UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA

LUIZ FELIPE SABBAGH DE ALMEIDA SANTOS

PLANEJAMENTO DE TRAJETÓRIAS DE ROBÔS COM CURVAS DE BÉZIER E SPLINES EM ROS/MOVEIT!

Joinville
2022

LUIZ FELIPE SABBAGH DE ALMEIDA SANTOS

PLANEJAMENTO DE TRAJETÓRIAS DE ROBÔS COM CURVAS DE BÉZIER E SPLINES EM ROS/MOVEIT!

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de Bacharel em Engenharia Mecatrônica, no curso Engenharia Mecatrônica da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Orientador: Prof. Dr. Roberto Simoni

Joinville
2022

AGRADECIMENTOS

Agradeço à todos os meus familiares, em especial, aos meus pais, Luiz Carlos e Claudia, por todo o suporte emocional e financeiro que tive durante toda a minha vida, principalmente durante a graduação, e ao meu irmão, Bernardo.
Agradeço aos meus colegas, em especial, os que estiveram comigo no Eficem e no DALEM.
Agradeço aos meus professores, em especial ao meu orientador Dr. Roberto Simoni, por todos os ensinamentos durante a graduação.
Agradeço a todos que ajudam, diariamente, a construir a UFSC: alunos, professores, TAEs e servidores terceirizados. Todos são fundamentais na qualidade de nossa Universidade.
Agradeço à toda sociedade brasileira por ter me permitido estudar em uma instituição pública, gratuita e de qualidade.

"O verdadeiro perigo não é que computadores começarão a pensar como homens, mas que homens começarão a pensar como computadores."
Sydney J. Harris

RESUMO

O uso de robôs industriais é parte fundamental da manufatura no século XXI. Dentre os robôs utilizados nas indústrias, se encontram os robôs manipuladores. Esta classe de robôs é importante para a realização de tarefas como movimentação de peças, paletização, pintura e soldagem. Nesses contextos, os manipuladores precisam realizar movimentos precisos e suaves, buscando atingir confiabilidade na realização das tarefas, não causar danos aos objetos manipulados e ao próprio robô. Para a realização de movimentos que cumpram esses requisitos são utilizados curvas e superfícies. Este trabalho apresenta a implementação de curvas do tipo B-Spline e Bézier para a simulação da trajetória de um robô manipulador utilizando os softwares ROS e MoveIt!.

Palavras-chave: ROS. MoveIt!. Planejamento de Trajetória. Splines. Curvas de Bezier.

ABSTRACT

The use of industrial robots is a fundamental part of manufacturing in the 21st century. Among the robots used in industries, there are the manipulators robots. This class of robots is important for performing tasks such as moving parts, palletizing, painting and welding. In these contexts, manipulators need to perform precise and smooth movements, seeking to achieve reliability in carrying out the tasks, not causing damage to the manipulated objects and to the robot itself. To perform movements that achieve these requirements, curves and surfaces are used. This work presents the implementation of B-Splines and Bézier curves for simulating the trajectory of a manipulator robot using ROS and MoveIt!.

Keywords: ROS. MoveIt!. Path Planning. B-Splines. Bézier Curves.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ASCII American Standard Code for Information Interchange

AUV Autonomous underwater vehicle

CAD Computer-Aided Design

CHOMP Covariant Hamiltonian Optimization for Motion Planning

IFR International Federation of Robotics

IGES Initial Graphics Exchange Specification

NURBS Non-Uniform Rational B-Splines

OMPL Open Motion Planning Library

ROS Robot Operating System

SCARA Selective Compliance Assembly Robot Arm

STEP Standard for the Exchange of Product model data

UGV Unmanned Ground Vehicle

URDF Unified Robot Description Format

LISTA DE SÍMBOLOS

q_i	Grau de Liberdade
n	Número de pontos
θ	Posição
$\dot{\theta}$	Velocidade
$\ddot{\theta}$	Aceleração
t	Tempo
a	Coeficiente de uma spline
u_i	Nós de uma B-Spline
U	Vetor de nós de uma B-Spline
N	Função de base de uma B-Spline
P_i	Pontos de controle
B	Função de base de uma Curva de Bézier

SUMÁRIO

1 INTRODUÇÃO

O termo robô apareceu pela primeira vez em uma peça de teatro do escritor checo Karel Čapek em 1920. Desde então, a robótica evoluiu dos livros de ficção científica de Isaac Asimov para máquinas de importância vital na sociedade moderna.

Hoje, robôs são indispensáveis em diversas indústrias, como a automotiva, a aeroespacial e a médica, além da recente popularização de dispositivos robóticos como eletrodomésticos. De acordo com dados da International Federation of Robotics (IFR), o número de robôs industriais em operação no mundo ao final de 2020 passava de 2,7 milhões de unidades, sendo cerca de 15 mil robôs no Brasil (INTERNATIONAL FEDERATION OF ROBOTICS, 2020).

Uma subclasse dos robôs são os robôs manipuladores. Para Craig (2012, p.4) "Os manipuladores consistem em elos quase rígidos que são conectados por juntas, as quais permitem o movimento relativo dos elos vizinhos.". Ao final da cadeia de elos, fixa-se um efetuador, que varia dependendo da aplicação do robô. O efetuador pode ser uma garra, uma tocha de solda ou outros dispositivos.

Robôs manipuladores são comumente utilizados na área de fabricação, podendo ser utilizados em processos de paletização, montagem, pintura, soldagem e movimentação de peças. Nesse contexto, os robôs precisam de uma trajetória suave, previsível e precisa, buscando realizar com sucesso suas funções, evitando danos aos materiais manipulados, riscos aos humanos envolvidos no processo e danos ao próprio manipulador (CRAIG, 2012).

No estudo da robótica, ferramentas como o Robot Operating System (ROS) são bastante populares para simulações, análise de colisões e cálculos de cinemática. Ao permitir a integração com sensores e permitir a simulação da interação do robô com o ambiente, o ROS pode ser utilizado para diversos tipos de aplicações, como a simulação de robôs autônomos, mostrada em Gatesichapakorn, Takamatsu e Ruchanurucks (2019), e o desenvolvimento de software para sistemas robóticos, como mostrado em Barth et al. (2014), onde o ROS é utilizado no desenvolvimento de robôs voltados à agricultura, sendo utilizado em sistemas de detecção, percepção, controle do efetuador final, controle de missão e da estrutura do sistema como um todo.

Para o cálculo de trajetórias cartesianas de manipuladores são utilizados softwares especializados que realizam o cálculo de cinemática inversa e analisam a interação do manipulador com o ambiente, como forma de evitar colisões. Neste contexto, o ROS junto ao plugin MoveIt! são bastante populares para a realização de cálculos de trajetórias e simulações.

Em Deng, Xiong e Xia (2017), o ROS é utilizado junto ao MoveIt! para criar a

trajetória de um manipulador móvel, utilizando um algoritmo criado pelos autores para o cálculo da cinemática inversa do manipulador. A possibilidade do uso de robôs manipuladores móveis na área da mineração, com trajetórias de movimentação calculadas usando o MoveIt! é mostrada em Cota et al. (2017), onde os robôs manipuladores podem ser usados como forma de substituir um ser humano ao posicionar e carregar os explosivos, trazendo mais segurança para as operações.

Duarte (2020) faz a movimentação de um robô antropomórfico, utilizando simulações de trajetória calculadas com auxílio do ROS e do MoveIt!, porém as trajetórias não são realizadas de forma suave e precisa, portanto, faz-se necessário o uso de cálculos numéricos para a melhoria desses quesitos.

De acordo com Craig (2012), para fazer com que o manipulador se movimente de forma suave e controlada, cada junta deve se movimentar conforme especificado por uma função suave de tempo. O problema do planejamento de trajetórias surge na maneira de calcular essas funções de movimento das juntas.

Para contornar os problemas causados pelo uso de trajetórias que não são contínuas em velocidade e aceleração, Tyapin e Hovland (2018) e Ferreira (2011) propõem em seus trabalhos o uso de curvas B-Spline para o planejamento das trajetórias de um manipulador robótico. O uso de curvas de Bézier para evitar singularidades entre os pontos, onde posição ou velocidade ou aceleração não são contínuas, é proposto em Rybus et al. (2013).

Utilizando uma pesquisa do tipo experimental, o presente trabalho tem como objetivo utilizar as nuvens de pontos geradas em Duarte (2020), implementar algoritmos para cálculos de B-Splines e curvas de Bézier para a otimização das trajetórias do manipulador e simular as trajetórias obtidas utilizando os softwares ROS e Moveit.

1.1 OBJETIVO GERAL

O objetivo geral deste trabalho é o desenvolvimento de algoritmos de planejamento de trajetória utilizando curvas B-Spline e curvas de Bézier. Após a implementação dos algoritmos, realizar-se simulações utilizando os softwares ROS e MoveIt! com as trajetórias geradas e o pacote ArmAgeddon desenvolvido por Duarte (2020).

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos desse trabalhos são:

a. Realizar estudos sobre o uso de curvas B-Spline e curvas de Bézier no planejamento de trajetórias;
b. Realizar estudos sobre o uso de ROS e MoveIt! no planejamento de trajetórias;

c. Desenvolver algoritmos que utilizem B-Splines e curvas de Bézier para o planejamento de trajetória de um robô antropomórfico;
d. Criar simulações com o ROS e o MoveIt! utilizando os caminhos gerados com Curvas de Bézier e B-Splines;
e. Utilizar os algoritmos de planejamento de trajetória para simulações baseadas em casos reais;

2 REVISÃO BIBLIOGRÁFICA

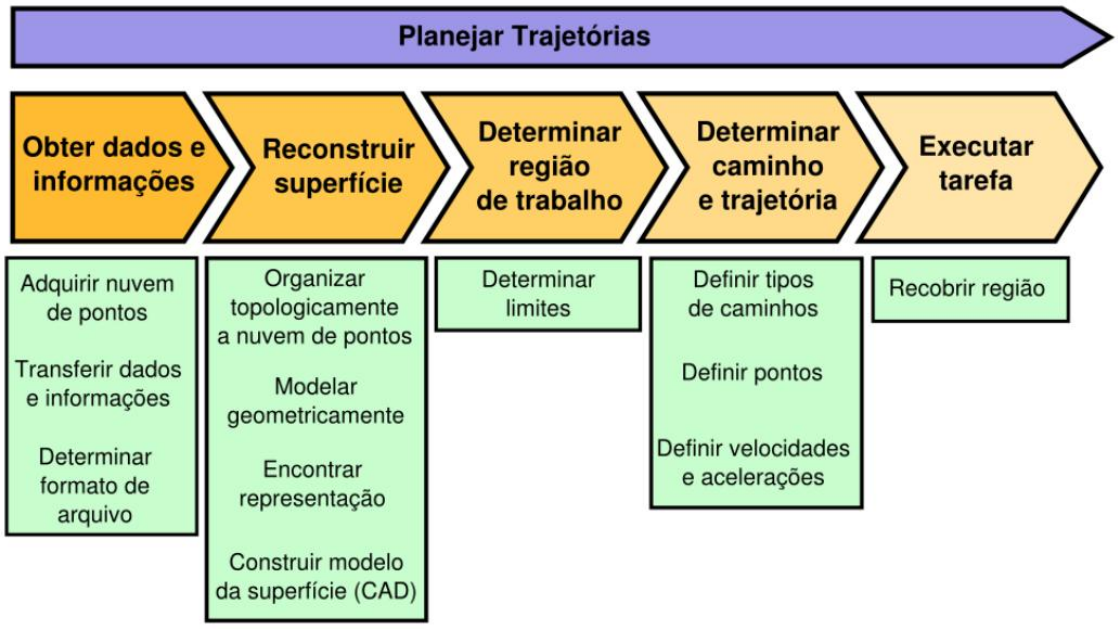
O planejamento de trajetórias é uma área bastante pesquisada dentro da robótica devido a importância dos robôs efetuarem trajetórias eficientes e que não causem prejuízos para os objetos que estão sendo manipulados e ao próprio manipulador, além de garantir a segurança de operadores humanos. Este capítulo mostra alguns trabalhos realizados na área, com o uso de splines e curvas de Bézier e a utiliação de ROS e MoveIt!.

2.1 PLANEJAMENTO DE TRAJETÓRIAS

Dias et al. (2000) mostram em seu trabalho a utilização de curvas B-spline para a geração de trajetórias em uma superfície parametrizada no espaço cartesiano, utilizando um manipulador do tipo Selective Compliance Assembly Robot Arm (SCARA) para a experimentação das trajetórias obtidas através dos cálculos.

Uma proposta de sistematização do processo de planejamento de trajetórias é proposta por Tonetto (2007). A autora propõe dividir o processo de planejamento de uma trajetória em cinco etapas, que estão apresentadas na Figura 1.

Figura 1 – Proposta para planejamento de trajetórias



Fonte: Tonetto (2007, p. 52).

A primeira etapa consiste na maneira de obter as informações da região de interesse, processo que pode ser feito com a utilização de uma máquina de medição

de coordenadas, varredura a laser ou outras técnicas. Nesse processo, as informações devem ser organizadas de maneira padronizada, utilizando arquivos baseados em padrões universais, como Initial Graphics Exchange Specification (IGES), Standard for the Exchange of Product model data (STEP) ou American Standard Code for Information Interchange (ASCII).

Com as informações obtidas na primeira etapa, é possível reconstruir a superfície para gerar um modelo virtual, utilizando uma representação topológica da região, que pode ser obtida com técnicas como nuvem de pontos, malha triangularizada e a reconstrução utilizando superfícies B-Spline. Também é possível utilizar um sistema CAD para realizar a reconstrução da superfície desejada.

Na terceira etapa, a região de trabalho é obtida partindo da superfície construída nas etapas anteriores, geralmente sendo definida uma região de interesse na superfície, já que, geralmente, há uma margem de segurança quando é feita a coleta de dados da superfície.

A determinação da trajetória é feita utilizando pontos e define o caminho que será percorrido pelo manipulador para cobrir a região de interesse. Nessa etapa também ocorre a determinação da forma de recobrimento da superfície, podendo ser com movimentos de zig-zag, espiral ou uma movimentação livre que passa por pontos já determinados. Nesta etapa, também podem ocorrer as definições de velocidade e aceleração do efetuador final.

Após a determinação do caminho, as informações são repassadas para a máquina e a tarefa é executada pelo manipulador, também podendo ocorrer simulações da tarefa antes da realização da tarefa de fato.

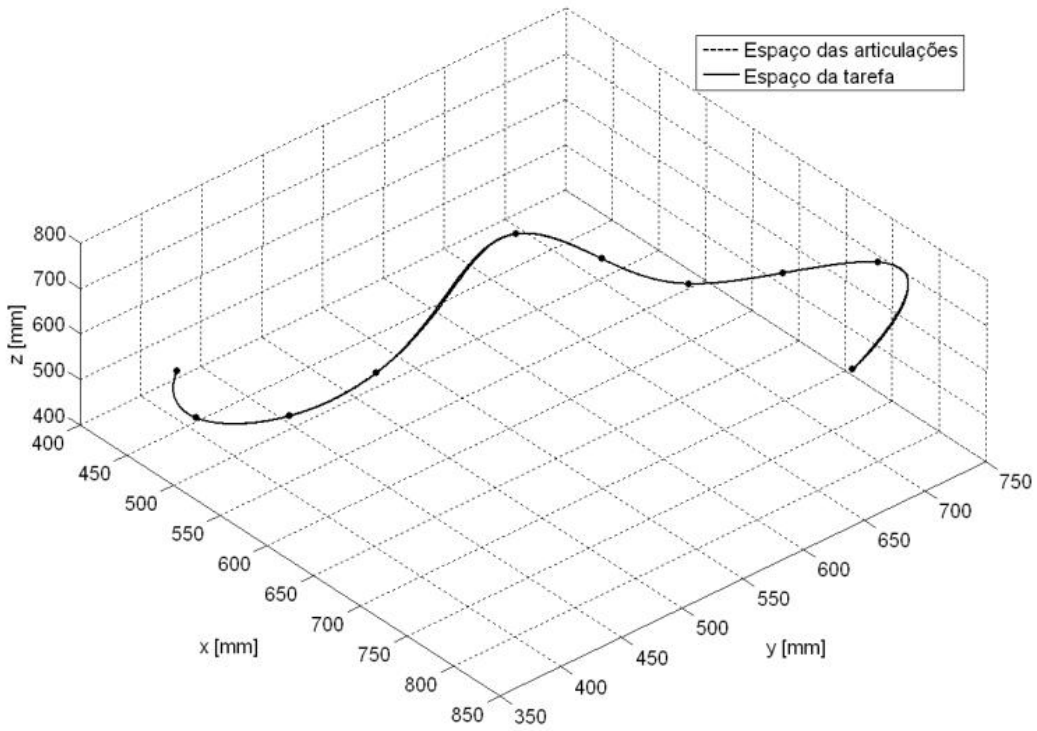
Este trabalho foca na quarta etapa do processo sistematizado por Tonetto (2007), a determinação do caminho e da trajetória. Fazendo a determinação dos pontos de interesse, o cálculo dos caminhos, com a utilização de B-Splines e Curvas de Bézier, e a definição das velocidades e acelerações das juntas através da simulação com os softwares ROS e MoveIt!.

Ferreira (2011) apresenta em seu trabalho o uso de B-splines para o planejamento da trajetória de um robô Motoman HP6. O autor apresenta em seu trabalho o estudo sobre B-splines e NURBS (Non-Uniform Rational B-Splines), da dinâmica do manipulador e exemplifica com três trajetórias e mostra que com a inclusão de pontos intermediários, os erros entre a trajetória planejada e o movimento realizado pelas articulações do manipulador diminuem.

A Figura 2 demonstra uma das trajetórias planejadas por Ferreira (2011) em seu trabalho com a utilização de curvas B-Spline. A trajetória passa pelos pontos $Q_0(400, 430, 252)$, $Q_1(460, 400, 252)$, $Q_2(520, 430, 276)$, $Q_3(535, 490, 303)$, $Q_4(490, 640, 354)$, $Q_5(655, 655, 393)$, $Q_6(760, 730, 444)$ e $Q_7(850, 640, 426)$, com a inclusão dos pontos intermediários $Q(572.5, 647.5, 610)$ e $Q(707.5, 692.5, 700)$ para a redução do erro entre

as trajetórias realizadas no espaço da tarefa e no espaço das articulações.

Figura 2 – Trajetória tridimensional aplicada ao robô industrial Motoman HP6



Fonte: Ferreira (2011, p. 102).

Bey, Boudjouad e Tafat-Bouzid (2007) apresentam em seu trabalho o uso de curvas B-Spline para a geração de trajetórias de uma ferramenta de usinagem, buscando reduzir o número de pontos de controle necessários para realizar a usinagem das superfícies.

O uso de curvas cúbicas de Bézier para planejar a trajetória de um manipulador móvel é demonstrado em Jiao et al. (2013). Os autores utilizam as curvas cúbicas de Bézier para planejar em tempo real a movimentação do manipulador móvel, desviando de obstáculos, que são detectados por sensores ultrassônicos, em um percurso desconhecido.

Processos de pintura utilizando robôs manipuladores apresentam a dificuldade de manter a mesma espessura de tinta em toda a superfície. Em Chen et al. (2017), os autores buscam resolver esta situação utilizando superfícies de Bézier para descrever as superfícies a serem pintadas e propõem uma nova função base para as curvas de Bézier para o cálculo da trajetória de manipuladores. O uso de uma nova função de base para a aplicação foi feito buscando uma maior eficiência computacional.

Steuben, Steele e Turner (2011) mostram em seu trabalho a utilização de NURBS para a geração de trajetória de um manipulador com seis graus de liberdade, utilizando como exemplo de aplicações de solda e de corte em superfícies complexas.

Os autores citam como benefícios para o uso de NURBS a facilidade para representar superfícies complexas, trajetórias com velocidade e aceleração constantes e a criação de rotas mais efetivas para o ajuste de um controlador PID.

O uso de B-Splines cúbicas para a interpolação de pontos de solda para vários manipuladores colaborativos é abordado em Liu et al. (2020). Os autores utilizam as curvas B-Spline para criar a trajetória e adicionam o uso de um algoritmo de otimização por exame de partículas para otimizar o tempo e o consumo de energia dos manipuladores.

2.2 SIMULAÇÃO DE MANIPULADORES COM MOVEIT!

O MoveIt! é uma ferramenta bastante popular no planejamento de trajetórias para manipuladores. O MoveIt! é capaz de encapsular em um mesmo framework diversas bibliotecas de planejamento de trajetórias, solucionadores de cinemática inversa e direta e verificadores de colisões.

Vários motivos podem ser destacados para justificar a popularidade do MoveIt!, dentre eles, estão a sua preocupação com diminuir as barreiras de entrada para iniciantes, a integração com diversas bibliotecas e a grande variedade de manipuladores comerciais disponibilizados para uso (COLEMAN et al., 2014).

Em seu trabalho, Duarte (2020) cria um pacote para o MoveIt! com base no robô UR10, fabricado pela Universal Robots, adicionando uma tocha de solda como efetuador final do manipulador. Utilizando o modelo criado para o robô, o autor criou cinco simulações de movimentações do robô, fazendo o robô se movimentar por pontos específicos ou aleatoriamente.

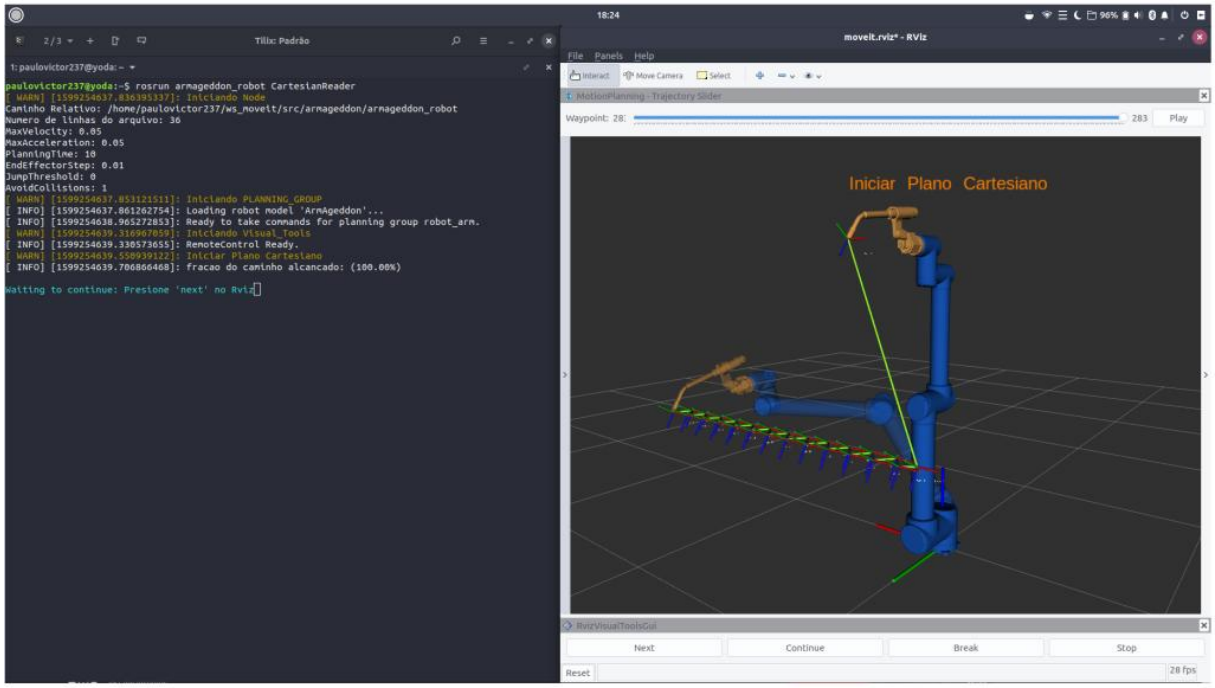
A Figura 3 mostra a aplicação Cartesian Reader, criada em Duarte (2020), que cria uma tarefa fictícia de solda e ocorre a simulação da movimentação do manipulador UR10.

O trabalho de Hernandez-Mendez et al. (2017) modela e implementa um manipulador utilizando ROS e MoveIt! com o objetivo de fazer movimentos de pick-and-place. Os autores criaram as partes do robô utilizando software CAD, criaram o modelo Unified Robot Description Format (URDF) necessário para integrar o robô ao ROS e fizeram o desenvolvimento da parte eletroeletrônica do manipulador, utilizaram o MoveIt! para o planejamento das trajetórias e utilizaram um controlador proporcional integral derivativo para fazer o controle da garra.

2.3 CONCLUSÃO

O planejamento de trajetórias é uma área bastante estudada na área de robótica, devido a sua importância para a execução da tarefa de forma eficiente e não prejudicial ao manipulador. O uso de curvas de Bézier e B-Splines para a descrição

Figura 3 – Aplicação Cartesian Reader



Fonte: Duarte (2020, p. 42).

tanto das superfícies e das trajetórias é demonstrado em trabalhos como o de Ferreira (2011) e Bey, Boudjouad e Tafat-Bouzid (2007).

Dividir o processo de planejamento de trajetórias de um robô manipulador, conforme proposto por Tonetto (2007), é uma forma de generalizar o processo de planejamento de trajetórias e tornar o processo mais sistemático.

É possível utilizar o método proposto em Tonetto (2007) de forma conjunta com softwares que auxiliam nas etapas do planejamento da trajetória. Como mostrado em Duarte (2020) e Hernandez-Mendez et al. (2017), softwares CAD podem ser utilizados para a construção de superfícies dos objetos de interesse e o MoveIt! pode calcular a trajetória do manipulador, simular a movimentação realizada pelo manipulador robótico e detectar possíveis colisões.

3 FUNDAMENTAÇÃO TEÓRICA

3.1 ROBÔS MANIPULADORES

Os robôs manipuladores são compostos, mecanicamente, por uma sequência de corpos rígidos, os elos, conectados por articulações chamadas de juntas. Um manipulador caracteriza-se por um braço que garante a mobilidade, um punho que confere destreza e um efetuador que executa a tarefa exigida do robô (SICILIANO et al., 2010).

O espaço de trabalho, comumente chamado pelo termo em inglês *workspace*, representa o espaço físico em que o atuador final do manipulador pode atingir. É o limite mecânico que o manipulador pode atingir, baseado no tamanho de seus elos e no ângulo que as suas juntas podem alcançar.

Siciliano et al. (2010) classifica os manipuladores de acordo com os tipos e a sequência dos graus de liberdade, começando da junta da base, em:

- Cartesiano;
- Cilíndrico;
- Esférico;
- SCARA;
- Antropomórfico.

Os manipuladores cartesianos são compostos por três juntas prismáticas, que são mutuamente ortogonais. Para Craig (2012), as vantagens do uso de braços cartesianos é a sua facilidade de projeto e sua rigidez. Por outro lado, a principal desvantagem é sua dificuldade de se adaptar a linhas já existentes, devido ao fato de todos os alimentadores e instrumentos precisarem estar dentro do robô.

A geometria cilíndrica é diferente da geometria cartesiana pelo fato da primeira junta ser rotacional, ao invés de prismática. Para Siciliano et al. (2010), a geometria cilíndrica oferece boa rigidez mecânica, mas a precisão da posição do punho diminui conforme o aumento do curso horizontal.

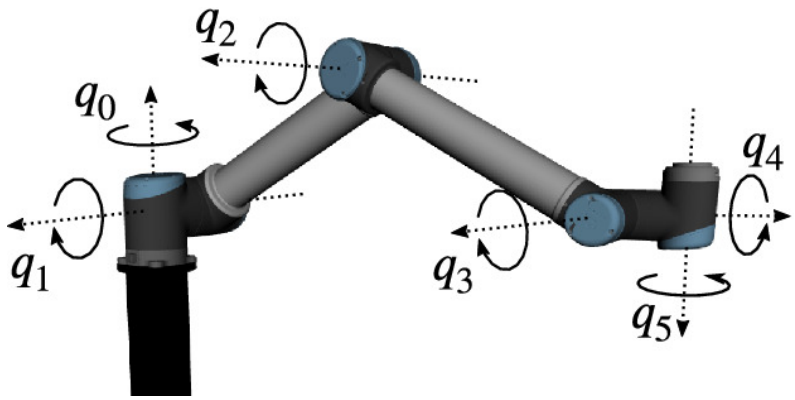
Um manipulador esférico diferencia-se de um manipulador cilíndrico pelo fato da segunda junta prismática ser substituída por uma junta rotacional. A rigidez mecânica desses manipuladores é menor que nas geometrias cartesiana e cilíndrica, além de sua construção ser mais complexa. A precisão do posicionamento do punho diminui na medida que o curso radial aumenta (SICILIANO et al., 2010).

Os manipuladores da geometria SCARA têm três juntas rotacionais, que são paralelas, e mais uma junta prismática, que permite a movimentação do efetuador na direção perpendicular. Sua principal vantagem é o fato das juntas rotacionais não precisarem suportar peso, nem do manipulador e nem da carga (CRAIG, 2012).

O manipulador antropomórfico é composto por três juntas rotacionais. O eixo de revolução da primeira junta é ortogonal aos eixos das outras duas juntas, que são paralelas. A maior parte dos robôs instalados em indústrias é do tipo antropomórfico, devido ao fato de ser o mais hábil de todos os manipuladores de cadeia cinemática aberta (SICILIANO et al., 2010).

Modelos populares de manipuladores antropomórficos são o Panda, fabricado pela empresa alemã Franka Emika, e o UR10, fabricado pela Universal Robots. O Panda é bastante utilizado para os estudos iniciais de planejamento de movimentação com o MoveIt, já que os tutoriais iniciais do plugin são feitos utilizando este robô. A Figura 4 mostra o robô UR10 e suas juntas, onde cada q_i representa um dos graus de liberdade do manipulador.

Figura 4 – UR10 e suas juntas



Fonte: Krämer et al. (2020, p. 1222).

3.2 ROS

O ROS é um meta sistema operacional, que provê serviços como abstração de hardware, controle de dispositivos em baixo nível, implementação de funcionalidades comumente utilizadas, troca de mensagens entre processos e gerenciamento de pacotes. O ROS também fornece ferramentas e bibliotecas que permitem suporte multi-plataforma (OPEN ROBOTICS, 2018).

O principal objetivo do ROS é suportar o reuso de código na pesquisa e no desenvolvimento da robótica. O ROS é um framework distribuído de processos, também chamados de nós, que permite que os executáveis sejam desenvolvidos individualmente. Esses processos podem ser agrupados em pacotes e pilhas, que podem ser facilmente compartilhadas e distribuídas. O ROS também suporta um sistema de repositórios que facilitam a colaboração entre os diversos desenvolvedores e a distribuição dos códigos desenvolvidos (OPEN ROBOTICS, 2014).

De acordo com OPEN ROBOTICS (2014), o ROS é dividido em três níveis de conceitos: O nível de arquivo (filesystem level), o de grafo computacional (computation

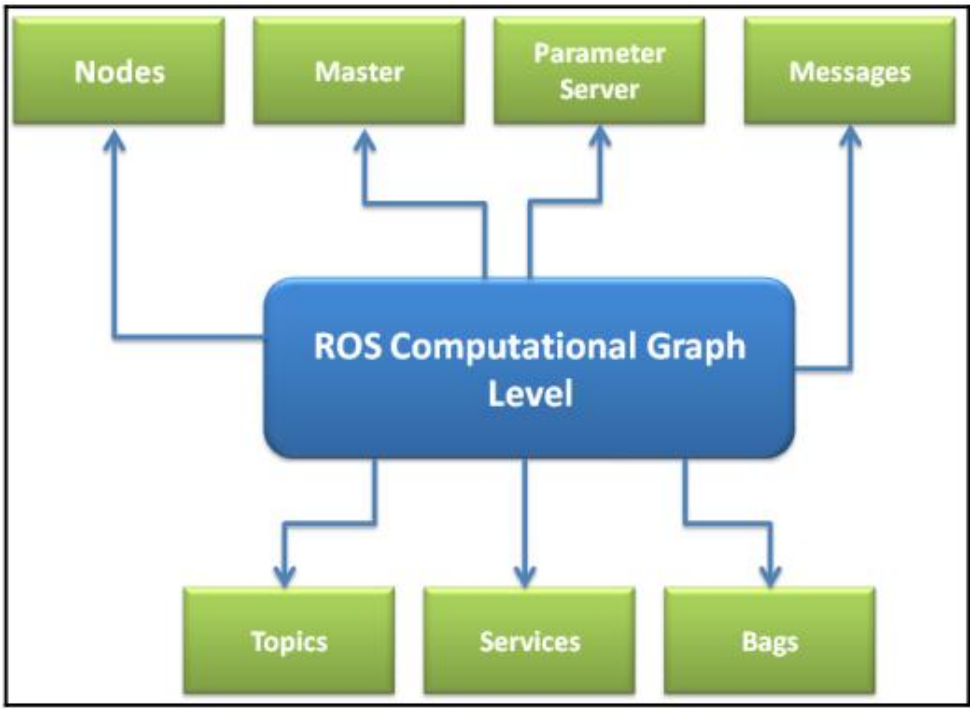
graph level) e o de comunidade (community level).

No filesystem level encontram-se recursos e arquivos utilizados pelo ROS e que estão no disco rígido onde o ROS encontra-se instalado. Nesse nível, estão os pacotes, metapacotes e os manifestos dos pacotes, os repositórios, além de definições de tipos de mensagens e serviços.

Segundo Joseph (2018, p.19) "A computação no ROS é feita é feita utilizando uma rede de um processo chamado de nodes ROS. Essa rede de computação é chamada de grafo computacional". No computation graph level estão os nodes, o master, o parameter server, as mensagens, tópicos, serviços e bags.

A Figura 5 mostra a estrutura do ROS computation graph level.

Figura 5 – ROS Computation Graph Level



Fonte: Joseph (2018, p.19).

Um breve resumo dos conceitos que estão na Figura 5 pode ser visto abaixo:

- Nodes, processos que realizam a computação. Um robô é composto por diversos nodes;
- Master, provê registro de nome e pesquisa para os diversos nodes, o que permite a troca de mensagem entre os nodes;
- Parameter server, permite que as informações sejam registradas em uma localização central e que os nodes acessem essas informações;
- Mensagens, estruturas de dados que permitem que ocorra a comunicação entre os nodes;
- Tópicos, forma como as mensagens são transportadas entre os nodes;
- Bags, um formato para salvar e reproduzir mensagens de dados.

O nível de comunidade permite que diferentes grupos de pesquisa possam compartilhar software e conhecimentos. Nesse nível estão recursos como distribuições, repositórios, a Wiki do projeto, listas de email e o sistema de tickets, utilizado para reportar falhas ou solicitar melhorias e novos recursos (OPEN ROBOTICS, 2014).

O ROS pode ser utilizado em projetos com diversos tipos de dispositivos robóticos, sejam Autonomous Underwater Vehicles (AUVs), Unmanned Ground Vehicles (UGVs), drones, carros autônomos ou manipuladores. O ROS, ao gerenciar atuadores e sensores, pode trabalhar no planejamento de trajetória, de movimentação, evitando colisões e no planejamento das tarefas desenvolvidas pelos robôs.

3.2.1 MoveIt!

O MoveIt! é uma plataforma de manipulação robótica para o ROS. Ele incorpora os avanços mais recentes no planejamento de movimentação, manipulação, percepção 3D, cinemática, controle e navegação (PICKNIK ROBOTICS, 2022b).

Como funciona através de uma interface de plugin, o MoveIt pode utilizar várias bibliotecas planejadoras de movimentação utilizando o nó move_group do ROS. Alguns dos planejadores que podem ser utilizados com o MoveIt são o Open Motion Planning Library (OMPL), o Covariant Hamiltonian Optimization for Motion Planning (CHOMP) e o planejador da Pilz (PICKNIK ROBOTICS, 2022a).

O usuário deve especificar a localização inicial e final do braço, assim como do efetuador final. O planejador de trajetória verifica se existem colisões. Então, o nó move_group, do ROS, gera a trajetória com base na requisição do usuário (PICKNIK ROBOTICS, 2022a).

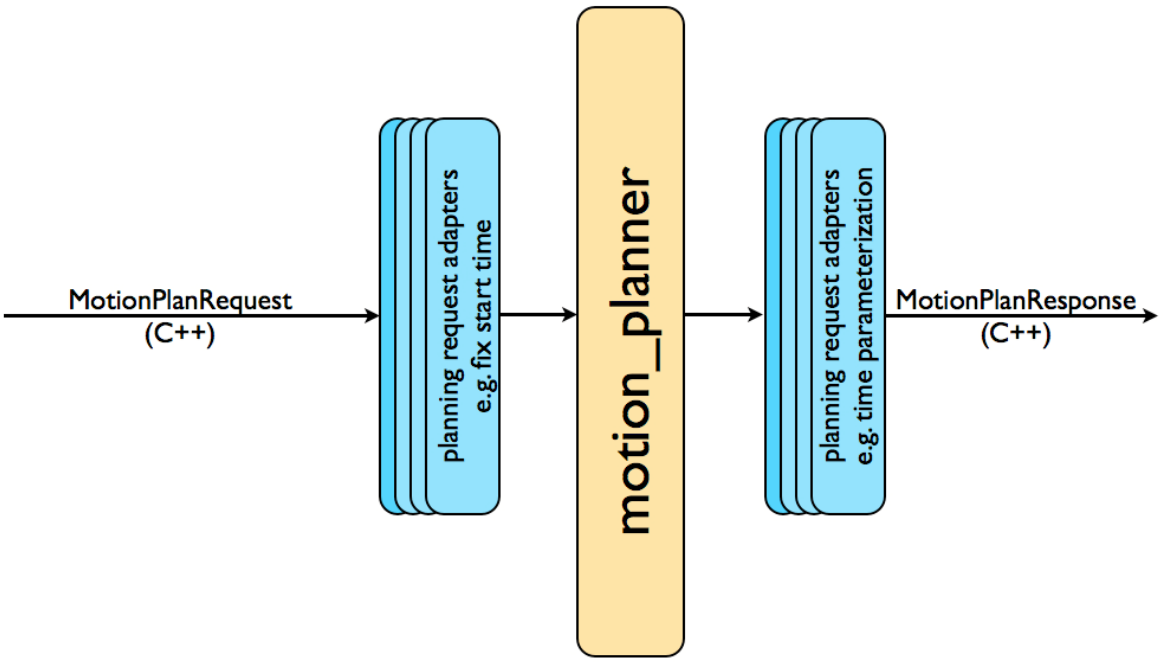
A Figura 6 mostra o fluxo do planejador da movimentação. Antes e após a atuação do planejador, ocorrem processamentos dos planos de movimentação, onde ocorrem operações como a verificação se o robô está em seu espaço de trabalho e a trajetória é parametrizada em relação ao tempo (PICKNIK ROBOTICS, 2022a).

3.2.2 ArmAgeddon

O *ArmAgeddon* é um pacote para ROS desenvolvido em Duarte (2020). Nesse trabalho, o autor utilizou como base o robô UR10, robô fabricado pela empresa Universal Robots. De acordo com a fabricante o robô conta com seis juntas rotativas e tem capacidade de carga útil de até 10kg (UNIVERSAL ROBOTS, 2014). Como manipulador final do robô, o autor adicionou uma tocha de solda.

Em Duarte (2020) também ocorreu o desenvolvimento do arquivo URDF e o modelo de colisão, além da validação dos parâmetros de Denavit-Hartenberg do robô.

Figura 6 – Fluxo do planejador de movimentação



Fonte: PICKNIK ROBOTICS (2022a).

O pacote ArmAgeddon também conta com as configurações iniciais do MoveIt! e algumas aplicações simuladas pelo desenvolvedor (DUARTE, 2020).

As aplicações simuladas em Duarte (2020) são:

a. GoRandom, que executa um movimento randômico;
b. GoHome, que retorna o robô para a posição inicial;
c. TypeMoves, que executa movimento através de valores das juntas e movimentos lineares através de valores de posição e orientação;
d. CartesianReader, que planeja uma trajetória através de um vetor de pontos;
e. StopMove, que desenvolve a interface de software para o acionamento de uma chave de fim de curso.

Na aplicação *CartesianReader*, o programa lê um vetor de pontos, que estão declarados em um arquivo do tipo Markdown, e planeja a trajetória utilizando movimentos lineares. Na aplicação, são utilizados pontos fictícios, que simulam uma trajetória de pontos utilizado em um processo de soldagem (DUARTE, 2020).

3.3 PLANEJAMENTO DE TRAJETÓRIAS

Para Craig (2012), o planejamento de trajetórias se preocupa em descrever a movimentação de um manipulador no espaço multidimensional. O planejamento da trajetória se preocupa em tornar fácil para o usuário criar a trajetória de um robô, onde ele apenas insere seus pontos de interesse, como as posições inicial e final, além de

pontos de passagem, e o sistema calcula a trajetória mais adequada para os objetivos pretendidos.

De acordo com Siciliano et al. (2010), o planejamento da trajetória consiste em gerar uma função de interpolação, tipicamente polinomial, que passe pelas posições desejadas pelo usuário. Craig (2012) também ressalta que é importante que a trajetória seja calculada utilizando funções suaves, sendo ela contínua e com a primeira derivada contínua, evitando que os movimentos bruscos causem desgastes no manipulador.

Craig (2012) afirma que, em geral, se considera os movimentos de um manipulador como os movimentos do sistema de referência da ferramenta em relação ao sistema de referência da estação. Essa consideração resulta em uma modularidade e permite que a mesma trajetória possa ser utilizada em outro manipulador.

3.3.1 Interpolação

Em diversas áreas do conhecimento, como a engenharia e a estatística, é comum haver um conjunto de dados discretos conhecidos previamente e a interpolação é um método matemático para construir um conjunto de dados contínuo, partindo do conjunto discreto, utilizando uma função matemática.

No caso de trajetórias de manipuladores, os dados discretos são os pontos de início, final e os pontos de interesse que o manipulador necessita passar para cumprir a tarefa designada para ele. Utilizando desses dados, é calculada uma função de interpolação que passe por esses pontos.

A forma mais simples de interpolação é a interpolação linear, que utiliza equações da reta, no formato da equação 1, para interpolar pontos já conhecidos. Para interpolar os pontos (x_1, y_1) e (x_2, y_2), a função utilizada é a função demonstrada na equação 2.

$$y = a + bx \tag{1}$$

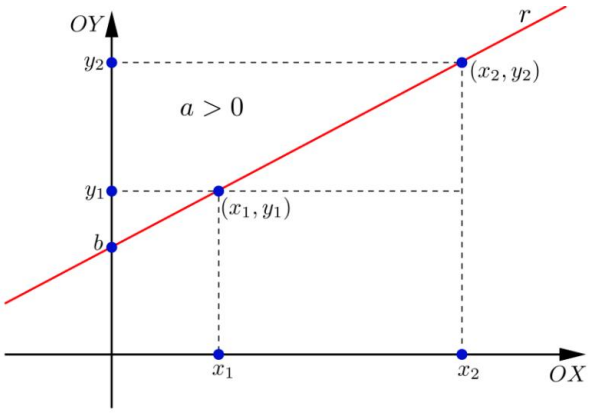
$$y = y_1 + (y_2 - y_1)\frac{x - x_1}{x_2 - x_1} \tag{2}$$

A representação gráfica da equação da reta do exemplo está demonstrada na Figura 7.

A interpolação linear tem a vantagem de ser simples computacionalmente, porém não consegue atender aos requisitos de ter velocidade e aceleração contínuas, não gerando uma trajetória suave, o que acaba prejudicando o manipulador.

Para conseguir atingir os requisitos de velocidade e aceleração, podem ser utilizadas interpolações polinomiais, como o método de coeficientes indeterminados e os métodos de Lagrange, de Newton e de Hermite.

Figura 7 – Interpolação linear



Fonte: Frensel e Delgado (2010, p.21).

O resultado da interpolação polinomial é uma função definida pelo polinômio $C(x)$ que passa por $n + 1$ pontos conhecidos. Tendo os pontos distintos conhecidos é possível obter um único polinômio $C_n(x)$ que passa pelos pontos, no formato da Equação 3 (FERREIRA, 2011).

$$C_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x^1 + a_0 \tag{3}$$

As interpolações polinomiais podem apresentar dois problemas: instabilidade e erros de truncamento. A instabilidade são oscilações de amplitude próximas aos pontos de interesse e os erros de truncamento são causados pelas limitações computacionais, quanto maior o grau do polinômio, menor são os valores dos coeficientes, então as limitações nos números de casas decimais podem causar erros e o resultado divergir do esperado (FERREIRA, 2011).

Ferreira (2011) apresenta como alternativa para contornar os problemas existentes em interpolações polinomiais com o uso de interpolações polinomiais por partes, que utiliza interpolações diferentes para um subconjunto de pontos, respeitando um grau de continuidade da curva entre os subconjuntos. As interpolações utilizadas para os subconjuntos podem ser as citadas anteriormente, como a linear, quadrática ou de Hermite.

3.3.2 Splines

De acordo com Ferreira (2011), o princípio básico de uma spline é a interpolação de Hermite por partes. Na interpolação de Hermite por partes é imposta a derivada em cada ponto, entretanto, em uma spline, é imposta uma condição de continuidade, igualando a derivada do final de um segmento com a derivada de início do segmento seguinte.

Para um conjunto de $n + 1$ pontos, a spline é definida por n polinomiais do p-ésimo grau, no formato da Equação 4. Tendo um polinômio do p-ésimo grau $(p + 1)$

coeficientes, são necessários $n(p+1)$ coeficientes para definir a spline, chamados, genericamente, de a (FERREIRA, 2011).

$$S_i(x) = a_{p,i}x^p + a_{p-1,i}x^{p-1} + \cdots + a_{1,i}x + a_{0,i} \tag{4}$$

São necessárias $n(p+1)$ equações para encontrar os $n(p+1)$ coeficientes da spline. As equações são obtidas a partir das condições de contorno dadas a partir do grau de continuidade desejado para a spline.

3.3.3 B-Spline

A B-Spline é uma pequena modificação das splines, que permitem o controle local das curvas. O controle local da curva permite que ao alterar um ponto de controle, apenas a curva próxima aos pontos vizinhos seja alterada, enquanto na spline clássica a alteração de um ponto pode alterar toda a curva.

As curvas B-spline são definidas com a utilização de funções de base. Piegl e Tiller (1997) utilizam o método de recorrência de deBoor para definir as funções de base de uma B-spline, devido a sua maior facilidade de implementação computacional.

Piegl e Tiller (1997) mostram que sendo $U = \{u_0, ..., u_m\}$ uma sequência de números reais não decrescente, ou seja, $u_i \leq u_{i+1}$, para $i = 0, ..., m-1$. Os u_i são chamados de nós e U o vetor de nós. A i-ésima função de base da B-spline de grau p, denotada por $N_{i,p}(u)$, é definida pela Equação 5.

$$N_{i,0}(u) = \begin{cases} 1, & \text{se } u_i \leq u < u_{i+1} \\ 0, & \text{em outros casos} \end{cases} \tag{5a}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \tag{5b}$$

Ferreira (2011) ressalta que, ao contrário das splines, as funções B-spline não são funções interpoladoras, já que a curva gerada não passa necessariamente pelos pontos de controle. Em razão disso, as funções B-spline são consideradas ajustadores de aproximação.

De acordo com Piegl e Tiller (1997), uma curva B-spline do p-ésimo grau é definida pela Equação 6.

$$\boldsymbol{C}(u) = \sum_{i=0}^{n} N_{i,p}(u)\boldsymbol{P_i}, \text{com } a \leq u \leq b \tag{6}$$

Onde $\boldsymbol{P_i}$ são os pontos de controle e $N_{i,p}(u)$ são as funções B-spline de base de grau p.

Para traçar uma curva B-Spline interpoladora partindo de um determinado número de pontos pré-determinados é necessário calcular um conjunto de pontos de controle e um novo vetor nó, partindo dos pontos dados (FERREIRA, 2011).

Considerando o conjunto de pontos $\boldsymbol{Q_k}$ e que para cada $\boldsymbol{Q_k}$ existe um nó $\overline{u}_k$, o vetor nó U pode ser escolhido para o cálculo das funções de base. Então, um sistema de $(n+1)$ equações pode ser resolvido pela Equação 7 (FERREIRA, 2011).

$$\boldsymbol{Q_k} = \boldsymbol{C}(\overline{u}_k) = \sum_{i=1}^{n} N_{i,p}(\overline{u}_k)\boldsymbol{P_i} \tag{7}$$

Tendo como incógnitas, os $(n+1)$ pontos $\boldsymbol{P_i}$. Após obter os pontos de controle, calcula-se a B-spline utilizando a Equação 6

A escolha de u_k e U afeta o formato da curva e pode ser feita de diversas maneiras, como os deixando os nós igualmente espaçados, utilizando um comprimento de corda ou o método centrípeto. Para Ferreira (2011), o método do comprimento de corda, definido pela Equação 8, é o método mais utilizado, pois resulta em uma curva mais comportada.

$$d = \sum_{k=1}^{n} |\boldsymbol{Q_k} - \boldsymbol{Q_{k-1}}| \tag{8a}$$

$$u_k = \begin{cases} \overline{u}_0 = 0 \\ \overline{u}_n = 1 \\ \overline{u}_k = \overline{u}_{k-1} + \frac{|\boldsymbol{Q_k} - \boldsymbol{Q_{k-1}}|}{d}, \quad k = 1, ..., n-1 \end{cases} \tag{8b}$$

O vetor U pode ser calculado utilizado uma distribuição uniforme ou o método de averaging, obtido a partir do parâmetro de interpolação $\overline{u}_k$.

O uso de funções chamadas de NURBS, variação das B-Splines que adicionam pesos aos pontos de controle, também podem ser usadas no planejamento de trajetória robóticas. Porém, Ferreira (2011) aponta que não é justificável a utilização de NURBS em trajetórias de manipuladores robóticos, visto que ao alterar o peso relacionado a um ponto, a função deixa de ser interpoladora, portanto a diferença das NURBS em relação as B-Splines torna-se uma desvantagem. Uma NURBS é melhor empregada em aplicações que exigem o desvio de obstáculos móveis, devido sua flexibilidade, e onde a passagem por pontos pré-determinados não seja obrigatória.

3.3.4 Curvas de Bézier

As curvas de Bézier foram desenvolvidas nos anos 60 por Piere Bézier e Paul de Faget de Casteljau, matemáticos que trabalhavam na indústria automotiva. A intenção dos autores era de desenvolver curvas que pudessem ser utilizadas para a

modelagem de formas aerodinâmicas complexas em programas CAD (Computer-Aided Design) para os carros.

Atualmente, as Curvas de Bézier, devido as suas características e simplicidade computacional, são utilizadas em diversas aplicações, em especial na computação gráfica e animação, usinagem e geração de caminhos (SILVA, 2015).

De acordo com Simoni (2005), as curvas de Bézier são curvas paramétricas, geralmente estando definidas no intervalo paramétrico [0,1], definidas através de pontos encontrados com o algoritmo de Casteljau ou utilizando os polinômios de Bernstein.

Uma curva de Bézier com n pontos de controle, tem um grau de $n+1$. Para Simoni (2005), uma curva de Bézier precisa de ao menos três pontos de controle para ser definida, mas a forma mais comum é a curva de Bézier cúbica, definida por quatro pontos de controle, sendo dois pontos iniciais e dois pontos de controle, sendo que a curva passa necessariamente pelos pontos iniciais e não passa pelos pontos de controle.

Piegl e Tiller (1997) definem uma curva de Bézier pela Equação 9.

$$\boldsymbol{C}(u) = \sum_{i=0}^{n} B_{i,n}\boldsymbol{P_i}, \text{com } 0 \leq u \leq 1 \tag{9}$$

As funções de base $B_{i,n}$ são os polinômios de grau n de Bernstein, definidos pela Equação 10 e $\boldsymbol{P_i}$ são os pontos de controle.

$$B_{i,n}(u) = \frac{n!}{i!\,(n-i)!}u^i(1-u)^{n-1} \tag{10}$$

Portanto, utilizando a definição dos polinômios de Bernstein, podemos representar uma Curva de Bézier cúbica conforme a Equação 11.

$$\boldsymbol{C^3}(u) = (1-u)^3\boldsymbol{P_0} + 3u(1-u)^2\boldsymbol{P_1} + 3u^2(1-u)\boldsymbol{P_2} + u^3\boldsymbol{P_3} \tag{11}$$

Através das equações 9, 10 e 11 é possível verificar que quanto maior o grau n, mais os cálculos dos polinômios se tornam complexos, exigindo um custo computacional mais elevado. Simoni (2005) observa que, por essa razão, as Curvas de Bézier cúbicas são as mais utilizadas.

Simoni (2005) aponta que dentre as características das curvas de Bézier estão a interpolação endpoint, que garante que a curva passará pelos pontos de controle inicial e final, e que as curvas de Bézier são da classe C^∞, ou seja, possuem derivadas contínuas de todas as ordens. Essas características são bastante interessantes do ponto de vista do planejamento de trajetórias para manipuladores, visto que garantem que o manipulador passará pelos pontos desejados, mas mantendo velocidades e aceleração contínuas, evitando problemas de desgaste para o manipulador robótico.

As Curvas de Bézier, por estarem restritas apenas ao primeiro e ao último ponto, também são comumente utilizadas no desvio de obstáculos. Ao detectar um

obstáculo, apenas a mudança de poucos pontos de controle permite que a colisão seja evitada (SILVA, 2015).

4 DESENVOLVIMENTO

4.1 B-SPLINES

O algoritmo para a criação das curvas B-Splines foi implementado utilizando o pacote SciPy e o seu sub-pacote interpolate. Após o usuário informar os pontos necessários para a criação das curvas, o programa utiliza as funções das bibliotecas para representar a curva B-Spline, gerar um gráfico da função e o arquivo com pontos que pertencem a curva para serem utilizados pelo programa CartesianReader para gerar as trajetórias no MoveIt!.

A função **splprep** recebe como parâmetro os pontos que serão interpolados, o grau da curva e um parâmetro que determina a suavização da curva, no caso, 0. Os retornos dessa função são uma tupla com o novo vetor de nós, os coeficientes da B-Spline e o grau da spline, além do vetor u onde os pontos da B-Spline são plotados.

Após a função splprep, é necessário utilizar a função splev para estimar a curva b-spline, que recebe o vetor u e a tupla calculados com a função splprep e retorna um vetor com os pontos que representam a b-spline.

Tendo as curvas calculadas, é possível passar os pontos delas para a aplicação "CartesianReader", do pacote "ArmAgeddon", e simular as trajetórias utilizando o MoveIt!.

A primeira aplicação de testes foi criada de forma que a curva se aproximasse de uma circunferência, com a intenção de testar a movimentação bidimensional do manipulador. Foram utilizados quatro pontos equidistantes de uma circunferência com raio de 0.1m e unindo o último ponto ao primeiro, de forma que a curva se tornasse fechada. Os pontos de controle utilizados nesta aplicação estão na Tabela 1.

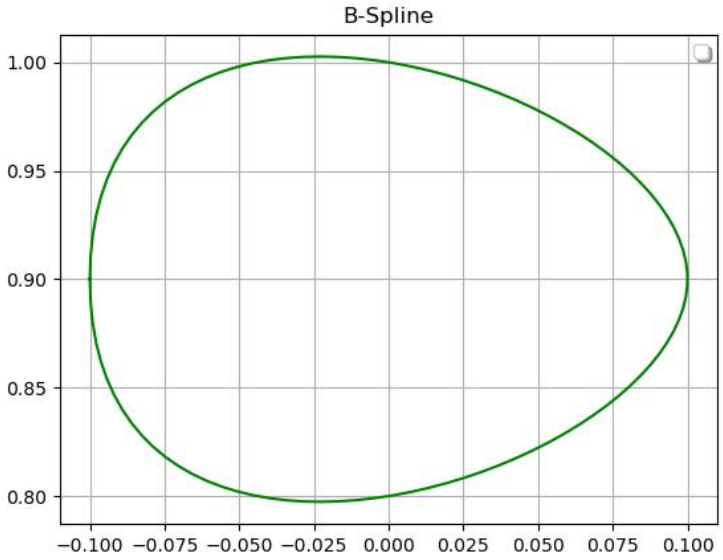
Tabela 1 – Pontos de controle utilizados na primeira aplicação com B-Splines

x	**y**	**z**
-0.1	0.9	1.15
0	1	1.15
0.1	0.9	1.15
0	0.8	1.15
-0.1	0.9	1.15

Fonte: Autor

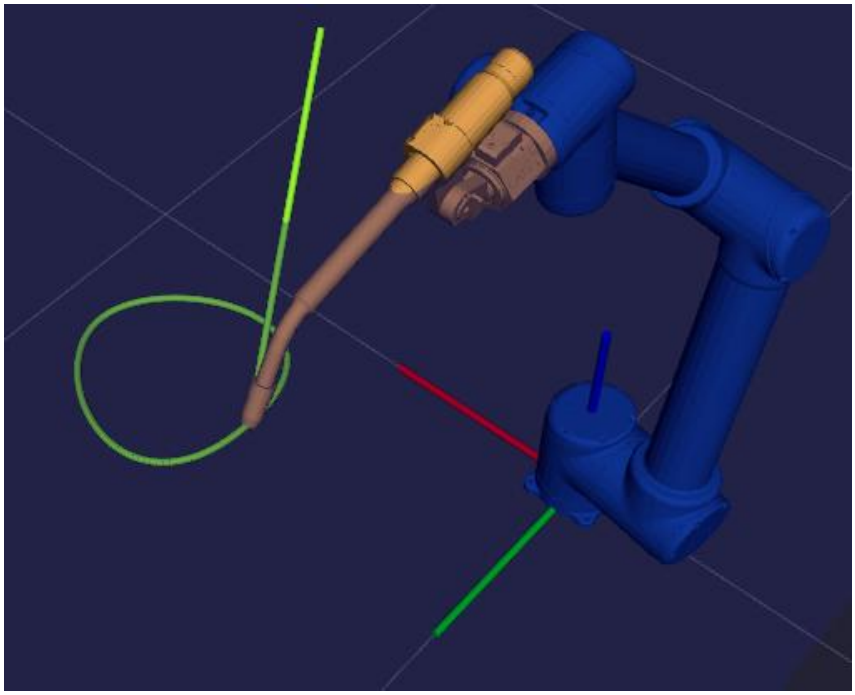
As Figuras 8 e 9 mostram a curva gerada pelo script e a trajetória simulada no MoveIt! com a curva calculada.

Figura 8 – B-Spline gerada na primeira aplicação



Fonte: Autor.

Figura 9 – Trajetória gerada na primeira aplicação com B-Splines



Fonte: Autor.

Para fins de teste do script e entendimento das funções B-Spline, foi criada uma aplicação simulando uma trajetória que passa por pontos aleatórios dentro da workspace do robô. Na Tabela 2 estão os pontos de controle, que foram escolhidos aleatoriamente, utilizados para criar a trajetória na segunda aplicação de testes.

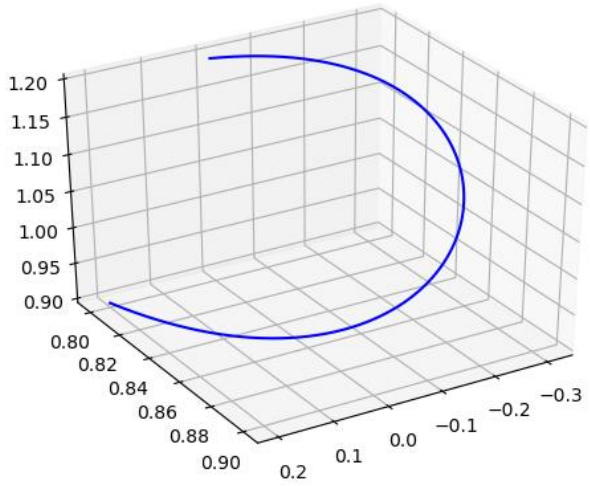
Tabela 2 – Pontos de controle utilizados na segunda aplicação com B-Splines

x	y	z
0.2	0.8	0.9
0	0.9	1
-0.3	0.85	1.1
0	0.8	1.2

Fonte: Autor

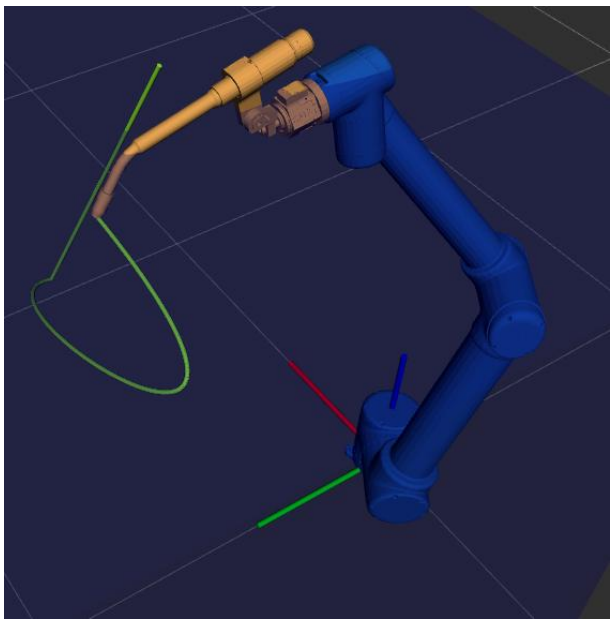
Nas figuras 10 e 11 é possível verificar a curva gerada a partir dos pontos criados na Tabela 2.

Figura 10 – B-Spline gerada pelo script em Python para a segunda aplicação



Fonte: Autor.

Figura 11 – Trajetória simulada no MoveIt! para a segunda aplicação com B-Splines



Fonte: Autor.

O script criado para o cálculo das curvas B-Spline está inserido no Apêndice Apêndice A.

4.2 CURVAS DE BÉZIER

Foi desenvolvido um script em Python para gerar as curvas de Bézier com base nos pontos de controle passados pelo usuário. O script recebe os quatro pontos necessários para o cálculo da curva de Bézier, calcula a curva, plota o gráfico dela e salva 100 pontos que passam sobre a curva em um arquivo de texto, além de salvar no arquivo de texto os parâmetros necessários para o cálculo da trajetória no MoveIt!, com a aplicação "CartesianReader", do pacote "ArmAgeddon".

Inicialmente, foi criada uma aplicação com quatro curvas de Bézier, de uma forma que o manipulador faça um movimento similar ao de uma circunferência. Esta aplicação teve como objetivo criar uma trajetória bidimensional, similar a criada na primeira aplicação utilizando as curvas B-Splines.

Os pontos de controle utilizados para a criação dessa aplicação de testes estão na Tabela 3:

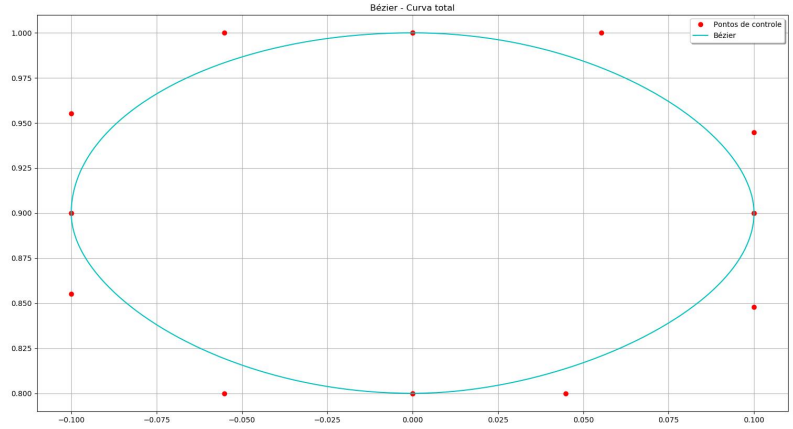
Tabela 3 – Pontos de controle utilizados na primeira aplicação com Curvas de Bézier

	x	**y**	**z**
Curva 1	-0.1	0.9	1.15
	-0.1	0.955	1.15
	-0.055	1	1.15
	0	1	1.15
Curva 2	0	1	1.15
	0.055	1	1.15
	0.1	0.944	1.15
	0.1	0.9	1.15
Curva 3	0.1	0.9	1.15
	0.1	0.847	1.15
	0.044	0.8	1.15
	0	0.8	1.15
Curva 4	0	0.8	1.15
	-0.055	0.8	1.15
	-0.1	0.855	1.15
	-0.1	0.9	1.15

Fonte: Autor

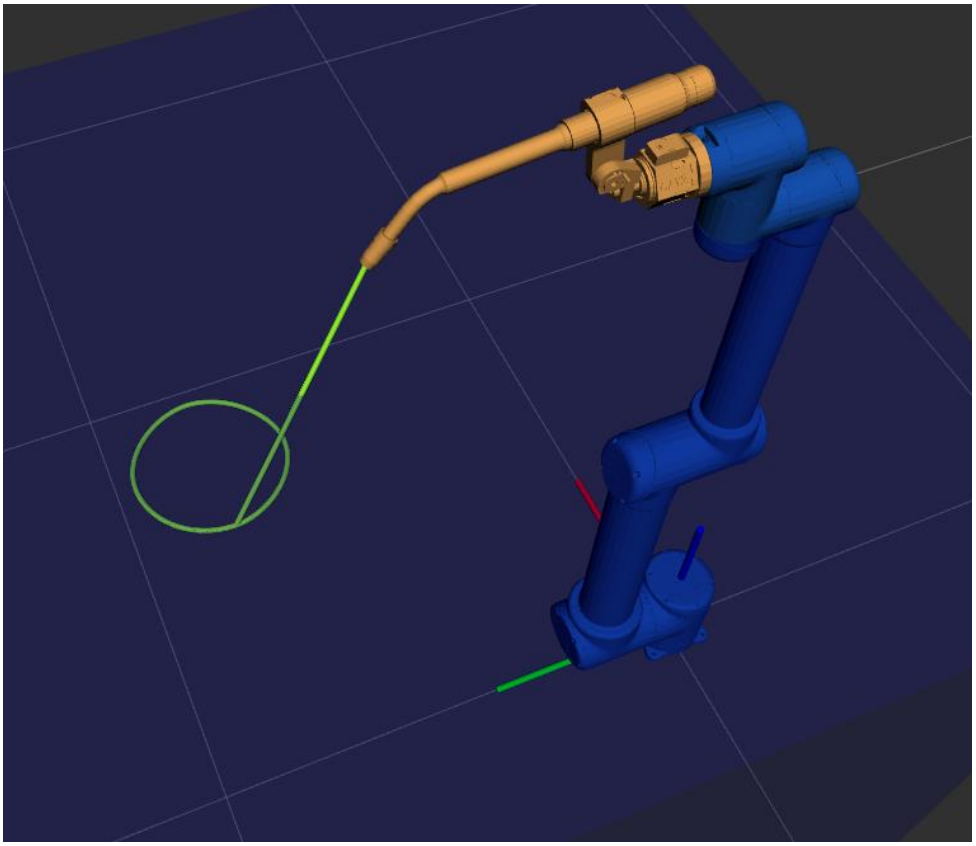
Nas Figuras 12 e 13 estão demonstrados o gráfico que resultou do uso das curvas geradas utilizando os pontos da Tabela 3 e a aplicação em funcionamento.

Figura 12 – Curva de Bézier gerada pelo script em Python



Fonte: Autor.

Figura 13 – Trajetória circular utilizando Curvas de Bézier



Fonte: Autor.

Utilizando o script desenvolvido, foi criada uma curva de Bézier tridimensional de testes, mostrando a capacidade do algoritmo de gerar uma curva de Bézier e de simular a trajetória com ela no MoveIt!. A trajetória foi criada utilizando pontos aleatórios dentro da workspace do manipulador e utilizando um ângulo fixo para a tocha.

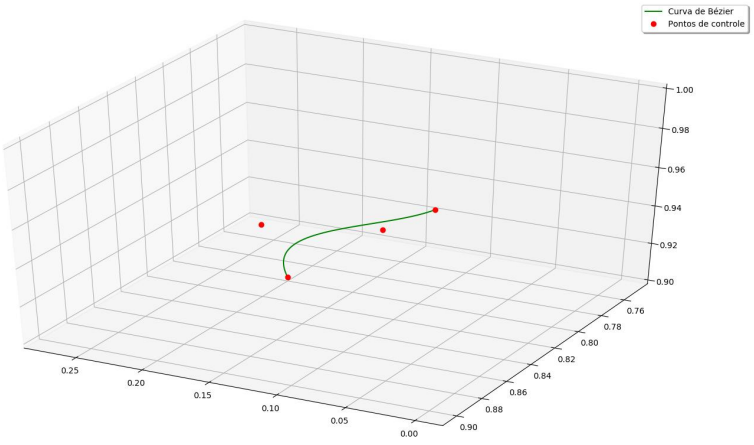
O objetivo, assim como na segunda aplicação de B-Splines, foi testar a movimentação tridimensional do manipulador. Na Tabela 4 estão listados os pontos de controle utilizados para criar a curva. As Figuras 14 e 15 mostram, respectivamente, a curva gerada em Python e a trajetória simulada no MoveIt!.

Tabela 4 – Pontos de controle utilizados na segunda aplicação com Curvas de Bézier

x	y	z
0.2	0.8	0.9
0.27	0.75	0.9
0.1	0.83	0.95
0	0.9	1

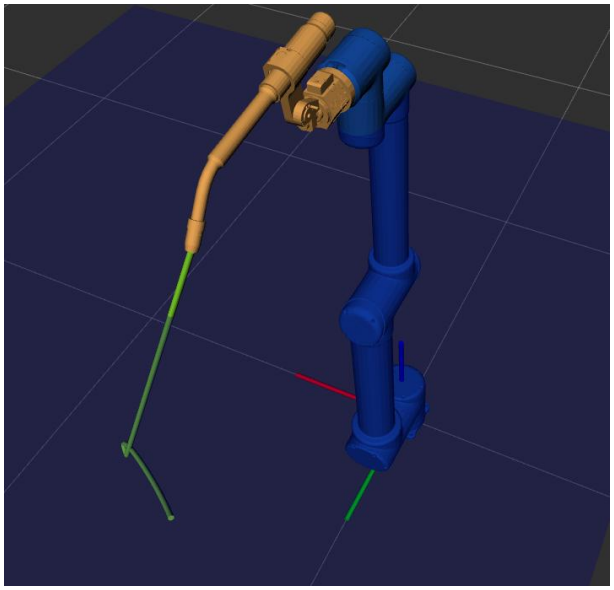
Fonte: Autor

Figura 14 – Curva de Bézier gerada pelo script em Python para a segunda aplicação



Fonte: Autor.

Figura 15 – Trajetória simulada no MoveIt! para a segunda aplicação com Curvas de Bézier



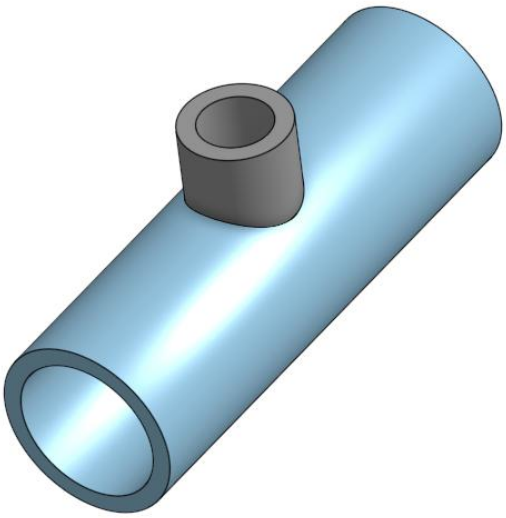
Fonte: Autor.

O script criado para o cálculo das curvas de Bézier está inserido no Apêndice B.

4.3 SIMULAÇÃO DE APLICAÇÕES

As trajetórias iniciais foram criadas em ambientes sem a presença de obstáculos, portanto o passo seguinte foi a criação de uma aplicação próxima ao uso real de um manipulador com uma tocha de solda acoplada. O objeto utilizado foi modelado utilizado o software CAD OnShape devido à sua portabilidade, podendo ser utilizado em um navegador web. Na figura 16 está representado o objeto modelado para essa simulação.

Figura 16 – Objeto utilizado para simulação da trajetória



Fonte: Autor.

O objeto foi posicionado com centro no ponto *(0, 0.9, 1)*. A aplicação consiste na criação das trajetórias que possam passar pela intersecção dos dois cilindros que compõem o objeto.

Para a criação das trajetórias foi necessário considerar que o ROS utiliza o ponto central da tocha para o cálculo das trajetórias, portanto, para evitar colisões é necessário considerar a distância entre o centro da ponta da tocha até o final do objeto de colisão. No modelo de colisão utilizado para as simulações, essa distância era de 9 milímetros.

Em ambos os casos o controle do ângulo da tocha de solda foi feito de forma experimental.

4.3.1 Alteração no modelo de colisão

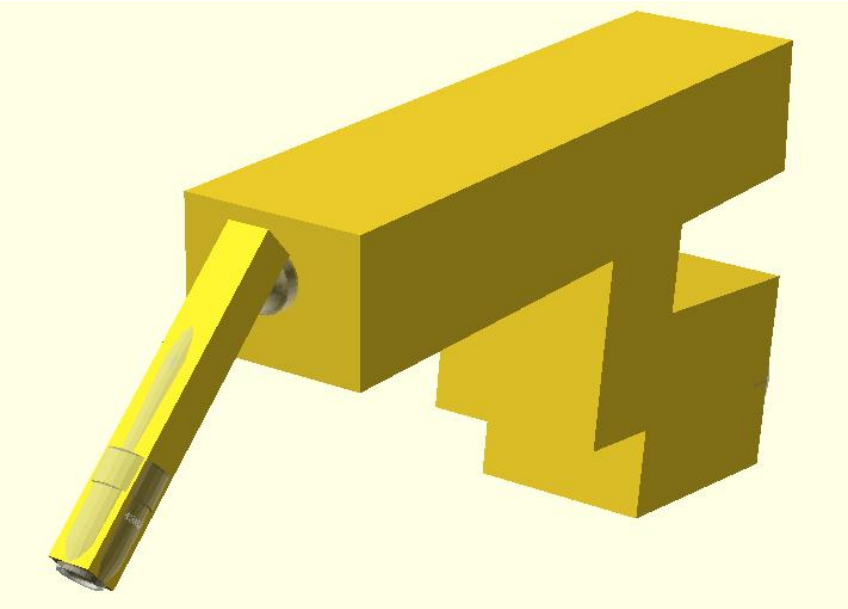
Ao iniciar as simulações para as trajetórias percebeu-se que o objeto de colisão do manipulador criado por Duarte (2020) não tinha um tamanho ideal para a realização

das simulações com objetos, pois o tamanho do objeto criado para simular a parte frontal da tocha era muito maior do que o real, causando colisões indevidas ao simular as trajetórias criadas.

Para corrigir este problema, foi necessário alterar o objeto utilizando o software OpenSCAD, que gera um modelo 3D com base em uma linguagem de descrição própria. A alteração foi apenas da parte frontal da tocha, representada por um retângulo, que originalmente tinha largura e altura de 32cm e comprimento de 143cm, que foi alterado para largura e altura de 18cm e comprimento de 137cm, de forma a se aproximar mais da tocha utilizada no manipulador.

A partir do novo modelo gerado no OpenSCAD foi possível gerar um novo arquivo STL e adicioná-lo nas simulações do MoveIt, através do arquivo URDF. O novo modelo de colisão, gerado no OpenSCAD, está presente na Figura 17.

Figura 17 – Modelo de colisão da tocha de solda alterado

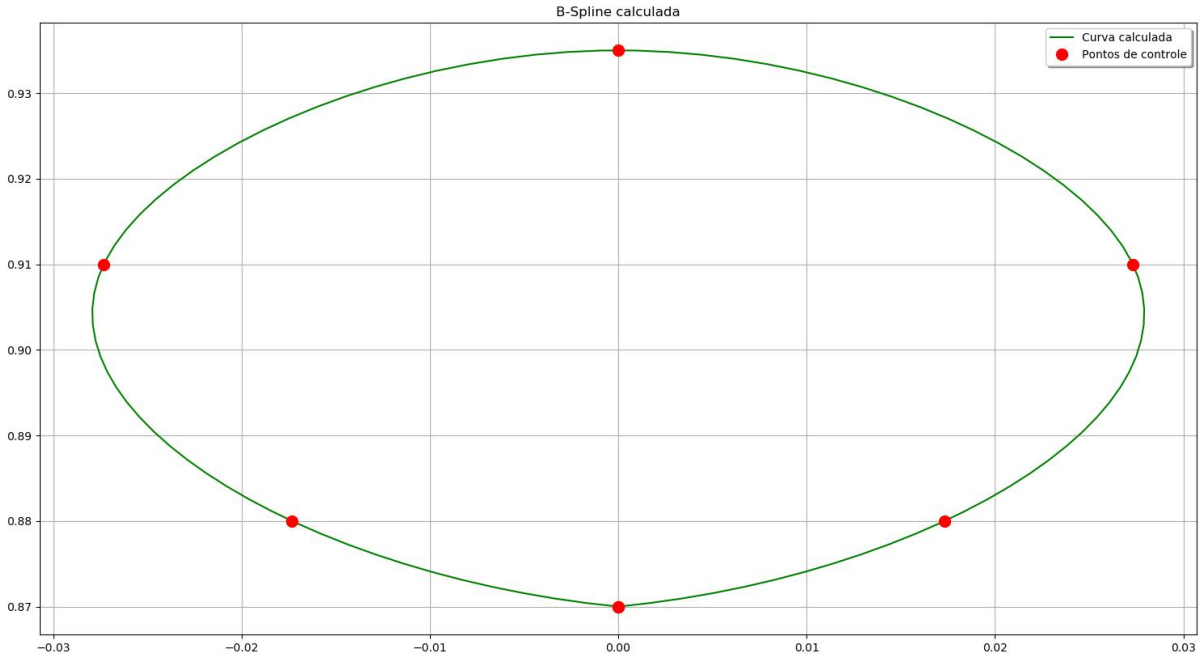


Fonte: Autor.

Após a alteração no modelo de colisão, tornando o modelo de colisão mais próximo ao manipulador real, as simulações com objetos ocorreram sem problemas, podendo ser realizados os testes com as Curvas de Bézier e as B-Splines.

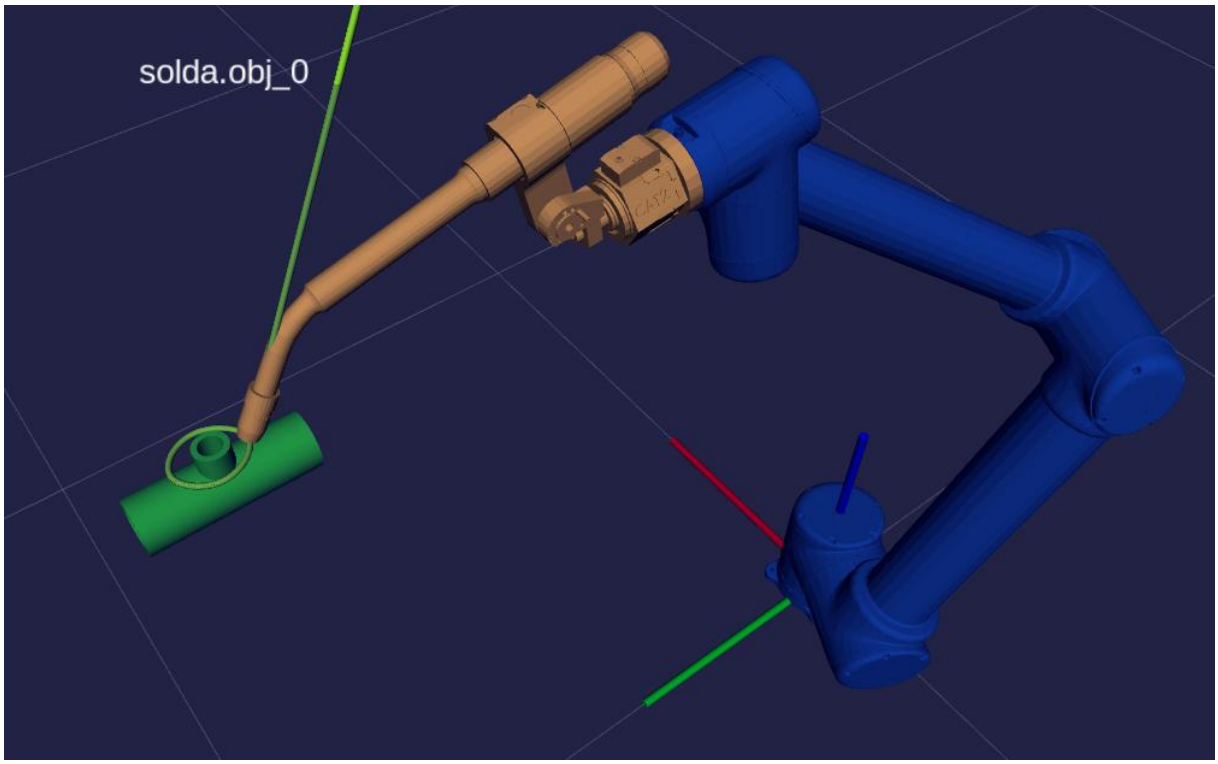
As Figuras 18 e 19 mostram, respectivamente, o gráfico gerado pela B-Spline calculada e a simulação no MoveIt! da aplicação de soldagem.

Figura 18 – B-Spline gerada para a simulação de aplicação real



Fonte: Autor.

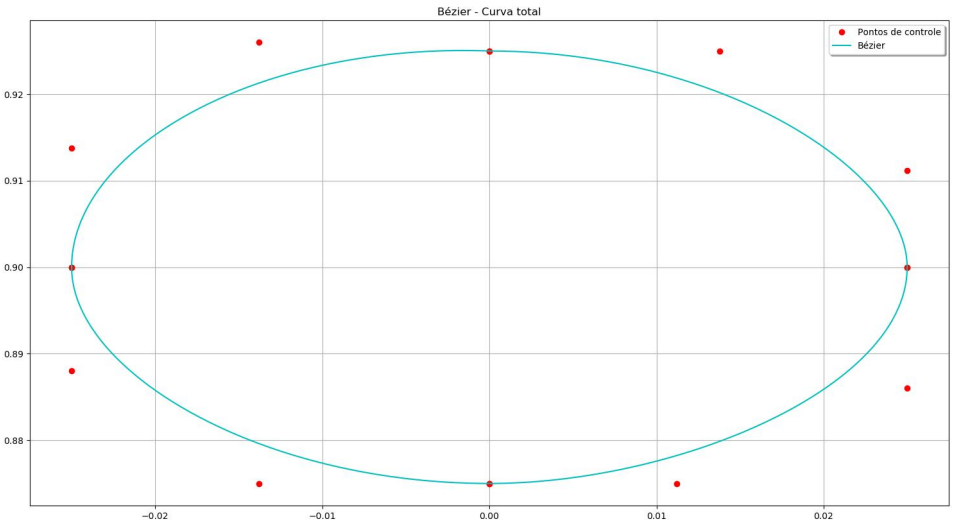
Figura 19 – Simulação de aplicação real com B-Splines



Fonte: Autor.

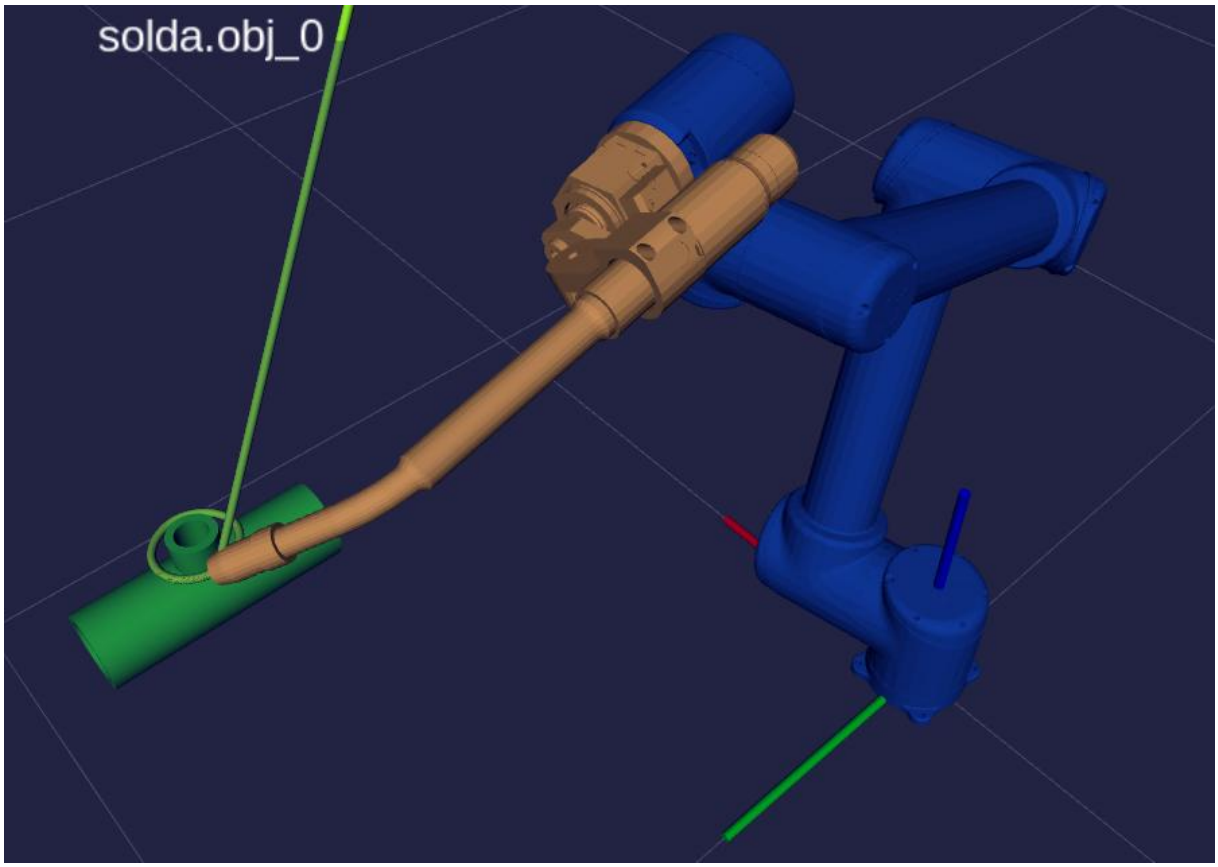
Na Figura 20 é possível verificar as curvas de Bézier calculadas para simular a aplicação de soldagem. A Figura 21 mostra a aplicação sendo simulada no MoveIt!.

Figura 20 – Curvas de Bézier geradas para a simulação de aplicação real



Fonte: Autor.

Figura 21 – Simulação de aplicação real com Curvas de Bézier



Fonte: Autor.

5 RESULTADOS

Inicialmente, buscou-se verificar se a trajetória simulada no MoveIt! era condizente com as curvas de Bézier e B-Splines calculadas nos scripts implementados. Para isso, foi feita uma comparação entre as curvas criadas através da implementação e os pontos gerados pela simulação no MoveIt!.

Para conseguir as posições, em tempo real, por onde a ponta do efetuador final passa em relação a base do manipulador foi utilizado o programa rosrun com o comando 5.1.

O rosrun é iniciado através do comando 5.1 no terminal do Linux. A função tf_echo do rosrun, que retorna a posição de um sistema de coordenadas em relação a outro, foi utilizada para receber os dados.

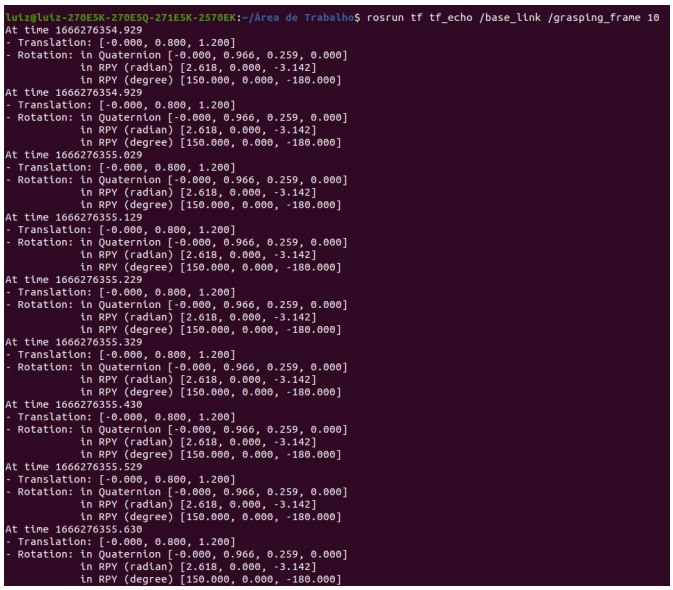
No exemplo utilizado, a posição retornada é a do sistema de coordenadas grasping_frame, localizado na ponta da tocha de solda, em relação ao sistema de coordenadas base_link, fixo na base do robô. O último parâmetro é a taxa de mensagens que a função irá retornar, no caso do comando 5.1 são 10Hz (10 mensagens por segundo). Os dados recebidos da execução do rosrun são utilizados para a comparação com as curvas criadas com os algoritmos implementados em Python.

Listing 5.1 – Comando utilizado para retornar a posição da ponta da tocha no Terminal

```
rosrun tf tf_echo /base_link /grasping_frame 10
```

A Figura 22 mostra o retorno do comando 5.1 no terminal.

Figura 22 – Resposta do comando 5.1



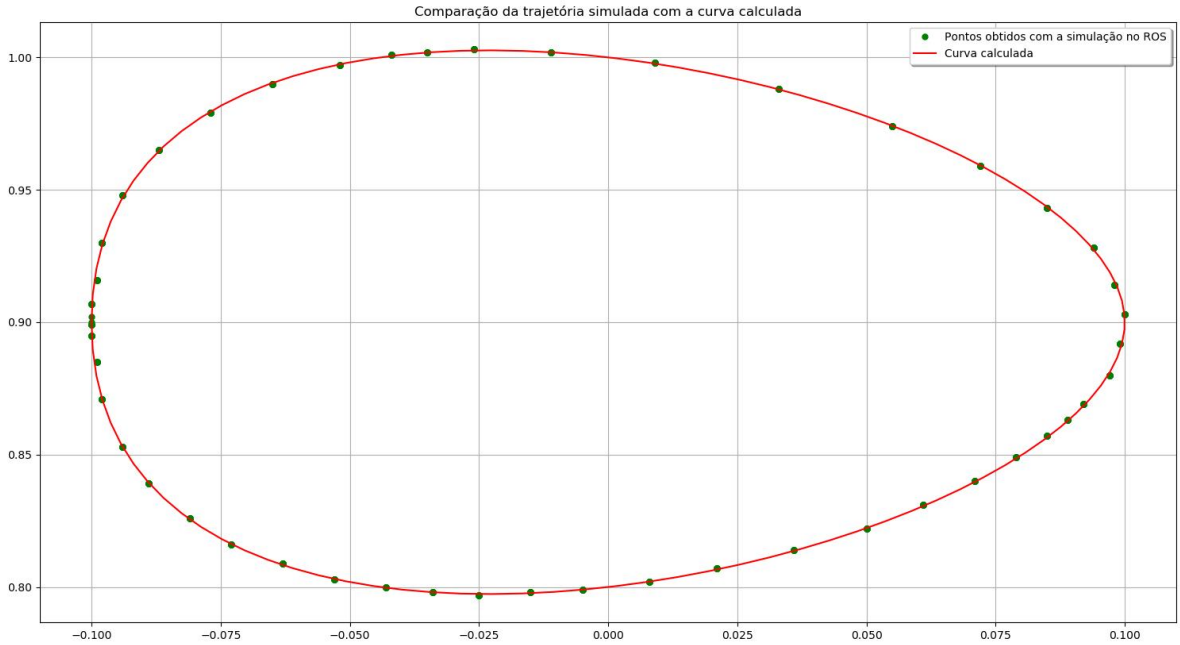
Fonte: Autor.

Como forma de otimizar o comando 5.1, foi utilizada a função grep para filtrar apenas a translação da ponta da tocha de solda em relação a base do manipulador e a saída do comando foi enviada para um arquivo de texto, buscando maior facilidade para ler os dados utilizando um script em Python.

As Figuras 23 e 24 mostram, respectivamente, a sobreposição dos pontos por onde a ponta do efetuador final passou nas simulações em relação a B-Spline e a Curva de Bézier calculadas nos exemplos bidimensionais do Capítulo 4. As curvas vermelhas representam a trajetória calculada, enquanto os pontos verdes representam os pontos obtidos através da trajetória simulada no MoveIt!.

As figuras mostram que os pontos gerados na trajetória simulada pelo MoveIt!, obtidos através do rosrun, estão sobrepostos a curva calculada.

Figura 23 – Comparação de B-Spline calculada com pontos da trajetória realizada no MoveIt!

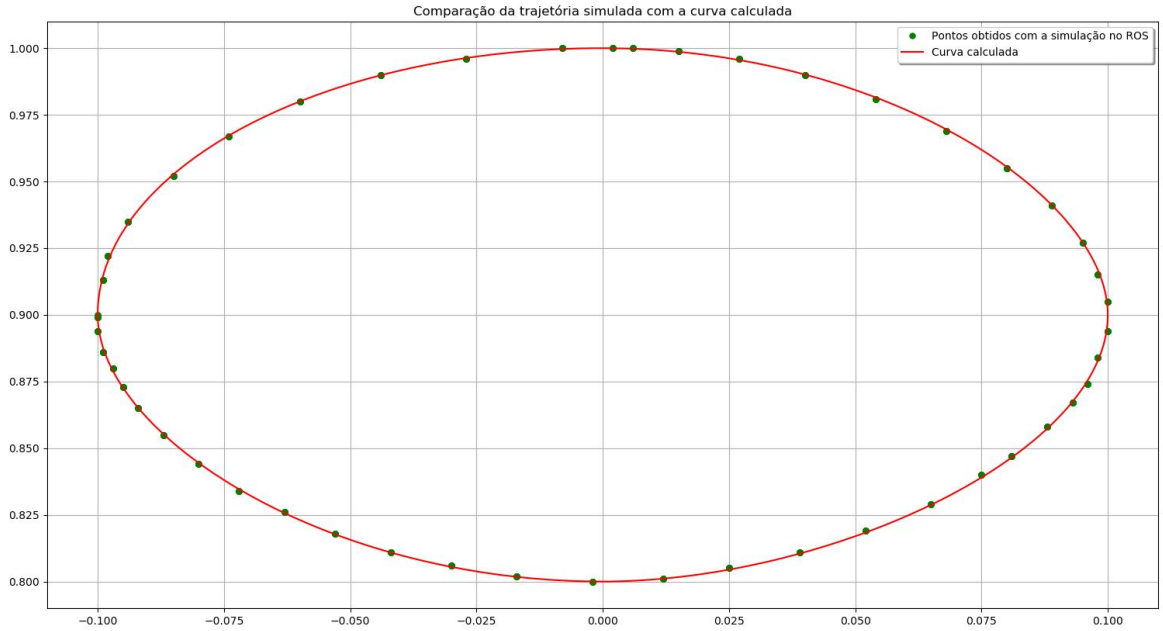


Fonte: Autor.

Fazendo a mesma sobreposição com os dados das posições obtidos para as curvas tridimensionais, também foi possível verificar que os pontos obtidos na simulação estão sobrepostos as curvas calculadas, como mostram as Figuras 25 e 26.

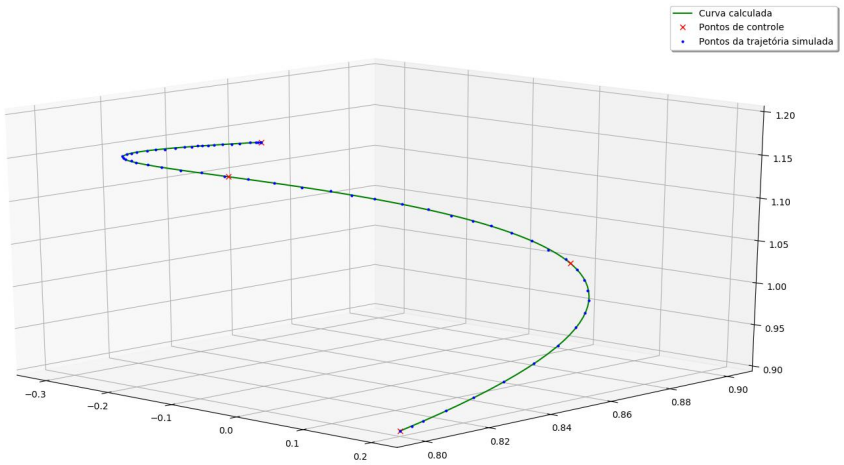
Dessa forma, foi possível verificar experimentalmente que o pacote ArmAgeddon consegue realizar de forma satisfatória as trajetórias que utilizam as curvas de Bézier e B-Splines calculadas em Python.

Figura 24 – Comparação de Curva de Bézier calculada com pontos da trajetória realizada no MoveIt!



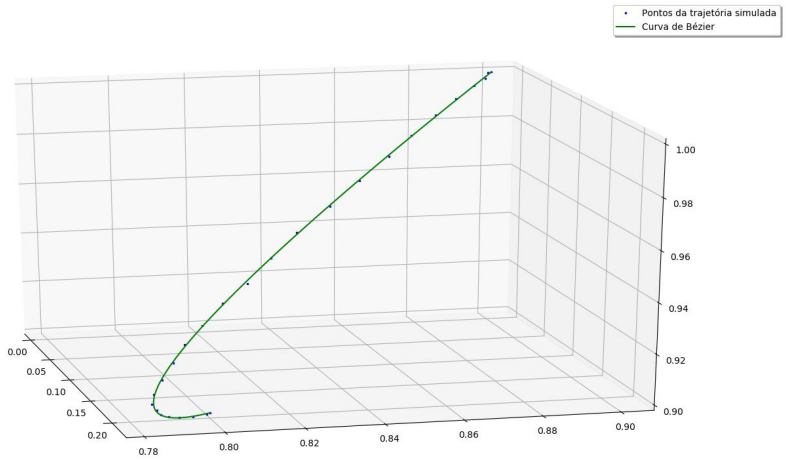
Fonte: Autor.

Figura 25 – Comparação de Curva B-Spline 3D calculada com pontos da trajetória realizada no MoveIt!



Fonte: Autor.

Figura 26 – Comparação de Curva de Bézier calculada com pontos da trajetória realizada no MoveIt!



Fonte: Autor.

6 CONCLUSÕES

6.1 CONSIDERAÇÕES FINAIS

O presente trabalho teve como objetivo o estudo de curvas B-Spline e Curvas de Bézier para o planejamento de trajetórias de manipuladores robóticas. Estas curvas por terem comportamento suave têm bastante utilidade no desenvolvimento dessas aplicações, pois trajetórias suaves, com posição, velocidade e aceleração descritas através de funções contínuas, garantem que o manipulador não sofra um desgaste excessivo.

A revisão bibliográfica mostra que as B-Splines e as Curvas de Bézier podem ser utilizadas para o planejamento de trajetórias robóticas, podendo interpolar pontos chave das aplicações de forma não prejudicial aos manipuladores, trazendo maior longevidade aos robôs, diminuindo o risco de falhas e diminuindo gastos com manutenção e paradas inesperadas durante o processo produtivo.

As simulações utilizando o MoveIt! ocorreram de forma satisfatória, representando curvas diversas que o manipulador poderia enfrentar durante o uso industrial, sendo bidimensionais, tridimensionais ou contornando objetos. As simulações demonstraram as limitações do projeto inicial do ArmAgeddon, especialmente no modelo de colisão, que precisaram ser corrigidas para simulações mais condizentes com a realidade.

6.2 MELHORIAS FUTURAS

O pacote ArmAgeddon pode ser expandido de forma que outros tipos de efetuadores finais possam ser utilizados, como garras ou ventosas para o uso em aplicações de pick and place, ferramentas para usinagem ou pistolas para pintura, entre outras aplicações. A alteração demandaria a criação de um modelo 3D e a alteração dos modelos URDF e de colisão do robô.

O ArmAgeddon também pode ser utilizado como base para o estudo mais avançado de aplicações de manipuladores robóticos na indústria. O estudo de técnicas de soldagem com o manipulador, assim como o controle de uma ferramenta de usinagem, com variações nos tipos de ferramentas e no controle do avanço ferramental, pode ser feito.

O estudo de outras aplicações para o robô também possibilitaria que outras curvas pudessem ser utilizadas para o planejamento das trajetórias. Um exemplo seria o uso de NURBS para o planejamento de trajetórias que não necessitam que todos os pontos de interesse sejam interpolados.

Também é possível sugerir como uma melhoria futura para o pacote, o uso de planejamento de trajetórias baseadas em superfícies, sendo possível a utilização de superfícies de Bézier para modelar as superfícies dos objetos de interesse. O planejamento de trajetórias utilizando superfícies facilitaria no uso de aplicações de pintura, usinagem e solda.

REFERÊNCIAS

BARTH, R. et al. Using ROS for agricultural robotics - design considerations and experiences. In: **Proceedings of the Second International Conference on Robotics and associated High-technologies and equipment for agriculture and forestry**. Madrid: RHEA, 2014. p. 509–518.

BEY, M.; BOUDJOUAD, S.; TAFAT-BOUZID, N. Tool-path generation for free-form surfaces with b-spline curves. **Strojniški vestnik - Journal of Mechanical Engineering**, v. 53, n. 11, p. 733–741, 2007. ISSN 0039-2480.

CHEN, W. et al. Path planning scheme for spray painting robot with bézier curves on complex curved surfaces. In: **2017 32nd Youth Academic Annual Conference of Chinese Association of Automation (YAC)**. Hefei, China: IEEE, 2017. p. 698–703.

COLEMAN, D. et al. Reducing the barrier to entry of complex robotic software: a moveit! case study. **Journal of Software Engineering for Robotics**, p. 3–16, May 2014.

COTA, E. et al. Robótica na mineração. In: **ABM Proceedings - 18º Simpósio de Mineração**. São Paulo: Editora Blucher, 2017. p. 359–370.

CRAIG, J. J. **Robótica**. 3. ed. São Paulo: Pearson Education do Brasil, 2012.

DENG, H.; XIONG, J.; XIA, Z. Mobile manipulation task simulation using ROS with MoveIt. In: **2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)**. Okinawa, Japão: IEEE, 2017. p. 612–616.

DIAS, A. et al. Uma metodologia para geração de trajetórias de manipuladores no espaço cartesiano. In: **Congresso Nacional de Engenharia Mecânica - ABCM CONEM**. Natal, RN: Congresso Nacional em Engenharia Mecânica, 2000.

DUARTE, P. V. **Simulação de Robôs Antropomórficos com Moveit**. Trabalho de Conclusão de Curso (Graduação em Engenharia Mecatrônica) — Centro Tecnológico de Joinville, Universidade Federal de Santa Catarina, Joinville, 2020.

FERREIRA, W. R. B. **Planejamento de Trajetórias Robóticas Utilizando B-Splines**. Dissertação (Mestrado em Engenharia Mecânica) — Faculdade de Engenharia Mecânica, Universidade Federal de Uberlândia„ Uberlândia, MG, 2011.

FRENSEL, K.; DELGADO, J. **Geometria Analítica. Capítulo 2. Notas de aula**. 2010. Disponível em: https://www.professores.uff.br/katiafrensel/wp-content/uploads/sites/115/2017/08/aula2.pdf. Acesso em: 09 abr 2022.

GATESICHAPAKORN, S.; TAKAMATSU, J.; RUCHANURUCKS, M. ROS based autonomous mobile robot navigation using 2D LiDAR and RGB-d camera. In: **2019 First International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP)**. Bancoque, Tailândia: IEEE, 2019. p. 151–154.

HERNANDEZ-MENDEZ, S. et al. Design and implementation of a robotic arm using ros and moveit! In: **2017 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)**. Ixtapa, Mexico: IEEE, 2017. p. 1–6.

INTERNATIONAL FEDERATION OF ROBOTICS. **IFR presents World Robotics Report 2020**. 2020. Disponível em: https://ifr.org/ifr-press-releases/news/record-2.7-million-robots-work-in-factories-around-the-globe. Acesso em: 05 mar. 2022.

JIAO, J. et al. Bezier curve based path planning for a mobile manipulator in unknown environments. In: **2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)**. Shenzhen, China: IEEE, 2013. p. 1864–1868.

LENTIN JOSEPH, J. C. **Mastering ROS for Robotics Programming**. 2. ed. Birmingham, Reino Unido: Packt Publishing, 2018.

KRäMER, M. et al. Model predictive control of a collaborative manipulator considering dynamic obstacles. **Optimal Control Applications and Methods**, v. 41, n. 4, p. 1211–1232, 2020. Acesso em: 06 mar 2022.

LIU, X. et al. Time-energy optimal trajectory planning for collaborative welding robot with multiple manipulators. **Procedia Manufacturing**, v. 43, p. 527–534, 2020. ISSN 2351-9789. Sustainable Manufacturing - Hand in Hand to Sustainability on Globe: Proceedings of the 17th Global Conference on Sustainable Manufacturing.

OPEN ROBOTICS. **ROS/Concepts**. 2014. Disponível em: https://wiki.ros.org/ROS/Concepts. Acesso em: 06 fev. 2022.

OPEN ROBOTICS. **ROS/Introduction**. 2018. Disponível em: https://wiki.ros.org/ROS/Introduction. Acesso em: 06 fev. 2022.

PICKNIK ROBOTICS. **Motion Planning**. 2022. Disponível em: https://moveit.picknik.ai/galactic/doc/concepts/motion_planning.html. Acesso em: 06 fev. 2022.

PICKNIK ROBOTICS. **MoveIt 2 Documentation**. 2022. Disponível em: https://moveit.picknik.ai/galactic/index.html. Acesso em: 06 fev. 2022.

PIEGL, L.; TILLER, W. **The NURBS Book**. 2. ed. Berlim: Springer, 1997.

RYBUS, T. et al. Experimental demonstration of singularity avoidance with trajectories based on the bézier curves for free-floating manipulator. In: **9th International Workshop on Robot Motion and Control**. Kuslin, Polônia: IEEE, 2013. p. 141–146.

SICILIANO, B. et al. **Robotics: Modelling, planning and control**. 1. ed. London: Springer, 2010.

SILVA, C. J. C. do Vale Fernandes da. **Planejamento de trajetórias e navegação de robôs móveis**. 94 p. Dissertação (Mestrado em Engenharia Mecânica) — Centro de Tecnologia. Universidade Federal do Rio Grande do Norte, Natal, RN, 2015.

SIMONI, R. **Teoria Local de Curvas**. Trabalho de Conclusão de Curso (Licenciatura em Matemática) — Centro de Ciências Físicas e Matemáticas, Universidade Federal de Santa Catarina, Florianópolis, 2005.

STEUBEN, J.; STEELE, J.; TURNER, C. Nurbs for robot manipulator trajectory generation. In: **Volume 6: 35th Mechanisms and Robotics Conference, Parts A and B**. Washington, DC, EUA: ASMEDC, 2011. p. 1121–1130.

TONETTO, C. P. **Uma proposta de sistematização do processo de planejamento de trajetórias para o desenvolvimento de tarefas de robôs manipuladores**. Dissertação (Mestrado em Engenharia Mecânica) — Programa de Pós-Graduação em Engenharia Mecânica, Universidade Federal de Santa Catarina, Florianópolis, SC, 2007.

TYAPIN, I.; HOVLAND, G. Long arm manipulator path interpolation using 4th order b-splines. In: **2018 18th International Conference on Control, Automation and Systems (ICCAS)**. Pyeongchang, Coreia do Sul: IEEE, 2018. p. 1698–1702.

UNIVERSAL ROBOTS. **Technical specifications UR10**. Odense, Dinamarca, 2014. Disponível em: https://www.universal-robots.com/media/50880/ur10_bz.pdf. Acesso em: 07 mar 2022.

APÊNDICE A

Código em Python que gera as Curvas B-Splines, escreve em um arquivo de texto os pontos que formam a curva e gera os gráficos das curvas em formato bidimensional, com as variações nos eixos X e Y, e tridimensional.

```
import numpy as np
from scipy import interpolate
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

curva = input("Curva: ")
curva = int(curva)

#Escreve os parametros no arquivo conforme o arquivo de teste
    da biblioteca original
f = open("../TCC/src/armageddon_robot/arquivos/teste.md", "w")
f.write("# Positions" + "\n\n")
f.write("```c++ \n")
f.write("MaxVelocity 0.01 MaxAcceleration 0.01 PlanningTime 5.0
    \n")
f.write("EndEffectorStep 0.01 JumpThreshold 0.0 \n")
f.write("AvoidCollisions 1 \n```\n\n")
f.write("| X | Y | Z | RX | RY | RZ |\n")
f.write("|--------|-------|-------|-----|----|----|")

if(curva == 1):
        #Aleatorio
        ctr = np.array([(0.2, 0.8, 0.9),
                (0, 0.9, 1),
                (-0.3, 0.85, 1.1),
                (0, 0.8, 1.2)])

if(curva == 2):
        #Circulo
        ctr = np.array([(-0.1, 0.9, 1.15),
                (0, 1, 1.15),
                (0.1, 0.9, 1.15),
```

```
                (0, 0.8, 1.15),
                (-0.1, 0.9, 1.15)])

if(curva == 3):
        #Solda
        #P1 = (0, 0.88), P2 = (-0.01732; 0.89), P3 = (-0.01732;
            0.91), P4 = (0; 0.925), P5 = (0.01732; 0.91), P6 =
           (0.01732; 0.89)
        ctr = np.array([(0, 0.88-0.01, 1.035),
                (-0.01732, 0.89-0.01, 1.035),
                (-0.01732-0.01, 0.91, 1.035),
                (0, 0.925+0.01, 1.035),
                (0.01732+0.01, 0.91, 1.035),
                (0.01732, 0.89-0.01, 1.035),
                (0, 0.88-0.01, 1.035)])

xControle = ctr[:,0]
yControle = ctr[:,1]
zControle = ctr[:,2]


tck, u = interpolate.splprep([xControle,yControle,zControle], k
   =3, s=0)
u = np.linspace(0,1, num=100, endpoint=True)
pontos = interpolate.splev(u, tck)

cont=0
i=0
angulo = ()
for i in pontos[0]:
        pontoString = "\n|␣" + str("%.5f" % pontos[0][cont]) +
           "␣|␣" + str("%.5f" % pontos[1][cont]) + "␣|␣" + str(
           "%.5f" % pontos[2][cont]) + "␣|␣" + str(150) + "␣|␣"
            + str(0) + "␣|␣" + str(180) + "␣|"
        cont+=1
        f.write(pontoString)

f.close()
```

```
# Plot 3D
fig2 = plt.figure(2)
ax3d = fig2.add_subplot(111, projection='3d')
ax3d.plot(pontos[0], pontos[1], pontos[2], 'b', color='green',
    label='Curva calculada')
ax3d.plot(xControle, yControle, zControle, 'x', color='red',
    label='Pontos de controle')
ax3d.legend(loc='best', fancybox=True, shadow=True)
fig2.show()
plt.show()

# Plot 2D
plt.title("Comparacao da B-Spline com os pontos da trajetoria")
plt.plot(pontos[0],pontos[1], color='green', label='Curva
    calculada')
plt.plot(xControle, yControle, 'x', color='red', markersize=10,
     label='Pontos de controle')
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid()
plt.show()
```

APÊNDICE B

Código em Python que gera as Curvas de Bézier, escreve em um arquivo de texto os pontos que formam a curva e gera os gráficos das curvas em formato bidimensional, com as variações nos eixos X e Y, e tridimensional.

```
import numpy as np
from pontosBezier import retornaPonto
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

#Funcao que calcula os pontos das curvas de Bezier
def calcPontos(x, y, z, rx, ry, rz):
        pontos = []
        cont = 0
        angulo=[]
        for u in i:
                x1 = ((1-u)**3)*x[0] + 3*u*((1-u)**2)*x[1] +
                   3*(1-u)*(u**2)*x[2] + (u**3)*x[3]
                y1 = ((1-u)**3)*y[0] + 3*u*((1-u)**2)*y[1] +
                   3*(1-u)*(u**2)*y[2] + (u**3)*y[3]
                z1 = ((1-u)**3)*z[0] + 3*u*((1-u)**2)*z[1] +
                   3*(1-u)*(u**2)*z[2] + (u**3)*z[3]
                ponto = [x1,y1, z1]
                angulo = [rx, ry, rz]
                pontoString = "\n|␣" + str(ponto[0]) + "␣|␣" +
                   str(ponto[1]) + "␣|␣" + str(ponto[2]) + "␣|␣
                   " + str(angulo[0]) + "␣|␣" + str(angulo[1])
                   + "␣|␣" + str(angulo[2]) + "␣|"
                f.write(pontoString)
                pontos.append(ponto)
                cont += 1
        return pontos


#Escreve os parametros no arquivo conforme o arquivo de teste
   da biblioteca original
f = open("../TCC/src/armageddon_robot/arquivos/teste.md", "w")
```

```
f.write("#Positions" + "\n\n")
f.write("‘‘‘c++␣\n")
f.write("MaxVelocity␣0.05␣MaxAcceleration␣0.05␣PlanningTime␣5.0
   ␣\n")
f.write("EndEffectorStep␣0.01␣JumpThreshold␣0.0␣\n")
f.write("AvoidCollisions␣1␣\n‘‘‘\n\n")
f.write("|␣X␣|␣Y␣|␣Z␣|␣RX␣|␣RY␣|␣RZ␣|\n")
f.write("|␣--------␣|␣-------␣|␣-------␣|␣-----␣|␣----␣|␣----␣|
   ")

i = np.arange(0.0,1.01,0.01)

curva = input("Curva:␣")
curva = int(curva)
#Curva aleatoria tridimensional
if(curva == 1):
        #Recebe os pontos que estao no arquivo pontosBezier.py
        x1, y1, z1, ang1, ang2, ang3 = retornaPonto(curva)

        #Coloca todos os pontos de controle na mesma variavel
        xControle = x1
        yControle = y1
        zControle = z1

        #Calcula a curva de Bezier
        pontos = calcPontos(x1, y1, z1, ang1, ang2, ang3)

#Curva circular bidimensional
if(curva == 2):
        #Recebe os pontos que estao no arquivo pontosBezier.py
        x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4, ang1,
           ang2, ang3 = retornaPonto(curva)

        #Coloca todos os pontos de controle na mesma variavel
        xControle = x1+x2+x3+x4
        yControle = y1+y2+y3+y4
        zControle = z1+z2+z3+z4
```

```
        #Calcula as 4 curvas e junta todos os pontos na
           variavel pontos
        pontos1 = calcPontos(x1, y1, z1, ang1, ang2, ang3)
        pontos2 = calcPontos(x2, y2, z2, ang1, ang2, ang3)
        pontos3 = calcPontos(x3, y3, z3, ang1, ang2, ang3)
        pontos4 = calcPontos(x4, y4, z4, ang1, ang2, ang3)
        pontos = pontos1+pontos2+pontos3+pontos4

#Curva com objeto
if(curva == 3):
        #Recebe os pontos que estao no arquivo pontosBezier.py
        x1, y1, z1, ang11, ang21, ang31, x2, y2, z2, ang12,
           ang22, ang32, x3, y3, z3, ang13, ang23, ang33, x4,
           y4, z4, ang14, ang24, ang34 = retornaPonto(3)

        #Coloca todos os pontos de controle na mesma variavel
        xControle = x1+x2+x3+x4
        yControle = y1+y2+y3+y4
        zControle = z1+z2+z3+z4

        #Calcula as 4 curvas e junta todos os pontos na
           variavel pontos
        pontos1 = calcPontos(x1, y1, z1, ang11, ang21, ang31)
        pontos2 = calcPontos(x2, y2, z2, ang12, ang22, ang32)
        pontos3 = calcPontos(x3, y3, z3, ang13, ang23, ang33)
        pontos4 = calcPontos(x4, y4, z4, ang14, ang24, ang34)
        pontos = pontos1+pontos2+pontos3+pontos4

f.close()

pontosArray = np.array(pontos)

Xs = pontosArray[:,0]
Ys = pontosArray[:,1]
Zs = pontosArray[:,2]

#Plot 2d
plt.title("Bezier - Curva total")
```

```
plt.plot(xControle, yControle, 'ro', label="Pontos de controle"
    )
plt.plot(Xs, Ys, '-c', label="Curva de Bezier")
plt.legend(loc='best', fancybox=True, shadow=True)
plt.grid()
plt.show()

#Plot 3D
fig2 = plt.figure(2)
ax3d = fig2.add_subplot(111, projection='3d')
ax3d.plot(Xs, Ys, Zs, 'b', color='green', label='Curva de
    Bezier')
ax3d.plot(xControle, yControle, zControle, 'o', color='red',
    label='Pontos de controle')
ax3d.legend(loc='best', fancybox=True, shadow=True)
fig2.show()
plt.show()
```