos que representam a curva. Quando utilizada uma alta resolução, cada movimento pode ser milimétrico, e, neste processo, o Gazebo pode apresentar falhas na representação deste movimento, que podem ser responsáveis pelo movimento em Z indesejado.

Como a causa do problema não foi resolvida no espaço de tempo disponível para o desenvolvimento deste trabalho, a equipe adotou uma solução provisória de adicionar a movimentação relativa entre os pontos uma compensação (offset) para o movimento indesejado, visando neutralizá-lo.

Devido a este efeito indesejado, o grupo também evitou utilizar os *datasets* mais complexos elaborados, a fim de evitar conflitos ao utilizar o Z-Hop.

A solução paliativa da compensação do movimento indesejado foi suficiente para que o grupo conseguisse validar a execução do sistema para os 3 *datasets* propostos, considerando uma aplicação básica de sucesso para a proposta elaborada.

5 Conclusão

Apesar de algumas dificuldades envolvendo o processo de configuração e aprendizado inerente ao primeiro contato com os frameworks e ambientes de simulação utilizados, o grupo conseguiu implementar uma solução que atendesse ao objetivo proposto inicialmente para o projeto.

Algumas simplificações tiveram de ser adotadas, bem como soluções paliativas para alguns problemas encontrados durante o processo, a fim de que produto funcional pudesse ser entregue dentro do prazo disponibilizado.

O grupo acredita que se houvesse mais tempo disponível, tendo em vista que o projeto foi efetivamente implementado em um período de cerca de um mês, em um momento conturbado de final de semestre, seria possível resolver a grande maioria dos problemas

encontrados e entregar uma solução mais completa e robusta, sendo capaz de solucionar todos os conjuntos de testes propostos inicialmente.

No entanto, apesar dos entraves, a dupla se demonstra satisfeita com o resultado apresentado e os conhecimentos adquiridos ao longo da disciplina ofertada e do processo de elaboração deste trabalho.

6 Referências Bibliográficas

ROS.org. Robot Operating System. Disponível em: https://www.ros.org/. Acesso em: 12 jul. 2023.

MoveIt. MoveIt - ROS Manipulation Framework. Disponível em: https://moveit.ros.org/. Acesso em: 12 jul. 2023.

Gazebo. Gazebo Simulator. Disponível em: https://gazebosim.org/home. Acesso em: 12 jul. 2023.

ROS-Industrial. Universal Robot. Disponível em: https://github.com/ros-industrial/universal_robot. Acesso em: 12 jul. 2023.

OpenCV. Documentação da biblioteca OpenCV. Disponível em: https://docs.opencv.org/. Acesso em: 12 jul. 2023.