Universidade de Brasília
Faculdade de Tecnologia

Planejamento de rota do robô manipulador UR3

Guilherme de Castro Ribeiro

TRABALHO DE GRADUAÇÃO

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Brasilía

2022

Universidade de Brasília
Faculdade de Tecnologia

Planejamento de rota do robô manipulador UR3

Guilherme de Castro Ribeiro

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

Orientador: Prof. Dr. Roberto de Souza Baptista

Brasilía
2022

Castro Ribeiro, Guilherme de.
C355p Planejamento de rota do robô manipulador UR3 / Guilherme de Castro Ribeiro; orientador Roberto de Souza Baptista. -- Brasília, 2022.
103 p.

Trabalho de Graduação em Engenharia de Controle e Automação -- Universidade de Brasília, 2022.

1. planejamento de rota. 2. robô manipulador. 3. ur3. 4. rrt. I. de Souza Baptista, Roberto, orient. II. Título

Universidade de Brasília
Faculdade de Tecnologia

Planejamento de rota do robô manipulador UR3

Guilherme de Castro Ribeiro

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

Trabalho aprovado. Brasilía, 13 de junho de 2022:

Prof. Dr. Roberto de Souza Baptista,
UnB/FGA
Orientador

Prof. Dr. Geovany Araújo Borges,
UnB/ENE
Examinador interno

Prof. Dr. Claudia Patricia Ochoa Diaz,
UnB/FGA
Examinador interno

Brasilía
2022

Agradecimentos

À toda minha família, agradeço pelo incentivo, suporte e apoio ao longo da minha trajetória na UnB.

À todos os membros e ex-membros da equipe Unbeatables, sou grato por todas as experiências proporcionadas e por todos os momentos de aprendizado, riso e tristeza que passamos juntos. Em especial à Débora, Lívia e Paulo, por todas conversas, conselhos e companheirismo. Fazer parte dessa equipe foi uma das melhores experiências que vivenciei na UnB.

À todos os amigos da empresa júnior Mecajun, que me inspirou e propiciou crescimento profissional e pessoal. Em especial ao Leonardo Fonseca e Matheus Bawden.

Não posso deixar de agradecer também aos meus amigos de longa data que a tantos anos me acompanham e me apoiam: Gabriel Gouveia, Vinicius Cavalcante, Igor Roberto, Wallisson Miqueias, Mateus Soares, Sammer Vinicius, Pedro Alves, Caio César, Caio Camilo, Gabriel Santos, Fernando Mota, Paulo Moreira e Matheus Escovedo.

Gostaria de agradecer ao meu orientador, professor Roberto Baptista, por toda atenção, suporte e sugestões para o desenvolvimento desse trabalho.

Por fim, agradeço a professora Mariana Bernardes pelo apoio, incentivo, atenção e orientação nos estágios iniciais desse e outros projetos.

Guilherme de Castro Ribeiro

Resumo

O desenvolvimento industrial vêm fortalecendo a tendência viabilizar ambiente para que humanos e robôs dividam o mesmo espaço de trabalho. Neste trabalho é desenvolvido um estudo sobre as arquiteturas de robôs manipuladores, mais especificamente, o robô UR3 e a realização do seu planejamento de rota em ambientes com obstáculos, em que é necessário um processo de coordenação para determinar a melhor ação a ser tomada antes de executá-la.

A estratégia de planejamento utilizada baseou-se no algoritmo RRT e, sua variação, RRT-Connect, sendo validado o funcionamento dessas técnicas e realizado sua aplicação para o planejamento de rota de um robô manipulador. Ao fim, foi possível comparar a eficiência entre cada algoritmo, provando que o RRT padrão consegue gerar rotas com perfil mais direto ao alvo, enquanto que o RRT-Connect consegue alcançar resultados com maior velocidade.

Palavras-chave: planejamento de rota. robô manipulador. ur3. rrt.

Abstract

Industrial development has strengthened the trend to enable an environment for humans and robots to share the same workspace. In this work, a study is developed on the architectures of manipulator robots, more specifically, the UR3 robot and the realization of its path planning in environments with obstacles, in which a coordination process is necessary to determine the best action to be taken before run it.

The planning strategy used was based on the RRT algorithm and, its variation, RRT-Connect, validating the operation of these techniques and carrying out their application for the path planning of a manipulator robot. In the end, it was possible to compare the efficiency between each algorithm, proving that the standard RRT can generate paths with a more direct profile to the target, while the RRT-Connect can achieve results with greater speed.

Keywords: path planning. manipulator robot. ur3. rrt.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

PRM	Probabilıstic Roadmaps 35
RRT	*Rapidly-exploring Random Tree* 18
UnB	Universidade de Brasília 17
URDF	Unified Robot Description Format 65

Lista de símbolos

Sumário

1 Introdução

1.1 Contextualização

Com a crescente demanda por tecnologias para suprir e otimizar a realização de certas atividades, reduzindo gastos e poupando esforços humanos, o estudo e desenvolvimento de robôs capazes de desempenhar atividades repetitivas, que exigem precisão e força física, se tornou cada vez mais relevante dentro de diferentes contextos da sociedade. O rápido crescimento do uso de robôs em inúmeras atividades, motivado pela exigência por melhoria em produtividade, qualidade e condições de trabalho, também evidenciou a necessidade de garantir um espaço de trabalho propícia para a movimentação do mesmo ou, alternativamente, garantir que o robô consiga realizar seus movimentos evitando qualquer colisão em pessoas ou objetos.

Desde os primeiros indícios de desenvolvimento de dispositivos capazes de exercer atividades feitas, até então, pelo homem, o ambiente industrial tem sido o principal setor para o desenvolvimento e aplicação de novas tecnologias. Apoiado nos primeiros modelos, como o Unimate (Figura 1) em 1968 (KAWASAKI, 2000), a relação robótica-indústria tornou-se, cada vez mais, numa associação que frequentemente envolve a pesquisa em robótica financiada pela indústria para alcançar níveis de otimização e automação cada vez maiores.

Figura 1 – Primeiro Robô Manipulador - Unimate

Fonte: Kawasaki (2000)

A aplicação de robôs na industria, principalmente os manipuladores, já é uma realidade em todo o contexto moderno. Temos que diversos segmentos industriais utilizam diferentes configurações para diferentes aplicações, como manipulação de materiais tóxicos, soldagem de peças de automóveis, pintura e montagem, mas que, muitas vezes, ainda há

necessidade de acompanhamento ou atuação humana no processo. Mesmo sendo uma tecnologia bastante interessante na automação de processos que envolvam a manipulação de objetos, o uso de braços robóticos pode gerar diversos riscos. Para garantir a segurança em ambiente industrial, foi visto a necessidade de desenvolvimento de ambientes capazes de proporcionar uma cooperação humano-robô.

Atualmente, embora haja grande gama de atividades com altas taxas de robotização da produção, existem setores de trabalho em que há uma indispensabilidade do trabalho humano. Embora tradicionalmente os robôs sejam muito bons para a execução das tarefas para as quais foram projetados, há possibilidade de ocorrer situações imprevistas, fugindo do que foram preparados para realizar.

Tendo em vista todas suas possíveis aplicações e a extrema importância para que trabalhem de forma segura e como foram projetados, é notório a necessidade de que o seu projeto de desenvolvimento seja feito a partir de estudos e validações com grande rigor. A modelagem, simulação e controle de um robô desse tipo recebe uma enorme atenção e possibilita a constante evolução de tal tecnologia. Basicamente, o propósito de um robô manipulador está em manter, com precisão, a resposta dinâmica determinada em projeto, seguindo os objetivos da atividade a ser desempenhada, e garantir segurança à todos que estejam nas proximidades.

1.2 Definição do problema

No Laboratório de Automação e Robótica (LARA), da Universidade de Brasília, encontra-se disponível o manipulador robótico UR3, Figura 2, desenvolvido pela Universal Robotics (UNIVERSAL..., 2017). Suas especificações são melhor descritas na seção 1.3.

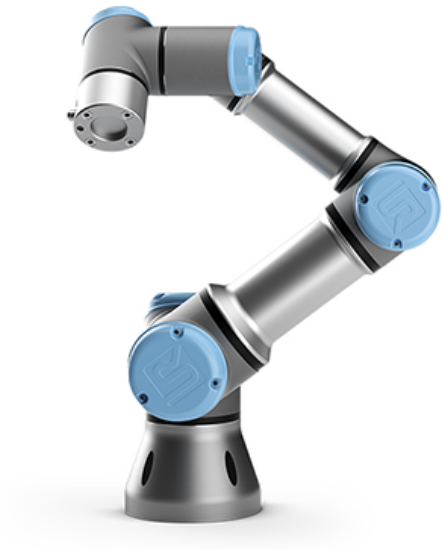

Figura 2 – Robô Manipulador UR3.

Fonte: UNIVERSAL... (2017)

Desenvolvido para realização de atividades repetitivas e de precisão em ambientes industriais, ele é utilizado amplamente como objeto de estudo em pesquisas na área de ambientes cooperativos entre humanos e robôs, com intuito de garantir seu perfeito funcionamento de forma que não represente comportamentos que podem caracterizar como uma ameaça ao humano que estiver trabalhando com o auxilio do mesmo.

Para tal, é necessário a realização de um profundo estudo da estrutura do robô, como cada componente deve atuar em conjunto no sistema, desenvolvimento da modelagem completa do sistema, controle e, por fim, planejamento do movimento desejado.

Como tais estudos não encontram-se abertamente divulgados e detalhados por parte da desenvolvedora do robô, para um completo entendimento de seu funcionamento, é necessário o estudo desde os fundamentos de robótica até a realização de testes práticos com o robô em questão.

Desta forma, neste trabalho, propõe-se realizar uma caracterização desta modelagem do robô, suas formas de controle, focado em métodos baseados na cinemática para compreender seu funcionamento, e a realização da implementação mais detalhada do planejamento da rota a ser seguida considerando diferentes obstáculos no ambiente inserido. Para esta ultima, será analisado diferentes métodos já conhecidos publicamente, dando foco e implementando os métodos apresentados por Lavalle e James J. Kuffner (2000), conhecidos como os algoritmos RRT, sigla em inglês para Árvore aleatória de exploração rápida (*Rapidly-exploring Random Tree*), e sua variação, o RRT-Connect.

Assim sendo, servirá como colaboração para o desenvolvimento dos estudos produzi-

dos no laboratório, propondo-se contribuir na caracterização dos sistemas disponíveis, bem como apresentar uma abordagem sobre a ótica da elaboração do planejamento de rota para manipuladores. Utilizando uma metodologia baseada na ideia de evoluir e desenvolver o conhecimento ao longo das etapas *Teoria*→ *Matlab*→ *Python*→ *Simulação*.

1.3 Objeto de estudo

1.3.1 Descrição do sistema

O UR3, figura 2, é um robô colaborativo de mesa. Com sua carga útil de 3 kg, ele é muito versátil e seu tamanho reduzido o torna adequado para situações de espaço de trabalho limitado. Com seu giro infinito na junta final, diversas atividades podem ser realizadas com as garras fixadas no conector da ferramenta do robô. Dentre suas possíveis aplicações, pode-se citar: polimento, soldagem, pintura, movimentação de objetos, operação com ferramentas e várias outras atividades de montagem.

Suas especificações estão descritas na tabela 1.

Tabela 1 – Especificações do UR3.

Peso	11,2 kg
Carga útil	3 kg
Raio de alcance	500 mm
Área ocupada	ϕ 128 mm
Graus de liberdade	6 juntas rotativas
Alcance das juntas	+/- 360°
Velocidade de rotação das juntas	360 graus/segundos

1.3.2 Caixa de Controle

A Caixa de Controle, figura 3, contém ambas entradas e saídas analógicas e digitais que podem ser usadas para fazer a interface de outros componentes ou dos próprios componentes do sistema. A unidade de programação pode ser usada para programar o robô de acordo com os requisitos do usuário e pode ser baseada em entradas e saídas.

Usando esta Caixa de Controle, o robô pode ser configurado rapidamente por um operador sem experiência em programação usando tecnologia patenteada (UNIVERSAL..., 2017) e pode ser operado com uma visualização 3D intuitiva. Requer um movimento simples do braço robótico, dando pontos de referência ou a partir dos controles fornecidos no touchpad.

Figura 3 – Caixa de Controle do sistema.

Fonte: UNIVERSAL... (2017)

1.3.3 Comunicação

Como ilustrado na figura 4, existem duas portas de comunicação principais neste sistema. Primeiramente, há uma porta de comunicação TCP / IP entre um computador com sistema operacional Linux (será chamado por *Linux-PC*) e a caixa de controle do UR3. Basicamente, é possível enviar comandos ROS diretamente do Terminal Linux ou softwares de simulação especializados. Em seguida, há uma porta de comunicação serial entre a Caixa de Controle e o Robô UR3, sendo responsável por enviar todos os comandos de posição e velocidade para o robô.

Especificações extras:

- TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX
- Ethernet socket Modbus TCP

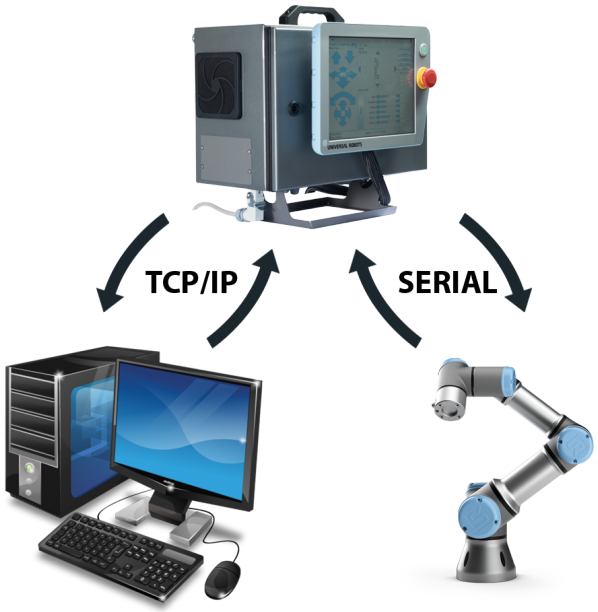

Figura 4 – Comunicação UR3 - Caixa de Controle - PC

1.3.4 Espaço de trabalho

Estabelecido pela região do volume de espaço em que é possível a movimentação do efetuador terminal, o UR3 possui um espaço de trabalho que se estende em até 500mm da sua junta da base, como ilustrado na Figura 5.

A configuração do UR3, baseada em 6 juntas de revolução, permitem, por definição (ROSARIO, 2010) e por analise, perceber que trata-se de uma configuração com classificação de *robô articulado*. No geral, esse é o tipo de configuração que permite a maior mobilidade a robôs, mas, também, seu volume de trabalho apresenta uma geometria mais complexa em relação as outras configurações.

Devido ao posicionamento e tipo das juntas, essa configuração possui a limitação de alcance no formato de um volume cilíndrico sobre a junta da base, como ilustrado na Figura 5. E, segundo o fabricante (UR3/CB3..., 2021), deve-se evitar realizar qualquer movimento muito próximo à esse volume cilíndrico, uma vez que forçaria as juntas a atuarem de forma rápida, embora o efetuador terminal mova-se lentamente.

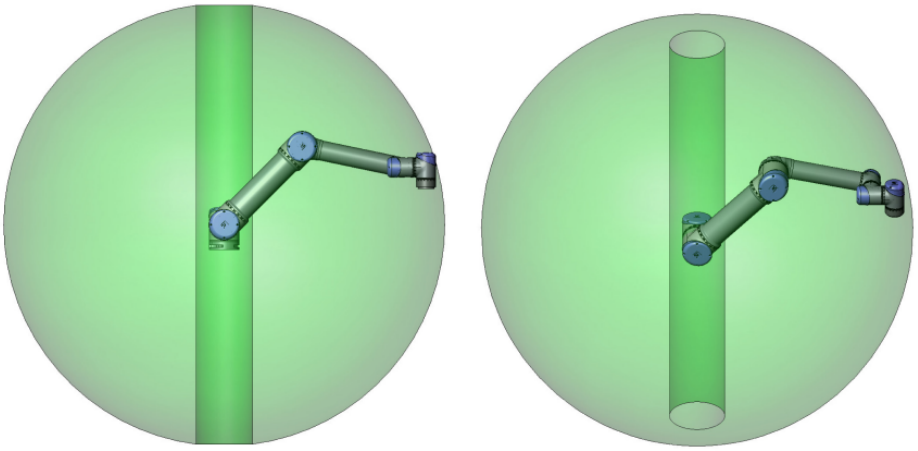

Figura 5 – Espaço de trabalho do robô UR3. Visão frontal e inclinada, respectivamente.

Fonte: UR3/CB3… (2021)

1.3.5 Sistema Linux

Além da Caixa de Controle, é possível controlar o sistema do robô a partir do Linux. Utilizando o ROS, C++ e/ou Python e garantindo que seu hardware, computador e robô, esteja devidamente configurados para se comunicarem, é possível realizar qualquer movimento ou percurso diretamente do seu PC, tão bom, ou melhor, como a partir da Caixa de Controle.

1.4 Ferramentas

1.4.1 ROS

O Robotics Operating System (ROS) é um framework de software amplamente utilizado para aplicações em robótica, reunindo as principais funcionalidades necessárias para implementações de projetos na área em conjunto com um sistema de comunicação distribuída que permite o uso de diferentes linguagens de programação no mesmo projeto (ROS, 2022). Sua aplicabilidade, versatilidade e funcionalidades foram as principais características que propiciou seu crescimento de popularidade dentro da comunidade de robótica, ocasionando em sua admissão em projetos comerciais, alcançando, até mesmo, aplicações em linhas de produção industriais, como ARM, BMW, Boeing e Bosch (INDUSTRIAL, 2022).

As bibliotecas implementadas para esse sistema possuem algoritmos de planejamento, percepção, mapeamento e ferramentas de simulação. Além do gerenciamento de processos programados em diferentes linguagens, possui também bibliotecas para realizar o controle de hardware em baixo nível. Os três conceitos fundamentais do ROS são:

- Nós: elementos responsáveis por executar as diferentes tarefas/processos do sistema.

O ROS foi projetado para ser altamente modular, funcionando de forma que cada nó consegue controlar/gerenciar parte do sistema independentemente. Por exemplo: um nó controla os motores das juntas, um nó faz o planejamento de trajetória e um nó realiza o processo de localização.

- Tópicos: canais utilizados para troca de mensagens entre nós. Cada tópico pode ser lido ou receber publicações de diferentes nós ao mesmo tempo.
- Mensagens: Tipos de estruturas utilizadas para transmissão de dados entre estruturas. Além dos tipos padrão de dados, o ROS permite a criação de estruturas personalizadas pelo projetista para atender necessidades específicas do sistema aplicado.

Outro aspecto importante desse sistema é a forma em que os arquivos devem ser organizados, ilustrado na Figura 6. O funcionamento, no geral, depende da formatação e definições dos pacotes (*Packages*), que contêm um ou mais programas/nós, bibliotecas, arquivos de configuração e assim por diante, que são organizados como uma única unidade.

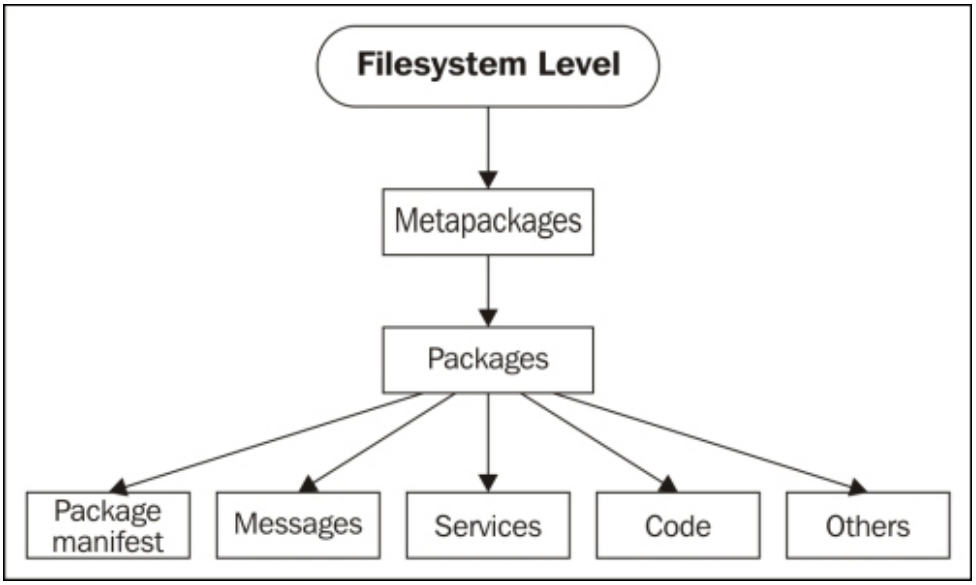

Figura 6 – Organização estrutural de arquivos em um projeto ROS padrão.

Fonte: MIT16.485 (2022)

1.4.2 MATLAB

No contexto de modelagem, análise e simulação de projetos para aplicações em robótica, os softwares MATLAB e Simulink são de extrema importância e são ferramentas excepcionalmente úteis para engenheiros e pesquisadores.

Atualmente, em qualquer tipo de instituição que se envolva com projetos em robótica, a simulação assume um papel considerável na forma como é implementado nos diferentes tipos de robôs, de forma a obter sucesso e validar seus processos. Sendo assim, tem-se

que práticas de análise prévia são importantes fatores que as soluções alcancem o que é pretendido, gerando benefícios como diminuição dos custos, economia de tempo e redução de risco. Portanto o uso de simulação possibilita utilizar o modelo que representa um sistema qualquer, um robô, e simular o seu comportamento de forma a poder-se efetuar uma avaliação antecipada do seu desempenho.

Com a utilização da ferramenta Robotics Toolbox, de Perter Corke (CORKE, 2013; ROBOTICS..., 2017) , no software MATLAB, através da modelagem de um braço robótico, e a possibilidade de elaboração da sua programação para simulação da movimentação dos seus eixos, percebe-se, mais uma vez, como software desse tipo podem impulsionar qualquer tipo de pesquisa.

Esta ferramenta nos permite a análise dos parâmetros de controle das juntas do equipamento, verificação da sua funcionalidade e sua movimentação no ambiente de trabalho, permitindo uma prévia análise de viabilidade de um sistema robótico.

1.4.3 CoppeliaSim

Já no contexto de simulação de fato, envolvendo representação em 3D de todo o ambiente desejado, incluindo robôs, objetos, móveis e, até mesmo, pessoas, foi utilizado o simulador CoppeliaSim, também conhecido pelo seu antigo nome: V-Rep.

É um simulador amplamente utilizado para comunidade de robótica por sua capacidade de criar ambientes com diversas situações aplicáveis para um robô físico, sem necessariamente depender da existência do mesmo durante o desenvolvimento e/ou testes (COPPELIA, 2022).

Permite o desenvolvimento com scripts Python e Lua, ou plug-ins C/C++ atuando como controladores síncronos individuais. Controladores assíncronos adicionais podem ser executados, também, através de comunicação externa por meio de soluções baseadas em ROS ou API's por meio de projetos em C/C++, Python, Java e Matlab.

Para simular as interações com corpos rígidos, é possível utilizar os mecanismo inerentes do software para cálculos de cinemática direta e inversa e várias bibliotecas de simulação de física (Bullet, ODE, Vortex, Newton Game Dynamics). Complementando as funções para simulação, também pode-se encontrar disponível os módulos responsáveis por representação de objetos (malhas, juntas, sensores, etc.) e funções fornecidas por plug-ins, que incluem: planejamento de movimento (via OMPL), visão sintética e processamento de imagens (por exemplo, via OpenCV), detecção de colisão, cálculo de distância mínima e entre outros.

1.5 Apresentação do manuscrito

O Capítulo 2 apresenta a fundamentação teórica utilizada para desenvolvimento do trabalho, abordando conceitos importantes para realização do mesmo, como modelagem cinemática do sistema do robô e algoritmos de planejamento de rota. São apresentadas equações, fórmulas e algoritmos, bem como o raciocínio por trás. O Capítulo 3 apresenta uma explicação de cada procedimento efetuado, demostrando as etapas aplicados no desenvolvimento do trabalho. O Capítulo 4 apresenta os resultados finais obtidos após integração de todas as etapas desenvolvidas. Por fim, o Capítulo 5 conclui este documento com comentários sobre os resultados obtidos e considerações para trabalhos futuros.

2 Fundamentos

2.1 Manipuladores Robóticos

Como definido por Rosário (ROSARIO, 2010), robôs são mecanismos de atuação programáveis com dois ou mais eixos com algum grau de autonomia, sendo capazes se movimentar de alguma forma no ambiente que está inserido com objetivo de executar alguma tarefa. São dotados de sistemas de controle como forma de garantir a percepção e os acionamentos conforme solicitado.

Para os robôs do tipo manipulador, por sua vez, tem-se que é uma classe de robôs formados por uma série de segmentos/elos conectados através de juntas ou com deslizamento entre si, com o propósito de agarrar e/ou mover objetos de um lugar a outro.

No desenvolvimento deste trabalho foi utilizado o manipulador robótico UR3 (seção 1.3), que é composto por 6 elos com atuadores rotativos e havendo possibilidade de acoplamento de um efetuador terminal, como uma garra.

No contexto de se controlar a posição e orientação dos manipuladores no espaço para execução de uma tarefa específica é preciso definir modelos matemáticos que permitam o relacionamento entre as características físicas do robô e as ações desejadas, tornando-se essencial o completo entendimento de todos aspectos geométricos do seu movimento. Para isso, existe a área de estudo da cinemática, que é o ramo da mecânica que estuda a descrição do movimento, e, ainda dentro do contexto de robôs manipuladores, a cinemática analisa a relação entre as posições das juntas e o posicionamento do efetuador terminal. A partir disso, é possível

2.2 Modelagem Cinemática

Para a representação e modelagem dos movimentos dos manipuladores é visto a necessidade utilização de técnicas para representar a posição de determinado ponto do braço dentro do ambiente em questão. Esta representação depende da posição das juntas e dos elos, sendo que, por convenção, é utilizado a base do robô como ponto de referência.

A modelagem cinemática de manipuladores permite, como ilustrado na figura 7, obter-se os parâmetros do problema observado de duas formas diferentes:

- Cinemática direta: Obter as coordenadas cartesianas do efetuador terminal no referencial da base, conhecendo as variáveis de junta, ou seja, os ângulos ou deslocamentos nas juntas.

- Cinemática inversa: Obter as variáveis de junta, ou seja, os ângulos ou deslocamentos conhecendo a posição do efetuador terminal no referencial da base.

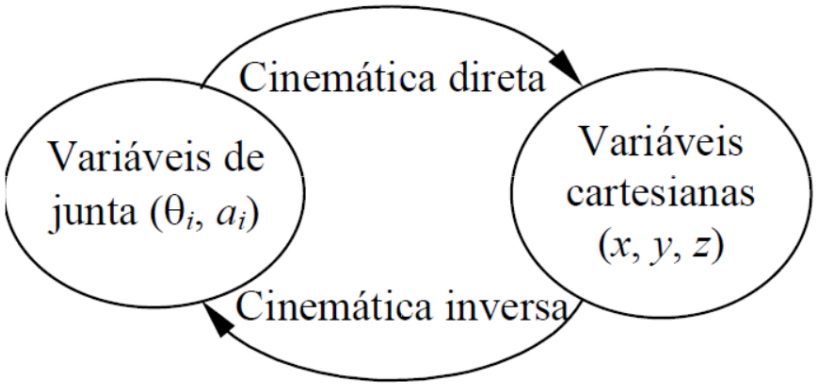

Figura 7 – Modelagem cinemática.

Fonte: Carrara (2015)

2.2.1 Cinemática Inversa

A cinemática inversa obtém as variáveis de junta, ou seja, os ângulos ou deslocamentos conhecendo a posição do efetuador terminal no referencial da base. Sendo assim, a grande importância desse método está em contribuir para um melhor entendimento e viabilizar um mais fácil transformação das especificações do movimento, designadas para o efetuador terminal no espaço operacional em movimentos no espaço das juntas, permitindo a execução do movimento desejado.

Em manipuladores como o UR3, a cinemática torna-se mais complexa devido sua quantidade de juntas e elos. Independentemente da geometria do manipulador, a solução da cinemática requer conhecimento de geometria, trigonometria e cálculo vetorial, mas, somada à complexidade da plataforma UR3, o desafio em desenvolver tal estudo pode se tornar muito maior. Sendo assim, a utilização de métodos estruturados para facilitação da descoberta das equações de cinemática representam formas mais indicadas para sistematizar ou simplificar esse processo. Uma de tais convenções, e a mais conhecida, é a notação de Denavit-Hartenberg.

2.2.2 Notação de Denavit-Hartenberg

Denavit e Hartenberg (ROSARIO, 2010; CRAIG, 2012), em 1955, propuseram uma notação sistemática para atribuir um sistema de coordenadas ortonormal a partir dos nomeados parâmetros de Denavit-Hartenberg, que são quatro indicadores que caracterizam a posição relativa entre dois eixos de coordenadas. Uma vez que os sistemas de coordenadas

fixados em cada elo são atribuídos, transformações básicas envolvendo rotações e translações entre sistemas de coordenadas adjacentes podem ser representadas por uma através de técnicas de matrizes de transformação de coordenadas homogêneas.

A representação Denavit-Hartenberg de um elo rígido depende de quatro parâmetros a ele associados, os quais descrevem completamente o comportamento cinemático de uma junta prismática ou revoluta. Na Figura 8 são indicados os parâmetros para uma melhor visualização.

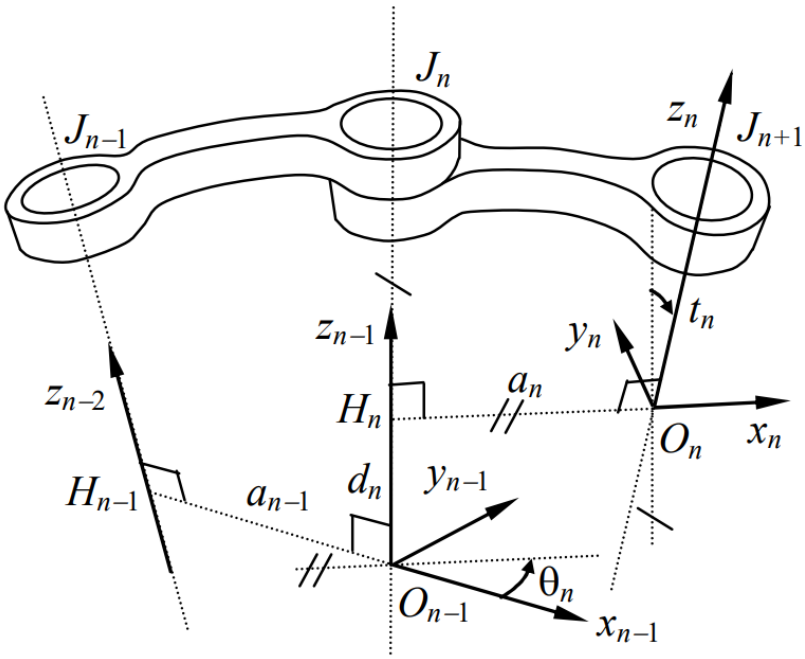

Figura 8 – Geometria de juntas rotativas e parâmetros das juntas.

Fonte: Carrara (2015)

Para se obter os quatro parâmetros, denominados a, α, d e θ, deve-se seguir uma série de diretivas de forma a definir dois eixos de coordenadas em duas juntas, i e $i+1$, e então determinar a relação entre elas. Para definir tais parâmetros, tem-se que:

- α: é o ângulo de z_{i-1} para z_i em torno de x_i;
- a: é a distância ao longo de x_i entre a interseção dos eixos x_i e z_{i-1} e O_i;
- d:é a distância ao longo de z_{i-1} entre O_{i-1} e a interseção dois eixos x_i e z_{i-1};
- θ: é o ângulo de x_{i-1} para x_i em torno de z_{i-1}.

E, para clara definição de cada um desses parâmetros, é necessário definir os eixos de coordenadas em cada junta. Para isso, é utilizado as seguintes regras:

1. Escolhe-se o eixo z_i e, é aconselhável, determina-lo de forma a alinhá-lo com o eixo de movimentação da junta, ou seja, eixo de rotação para juntas de revolução;
2. Estabelece-se o sistema da base O_0, x_0, y_0 e z_0;
3. Localiza-se a origem O_i, sendo que é normal comum a z_i e z_{i-1};
4. O eixo x_i deve ser posicionado paralelo à normal comum entre z_i e z_{i-1}. Caso i seja a primeira junta, a posição de x_i pode ser arbitrária.
5. O eixo y_i é definido de acordo com x_i e z_i, utilizando-se a Regra da Mão Direita, um para cada elo numa cadeia cinemática aberta de elos.

Em seguida, é possível calcular cada transformação entre as juntas do robô, aplicando todas as operações matriciais de translação e rotação necessárias, com representado na equações 2.1, 2.2, 2.3 e 2.4, sendo que A^i_{i-1} representa a transformação homogênea entre o elo *i* e *i-1*, e H^n_{i-1} representa a matriz final de transformações.

$$A^i_{i-1} = Rot(z,\theta_i)Trans(z,d_i)Trans(x,a_i)Rot(x,\alpha_i) \tag{2.1}$$

$$A^i_{i-1} = \begin{bmatrix} cos\theta_i & sin\theta_i & 0 & 0 \\ sin\theta_i & cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha_i & -sin\alpha_i & 0 \\ 0 & sin\alpha_i & cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$A^i_{i-1} = \begin{bmatrix} cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i sin\alpha_i & a_i cos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & a_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

$$H^n_{i-1} = A^1_0 A^2_1 ... A^n_{n-1} \tag{2.4}$$

2.3 Planejamento de rota

Ao longo dos anos, diferentes técnicas e abordagens foram desenvolvidas para resolver o problema de planejamento de caminhos. No geral, existe um grande número de métodos para resolver problemas de planejamento de rota. Entretanto, nem todos resolvem todas as nuances deste problema.

Por exemplo, alguns métodos necessitam que o espaço de trabalho seja bidimensional e os obstáculos poligonais. No entanto, apesar de muitas diferenças externas, muitos métodos

são baseados em algumas técnicas gerais. Segundo Corke (2013) e Latombe (1991), essas técnicas podem ser listadas em três diferentes categorias:

- Campos potenciais;
- Decomposição em células;
- Mapa de rotas.

2.3.1 Abordagens clássicas

No geral, para a realização de um movimento desejado e de forma planejada para um braço robótico, precisa-se, primeiramente, definir-se quais são os pontos de partida e de destino; identificar/definir obstáculos no ambiente; traçar pelo menos 1 caminho sem colisões, considerando as configurações factíveis para as juntas do robô; e, por fim, seguir o caminho. Seguindo essa premissa, as principais características de cada uma das técnicas citadas na seção 2.3 serão brevemente descritas nas subseções abaixo.

2.3.1.1 Campos potenciais

Os campos potenciais foram propostos para planejar rotas por (KHATIB, 1986). Nessa técnica, um campo potencial é criado considerando o objetivo (destino) e os obstáculos existentes no ambiente. Os obstáculos influenciam o campo potencial, criando uma força repulsiva que faz a rota de possível escolha ser afastada desse obstáculo. Já o objetivo influencia no campo potencial criando uma força atrativa, levando o caminho traçado até o seu destino. Por fim, têm-se que um objeto/robô que fosse deixado no ponto de inicio indicado, tenderia a seguir um caminho a partir das influencias de força descritas no ambiente desejado, sendo então guiado até o objetivo.

Como ilustrado na Figura 9, a ideia então é que a rota gerada para alcançar determinado ponto no espaço seja determinada pela força resultante proveniente do campo potencial nesta determinada configuração de ambiente.

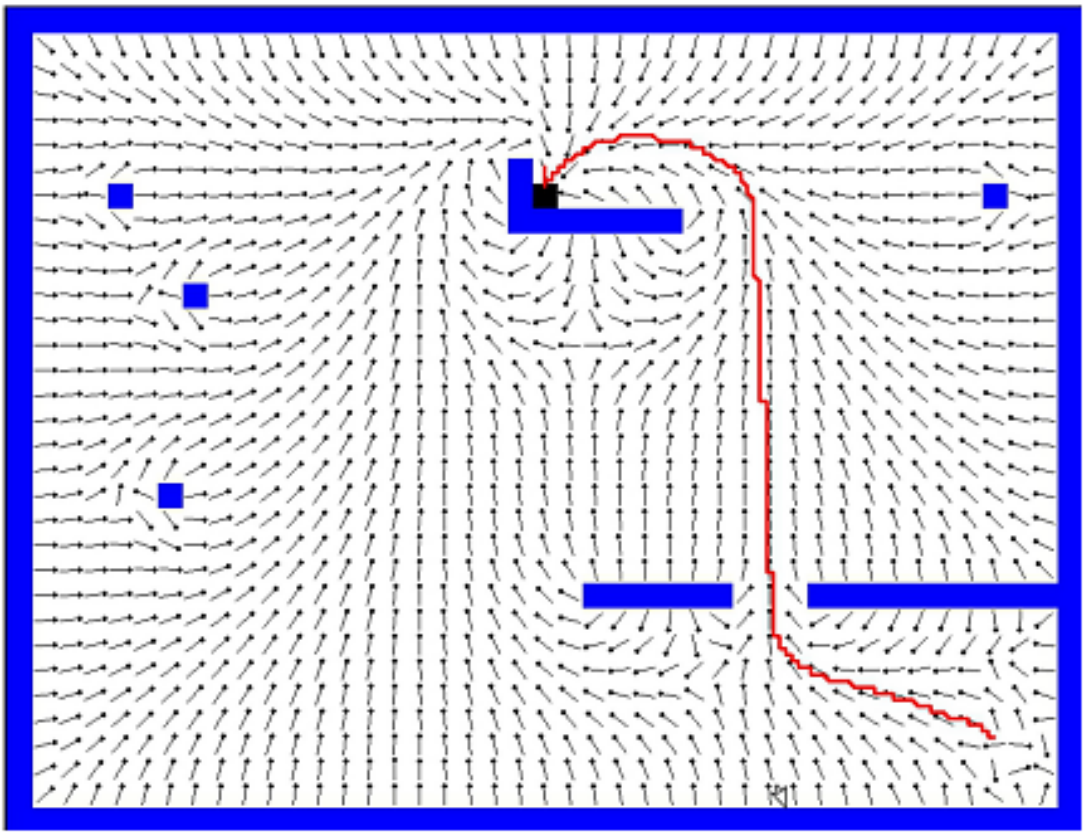

Figura 9 – Exemplo da aplicação do método de Campos Potenciais para planejamento de rota.

Fonte: Ubiratan de M. Pinto Jr., Maria P. Carvalho e Andre G. S. Conceição (2020)

2.3.1.2 Decomposição em células

Os métodos baseados em decomposição em células consistem em utilizar conjuntos de células não sobrepostas para representar a conectividade do espaço livre no ambiente. Em outras palavras, é criado um grafo, chamado de grafo de conectividade, que representa a relação de adjacência entre as células da região livre (CHATILA, 1982), como ilustrado na Figura 10. Segundo Chatila (1982) e Šeda (2007), a principal diferença entre os métodos baseados em decomposição de célula e os outros métodos baseados em grafos está justamente na forma em que os grafos são construídos.

Enquanto os mapa de rotas, que serão descritos na subseção 2.3.1.3, foca na criação de arestas que conectam os diferentes nós do grafo, no método de decomposição em células é dada uma maior importância aos nós do grafo que representam regiões inteiras no espaço. Dessa forma, fazendo com que as arestas que conectam um nó a outro sejam definidas trivialmente por uma conexão direta entre células vizinhas.

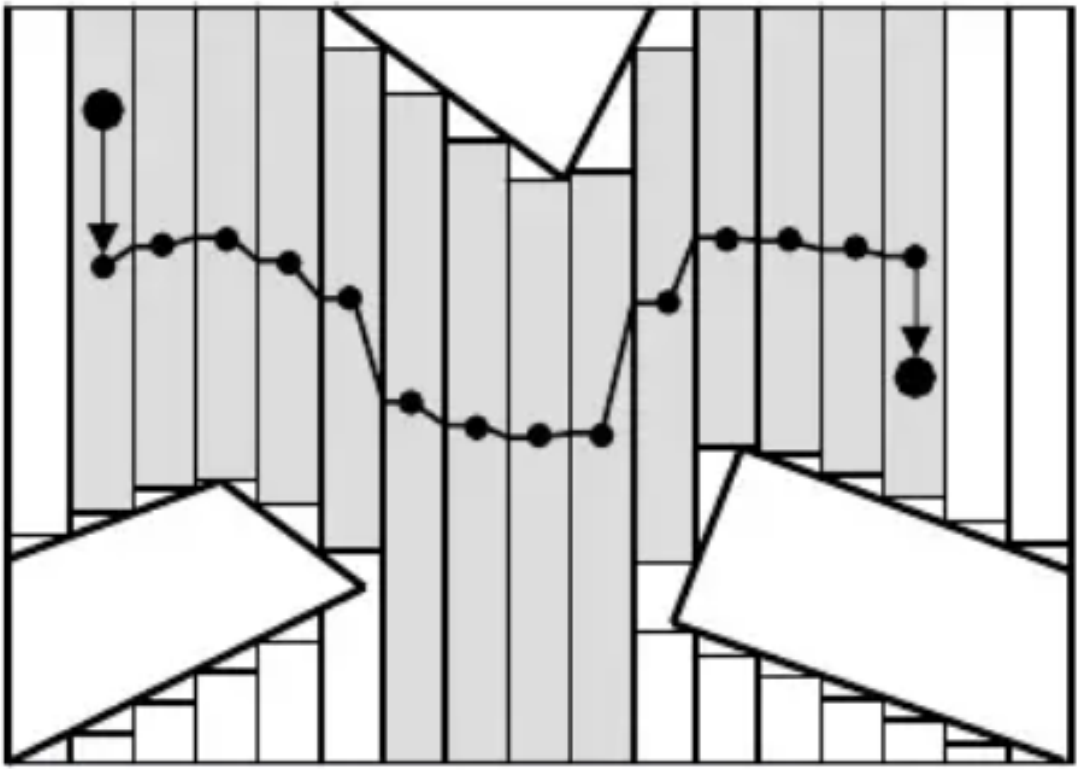

Figura 10 – Decomposição de célula e grafo resultante para exemplo com 3 obstáculos poligonais.

Fonte: Šeda (2007)

2.3.1.3 Mapas de rotas

Os métodos baseados em mapas de rotas, assim como o método citado anteriormente (decomposição em células), têm como princípio básico a criação de um grafo não direcionado que consiga representar a conectividade entre as diferentes regiões dentro do ambiente de interesse (LATOMBE, 1991). Assim que este grafo for obtido, ele pode ser considerado como um vasto conjunto de possibilidades de caminhos. O planejamento de rota reduz-se então a conectar as posições iniciais e finais do desejadas no mapa de rotas e buscar neste um caminho entre estes dois pontos.

Existem diferentes técnicas de planejamento que utilizam, como forma de representação do ambiente o mapa de rotas, as duas principais são o grafo de visibilidade (NASCIMENTO et al., 2020) (LATOMBE, 1991) e diagrama de Voronoi (AURENHAMMER, 1991) (CORKE, 2011).

No método de grafo de visibilidade, inicialmente são inseridos os nós da posição inicial e de destino. Os obstáculos são representados por polígonos e em cada vértice desses polígonos é colocado um nó do grafo. Após inserção de todos os nós no ambiente, são geradas as arestas com todas as retas que podem conectar dois nós. Se existir algum caminho entre as configurações inicial e destino, o menor caminho possível será constituído por um subconjunto destas arestas geradas. Após construído o grafo de visibilidade, pode-se utilizar

técnicas para escolha de um caminho viável, sendo ótimo ou não. Sua execução está ilustrada na Figura 11.

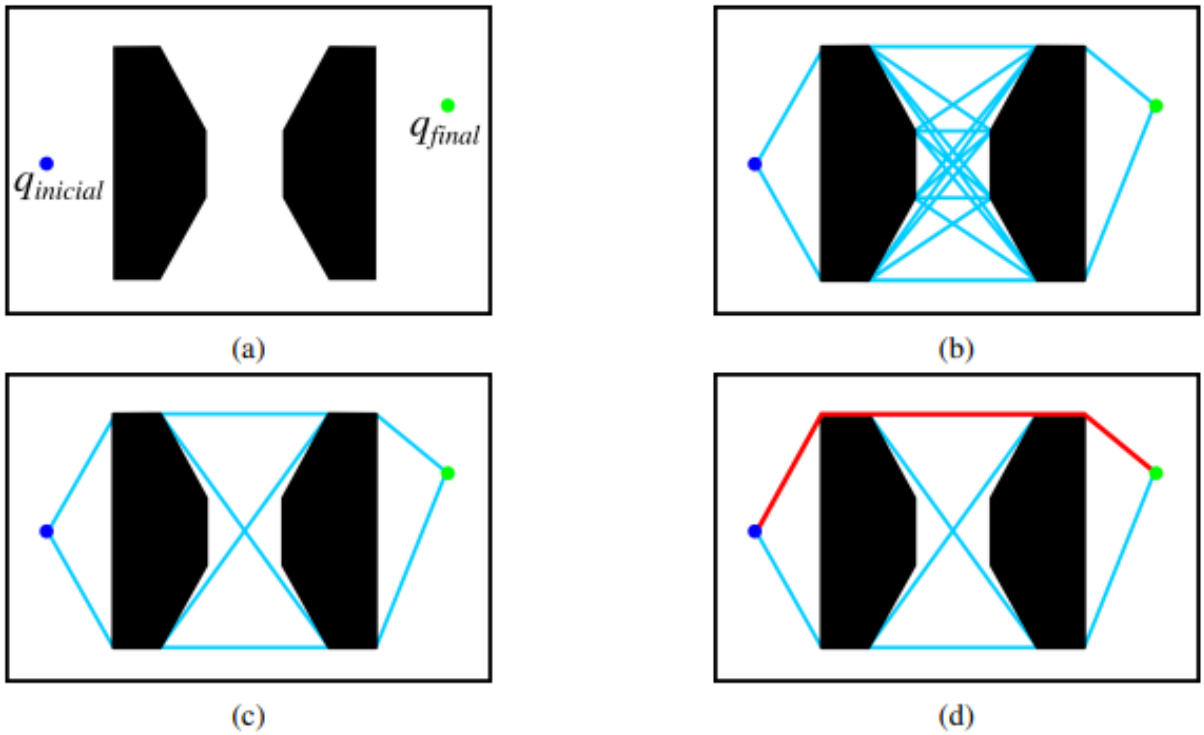

Figura 11 – Planejamento com Grafo de Visibilidade. (a) Mapa no espaço de configuração. (b) Arestas do Grafo de visibilidade. (c) Arestas do Grafo de visibilidade reduzido. (d) Caminho livre de obstáculos encontrado.

Fonte: Nascimento et al. (2020)

Apesar de conseguir, no geral, alcançar um resultado factível, esse método pode enfrentar certos problemas dependendo do ambiente em que está sendo trabalhado. O primeiro problema está associado ao número de obstáculos no mapa, sendo que, para ambientes com muitos obstáculos, esse método pode apresentar um tempo de execução alto e resultados ineficientes, pois à medida que cresce o número de obstáculos cresce também o número de nós no grafo (NASCIMENTO et al., 2020). O outro problema está justamente na aplicação para navegação de um robô, uma vez que o grafo resultante pode ser gerado apresentando uma trajetória ótima no sentido de menor caminho, o grafo de visibilidade pode levar o robô a navegar muito próximo dos obstáculos, o que não é seguro para aplicações práticas. Em casos onde a questão de segurança é mais importante, uma boa alternativa é a utilização do diagrama de Voronoi.

O diagrama de Voronoi é definido como um conjunto de pontos discretos em um plano, em que o espaço é dividido em regiões mais próximas de cada ponto (AURENHAMMER, 1991) (CORKE, 2013). Nesta técnica, ilustrada na Figura 12, o objetivo principal é maximizar a distância entre o caminho a ser traçado e os obstáculos proporcionando, assim,

uma certa margem de segurança inerente no algoritmo para o dispositivo/robô. No geral, o diagrama de Voronoi resulta em caminhos mais longos porém são caminhos mais factíveis.

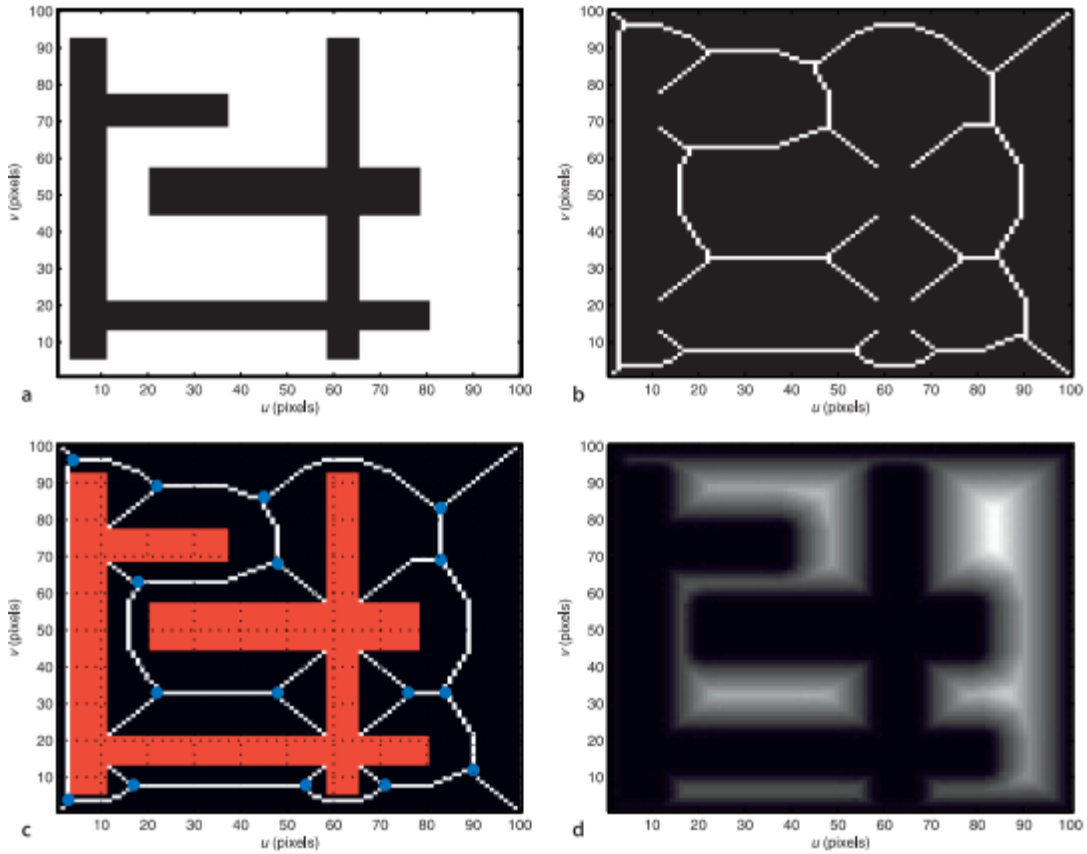

Figura 12 – Etapas na criação de um planejamento de rota a partir de diagrama de Voronoi. **a**: Espaço livre é indicado por células brancas; **b**: O "esqueleto"do espaço livre é uma rede de células adjacentes (máximo de distancia de 1 célula); **c**: O "esqueleto"com o obstáculos sobrepostos em vermelho e pontos de junção da rota indicados em azul; **d**: Representação da distancia até um obstaculo a partir dos valores de tom entre preto e branco em cada pixel.

Fonte: Corke (2011)

A partir dos estudos realizados por Latombe (1991), Aurenhammer (1991), Spong, Hutchinson e Vidyasagar (2006) e Corke (2013), pode-se pontuar que o diagrama de Voronoi tem o mesmo problema do grafo de visibilidade no que se refere número de obstáculos no ambiente, aumentando assim a dimensão do espaço de configuração. Nesses casos, pode ser inviável, ou até mesmo impossível, a obtenção dos grafos de forma determinística.

Com essa situação, indica-se a utilização de métodos probabilísticos para realizar a escolha dos pontos no ambiente que irão representar os nós do grafo. A partir da bibliografia utilizada, têm-se que os maiores representantes desses métodos são o PRM, sigla em inglês para Mapa de Rotas Probabilísticas (*Probabilistic Roadmaps*), e o RRT, sigla em inglês para Árvore Aleatória de Exploração Rápida (*Rapidly-exploring Random Tree*). Esses métodos probabilísticos, também chamados como de amostragem, serão apresentadas na

subseção 2.3.2, subseção 2.3.3 e subseção 2.3.4.

2.3.2 Métodos probabilísticos

Uma das formas de descrever o movimento desejado para o manipulador robótico é em função de variáveis cartesianas, ou seja, uma sequencia de pontos que fornecem a posição e orientação do efetuador em função do tempo. Nesse tipo de abordagem, algumas complicações podem ser geradas no planejamento no espaço devido as singularidades dos mecanismos, resultante da não-inversibilidade da matriz jacobiana que representa as transformações lineares correspondentes a velocidade da junta com a velocidade cartesiana (TONETTO, 2007), mas, tais problemas podem ser reparados a partir de técnicas computacionais para validação de configurações das juntas.

Ao longo do tempo, métodos probabilísticos, como PRM, RRT e RRT-Connect, demonstraram-se eficientes para a tarefa de planejamento de rota. Aliada ao poder computacional das tecnologias atuais, os algoritmos baseados em um grande conjunto de amostragem, juntamente com a lógica para verificação de colisão, fornece as informações necessárias para construção de um caminho válido almejado. Mesmo analiticamente podendo não ser a forma mais eficiente para tal finalidade, suas características probabilísticas garantem que pelo menos 1 caminho válido será encontrado, uma vez que o número de amostras geradas se aproxima do infinito (KLEINBORT et al., 2019).

O PRM, sigla em inglês para Mapa de Rotas Probabilístico, foi um dos primeiros algoritmos para planejamento de caminhos entre dois pontos e também foi um dos primeiros algoritmos baseados na amostragem do espaço de configurações a ser viável na prática (HSU; LATOMBE; KURNIAWATI, 2006) (KAVRAKI et al., 1996).

A ideia por trás do PRM é coletar amostras aleatórias dentro do ambiente em que está inserido, verificando se estão no espaço livre e usando um planejador local para tentar conectar essas amostras a outras amostras próximas. Os pontos de início e objetivo são adicionadas e um algoritmo de pesquisa é aplicado ao grafo resultante para determinar um caminho entre as configurações de início e objetivo, como ilustrado na Figura 13.

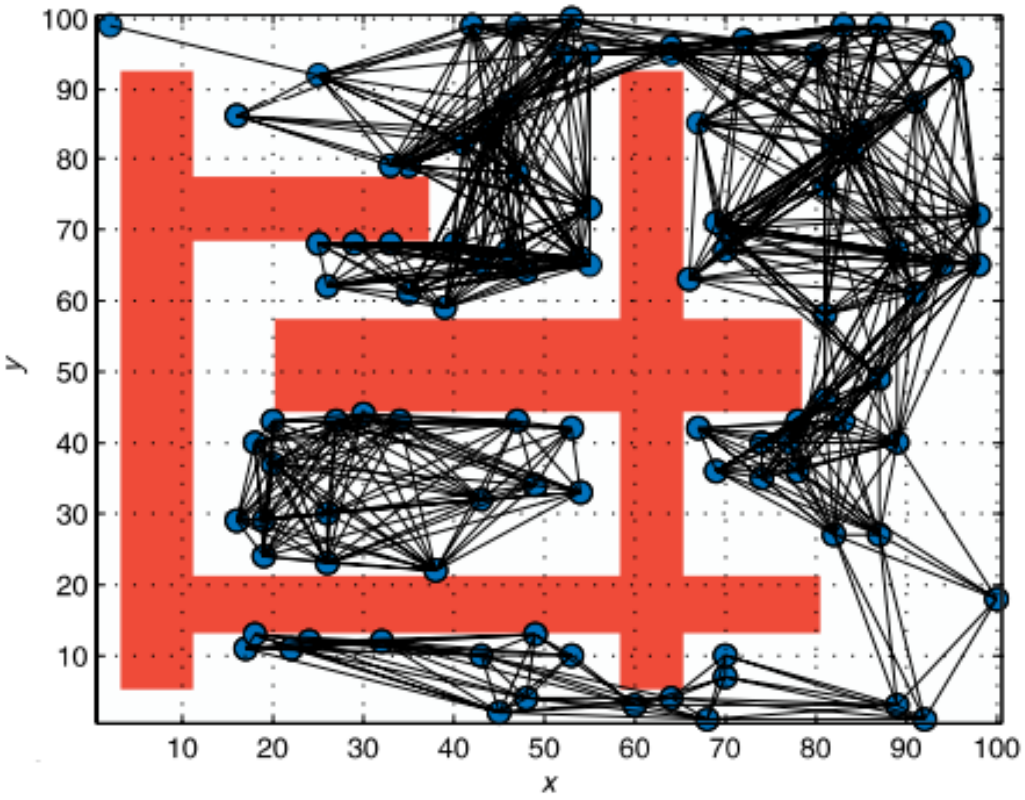

Figura 13 – Exemplo do Mapa de Rotas Probabilístico (PRM) utilizado para planejamento de rota.

Fonte: Corke (2011)

Pode-se concluir que o algoritmo PRM é, de fato, muito útil e funcional, mas não fornece a solução ideal todas as vezes, principalmente quando é aplicado à um ambiente com um grande número de obstáculos situados muito próximos uns dos outros (LATOMBE, 1991) (HSU; LATOMBE; KURNIAWATI, 2006). Devido a estratégia de amostragem de nós aleatórios, a probabilidade de geração de nós entre obstáculos próximos é muito pequena.

Intuitivamente, poderia ser utilizado como estratégia aumentar o número de iterações para aumentar as chances de amostrar nós em locais menos prováveis, mas, ainda sim, não seria alcançado resultados conclusivos na generalização da solução. Nessa situação, uma vez que o sistema não consiga gerar um caminho para tal espaço, não seria possível saber se o motivo é a não existência de um caminho hábil ou o número de iterações é muito pequeno para ambiente.

Alternativamente ao PRM, o algoritmo *Rapidly-exploring Random Tree* (RRT), citado anteriormente, é um dos mais populares na academia e em aplicações no mercado, conseguindo superar as desvantagens citadas para o PRM com a construção de árvores ao invés de grafos. Sua versão original e a variação bidirecional(*RRT-Connect*) serão utilizadas nesse trabalho e, por isso, serão abordados de forma mais completa na subseção 2.3.3 e subseção 2.3.4.

2.3.3 Algoritmo Rapidly-exploring Random Tree (RRT)

O algoritmo *Rapidly-exploring Random Tree* (RRT), proposto por Steven M. LaValle (LAVALLE, 1998; LAVALLE, S. M., 2006) e com contribuições de J. J. Kuffner (LAVALLE; KUFFNER, J. J., 2000), é definido pela construção de uma árvore de caminhos válidos entre duas posições no ambiente desejado. Pontos aleatórios no espaço são utilizados como tentativa de caminho, o nó mais próximo do referido ponto é encontrado e, se os pontos não tiverem colisão, eles serão conectados. Na Figura 14 pode-se visualizar a evolução da árvore de caminhos em diferentes momentos ao longo das iterações do RRT, demonstrando bem a efetividade do algoritmo para descoberta de espaços em um ambiente.

Figura 14 – Exemplo da evolução de uma árvore RRT.

Fonte: S. M. LaValle (2006)

2.3.3.1 Lógica de execução

Como descrito no algoritmo 1, a partir do espaço de trabalho propiciado para a realização da busca pelo caminho, juntamente com os pontos de inicio q_{init} e de destino q_{goal}, o algoritmo, essencialmente, realiza tentativas de encontrar um caminho entre os dois pontos no espaço, como ilustrado na Figura 15.

Com a árvore tendo inicio no ponto q_{init} informado, são realizadas K tentativas de expansão da árvore pelo espaço adicionando um novo nó. Para isso, a cada iteração, um novo ponto q_{rand} é gerado e, em seguida, o algoritmo encontra algum nó, q_{near}, mais próximo de q_{rand} que já esteja presente na árvore. Uma vez que o nó gerado não está dentro ou muito próximo de um obstáculo e não possua obstáculos entre ele e o nó mais próximo, esse nó gerado será adicionado à árvore.

Algorithm 1 RRT (LAVALLE, S. M., 2006)

Entrada:
q_{init}: Configuração Inicial
Δq: Distância
K: Quantidade de vértices
Saída:
Arvore RRT(G)

function RRT(q_{init},K,Δq)
  $G.init(q_{init})$
  for $k = 1 : K$ **do**
    $q_{rand} \leftarrow RAND\_POSITION()$
    $q_{near} \leftarrow NEAREST\_VERTEX(q_{rand},G)$
    $q_{new} \leftarrow NEW\_POSITION(q_{near},q_{rand},\Delta q)$
    if q_{new} é livre de colisões **then**
      $G.add\_vertex(q_{new})$
      $G.add\_edge(q_{near},q_{new})$
    end if
  end for
  return G
end function

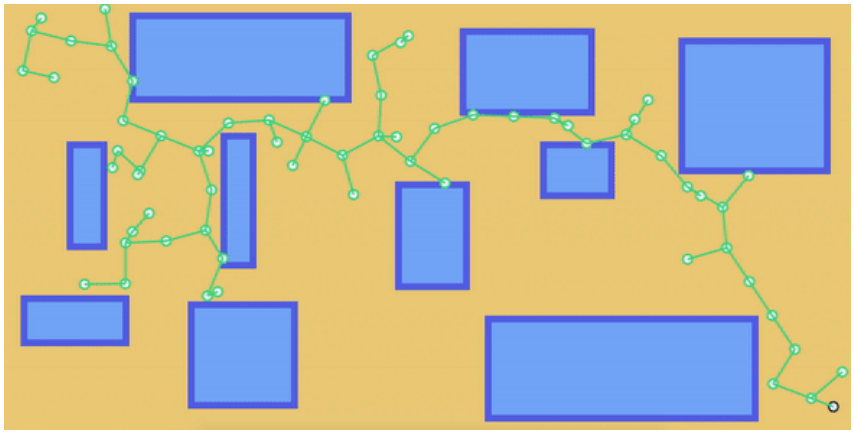

Figura 15 – Representação da busca pela posição final desejada utilizando RRT.

Fonte: Kaufer (2017)

Para determinar se o q_{goal} foi alcançado, pode-se considerar uma região ao redor do ponto destino como zona tolerável para ter-se como acerto pela árvore gerada, finalizando o processo de expansão.

Por fim, normalmente, após encontrado o caminho para a posição final dada como objetivo, o caminho é suavizado a partir de uma interpolação. Existem muitos métodos que podem ser usados para encontrar ou interpolar os pontos gerando trajetórias das posições ou orientações do robô. Isso inclui a interpolação por segmentos de retas, curvas polinomiais, curvas cubicas (por exemplo, splines cubicas de Hermite, curvas de Bezier, B-Splines, B-Splines racional n~ao uniforme)(ZEID, 1991).

2.3.3.2 Propriedades e vantagens

Baseado nos estudos apresentados em (LAVALLE, S. M., 2006), (LAVALLE; KUFFNER, J. J., 2000) e (CORKE, 2013), as principais particularidades e vantagens do algoritmo RRT, além das já citadas, são:

- Pode ser considerado um algoritmo probabilisticamente completo em condições gerais;
- Em essência, possui uma estrutura simples;
- Possibilidade de implementação de extensões para melhor aplicação em situações específicas;
- A expansão da árvore é fortemente tendenciosa para porções inexploradas do mapa;
- Uma RRT pode ser considerada um módulo de planejamento de caminho, aumentando sua aplicabilidade numa variedade de sistemas de planejamento.

Após a publicação do algoritmo RRT, foram desenvolvidas diversas variações do algoritmo com o intuito de alcançar melhores resultados em diferentes situações e ambiente, como o RRT-Connect que será descrito na subseção 2.3.4.

2.3.4 Algoritmo Rapidly-exploring Random Tree Connect (RRT-Connect)

Desenvolvido a partir das contribuições de J. J. Kuffner (LAVALLE; KUFFNER, J. J., 2000), o algoritmo *Rapidly-exploring Random Tree - Connect* (RRT-Connect) é definido pela descoberta de um caminho válido de forma bidirecional, construindo árvores a partir do ponto de inicio e ponto de destino.

Na Figura 16 pode-se visualizar a evolução da árvore de caminhos em diferentes momentos ao longo das iterações do RRT, demonstrando bem a efetividade do algoritmo para descoberta de espaços em um ambiente.

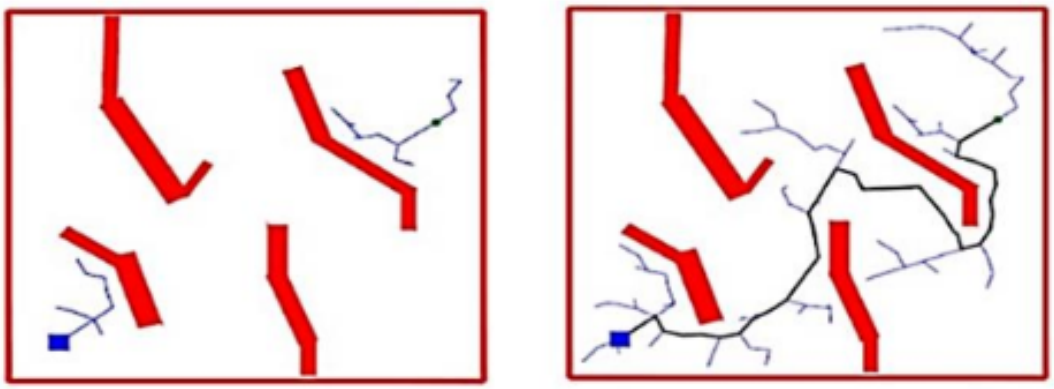

Figura 16 – Representação da busca pelo caminho entre posição inicial e final desejada utilizando RRT-Connect. Inicio da expansão de cada árvore e momento em que ocorre a conexão, respectivamente.

Fonte: Noreen, Khan e Habib (2016)

2.3.4.1 Lógica de execução

Similar ao RRT original, descrito na subseção 2.3.3, o RRT-Connect, representado no algoritmo 2, segue praticamente a mesma conceituação para encontrar um caminho válido entre dois pontos, mas, com a diferença de criar árvores a partir do ponto de inicio (chamaremos de árvore-inicio) e, também a partir do ponto de destino (árvore-destino). Os nós em cada árvore são geradas alternadamente até que ambas árvores estejam conectadas. Por fim, através dessa conexão entre as duas árvores, o algoritmo traça o menor caminho possível, no momento, para ligar o ponto de inicio e destino através das árvores geradas (KUFFNER, J.; LAVALLE, S., 2000).

Algorithm 2 RRT-Connect (LAVALLE; KUFFNER, J. J., 2000)

Entrada:
T_{init}: Árvore com origem no ponto de inicio
T_{goal}: Árvore com origem no ponto de destino
Saída:
Arvore RRT(G)

1: **function** RRT-CONNECT(q_{init},K,Δq)
2: $G.init(q_{init})$
3: **for** $k = 1 : K$ **do**
4: $q_{rand} \leftarrow RAND\_POSITION()$
5: $q_1 \leftarrow Extend - Tree(T_{init},K,\Delta q)$
6: $q_2 \leftarrow Extend - Tree(T_{goal},K,\Delta q)$
7: **if** $d(q_1,q_2)$ é menor que o limite determinado e livre de colisões **then**
8: $Adicionaconex\cap oentreq_1eq_2paraconectarT_{init}eT_{goal}$
9: $G.add\_edge(q_{near},q_{new})$
10: **return** G
11: **end if**
12: **end for**
13: **end function**

2.3.4.2 Propriedades e vantagens

Assim como diversos outros algoritmos originados a partir da publicação das primeiras conceituações do RRT por S. M. LaValle (2006), o RRT-Connect se apegou a ideia de otimização dos resultados já alcançados primeiramente, sendo uma boa alternativa para aplicação em situações onde a principal prioridade está e descobrir um caminho o mais rápido possível.

Quando comparado a tentativas de expansão de apenas uma única árvore RRT, o método proposto para a expansão de árvores a partir do inicio e do fim, simultaneamente, demonstrou grandes ganhos em eficiência quando são aferidos questões de tempo de execução, principalmente (LAVALLE; KUFFNER, J. J., 2000).

Baseado nos estudos apresentados em (LAVALLE; KUFFNER, J. J., 2000), as principais particularidades e vantagens do algoritmo RRT-Connect está na capacidade de reproduzir quase que todas as vantagens do RRT padrão, como ser probabilisticamente completo e forte tendência de expansão para regiões ainda não exploradas e, ainda, aliar isso à grande diminuição no tempo de execução para que se consiga encontrar um caminho válido.

2.4 Considerações

Este capítulo teve com principal proposta apresentar algumas das inúmeras possibilidades para o desenvolvimento de aplicações para planejamento de rota, podendo ser aplicadas diferentes técnicas para inúmeras situações. As ferramentas apresentadas serão substancialmente utilizadas para a abordagem da proposta.

Entre os algoritmos existentes, o algoritmo *Rapidly-exploring Random Tree* (RRT) provou-se ser um dos mais eficientes quando se tem o objetivo de trazer uma rota de movimentação para evite colisão entre obstáculos, mantendo uma margem de proximidade arbitraria. O que legitima sua reputação de ser um dos mais populares na academia e em aplicações no mercado.

Como dito ao longo do capítulo, devido suas característica de natureza aleatória, possui uma excelente capacidade de exploração de ambientes e representa uma excelente técnica quando se deseja traçar caminhos entre diferentes regiões em um ambiente.

Portanto, devido a esses e outros motivos citados ao longo deste capítulo, sua versão original e a variação bidirecional (*RRT-Connect*) serão utilizadas nesse trabalho e estão melhor descritas no Capítulo 3. Realizando um paralelo direto entre as duas versões, será possível validar a efetividade de ambos algoritmos e ainda comparar quão mais eficiente é um em relação ao outro.

3 Desenvolvimento

3.1 Modelagem cinemática

3.1.1 Notação de Denavit-Hartenberg

Sabendo das especificações e medidas do robô, foi possível construir e analisar cada eixo atribuído à cada junta e, em seguida, verificar cada parâmetro a ser utilizado no desenvolvimento da notação de Denavit-Hartenberg, como ilustrado na figura 17.

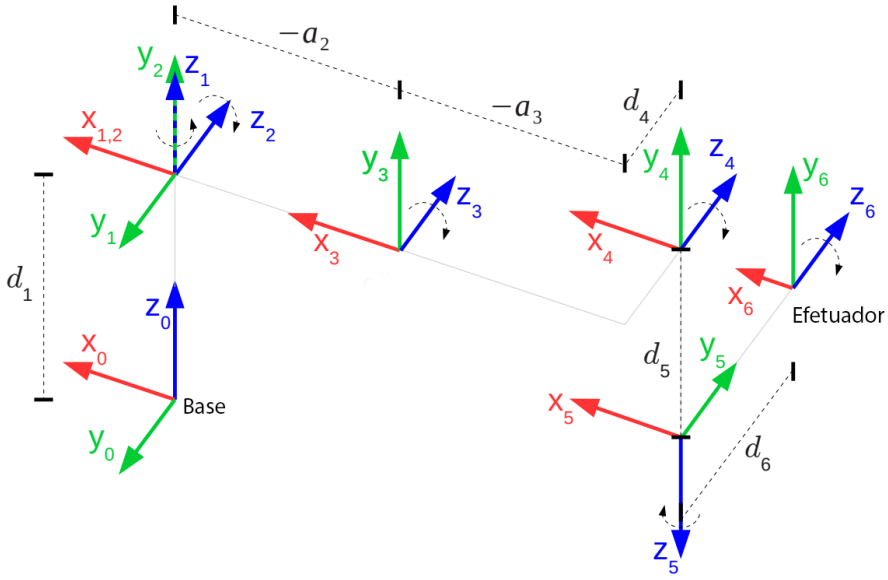

Figura 17 – Sistema cinemático do UR3

Percebendo que o UR3 possui apenas juntas de revolução, é possível verificar que os parâmetros d_i, a_i e α_i são fixos, enquanto que θ_i corresponde ao deslocamento angular das juntas produzido pelos atuadores, sendo assim, variável. A partir disso, foi possível desenvolver as matrizes de transformação, como descrito na equação 2.4, obtendo a cinemática direta do robô.

Sendo assim, foi encontrado os parâmetros de Denavit-Hartenberg, os quais foram representados na tabela 2.

Junta	θ (rad)	a (m)	d (m)	α (rad)	Massa (Kg)
Junta 1	$q1$	0	0,1519	$\pi/2$	2
Junta 2	$q2$	-0,24365	0	0	3,42
Junta 3	$q3$	-0,21325	0	0	1,26
Junta 4	$q4$	0	0,11235	$\pi/2$	0,8
Junta 5	$q5$	0	0,08535	-$\pi/2$	0,8
Junta 6	$q6$	0	0,0819	0	0,35

Tabela 2 – Parâmetros Denavit-Hartenberg do UR3

3.1.2 Implementação em MATLAB

A partir dos dados alcançados, foi possível realizar a validação dos resultados. Com auxílio da Robotics Toolbox (ROBOTICS..., 2017) no MATLAB, consegue-se criar um ambiente interativo de simulação visualizar em tempo real a movimentação do robô estudado. Como ilustrado na Figura 18, após a inserção dos parâmetros encontrados nas variáveis do ambiente do MATLAB, é possível gerar a visualização desejada e alterar as angulações de cada elo em tempo real, como ilustrado na Figura 19.

```
six link:: 6 axis, RRRRRR, stdDH, slowRNE
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|       0.15|          0|     1.5708|          0|
|  2|         q2|          0|      -0.24|          0|          0|
|  3|         q3|          0|      -0.21|          0|          0|
|  4|         q4|       0.11|          0|     1.5708|          0|
|  5|         q5|       0.08|          0|    -1.5708|          0|
|  6|         q6|       0.08|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+
```

Figura 18 – Parâmetros Denavit-Hartenberg a partir da Robotics Toolbox

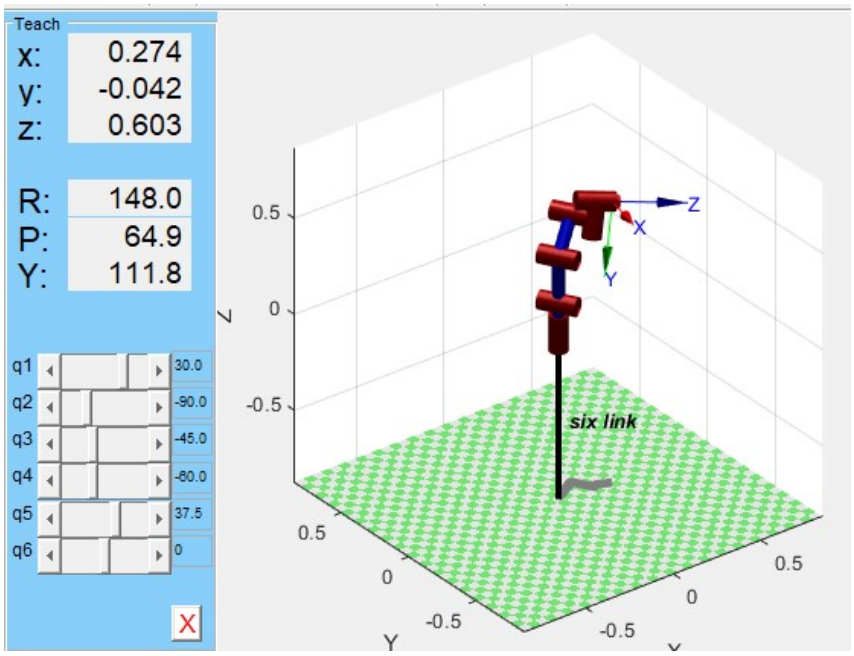

Figura 19 – Simulação gerada a partir da Robotics Toolbox

3.1.3 Implementação em Python

Como o objetivo é realizar a comunicação, planejamento e controle do robô diretamente por Python, o próximo passo adotado foi a implementação dos mesmos resultados, vistos na subseção anterior, em um ambiente Python. Sendo assim, utilizando a biblioteca *Robotics Toolbox para Python* (ROBOTICS..., 2022) , foi possível realizar as verificações do modelo do robô.

Assim como a biblioteca utilizada em MATLAB, a *Robotics Toolbox* realiza a leitura do modelo URDF do robô e consegue manipular as informações de forma a calcular parâmetros de cinemática para o UR3.

Como ilustrado na Figura 20, o modelo é representado seguindo os parâmetros calculados na subseção 3.1.1 e, também, correspondem aos encontrados no MATLAB.

```
UR3 = models.DH.UR3()
```

✓ 0.6s

```
DHRobot: UR3 (by Universal Robotics), 6 joints (RRRRRR), dynamics, standard DH parameters
┌──────┬──────────┬──────────┬──────────┐
│θⱼ    │   dⱼ     │   aⱼ     │   αⱼ     │
├──────┼──────────┼──────────┼──────────┤
│ q1   │  0.1519  │        0 │  90.0°   │
│ q2   │        0 │  -0.2437 │   0.0°   │
│ q3   │        0 │  -0.2132 │   0.0°   │
│ q4   │  0.1124  │        0 │  90.0°   │
│ q5   │ 0.08535  │        0 │ -90.0°   │
│ q6   │  0.0819  │        0 │   0.0°   │
└──────┴──────────┴──────────┴──────────┘
```

Figura 20 – Parâmetros Denavit-Hartenberg a partir da Robotics Toolbox para Python

Utilizando as funções de cálculo de cinemática direta ou inversa, é possível alcançar todos os parâmetros das juntas a ponto de simular/ilustrar o posicionamento do robô na configuração desejada. Informando cada valor dos parâmetros das juntas, como ilustrado na Figura 21, pode-se determinar a configuração desejada, seguido das transformações de cinemática. Por fim, consegue-se verificar o resultado visualmente, como ilustrado na Figura 22.

```
Config = [180,0,0,0,90,0] #Graus
Config = [180*pi/180,0*pi/180,0*pi/180,0*pi/180,90*pi/180,0*pi/180] #Radianos
```

```
Config_FK = UR3.fkine(Config)
Config_IK = UR3.ikine_LM(Config_FK)
```

```
UR3.plot(Config_IK.q, block=False,movie='ur3.gif');
```

Figura 21 – Parâmetros das juntas e transformações de cinemática.

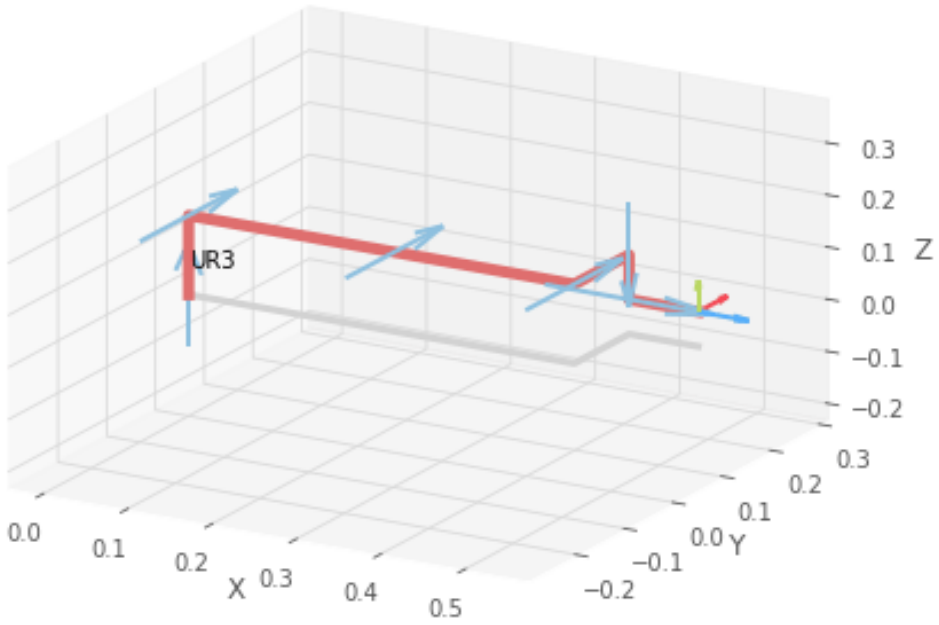

Figura 22 – Representação gráfica da configuração informada.

3.2 Planejamento de trajetória

Para implementação do planejamento de trajetória utilizando o algoritmo RRT, foi realizado, primeiramente, uma revisão bibliográfica por meio de execução dos modelos de planejamento de rota já projetados dentro do MATLAB e documentados no livro Corke (2013); em seguida, com o conhecimento melhor consolidado sobre o assunto, foi realizado as implementações utilizando Python, ROS e o simulador CoppeliaSim.

3.2.1 Simulação no MATLAB

Seguindo os modelos presentes no MATLAB, fornecidos, principalmente, pela *Robotics Toolbox* (CORKE, 2011), pode-se compreender melhor como são projetados.

Por se tratar de ferramentas que já estão populares entre pesquisadores da área e com resultados consolidados , foi utilizado todas as implementações de planejamento de rota como motivação e inspiração para o desenvolvimento desse trabalho. Mas, por razões obvias e por não se tratar de implementações de código aberto (*opensource*), nada além da conceituação do método é compartilhado, restringindo informações mais profundas de como o algoritmo funciona.

Primeiramente, seguindo os modelos e exemplos de implementação, foi possível entender ainda melhor como é esperado que a execução dos algoritmos se comportem nas situações de ambiente 2D e 3D, ilustrado nas figuras 23 e 24.

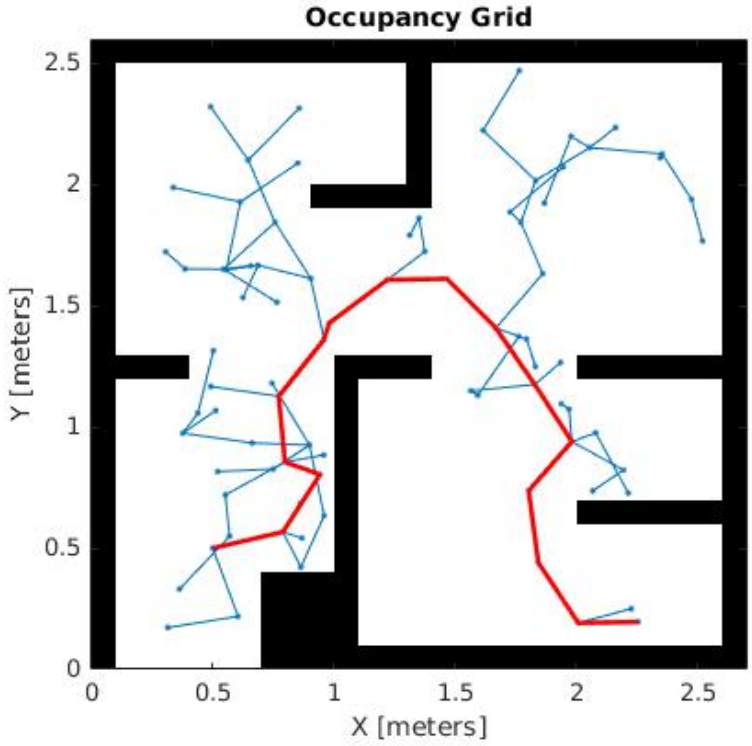

Figura 23 – Exemplo de implementação do RRT em plano 2D no MATLAB.

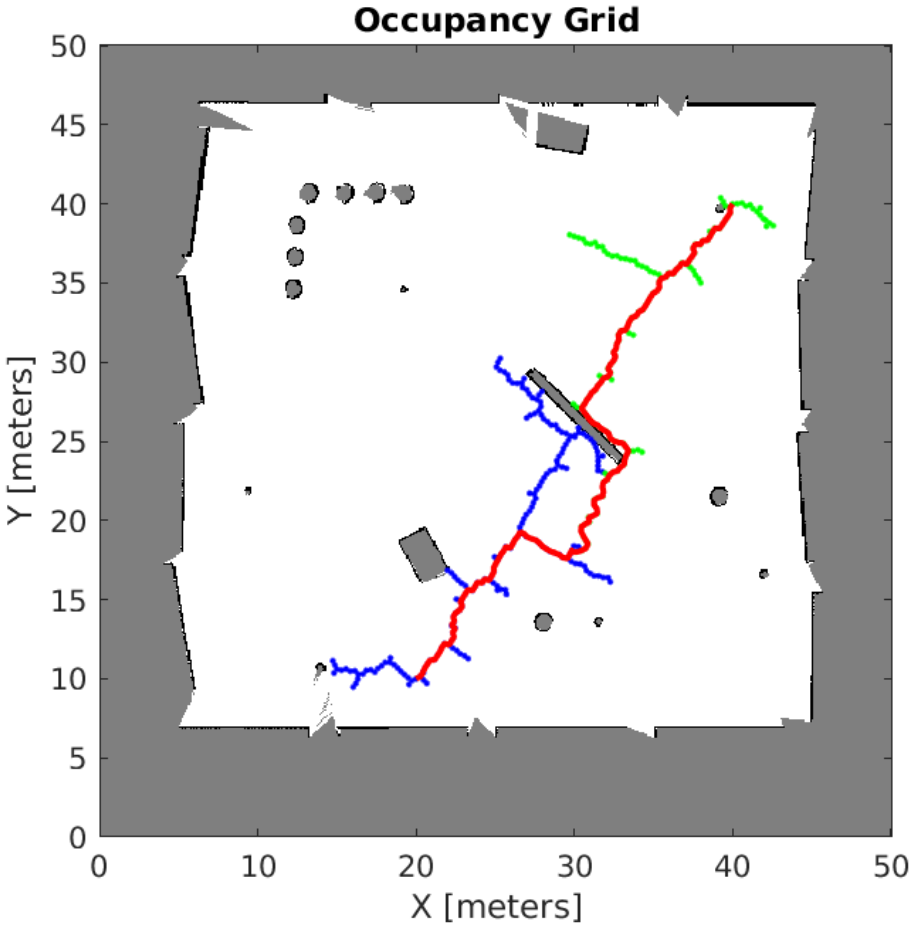

Figura 24 – Exemplo de implementação do RRT-Connect em plano 2D no MATLAB.

Em seguida, foi visto as implementações com cunho mais direcionado para o intuito desse trabalho. O exemplo visto na figura 25 representa bem o que pode-se alcançar em nível de simulação para a resolução do problema de planejamento de rotas.

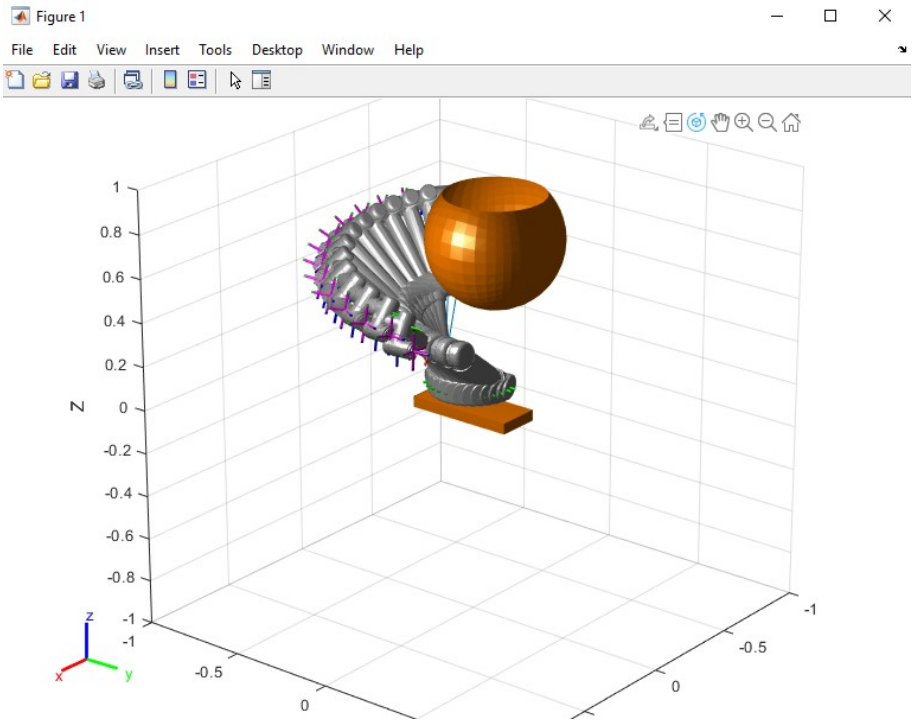

Figura 25 – Movimentação do UR3 utilizando RRT no MATLAB.

3.2.2 Implementação em Python

Para implementação em Python, foi utilizado a estratégia de desenvolvimento do algoritmo progressivamente até alcançar uma representação útil em um ambiente 3D de simulação.

Primeiramente, foi empregado a definição dos algoritmos para aplicação em um ambiente 2D. Apesar de não ser exatamente a aplicação desejada, é de extrema importância para uma consolidação do conhecimento desse algoritmo. Após essa etapa, por fim, foi adaptado o mesmo algoritmo para um ambiente de trabalho 3D, alcançando a finalidade desejada.

3.2.2.1 RRT

3.2.2.1.1 Implementação bidimensional

Utilizando as definições descritas na subseção 2.3.3, foi desenvolvido o algoritmo que representa o comportamento conceituado por Lavalle (1998).

Assim como apresentado anteriormente, o algoritmo recebe os pontos inicial e destino e, considerando os obstáculos, constrói uma árvore de pontos afim de formar um caminho até o ponto final.

Com a função *expand*(), demonstrada na Figura 26, o algoritmo consegue gerar o novo ponto randomicamente, adiciona ao espaço de trabalho, checa colisão com obstáculos e procura e conecta com ponto já existente e mais próximo.

```
# Expande a árvore
def expand(self):
    # Gera novo ponto randomicamente
    x = random.uniform(E.xmin, E.xmax)
    y = random.uniform(E.ymin, E.ymax)
    n = self.number_of_nodes()  # new node number
    self.add_node(n, x, y)
    if E.isfree() != 0:
        # Procura ponto já existente mais próximo
        nnear = self.near(n)
        self.step(nnear, n)
        # Conecta os dois pontos com uma aresta
        self.connect(nnear, n)
```

Figura 26 – Função de expansão da árvore de caminhos do RRT.

Com a função *checkCollision*(), demonstrada na Figura 27, é checada a colisão de algum ponto para a construção do caminho escolhido. É feito a comparação ponto a ponto com cada um dos obstáculos presentes no ambiente. Como garantia de um melhor funcionamento, visando a aplicabilidade com o robô UR3, foi adicionado um fator de segurança nas dimensões dos obstáculos. Dessa forma, consegue-se garantir que mesmo que as dimensões reais do manipulador, não haverá qualquer colisão com quinas ou qualquer outro tipo de superfície.

```
# Checa colisão
    def checkCollision(self, x1, y1, x2, y2):
        c = True # Default: Há colisão
        numObs = len(self.x)/4  # Número de obstaculos
        for i in range(1, int(numObs)+1):
            #Define posições minimas e maximas para cada ponto
            #Considera-se um offset de segurança para lidar com dimensoes reais do manipulador
            xo_min = self.x[4*(i-1)] - security_offset
            xo_max = self.x[4*(i-1)+2] + security_offset
            yo_min = self.y[4*(i-1)] - security_offset
            yo_max = self.y[4*(i-1)+1] + security_offset
            for j in range(0, 101):
                u = j/100.0
                x = x1*u+x2*(1-u)
                y = y1*u+y2*(1-u)
                if (x >= xo_min) and (x <= xo_max) and (y >= yo_min) and (y <= yo_max): #Checa colisão comparando ponto de entrada
                    c = False                                                           #coordenadas dos obstaculos
                    break
            if c == False:
                break
        return c
```

Figura 27 – Função que checa colisão dos caminhos com obstáculos do espaço de trabalho.

Como ilustrado na Figura 28 e Figura 29, o resultado foi como esperado. As árvores partiram do ponto estipulado e alcançaram o destino com sucesso. É importante pontuar como é possível notar diferenças nas árvores de cada execução realizada Percebe-se, também, que as árvores geradas não ultrapassaram os obstáculos determinados e o caminho escolhido não extrapolou o fator de segurança definido, não aproximando-se demais dos obstáculos.

Além disso, é possível verificar a quantidade de iterações que o algoritmo precisou realizar até alcançar o resultado representado. Em ambas execuções, foram necessárias 551 iterações, mas, como poderá ser visto na resultados seguintes, o valor pode variar consideravelmente, ainda mais por conta das propriedades de aleatoriedade que os dois algoritmos abordados possuem. Sendo assim, a cada execução realizada, há grandes chances de que o número de iterações seja diferente, influenciando diretamente em como a árvore estará estruturada visualmente.

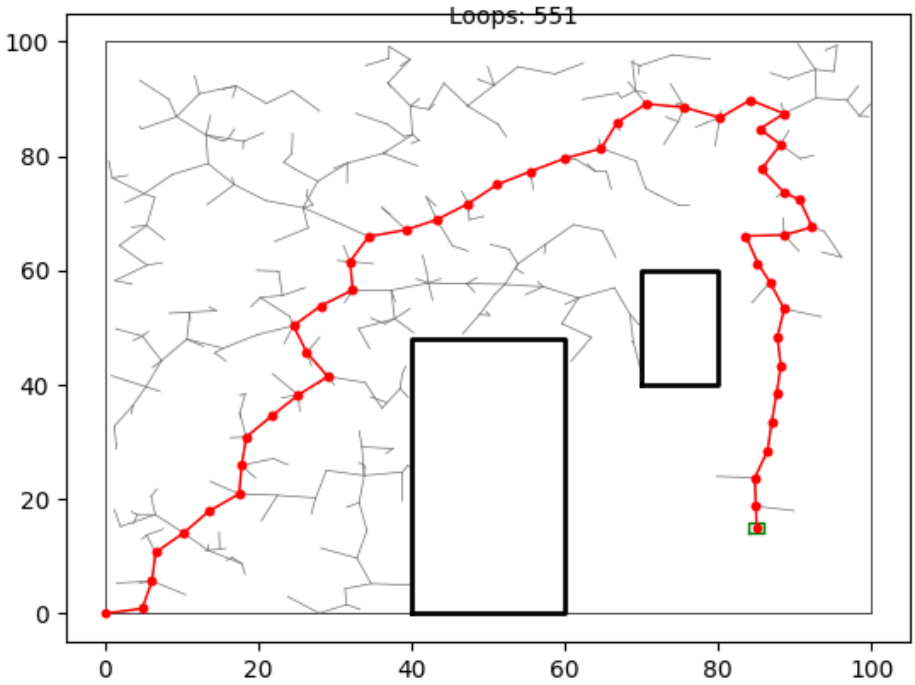

Figura 28 – Planejamento de rota 2D utilizando o RRT em Python - Execução 1. 551 iterações.

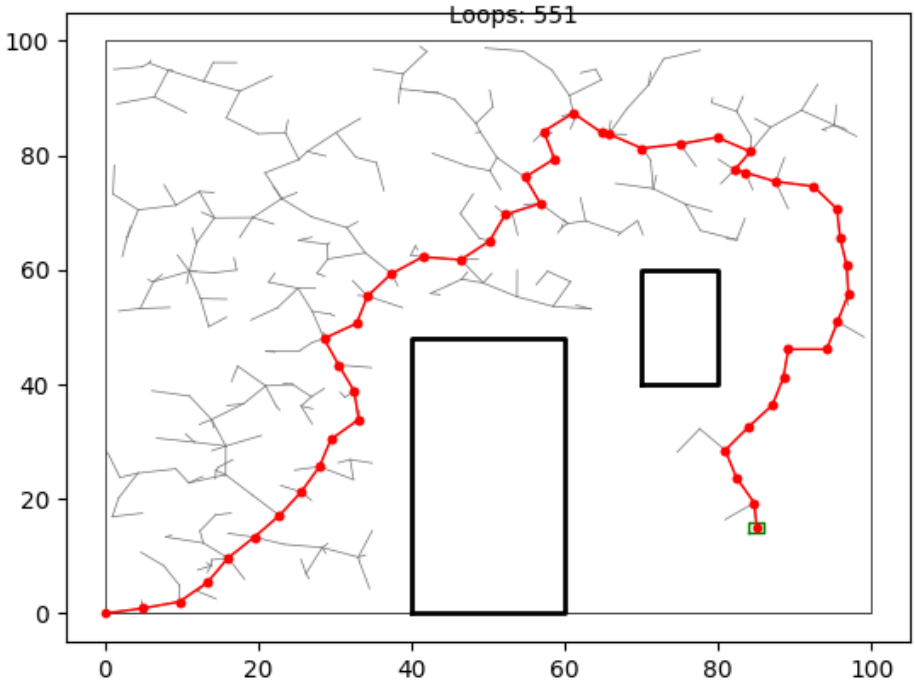

Figura 29 – Planejamento de rota 2D utilizando o RRT em Python - Execução 2. 551 iterações.

Após encontrar o caminho até o ponto de destino, também foi implementado uma suavização desse trajeto. Cada um dos nós selecionados dentro desse caminho é comparado e, caso uma linha possa ser desenhada entre esses nós sem colisão, os nós intermediários são desconsiderados para simplificar o caminho a ser percorrido. Nas figuras 30 e 31 pode-se visualizar os resultados obtidos.

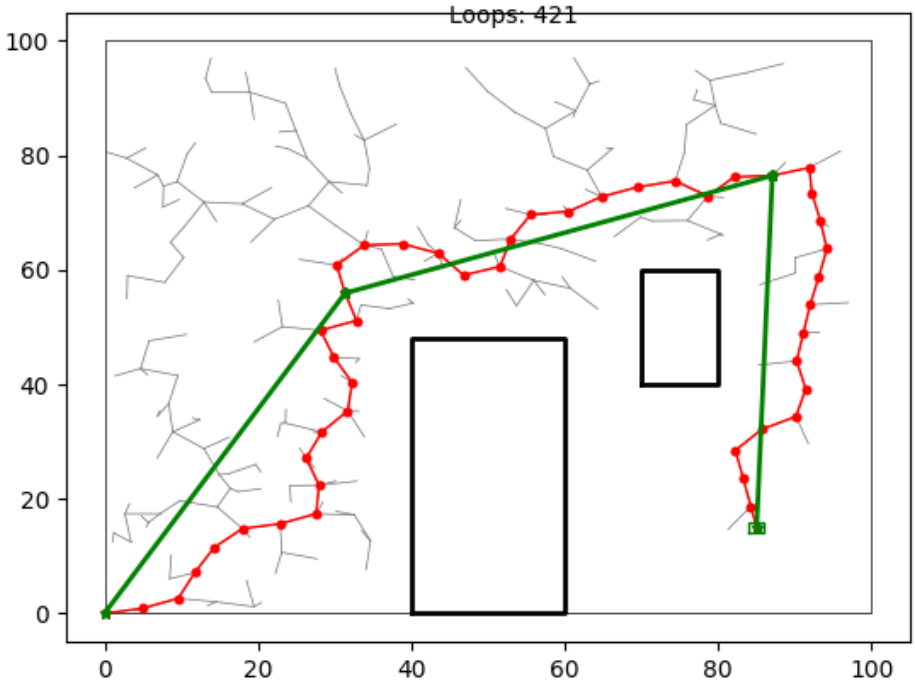

Figura 30 – Planejamento de rota 2D utilizando o RRT e suavização de caminho em Python - Execução 1. 421 iterações.

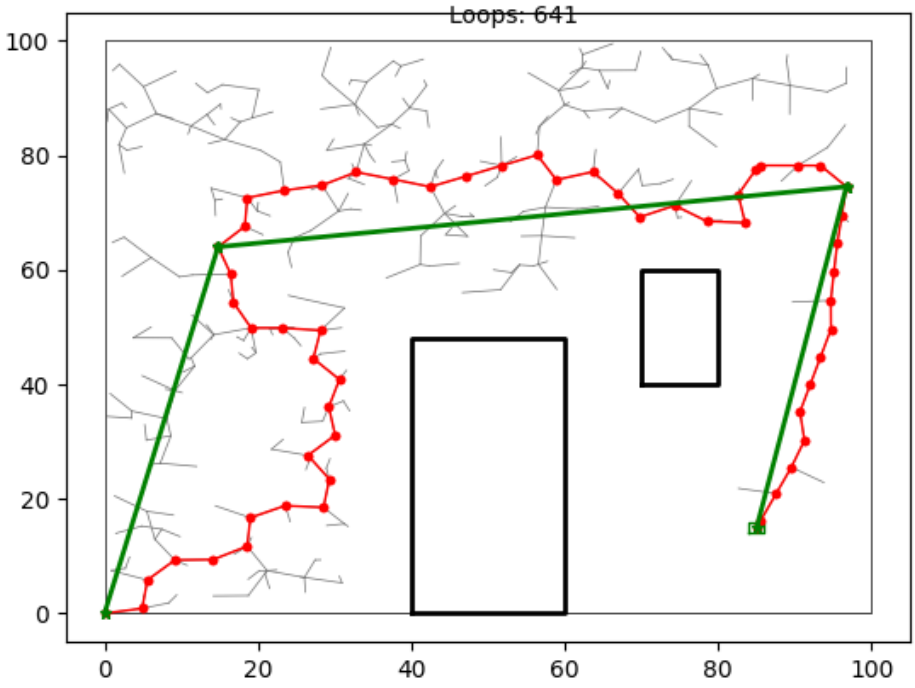

Figura 31 – Planejamento de rota 2D utilizando o RRT e suavização de caminho em Python - Execução 2. 641 iterações.

3.2.2.1.2 Implementação tridimensional

Em seguida, foi desenvolvido o algoritmo aplicado ao contexto 3D. Basicamente, seu funcionamento segue a mesma premissa do modelo em 2D desenvolvido na subseção anterior, mas, com a incorporação de dimensionalidade e complexidade do sistema ao longo do eixo Z do plano cartesiano.

Os resultados alcançados foram satisfatórios e conseguem representar com sucesso o funcionamento do algoritmo proposto. A partir das figuras 32 e 33, pode-se perceber como a árvore de caminhos foi construída. Devido a tridimensionalidade, a complexidade, e a possibilidade de não encontrar o ponto de destino facilmente, aumenta consideravelmente.

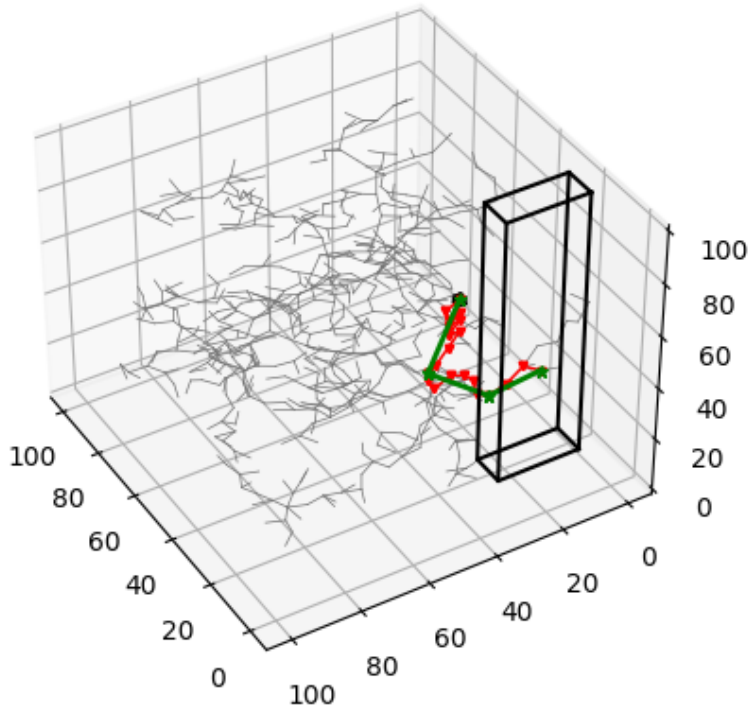

Figura 32 – Planejamento de rota 3D utilizando o RRT em Python - Execução 1

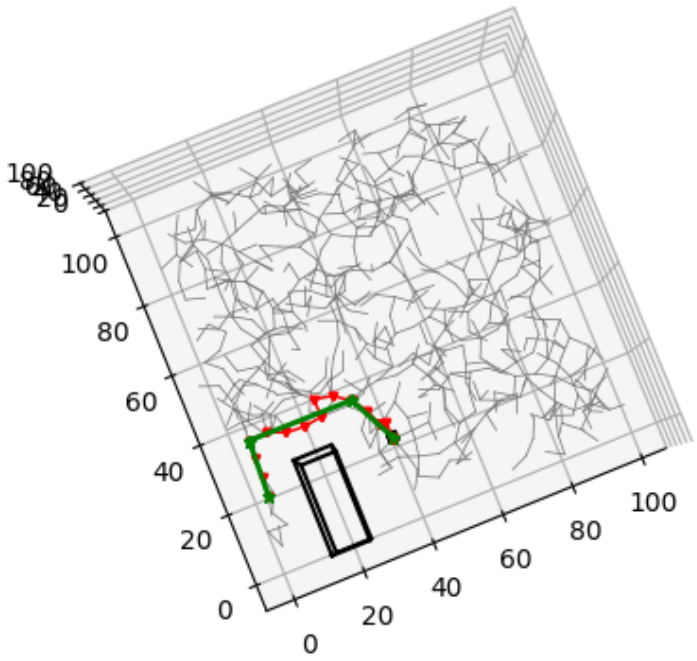

Figura 33 – Planejamento de rota 3D utilizando o RRT em Python - Vista superior - Execução 1

Realizando uma segunda execução, ilustrada nas figuras 34 e 35, pode-se verificar o fator de aleatoriedade entre cada execução.

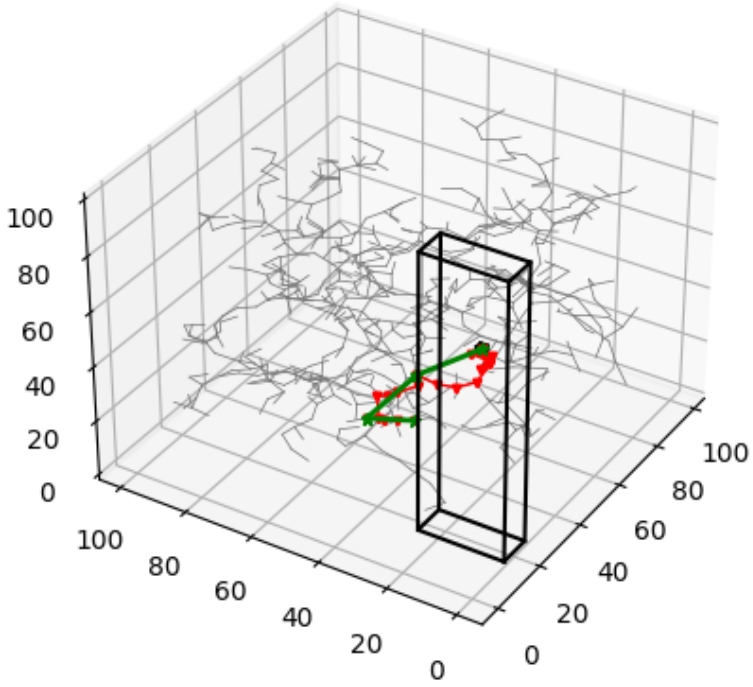

Figura 34 – Planejamento de rota 3D utilizando o RRT em Python - Execução 2

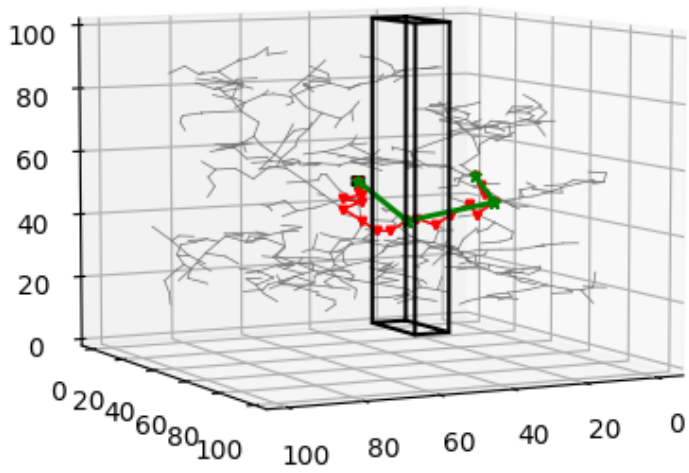

Figura 35 – Planejamento de rota 3D utilizando o RRT em Python - Vista lateral - Execução 2

3.2.2.2 RRT-Connect

3.2.2.2.1 Implementação bidimensional

Para a RRT-Connect, a variação bidirecional do RRT, como definido na subseção 2.3.4, tem-se uma caracterização muito similar ao já implementado anteriormente para o RRT, mas, com a particularidade de desenvolvimento de uma árvore a partir do ponto inicial e outra do ponto de destino.

Nesse caso, vale pontuar a principal diferença dentro do algoritmo desenvolvido, ilustrado na Figura 36, a função *connectedTrees*() checa a existência de um mesmo ponto nas duas árvores a partir da colisão dos nós dessas árvores.

```
#Verifica se as duas árvores estão conectadas
    def connectedTrees(self,A):
        n1=self.number_of_nodes()-1
        (x1,y1)= (self.x[n1],self.y[n1])
        c= False #Default: Não há conexão
        num=A.number_of_nodes()
        for i in range (0,num-1):
            (x2,y2)= (A.x[i],A.y[i])
            if E.checkCollision(x1,y1,x2,y2)==1:
                self.add_node(n1+1,x2,y2)
                self.add_edge(n1,n1+1)
                self.nodeConnected=n1+1
                A.nodeConnected=i
                c= True
                break
        return c
```

Figura 36 – Função que checa a coexistência de um mesmo ponto nas duas árvores geradas.

Partindo para os resultados, percebe-se o comportamento já esperado. No casos das figuras 37 e 38, é possível perceber como cada uma das árvores, representadas em cinza e azul claro, foi amostrada ao longo das iterações e o ponto onde ocorreu a conexão das duas. Já é possível concluir que, com o desenvolvimento de duas árvores paralelamente, a quantidade de iterações necessárias até que se encontre uma solução final é muito menor se comparado ao algoritmo original do RRT. Em seguida, tem-se a representação desse experimento com os caminhos suavizados, ilustrados nas figuras 39 e 40.

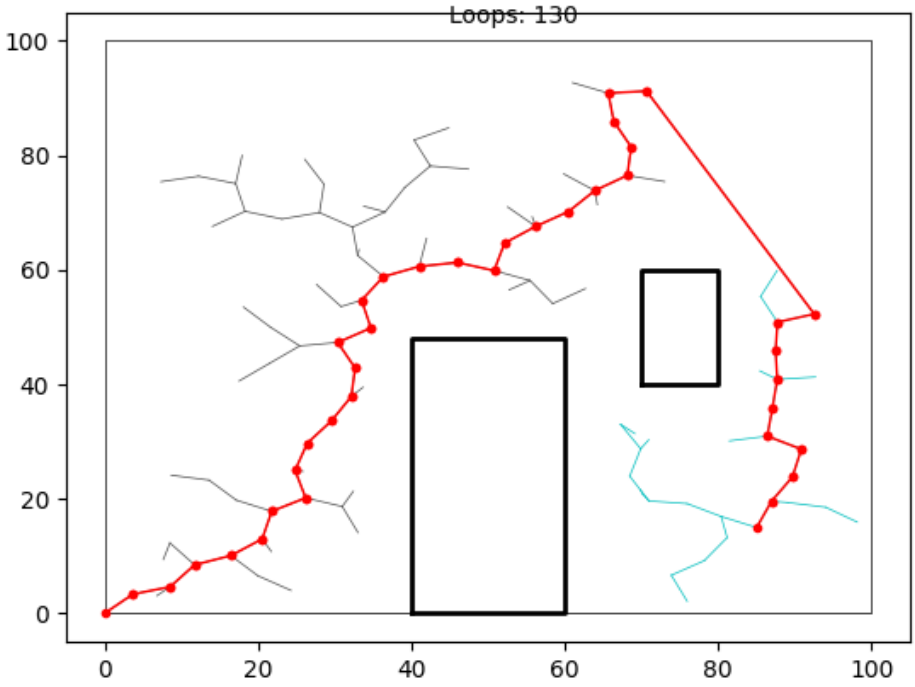

Figura 37 – Planejamento de rota 2D utilizando o RRT-Connect em Python - Execução 1.

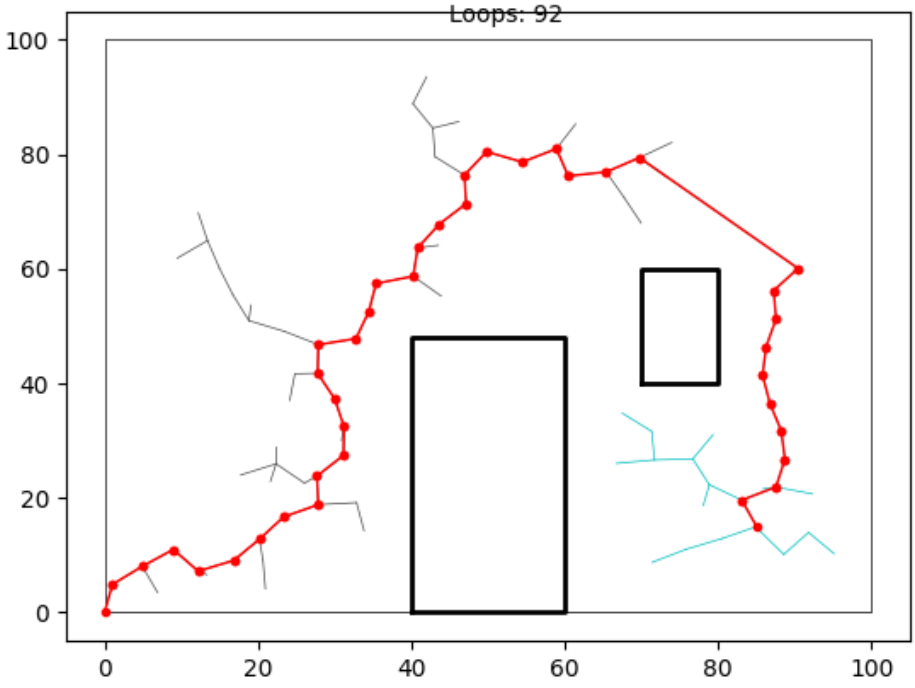

Figura 38 – Planejamento de rota 2D utilizando o RRT-Connect em Python - Execução 2.

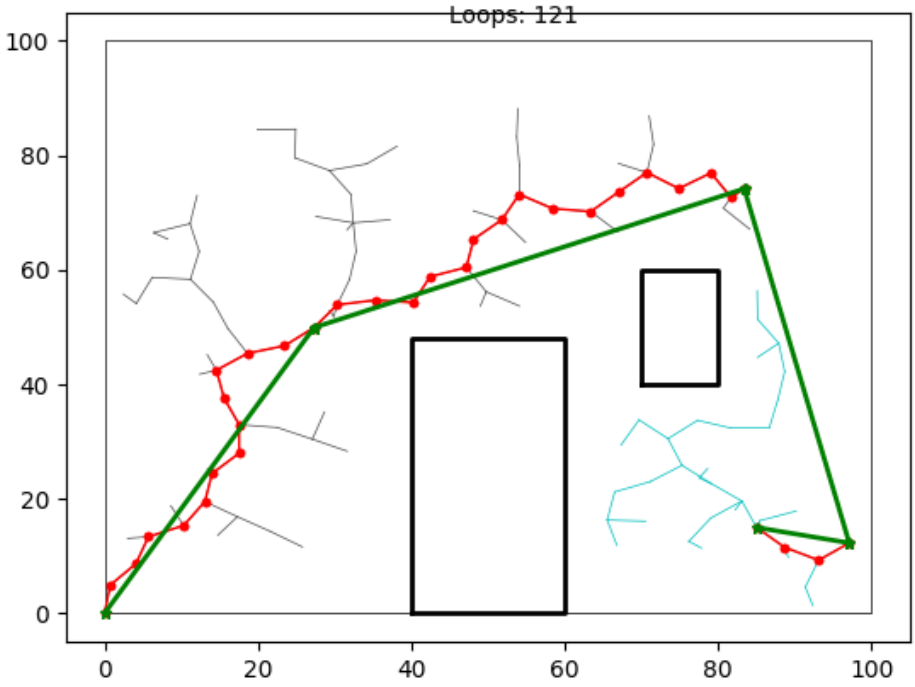

Figura 39 – Planejamento de rota 2D utilizando o RRT-Connect e caminho suavizado em Python - Execução 1.

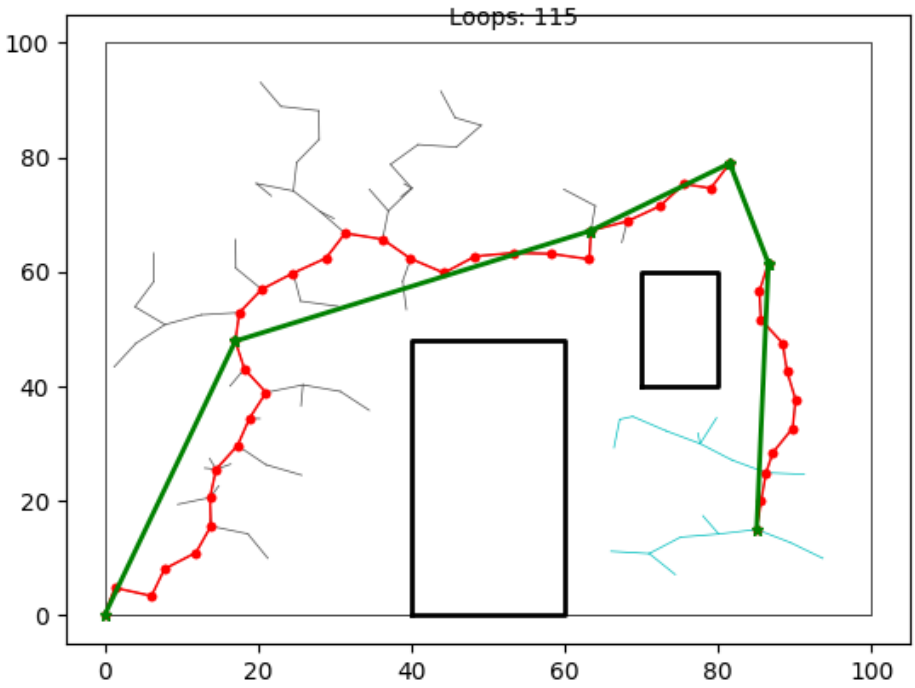

Figura 40 – Planejamento de rota 2D utilizando o RRT-Connect e caminho suavizado em Python - Execução 2.

3.2.2.2.2 Implementação tridimensional

Seguindo os mesmos testes realizados para o algoritmo RRT original, agora, foi implementado a versão 3D do RRT-Connect. Utilizou-se as mesmas conceituações utilizadas para o modelo 2D, com ampliação da varredura de posições pelo eixo Z do plano cartesiano. Alguns dos testes está representados nas figuras 41, 42, 43 e 44

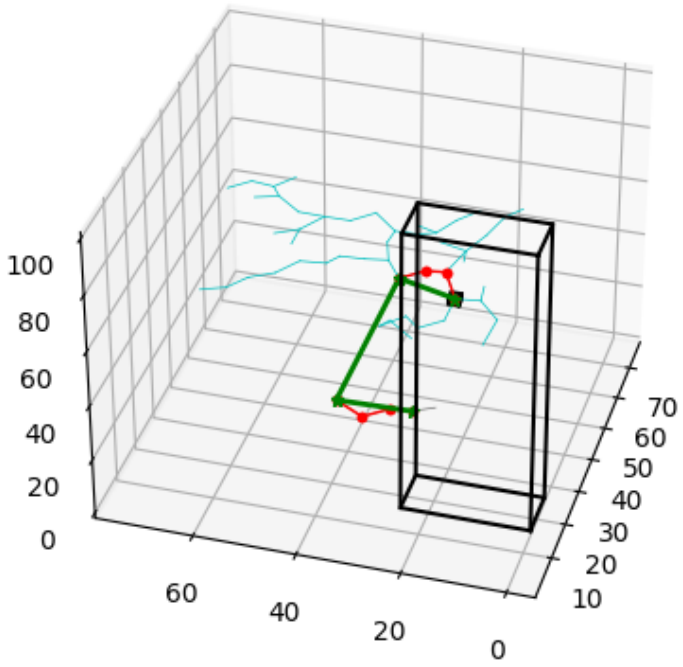

Figura 41 – Planejamento de rota 3D utilizando o RRT-Connect em Python - Execução 1

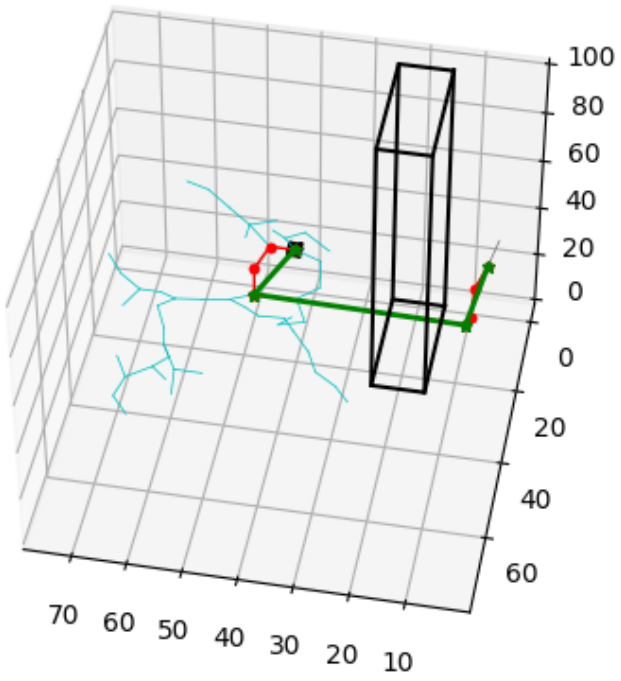

Figura 42 – Planejamento de rota 3D utilizando o RRT-Connect em Python - Vista lateral - Execução 1

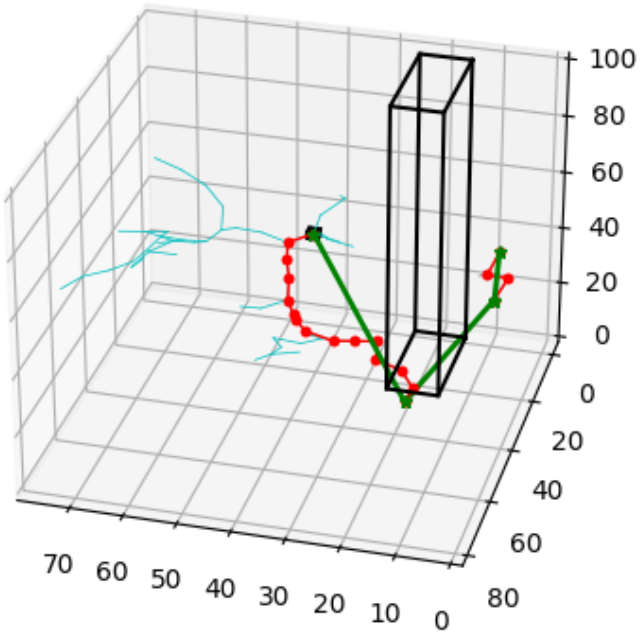

Figura 43 – Planejamento de rota 3D utilizando o RRT-Connect em Python - Execução 2

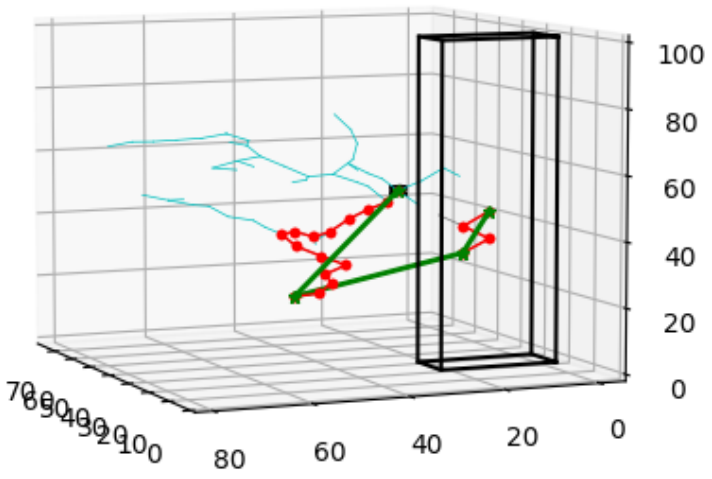

Figura 44 – Planejamento de rota 3D utilizando o RRT-Connect em Python - Vista lateral - Execução 2

3.3 Integração dos algoritmos

Com a descrição do modelo do UR3 e a implementação do planejamento do rota prontas, pôde-se seguir para a integração de cada parte desenvolvida, com o objetivo de alcançar uma representação em ambiente simulado.

Seguindo os padrões adotados para integração de dispositivos robóticos , foi utilizado o framework ROS implementado em Python para desempenhar o papel de comunicação do projeto.

Normalmente, projetos desenvolvidos em ROS são vistos com implementação em C++ devido a consolidação da linguagem, mas, cada vez mais sua implementação e Python vem sendo utilizada pela comunidade ROS por sua sintaxe mais concisa. Uma vez que o ROS já está sendo nativamente projetado para ser programado em C++ e/ou Python, todas as funções comumente utilizadas em um projeto ROS/C++ também existem na versão Python, com a vantagem de não necessitar diversas linhas do código.

3.3.1 Python + ROS

Uma vez implementado os algoritmos de planejamento em Python, foi necessário adaptar a estrutura do projeto para o modelo padrão do ROS. Utilizando a estrutura convencional de publicador e subscritor (OPENROBOTICS, 2017), ilustrada na Figura 45, a aplicação consegue manter comunicação continua com o objeto alvo a ser controlado/monitorado. Nesse caso, o alvo está sendo o modelo URDF do UR3 dentro do simulador CoppeliaSim.

```
if __name__ == '__main__':
    # Init ROS node
    rospy.init_node('UR3', anonymous=True)

    # Create publisher to send joints configuration
    pub = rospy.Publisher('/sim_ros_interface/ur3_joints', Float32MultiArray, queue_size=10)
    # Create subscriber to listen to robot configuration
    sub = rospy.Subscriber("/sim_ros_interface/jointState", PoseStamped, callback)

    rospy.spin()
```

Figura 45 – Função *main*() com inicialização do ROS node e criação do publicador e subscritor.

3.3.2 CoppeliaSim

Com o ambiente construído dentro do CoppeliaSim, foi possível realizar os testes de funcionamento do projeto. Como ilustrado nas figuras 50 e 51, foi realizado testes com diferentes configurações de obstáculos para movimentação do UR3.

A partir da publicação de dados para movimentação do robô pelo projeto ROS, o algoritmo existente dentro da cena simulada no CoppeliaSim consegue receber os dados e traduzi-los de forma a movimentar o corpo rígido do UR3 representado.

Para definição do ambiente da cena que será utilizada, foi inserido o corpo rígido, que representa o UR3, e um bloco retangular, que representa um obstaculo a ser evitado, como ilustrado na Figura 46. Apesar de não se tratar de uma cena muito complexa, servirá perfeitamente para teste e visualização clara da movimentação do robô. Posteriormente, já nos experimentos que serão utilizados para avaliar melhor o desempenho do projeto, no Capítulo 4, será utilizado diferentes configurações de ambiente de forma a inserir a validar os algoritmos.

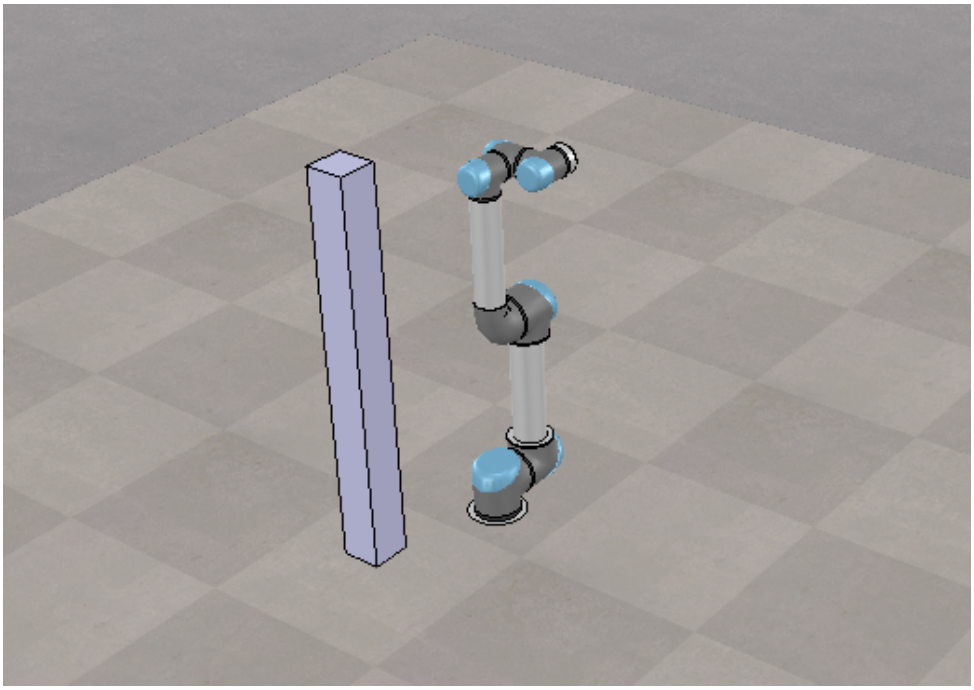

Figura 46 – Cena escolhida para realização dos testes.

Como já citado, o UR3 é definido como um corpo rígido dentro do simulador, mas com a característica de ser formado um conjunto de pequenos corpos que descrevem toda a estrutura do robô, assim como o modelo real. Como ilustrado na Figura 47, cada uma das juntas e links do robô são representadas por objetos separados que são interligados seguindo os padrões do formato URDF[1] e do simulador, assim como descrito na subseção 1.3.1.

[1] Formato padrão amplamente utilizado para descrever características de robôs, como representação visual, dinâmica e modelo de colisão.

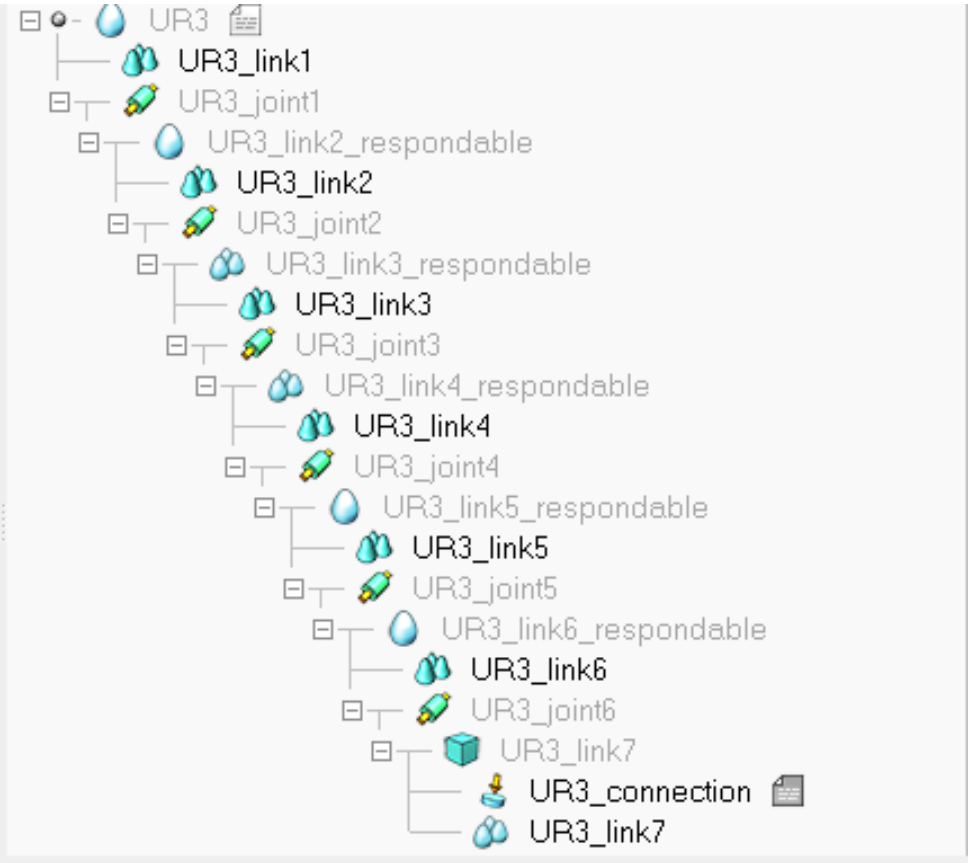

Figura 47 – Conjunto de objetos que descrevem o UR3.

Por fim, para descrever como deve ser dado o comportamento e movimentação do robô durante a simulação, têm-se as diversas funções declaradas no script de definição de interações do sistema na simulação[2]. Pode-se ilustrar as funções *moveJoints*() e $findCollisionFreeMovement$, representadas nas figuras 48 e 49. A primeira, realiza a função de determinar o valor de movimentação de cada junta a cada instante de tempo da simulação, incluindo definições de velocidade e aceleração.

[2] Disponível em: http://github.com/guilhermecstr/planejamento-rota-ur3

```
-- Main function:
while true do
    local path=sim.getStringSignal(modelName..'runPath')

    if path and #path>0 then
        path=sim.unpackFloatTable(path)
        pathConfig={0.0,0.0,0.0,0.0,0.0,0.0}

        sim.setThreadAutomaticSwitch(false)
        for i=1,#path/6,1 do
            for j=1,6,1 do

                pathConfig[j] = path[(i-1)*6+j]

            end
            sim.rmlMoveToJointPositions(jh,-1,currentVel,currentAccel,maxVel,maxAccel,maxJerk,pathConfig,targetVel)

            sim.switchThread()
        end
        sim.setThreadAutomaticSwitch(false)
        sim.clearStringSignal(modelName..'runPath')
    end
    sim.switchThread()
end
```

Figura 48 – Função *moveJoints*() implementada em Lua no CoppeliaSim.

Já na função *findCollisionFreeMovement*() descreve-se a importante tarefa de definir a movimentação do robô para que a rota, dada como entrada no sistema, seja seguida e que nenhuma junta do robô acabe colidindo com o obstáculo. Foi percebido que além de encontrar a movimentação que satisfaça o requisito de não colisão, ela também é capaz de interpolar os pontos informados como nós no caminho RRT dado como entrada, suavizando o percusso gerado para o ambiente de simulação e, possivelmente, para aplicações práticas.

```
]findCollisionFreeConfigAndCheckApproach=function(task)
    -- Here we search for a robot configuration..
    -- 1. ..that matches the desired pose (task.goalPose)
    -- 2. ..that does not collide in that configuration
    -- 3. ..that does not collide and that can perform the IK linear approach
    sim.setObjectMatrix(task.ikTarget,-1,task.goalPose)
    print('findCollisionFreeConfigAndCheckApproach',task.ikTarget,-1,task.goalPose)
    -- Here we check point 1 & 2:
    local c=sim.getConfigForTipPose(task.ikGroup,task.jh,0.65,20,nil,task.collisionPairs)
]   if c then
]       if task.approachVector[1]~=0 or task.approachVector[2]~=0 or task.approachVector[3]~=0 then
            -- Here we check point 3:
            local m=getShiftedMatrix(task.goalPose,task.approachVector,false)
            local path=generateIkPath(c,m,task)
]           if path==nil then
                c=nil
-           end
-       end
-   end
    return c
end
```

Figura 49 – Função *findCollisionFreeMovement*() implementada em Lua no CoppeliaSim.

Abaixo, nas figuras 50 e 51, é possível visualizar simulações utilizadas para teste de funcionamento nas cenas estudadas e desenvolvidas.

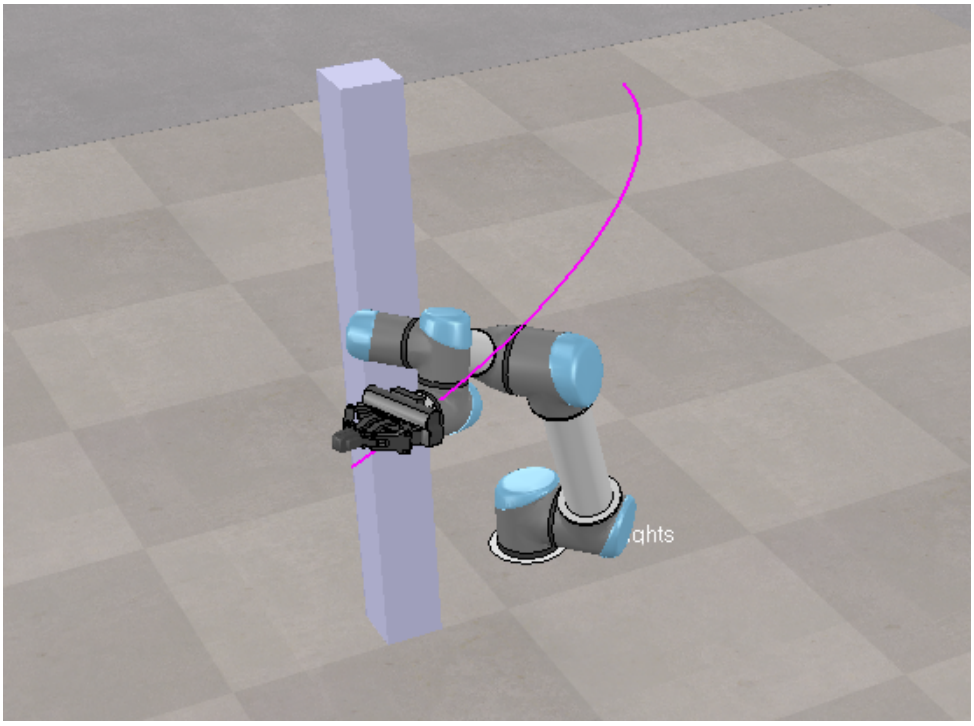

Figura 50 – Simulação 1 sendo executada no CoppeliaSim a partir dos dados recebidos por ROS/Python.

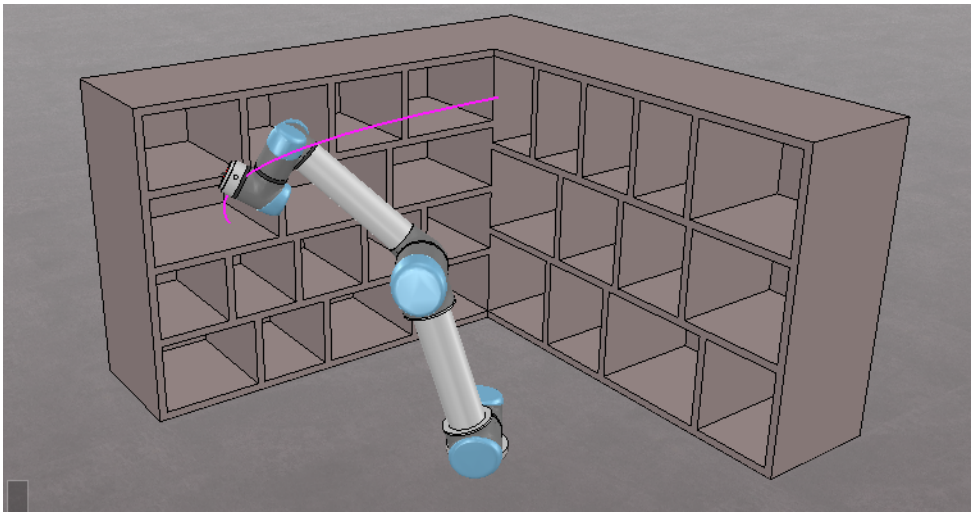

Figura 51 – Simulação 2 sendo executada no CoppeliaSim a partir dos dados recebidos por ROS/Python.

3.4 Execução

Assim como apresentado nas seções anteriores, o projeto, como um todo, foi pensado de forma modularizada. Dessa forma, foi trabalhado cada etapa separadamente e, por fim, foi feito a integração de algoritmos, como detalhado na seção 3.3.

Como preparação do computador a ser utilizado para o projeto, sugere-se os seguintes requisitos do sistema:

- Ubuntu 18.04 LTS
- Python 3.8
- ROS Melodic
- CoppeliaSim V4.3.0

Descrevendo melhor os modos de operação e execução, pode-se listar as seguintes possibilidades:

- Execução RRT: Realiza a execução do algoritmo RRT padrão em um ambiente 3D para alcançar as coordenadas informadas como parâmetro na linha de comando.

$ python RRT-3D.py --goal 0.4 -0.1 0.5

- Execução RRT-Connect: De forma similar ao anterior, realiza a execução do algoritmo RRT-Connect em um ambiente 3D para alcançar as coordenadas também informadas na linha de comando.

$ python RRT-Connect-3D.py --goal 0.4 -0.1 0.5

- Execução do projeto completo: Tratando-se de uma execução que envolverá diferentes sistemas e a integração entre eles, é necessário uma preparação do ambiente. Sendo assim, precisa-se seguir os seguintes passos na ordem indicada:

  1. Inicialização do servidor ROS: Utilizar o comando

  $ roscore

  2. Inicializar o CoppeliaSim: Ao inicializar o programa, é importante se certificar que o mesmo conseguiu se conectar com o servidor ROS inicializado anteriormente. Para isso, conferir mensagens de registro retornadas no terminal, como ilustrado na Figura 52.

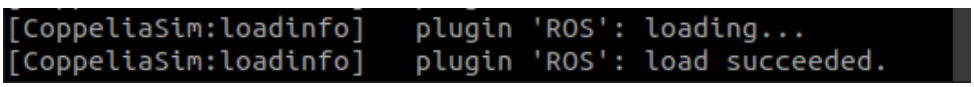

Figura 52 – Mensagem de confirmação de conexão entre CoppeliaSim e servidor ROS.

  3. Abrir cena que representa ambiente que será feito o planejamento: Cena deve estar já preparada com os obstáculos posicionados no local desejado, como ilustrado na Figura 53.

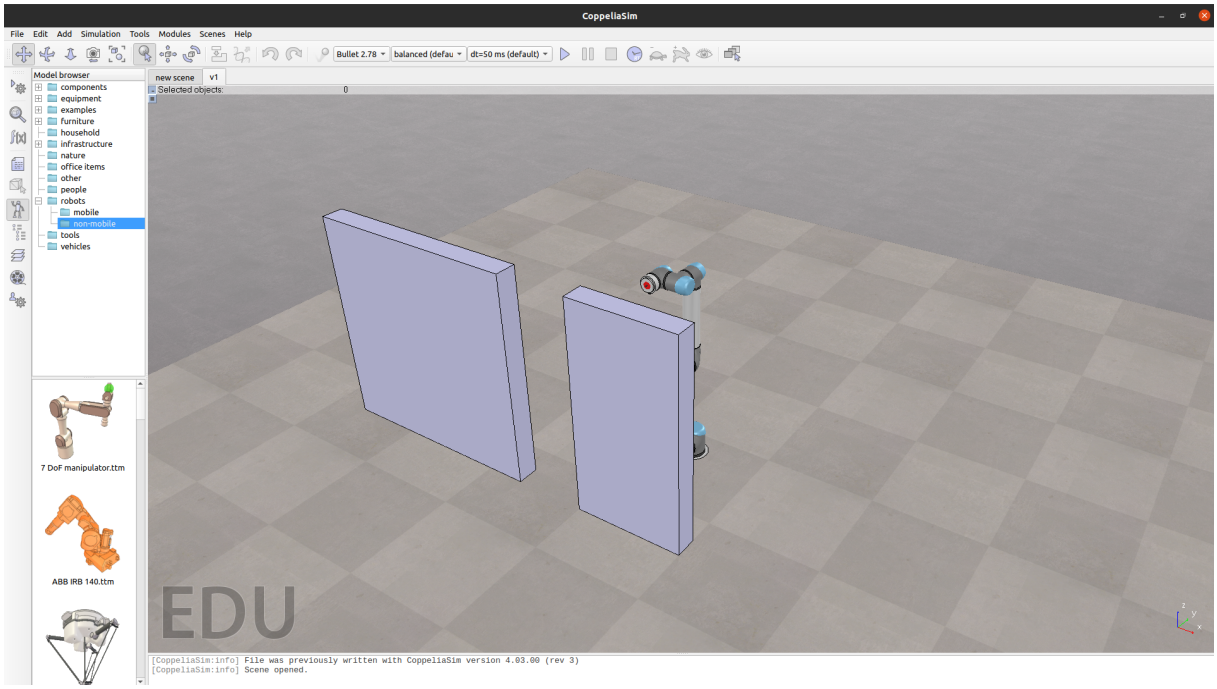

Figura 53 – CoppeliaSim com cena preparada para execução.

4. Executar arquivo *Run.py*: Com todas as etapas anteriores seguidas, basta executar o arquivo principal do projeto, informando os parâmetros necessários, como exemplificado a seguir. Como modo de operação (*mode*) tem-se *"default"* para o RRT padrão e *"connect"* para o RRT-Connect.

$ python Run.py --mode default --goal 0.4 -0.1 0.5

4 Resultados

Neste capítulo, será discutido os resultados obtidos para os experimentos realizados em 3 ambientes com características diferentes para simular o situações de trabalho em que o robô pode estar inserido.

Primeiramente, será feita a análise do desempenho de cada algoritmo considerando apenas a execução do planejamento realizado. Para tal tarefa, foi realizado cerca de 150 planejamentos para cada algoritmo em cada ambiente projetado. A partir disso, é possível avaliar parâmetros resultantes como tempo de execução e número de iterações necessárias até a árvore de posições do RRT e RRT-Connect alcancem o destino esperado. Depois, são apresentados e discutidos os resultados com o projeto sendo executado juntamente com os algoritmos que descrevem a movimentação do robô no simulador CoppeliaSim.

Sendo assim, como resultado pode-se avaliar a execução dos algoritmos do inicio até o ponto de visualização em simulação. Uma vez que todos os experimentos possuem o mesmo ponto de inicio e destino, é possível realizar uma melhor comparação entre resultados obtidos. Dessa forma, foi escolhido que a rotas sejam geradas partindo da posição inicial $X = -0{,}09$, $Y = -0{,}20$ e $Z = 0{,}88$ do efetuador e que percorra algum caminho até a posição final $X = 0{,}40$, $Y = -0{,}1$ e $Z = 0{,}50$ no plano 3D.

4.1 Experimento 1

Para a composição da cena desse experimento, foi escolhido um posicionamento em paralelo de dois obstáculos com as medidas de $10x80x100$ centímetros, posicionado logo à frente do manipulador, e $10x40x100$ centímetros, ligeiramente deslocado para o lado no eixo Y, permitindo apenas um pequeno vão de 28 centímetros para que o robô possa operar, como ilustrado na Figura 54.

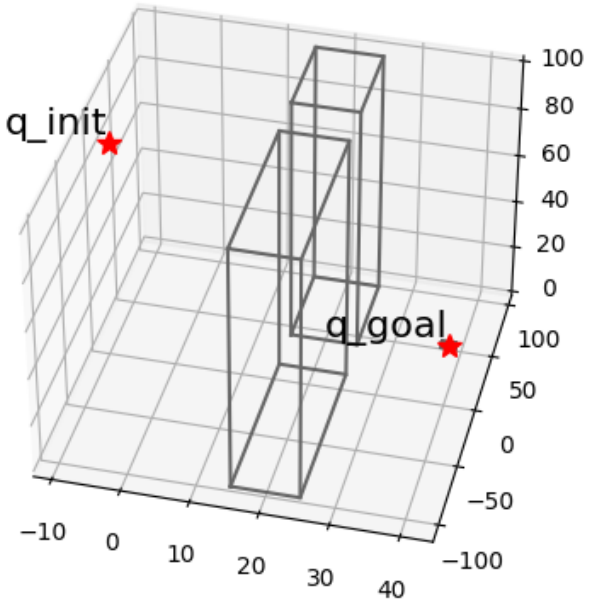

Figura 54 – Ambiente projetado para o Experimento 1.

4.1.1 RRT

Assim como descrito anteriormente, o planejador foi executado 150 vezes para possibilitar a construção de uma certa base de dados de desempenho do algoritmo para o ambiente proporcionado para o experimento 1. Como apresentado na Tabela 3, têm-se que, em média, foi necessário cerca de 755 iterações, isto é, amostragem de nós, para que o RRT consiga encontrar o um caminho viável entre q_{init} e q_{goal} e 0,4353 segundos de execução.

Continuando nos dados informados na Tabela 3, também é possível verificar os resultados extremos encontrados durante o experimento, como o caso extremo superior de 4740 iterações e 4,545 segundos , e o caso extremo inferior de 100 iterações e 0,0604 segundos. Tais situações serão melhor discutidas na seção 4.4.

Tabela 3 – Experimento 1 - Desempenho do RRT

RRT		
	Quantidade de Iterações	Tempo de execução
Média	755,9514	0,4353 s
Mediana	340	0,1282 s
Máximo	4740	4,5450 s
Mínimo	100	0,0604 s

Com o algoritmo RRT, dentre os 150 planejamentos amostrados, foi selecionado

o seguinte resultado para árvore de caminhos, ilustrada nas figuras 55, 56 e 57. Como já citado, os dois obstáculos forçam para que o planejador trace um caminho que passe pelo vão disponibilizado.

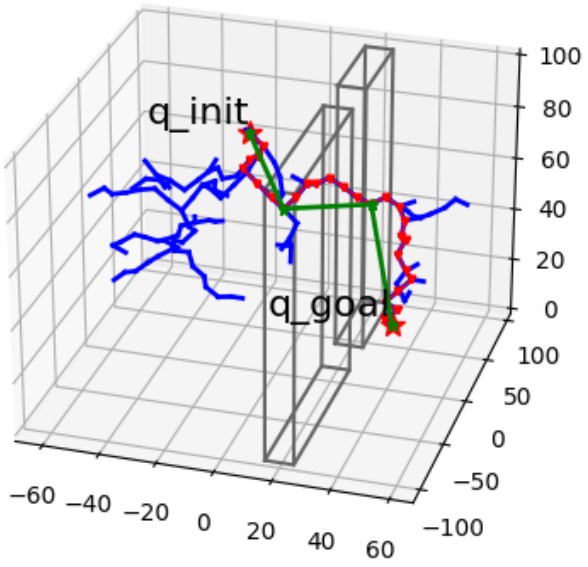

Figura 55 – Árvore RRT e caminho selecionado para experimento 1. Vista 1

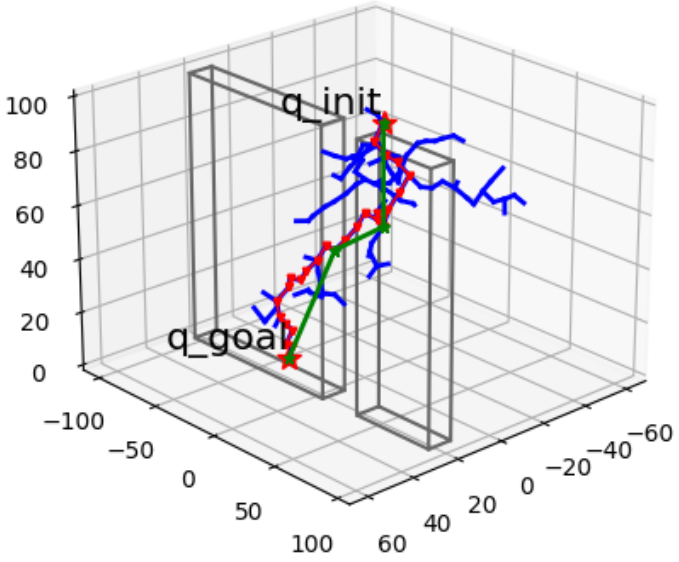

Figura 56 – Árvore RRT e caminho selecionado para experimento 1. Vista 2

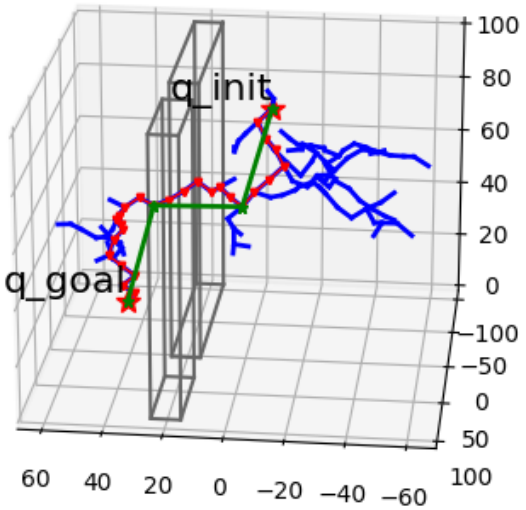

Figura 57 – Árvore RRT e caminho selecionado para experimento 1. Vista 3

Agora, com o prosseguimento da execução, já é possível verificar os resultados obtidos dentro da simulação no CoppeliaSim, uma vez que está sendo utilizado uma integração $Python \leftrightarrow ROS \leftrightarrow CoppeliaSim$. Como ilustrado nas figuras 58,59 e 60.

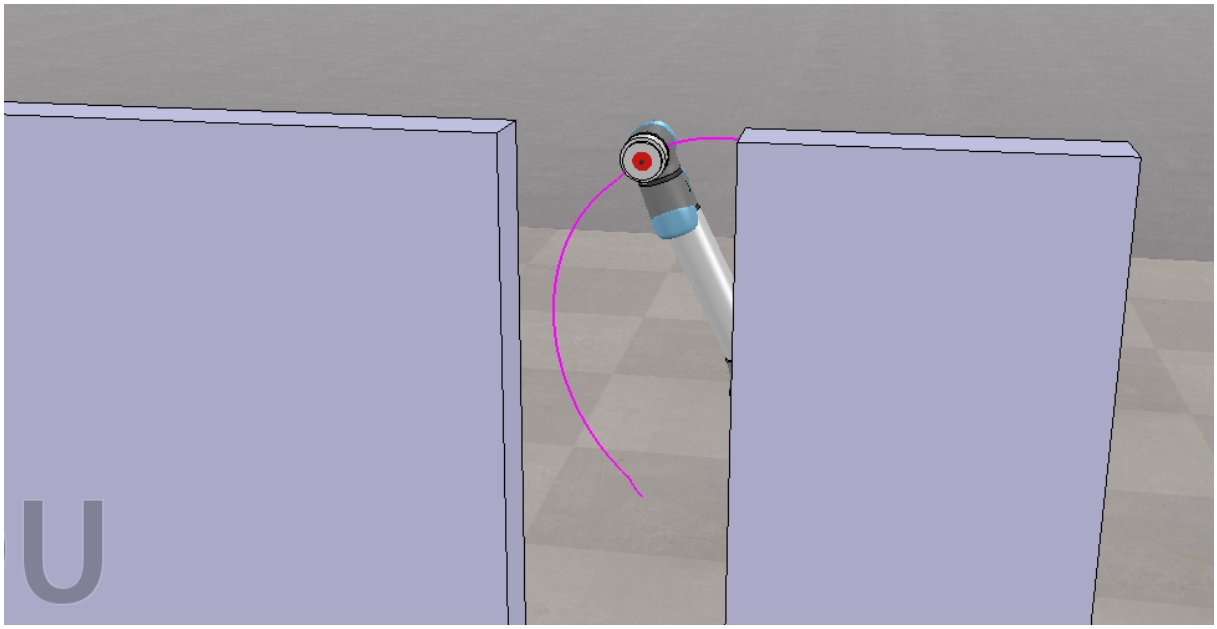

Figura 58 – Experimento 1 realizado com o planejamento de rota RRT dentro do CoppeliaSim. Instante 1

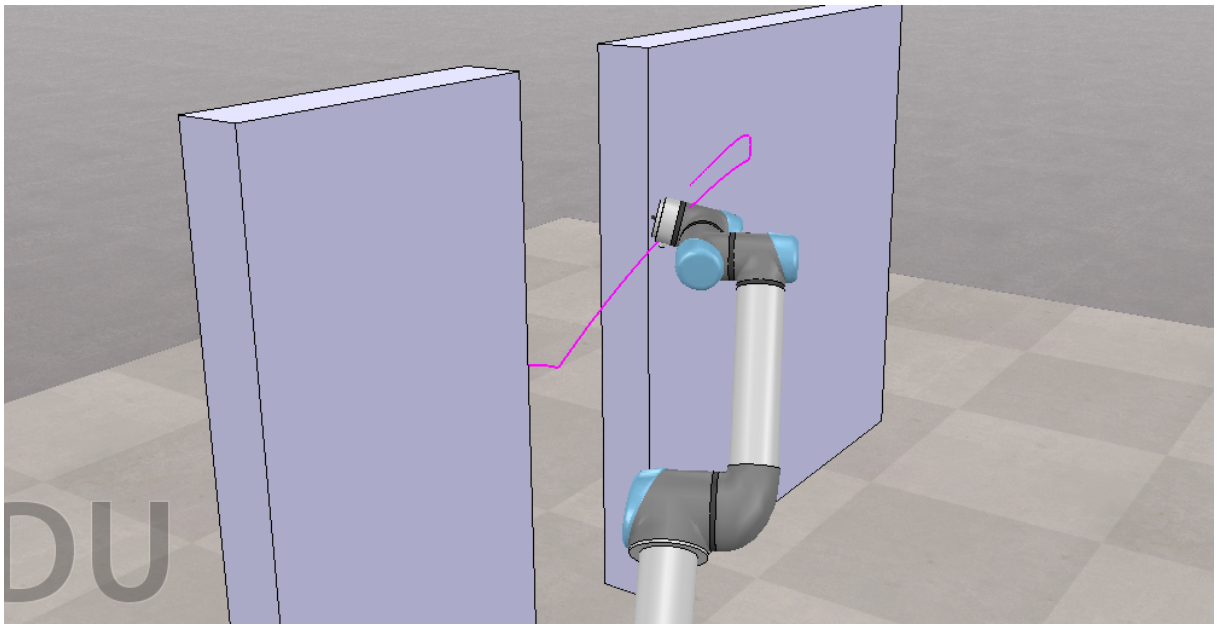

Figura 59 – Experimento 1 realizado com o planejamento de rota RRT dentro do CoppeliaSim. Instante 2

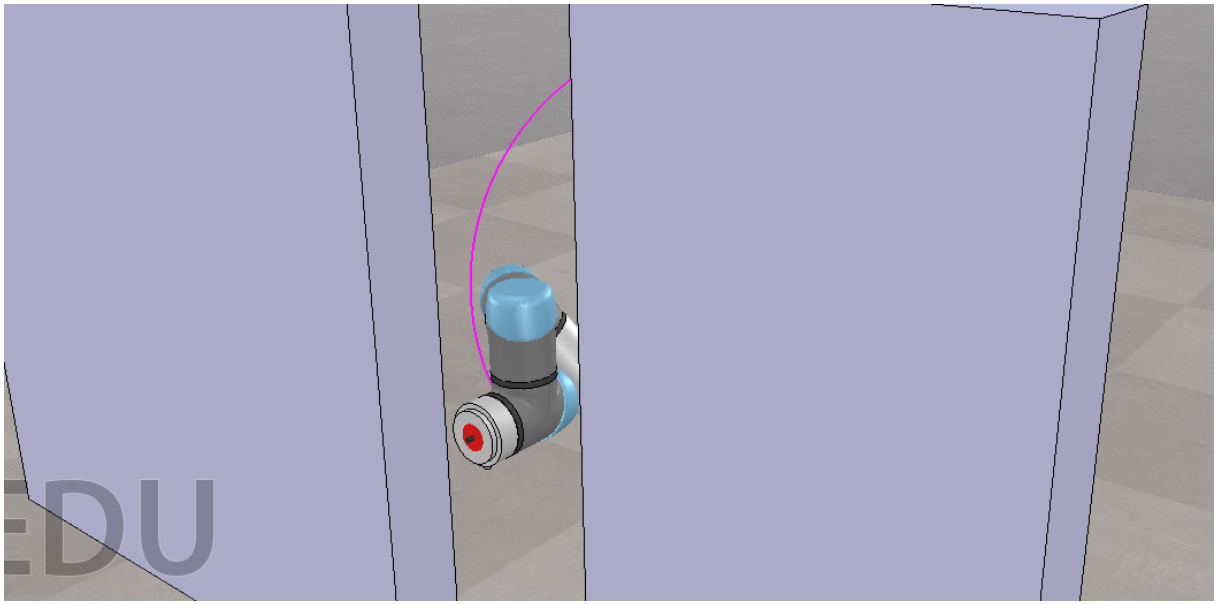

Figura 60 – Experimento 1 realizado com o planejamento de rota RRT dentro do CoppeliaSim. Instante 3

Como citado na subseção 3.3.2, percebe-se que as funções utilizadas para transcrição do caminho RRT gerado realizam um processo de interpolação das coordenadas, suavizando ainda mais a rota a ser seguida pelo robô. Também, é possível verificar o robô conseguiu alcançar o ponto de destino com sucesso seguindo o planejamento feito.

4.1.2 RRT-Connect

Para o algoritmo RRT-Connect, reforçando, foi utilizado as mesmas definições de obstáculo, ponto de inicio e destino, afim de propiciar uma comparação mais fácil entre os algoritmos. Analisando os resultados obtidos através das execuções realizadas, descritas na Tabela 4, e já fazendo um paralelo com os resultados abordados na subseção 4.1.1, é possível verificar um ganho de desempenho considerável ao se utilizar o RRT-Connect.

Para esse primeiro experimento, foi verificado uma média de 38,538 iterações e, apenas, 0,0418 segundos de execução para conseguir chegar à um resultado viável. Na mesma proporção, é possível constatar um considerável ganho em desempenho, até mesmo, ao se comparar o casos extremos amostrados, como valores os máximos e mínimos verificados.

Tabela 4 – Experimento 1 - Desempenho do RRT-Connect

RRT- Connect		
	Quantidade de Iterações	Tempo de execução
Média	38,538	0,0418 s
Mediana	32	0,02152 s
Máximo	63	0,5 s
Mínimo	26	0,0121 s

Através do caminho selecionado para este experimento do RRT-Connect, ilustrado nas figuras 61, 62 e 63, percebe-se, novamente, como o RRT-Connect consegue ser mais eficiente em questão de tempo de execução que o RRT normal, o que reflete na quantidade de nós gerados até alcançar um caminho entre os dois pontos.

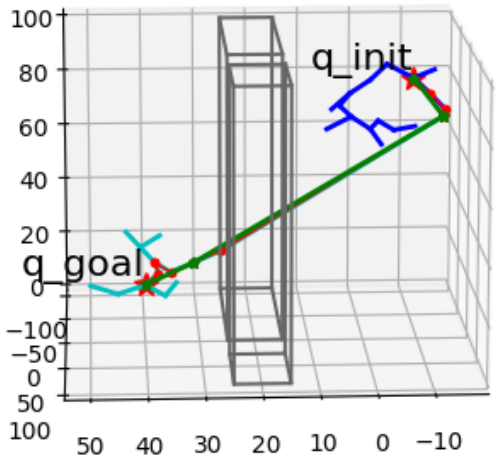

Figura 61 – Árvore RRT-Connect e caminho selecionado para experimento 1. Vista 1

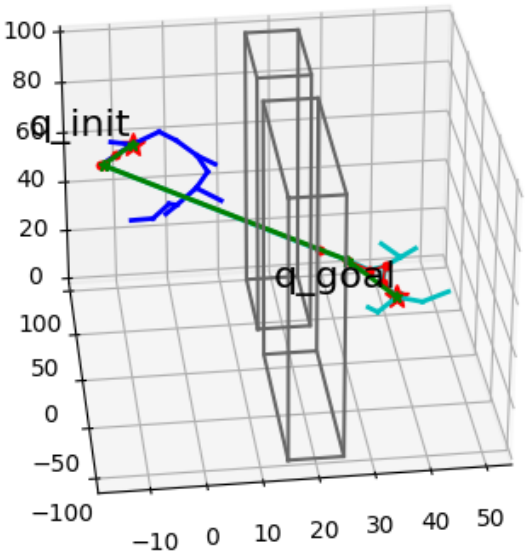

Figura 62 – Árvore RRT-Connect e caminho selecionado para experimento 1. Vista 2

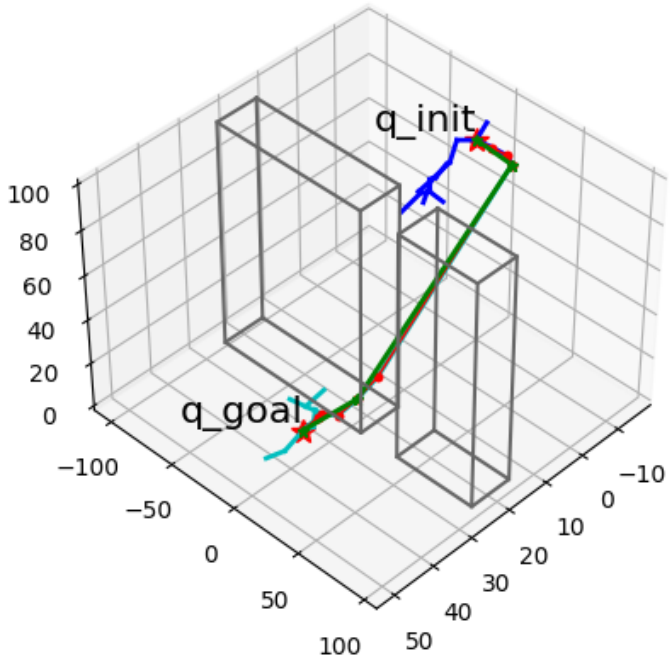

Figura 63 – Árvore RRT-Connect e caminho selecionado para experimento 1. Vista 3

Suas árvores são consideravelmente menores, reduzindo a quantidade de informação a ser avaliada pelo algoritmo final para traçar o caminho. Porém, percebe-se que essa característica também pode ser considerada de forma negativa. Por encontrar o caminho até o ponto destino muito rapidamente, percebe-se que, muita das vezes, o caminho traçado tem um perfil muito grosseiro e corresponde a uma rota pouco eficiente, em que o manipulador precisa se movimentar muito mais do que seria esperado.

Tal situação pode ser verificada na figura 61 durante o planejamento e, inclusive, também, durante a simulação realizada no CoppeliaSim, ilustrada nas figuras 64,65 e 66. É possível perceber que, devido a estratégia utilizada para o desenvolvimento desse algoritmo, já abordado na subseção 2.3.4, o caminho descoberto chega a obrigar o robô a se movimentar para trás em uma situação que não seria necessário, visto o resultado obtido com o RRT anteriormente.

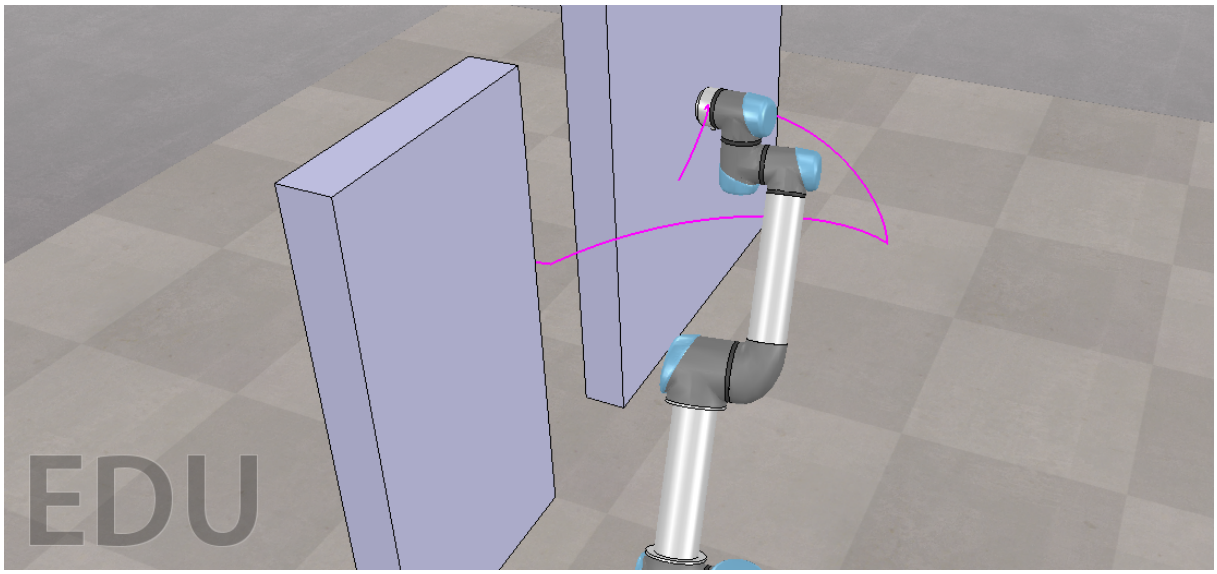

Figura 64 – Experimento 1 realizado com o planejamento de rota RRT-Connect dentro do CoppeliaSim. Instante 1

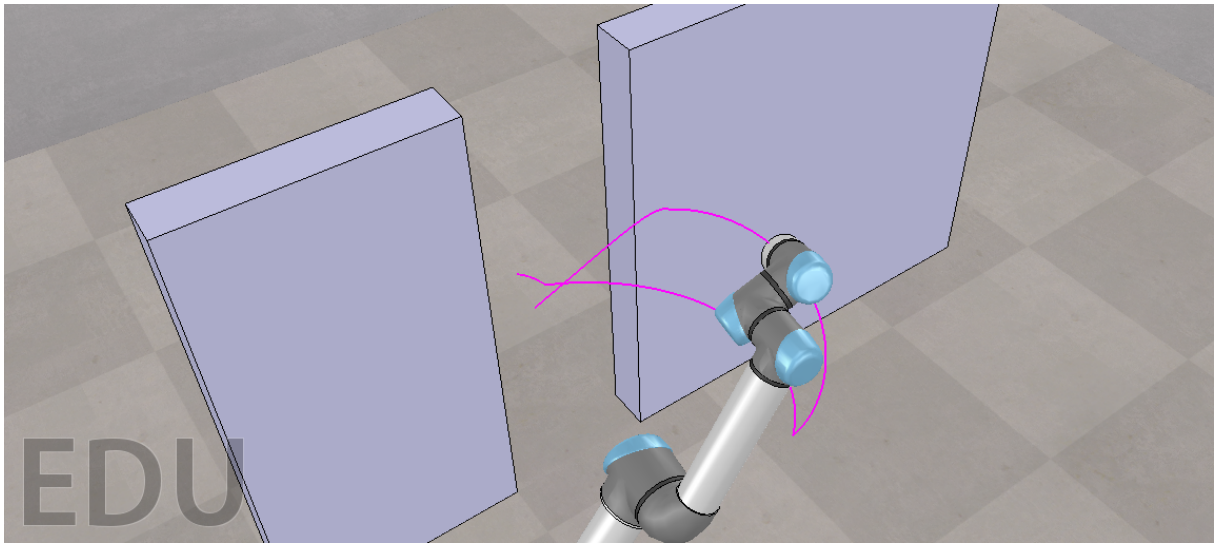

Figura 65 – Experimento 1 realizado com o planejamento de rota RRT-Connect dentro do CoppeliaSim. Instante 2

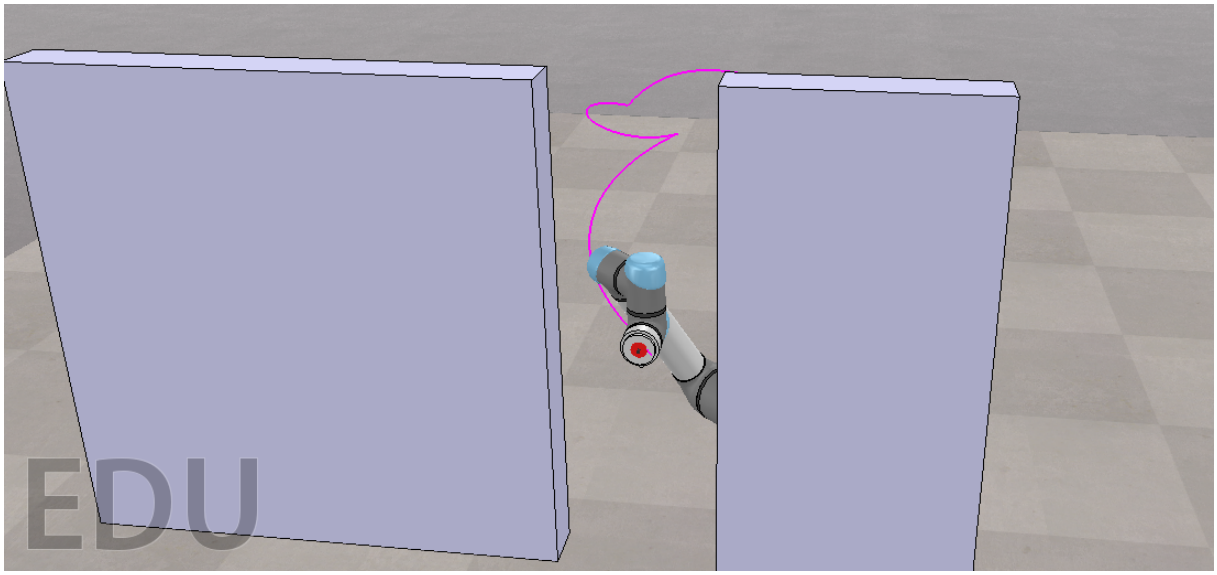

Figura 66 – Experimento 1 realizado com o planejamento de rota RRT-Connect dentro do CoppeliaSim. Instante 3

4.2 Experimento 2

Para a composição da cena desse experimento, foi escolhido um posicionamento de obstáculos de forma similar ao experimento 1, porém com algumas especificidades. Têm-se dois obstáculos: o primeiro com as medidas de $10x80x100$ centímetros, posicionado logo à frente do manipulador; o segundo medindo $10x40x100$ centímetros, posicionado ligeiramente deslocado para o lado no eixo Y e para frente no eixo X. Dessa forma, é ainda mantido a condição de possuir apenas um pequeno vão de 28 centímetros para que o robô possa operar, mas com uma diferente área livre para operação do robô sem colisões, como ilustrado na Figura 67.

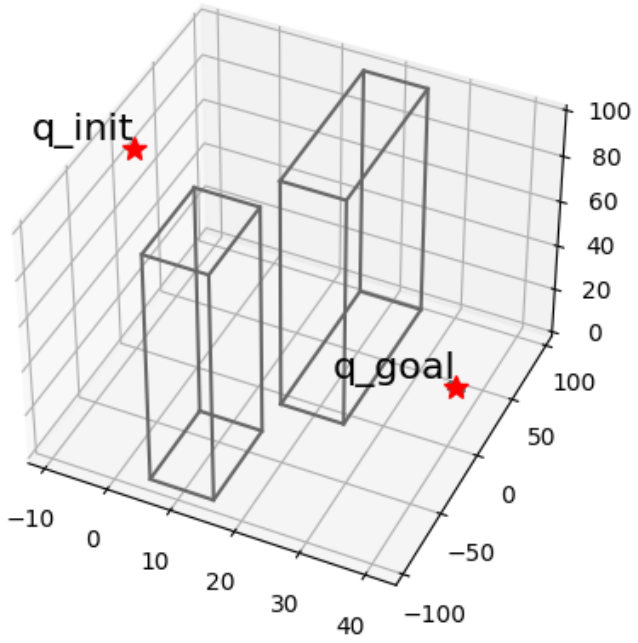

Figura 67 – Ambiente projetado para o Experimento 2.

4.2.1 RRT

Como apresentado na Tabela 5, têm-se que, em média, foi necessário cerca de 307 iterações para que o RRT consiga encontrar o um caminho viável entre q_{init} e q_{goal} e 0,1353 segundos de execução. Os resultados extremos encontrados durante o experimento, seguindo como referencia o experimento 1, está dentro do esperado. Ainda mais pelo nível de similaridade entre os ambientes.

Tabela 5 – Experimento 2 - Desempenho do RRT

RRT		
	Quantidade de Iterações	Tempo de execução
Média	307	0,1353 s
Mediana	180	0,0882 s
Máximo	3570	4,4199 s
Mínimo	60	0,0304 s

Foi selecionado o seguinte resultado para árvore de caminhos, ilustrada nas figuras 55, 56 e 57. Como já citado, os dois obstáculos forçam para que o planejador trace um caminho que passe pelo vão disponibilizado.

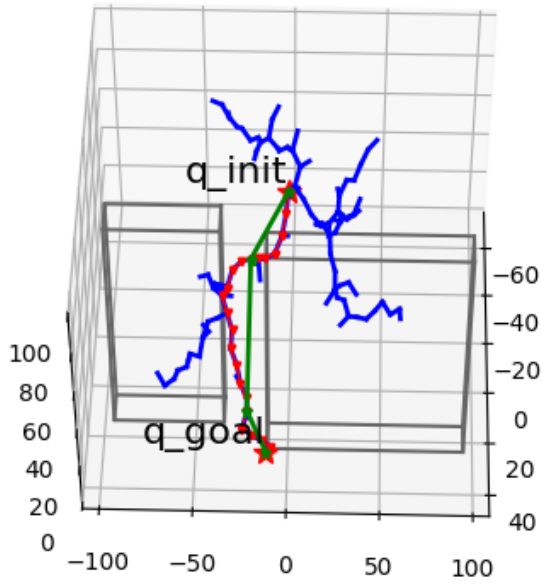

Figura 68 – Árvore RRT e caminho selecionado para experimento 2. Vista 1

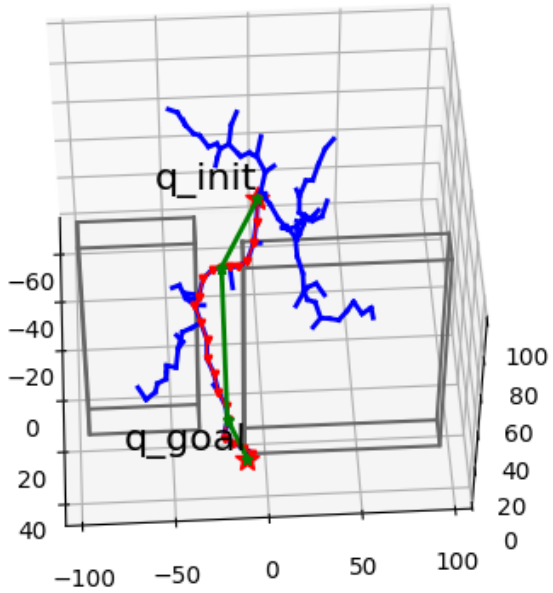

Figura 69 – Árvore RRT e caminho selecionado para experimento 2. Vista 2

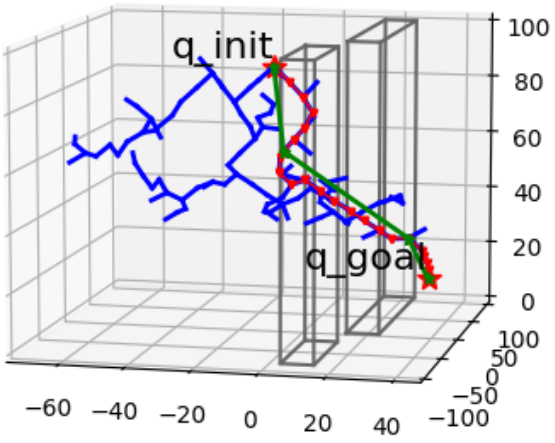

Figura 70 – Árvore RRT e caminho selecionado para experimento 2. Vista 3

Agora, prosseguindo para a execução juntamente com o simulador CoppeliaSim, ilustrado nas figuras 58,59 e 60, é possível perceber movimentações fora do esperado. A integração entre projeto Python e descrição do robô no CoppeliaSim não obteve sucesso para movimentar o manipulador ao longo do caminho planejado.

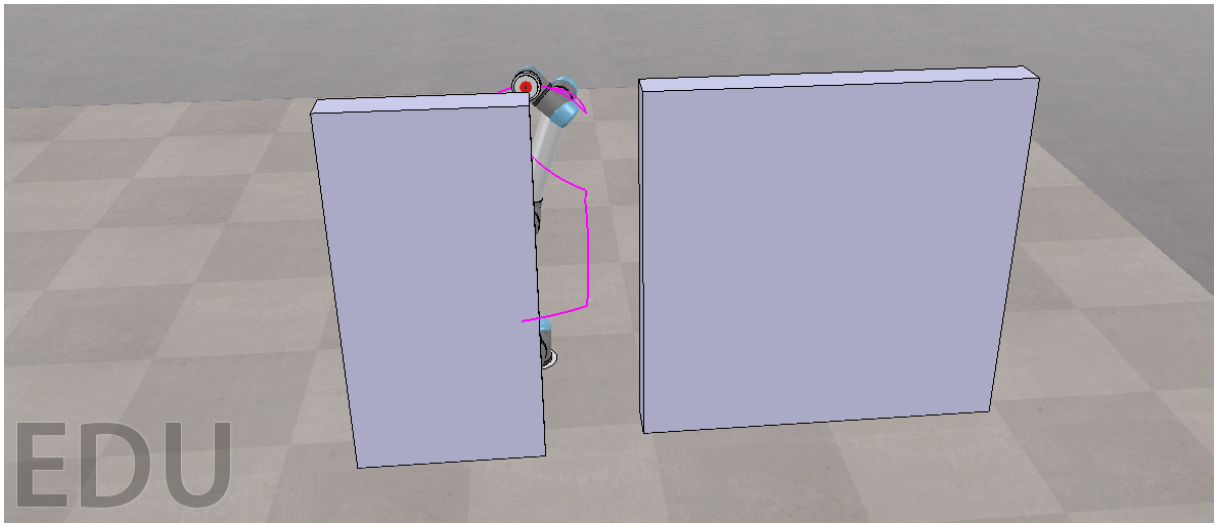

Figura 71 – Experimento 2 realizado com o planejamento de rota RRT dentro do CoppeliaSim. Instante 1

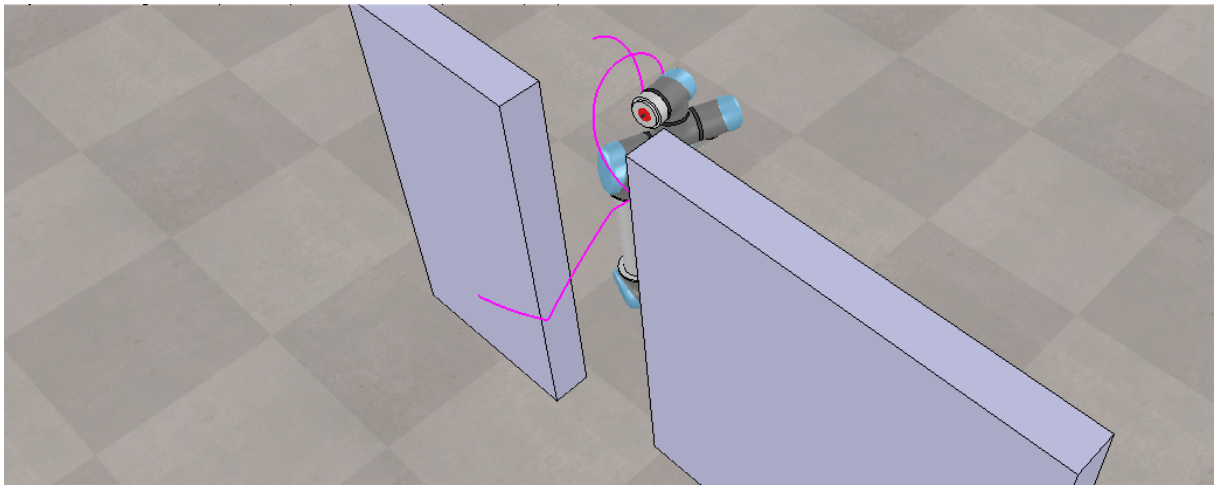

Figura 72 – Experimento 2 realizado com o planejamento de rota RRT dentro do CoppeliaSim. Instante 2

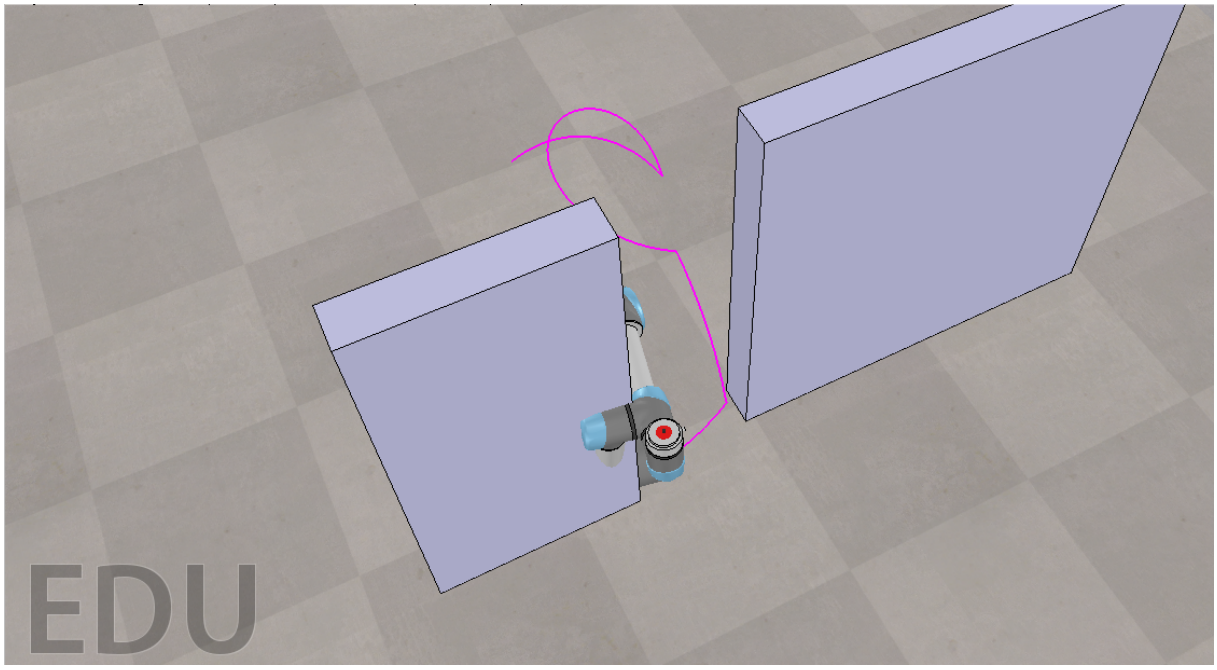

Figura 73 – Experimento 2 realizado com o planejamento de rota RRT dentro do CoppeliaSim. Instante 3

Percebe-se que cálculo de cinemática inversa para determinação das poses e verificação de colisão não conseguiu alcançar um resultado considerado válido para seguir o caminho desejado sem causar o choque entre robô e obstáculo.

Analisando cada instante, percebe-se que, no primeiro instante, o planejador adotou uma percurso inesperado com o objetivo de rotacionar as juntas para o efetuador terminal alcance uma orientação mais ajustada para tal percurso que será seguido na sequencia. Mas, ao chegar no final da rota, não tomou nenhuma medida para contornar o obstaculo, causando a colisão.

4.2.2 RRT-Connect

Mantendo as mesmas condições de ambiente e alvo, foi realizado o experimento para o RRT-Connect. Analisando os resultados obtidos através das execuções realizadas, descritas na Tabela 6, percebe-se que o algoritmo manteve o perfil de ser mais eficiente que o RRT padrão, com uma média de 34,811 iterações e 0,0261 segundos de execução para conseguir chegar à um resultado viável.

Tabela 6 – Experimento 2 - Desempenho do RRT-Connect

RRT-Connect		
	Quantidade de Iterações	Tempo de execução
Média	34,811	0,02611 s
Mediana	32	0,01953 s
Máximo	90	0,05497 s
Mínimo	16	0,0098 s

Através do caminho selecionado para este experimento do RRT-Connect, ilustrado nas figuras 74, 75 e 76, percebe-se que, assim como o esperado, o RRT-Connect consegue ser mais eficiente em questão de tempo de execução que o RRT e menor número de nós.

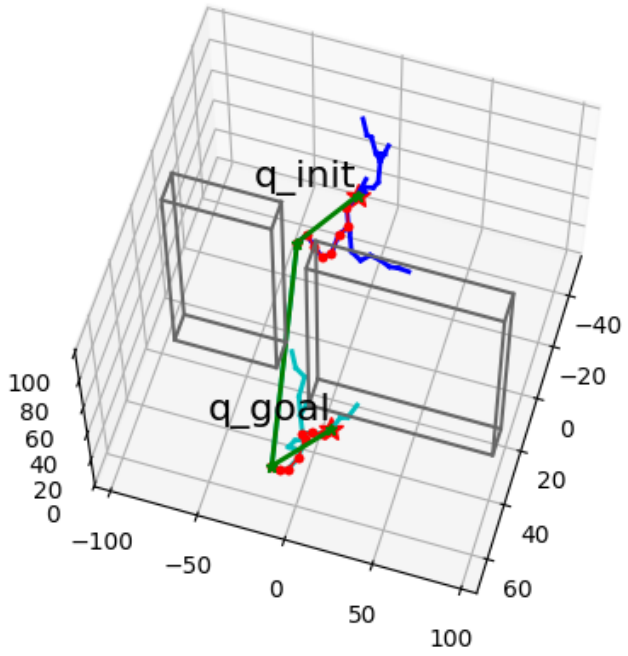

Figura 74 – Árvore RRT-Connect e caminho selecionado para experimento 2. Vista 1

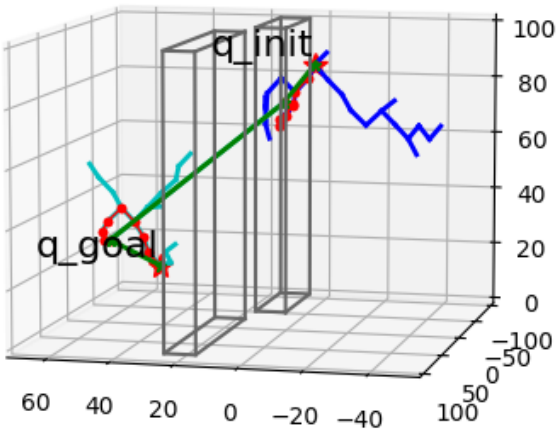

Figura 75 – Árvore RRT-Connect e caminho selecionado para experimento 2. Vista 2

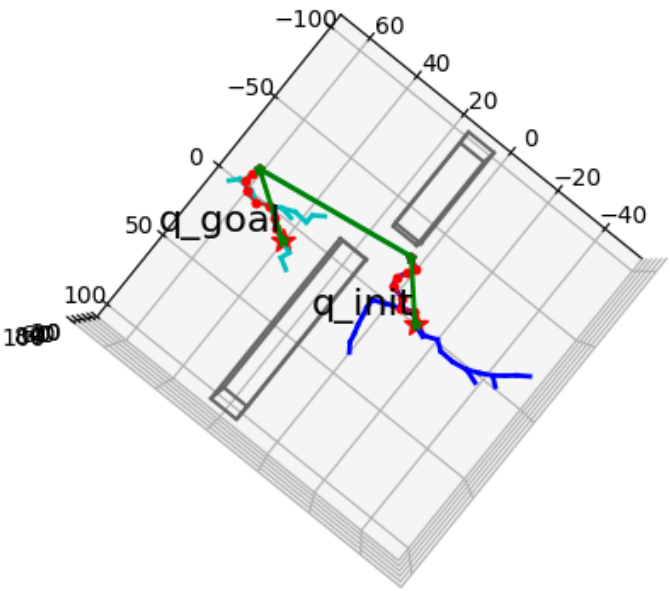

Figura 76 – Árvore RRT-Connect e caminho selecionado para experimento 2. Vista 3

Durante este experimento, também, foi notado uma grande recorrência da situação de escolha de caminhos não ótimos ou com rotas não intuitivas, como ilustrado nas figuras 77 e 78. Novamente o planejador seguindo um caminho que recuava em relação ao objetivo final, mas ainda possuindo uma árvore consideravelmente menor que o caso do RRT padrão.

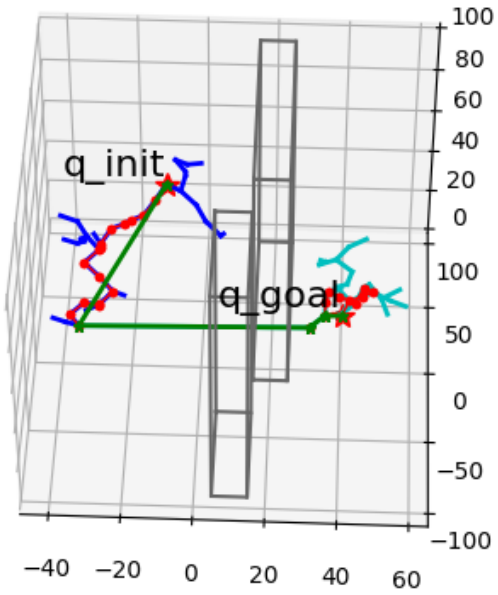

Figura 77 – Árvore RRT-Connect e caminho seguindo um caminho que recua em relação ao ponto destino. Vista 1

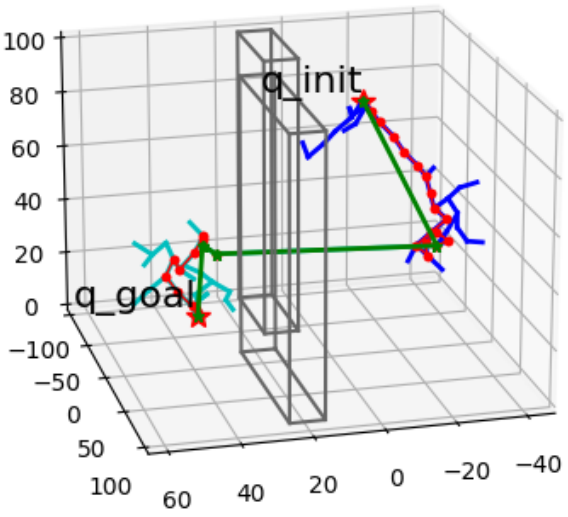

Figura 78 – Árvore RRT-Connect e caminho seguindo um caminho que recua em relação ao ponto destino. Vista 2

Seguindo para a simulação no CoppeliaSim, ilustrado nas figuras 79 e 80, foi possível verificar que novamente o planejador não obteve sucesso ao determinar a pose final para o manipulador. É possível perceber que, comparando ao caso do experimento 2 com RRT padrão, a movimentação inicial do robô seguiu um perfil mais próximo do esperado baseado no caminho escolhido, mas, ainda sim, não alcançou o final da trajetória como esperado.

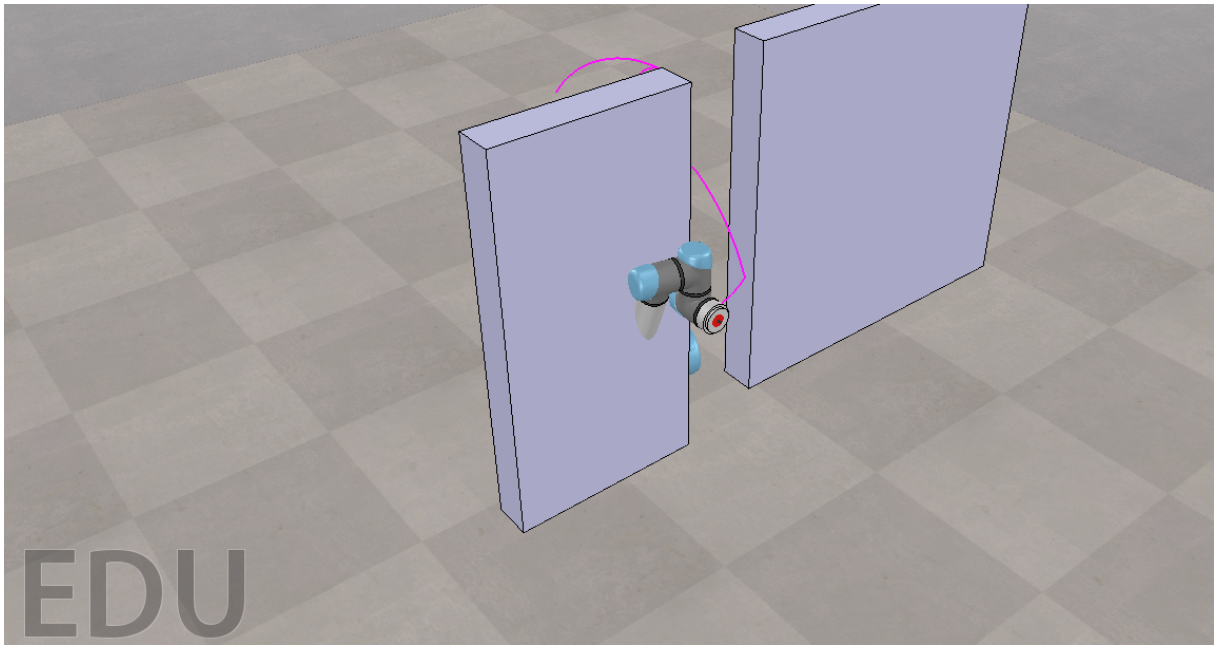

Figura 79 – Experimento 2 realizado com o planejamento de rota RRT-Connect dentro do CoppeliaSim. Instante 1

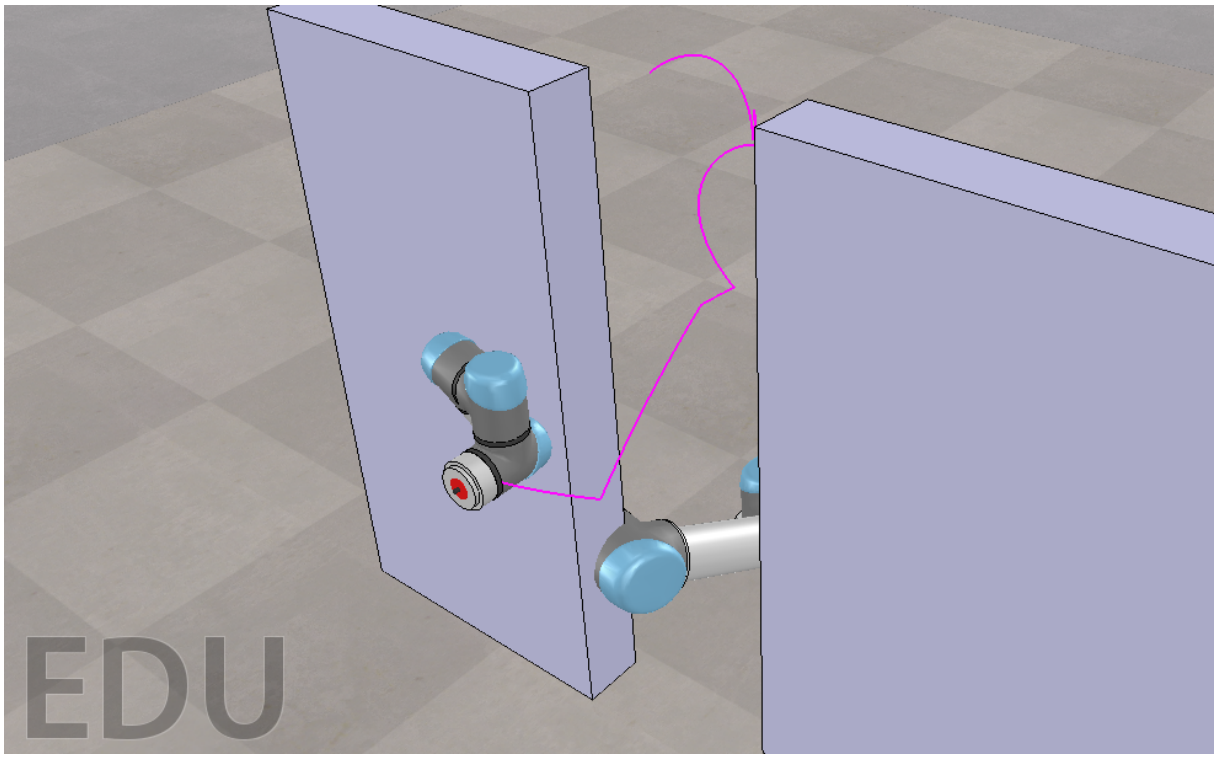

Figura 80 – Experimento 2 realizado com o planejamento de rota RRT-Connect dentro do CoppeliaSim. Instante 2

4.3 Experimento 3

Para o experimento final, foi decidido utilizar um ambiente que torna totalmente intuitivo e claro qual a trajetória deve ser seguida pelo robô. Sendo assim, como ilustrado nas

figuras 81 e 82, foi definido um ambiente com dois obstáculos posicionados de forma com que o robô consiga se movimentar em linha reta sem necessitar realizar grandes movimentações ou desvios.

Apesar de ser um caminho consideravelmente mais simples, acredita-se ser importante para validação do funcionamento dos algoritmos diferentes situações. Os obstáculos inseridos são idênticos, possuindo as medidas de $10x80x100$ centímetros e posicionados de forma a proporcionar um vão de 28 centímetros.

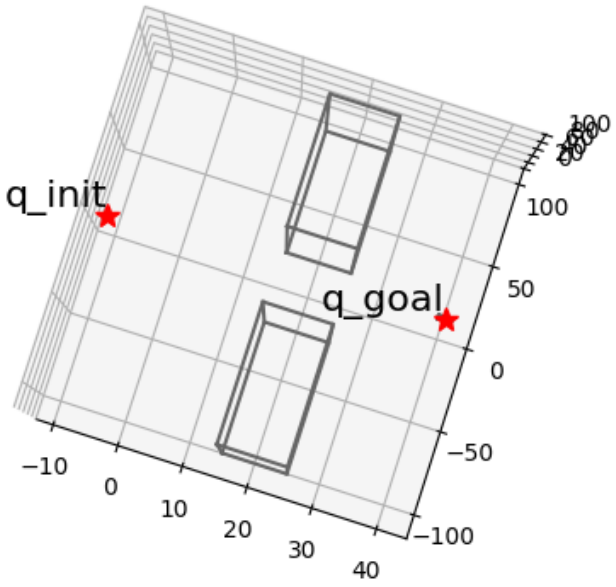

Figura 81 – Ambiente projetado para o Experimento 3. Vista 1

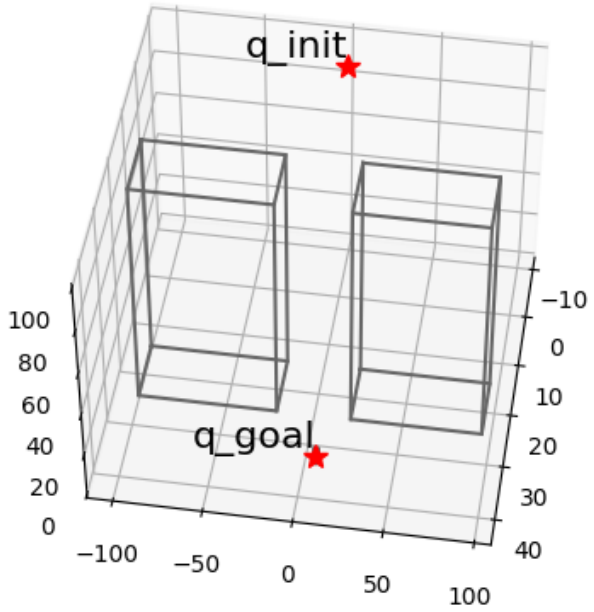

Figura 82 – Ambiente projetado para o Experimento 3. Vista 2

4.3.1 RRT

Seguindo as mesmas etapas, têm-se a Tabela 7 com as informações de desempenho verificadas para tal ambiente com execução do RRT. Percebe-se que foi necessário cerca de 74,4711 iterações e 0,1353 segundos de execução para que o RRT consiga finalizar o planejamento. Devido as condições proporcionadas no ambiente, têm-se que os resultados foram muito condizentes com o esperado.

Tabela 7 – Experimento 3 - Desempenho do RRT

RRT		
	Quantidade de Iterações	Tempo de execução
Média	74,4711	0,02103 s
Mediana	70	0,01093 s
Máximo	250	0,05899 s
Mínimo	40	0,00571 s

Ilustrado nas figuras 83, 84 e 85, é possível verificar o caminho escolhido para o experimento. Percebe-se que o algoritmo não teve grandes problemas para alcançar o objetivo, custando poucas iterações e nós para tal.

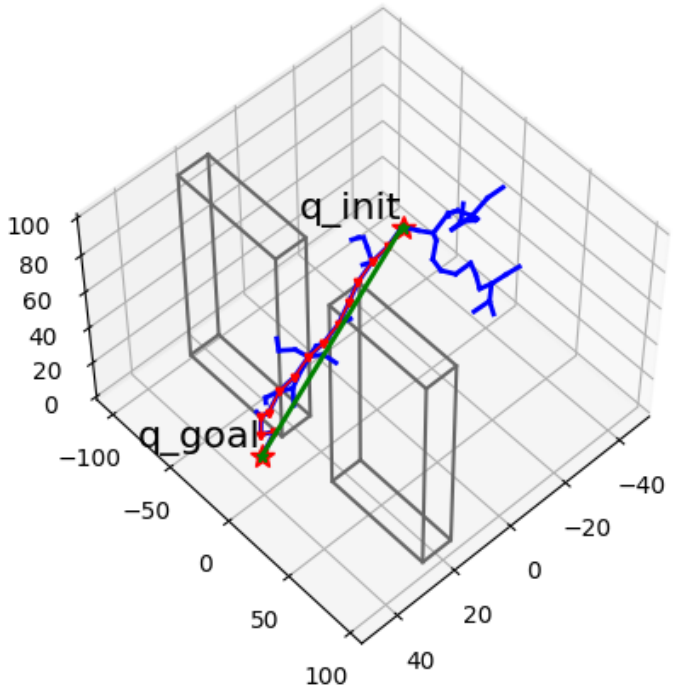

Figura 83 – Árvore RRT e caminho selecionado para experimento 3. Vista 1

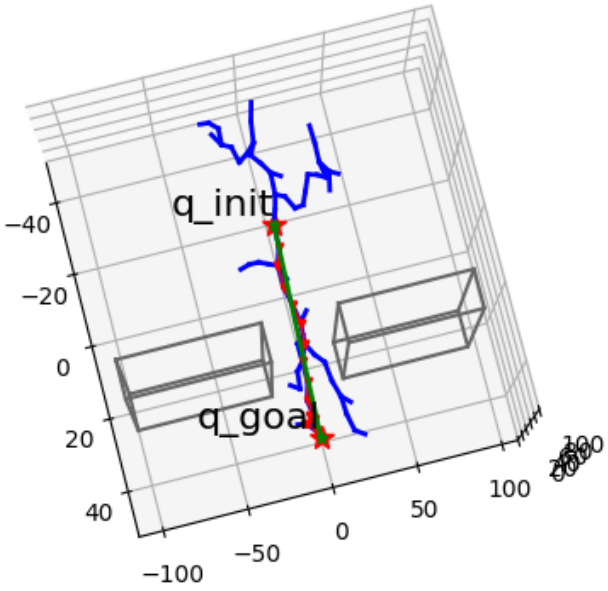

Figura 84 – Árvore RRT e caminho selecionado para experimento 3. Vista 2

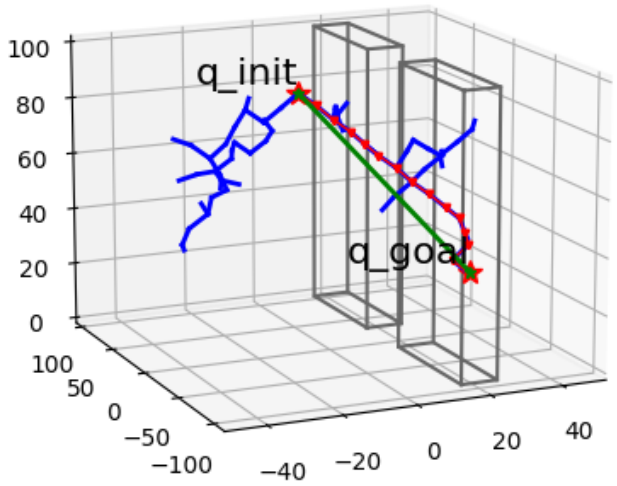

Figura 85 – Árvore RRT e caminho selecionado para experimento 3. Vista 3

Por fim, prosseguindo para a execução juntamente com o simulador CoppeliaSim, ilustrado nas figuras 86,87 e 88, é possível verificar que a movimentação realizada esteve dentro do esperado. O robô não se submeteu a percursos anormais e chegou ao destino com facilidade.

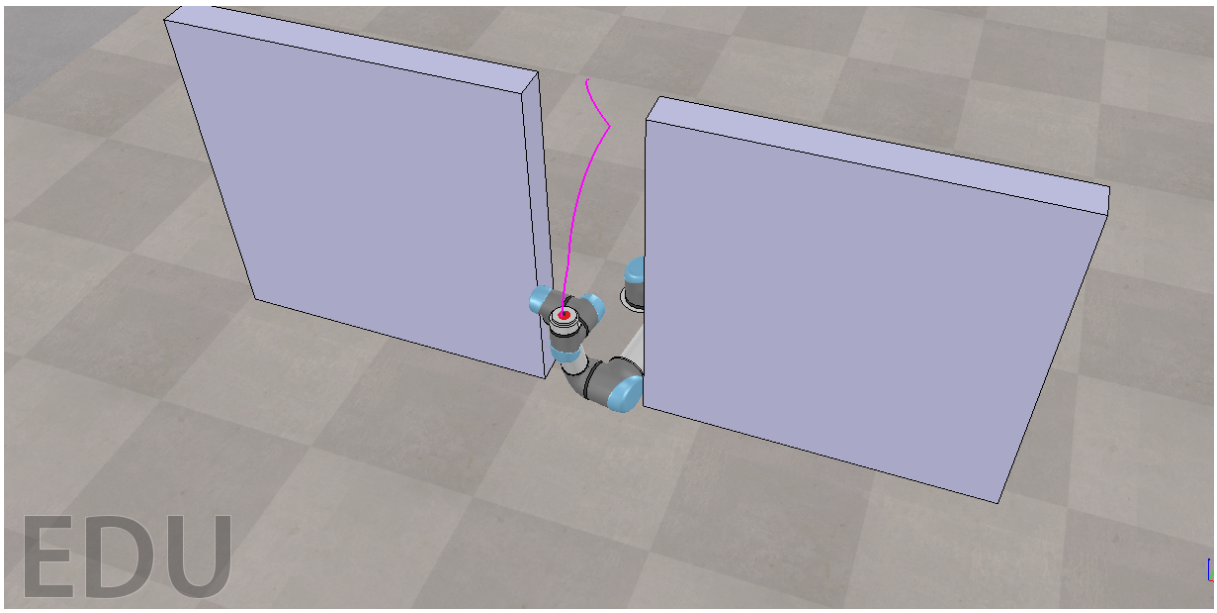

Figura 86 – Experimento 3 realizado com o planejamento de rota RRT dentro do CoppeliaSim. Instante 1

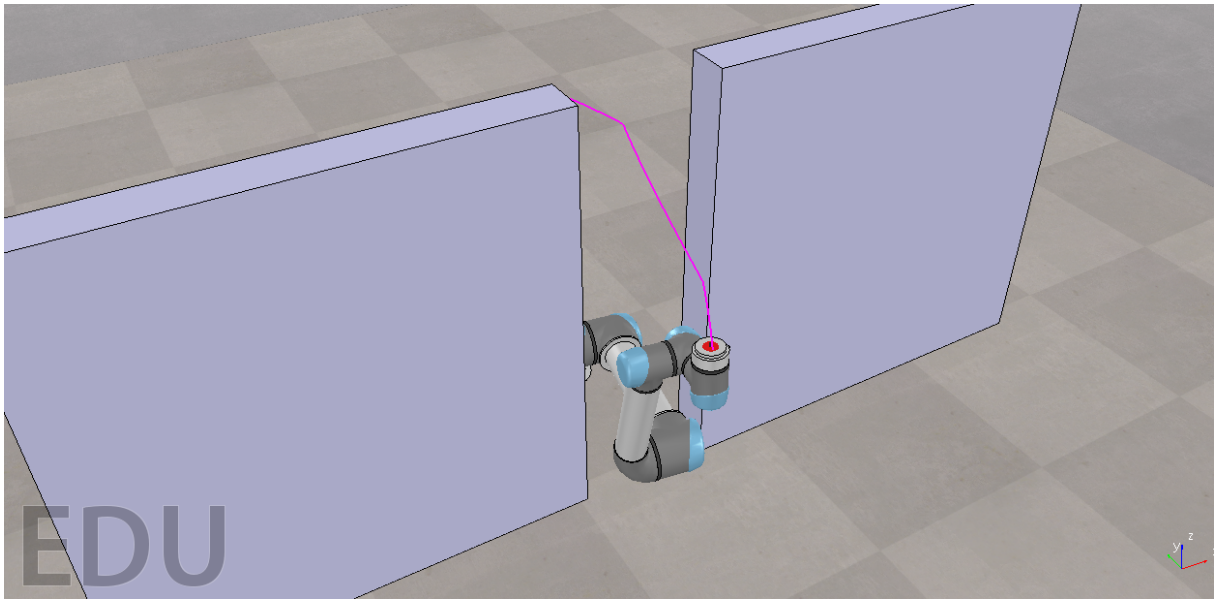

Figura 87 – Experimento 3 realizado com o planejamento de rota RRT dentro do CoppeliaSim. Instante 2

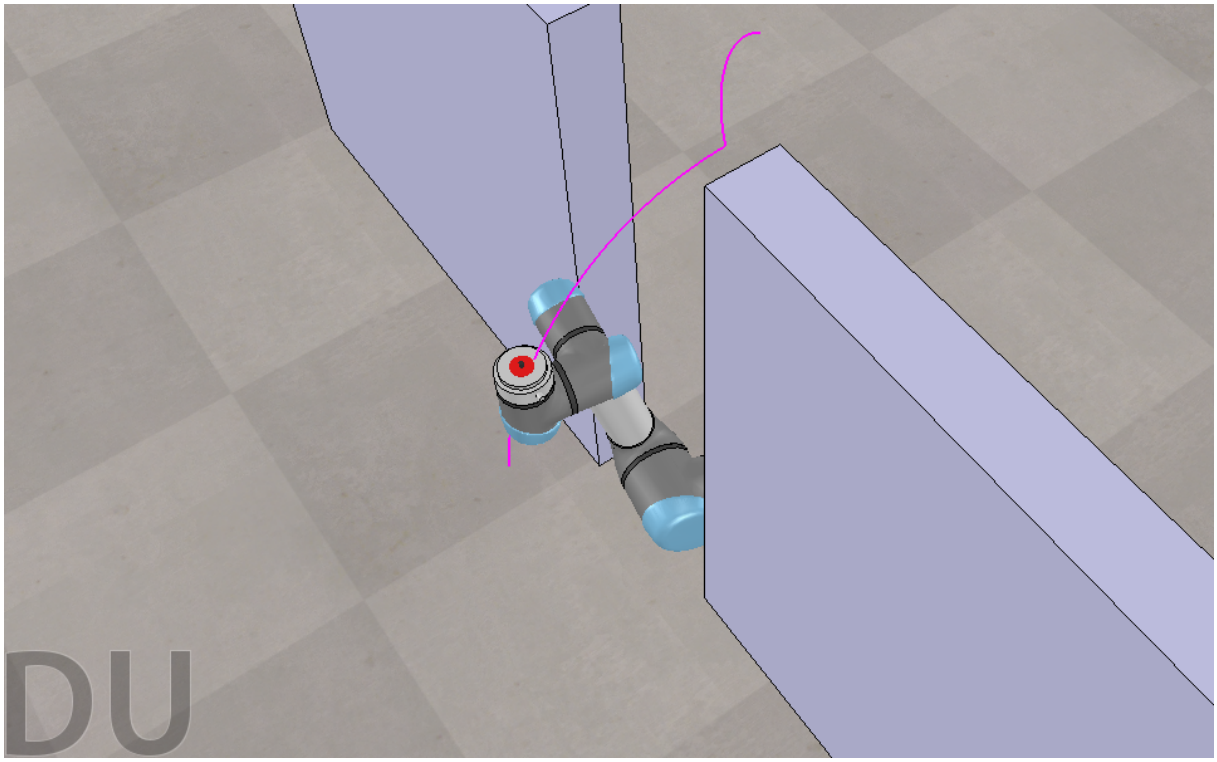

Figura 88 – Experimento 3 realizado com o planejamento de rota RRT dentro do CoppeliaSim. Instante 3

4.3.2 RRT-Connect

Novamente, realizando o mesmo experimento, mas agora para o RRT-Connect, foi obtido os resultados descritos na Tabela 8. Percebe-se que o RRT-Connect conseguiu alcançar

resultados de forma quase que instantânea. Precisando de cerca de 7 ou menos iterações para tal.

Tabela 8 – Experimento 3 - Desempenho do RRT-Connect

RRT-Connect		
	Quantidade de Iterações	Tempo de execução
Média	7,0671	0,0019 s
Mediana	5	0,0011 s
Máximo	90	0,0506 s
Mínimo	4	0,0009 s

Com o caminho selecionado para este experimento do RRT-Connect, ilustrado nas figuras 89 e 90, percebe-se que, assim como o citado anteriormente, o RRT-Connect alcança o resultado final de forma extremamente rápida, não deixando brecha nem mesmo para existência de algum nó que levaria à um trajeto pouco eficiente.

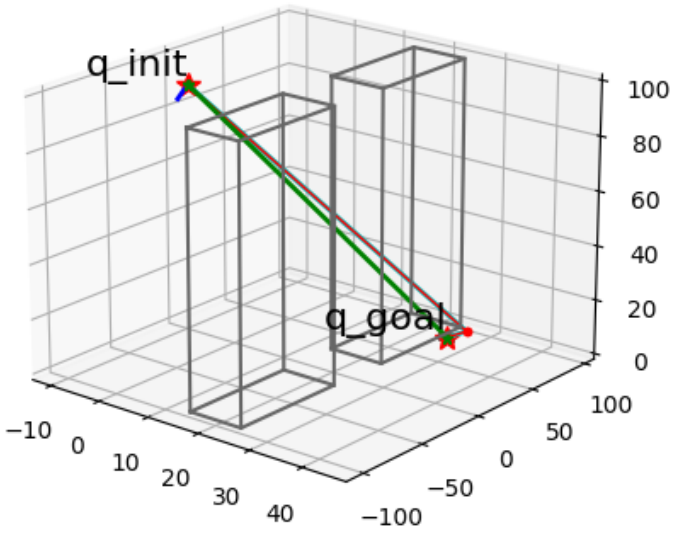

Figura 89 – Árvore RRT-Connect e caminho selecionado para experimento 3. Vista 1

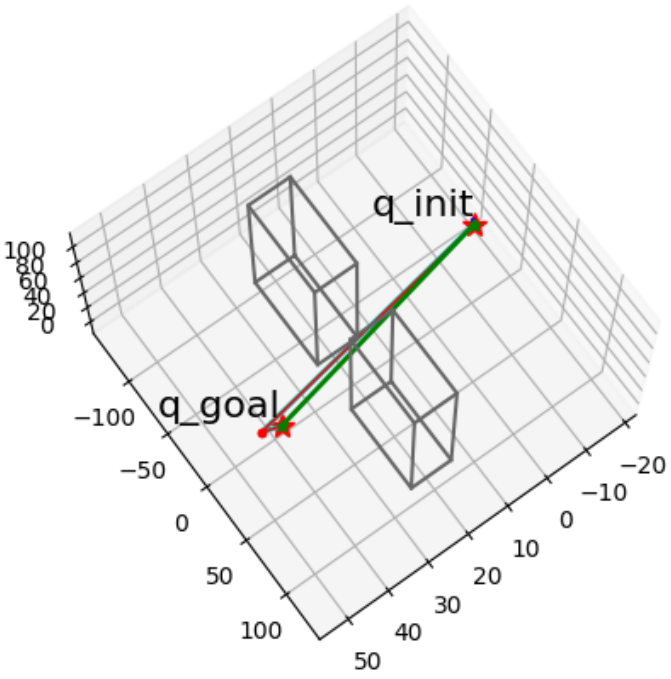

Figura 90 – Árvore RRT-Connect e caminho selecionado para experimento 3. Vista 2

Seguindo para a simulação no CoppeliaSim, ilustrado nas figuras 91 e 92, foi possível confirmar o que já se mostrou muito claro com os resultados anteriores. O robô conseguiu percorrer o trajeto especificado, sem prolemas com orientação ou trajetória inesperada.

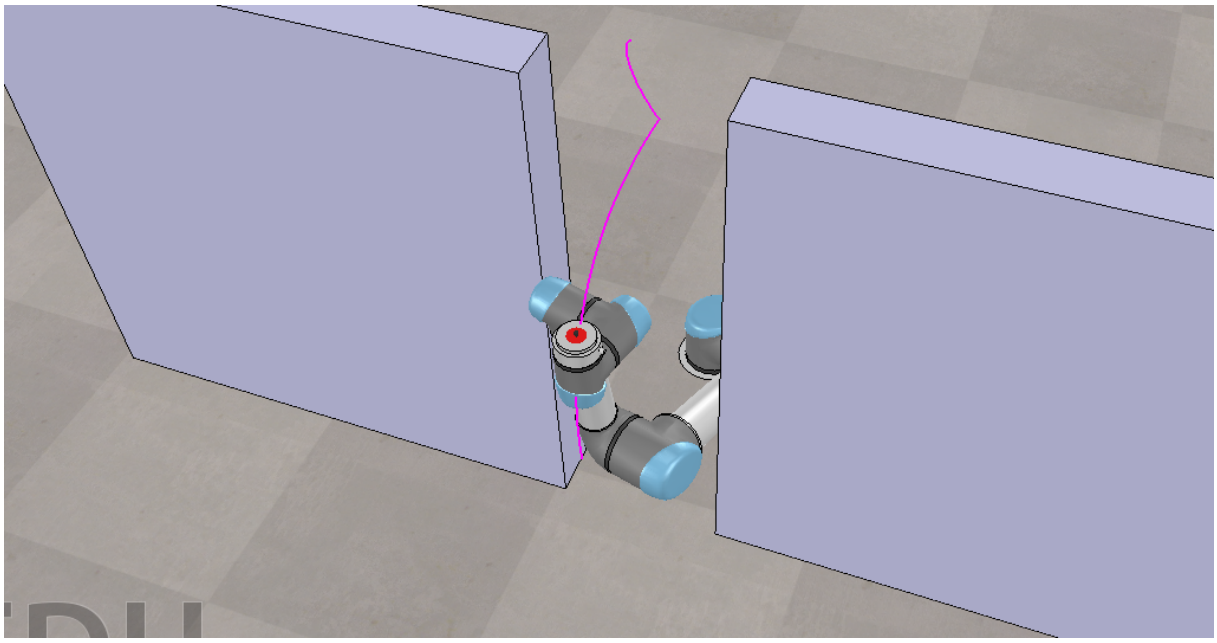

Figura 91 – Experimento 3 realizado com o planejamento de rota RRT-Connect dentro do CoppeliaSim. Instante 1

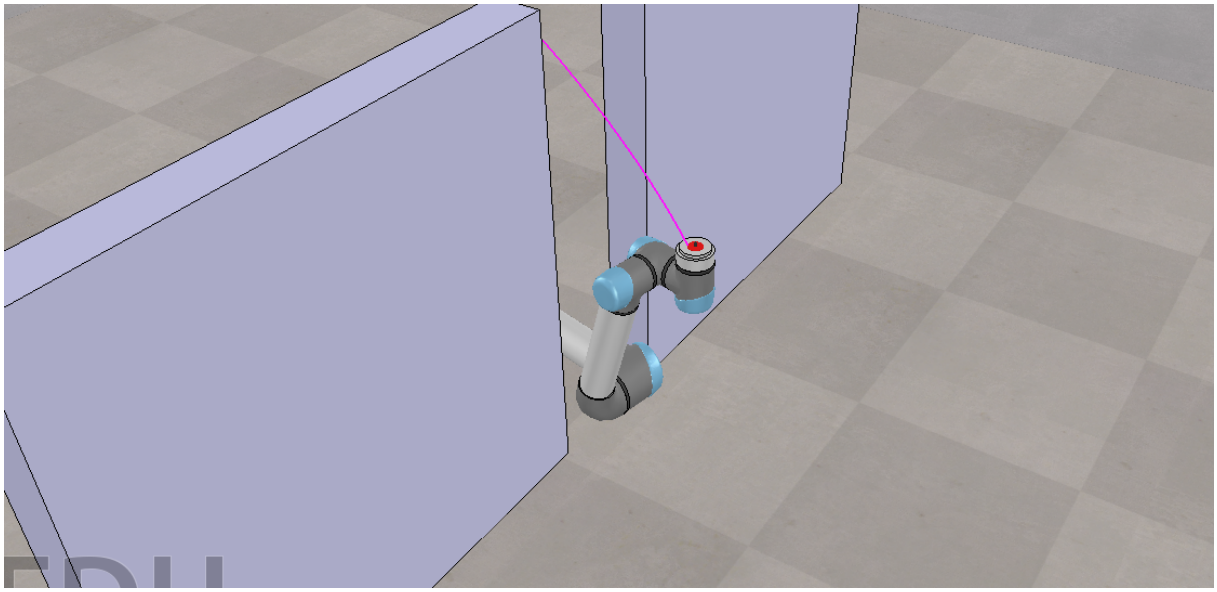

Figura 92 – Experimento 3 realizado com o planejamento de rota RRT-Connect dentro do CoppeliaSim. Instante 2

4.4 Considerações

A partir dos resultados alcançados pode-se pontuar certas características notadas ao longo dos experimentos.

Primeiramente, como citado em subseção 4.1.1, em diversas situações foram notados casos com certos valores atípicos, chamados de *outliers*, que podem ser julgados como importantes elementos para corroborar com os conceitos estudados e demostrados por toda bibliografia.

Tratando-se de modelos probabilísticos, é esperado que situações com resultados de valores inesperados ocorram num certo ponto que pode ser considerado comum. Ao longo dos experimentos realizados, foi notado inúmeras vezes situações em que as árvores dos algoritmos se expandiram de forma totalmente não favorável ao encontro com o local desejado. Situações onde foi necessário aguardar quase 5000 iterações foram vistas com certa frequência.

Realizando um comparativo direto sobre a eficiência entre os dois algoritmos abordados em cada experimento, como mostrado na Tabela 9, é possível verificar que em todas as situações experimentadas o algoritmo RRT-Connect conseguiu reduzir o tempo de execução e quantidade de iterações de forma muito expressiva. Em uma média, a melhora vista foi de 91,34% para o tempo de execução e 87,35% para quantidade de iterações.

Tabela 9 – Comparativo de desempenho utilizando valores de média extraídos em cada experimento

	RRT		RRT-Connect		RRT x RRT-Connect	
	Iterações	Δt	Iterações	Δt	Iterações	Δt
Experimento 1	755,9514	0,4353 s	38,538	0,0418 s	- 94,902%	- 90,397%
Experimento 2	307	0,1353 s	34,811	0,02611 s	- 88,611%	- 80,702%
Experimento 3	74,4711	0,02103 s	7,0671	0,0019 s	- 90,51%	- 90,965%

Δt : Tempo de execução

Por fim, vale citar as situações enfrentadas no experimento 2, onde não foi possível obter resultados satisfatórios para simulação no UR3. Acredita-se que melhorias devem ser aplicadas no projeto desenvolvido para que seja possível alcançar um nível de confiança no sistema a ponto de ser possível aplicação no robô real.

5 Conclusão

Nesse trabalho foi apresentado a implementação de um planejamento de rota para o robô manipulador UR3, baseado nos algoritmos RRT e RRT-Connect.

Para isso, foi realizado uma profunda pesquisa sobre o funcionamento de robôs dessa configuração, tanto quanto as conceituações definidas para modelagem e caracterização de sua cinemática.

Ao longo do desenvolvimento do planejador de rotas, foi feita uma minuciosa implementação dos algoritmos Rapidly-exploring Random Tree (RRT) e Rapidly-exploring Random Tree Connect (RRT-Connect). Com a implantação, etapa por etapa, desde planos 2D até planos 3D, foi possível consolidar e esclarecer muito bem cada nuance desse algoritmo que tanto impactou no avanço tecnológico de diversas áreas, como robótica móvel e de manipuladores e jogos eletrônicos, por exemplo.

Percebeu-se que ambos algoritmos conseguem desempenhar um bom resultado para o objetivo proposto. Podendo caracteriza-los como probabilisticamente completos, torna-se fácil adicionar restrições mais complexas, como grande número de obstáculos, sabendo que eventualmente um caminho será descoberto pela árvore gerada. Porém, percebe-se, também, que muita das vezes os trajetos determinados não são exatamente eficientes. Ocasionalmente, ambos algoritmos geram, de forma aleatória, caminhos que afastam-se demais do ponto de destino, desnecessariamente.

Pode-se dizer que em uma comparação direta entre os dois algoritmos abordados, cada um pode ser a melhor escolha dependendo do ambiente de aplicação. Caso esteja sendo buscado um mapeamento mais cuidadoso do ambiente, com objetivo de alcançar um resultado próximo do ótimo, o RRT seria o mais indicado pelo fato de que a construção da árvore RRT, na grande maioria das vezes, é representada com um número maior de amostras de posição no espaço.

Já no caso da RRT-Connect, seria mais indicado para situações em que é necessário um planejamento rápido, uma vez que, a partir das métricas utilizadas nos experimentos, foi possível verificar que o RRT-Connect consegue reduzir em mais de 90% o tempo necessário para planejamento de uma rota viável. Mas, também deve ser considerado que deve ser aplicado em situações que não necessite uma certa economia de recursos ou espaço para movimentação, uma vez que nem sempre será traçado o caminho ótimo dentro do espaço concedido.

Com a implementação do planejador, juntamente com ROS e visualização do comportamento no simulador CoppeliaSim, foi notado o desafio para, além de planejar a rota a

ser seguida para evitar colisões, mas também o desafio para planejar a movimentação do robô em si, envolvendo o comportamento das juntas ao longo de todo trajeto determinado.

Apesar de não ter sido possível realizar testes com o robô real, acredita-se que os resultados foram muito conclusivos e alcançaram o objetivo proposto. No geral, as simulações foram satisfatórias e demonstraram a efetividade do planejamento de rota do manipulador afim de evitar colisões, apesar de ter sido enfrentado situações onde a rota e a pose planejadas não se mostraram satisfatórias.

5.1 Perspectivas futuras

Um dos principais desafios enfrentados ao longo do desenvolvimento deste trabalho foi a reunião e compreensão das ferramentas capazes de realizar as tarefas desejadas. Muitas das aplicações existentes, por motivos claramente conhecidos, não difundem o conhecimento abertamente. Sendo necessário tentar compreender certos aplicações apenas com uma visão de externa e totalmente teórica.

Os trabalhos futuros poderão focar em realizar melhorias no sistema desenvolvido neste projeto. Ajustes no código poderão ser realizados a fim de garantir uma melhor definição das poses que serão seguidas ao longo de toda trajetória planejada pelo RRT ou RRT-Connect. Sendo assim, acredita-se que a implementação de módulos para determinação da pose do robô seria o melhor caminho para garantir resultados ainda mais eficientes.

Por fim, como continuação direta do trabalho, assim como sequência para as etapas descritas na definição do problema e abordagem seguida, tem-se como prosseguimento natural a aplicação do projeto desenvolvido no robô real.

Seguindo na mesma área de atuação e almejando aplicações que atendam questões ainda mais especificas no mercado e industria, sugere-se ainda:

- Implementação do planejamento de rota para uma tarefa de interação com objetos no ambiente (Pick and Place);
- Implementação de percepção do ambiente em que o UR3 está inserido utilizando visão computacional;

Referências

AURENHAMMER, F. Voronoi diagrams—a survey of a fundamental geometric data structure. **ACM Comput. Surv.**, v. 23, p. 345–405, 1991. Citado nas pp. 32–34.

CARRARA, V. **Introdução à robótica industrial**. São José dos Campos: INPE, out. 2015. Citado nas pp. 27, 28.

CHATILA, R. Path Planning and Environment Learning in a Mobile Robot System. In: ECAI. 1982. Citado na p. 31.

COPPELIA, R. **CoppeliaSimm**. 2022. Acessado em 10/01/2022. Disponível em: <https://www.coppeliarobotics.com/>. Citado na p. 24.

CORKE, P. **Robotics, Vision and Control - Fundamental Algorithms in MATLAB®**. Springer, 2011. v. 73, p. 1–495. (Springer Tracts in Advanced Robotics). ISBN 978-3-642-20143-1. Disponível em: <http://dblp.uni-trier.de/db/series/star/index.html#Corke11>. Citado nas pp. 32, 34, 36, 46.

CORKE, P. **Robotics, Vision and Control: Fundamental Algorithms in MATLAB**. 1st: Springer Publishing Company, Incorporated, 2013. ISBN 3642201431. Citado nas pp. 24, 30, 33, 34, 39, 46.

CRAIG, 1. **Introduction to robotics : mechanics & control / John J. Craig.** Reading, Mass.: Addison-Wesley Pub. Co., 2012. Includes bibliographies and index. ISBN 0201103265. Citado na p. 27.

HSU, D.; LATOMBE, J.-C.; KURNIAWATI, H. On the Probabilistic Foundations of Probabilistic Roadmap Planning. **The International Journal of Robotics Research**, v. 25, n. 7, p. 627–643, 2006. DOI: 10.1177/0278364906067174. eprint: https://doi.org/10.1177/0278364906067174. Disponível em: <https://doi.org/10.1177/0278364906067174>. Citado nas pp. 35, 36.

INDUSTRIAL, R. **ROS-Industrial Members**. 2022. Acessado em 02/01/2022. Disponível em: <https://rosindustrial.org/ric/current-members>. Citado na p. 22.

KAUFER, M. **WebRRT - This is a web-based visualization of the rapidly-exploring random tree algorithm (RRT)**. 2017. https://github.com/mjkaufer/WebRRT. Acessado em 09/03/2022. Citado na p. 38.

KAVRAKI, L.; SVESTKA, P.; LATOMBE, J.-C.; OVERMARS, M. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. **IEEE Transactions on Robotics and Automation**, v. 12, n. 4, p. 566–580, 1996. DOI: 10.1109/70.508439. Citado na p. 35.

KAWASAKI, R. **A Project to Make Japan's First Domestically Manufactured Industrial Robot Begins**. 2000. Acessado em 25/01/2022. Disponível em: <https://robotics.kawasaki.com/en1/anniversary/history/history_02.html>. Citado na p. 16.

KHATIB, O. Real-time obstacle avoidance for manipulators and mobile robots. **Int. J. Robot. Res.**, v. 5, n. 1, p. 90–98, 1986. Citado na p. 30.

KLEINBORT, M.; SOLOVEY, K.; LITTLEFIELD, Z.; BEKRIS, K. E.; HALPERIN, D. Probabilistic Completeness of RRT for Geometric and Kinodynamic Planning With Forward Propagation. **IEEE Robotics and Automation Letters**, v. 4, n. 2, p. x–xvi, 2019. DOI: 10.1109/LRA.2018.2888947. Citado na p. 35.

KUFFNER, J.; LAVALLE, S. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In: v. 2, p. 995–1001. DOI: 10.1109/ROBOT.2000.844730. Citado na p. 40.

LATOMBE, J.-C. **Robot Motion Planning**. USA: Kluwer Academic Publishers, 1991. ISBN 0792391292. Citado nas pp. 30, 32, 34, 36.

LAVALLE, S. M. **Planning Algorithms**. Cambridge, U.K.: Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/. Citado nas pp. 37–39, 41.

LAVALLE, S. M. **Rapidly-Exploring Random Trees: A New Tool for Path Planning**. 1998. Citado nas pp. 37, 48.

LAVALLE, S. M.; KUFFNER, J. J. Rapidly-Exploring Random Trees: Progress and Prospects. In: ALGORITHMIC and Computational Robotics: New Directions. 2000. P. 293–308. Citado nas pp. 18, 37, 39–41.

MIT16.485. **ROS - File System**. 2022. Acessado em 18/03/2022. Disponível em: <https://vnav.mit.edu/labs/lab2/ros101.html>. Citado na p. 23.

NASCIMENTO, L.; GABOARDI DOS SANTOS, V.; PEREIRA, D.; FERNANDES, D.; ALSINA, P. Planejamento de Caminho para Sistemas Robóticos Autônomos. In: mar. 2020. P. 69–89. ISBN 978-85-7669-489-2. DOI: 10.5753/sbc.4389.2.4. Citado nas pp. 32, 33.

NOREEN, I.; KHAN, A.; HABIB, Z. Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions. **International Journal of Advanced Computer Science and Applications**, v. 7, 2016. Citado na p. 40.

OPENROBOTICS. **rospy**. 2017. Acessado em 28/03/2022. Disponível em: <http://wiki.ros.org/rospy>. Citado na p. 64.

ROBOTICS Toolbox. 2017. https://petercorke.com/toolboxes/robotics-toolbox/. Acessado em 05/01/2022. Citado nas pp. 24, 43.

ROBOTICS Toolbox for Python. 2022. Acessado em 20/02/2022. Disponível em: <https://petercorke.github.io/robotics-toolbox-python/>. Citado na p. 44.

ROS. **ROS - Robot Operating System**. 2022. Acessado em 02/01/2022. Disponível em: <https://www.ros.org/>. Citado na p. 22.

ROSARIO, J. M. **Robótica Industrial**. Ed. Baraúna, São Paulo, 2010. Citado nas pp. 21, 26, 27.

ŠEDA, M. Roadmap Methods vs. Cell Decomposition in Robot Motion Planning. In: PROCEEDINGS of the 6th WSEAS International Conference on Signal Processing, Robotics and Automation. Corfu Island, Greece: World Scientific, Engineering Academy e Society (WSEAS), 2007. (ISPRA'07), p. 127–132. ISBN 1237890123456. Citado nas pp. 31, 32.

SPONG, M.; HUTCHINSON, S.; VIDYASAGAR, M. Robot modeling and control. English (US). **IEEE Control Systems**, Institute of Electrical e Electronics Engineers Inc., v. 26, n. 6, p. 113–115, dez. 2006. ISSN 1066-033X. DOI: 10.1109/MCS.2006.252815. Citado na p. 34.

TONETTO, C. P. **Uma proposta de sistematização do processo de planejamento de trajetórias para o desenvolvimento da tarefas de robôs manipuladores.** 2007. Dissertação de Mestrado – Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia Mecânica. Citado na p. 35.

UBIRATAN DE M. PINTO JR.; MARIA P. CARVALHO; ANDRE G. S. CONCEIÇÃO. Campos Potenciais Artificiais aplicado ao planejamento de trajetórias do braço robótico JACO. **Universidade Federal da Bahia - UFBA**, 2020. Citado na p. 31.

UNIVERSAL Robotics. 2017. https://www.universal-robots.com/br/. Acessado em 05/01/2022. Citado nas pp. 17–20.

UR3/CB3 User Manual. 2021. Acessado em 05/01/2022. Disponível em: <https://www.usna.edu/Users/weaprcon/kutzer/_files/documents/User%5C%20Manual,%5C%20UR3.pdf>. Citado nas pp. 21, 22.

ZEID, I. **CAD/CAM theory and practice**. McGraw-Hill Higher Education, 1991. Citado na p. 38.