INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
CAMPUS TERESINA CENTRAL
Engenharia Mecânica

BRUNO CLAUDIO DE ARAUJO FRANCO

SIMULAÇÃO ROBÓTICA DE MANIPULADORES INDUSTRIAIS UTILIZANDO SOFTWARES DE CÓDIGO ABERTO: UM ESTUDO SOBRE A APLICABILIDADE DESTAS FERRAMENTAS COMO RECURSO EDUCACIONAL, TECNOLÓGICO E CIENTÍFICO

TERESINA, PIAUÍ

2021

BRUNO CLAUDIO DE ARAUJO FRANCO

SIMULAÇÃO ROBÓTICA DE MANIPULADORES INDUSTRIAIS UTILIZANDO SOFTWARES DE CÓDIGO ABERTO: UM ESTUDO SOBRE A APLICABILIDADE DESTAS FERRAMENTAS COMO RECURSO EDUCACIONAL, TECNOLÓGICO E CIENTÍFICO

Projeto apresentado à Banca Examinadora como requisito para aprovação na disciplina de Trabalho de Conclusão de Curso II do Curso Superior de Engenharia Mecânica do Instituto Federal de Educação, Ciência e Tecnologia do Piauí.

Orientador: Prof. MSc. Francisco Marcelino Almeida de Araújo

TERESINA, PIAUÍ

2021

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD

Franco, Bruno Claudio de Araujo

F825s Simulação robótica de manipuladores industriais utilizando softwares de código aberto : um estudo sobre a aplicabilidade destas ferramentas como recurso educacional, tecnológico e científico / Bruno Claudio de Araujo Franco. - 2021.

74 f.: il. color.

Trabalho de Conclusão de Curso (Bacharelado em Engenharia Mecânica) - Instituto Federal do Piauí, Campus Teresina Central, 2021.

Orientador : Prof. Me. Francisco Marcelino Almeida de Araújo.

1. Manipulação robótica . 2. Simulação robótica. 3. ROS (Robotics Operating System). 4. Gazebo. 5. Educação. I.Título.

CDD - 620

Elaborado por Sindya Santos Melo CRB 3/1085

BRUNO CLAUDIO DE ARAUJO FRANCO

SIMULAÇÃO ROBÓTICA DE MANIPULADORES INDUSTRIAIS UTILIZANDO SOFTWARES DE CÓDIGO ABERTO: UM ESTUDO SOBRE A APLICABILIDADE DESTAS FERRAMENTAS COMO RECURSO EDUCACIONAL, TECNOLÓGICO E CIENTÍFICO

Projeto apresentado à Banca Examinadora como requisito para aprovação na disciplina de Trabalho de Conclusão de Curso II do Curso Superior de Engenharia Mecânica do Instituto Federal de Educação, Ciência e Tecnologia do Piauí.

Aprovado pela banca examinadora em _____.

BANCA EXAMINADORA:

Prof. MSc. Francisco Marcelino Almeida de Araújo
Instituto Federal de Educação, Ciência e Tecnologia do Piauí - IFPI

Prof. Me. Helton Rodrigo de Souza Sereno
Instituto Federal de Educação, Ciência e Tecnologia do Rio de Janeiro - IFRJ

Prof. Dr. Nuno Miguel Fonseca Ferreira
Instituto Politécnico de Coimbra

TERESINA, PIAUÍ

2021

Dedico este trabalho a todos amigos, familiares e tutores que estiveram envolvidos.

AGRADECIMENTOS

Gostaria de agradecer primeiramente a **Edina Maria Galvão de Araujo**, minha mãe, por todo carinho e apoio nas horas que mais precisei. Gostaria de agradecer por ser seu filho e poder seguir seu exemplo de dedicação, perseverança e esforço. Agradeço por sua saúde e poder compartilhar este momento da minha vida profissional.

Sinto uma alegria e conforto em compartilhar minha trajetória com meus amigos e, em especial a **Francisco Sávio, Erbert Barbosa, Carlos Victor, Yago Pablo, Francisco das Chagas** por sempre acreditar no meu potencial e sempre incentivar a dedicar o meu máximo. Gostaria de Agradecer também a **Ana Beatriz Moraes Baptista**, minha noiva, por seu carinho e paciência quando precisei dedicar madrugadas em meus projetos.

Muitos foram os desafios enfrentados neste processo de aprendizagem e fazer parte de um projeto com pessoas inteligentes e dedicadas fez com que ele tornasse mais motivador. Em especial, gostaria de agradecer a todos da minha equipe do projeto de robótica no laboratório Labiras, sem **Joselito Lima, Gabriel Duarte e Lucas Dutra** pouco conseguiria fazer neste trabalho. Obrigado por me permitir guiar nossos projetos, obrigado por todo o esforço, obrigado por sempre trazer ideias incríveis. Espero que possamos colaborar em trabalhos mais incríveis no futuro.

Nestes projetos e disciplinas foram necessários diversos tutores, uma lista tão extensa que não caberia aqui, foram fundamentais para que pudesse entender assuntos complexos e conseguir executar melhor meus trabalhos. Gostaria de agradecer em especial dois mestres que me acompanharam com maior proximidade, obrigado **Prof. MSc Francisco Marcelino Almeida de Araújo** por permitir aprender mais sobre tecnologia, por sempre me instigar a ser um profissional melhor e me permitir conhecer pessoas incríveis que atuam internacionalmente com Robótica, participar do Labiras foi uma experiência incrível, foi o projeto em que aprendi a sonhar grande. Obrigado **Prof. MSc Ricardo Soares** por todos os concelhos, por acreditar no meu potencial, por incentivar meu crescimento acadêmico com cursos e treinamentos, por me ensinar a ser mais pragmático e a apreciar o conhecimento.

Por fim, agradeço a todos que estão lendo este trabalho e que me permita melhorá-lo com suas sugestões. Espero que gostem do resultado.

"Dor é temporária. Pode durar um minuto, uma hora, um dia ou até mesmo um ano, mas eventualmente ela será diminuída e então alguma coisa mais valiosa vai tomar o lugar dela. Se eu desistir, entretanto, vai durar para sempre"

(Lance Amstrong, tradução livre)

RESUMO

A simulação robótica de manipuladores industriais oferece possibilidades para o desenvolvimento ágil que o mercado necessita dentro do contexto da Industria 4.0, além de permitir utilizar seus desafios para o ensino ativo de diversas disciplinas, como matemática, física e programação, já que é uma área multidisciplinar. Dentro do contexto da simulação robótica, o presente trabalho tem como proposta avaliar a utilização de ferramentas de código aberto ROS, Moveit e Gazebo, tendo como objeto de estudo, a análise de trajetórias, para então discutir a utilização delas como recurso de ensino, pesquisa e desenvolvimento no ambiente do Instituto Federal do Piauí Campus Central. Para isso foi feito um breve levantamento bibliográfico acerca das ferramentas utilizadas para simulação robótica. Em adjunto, foi feito o estudo dos fundamentos da Robótica de Manipuladores, contemplando os desafios da Cinemática, Dinâmica, Controle e Planejamento de Trajetória. Logo após, foi definido os parâmetros necessários para a simulação, obtidos nos documentos oficiais do manipulador escolhido, ABB do modelo Irb1200 5.90, e nos arquivos do repositório Ros-Industrial, onde foi obtido as variáveis dinâmicas e de controle. Para a modelagem cinemática foi utilizado o modelo do Produto das Exponenciais. Para a trajetória foi utilizado o pacote de arquivos(*Packages*) do Joint_Trajectory_Control e para o controle foi utilizado o Ros_Control na condição de controle de trajetória, ambos utilizados no repositório GIT Ros_industrial. De forma geral foi possível fazer o manipulador executar as trajetórias do tipo Ponto-a-Ponto, parametrizada e Linear de forma satisfatória, porém é necessário utilizar outros modelos para desenvolver curvas suaves de aceleração e controladores mais robustos. Além de que, a visualização dos gráficos de posição, velocidade e aceleração não se deu de forma satisfatória utilizando o ROS, então foi necessário desenvolver algoritmos para melhorar a visualização destes dados. No contexto da educação, os softwares mostraram-se relevantes devido a não apresntar custo de aquisição, poder executar as soluções de forma segura e descentralizada do Hardware do Robô. No contexto da pesquisa e desenvolvimento é necessário utilizar modelos de controle de trajetória mais sofisticados do que utilizados neste trabalho.

Palavras-chave: Manipulação Robótica. Simulação Robótica. ROS. Gazebo. Educação.

ABSTRACT

Robotic simulation of industrial manipulators offers possibilities for the agile development that the market requires within the context of Industry 4.0, in addition to allowing the use of its challenges for the active teaching of various disciplines, such as mathematics, physics, and programming, as it is a multidisciplinary area. Within the context of robotic simulation, this work proposes to evaluate the use of open source tools ROS, Moveit, and Gazebo, having as object of study, an analysis of trajectories, to then discuss their use as teaching, research, and development in the environment of the Federal Institute of Piauí Campus Central. For this, a brief bibliographical survey on tools used for robotic simulation was carried out. In an adjunct, a study of the fundamentals of Manipulator Robotics was carried out, contemplating the challenges of Kinematics, Dynamics, Control, and Trajectory Planning. Than, the parameters for the simulation were defined, from the official documents of the chosen manipulator, ABB of the Irb1200 5.90 model, and from the files of the Ros-Industrial repository, where they were created as dynamic and control variables. For kinematic modeling, the Product of Exponential model was used. For the trajectory, we used the file packages from the Joint_Trajectory_Control and to the control was used the Ros_Control in the trajectory control condition, both used in the GIT Ros_industrial repository. In general, it was possible to make the manipulator execute as Point-to-Point, Parameterized, and Linear trajectories satisfactorily, but it was not possible, using the standard Moveit packages, to develop smooth velocity and acceleration curves, as recommended in the literature. Besides, the visualization of the position, velocity, and acceleration graphs was not done satisfactorily using ROS, so it was necessary to develop algorithms to improve the visualization of these data. In the context of education, the fundamental software is relevant due to not presenting acquisition cost, being able to execute the solutions in a safe and decentralized way of the Robot's Hardware. In the context of research and development, it is necessary to use more sophisticated trajectory control models than those used in this work.

Key-words: Robotic manipulation. Robotic Simulation. ROS. Gazebo. Education.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
ROS	Robotic Operating System
DH	Denavit Hartenberg
PoE	Product of Exponents
eef	End Efector

SUMÁRIO

1 INTRODUÇÃO

Segundo a International Federation of Robotics (2017), ao final de 2016, 1.8 milhões de robôs industriais haviam sido instalados mundialmente, com uma taxa anual de crescimento de 10% desde 2010, e para continuar este crescimento, se faz necessário recursos para desenvolver novas tecnologias de forma eficiente, neste contexto a Simulação robótica pode ser utilizada para atender esta demanda(AKSU; MICHALOSKI; PROCTOR, 2018). Por outro lado, se faz necessário iniciativas de ensino, pesquisa e desenvolvimento para aprimorar o aprendizado de alunos para que consigam estar preparados para estas tecnologias e ter vantagem competitiva em suas carreiras (ÖZCAN, 2019).

Dentro do contexto da robótica, a quarta revolução industrial, conhecida como indústria 4.0, compreende tecnologias como robótica, Internet das Coisas (Internet of Things - IoT), nuvem e simulação. Estas tecnologias são utilizadas em produção de bens de qualidade com incremento de diversidade de produtos e muitas vezes garantindo custos mais baixos, graças a técnicas de otimização e técnicas de produção inteligente (GUNAL, 2019)

E segundo Guna, 2019, a simulação computacional é intrínseca a vários desses conceitos e tecnologias da Indústria 4.0, por exemplo, a aplicação de simulação em análise de dados, modelagem híbrida ou Digital Twins, treinamento baseado em simulação e projeto de produto simulados. Além de que a simulação garante maior agilidade para testar rotinas, prototipar soluções sem paradas na produção ou utilização do robô real e, com isso, garantir maior adaptabilidade, segurança para os humanos que interagem com estas máquinas e menos custos com erros na implantação de funcionalidades, por exemplo (JUNG et al., 2020).

Por outro lado, existe o esforço para integrar a aplicação simulada com as condições do ambiente real no qual o manipulador se encontra. Somados a isso, se tem a dificuldade de desenvolver softwares para robótica pois é um processo complexo que necessita um escopo de conhecimento vasto, como por exemplo os desafios de análise cinemática, dinâmica, de controle e planejamento de trajetória e enfrenta a realidade de que sistemas robóticos estão constantemente evoluindo e em diversas áreas.

Pensando nisso, softwares como Ros e Gazebo foram desenvolvidos. O primeiro deles se trata de um Framework e Midware, que permite a reutilização de códigos e facilita a comunicação entre hardware e software, além de ser de código aberto e apresentar uma comunidade ativa com diversas soluções prontas. Já o segundo se trata de um simulador também de código aberto, com uma extensa biblioteca de modelos de robôs, apresenta possibilidade para o usuário colocar mais complexidade à simulação como variáveis de atrito, arrasto, entre outros. E o Gazebo juntamente com o ROS permite uma maior portabilidade da simulação para o robô real, ou seja, melhor integração entre desenvolvimento e aplicação.

Tendo vista os recursos da simulação para o desenvolvimento da Industria 4.0 e atender a necessidade de desenvolvimento de soluções robóticas de maneira ágeis, além de ter disponível ferramentas de código aberto como Ros e Gazebo, que se ver uma oportunidade para implementar estes recursos em instituições de ensino para o desenvolvimento de tecnologias e ensino de robótica e engenharia.

Contudo, muitas instituições de ensino ainda utilizam os modelos tradicionais para ensinar, no qual o aluno participa de maneira passiva na construção do seu conhecimento (HARA et al., 2020), (HABIB, 2020). Em contrapartida, a robótica oferece possibilidades de utilizar seus desafios para o ensino ativo de diversas disciplinas, como matemática, física e a programação, já que é uma área multidisciplinar. Com relação direta na aplicação com a programação, pois os robôs fornecem aos alunos um modelo físico para demonstrar visualmente conceitos e ideias (CALVO et al., 2017),(HISAZUMI et al., 2018),(CASAN et al., 2015) .

Dito isso, o presente trabalho tem como objetivo utilizar ambientes de simulação de ferramentas de código aberto para analisar sua utilização como recurso educacional e de desenvolvimento de tecnologia para ser implementado na rede de ensino dos Institutos Federais do Piauí Campus Central. Para isso será utilizado como objeto de estudo o planejamento de trajetórias pois se trata de um desafio recorrente na adaptabilidade dos robôs de forma ágil e então avaliado os parâmetros de disponibilidade de equipamentos, facilidade do usuário e custos de implementação das ferramentas.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O presente trabalho tem como objetivo geral o estudo da implementação dos softwares de código aberto ROS e Gazebo voltados ao ensino, pesquisa e desenvolvimento de sistemas robóticos.

1.1.2 Objetivos Específicos

- Pesquisa bibliográfica sobre as ferramentas de simulação robótica e os procedimentos para utilizá-las.
- Pesquisa bibliográfica sobre os conceitos fundamentais da análise de trajetórias;
- Descrever o robô de acordo com os conceitos da Cinemática de manipuladores;
- Descrever o robô de acordo com os conceitos da Dinâmica de manipuladores.
- Implementar os conceitos fundamentais nas ferramentas de simulação.
- Analisar diferentes tipos de trajetória nas ferramentas de simulação.
- Analisar e discutir sobre a viabilidade da aplicação das ferramentas no contexto dos Institutos Federais de Ensino.

RESUMO

Os próximos capítulos estão organizados da seguinte forma:

- **Fundamentação Teórica:** Neste capítulo são apresentados todos os conceitos teóricos utilizados no desenvolvimento da solução proposta no presente trabalho;
- **Metodologia:** Capítulo dedicado a descrever os parâmetros necessários para a execução das simulações;
- **Resultados:** Capítulo dedicado a apresentar tanto os resultados das simulações, quanto discutir sobre a utilização das ferramentas para colaborar com o ensino, pesquisa e desenvolvimento no ambiente do Instituto Federal do Piauí Campus Central;
- **Conclusão:** Neste capítulo é feita a conclusão do trabalho dado seus objetivos propostos e são listados os trabalhos futuros para melhorar e/ou expandir a utilização da solução.

2 FUNDAMENTAÇÃO TEÓRICA

Para melhor compreender as decisões de projeto tomadas durante o desenvolvimento da análise e simulação proposta por este trabalho é necessário entender alguns conceitos teóricos relacionados a Cinemática, Dinâmica e Planejamento de Trajetórias para manipuladores robóticos, além de conceituar as ferramentas para simulação utilizadas Ros, Moveit e Gazebo.

2.1 *MANIPULADORES ROBÓTICOS*

Um manipulador robótico é um sistema complexo dependente de diversos componentes que interagem para garantir que o robô consiga executar sua missão ou tarefa. Eles podem ser classificados em: Estrutura Mecânica; Atuadores; Transmissão mecânica; Sensores e Unidade de controle (GÉRADIN; DUYSINX, 2004).

O primeiro deles, a estrutura mecânica ou mecanismo articulado, é feito idealmente de membros rígidos, chamados de elos, que são conectados entre si por juntas mecânicas. A disposição entre elos e juntas cria uma cadeia, motivo para chamar os manipuladores desta classe como manipuladores seriais. No final dessa cadeia pode ser acoplado diferentes efetuadores (do inglês, *end-effector*), que podem ser ferramentas, garras ou outro dispositivo específico para a aplicação do manipulador (SPONG et al., 2006).

Em segundo, os atuadores são os responsáveis por proporcionar a potência mecânica de forma a mover a estrutura mecânica contra a ação de solicitações externas como a gravidade, inércia, cargas, entre outros. Estes atuadores são os responsáveis por modificar a configuração, e portanto, a posição geométrica do efetuador no espaço. Eles podem ser elétricos, hidráulicos ou pneumáticos, e devem ser controlados apropriadamente para cada típo específico (SPONG et al., 2006).

Já a transmissão mecânica é a responsável por conectar e adaptar os atuadores à estrutura mecânica de forma a atender a demanda de torque ou velocidade especificadas. Podem ser de diversas formas como por contato de engrenagens, correias flexíveis, correntes, entre outros (GÉRADIN; DUYSINX, 2004).

Os sensores, por sua vez, proporcionam os sentidos do robô em relação ao ambiente onde está. Eles podem, por exemplo, providenciar informações táteis, de visão e temperatura, que podem ser divididos em dois grupos: Sensores que fornecem informações sobre o próprio robô ou os que informam sobre o ambiente externo (SPONG et al., 2006).

Por fim, tem-se a unidade de controle, que assume algumas funções, como: coletar e processar as informações dadas pelos sensores; Tomar decisões sobre o plano de movimento do manipulador; Organizar o fluxo de informações para se comunicar com as juntas do manipulador

e o ambiente (GÉRADIN; DUYSINX, 2004).

2.1.1 Classificação dos sistemas robóticos

A definição de "robô" não é uma tarefa simples, até mesmo para a literatura formal. Para a *Robot Institute of America* (Instituto de Robótica da América) designa um robô como "um manipulador multifuncional reprogramável, projetado para mover materiais, peças, ferramentas ou dispositivos especializados através de programas para o modificar o desempenho em uma variedade de tarefas"(HOGAN; BUERGER, 2004). Já RICHARDS; SMART(2016) define robô como um sistema construído que exibe tanto capacidades físicas como "mentais"(processamento em softwares) que, contudo, não está vivo no aspecto biológico

Os manipuladores podem ser classificados por diversos critérios, como por seu número de graus de liberdades, elos, configuração cinemática, geometria do espaço de trabalho e características de movimento. O presente trabalho será enfatizado à classificação de um manipulador robótico do tipo serial segundo sua estrutura cinemática, como:

- **Serial**: Estrutura cinemática possui a forma de cadeia cinemática aberta;
- **Paralelo**: Contituíndo de cinemática fechada ;
- **Híbrido**: Constitui cadeia cinemática tanto aberta quanto fechada;

A comparação destes dois tipos é feita basicamente em relação às suas características mecânicas e problemas de controle para determinar que tipo de tarefas cada um pode ser utilizado com maior vantagens (HOGAN; BUERGER, 2004).

Em termos mecânicos os robôs manipuladores do tipo serial são constituídos por atuadores nas suas juntas, implicando massa e momentos de inércias relativamente altos (sistema rígido). De modo diferente, os atuadores dos manipuladores paralelos são montados próximo da base e com isso possibilita redução de massa nos seus elos, implicando em características dinâmicas melhores se comparados com o manipulador serial (HOGAN; BUERGER, 2004).

Em relação ao volume de trabalho, são maiores para manipuladores seriais, se comparado com os do tipo paralelo. Além de que os manipuladores paralelos necessitam maior área para instalação da sua estrutura, o que limita a sua célula de trabalho, em alguns casos essa limitação pode ser vantajosa, por exemplo em tarefas com pequenos volumes de trabalho e áreas disponíveis para instalação, como salas de cirurgia (HOGAN; BUERGER, 2004).

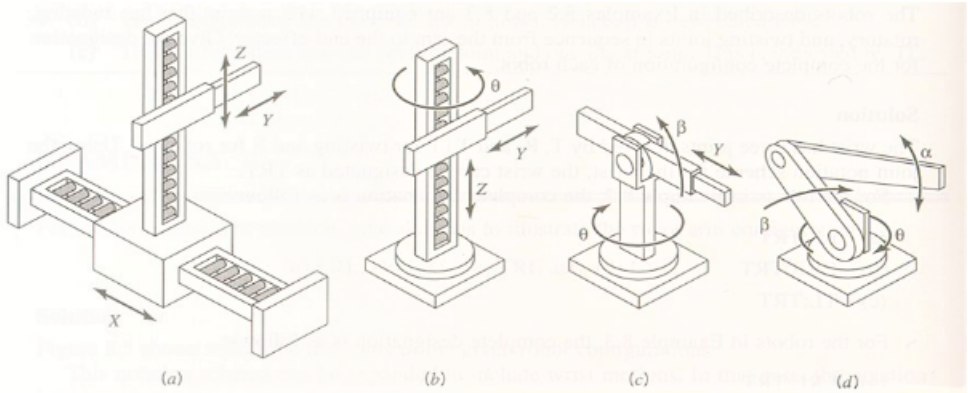
2.1.2 Manipuladores Articulados

Um manipulador também pode ser classificado segundo a sua configuração geral, sendo a mais comum a configuração antropomórfica e, de acordo com CRAIG (2009), um manipulador antropomórfico, ou articulado, geralmente consiste de duas juntas no ombro, sendo uma para elevação para fora do plano horizontal e outro para rotação em torno de um eixo vertical

(geralmente o eixo z), uma junta no cotovelo, que é muitas vezes paralelo à junta de elevação do ombro e duas ou três juntas no punho.

Os robôs articulados têm a vantagem de serem capazes de alcançar espaço confinados, como a soldagem dentro de um automóvel, por exemplo, o que minimiza a intrusão da estrutura do robô no espaço de trabalho. Outra vantagem é que eles geralmente requerem uma estrutura muito menor do que a dos robôs cartesianos, o que os tornam mais baratos e mais comuns em aplicações industriais (CRAIG, 2009). Na Figura 1 é demonstrado as quatro configurações básicas de um manipulador robótico:

Figura 1 – Configurações básicas de um manipulador robótico: (a: cartesiano; b: cilíndrico; c: polar; d: articulado)

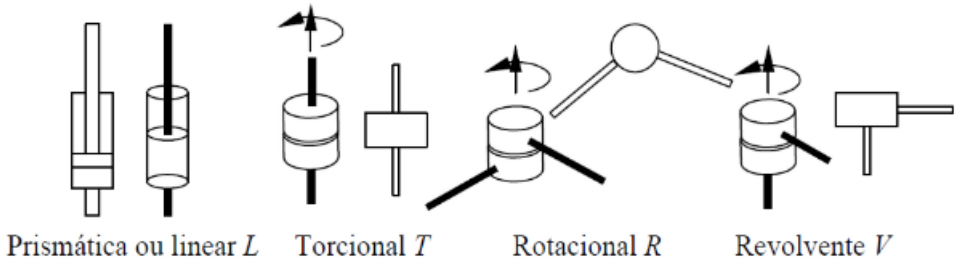


Fonte: GÉRADIN; DUYSINX, (2004)

2.1.2.1 Esquema de notação de juntas

Manipuladores podem utilizar diversos tipos de juntas em sua cadeia, são escolhidas de acordo com as configurações cinemáticas para trabalho, as condições de montagem, entre outros fatores. Sua nomenclatura começa-se pela junta mais próxima a base e prosseguindo para as outras 2 juntas seguintes. Para um manipulador do tipo configuração articulada, a denominação se torna, como exemplo: TRR ou VVR. Onde T é uma junta de torção, R junta de revolução e V é uma junta revolvente, como mostrado na Figura 2.

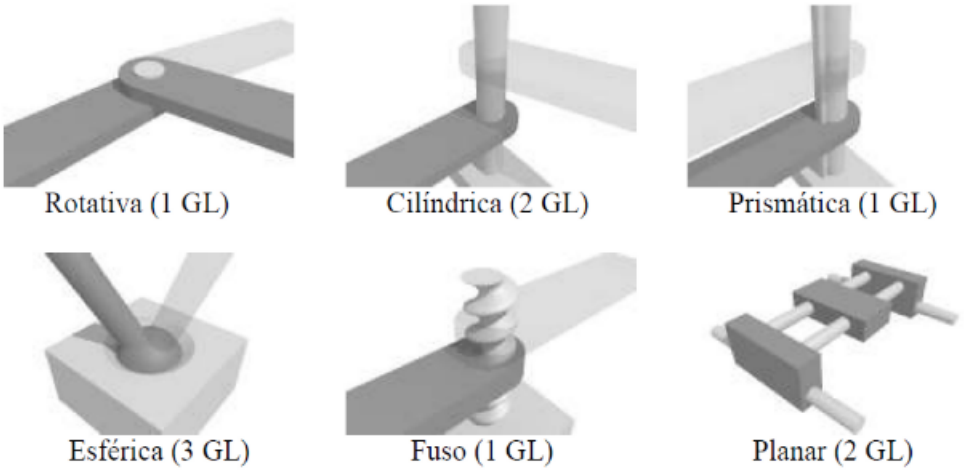
Figura 2 – Representação esquemática das juntas



Fonte: GÉRADIN; DUYSINX, (2004)

A notação do robô pode incluir os movimentos do punho, mediante designação de dois ou três tipos de juntas de punho. A importância de se designar um manipulador em relação ao tipo de suas juntas se deve ao fato que a configuração do manipulador vai determinar o seu volume de trabalho.

Figura 3 – Tipos de juntas empregadas em robôs



Fonte: GÉRADIN; DUYSINX, (2004)

2.1.2.2 Movimentos do robô e Graus de liberdade

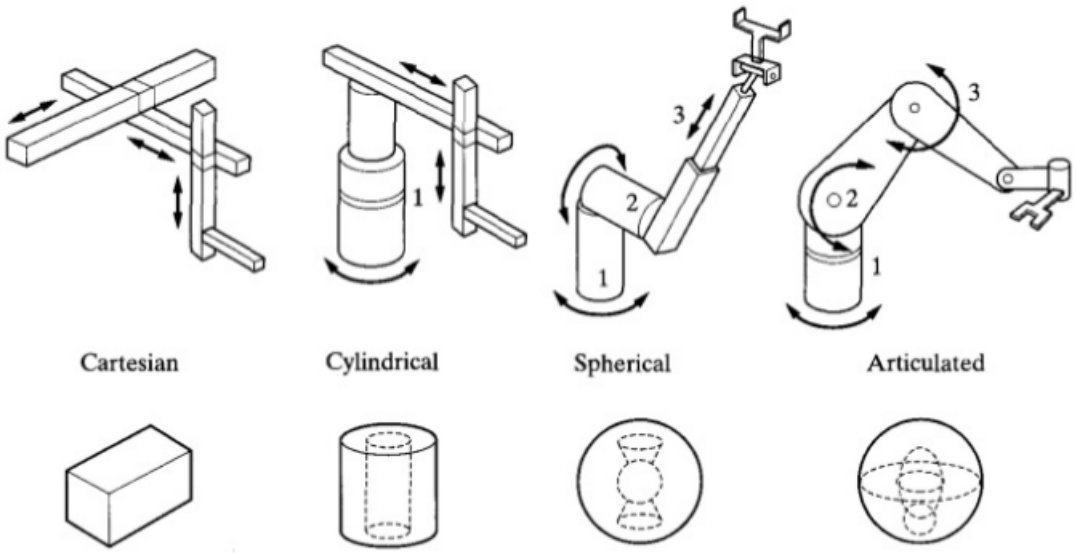
Os movimentos de juntas individuais e do punho são chamados de graus de liberdade, O conceito de graus de liberdade (GDL) ou mobilidade é fundamental para síntese e análise de mecanismos, pode ser definido como: "O número de entradas que precisam ser dadas para criar uma saída previsível" ou "O número de coordenadas independentes necessárias para definir sua posição" (NORTON, 2010). Sendo um robô industrial típico equipado com 4 a 6 graus de liberdade (GROOVER, 1988).

2.1.2.3 Volume de Trabalho

De acordo com Groover (1988), o volume de trabalho se refere ao espaço em que um dado manipulador consegue posicionar seu efetuador. Para um braço Cilíndrico (TLL), por exemplo, seu volume de trabalho seria, teoricamente, um cilindro cujo raio da base corresponde à extensão máxima do braço esticado. A área de trabalho do robô é determinada pelas seguintes características:

- Estrutura cinemática do robô;
- Comprimento dos seus elos;
- Limitações dos movimentos das juntas

Figura 4 – Tipos de volume de trabalho para configurações comuns de robôs



Fonte:(GROOVER, 1988)

2.1.2.4 Funcionamento de um manipulador

O processo de simular ou programar um manipulador mecânico baseado em interface gráfica inicia-se com o usuário especificando uma posição para o efetuador (alvo), esta posição pode ser uma coordenada final ou uma trajetória no espaço; então o usuário define a velocidade do efetuador que poderá ser constante ou em função do tempo, uma outra forma é especificar o tempo desejado para execução da tarefa, neste último caso, o algoritmo converte as coordenadas (coordenadas cartesianas: xyz) definidas no input do sistema em coordenadas das juntas (ângulo θ), esta conversão é feita usando as equações da cinemática e é chamada de cinemática inversa.

Quando o usuário especifica apenas o ponto final (meta) para o efetuador, o computador calcula os ângulos das juntas apenas para este ponto e faz uma interpolação polinomial utilizando o tempo do trajeto para fazer um movimento suave, evitando picos de velocidade e aceleração. Enquanto para o caso de ser definido uma trajetória, então o computador deve calcular os ângulos das juntas para cada ponto, neste caso a velocidade e aceleração das juntas podem apresentar descontinuidades (GÉRADIN; DUYSINX, 2004).

Uma vez obtidos a posição, velocidade e aceleração de cada junta em qualquer tempo t, os torques necessários para efetuar este movimento são computados em cada junta. Este torque é calculado usando as equações de movimento do manipulador. As equações da dinâmica podem apresentar imprecisões no valor do torque devido a forças que são difíceis de quantificar, como atritos presentes na transmissão da junta. Por este motivo utiliza-se sensores que medem a posição (ângulo) da junta a cada período, esta informação é utilizada em um sistema de controle, que calcula a quantidade de torque que deve ser adicionado ao valor inicial para garantir que o

manipulador chegue à sua meta nas condições (velocidade, ou tempo) impostas pelo usuário.

2.2 SIMULADORES ROBÓTICOS

O desenvolvimento e prototipação de sistemas robóticos complexos como manipuladores móveis subaquáticos ou sofisticados robôs de cadeias cinemáticas fechadas não seriam viáveis sem utilizar ferramentas de simulação. Em adição, simular é um excelente método para experimentação enquanto reduz custos com acidentes entre robôs e homens, além de que é possível criar ambientes sem distúrbios, com isso manter o controle das variáveis estudadas e posteriormente aplicar no robô real (TORRES-TORRITI; ARREDONDO; CASTILLO-PIZARRO, 2016).

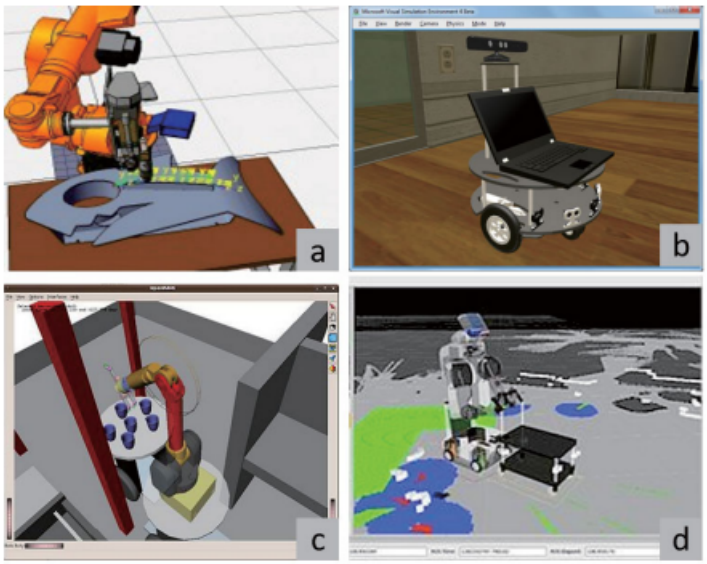
A evolução dos simuladores veio devido ao progresso em tecnologias computacionais durante 1990s que contribuiu significativamente para o desenvolvimento de vários softwares, para modelagem e simulação de manipuladores robóticos, tais como Animate, Robotics Toolbox do Simulink e Matlab, o pacote Robótica para o software Mathematica. Com o aumento do poder computacional durante a virada do século XXI, foi possível que os pesquisadores se concentrassem em problemas de modelagem e simulação de complexidade crescente, tais como manipulação robótica e Grasping (captura), sistemas robóticos multicorpos com interações de contato descontínuas e robótica móvel (TORRES-TORRITI; ARREDONDO; CASTILLO-PIZARRO, 2016).

Uma grande quantidade de simuladores vem sendo desenvolvidos, tais como Open Dynamics Engine (ODE), OpenRave, OpenSim, RoboCode,Simbad, entre outros. E, dentre estes, alguns mais utilizados para robótica industrial de manipuladores desenvolvidos por empresas conhecidas no mercado como da ABB e KUKA. Outro simulador comumente utilizado criado para funcionar em ambiente Windows é o Microsoft Robotic Developer Studio (MRDS). Entre vasta gama de simuladores também encontra-se Gazebo, desenvolvido para executar inicialmente em ambiente Linux e Integrar ao ROS (*robotics Operation System*) (DENG; XIONG; XIA, 2017).

Diante dessa grande variedade de simuladores, tanto gratuitos, quanto por iniciativas privadas, se torna difícil escolher o software que mais atende as necessidades dos desenvolvedores. Dentre os simuladores voltados para robótica de manipuladores industriais podemos destacar vantagens e desvantagens (DENG; XIONG; XIA, 2017).

ABB, KUKA possibilitam escrever programas para robôs de forma conveniente, depuração (do inglês debugged) Off-line e testes com robôs reais, porém em aplicações mais complexas encontra-se dificuldade para integrar os programas com o hardware de terceiros, dificuldade de integrar sensores ou simuladores dinâmicos, como exemplos. Já o MRDS é um ambiente de simulação para Windows, consegue lidar com uma grande variedade de hardware robóticos, inclui pacotes que adicionam outros serviços e com isso consegue executar comportamentos complexos, porém não é atualizado desde 2012 (DENG; XIONG; XIA, 2017).

Figura 5 – Exemplos de plataformas de simulação de manipuladores robóticos existentes. (a) KUKA WorkVisual, (b) Microsoft Robotic Developer Studio (MRDS), (c) Open Robotics Automation Virtual Environment (OpenRAVE), (d) Rviz em Robot Operation System (ROS).



Fonte: (DENG; XIONG; XIA, 2017)

Já Gazebo é um ambiente de simulação com motores físicos(*Engines*) para obter dados do ambiente requerido, permite integrar diversos sensores e modelos dinâmicos, é open-source e continuamente atualizado, tem uma grande biblioteca com modelos de robô que podem ser utilizados, apresenta a possibilidade de expandir suas aplicações utilizando plugins de ruídos, condições de contorno como atrito arrasto, e tem fácil portabilidade para o robô real ao se utilizar juntamente com o ROS.

2.2.1 ROS

Desenvolver softwares para robôs é uma tarefa difícil, existe uma grande variedade de tipos de robôs além de que o escopo da robótica é vasto e se amplia rapidamente. Em adição, a robótica exige diversas habilidades difíceis para as capacidade de um único pesquisador. Diante disso existe a necessidade em desenvolver arquitetura de software robóticos e apresentar integração entre eles, por isso diversos frameworks para gerenciar essas complexidades foram desenvolvidos (QUIGLEY et al., 2009).

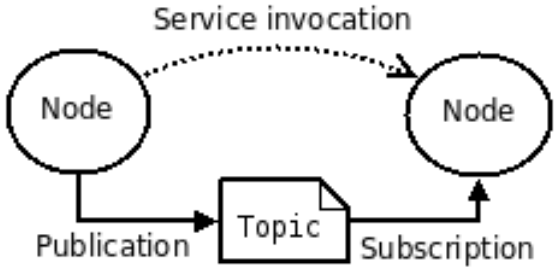
ROS(Robotics Operating System) é um destes frameworks, de código aberto, baseado em ferramentas de multilinguagem disponível. Foi escolhido por apresentar, entre outros do seu tipo, maior quantidade de pacotes de softwares reutilizáveis além de atualmente ter uma comunidade ativas para auxiliar na resolução de erros. De fato, ROS não atende toda a gama de software para robótica, porém apresenta um bom suporte para robótica de manipuladores, que será baseado este trabalho.

A concepção fundamental da implementação do ROS são Nós, Mensagens, Tópicos e Serviços. "Nós"são processos que executam computação, em outras palavras serão os códigos objetivo da aplicação, eles comunicam entre si passando mensagens, que é um tipo de dado estruturado (data structure), por padrão utiliza-se tipos primitivos (inteiros, floating, booleanas , caracteres). "Nós"mandam mensagens publicando em um dado "Tópico", que é um simples caracter como “Velocidade” ou “Posição”, um "Nó"que está interessado em um certo tipo de dado irá assinar no tópico apropriado e acessar a informação.

Poderá existir múltiplos publicadores ou assinantes concorrentes para um único tópico, como pode haver um simples Nó publicando e assinando múltiplos tópicos. De forma geral, publicadores e assinantes não se interferem. Embora esse tipo de comunicação baseado em tópicos é flexível, esta rotina “broadcast” (transmitir informação para muitos receptores) não é apropriada para transações síncronas, o que pode simplificar a criação de alguns Nós, a representação da arquitetura utilizada pelo ROS é exemplificada na figura 6.

No ROS, é chamado essa forma de comunicação síncrona de "Serviços"e "Actions", definido por um nome em caractere e um par estritamente do tipo mensagem: um para requerer e outro para responder. Isso é análogo a serviços web, onde são definidos por URIs (Identificador Universal ou Uniforme de Recurso) e tem documentação de requerimentos e respostas de tipo bem definido. Diferentemente de tópicos, um único Nó consegue anunciar um Serviço de qualquer nome particular, em outras palavra poderá existir apenas um serviço chamado “Classificar-imagem”, por exemplo (QUIGLEY et al., 2009).

Figura 6 – Conceitos básicos sobre ROS. Na imagem está demonstrado a comunicação do tipo assíncrona e síncrona explicado anteriormente



Fonte: (QUIGLEY et al., 2009)

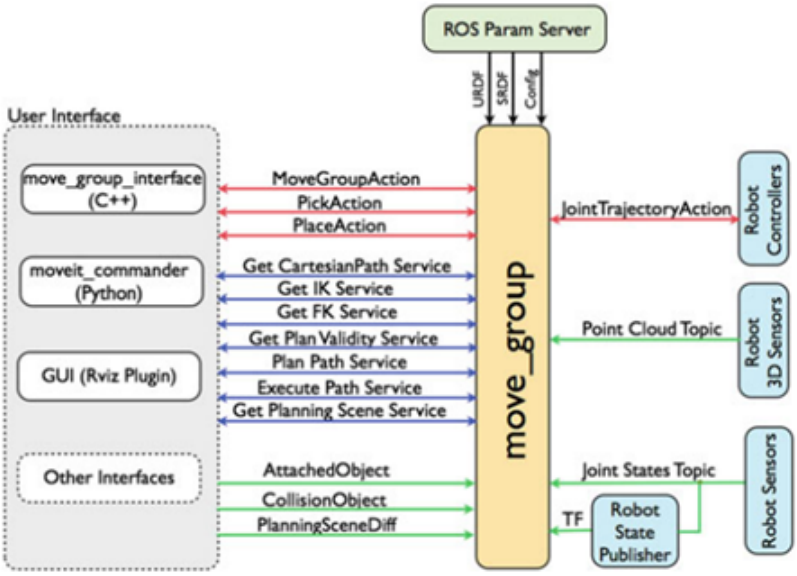
2.2.2 Moveit!

O advento de novos frameworks como Ros faz a robótica mais acessível para novos usuários, com o Moveit não é diferente (KOUBÂA et al., 2017),(MOVEIT!, 2020). Este tem como objetivo providência funcionalidades para manipuladores no Ros, e tem como pilares: **Biblioteca de Recursos** como, por exemplo, planejamento de movimento, controle e manipulação móvel; **Forte Comunidade** que auxilia na manutenção e no seu avanço; **Ferramentas**

que permitem novos usuários integrar o Moveit com seus próprios robôs e usuários avançados implantar suas novas aplicações.

A arquitetura do Moveit é apresentada na Figura 7. O nó principal é chamado de move_group, foi criado para ser leve, gerenciar diferentes capacidades e integrar cinemática, planejamento de movimento e percepção.

Figura 7 – Arquitetura do Moveit_Group

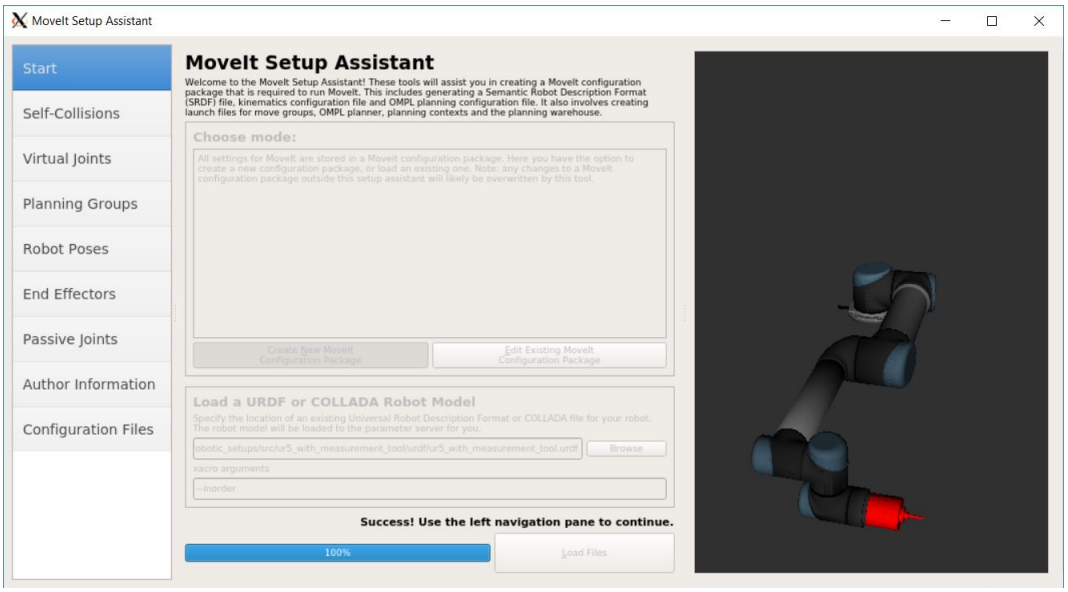


Fonte: (KOUBÂA et al., 2017)

O usuário pode acessar Serviços providenciados pelo move_group de três maneiras: **Com C++**, utilizando o move_group_interface package que permite uma maneira simples de configurar a interface move_group utilizando C++ API, esta maneira é indicada para usuários mais avançados; **Através de GUI**, utiliza o Motion Planning Plugin para RVIZ (visualizador do ROS), é recomendada para interação inicial com o robô por meio do Moveit e para rápidas demonstrações; **Com Python**, usando o moveit_commander package, esta API é recomendada para scripts de demonstração e para construir aplicações (MOVEIT!, 2020).

Também permite utilizar diferentes bibliotecas para planejar movimento, por padrão, utiliza a OMPL (*Open Motion Planning Library*), trata-se de uma biblioteca de planejamento de trajetórias open source. Em adjunto, no moveit é possível definir posições específicas, definir grupos de controle, definir o atuador final do robô, entre outras funcionalidades através do Moveit Setup Assistant como exemplificado na figura 8 (MOVEIT!, 2020).

Figura 8 – Moveit Setup Assistant. Na figura está exmplificado o processo de configuração do Moveit utilizando o robô industrial UR5

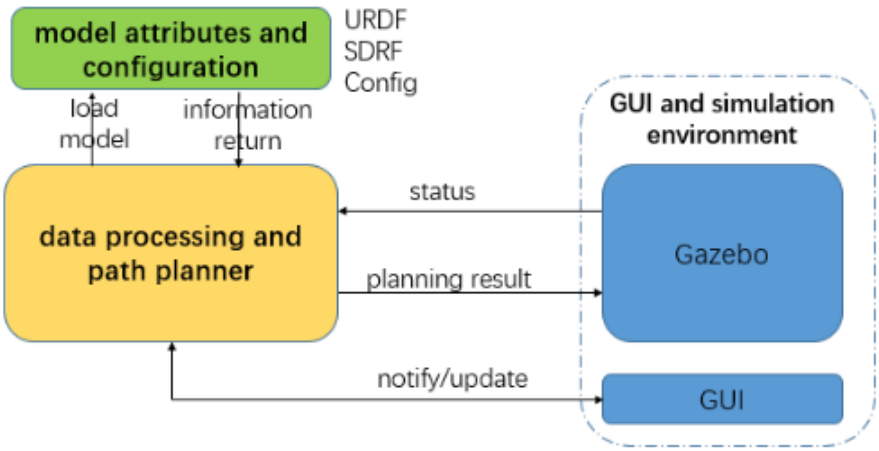


Fonte: (MOVEIT!, 2020)

2.2.3 Gazebo

Uma das grandes vantagens de se utilizar o Gazebo é seu mecanismo de comunicação com o ROS e com o hardware do robô. Ou seja, a simulação pode ser facilmente integrada ao manipulador real. Para facilitar a simulação e entendimento da ferramenta, HUANG; LI; XU(2020) propôs o sistema que depende de três principais partes: 1) Configuração e atributos do modelo; 2) Planejamento de trajetória e processamento de dados, por fim; 3) Ambiente de simulação e interface gráfica para o usuário, como mostrado na Figura 9:

Figura 9 – Arquitetura da Simulação



Fonte: (HUANG; LI; XU, 2020)

a. Configuração e atributos do modelo

Os atributos e configurações são descritas em arquivos do tipo Unified Robot Description Format (URDF) , Semantic Robot Description Format (SRDF) e no formato Config. O primeiro descreve atributos físicos e métodos de montagem das partes do robô. Já o segundo recebe os parâmetros ou comandos, carrega o arquivo do modelo e obtém o resultado do planejamento da trajetória. E a última parte é responsável pelas entradas e saídas do sistema, sem ela a visualização dos dados como aceleração e velocidade se tornam relativamente abstratas e desorganizada.

O arquivo com o modelo computacional do robô consiste em três partes: elos(links), juntas (joint) e mecanismo de transmissão (transmission mechanism). O atributo Link descreve as características do elo em três aspectos: 1) Visual, referente ao aspectos de posição geométrica, as características da superfície, materiais entre outras entidades; 2)Collision, descreve os aspectos relacionados aos limites de colisão do link; 3)Inertial, descreve o sistema de coordenadas de referência inercial, massa, e o próprio momento de inércia (MOI), a Figura 10 exemplifica o *link*1 do robô ABB IRB 1200 5/90 em que a estrutura do arquivo foi modificada afim de facilitar a compreensão. As Joints são as seções que descrevem a amplitude de rotação de cada junta e o limite da velocidade de rotação como exemplificado na Figura 11. A seção relacionada ao Transmission Mechanism é a seção onde os valores da razão de transmissão ou a relação de redução são informadas.

Figura 10 – Seção do arquivo URDF que descreve o *Link* (Elo)

```
<link name="link_1">
    <inertial>
      <mass value="11.8419"/>
      <origin xyz="0.000877 -0.000631 -0.062883"/>
      <inertia ixx="0.11194" ixy="-4.54988e-05" ixz="0.000280961"
               iyy="0.0915159" iyz="-0.000109905" izz="0.0876456"/>
    </inertial>
    <collision name="collision">
      <geometry>
        <mesh filename="~link_1.stl"/>
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <mesh filename="~link_1.dae"/>
      </geometry>
    </visual>
</link>
```

Fonte: Elaborado pelo Autor

Em "mesh filename”, deve conter o caminho para chegar aos arquivos que descrevem a parte visual e a parte de contato do objeto. Estes arquivos podem ser modelados em softwares de terceiros como o SolidWorks e Blender e então exportados no formato URDF.

Figura 11 – Seção do arquivo URDF que descreve a *Joint*(junta)

```
<joint type="revolute" name="joint_1">
    <origin xyz="0 0 0.3991" rpy="0 0 0"/>
    <axis xyz="0 0 1"/>
    <parent link="link_1"/>
    <child link="link_2"/>
    <limit effort="1000" lower="-2.967" upper="2.967" velocity="5.027"/>
    <dynamics damping="50.0" friction="1.0"/>
</joint>
```

Fonte: Elaborado pelo Autor

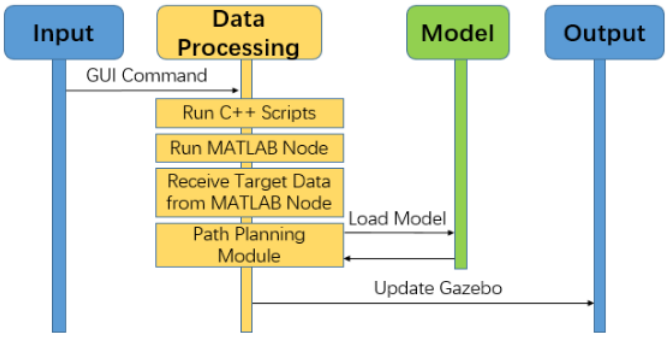
Na Figura 11 mostra o exemplo de que se deve referênciar qual o *link* de origem ou aquele que vem antes na cadeia de elos como *parent* e o elo sucessor deve ser especificado em *child*. Deve-se definir qual o tipo de junta, no exemplo a junta é do tipo rotacional(R).

b. Planejamento de trajetória e processamento de dados

A seção de Planejamento de trajetória e processamento de dados é responsável por aplicar os modelos matemáticos para transformar as entradas (*inputs*) do usuário em um determinado objetivo ou saída(*output*). Por exemplo, o usuário define um ponto de partida para o atuador e um ponto objetivo, além de especificar o tempo de trajetória entre outras variáveis do processo; então é feito em um script as rotinas para executar a trajetória desejada, essa rotina pode ser feita em diversas linguagens, as mais comuns utilizadas no ROS são Python ou C++, esses *scripts* publicam ou subscrevem em nós específicos para executar o movimento.

Outra forma de executar esse processamento é utilizando softwares como o Matlab, em que o script é escrito em sua linguagem própria da ferramenta e então utilizado os packagens que criam Nós para conectar com o ROS. Exisite também a possibilidade de utilizar o framework Moveit em que se torna mais simples definir o início e objetivo da trajetória, pois existe uma interface gráfica amigável para o usuário, o Moveit utiliza o Move-group para fazer essa conexão e processamento das informações, como explicado anteriormente.

Figura 12 – Processamento de dados para a simulação robótica



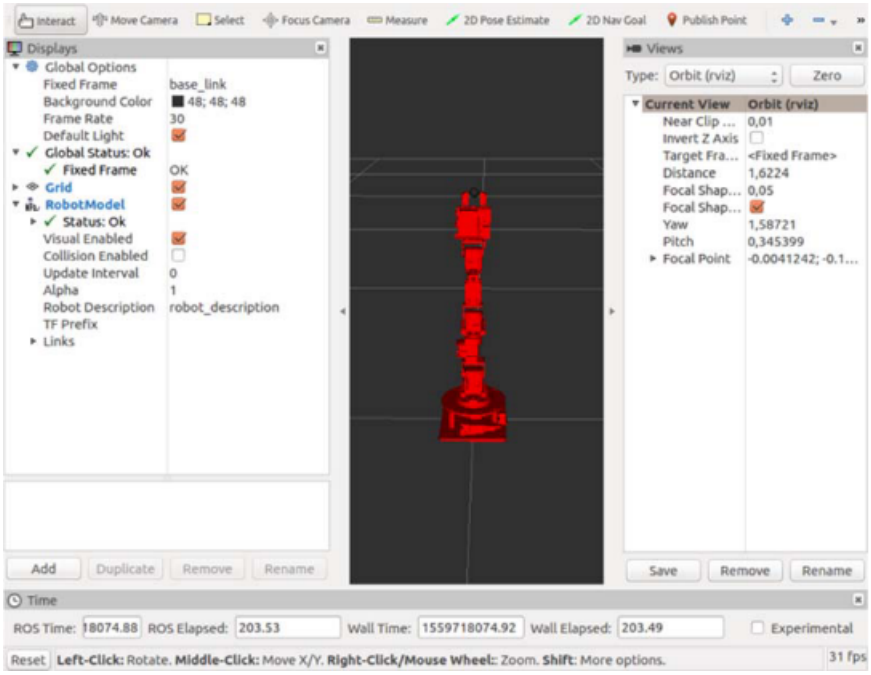
Fonte: (HUANG; LI; XU, 2020)

Na Figura 12 é descrito o processo de receber os dados provenientes de *inputs* do usuário, passando pelos scripts e os módulos de planejamento de trajetória, esses módulos carregam dados do modelo como dimensões, condições físicas do modelo, entre outros. Por fim é atualizado na ferramenta de simulação, nesse caso o Gazebo.

c. Ambiente de simulação e interface gráfica

A interpretação, assim como a configuração da simulação robótica pode ser complicada para o usuário, dessa forma uma interface gráfica facilita esse processo. No ROS, existem diversas aplicações para visualização de informações e modelos, uma delas é o Rviz que, pode-se avaliar implementações como sensores, trajetórias, ambientes, sistemas de coordenadas, entre outras funcionalidades como exemplificado na Figura 13. A outra ferramenta rqt permite visualizar os dados em forma gráfica, o que facilita a interpretação desses dados, pode-se avaliar também os Nós, Tópicos, Serviços de forma esquemática assim como os *publishers* e *subscribers*. Além dessas ferramenta, o Gazebo permite visualizar o modelo em simulação em uma interface própria mostrada na Figura

Figura 13 – Interface do Rviz - Visualização de manipulador de 7 GDL



Fonte: (XU; DUGULEANA, 2019)

2.3 CINEMÁTICA DIRETA

A Cinemática é o ramo da mecânica que descreve o movimento dos corpos sem relacioná-los com as forças que o causaram. Em outras palavras, é o estudo da geometria do movimento. No caso da cinemática de manipuladores, é a ciência que se refere às propriedades geométricas do movimento variáveis no tempo (CRAIG, 2009).

Segundo (FERREIRA, 1999) esta etapa da modelagem trata das relações geométricas que envolvem as coordenadas na região de trabalho x, y e as variáveis das juntas (θ). Podendo ser dividida em cinemática direta, que relaciona as configurações das juntas em relação a posição de um ponto do mecanismo no espaço de trabalho e a cinemática inversa, que computa o oposto da anterior, ou seja, tem como objetivo avaliar quais as configurações de juntas são possíveis partindo de um ponto no espaço de trabalho(x, y $\rightarrow$ theta). Para a cinemática direta só existe uma solução, enquanto na cinemática inversa podem existir várias soluções, se estas estiverem dentro da região de trabalho do braço.

Existem diversos métodos para descrever a cinemática direta, para cadeia aberta. Uma delas é a convenção de Denavit-Hartenberg (convenção D-H). Outra forma de representar é utilizando a fórmula do Produto das Exponenciais (PoE, do inglês Product of Exponentials).

A vantagem da primeira em relação a segunda é que requer um número menor de parâmetros para descrever a cinemática direta de um manipulador: Para descrever um robô de n juntas será necessário 3n valores que descrevem a estrutura e n números que descrevem os valores das juntas. Em contrapartida, a forma PoE requer 6n números relacionados a velocidade em adição a n número de juntas, porém este método fornece algumas vantagens em comparação ao anterior, e será o escolhido para o trabalho (FERNANDEZ; LYNCH; WENG, 2019).

As desvantagens da convenção de D-H é que é necessário atribuir coordenadas dos links de forma que seja válido para os parâmetros existentes; caso mal atribuído, a acurácia da medição e identificação torna-se difícil. Enquanto ao método PoE uma vez atribuído o sistema de coordenadas da base e o sistema de coordenadas do atuador final por meio da matriz M (relaciona as posições de cada elo para a configuração zero de cada junta) estará completamente definido, nenhuma coordenada de um elo de referência se torna necessário. Outra vantagem é a interpretação dos parâmetros do PoE, como o vetor velocidade das juntas (Screw) é natural e intuitivo (FERNANDEZ; LYNCH; WENG, 2019). Além de que se torna mais intuitivo observar o sistema de coordenadas de manipuladores no ROS e Gazebo, pois esses são atribuídos de forma diferente do método de D-H.

A utilização do PoE apresenta uma visão muito mais geométrica do manipulador. Em adição, o PoE realiza o tratamento de juntas prismáticas e de revolução da mesma maneira, o que representa uma interessante vantagem na implementação, uma vez que o método de D-H precisa saber anteriormente qual tipo de junta, pois oferece tratamentos diferentes para uma ou outra. Vale destacar, também, que existe vantagens na implementação computacional da utilização do POE em comparação com o método DH, pois tem um menor custo de processamento. Outra vantagem está na utilização do POE aplicado a métodos de calibração de robôs (FERNANDES, 2013).

Calibrações de cinemática são realizadas por conta de erros de fabricação, devido a existência de tolerâncias, ou de erros nos parâmetros com relação aos valores nominais. Desta forma a calibração do manipulador robótico faz-se necessária para garantir a precisão de

posicionamento (FERNANDES, 2013).

Isto posto, para a modelagem de manipuladores, deve-se começar com a descrição da cinemática direta. Antes de descrever as equações que regem a Cinemática direta é necessário entender os conceitos de Movimento de corpos rígidos, entender quais são as formas de representação exponencial de rotação e de movimento de corpos rígidos e então transportar estas definições para cadeias cinemáticas.

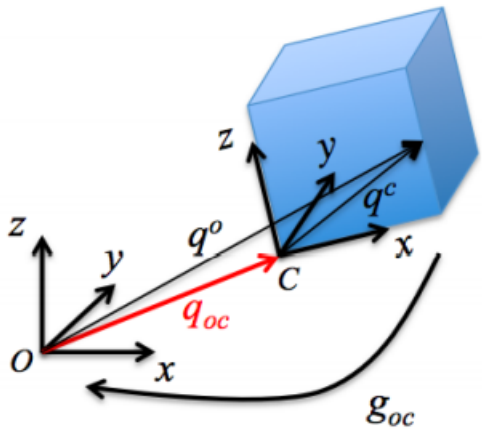
2.3.1 Movimento de corpos rígidos

Entende-se que um corpo rígido é representado por um sistema de partículas que, por ser rígido, a distância relativa entre elas permanece inalteradas. Em adição a essa condição, uma segunda necessária é que o produto vetorial entre os vetores de posição dos pontos deve ser preservado. Com isso, pode-se definir a posição e orientação em relação a um referencial fixo. A posição do sistema Cartesiano "anexado"ao corpo com relação ao sistema Cartesiano fixo no referencial inercial fornece a informação de posição. A rotação apresentada pelo sistema móvel com relação ao sistema fixo fornece a informação de orientação do corpo rígido.

a. **Representação exponencial de rotação e do movimento de corpos rígidos**

A orientação de um sistema de coordenadas qualquer C anexada ao corpo com relação ao sitema fixo O pode ser descrito por um movimento de rotação em torno de um único eixo, como demonstrado na figura 14. Considerando a velocidade escalar unitária $\dot{q}(t)$ de um ponto anexado a um corpo rígido que se movimenta com velocidade de rotação vetorial $\vec{\omega}$ pode-se escrever a equação do movimento de acordo com a Equação 2.1.

Figura 14 – Representação de corpo rígido em relação a referencial fixo



Fonte: (FERNANDEZ; LYNCH; WENG, 2019)

$$\dot{q}(t) = \omega \times q(t) = \hat{\omega} \cdot q(t) \tag{2.1}$$

Na equação 2.1, $\hat{\omega}$ representa o produto vetorial em forma matricial, este é mostrado na equação 2.2

$$\begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{2.2}$$

A solução da equação 2.1 é dada através da equação 2.3. Em que $q(0)$ representa a posição inicial do corpo. Em FERNANDEZ; LYNCH; WENG denota esta posição inicial como M, que também será utilizado neste trabalho.

$$q(t) = e^{\hat{\omega}t} \cdot q(0) = e^{\hat{\omega}t} \cdot M \tag{2.3}$$

Na equação 2.3 t pode ser substituido por θ que representa o valor do grau de rotação em um determinado instante. A solução do exponencial matricial $e^{\hat{\omega}\theta}$ é dada pela Fórmula de Rodrigues para a condição, simplificada de $\vec{\omega}$ quando seu módulo é unitário, ou seja $\|\omega\| = 1$. A solução é dada pela equação 2.4

$$e^{\hat{\omega}\theta} = I + \hat{\omega} \cdot sen(\theta) + \hat{\omega}^2 \cdot (1 - cos(\theta)) \tag{2.4}$$

Em que I representa a matriz identidade.

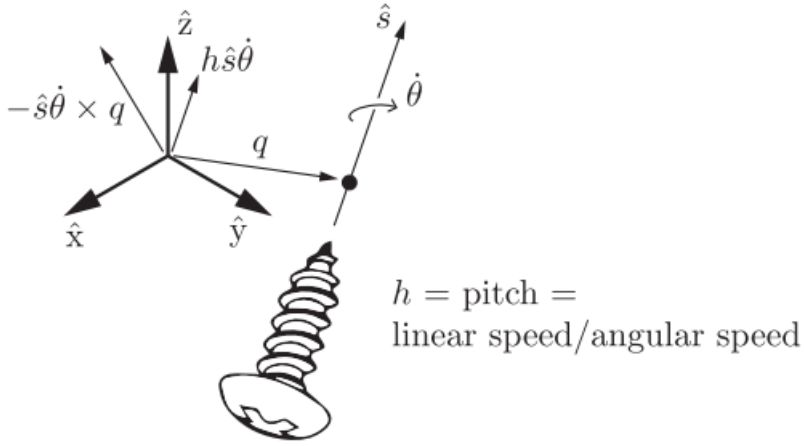
Já a representação de um movimento de corpo rígido é dada pela equação matricial 2.5. A transformação T_{oc} descreve o ponto representado por $q^c \in R^3$ no sistema C sendo transformado para o sistema O, representado pelo ponto $q^o \in R^3$. O vetor p_{oc} é a translação entre os sistemas O e C. A matriz R_{oc} é a rotação entre os mesmos sistemas de coordenadas.

$$q^o = \begin{bmatrix} q^o \\ 1 \end{bmatrix} = \begin{bmatrix} R_{oc} & p_{oc} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} q^c \\ 1 \end{bmatrix} = T_{oc} \cdot q^c \tag{2.5}$$

2.3.1.1 Screw Axis ou Helicogiro

Durante a análise cinemática as características de movimentos são necessárias, para isso é necessário avaliar as condições de velocidade de um determinado eixo, neste contexto se define o Screw Axis, ou Helicogiro, que representa o movimento familiar de um parafuso, que rotaciona em torno de seu eixo enquanto também translada ao longo deste mesmo eixo como demonstrado na Figura 15:

Figura 15 – Screw Axis S representado por um ponto q, a direção unitária $\hat{s}$, e um movimento de *pitch* h



.

Fonte: (FERNANDEZ; LYNCH; WENG, 2019)

Um representação do Screw Axis S é o conjunto de $q, \hat{s}, h$ onde $q \in R^3$ é um ponto no eixo, $\hat{s}$ é o vetor direcional unitário do eixo e h é chamado de *screw pitch* que define a razão da velocidade linear ao longo do screw axis em relação a velocidade angular $\dot{\theta}$ entorno do screw axis, como mostrado na Figura 15.

Utilizando o esquema da Figura 15 e os recursos da geometria, pode-se definir o vetor velocidade $V = (\omega, v)$ correspondente a velocidade angular $\dot{\theta}$ entorno de S como:

$$V = \begin{bmatrix} \omega \\ v \end{bmatrix} = \begin{bmatrix} \hat{s}\dot{\theta} \\ h\hat{s}\dot{\theta} - \hat{s}\dot{\theta} \times q \end{bmatrix} \tag{2.6}$$

Em que a velocidade linear v é a soma de dois termos: Um para representar a translação ao longo do Screw Axis, $h\hat{s}\dot{\theta}$, e a outra para representar o movimento linear da origem induzida pela a rotação entorno do eixo, $-\hat{s}\dot{\theta} \times q$. O primeiro termo está na direção de $\hat{s}$ enquanto o segundo termo está em um plano ortogonal a $\hat{s}$.

Caso $\omega = 0$, então o *pitch* h do Helicoide é infinito. Neste caso $\hat{s}$ é escolhido como $\frac{v}{\|v\|}$, e $\dot{\theta}$ é interpretado como a velocidade linear $\|v\|$ ao longo de $\hat{s}$. Caso $\omega \neq 0$ então existe um *screw axis* equivalente $q, \hat{s}, h$ e velocidade angular $\dot{\theta}$, onde $\hat{s} = \frac{\omega}{\|\omega\|}$, $\dot{\theta} = \|\omega\|$, $h = \bar{\omega}^T \frac{v}{\dot{\theta}}$, em que $\bar{\omega}$ é o vetor unitário da velocidade angular e q é escolhido de forma que o termo $-\hat{s}\dot{\theta} \times q$ representa a porção ortogonal v em relação ao *screw axis*.

Por fim, deseja-se definir o *screw axis* como a porção normalizada do vetor de velocidade V, tem-se:

- (a) Se $\omega \neq 0$ então $S = \frac{V}{\|\omega\|} = (\frac{\omega}{\|\omega\|}, \frac{v}{\|\omega\|})$. O *screw axis* S se torna simplesmente V

normalizado pelo o módulo da velocidade angular. A velocidade angular entorno do *screw axis* é dado por $\dot{\theta} = \|\omega\|$, e então $V = S\dot{\theta}$.

- (b) Se $\omega = 0$ então $S = \frac{V}{\|v\|} = (0, \frac{v}{\|\omega\|})$. O *screw axis S* se torna simplesmente *V* normalizado pelo o módulo da velocidade linear. A velocidade linear ao longo do *screw axis* é dado por $\dot{\theta} = \|v\|$, e então $V = S\dot{\theta}$.

Dessa forma para um dado sistema de coordenada, o *screw axis S* é definido como:

$$S = \begin{bmatrix} \omega \\ v \end{bmatrix} \in R^6 \tag{2.7}$$

Onde (i) $\|\omega\| = 1$ ou (ii) $\|\omega\| = 0$ e $\|v\| = 1$. Se (i) então $v = -\omega \times q + h\omega$, onde q é um ponto no eixo screw e h é o movimento *pitch* deste eixo (h = 0 quando apenas existir rotação entorno do *screw axis*). Se (ii) então o movimento *pitch* do *screw axis* é infinito e a velocidade de translação ao longo do eixo é definido por v.

2.4 CINEMÁTICA INVERSA

A cinemática inversa tem como objetivo definir um ponto ou trajetórias no plano cartesiano e então transformar essas coordenadas em ângulos ou deslocamentos para os atuadores, este tipo de procedimento é frequente na programação de manipuladores pois o usuário apenas precisa focar nos valores de x, y e z que o efetuador deve estar posicionado. Em outras palavras, dado uma cadeia aberta no espaço com cinemática direta definida por $T(\theta)$, em que $\theta \in R^n$, o problema da cinemática inversa consiste em encontrar, para uma desejada configuração do end-effector, X $\in$ SE(3), uma solução θ que satisfaça $X = T(\theta)$. (FERNANDEZ; LYNCH; WENG, 2019); (CRAIG, 2009).

Diferentemente da cinemática direta, a cinemática inversa pode possuir múltiplas soluções, ou mesmo nenhuma solução nos casos em que X encontra-se fora do espaço de trabalho. Para uma cadeia aberta espacial com j juntas e X pertencente ao espaço de trabalho, $j = 6$ tipicamente lida com um número de soluções para cinemática inversa finito, enquanto para $j > 6$ lida com número infinito de soluções (FERNANDEZ; LYNCH; WENG, 2019).

Assim sendo, a solução da cinemática inversa é considerada mais complexa que a da cinemática direta (BUSS, 2004). Existe uma extensa quantidade de métodos para se resolver cinemática inversa, estes são divididos em dois grandes grupos: soluções analíticas e numéricas, sendo o primeiro utilizado para manipuladores com seis juntas ou menos em que o punho é esférico. Já os métodos numéricos são divididos pela forma como são abordados, sendo os principais: uso da matriz Jacobiana, uso de um sistema de controle, e uso de métodos de otimização que minimizam uma função erro (objetiva). Em adjunto, métodos numéricos podem

ser aplicados se as equações da cinemática inversa não admitirem soluções analíticas. Também são utilizados para melhorar a acuracidade dessas soluções

De acordo com (FEDOR, 2003), a maioria dos sistemas usam o método da peseudoinversa Jacobiana ou sua transposta para solucionar o problema, contudo este método sofre de singularidades. Muitos métodos têm sido desenvolvidos para calcular or aproximar a inversa do Jacobiano, tais como: Transposta Jacobiano, Mínimos quadrados amortecido (DLS), mínimos quadrados amortecido com decomposição de valor único (SVD-DLS) e mínimos quadrados amortecido seletivo (SDLS) e outras extensões (BUSS, 2004).

Outros tipo de solução é baseado no método de Newton-Raphson, este é utilizado por (FERNANDEZ; LYNCH; WENG, 2019), o qual apresenta uma vasta biblioteca de funções para robótica, tanto para Matlab quanto para a linguagem Python. Estes algoritmos procuram uma posição e orientação do efetuador que irá minimizar a função objetiva, retornando, portanto, um movimento suave sem descontinuidades. Os mais conhecidos são o método de Powell, o método de Broyden e o método de Broyden, Fletcher, Goldfarb e Shanno (BFGS) utilizados extensivamente em otimização. O algoritmo se encontra no anexo A e foi comentado para facilitar o entendimento (BUSS, 2004).

Estes métodos são extensivamente empregados em animações gráficas e em manipuladores robóticos com elevado número de graus de liberdade. Podem ser usados como soluções em tempo real onde a posição da junta do manipulador é atualizada a cada iteração do algoritmo ou pode-se computar todas as iterações previamente e obter os valores de ângulos de cada junta para só então depois atualizar a posição do manipulador.

2.5 DINÂMICA

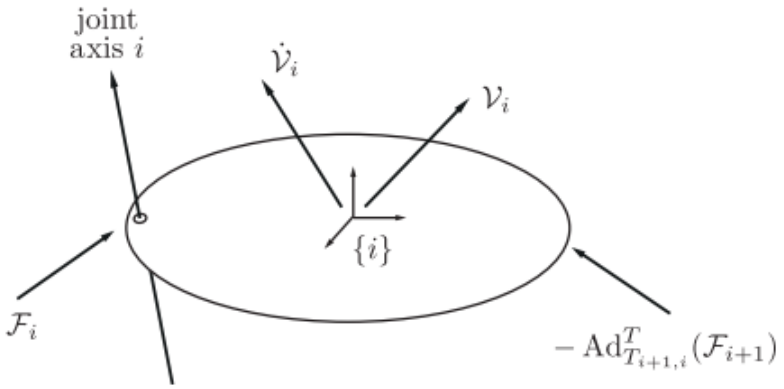
A modelagem dinâmica de um manipulador robótico tem como desafio a análise de todas as forças internas envolvidas no processo de movimentação dele (CRAIG, 2009). Essa modelagem segue dois tipos de abordagem, a dinâmica direta e inversa. A primeira tem como princípio encontrar o vetor dos torques nas juntas τ dado os pontos da trajetória, os deslocamentos, velocidade e acelerações das juntas. Esse tipo de abordagem é relevante para problemas relacionados a controle do robô.

O segundo tipo de desafio está relacionado à como o mecanismo moverá a partir da aplicação de um conjunto de torque. Ou seja, aplicado τ deseja-se saber qual os valores de θ, $\dot{\theta}$ e $\ddot{\theta}$. Já esta abordagem é frequentemente utilizada para simulação de manipuladores robóticos (CRAIG, 2009).

Existem diversos métodos para descrever a dinâmica de mecanismos, para a dinâmica de robôs manipuladores os mais utilizados são o método de Newton-Euler e o método Lagrangiano (FERNANDEZ; LYNCH; WENG, 2019), (CRAIG, 2009). A formulação por Newton-Euler se resume em definir as equações de movimento de cada corpo levando em consideração o

balanceamento das forças de cada um dos elos, de forma que isto é feito por meio do diagrama de corpo livre como demonstrado na figura 16. Várias implicações são observadas neste método, e a mais relevante é que para um manipulador de vários GDL a discrição do diagrama de corpo livre pode se tornar uma tarefa demasiadamente complexa.

Figura 16 – Diagrama de corpo livre das forças atuando em um elo qualquer *i* (*end-effector*)



.

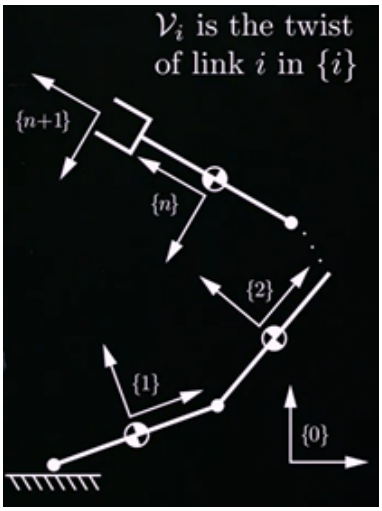
Fonte: (FERNANDEZ; LYNCH; WENG, 2019)

Já a formulação Lagrangiana usa a conservação de energia do sistema (princípio da ação mínima de Hamilton). Tem como maior desvantagem o custo computacional se comparado com o método anterior. Para uma típica situação em que o número de elos (n) é igual a seis, o método iterativo de Newton-Euler chega a ser 100 vezes mais eficiente em relação a custo computacional (CRAIG, 2009). Diante disso, os dois métodos são relevantes para a robótica de manipuladores, mas devido a que as ferramentas de código aberto utilizadas neste trabalho priorizam o método Newton-euler, este será fundamentado com maior ênfase.

2.5.1 Formulação do método Newton-Euler

O sistema de referência de cada corpo será definido como i , definido no centro de massa de cada elo, com i =1 ... n números de elos. O sistema de coordenada da base é denotado como 0, e o sistema de coordenada referente ao atuador final ou *end-effector* denotado como n+1, como demostrado na figura a seguir.

Figura 17 – Distribuição do sistema de coordenada referente ao atuador final (*end-effector*)



Fonte: (FERNANDEZ; LYNCH; WENG, 2019)

A matriz M de transformação refere-se às dimensões de cada um dos elos em relação a um sistema de coordenada específica. Até o momento ela vem sendo referenciada em relação ao sistema de coordenada da base 0, para a modelagem dinâmica será feito uma adaptação para referenciar cada elo separadamente em cadeia e então analisar cada um deles isoladamente mais com interdependência dos outros. Em outras palavras será definido $M_{i,j} \in SE(3)$ como sendo a configuração para o elo na posição zero (home position) com o sistema de coordenada j em relação a i. E como abreviação $M_i = M_{0,i}$ para representar a configuração do sistema de coordenada i em relação a base 0. Dessa forma $M_{i-1,i}$ e $M_{i,i-1}$ podem ser calculados como:

$$M_{i-1,i} = M_{i-1}^{-1} M_i e M_{i,i-1} = M_i^{-1} M_{i-1} \tag{2.8}$$

O Eixo de velocidade ou helicogiro (Screw Axis) para a junta i, expresso em relação ao sistema de coordenada do elo i é dado por A_i. O mesmo Screw Axis é expresso em relação ao sistema de coordenada da base como S_i e podem ser relacionados de acordo com a equação, em que Ad é conhecido como lea bracket:

$$A_i = Ad_{M_i^{-1}(S_i)} \tag{2.9}$$

a. Matriz transformada

Define-se a matriz transformada que referência a cinemática do manipulador como $T_{i,j} \in SE(3)$ para ser a configuração do sistema de coordenada j em relação a i para um arbitrário valor de θ, com isso se tem $T_{i-1,i}(\theta_i)$. E que $T_{i,i-1}(\theta_i) = T_{i-1,i}^{-1}(\theta_i)$ são calculadas como:

$$T_{i-1,i}(\theta_i) = M_{i-1,i} e^{[A_i]\theta_i} e T_{i,i-1}(\theta_i) = e^{[A_i]\theta_i} M_{i,i-1} \tag{2.10}$$

Algumas demais notações devem ser especificadas antes de definir as equações que regem a dinâmica do manipulador. $V_i = (w_i, v_i)$ representa a vetor que contém as velocidades angulares(w_i) e velocidades lineares(v_i) expressas no sistema de coordenada i de cada elo i. $F_i = (m_i, f_i)$ (Chamado de Wrench por Modern Robotics) representa o vetor relacionado às solicitações em cada link, em que m_i representa um momento aplicado no elo i e f_i representa uma força qualquer atuando no mesmo elo.

$G_i \; \varepsilon \; R^{6\times6}$ denota a matriz espacial de inércia do elo i expressa em relação ao sistema de coordenada do elo i, em que m_i representa a massa do elo i, II_i denota a matriz 3×3 rotacional de inércia do elo i, esta será encontrada no arquivo URDF do robô disponível pela comunidade ROS e do Fabricante, cabe lembrar que o momento de inércia será em relação ao centro de massa do elo, dito isso, G_i é dado pela equação a seguir:

$$G = \begin{bmatrix} II_i & 0 \\ 0 & m_i I \end{bmatrix} \tag{2.11}$$

b. Velocidade e aceleração

Com essas definições pode-se calcular de forma recursiva o vetor V_i (twist) e a aceleração de cada elo. O vetor twist do elo i será a soma do vetor twist do elo anterior, i-1 expresso no sistema de coordenada i, mais o acréscimo dado pelo a taxa de variação da junta i como demonstrado na equação 2.12.

$$V_i = A_i\dot{\theta}_i + [Ad_{T_{i,i-1}}]V_{i-1} \tag{2.12}$$

A aceleração $\dot{V}_i$ também pode ser definida de forma recursiva derivando a equação 2.12. A aceleração do elo i é dado pela soma de três componentes: o componente relacionado a aceleração angular da junta $\ddot{\theta}_i$, a componente relacionada a aceleração proveniente do elo anterior i-1 expressa no sistema de coordenada i e o produto das velocidades, como demonstrado na equação a seguir:

$$\dot{V}_i = A_i\ddot{\theta}_i + [Ad_{T_{i,i-1}}]\dot{V}_{i-1} + [ad_{V_i}]A_i\dot{\theta}_i \tag{2.13}$$

As equações anteriores são definidas em relação ao conjunto de valores de θ, $\dot{\theta}$ e $\ddot{\theta}$ e então encontra-se as velocidades e acelerações no espaço de realização da tarefa, ou seja, no espaço cartesiano. Deseja-se o processo inverso, em que a saída deve ser os valores de posição, velocidade e aceleração das juntas, para isso tem-se a relação:

$$\dot{\theta} = J^{-1}V\ddot{\theta} = J^{-1}\dot{V} - J^{-1}\dot{J}J^{-1}V \tag{2.14}$$

Em que $J(\theta)$ e V são expressas no mesmo sistema de coordenada, e não será necessária a pseudojacobiana pois o robô utilizado tem seis graus de liberdade, logo J é invertível.

c. Forças e Torques

Dados a velocidade e aceleração calculadas partindo da base para a ponta(tip) pode-se calcular os torques e forças atuando em cada uma das juntas fazendo o processo contrário, ou seja, calculando da ponta para a base. O vetor Wrench total atuando em cada elo i é dado pela soma do vetor F_i transmitido através da junta i e o vetor F_{i+1} aplicado pelo elo através da junta i+1 expresso no sistema de coordena i, com isso se tem a igualdade a seguir:

$$G_i\dot{V}_i - Ad^T_{V_i}(G_iV_i) = F_i - Ad^T_{T_{i+1,i}}(F_{i+1}) \tag{2.15}$$

Resolvendo da ponta a base do manipulador para cada elo i tem-se como variável desconhecida apenas F_i. Como a junta i tem apenas um grau de liberdade, cinco das seis dimensões de F_i são consideradas "livres", devido a estrutura da junta. Dessa forma o atuador deve apenas providenciar uma força ou torque escalar na direção do eixo de rotação (joint's screw axis) dado pela equação a seguir, em que τ representa o torque de cada junta para responder a solicitação F_i:

$$\tau_i = F_i^T A_i \tag{2.16}$$

A figura a seguir demonstra o diagrama de corpo livre de cada elo com as solicitações proveniente das juntas i e i+1, assim como o sistema de coordenada especificado no centro de massa dele.

Definido cada um dos itens das subseções anteriores, especificando $\dot{V}_0$ como o vetor aceleração no espaço cartesiano da base e como $\dot{V}_0 = (0, -g)$, ou seja considerando uma base fixa tendo como aceleração a gravidade. Define-se também, $F_{n+1} = (M_tip, F_tip)$ como a solicitação atuando no end-effector, em que M_tip representa um momento qualquer aplicado nele. Como isso pode-se definir o algoritmo para computar o valor de τ, como demonstrado a seguir:

Passo 1. Iteração direta, dado θ, $\dot{\theta}$ e $\ddot{\theta}$, de i=1 até n, faça:

$$T_{i,i-1} = e^{-[A_i]\theta_i}M_{i,i-1}V_i = A_i\dot{\theta}_i + [Ad_{T_{i,i-1}}]V_{i-1}\dot{V}_i = A_i\ddot{\theta}_i + [Ad_{T_{i,i-1}}]\dot{V}_{i-1} + [ad_{V_i}]A_i\dot{\theta}_i \tag{2.17}$$

Passo 2. Iteração inversa, de n até $i = 1$, faça:

$$F_i = G_i\dot{V}_i - Ad^T_{V_i}(G_iV_i) + Ad^T_{T_{i+1,i}}(F_{i+1})\tau_i = F_i^T A_i \tag{2.18}$$

Já a dinâmica na forma fechada, para facilitar o entendimento, a equação recursiva que define τ pode ser organizado em uma forma fechada do tipo:

$$\tau = M(\theta)\ddot{\theta} + c(\theta, \dot{\theta}) + g(\theta) + J^T(\theta)F_{tip} \tag{2.19}$$

Os termos dependentes de massa, representados por $M(\theta)$, são fundamentalmente as massas efetivamente dos elos, os centros de massa e as inercias dele. Toda essa configuração influência estará relacionada com a posição angular do elo. Os termos dependentes de velocidade, representados na equação por $C(\theta, \dot{\theta})$ são fundamentalmente os termos relacionados com aceleração centrípeta e forças de Colioris, entretanto, forças de atrito são frequentemente relacionadas a velocidade também. o terceiro termo define a carga pela gravidade e, por fim, o último termo descreve os efeitos caso uma solicitação for aplicada ao end-effector, onde $J(\theta)$ denota a matriz jacobiana da cinemática direta expressa no mesmo sistema de coordenada de F_tip.

2.6 PLANEJAMENTO DE TRAJETÓRIA

Durante a programação e simulação de manipuladores são empregadas tarefas para que eles as executem. Estas tarefas são especificadas pelo usuário de diferentes formas, como por exemplo definindo o conjunto de pontos que o end-efector precisar passar em um determinado instante de tempo. Este histórico de posições, velocidades e acelerações das coordenadas das juntas em função do tempo é conhecido como trajetória e descreve o movimento desejado de um manipulador no espaço multidimensional, o planejamento desta é uma atividade recorrente para engenheiros de robôs. Diversos métodos podem ser utilizados para definir o trajeto, (FERNANDEZ; LYNCH; WENG, 2019) e (CRAIG, 2009) apontam três deles, Ponto-a-Ponto, Parametrizada e Trajetória linear.

2.6.1 Trajetória Ponto-a-Ponto

A primeira forma e a mais simples, define o caminho de um ponto a outro por meio de uma função polinomial, pode ser utilizada tanto para o espaço das juntas quanto para o espaço cartesiano. Neste tipo de movimento o manipulador inicia-se em uma determinada posição e orientação e deseja-se que ele vá para uma posição e orientação específica.

O que se deseja neste trabalho é que o computador converta essa meta descrita no espaço cartesiano em espaço das juntas usando a cinemática inversa e defina como será a curva de posição da junta com base na velocidade ou tempo estipulado pelo usuário (CRAIG, 2009). Para missões de transferência de objetos, este tipo de movimento é indicado quando a área de trabalho está livre de obstáculos (SPONG et al., 2006).

Algumas condições de contorno devem ser estabelecidas para se definir um percurso adequado. Define-se a velocidade inicial e velocidade final como sendo iguais a zero e, como ponto de partida, define-se o vetor com o conjunto dos valores das juntas igual a θ_0 e para a

configuração desejada como, θ_f para um tempo total de t_f, portanto:

$$\begin{aligned} \theta(0) &= \theta_0, && (2.20)\\ \theta(t_f) &= \theta_f, && (2.21)\\ \dot{\theta}(0) &= 0, && (2.22)\\ \dot{\theta}(t_f) &= 0. && (2.23) \end{aligned}$$

Estas quatro condições de contorno podem satisfazer um polinômio de terceiro grau (forma cúbica). A forma cúbica se dá por:

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{2.24}$$

Com a derivada de primeira ordem e de segunda ordem da equação anterior em relação a t tem se a velocidade da junta e a aceleração da junta.

$$\begin{aligned} \dot{\theta}(t) &= a_1 + 2a_2 t + 3a_3 t^2, && (2.25)\\ \ddot{\theta}(t) &= 2a_2 + 6a_3 t, && (2.26) \end{aligned}$$

Combinando 2.20 e 2.24 com as condições de contorno tem se que:

$$\begin{aligned} a_0 &= \theta_0, && (2.27)\\ a_1 &= 0, && (2.28)\\ a_2 &= \frac{3}{t_f^2}(\theta_f - \theta_0), && (2.29)\\ a_3 &= \frac{2}{t_f^3}(\theta_f - \theta_0) && (2.30) \end{aligned}$$

Com esses coeficientes calcula-se o polinômio cúbico que conecta qualquer posição inicial com qualquer posição final em relação ao espaço das juntas do manipulador em um movimento suave, ou seja, para velocidade inicial e final iguais a zero. Substituindo os coeficientes nas equações 2.24 e 2.25 encontra se as funções $\theta(t)$, $\dot{\theta}(t)$, $\ddot{\theta}(t)$.

É possível descrever alguns pontos de passagem (A expressão "ponto" se refere a sistemas de referência que fornecem tanto a posição quanto a orientação do efetuador) para o efetuador percorrer por eles sem que interrompa seu movimento. Estes pontos são convertidos para o espaço das juntas utilizando a cinemática inversa. As únicas diferenças nas condições de contorno são que $\dot{\theta}(0) = \dot{\theta}_0$ e $\dot{\theta}(t_f) = \dot{\theta}_f$.

Dessa forma os coeficientes passam a ser iguais a:

$$a_0 = \theta_0, \tag{2.31}$$

$$a_1 = \dot{\theta}_0, \tag{2.32}$$

$$a_2 = \frac{3}{t_f^2}(\theta_f - \theta_0) - \frac{2}{t_f}\dot{\theta}_0 - \frac{1}{t_f}\dot{\theta}_f, \tag{2.33}$$

$$a_3 = -\frac{3}{t_f^3}(\theta_f - \theta_0) + \frac{1}{t_f^2}(\dot{\theta}_f - \dot{\theta}_0) \tag{2.34}$$

Substituindo os coeficientes nas equações 2.24 e 2.25, para obter:

$$\theta(t) = (-\frac{3}{t_f^3}(\theta_f - \theta_0) + \frac{1}{t_f^2}(\dot{\theta}_f - \dot{\theta}_0))t^3 + (\frac{3}{t_f^2}(\theta_f - \theta_0) - \frac{2}{t_f}\dot{\theta}_f)t^2 + \dot{\theta}_0 t + \theta_0 \tag{2.35}$$

$$\dot{\theta}(t) = (-\frac{9}{t_f^3}(\theta_f - \theta_0) + \frac{3}{t_f^2}(\dot{\theta}_f - \dot{\theta}_0))t^2 + (\frac{6}{t_f^2}(\theta_f - \theta_0) - \frac{4}{t_f}\dot{\theta}_f - \frac{2}{t_f}\dot{\theta}_f)t + \dot{\theta}_0 \tag{2.36}$$

$$\ddot{\theta}(t) = (-\frac{18}{t_f^3}(\theta_f - \theta_0) + \frac{6}{t_f^2}(\dot{\theta}_f - \dot{\theta}_0))t + (\frac{6}{t_f^2}(\theta_f - \theta_0) - \frac{4}{t_f}\dot{\theta}_f - \frac{2}{t_f}\dot{\theta}_f) \tag{2.37}$$

As velocidades nos pontos de passagem devem ser conhecidas para se calcular o polinômio para cada junta. Estas velocidades podem ser encontradas pelas seguintes abordagens: 1) O usuário especifica os valores da velocidade cartesiana linear e angular para o efetuador para aquele instante; 2) O sistema escolhe automaticamente as velocidades adequadas no espaço das juntas ou no espaço cartesiano por meio de uma heurística; 3) O sistema escolhe as velocidades de forma que as acelerações no ponto de passagem seja contínua.

2.6.2 Trajetória Parametrizada

Em algumas situações recorrentes como pintura, soldagem, deseja-se que o manipulador percorra um caminho específico, e não apenas percorra um ponto inicial e uma meta. Esta trajetória pode ser descrita com uma função paramétrica no espaço cartesiano, em que as coordenadas x,y,z são funções do tempo. Para isso o usuário pode tanto definir o tempo desejado para completar o percurso ou definir uma velocidade linear constante.

Utilizando o método da cinemática inversa, calcula-se os valores das variáveis das juntas para cada ponto da trajetória no espaço cartesiano. Nos casos em que o manipulador possuir mais de três juntas ou graus de liberdade, um método numérico é usado para calcular as variáveis das juntas para os pontos da trajetória, com espaçamento Δs entre cada ponto, um tabela pode ser criada para armazenar os dados das juntas para cada instante da trajetória.

Em trajetórias parametrizadas, os pontos estão muito próximos um dos outros, isto implica que se torna inviável computar um polinômio para cada ponto. Dessa forma, uma derivada numérica pode ser empregada de forma a obter a velocidade e aceleração do espaço de

junta ao invés da utilização de uma forma algébrica. Uma derivada numérica é obtida utilizando a definição de derivada:

$$f^{'} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{2.38}$$

Onde a aproximação numérica se torna:

$$f^{'} \approx \frac{f(x+h) - f(x)}{h} \tag{2.39}$$

A aceleração pode ser obtida pela segunda derivada:

$$f^{''} \approx \frac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{h^2} \tag{2.40}$$

Em que $f_{(x0)}$ é o valor do ângulo de θ da junta i para um instante t. O valor de h deve ser pequeno o suficiente de maneira que se tenha uma boa aproximação, mas não tão pequeno para se elevar o custo computacional demasiadamente.

2.6.3 Trajetória linear

A trajetória retilínea pode ser entendida como um caso específico da trajetória parametrizada. Para este caso, também é necessário calcular os ângulos das juntas para pontos igualmente espaçados do percurso da linha. O efetuador terá sua velocidade linear constante durante o percurso, fazendo com que a curva de variável de junta se torne complexa.

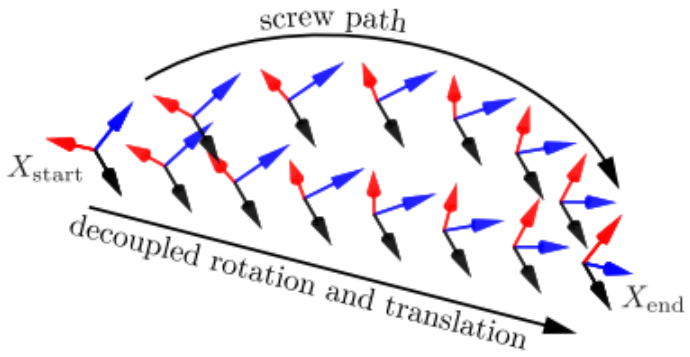
Uma das formas de calcular essa trajetória em linha reta é desacoplar o movimento de rotação do movimento de translação. Escrevendo $X = (R, p)$ como a matriz de transformação que referência um ponto desejado no espaço cartesiano, em que R representa a rotação dos eixos coordenados em relação a um referencial, comumente S e p é a translação entre os eixos. Pode-se definir o caminho como:

$$p(s) = p_{inicio} + s(p_{final} - p_{final}), \tag{2.41}$$

$$R(s) = R_{inicio} exp(\log(R_{inicio}^{T} R_{final}) \tag{2.42}$$

Onde s se trata de um valor escalar dependente do tempo $\in [0,1]$ que pode-se ser formulado como a equação 2.35, substituindo θ por s. A equação 2.41 representa a origem do sistema de coordenadas início seguindo uma linha reta enquanto rotaciona em relação ao eixo do efetuador de forma constante, a Figura 18 mostra o movimento em linha reta da origem do sistema de coordenada (*Decoupled*)

Figura 18 – Uma trajetória seguindo um movimento com um eixo fixo (Screw Motion) versus uma trajetória onde a origem do sistema de coordenada segue uma linha reta e velocidade angular constante



Fonte: (FERNANDEZ; LYNCH; WENG, 2019)

De forma parecida, pode-se definir um movimento em torno de um eixo de rotação fixo(*Screw Axis*) em que o movimento será constante em torno deste, porém não seguirá uma linha reta ou velocidade de rotação constante. Em que a equação é definida como:

$$X(s) = X_{inicio} exp(log(X_{inicio}^{-1} X_{final})s) \tag{2.43}$$

Alguns problemas podem aparecer quando é utilizado trajetórias lineares. O primeiro tipo é o problema de a trajetória possuir pontos intermediários inalcançáveis. Neste caso, embora a localização inicial do manipulador e a meta estejam dentro do volume de trabalho, é possível que alguns pontos da reta sejam inalcançáveis, um exemplo disso seria uma linha reta em que seu meio passe muito próximo à base do manipulador, não sendo possível o efetuador alcançar esses pontos sem que haja uma colisão do efetuador com algum elo.

O segundo problema ocorre quando os pontos da trajetória são próximos às singularidades. Se um manipulador está seguindo uma trajetória cartesiana em linha reta e se aproxima de uma configuração singular do mecanismo, a velocidade de uma ou mais juntas pode aumentar ao infinito. Sendo as velocidades de mecanismos limitadas, o que ocorre na prática é que existirá um desvio do manipulador do curso desejado (CRAIG, 2009).

O problema do tipo três surge quando um manipulador tem os pontos inicial e alvo atingidos por diferentes soluções. Neste caso o manipulador consegue alcançar todos os pontos da trajetória com algumas soluções ao invés de uma única. Neste caso, o sistema de planejamento pode detectar o problema antes de mover o robô pelo percurso (CRAIG, 2009).

2.6.4 Controle

Para que o manipulador robótico execute sua missão de forma satisfatória é necessário que existam técnicas de controle para realização do movimento desejado. Existem vários tipos de controle para robôs (CRAIG, 2009), sendo o tipo mais simples deles conhecido como controle de junta independente. Neste tipo de controle cada eixo do manipulador é controlado com um sistema de entrada-única e saída-única (do ingês, SISO: Single-input / single-output). Qualquer efeito de acoplamento devido ao movimento de outras juntas é tratado como distúrbio (SPONG; VIDYASAGAR, 2008)

O objetivo deste tipo de sistema de controle é escolher um compensador de tal forma que a saída da (plant) consiga rastrear ou seguir uma saída desejada (meta, velocidade, aceleração ou torque), dado um sinal de referência. O sinal de controle, contudo, não é a única entrada agindo no sistema (SPONG; VIDYASAGAR, 2008). O controlador é projetado, portanto, de forma que os efeitos dos distúrbios são reduzidos na saída.

Para isso considera-se que cada elo do manipulador está munido de um sensor que informe o ângulo e a velocidade angular. Cada junta deverá seguir uma trajetória que é especificada de forma única. Assim, os comandos são passados para os atuadores de cada junta de forma com que o torque (junta rotacional) ou a força (junta prismática) seja coerente para o ângulo ou posição requerida. Desta forma, é necessário que exista um processo de realimentação dos sensores para que a força, ou, o torque respectivo seja o necessário para se alcançar o ponto no instante requerido.

Isso posto, um dos métodos de se rastrear uma meta é utilizando um compensador PID (do inglês: Proportional, Integral, Derivative). Este tipo de controle é adequado para as aplicações que não envolvam movimentos muito rápidos, especialmente em robôs com altas reduções de engrenagem entre o atuador e o elo. É para rastreio de uma meta apenas (não variante no tempo), uma vez que é limitado quando efeitos adicionais como entrada de saturação, distúrbios e dinâmica não-modelada devem ser consideradas (SPONG; VIDYASAGAR, 2008).

2.6.4.1 Controlador PID

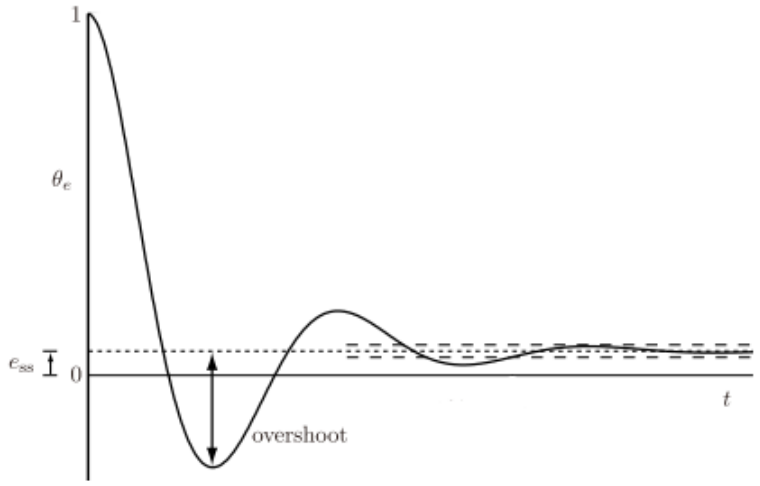
As estratégias para o controle das juntas em uma determinada trajetória partem do princípio de reduzir o erro durante sua execução, a equação 2.44 define o erro de uma junta, que por sua vez pode ser facilmente generalizado para n juntas.

$$\theta_{erro}(t) = \theta_d(t) - \theta(t) \tag{2.44}$$

Com isso, um controlador ideal tem o objetivo de levar este erro instantaneamente para zero e continuar nulo com o tempo. Porém, na prática não acontece de forma instantânea e com frequência o erro pode não ser completamente eliminado.

Na figura 19 está representada uma típica resposta de erro. Esta resposta pode ser descrita por uma região transiente, caracterizada pelo efeito de *overshoot* e por uma resposta de estado estacionário ("*steady-state response*"ou SSR). A SSR é caracterizada pelo *steady-syate error* (e_{ss}) que é o erro assintótico quanto o tempo tende ao infinito, como demonstrado na figura 19. De forma que um bom sistema de controle é aquele que tem um pequeno e_{ss} e um pequeno ou nenhum *overshoot*.

Figura 19 – Exemplo de resposta de erro descrevendo o efeito de *Overshoot* e a EES



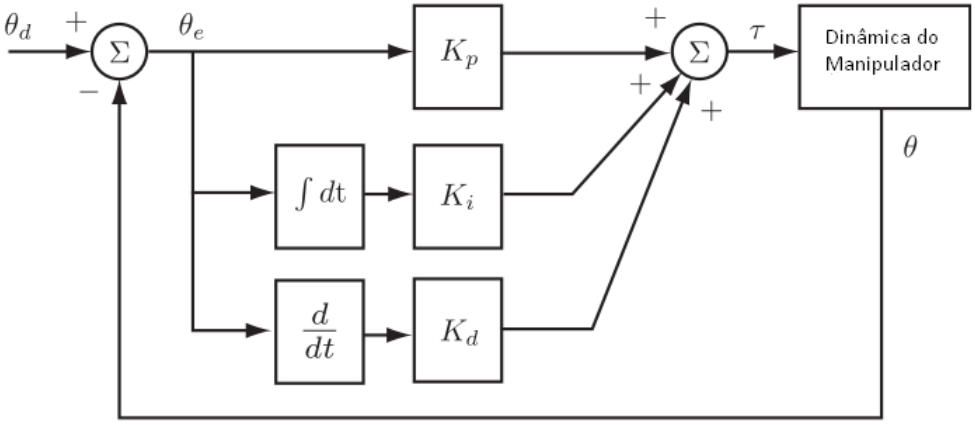
Fonte: (FERNANDEZ; LYNCH; WENG, 2019)

Pensando em minimizar este erro e nas condições apresentadas para um bom controlador, a estratégia do controle PID é medir a posição atual de cada uma das juntas constantemente e implementar um controlador de resposta ou *feedback controller*. Para isso utiliza o sistema de ganhos para corrigir a posição de acordo com a resposta encontrada pelos sensores. Estes ganhos são representados pelas constantes Kp (ganho proporcional), Ki(ganho integral) e Kd (Ganho diferencial) na equação para minimizar o erro, como demonstrado na equação 2.45

$$\tau = K_p\theta_e + K_i \int \theta_e(t)dt + K_d\dot{\theta}_e \tag{2.45}$$

Em que τ representa o torque do motor e θ_e é o erro que deve ser minimizado. Para melhor entender este sistema, se faz a analogia com o sistema massa-mola-amortecedor. Em que o ganho proporcional K_p atua como uma mola procurando reduzir o erro de posição $\theta_e = \theta_d - \theta$. O ganho derivativo K_d, por sua vez, atua como um amortecedor que procura reduzir o erro de velocidade $\dot{\theta}_e = \dot{\theta}_d - \dot{\theta}$. Por fim, o Ganho integral K_i é utilizado para eliminar o EES. O diagrama de bloco do controlador PID é dado na figura 20

Figura 20 – Diagrama em blocos de um controlador PID



Fonte: Autoria Própria

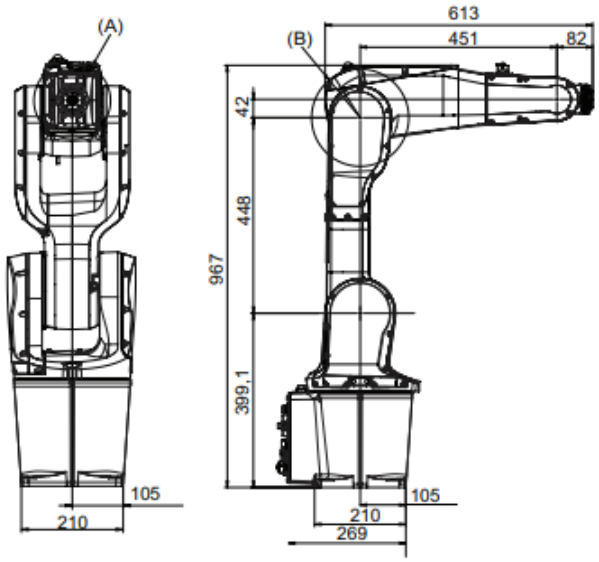
3 METODOLOGIA

3.0.1 Manipulador utilizado

O sistema robótico utilizado para análise cinemática e dinâmica é o ABB IRB 1200, um robô manipulador industrial, do tipo antropomórfico, com seis eixos de revolução (6 GDL) sendo que são 3 gdl para posicionar e 3 gdl para definir a orientação do atuador final, tem alta adaptabilidade para diversos ambientes e aplicações. O robô pesa 54 kg com alcance de 903 mm, com capacidade para cargas de 5 kg. A velocidade máxima do ponto central da ferramenta (TCP *tool centre point*) é de 7,3m/s e a repetibilidade é de 0,02mm. As dimensões principais são mostradas na figura 21.

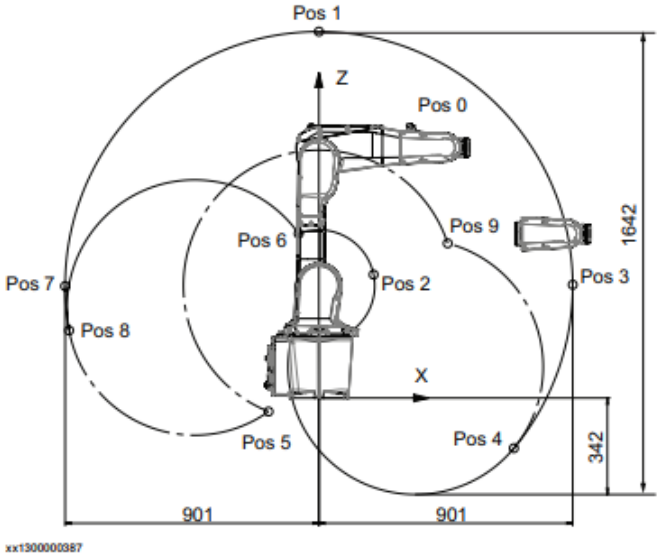
O espaço de trabalho definido para os eixos 2 e 3 segundo o fabricante é demonstrado na figura 22. Os pontos de calibração levando em consideração o centro da ferramenta é demonstrado na figura 23

Figura 21 – Dimensões do robô ABB IRB 1200 5/09



Fonte: <https://new.abb.com/products/robotics/industrial-robots/irb-1200/irb-1200-data>

Figura 22 – Espaço de Trabalho do robô ABB IRB 1200 5/09



Fonte: <https://new.abb.com/products/robotics/industrial-robots/irb-1200/irb-1200-data>

Figura 23 – Pontos de calibração do robô ABB IRB 1200 5/09

Position in the figure	Positions at wrist center (mm)		Angle (degrees)	
	X	Z	Axis 2	Axis 3
Pos0	451	889	0º	0º
Pos1	0	1300	0º	-85º
Pos2	194	438	0º	+70º
Pos3	901	402	+90º	-85º
Pos4	692	-178	+130º	-85º
Pos5	-179	-48	-100º	-200º
Pos6	-72	583	-100º	+70º
Pos7	-901	397	-90º	-85º
Pos8	-887	240	-100º	-85º
Pos9	458	549	+130°	-200°

Fonte: <https://new.abb.com/products/robotics/industrial-robots/irb-1200/irb-1200-data>

As variáveis dinâmicas como massa (em kg) e momento de inercia (em $kg.m^2$) foram obtidas a partir dos modelos CAD fornecidos pelo fabricante e também pelos dados fornecidos pela Ros_industrial, as informações foram organizadas na tabela 3.0.1

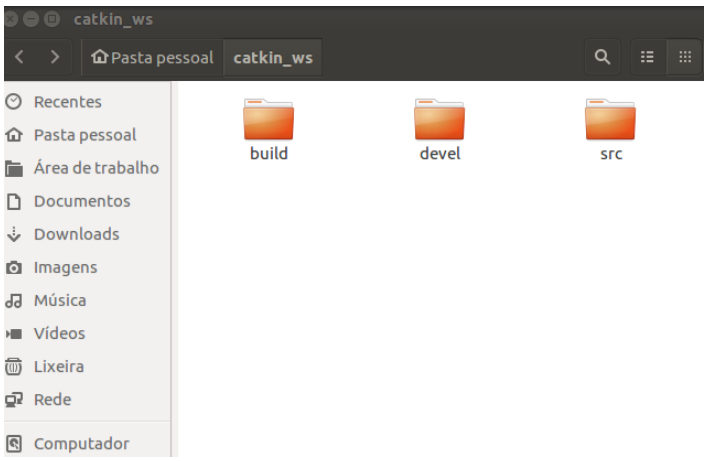
	Base	**link 1**	**link2**	**link 3**	**link 4**	**link 5**	**link 6**
Massa	13.77	11.84	17.54	7.46	2.74	0.63	0.14
Ixx	1.02E-01	1.12E-01	4.94E-01	2.52E-02	5.73E-03	5.03E-04	1.00E-03
Ixy	4.95E-04	-4.55E-05	1.32E-05	1.43E-04	-1.31E-04	-1.03E-05	0.00E+00
Ixz	3.11E-04	2.81E-04	2.09E-04	-5.66E-03	3.80E-04	-8.74E-08	0.00E+00
Iyy	1.40E-01	9.15E-02	4.63E-01	9.06E-02	1.18E-02	1.09E-03	1.00E-03
Iyz	-2.45E-04	-1.10E-04	-1.80E-03	1.42E-04	-2.27E-05	2.08E-07	0.00E+00
Izz	1.30E-01	8.76E-02	8.94E-02	8.25E-02	1.14E-02	9.19E-04	1.00E-03

3.0.2 Configuração do ambiente

Os Packages utilizado neste trabalho foram retirados do repositório GIT <https://github.com/ros-industrial/abb_experimental.git> e então alguns arquivos foram modificados pelo autor e organizados no repositorio GIT <https://github.com/brunoclaudio/TCC_Simulacao_robotica.git>. Nesta seção será comentado algum destes arquivos para melhor entender os procedimentos realizados.

Inicialmente para que se possa usar o ROS é necessário configurar o espaço de trabalho, neste trabalho foi utilizado a ferramenta Catkin_make por ser mais recomendada nos tutoriais presentes no site www.ros.org, como demonstrado na figura 24. Foi utilizado a versão Kinetic do Ros, no sistema operacional Linux Ubuntu 16.04.7, porém para projetos científicos e desenvolvimento de tecnologias esta versão não é recomendada, recomenda-se, até o presente momento deste trabalho, utilizar a versão ROS Noetic Ninjemys. A versão Kinetic foi escolhida unicamente devido a que os packages utilizados estarem estáveis e devido a facilidade do autor com esta versão, além de o autor ter encontrado inicialmente maior suporte na comunidade para o ros Kinetic.

Figura 24 – Espaço de trabalho



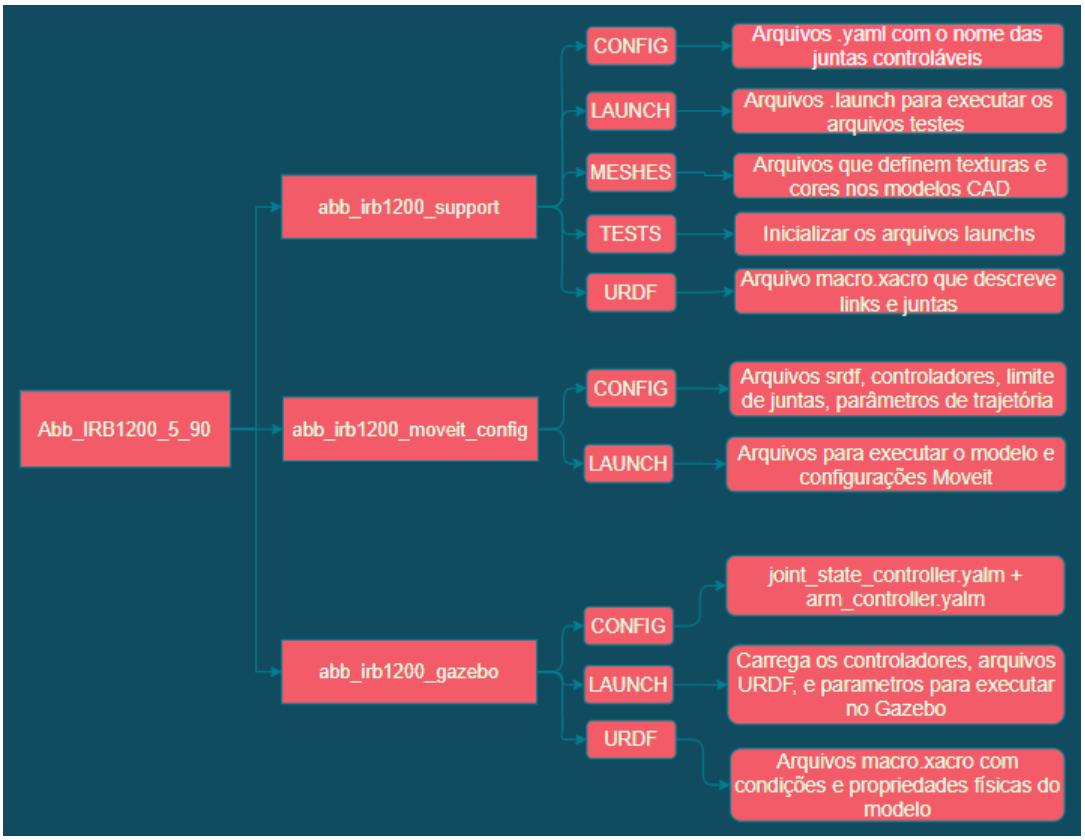
Fonte: Autoria própria

Em seguida, foi instalado o conjunto de Packages do repositório git https://github.com/ros-industrial/abb_experimental deste, três Packages foram baixados, a hierarquia das pastas baixadas e uma breve explicação de cada uma delas é esquematizado no diagrama da figura 25.

O primeiro Packages, abb_irb1200_support, contém os arquivos de suporte, com as definições das juntas e alguns arquivos do tipo launch para visualizar o modelo em 3D, com objetivo de verificar se existe alguma inconsistência no modelo.

O segundo arquivo é o Abb_irb1200_moveit_config, referente as configurações do framework Moveit!, nele foi definido o grupo de elos ao manipulador para que o software compute como objetos em uma cadeia cinemática. Foi definido também a referência entre o elo

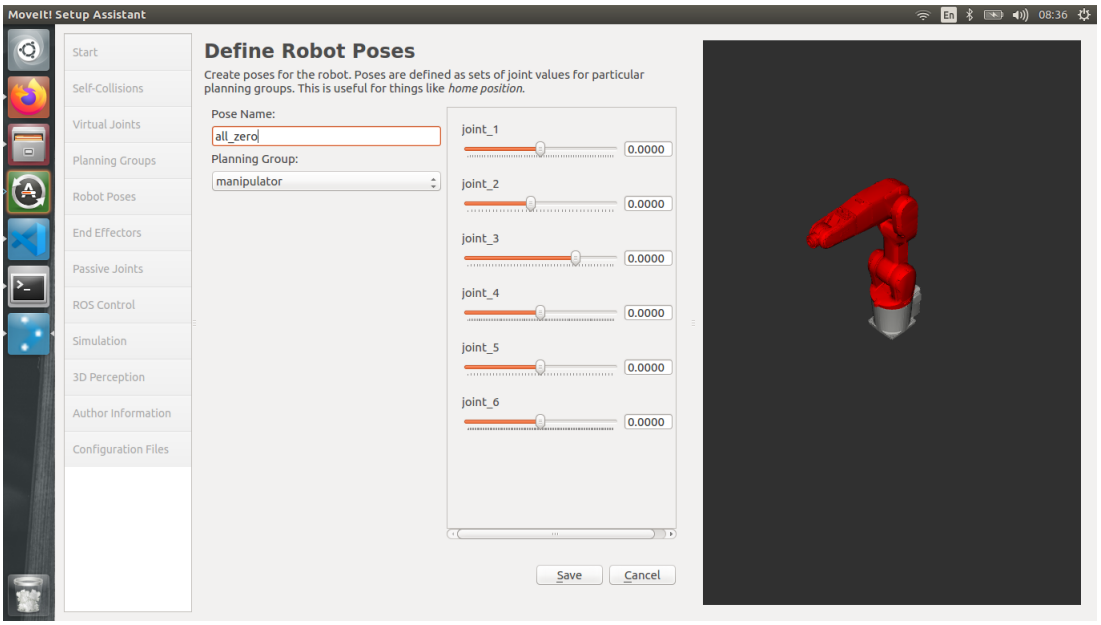
Figura 25 – Diagrama da hierarquia de arquivos proveniente do repositório Git Ros_Industrial



Fonte: Autoria própria

da base do manipulador e um elo do ambiente simulado, chamando este último de Link Virtual e então foi gerado a condição de colisão para cada um dos objetos. Utilizando o Setup Assistent do Moveit, foi definido o Grupo de Planejamento como "Manipulator", que se trata da cadeia cinemática a qual será manipulada utilizando o moveit_group_python_API.

Figura 26 – Grupo de links para planejamento de trajetória definidos no Moveit Setup Assistant

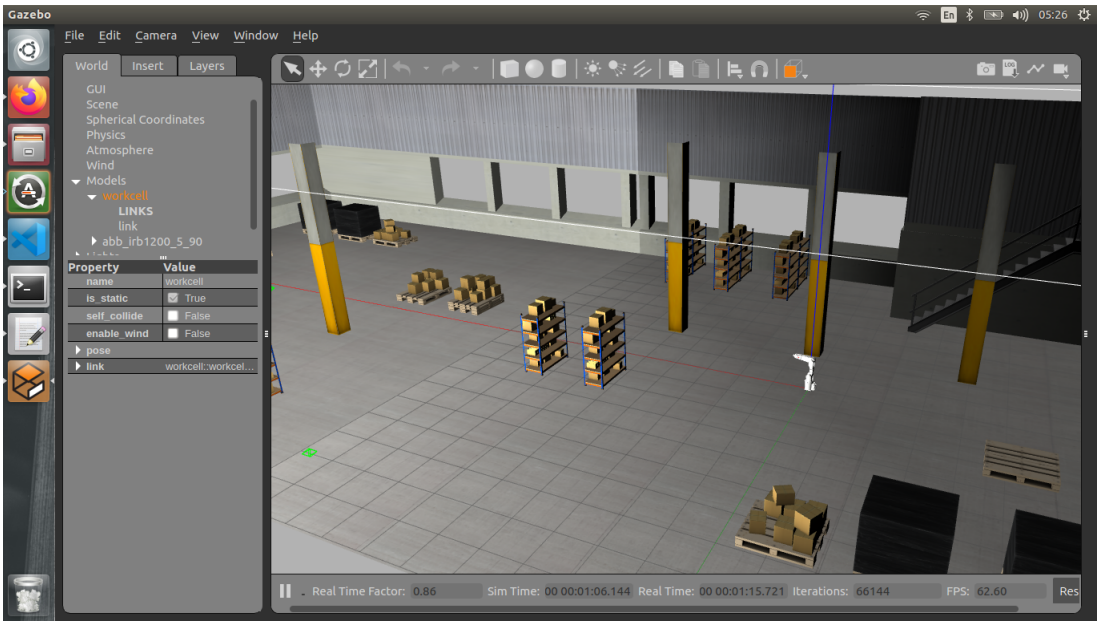


Fonte: Autoria própria

Ainda neste Packages foi criado um arquivo launch para inicializar, características importantes como, o nome das juntas, os parâmetros do robô, inicializar o joint_state_publisher que é responsável por publicar e receber os comandos da trajetória e também fazer com que o Gazebo consiga estar sincronizado com a visualização do RVIZ, o arquivo modificado se encontra no Anexo B.

O terceiro, abb_irb1200_gazebo encontra-se as condições para simular o modelo no ambiente Gazebo. Vale destacar alguns destes arquivos, a execução inicia com o script gazebo.launch, nele é iniciado a simulação do ambiente, carregado os arquivos URDF que se encontram tanto nas pastas de Support, quanto na pasta URDF deste mesmo package. Para este trabalho foi implementado o cenário industrial da competição ARIAC (Agile Robotics for Industrial Automation Competition) como demonstrado na figura 27.

Figura 27 – Ambiente de simulação Gazebo



Fonte: Autoria própria

Segundo, "control.launch"executa as configurações de controle das juntas (arquivo arm_state_controller.yalm) com os controles PID de cada uma das juntas, limite de tempo de cálculo e tipo de controlador (foi utilizado o controle por trajetória) definidas na pasta "CONFIG". Por fim, é publicado atualizações do estado da cadeia cinemática do robô do tópico "robot_state_publisher"para o tópico "TF"(*transpose frame*) onde é computado as matrizes de transformação em cada instante de tempo da trajetória. A hierarquia das pastas e arquivos deste Packages são representadas no diagrama da figura 28.

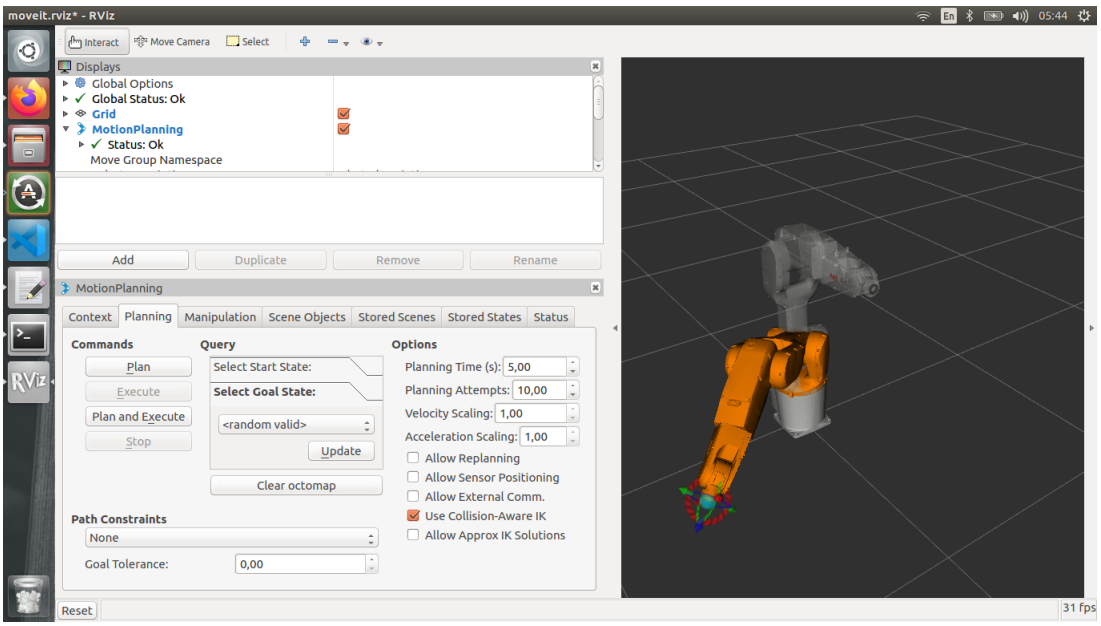
Figura 28 – Hierarquia dos arquivos de Abb_IRB1200_gazebo.launch

Fonte: Autoria própria

Feito as configurações no simulador Gazebo e também no framework Moveit, o planejamento das trajetórias pode ser executado. Neste trabalho foi utilizado a API do move_group

para a linguagem python, foi criado um algoritmo para o planejamento de trajetória, nele foi especificado o grupo de manipulação como "manipulator", definido na fase de configuração do Moveit, posteriormente foi criado sub-rotinas para executar trajetórias: 1) Ponto a Ponto; 2) Trajetórias Lineares e 3) Trajetórias parametrizadas. Também foi utilizado a visualização no RVIZ para verificar a execução do algoritmo e verificar os limites de trabalho do manipulador como mostra a figura 29.

Figura 29 – Visualização e planejamento utilizando a Interface Gráfica do RVIZ e Moveit



Fonte: Autoria própria

Para analisar as trajetórias foi criado um algoritmo para plotar os valores de posição, velocidade e aceleração e torque das juntas. Também foi avaliado o erro de trajetória para cada uma das juntas. Para criar o algoritmo utilizou o tópico "joint_trajectory_action"que contém os dados de posição, velocidade, aceleração desejados e reais. Foi utilizado para plotar os gráficos a biblioteca Math_plot lib para linguagem Python.

4 RESULTADOS

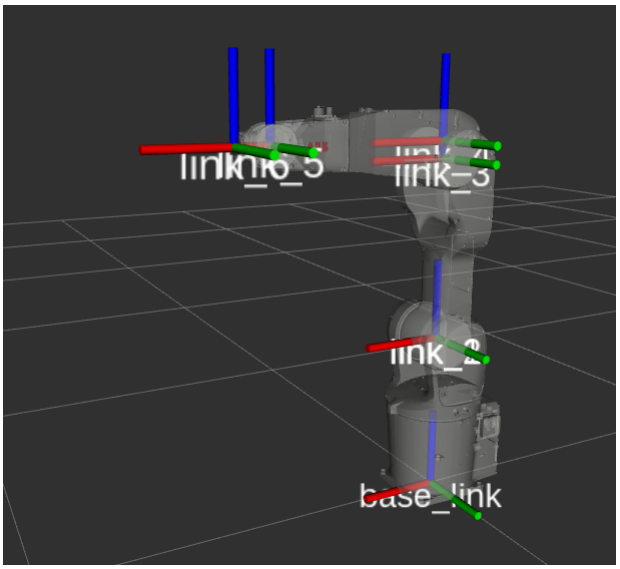
Este capítulo tem como objetivo apresentar os resultados da simulação robótica do manipulador industrial ABB IRB 1200 5/90 nos softwares de código aberto Ros, Moveit e Gazebo fundamentados nos capítulos anteriores. Será avaliado durante a simulação os principais tipos de trajetórias que robôs manipuladores executam nos seus ambientes de trabalho, estas são as trajetórias do tipo Ponto-a-Ponto, trajetórias parametrizadas e Linear.

Este capítulo tem como objetivo também, discutir a aplicação das ferramentas apresentadas em ambiente acadêmico para fins de desenvolvimento, pesquisa e ensino de acordo com a realidade do Instituto Federal do Piauí Campus Central.

4.1 PRÉ-SIMULAÇÃO

A Figura mostra a representação cinemática do manipulador proposto, é inserido nessa figura o alinhamento dos eixos de referência de cada junta. É possível perceber que os eixos da junta 1 e junta 2 estão sobrepostos, isto é feito com o intuito de simplificar as matrizes de cinemática direta, nota-se também que o Sistema de Coordenada das juntas estão no mesmo sentido. Na figura 30, o eixo z é representado em azul, o eixo e y em verde e, por fim, o eixo x é representado pela cor vermelha.

Figura 30 – Disposição dos eixos de rotação e de referência.



Fonte: Autoria Própria - retirada da ferramenta RVIZ

Na tabela 1 está representada o Helicoide dos eixos de cada uma das juntas. Nas linhas da tabela encontra-se as coordenadas do vetor unitário que representa a direção de rotação dos eixos (wx,wy,wz) e também a velocidade linear em cada um deles.

Tabela 1 – Helicoide de cada um dos eixos(representado pelas colunas da tabela)

	S1	**S2**	**S3**	**S4**	**S5**	**S6**
wx	0	0	0	1	0	1
wy	0	1	1	0	1	0
wz	1	0	0	0	0	0
vx	0	-0.399	-0.847	0	-0.889	0
vy	0	0	0	0.889	0	0.889
vz	0	0	0	0	0.451	0

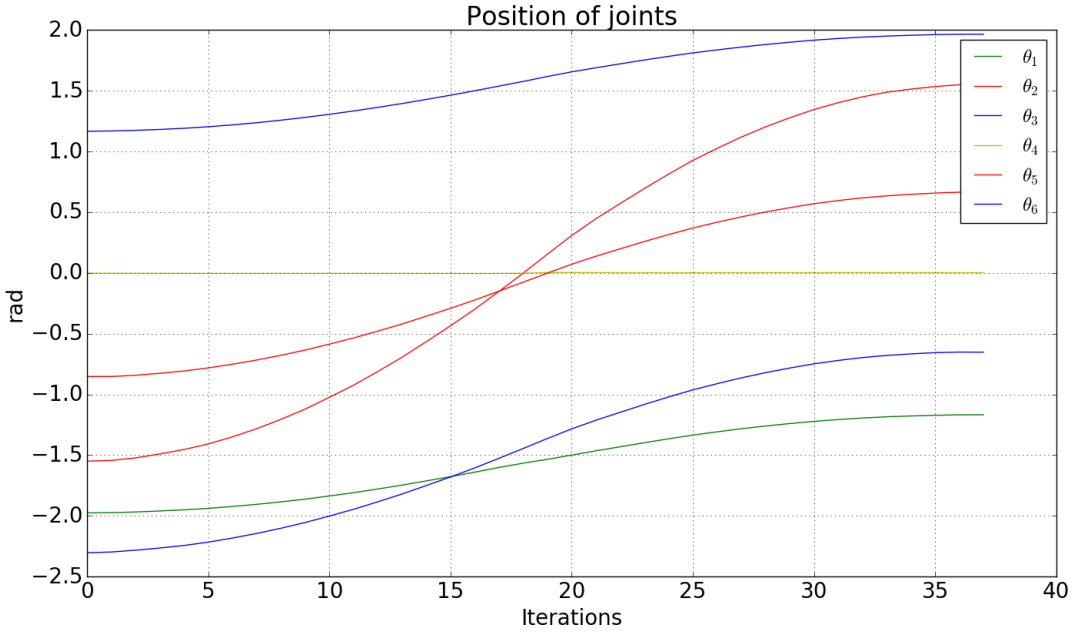
4.2 EXEMPLOS DE TRAJETÓRIA

A seguir são apresentados alguns exemplos de trajetórias com o intuito de analisar o comportamento cinemático do manipulador.

4.2.1 Trajetória Ponto a Ponto

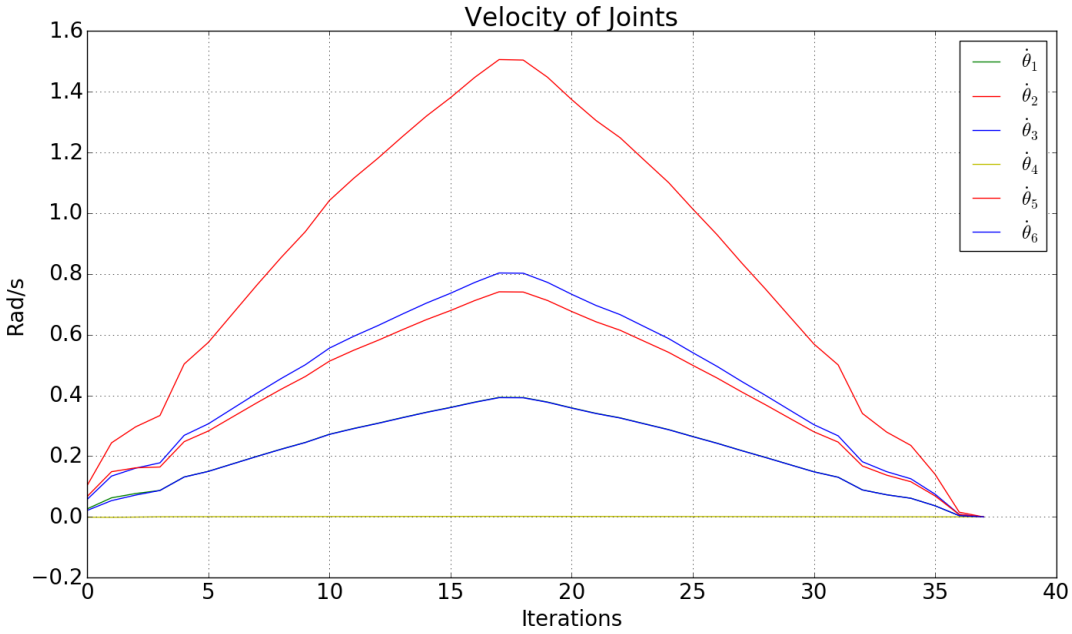
O primeiro exemplo consiste em um movimento coordenado através de interpolação de terceiro grau das juntas. O manipulador estava posicionado na origem (em milímetros): $x = 300$, $y = 700$, $z = 0$ e foi movido para a posição: $x = 300$, $y = -700$, $z = 0$ com orientação não arbitrária, definida de forma que o último elo fique orientado em direção ao plano da base, o tempo de trajetória igual a 3,7 segundos. Os gráficos foram obtidos utilizando um Nó que subscreve no tópico *Joint_trajectory_action*. Neste tópico foi possível obter os valores de posição, velocidade e aceleração, entretanto os valores de esforço das juntas não foram atualizados. Consultando a documentação e foruns do ROS observou que em algumas versões da ferramenta a mensagem "joint_state/effort"está vazia. As figuras 31, 32 e 33 mostram, respectivamente, as curvas de posição, velocidade e aceleração. A trajetória foi calculada em 37 iterações.

Figura 31 – Posição das juntas em rad por número de iterações para a trajetória do tipo Ponto a Ponto.



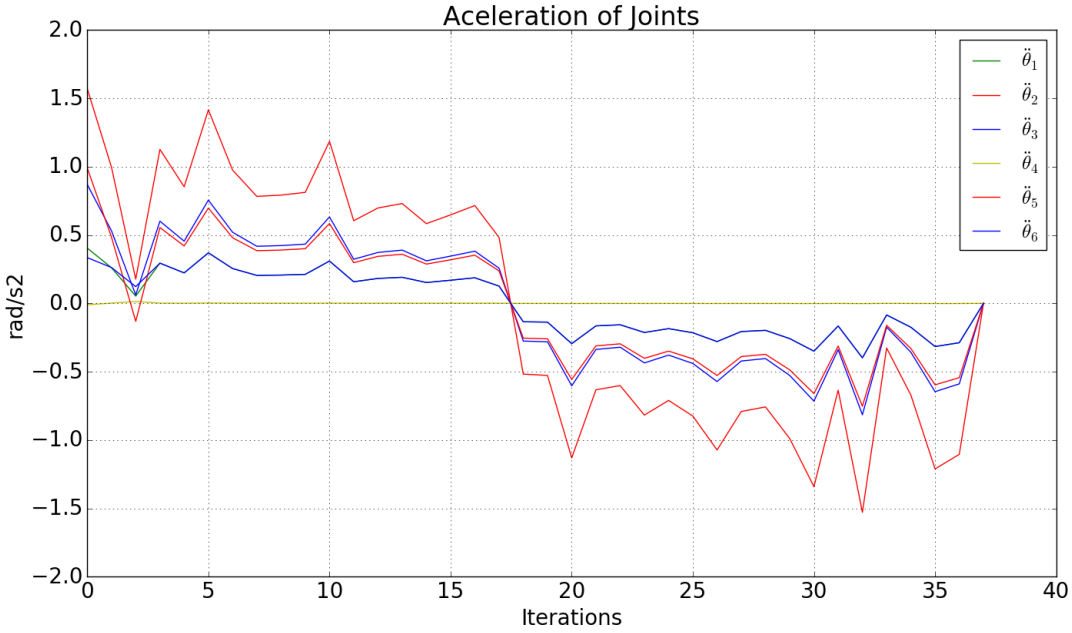
Fonte: Autoria Própria

Figura 32 – Velocidade das juntas em rad/s por número de iterações para a trajetória do tipo Ponto a Ponto.



Fonte: Autoria Própria

Figura 33 – Aceleração das juntas em rad/s^2 por número de iterações para a trajetória do tipo Ponto a Ponto.

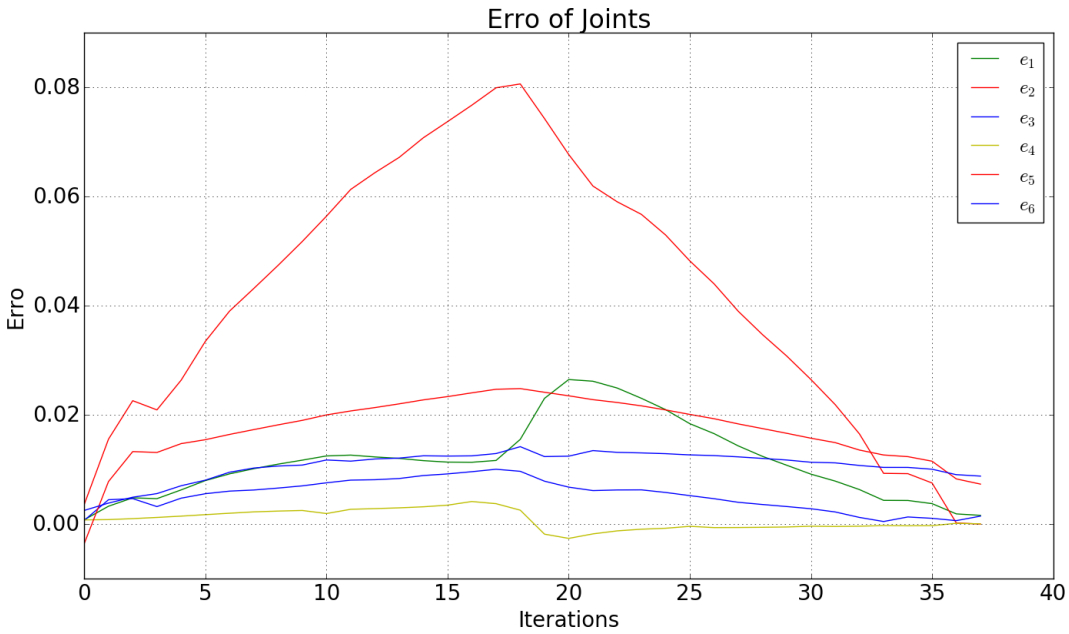


Fonte: Autoria Própria

É possível observar que o gráfico das posições das juntas apresenta curva suave, enquanto o da velocidade apresenta algumas pequenas variações durante a execução, isso é referente ao polinômio de terceiro grau escolhido para fazer a interpolação dos pontos da trajetória que o Serviço Joint_trajectory_action utiliza, que é um Serviço de baixo nível, ele tem como objetivo apenas a execução da geometria da trajetória e não a execução de curvas suaves de aceleração e nem mesmo o desvio de objetos. Dessa forma pode-se observar durante a execução da trajetória trepidações por parte do robô, isto em um caso real não é adequado, pois a trajetória do tipo ponto a ponto é frequentemente utilizada para movimentação de carga, logo estas descontinuidades na aceleração causam vibrações em excesso. Para movimentos mais suaves recomenda-se utilizar o Serviço Move_arm, porém sua implementação requer domínio da linguagem de programação C++ e não funciona corretamente em algumas versões do ROS.

No gráfico do erro das juntas representado na figura 34 observa-se que o erro aumenta até a metade da trajetória, tendo seu pico máximo na iteração 17 e começa a diminuir até finalizar com erro máximo de posição de 0,015 da junta 3.

Figura 34 – Erro de posição das juntas é dado pela diferença entre o real e planejado pelo número de iterações para a trajetória do tipo Ponto a Ponto.



Fonte: Autoria Própria

4.2.2 Trajetória parametrizada - Círculo

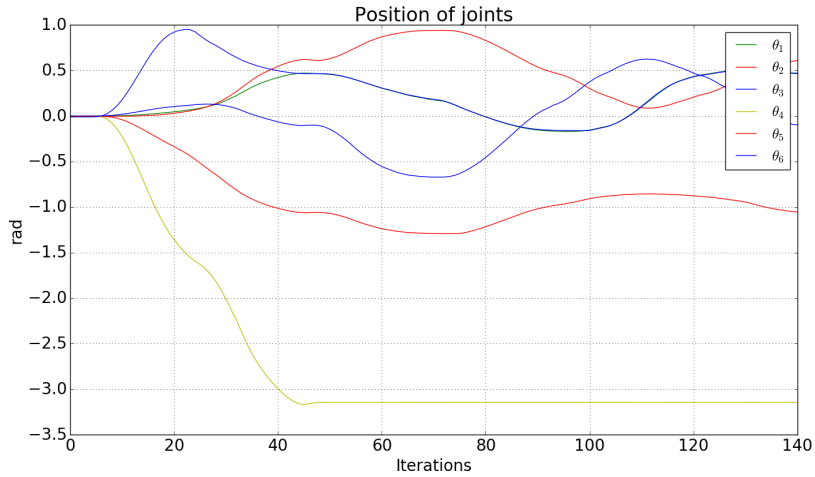
Como exemplo de trajetória parametrizada, foi escolhido um círculo de raio r e centro em (h, k) que pode ser parametrizado pelas seguintes equações:

$$x = a, \tag{4.1}$$
$$y = r.cos(n) + h, \tag{4.2}$$
$$z = r.sin(n) + k, n \in [0, 2\pi]. \tag{4.3}$$
$$\tag{4.4}$$

Em que r = 200mm, a = 500 mm, h = 100 mm, k = 600, escolhidos arbitrariamente desde que a trajetória se encontre dentro do espaço de trabalho do robô.

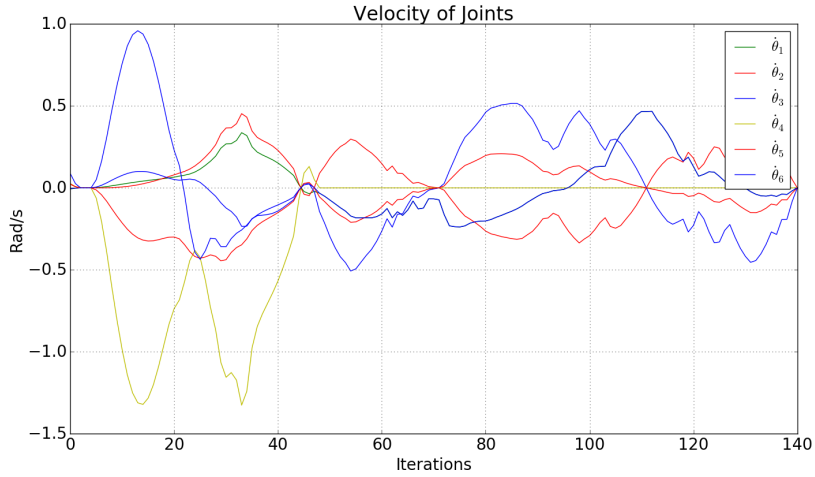
Os gráficos de posição das juntas, assim como os de velocidade e aceleração estão representados, respectivamente na figura 35, figura 36 e figura 37. A trajetória foi executada em 9.5 segundos, levando 95 iterações. Observou-se que a execução da trajetória no plano cartesiano foi feita corretamente, porém durante a trajetória o manipulador teve curtas paradas, isto pode ter ocorrido devido a que em alguns pontos da trajetória estar próximo a pontos de singularidade do manipulador, este índice pode ser acompanhado na ferramenta de visualização Rviz durante a trajetória, estas paradas podem ser observadas na iteração 69 no gráfico da velocidade das juntas.

Figura 35 – Posição das juntas em rad por número de iterações para a trajetória do tipo Parametrizada Circular.



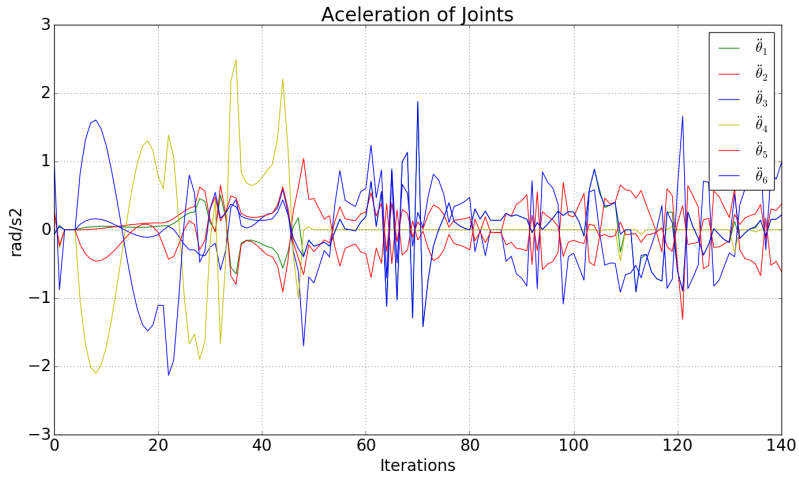
Fonte: Autoria Própria

Figura 36 – Velocidade das juntas em rad/s por número de iterações para a trajetória do tipo Parametrizada Circular.



Fonte: Autoria Própria

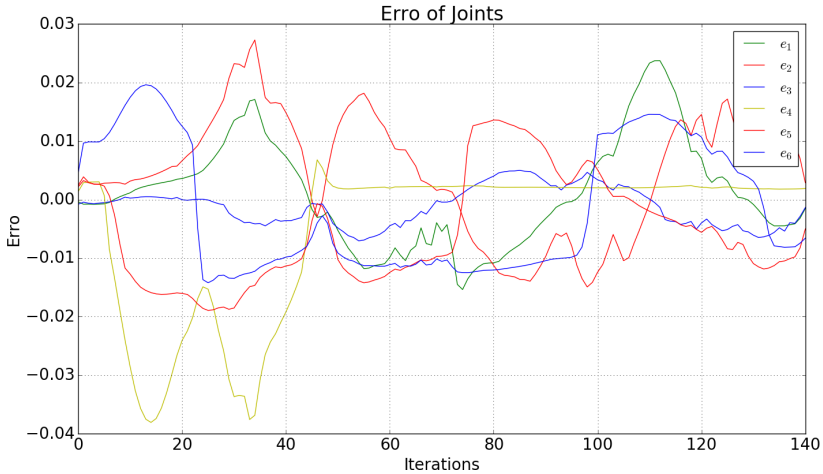
Figura 37 – Aceleração das juntas em rad/s^2 por número de iterações para a trajetória do tipo Parametrizada Circular.



Fonte: Autoria Própria

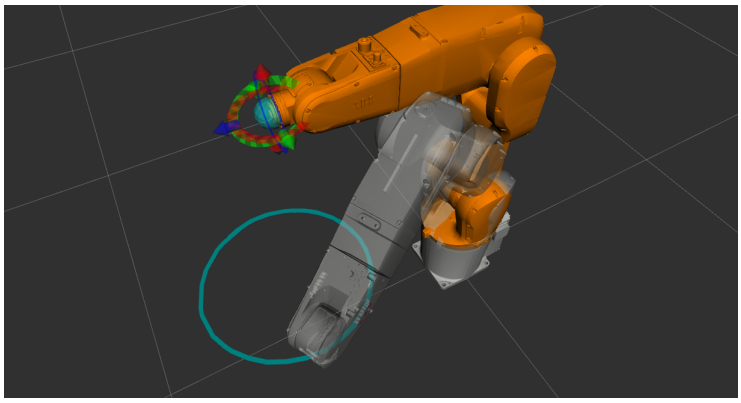
Na figura 38 observa que o erro oscila durante toda a trajetória, com maior variação na iteração 65 para a junta 1. A trajetória finaliza com erro máximo de -0.007 em rad para a junta 3. Observa-se que a junta 4 teve baixo desvio da trajetória e consequentemente erro estável durante o processo, isso ocorre devido a que, para esta trajetória, o movimento de rolagem dessa junta foi pouco utilizado. O que inversamente ocorre com a junta 1, 2 e 3, já que são elas que suportam maior quantidade de massa, logo durante o movimento são estas juntas que estão mais propensas a variação da energia inercial dos respectivos e demais elos da cadeia, pois o centro de massa se encontra distante do eixo das juntas em questão.

Figura 38 – Erro das juntas em rad por número de iterações para a trajetória do tipo Parametrizada Circular.



Fonte: Autoria Própria

Figura 39 – Representação da trajetória no espaço cartesiano pela ferramenta Rviz.



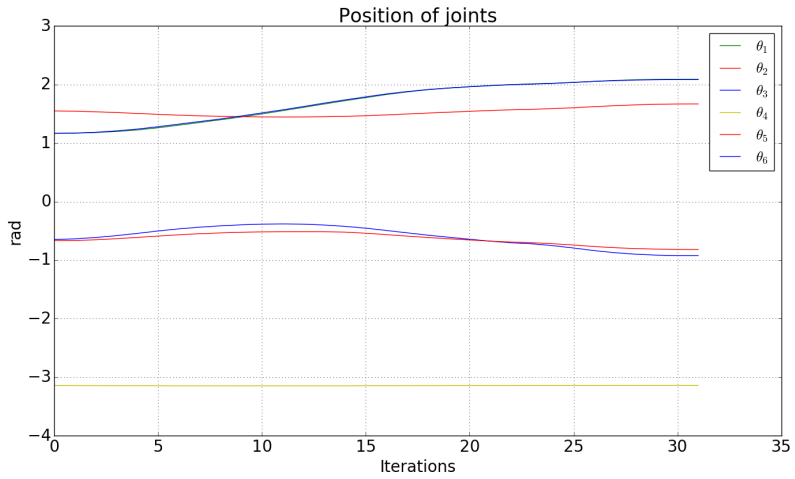
Fonte: Autoria Própria

4.2.3 Trajetória Linear

Outro tipo de trajetória parametrizada bastante utilizada, principalmente para missões de soldagem, é a trajetória do tipo linha reta. Esta trajetória enfrenta a particularidade que, devido as juntas executarem movimentos rotativos, o trajeto final em linha reta será a soma de pequenos arcos executados com a combinação das posições da cadeia cinemática.

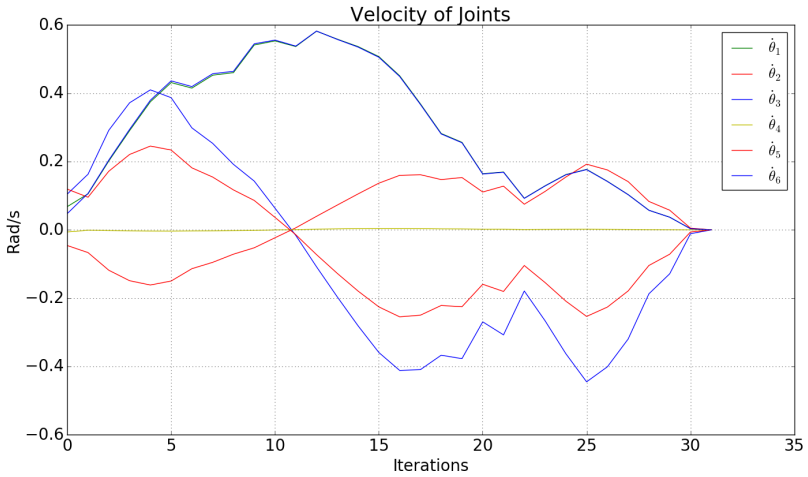
O manipulador estava posicionado (em milimetros): $x = 300$, $y = 700$, $z = 0$ e foi movido para a posição: $x = -400$, $y = 700$, $z = 0$ com orientação não arbitrária, definida de forma que o último elo fique orientado em direção ao plano da base, como as demais trajetórias. As figuras 40, 41 e 42 mostram, respectivamente, as curvas de posição, velocidade e aceleração. A trajetória foi calculada em 30 iterações com tempo de execução igual a 3 segundos.

Figura 40 – Posição das juntas em rad por número de iterações para a trajetória do tipo Parametrizada em Linha.



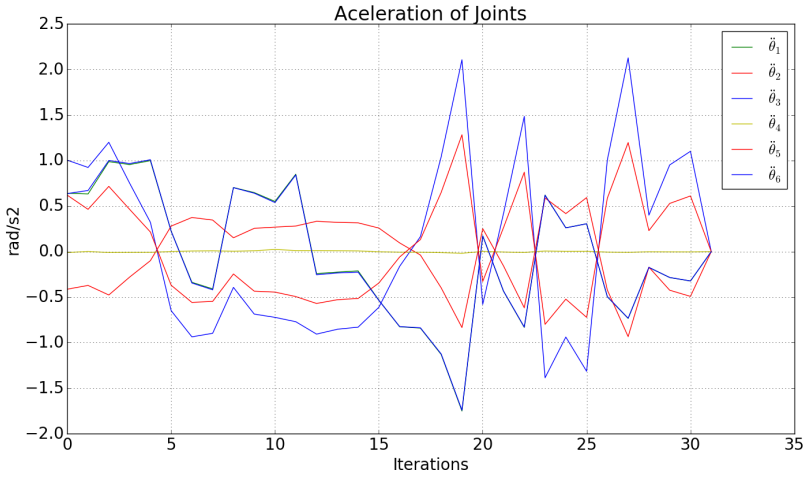
Fonte: Autoria Própria

Figura 41 – Velocidade das juntas em rad/s por número de iterações para a trajetória do Parametrizada em Linha.



Fonte: Autoria Própria

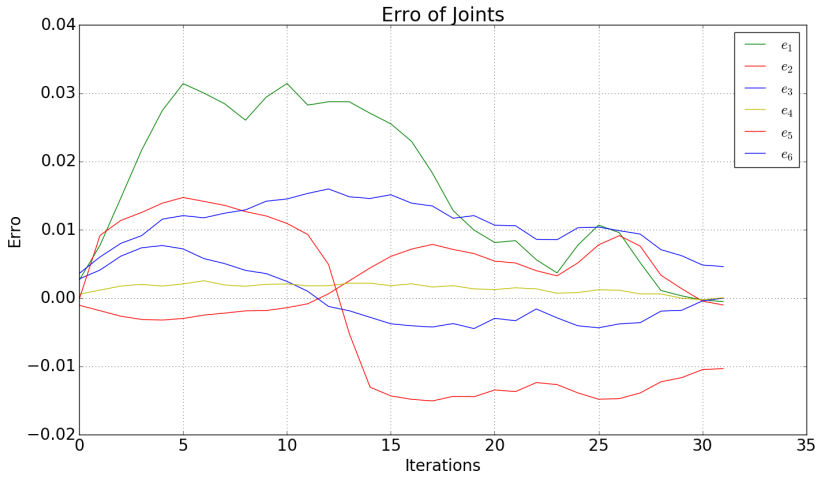
Figura 42 – Aceleração das juntas em rad/s^2 por número de iterações para a trajetória do Parametrizada em Linha.



Fonte: Autoria Própria

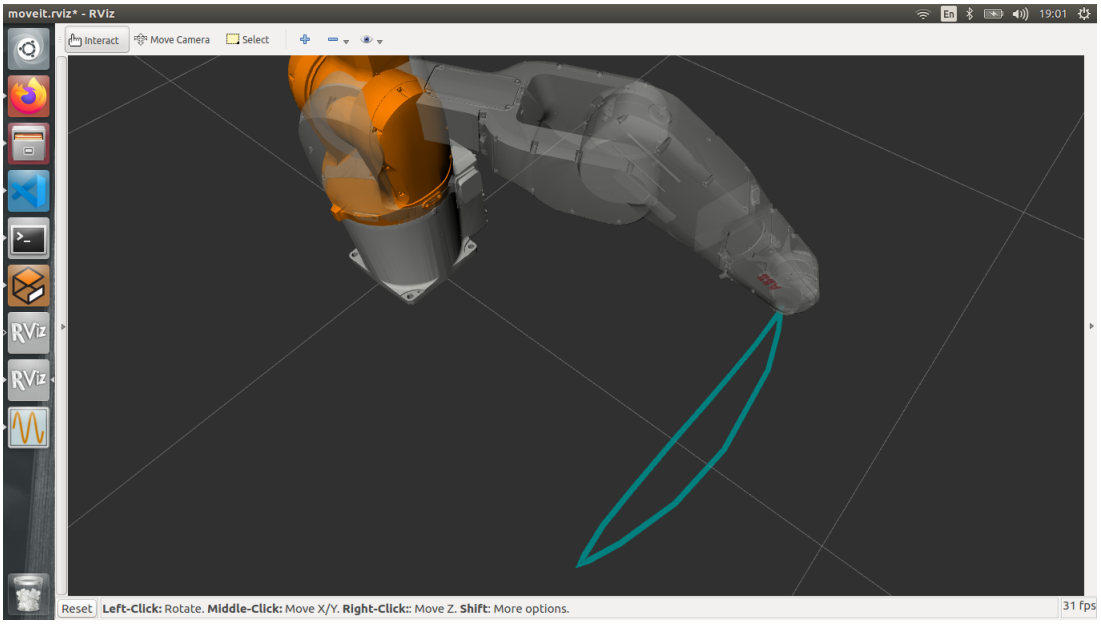
Observa-se na figura 43 que o erro para este exemplo de trajetória foi maior durante o percurso do que se comparado a trajetória circular, e teve picos menores do que se comparado ao exemplo da trajetória Ponto a Ponto. O erro máximo final foi de -0.01 para a junta 2.

Figura 43 – Erro das juntas em rad por número de iterações para a trajetória do tipo Parametrizada em Linha.



Fonte: Autoria Própria

Figura 44 – Representação da trajetória no espaço cartesiano pela ferramenta Rviz - Trajetória em Linha.



Fonte: Autoria Própria

4.2.4 Aplicação da ferramenta

De forma geral a implementação de sistemas robóticos não é uma tarefa simples, muito menos conseguir um modelo digital para simular ambientes e robôs reais. Porém de forma geral a ferramenta ROS apresentou facilidade em relação a flexibilidade de aplicar soluções de outros usuários em uma aplicação própria.

Porém observa-se que o escopo de conhecimento necessário para se utilizar a ferramenta ainda é elevado. É necessário que o usuário entenda de alguns conteúdos básicos como Sistemas Operacionais, Linguagens de programação como Python, C++, XML e entender propriamente o midware ROS. Dessa forma, o desenvolvimento de interfaces gráficas para enviar comandos e obter gráficos relacionados a parâmetros de trajetória é necessário para facilitar a jornada do usuário durante a utilização da ferramenta.

Em adição, o software de simulação Gazebo se mostrou relevante para utilização em meios acadêmicos e industriais, como mostrado na literatura. Neste trabalho foi possível observar que a implementação dos modelos no ambiente virtual foi simples, pois diversos repositórios permitiram reutilizar suas soluções, necessitando de poucas alterações. Além de que não foi necessário computadores com processadores mais avançados para conseguir executar a simulação, o que se torna vantagem se comparado com softwares como Simulik-Matlab.

Já a interface do framework Moveit se mostrou intuitiva e suas funções permitiram que a execução do planejamento de trajetória tornasse uma tarefa menos complexa, pois depois de configurar os Packages relacionados ao Moveit, as trajetórias foram executadas, tanto utilizando a ferramenta de visualização RVIZ, como utilizando o Move_group_comand para enviar comandos ao robô de forma simplificada. Porém para aplicações de desenvolvimento e pesquisa não se recomenda utilizar a API na linguagem python do move_group para planejamento de trajetórias, pois está é designada apenas para usuários iniciantes.

Em adição, as trajetórias foram executadas utilizando a interface Joint Trajcetory Controller. Esta interface de baixo nível (ou seja, mais simples) é mais eficiente em relação a processamento, porém deixa de focar em alguns aspectos relacionados com o planejamento de trajetórias de robôs manipuladores como: 1) Gerar trajetórias com curvas de velocidade e aceleração suaves, como mostrado no capítulo da 2; 2) Desvio de objetos. Dessa forma, para a execução em aplicações reais e repetitivas é necessário utilizar outras interfaces como a Move_arm ou até mesmo desenvolver e aplicar um novo algoritmo, para esta opção, o Moveit permite que os usuários integrem seus próprios modelos cinemáticos ao Packages do framework.

4.2.5 Implementação das ferramentas nas Instituições Federais

Automação e robotização são eventos em constante expansão, dessa forma instituto de ensino e tecnologia devem ter recursos para preparar os alunos a estas novas necessidades e a futuras oportunidades no mercado de trabalho. As ferramentas apresentadas neste trabalho permitem que sejam utilizadas para este fim, porém existem vantagens, desvantagens, oportunidades e ameaças na sua implementação que serão discutidas nesta seção.

Primeiramente, as principais vantagens de se utilizar as ferramentas para simulação robótica de código aberto é que não se faz obrigatoriedade de compra de hardwares de robôs para desenvolver soluções, tendo em vista que estas tecnologias são caras para a realidade dos Institutos Federais em geral. Outra vantagem é que ao utilizar um ambiente virtual o aprendizado

pode acontecer de forma mais segura e com menos restrições, pois o desenvolvimento, testes e implementações podem ser executados sem receio de danificar equipamentos caros ou causar acidentes durante a operação. Outra vantagem é que, caso os institutos já tenham computadores em seus laboratórios, a utilização das ferramentas não apresenta custos operacionais adicionais, já que são de código aberto.

Segundo estas vantagens criam algumas oportunidades. A primeira delas é permitir que os alunos consigam desenvolver projetos e produções cientificas para a instituição se tornar mais reconhecida e consequentemente captar mais recursos. Também permite que os alunos consigam desenvolver soluções para empresas sem que necessariamente precisem se deslocar para fora das suas instituições, já que as soluções podem ser implementadas remotamente. Em relação ao ensino, os educadores podem utilizar as ferramentas para desenvolver projetos com o instituto de aplicar o ensino ativo como a metodologia PBL (Problem-Based-Learning) sugerem, já que a robótica é um campo multidisciplinar, repleto de desafios de diversos níveis.

Em relação as desvantagens, deve ser destacado que a ferramenta exige uma curva de aprendizagem acelerada, logo porque deve-se ter conhecimento inicial de sistemas de computadores, linguagem de programação como Python e C++ e a própria ferramenta ter um escopo vasto sobre suas funcionalidades. Outra desvantagem é que o aprendizado da ferramenta exige que os alunos tenham familiaridade com a língua inglesa pois a maioria das documentações não estão disponíveis para o português. Por fim, pode ser demasiadamente difícil a aplicação de projetos robóticos com estas ferramentas em apenas um semestre acadêmico, devido ao escopo de trabalho.

5 CONCLUSÃO

O presente trabalho apresentou as ferramentas ROS, Moveit e Gazebo de código aberto aplicadas ao contexto da simulação robótica, mais especificamente a robótica de manipuladores industriais. Em adição, foi feito algumas análises de trajetórias comuns ao escopo de trabalho do manipulador ABB IRB 1200 5/9, para isso foi necessário descrever e compreender os aspectos das ciências Cinemática, Dinâmica, Controle e Planejamento de trajetórias de manipuladores industriais, para então avaliar a implementação das ferramentas na realidade dos Instituto Federais com objetivo de utilizar no ensino, pesquisa e desenvolvimento de tecnologias.

Foi observado que, primeiramente, ao utilizar o framework Moveit se tornou simples a configuração da cadeia cinemática, a visualização da trajetória e implementação do modelo simulado com o simulador Gazebo. Este simulador permitiu visualizar como o robô executaria as trajetórias em um ambiente físico simulado. Isto permitiu fazer correções nos algoritmos de comando das trajetórias.

Por fim, e mais importante, o midware ROS se mostrou de grande utilidade para o desenvolvimento de softwares voltados para robótica, pois com sua arquitetura modular, foi possível reutilizar aplicações de terceiros com certa facilidade, como foi o exemplo da utilização do repositório GIT ros_industrial.

Já para aplicação delas nas instituições de ensino foi demonstrado as principais vantagens, desvantagens, oportunidades e ameaças. De forma que as ferramentas se mostraram de grande valia para o desenvolvimento de pesquisas dentro do contexto dos institutos pois sua instalação e utilização são gratuitas, além de que as soluções feitas com elas podem ser implementadas no contexto real e prático, implicando na execução de projetos que podem gerar reconhecimento para as instituições e oportunidade de inserção no mercado de trabalho dos alunos. Foi observado também que algumas dificuldades devem ser contornadas para aumentar a eficiência de sua utilização, como por exemplo, deve-se atentar que é necessário capacitação em informática e metodologias ativas de ensino para professores e demais profissionais envolvidos, com isso facilitar o aprendizado dos alunos.

Dessa forma o presente trabalho contribui na implementação destes recursos ao ensino e desenvolvimento de tecnologia pois mostrou os principais conceitos e referências aplicados ao planejamento de trajetórias utilizando robôs manipuladores industriais. Também contribui com uma revisão das principais vantagens e desvantagens ao utilizá-las, e em adição contribui com exemplos de desafios que podem ser utilizados nas salas de aula para ao ensino de disciplinas, como por exemplo, Cinemática e Dinâmica em cursos como Engenharia Mecânica.

5.1 TRABALHOS FUTUROS

Diante do que foi apresentado referente a simulação robótica de manipuladores em softwares de código aberto, diversas linhas de pesquisa podem ser executadas, uma delas é a utilização de metodologias de ensino ativo aplicadas ao curso de Engenharia Mecânica do Instituto Federal do Piauí.

Como exemplo destas metodologias ativas, se tem o Aprendizado Baseado em Problemas, do inglês *Problem-based Learning* (PBL). Se trata de uma abordagem centrada no aluno, em que ele é exposto a desafios estruturados, de forma que tenha que desenvolver outros aspectos de sua inteligência para resolver problemas reais e relevantes (ACHAPPA et al., 2020). A metodologia foi originada no campo médico, porém é utilizada em diversas outras áreas, e com bastante frequência no ensino de Engenharia (HARA et al., 2020). E diante da presente Indústria 4.0 se faz necessário que cientistas e engenheiros estejam expostos a treinamento eficiente em cursos avançados como Automação industrial e Robótica (HABIB, 2020),(HARA et al., 2020),(CALVO et al., 2017).

Dessa forma, a análise da performance da metodologia aplicada ao ensino de disciplinas relacionadas a Indústria 4.0 utilizando softwares de código aberto se vê como uma oportunidade para pesquisas futura (CORRELL; WING; COLEMAN, 2012). Em que o presente trabalho pode servir como base para desenvolver os desafios relevantes que o PBL requer.

REFERÊNCIAS

ACHAPPA, S. et al. Implementation of project-based-learning (pbl) approach for bioinformatics laboratory course. *Journal of Engineering Education Transformations*, v. 33, p. 247–252, 2020. Citado na página 69.

AKSU, M.; MICHALOSKI, J. L.; PROCTOR, F. M. Virtual experimental investigation for industrial robotics in gazebo environment. In: AMERICAN SOCIETY OF MECHANICAL ENGINEERS DIGITAL COLLECTION. *ASME 2018 International Mechanical Engineering Congress and Exposition*. [S.l.], 2018. Citado na página 15.

BUSS, S. R. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, v. 17, n. 1-19, p. 16, 2004. Citado 2 vezes nas páginas 35 e 36.

CALVO, I. et al. A multidisciplinary pbl approach for teaching industrial informatics and robotics in engineering. *IEEE Transactions on Education*, IEEE, v. 61, n. 1, p. 21–28, 2017. Citado 2 vezes nas páginas 16 e 69.

CASAN, G. A. et al. Ros-based online robot programming for remote education and training. In: IEEE. *2015 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.], 2015. p. 6101–6106. Citado na página 16.

CORRELL, N.; WING, R.; COLEMAN, D. A one-year introductory robotics curriculum for computer science upperclassmen. *IEEE Transactions on Education*, IEEE, v. 56, n. 1, p. 54–60, 2012. Citado na página 69.

CRAIG, J. J. *Introduction to robotics: mechanics and control, 3/E*. [S.l.]: Pearson Education India, 2009. Citado 9 vezes nas páginas 19, 20, 30, 35, 36, 37, 41, 45 e 46.

DENG, H.; XIONG, J.; XIA, Z. Mobile manipulation task simulation using ros with moveit. In: IEEE. *2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*. [S.l.], 2017. p. 612–616. Citado 2 vezes nas páginas 23 e 24.

FEDOR, M. Application of inverse kinematics for skeleton manipulation in real-time. In: *Proceedings of the 19th spring conference on Computer graphics*. [S.l.: s.n.], 2003. p. 203–212. Citado na página 36.

FERNANDES, G. *Exploração de ambientes não estruturados através de manipulador robótico implementando controlador de impedância com parâmetros variáveis*. Tese (Doutorado) — Universidade de São Paulo, 2013. Citado 2 vezes nas páginas 31 e 32.

FERNANDEZ, T.; LYNCH, K.; WENG, H. Modern Robotics: Mechanics, Planning, and Control [Bookshelf]. *IEEE Control Systems Magazine*, v. 39, n. 6, 2019. Citado 12 vezes nas páginas 31, 32, 33, 34, 35, 36, 37, 38, 41, 45, 47 e 73.

FERREIRA, N. M. F. *Simulaçao dinâmica e controlo de robôs industriais*. Tese (Doutorado) — PhD thesis, Dissertaçao de mestrado em Engenharia Electrotécnica e de . . . , 1999. Citado na página 31.

GÉRADIN, M.; DUYSINX, P. An introduction to robotics: mechanical aspects. 2004. Citado 5 vezes nas páginas 18, 19, 20, 21 e 22.

GROOVER, e. a. m. p. Robótica, tecnologia e programação. McGraw-Hill, São Paulo, 1988. Citado 2 vezes nas páginas 21 e 22.

GUNAL, M. M. Simulation and the fourth industrial revolution. In: *Simulation for Industry 4.0*. [S.l.]: Springer, 2019. p. 1–17. Citado na página 15.

HABIB, S. A. Training undergraduate engineering students in robotics and automation through model-based design training: A case study at assumption university of thailand. *International Journal of Educational and Pedagogical Sciences*, v. 14, n. 2, p. 122–126, 2020. Citado 2 vezes nas páginas 16 e 69.

HARA, S. et al. Pbl program producing flying robot in mechanical and aerospace engineering department. In: EDP SCIENCES. *MATEC Web of Conferences*. [S.l.], 2020. v. 306, p. 05003. Citado 2 vezes nas páginas 16 e 69.

HISAZUMI, K. et al. An interdisciplinary and university pbl curriculum using robot challenge. In: IEEE. *2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. [S.l.], 2018. p. 308–315. Citado na página 16.

HOGAN, N.; BUERGER, S. Robotics and automation handbook. *Chapter*, v. 19, p. 19–1, 2004. Citado na página 19.

HUANG, Z.; LI, F.; XU, L. Modeling and simulation of 6 dof robotic arm based on gazebo. In: IEEE. *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*. [S.l.], 2020. p. 319–323. Citado 2 vezes nas páginas 27 e 29.

JUNG, H. et al. Implementation of a unified simulation for robot arm control with object detection based on ros and gazebo. In: IEEE. *2020 17th International Conference on Ubiquitous Robots (UR)*. [S.l.], 2020. p. 368–372. Citado na página 15.

KOUBÂA, A. et al. *Robot Operating System (ROS)*. [S.l.]: Springer, 2017. v. 1. Citado 2 vezes nas páginas 25 e 26.

MOVEIT! *MoveIt! Tutorials*. 2020. Moveit! Disponível em: <http://docs.ros.org/en/kinetic/api/moveit_tutorials/html/index.html/>. Acesso em: 1 dez. 2020. Citado 3 vezes nas páginas 25, 26 e 27.

NORTON, R. L. *Cinemática e dinâmica dos mecanismos*. [S.l.]: AMGH Editora, 2010. Citado na página 21.

ÖZCAN, R. The rise of robots! effects on employment and income. *Öneri Dergisi*, v. 14, n. 51, p. 1–17, 2019. Citado na página 15.

QUIGLEY, M. et al. Ros: an open-source robot operating system. In: KOBE, JAPAN. *ICRA workshop on open source software*. [S.l.], 2009. v. 3, n. 3.2, p. 5. Citado 2 vezes nas páginas 24 e 25.

RICHARDS, N. M.; SMART, W. D. How should the law think about robots? In: *Robot law*. [S.l.]: Edward Elgar Publishing, 2016. Citado na página 19.

ROBOTICS, I. F. of. *Executive summary World Robotics 2017 industrial robots*. 2017. Citado na página 15.

SPONG, M. W. et al. *Robot modeling and control*. [S.l.: s.n.], 2006. Citado 2 vezes nas páginas 18 e 41.

SPONG, M. W.; VIDYASAGAR, M. *Robot dynamics and control*. [S.l.]: John Wiley & Sons, 2008. Citado na página 46.

TORRES-TORRITI, M.; ARREDONDO, T.; CASTILLO-PIZARRO, P. Survey and comparative study of free simulation software for mobile robots. *Robotica*, Cambridge University Press, v. 34, n. 4, p. 791, 2016. Citado na página 23.

XU, X.; DUGULEANA, M. Trajectory planning of 7-degree-of-freedom manipulator based on ros. In: IOP PUBLISHING. *IOP Conference Series: Materials Science and Engineering*. [S.l.], 2019. v. 677, n. 5, p. 052072. Citado na página 30.

ANEXO A – CINEMÁTICA INVERSA

Este algoritmo foi obtido do autor (FERNANDEZ; LYNCH; WENG, 2019), para descrever a cinemática inversa de manipuladores robóticos. A rotina foi desenvolvida em MatLab e utilizou-se o método de Newton-Raphson

```
function [thetalist, success] ...
         = IKinSpace(Slist, M, T, thetalist0, eomg, ev)


% *** CHAPTER 6: INVERSE KINEMATICS ***
% Entradas:
%       Slist: Matriz contendo o helicoide de cada uma
%          das juntas como na metodologia
%       M: Configuração na posição inicial do Atuador Final,
%       T: Matriz de transformação da posição desejada,
%          thetalist0: Estimativa inicial para as juntas
%       eomg: Uma pequena tolerancia para a orientação do
%          Atuador final.
%       ev: Uma pequena tolerancia para a posição do Atuador
%          Final.
% Saida:
%        thetalist: Anglos das juntas para as condições
%        anteriores,
% Functions:
%        pinv: é uma função nativa ao Matlab que
%              calcula a pseudoinversa de uma matrix.
%        FKinSpace: função que calcula a cinemática direta.
%        Adjoint: Função para a operação de comutação de
%        vetores ou Lie Bracket

thetalist = thetalist0;
i = 0;
maxiterations = 20;
Tsb = FKinSpace(M, Slist, thetalist);
Vs = Adjoint(Tsb) * se3ToVec(MatrixLog6(TransInv(Tsb) * T));
```

```
err = norm(Vs(1: 3)) > eomg || norm(Vs(4: 6)) > ev;

while err && i < maxiterations

    thetalist = thetalist + ...
    pinv(JacobianSpace(Slist, thetalist)) * Vs;
    i = i + 1;
    Tsb = FKinSpace(M, Slist, thetalist);
    Vs = Adjoint(Tsb) * se3ToVec(MatrixLog6(TransInv(Tsb) * T));
    err = norm(Vs(1: 3)) > eomg || norm(Vs(4: 6)) > ev;
end
success = ~ err;
end
```

ANEXO B – TCC_PLANING_TRAJECTORY_MOVEIT

```
<launch>
  <!-- Este arquivo do tipo launch foi baseado no arquivo padrão
  gerado pelo Moveit e adaptado para os objetivos deste trabalho -->

  <!-- Carrega os nomes das juntas -->
  <rosparam command="load" file="$(find abb_irb1200_support)/config/joint_names_irb1200_5_90.yaml" />

    <!-- Carrega a descrição do robô antes dos Nós serem carregados -->
  <include file="$(find abb_irb1200_5_90_moveit_config)/launch/planning_context.launch" >
    <arg name="load_robot_description" value="true" />
  </include>

  <!-- Carrega o Nó joint_state_publisher, tem como objetivo sincronizar os comandos
  no move_group com a simulação no gazebo -->
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
    <param name="/use_gui" value="false"/>
    <rosparam param="/source_list">[/joint_states]</rosparam>
  </node>

  <!-- Tem como objetivo carregar o Move_group -->
  <include file="$(find abb_irb1200_5_90_moveit_config)/launch/move_group.launch">
    <arg name="publish_monitored_planning_scene" value="true" />
  </include>

  <!-- Tem como objetivo carregar o Move_group -->
  <include file="$(find abb_irb1200_5_90_moveit_config)/launch/moveit_rviz.launch">
    <arg name="config" value="true"/>
  </include>

  <!-- Publica o robot state (tf transforms) -->
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" />
</launch>
```