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

PAULO VICTOR DUARTE

SIMULAÇÃO DE ROBÔS ANTROPOMÓRFICOS COM MOVEIT

Joinville
2020

PAULO VICTOR DUARTE

SIMULAÇÃO DE ROBÔS ANTROPOMÓRFICOS COM MOVEIT

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

Orientador: Prof. Dr. Roberto Simoni

Coorientador: Prof. Dr. Andrea Piga Carboni

Joinville
2020

RESUMO

Este trabalho aborda a simulação de planejamento de trajetórias para robôs antropomórficos através dos *softwares* ROS e MoveIt. O trabalho inicia obtendo o modelo de um robô UR10 e uma tocha de solda através do *site* de modelos gratuitos GradCad. Em seguida, os modelos são transferidos para a plataforma CAD Onshape, onde é realizado a montagem das partes individuais do robô. O Onshape possui uma extensão capaz de gerar automaticamente o arquivo URDF para mapear a cadeia de elos e descrever todas as informações físicas do robô para o MoveIt. O trabalho utiliza técnicas e ferramentas avançadas para obter o TCP do robô e o modelo de colisão com polígonos simples. Através do assistente de configuração do MoveIt é gerado o pacote base para as simulações. Para construir o ambiente de simulação e visualizar trajetórias foi utilizado outro pacote do ROS chamado Rviz. Foram desenvolvidas cinco aplicações em C++ para movimentar o robô através do MoveIt. Ao final, foi implementado um módulo de interface de *hardware* para capturar o estado das juntas do robô no ambiente de simulação e mover cinco motores de passos para os respectivos valores.

Palavras-chave: ROS. MoveIt. Simulação.

ABSTRACT

This work presents the simulation of trajectory planning for anthropomorphic robots using ROS and MoveIt. The work starts by obtaining the model of a UR10 robot and a welding torch through the free model website GradCad. Moreover, the models are transferred to the Onshape CAD platform, where the individual robot's parts are assembled. The Onshape has an extension capable of automatically generating the URDF file to map the link chain and describe all the robot's physical information for MoveIt. The work uses advanced techniques and tools to obtain the robot's TCP and the collision model with simple polygons. The MoveIt setup assistant was used to generate the base package for the simulations. To build the simulation environment and visualize trajectories, another ROS package called Rviz was used. Five C++ applications were developed to move the robot through MoveIt. Finaly, a hardware interface module was implemented to capture the state of the robot's joints in the simulation environment and move five stepper motors to their respective values.
Keywords: ROS. MoveIt. Simulation.

LISTA DE ILUSTRAÇÕES

Figura 1 – Resultado da busca de trajetórias com RRT 13
Figura 2 – Five DOF Arm e Arduino 17
Figura 3 – sistema de referência de coordenadas. 19
Figura 4 – Interfaces de comunicação ROS 21
Figura 5 – Camadas do URDF 22
Figura 6 – Simulação com o uso do Matlab e o pacote Robotics Toolbox (PeterCorke). 23
Figura 7 – Arquitetura MoveIt. 25
Figura 8 – Assistente de configuração MoveIt. 28
Figura 9 – Simulação do Panda Arm no Rviz. 29
Figura 10 – Projeto modelado no Onshape 32
Figura 11 – OpenSCAD 33
Figura 12 – Aproximação por modelos geométricos primitivos 33
Figura 13 – Conversão de URDF para Graphiz. 34
Figura 14 – Visualização 3D do URDF. 35
Figura 15 – Grasping Frame 36
Figura 16 – Motion Planning. 37
Figura 17 – Moveit assistente. 37
Figura 18 – Resultado da aplicação GoRandom. 40
Figura 19 – Resultado da aplicação GoHome. 41
Figura 20 – Resultado da aplicação Type Moves. 41
Figura 21 – Resultado da aplicação Cartesian Reader. 42
Figura 22 – Resultado da aplicação Stop Move. 44
Figura 23 – Mapa gráfico da relação entre os *nodes* 44
Figura 24 – *Topic* Feedback. 45
Figura 25 – Painel com um *drive* e um motor. 46
Figura 26 – Painel com um *drive* e um motor. 54

LISTA DE TABELAS

Tabela 1 – Sistema de comunicação ROS 20
Tabela 2 – Parâmetros de Denavit Hartenberg 23

LISTA DE SIGLAS

ACM Matriz de colisões permitidas

AUV Autonomous underwater vehicle

DH Denavit Hartenberg

DOF Degrees Of Freedom

FCL FLexible Collision Library

GUI Interface gráfica do Usuário

KDL Kinematics and Dynamics Library

LaSiN Laboratório de Simulação Naval

OMPL Open Motion Planning Library

ROS Robot Operating System

RPY Roll Pitch Yall

RRT Rapidly-exploring Random Trees

SRDF Semantic Robot Description Format

TCP Tool Center Point

URDF Unified Robot Description Format

SUMÁRIO

1	**INTRODUÇÃO**	**9**
1.1	**Objetivo Geral**	**10**
1.2	**Objetivos Específicos**	**10**
1.3	**Metodologia de desenvolvimento do trabalho**	**10**
1.3.1	Estudo do ROS e MoveIt	10
1.3.2	Estratégia de desenvolvimento do projeto com o MoveIt	11
1.3.3	Lista de programas utilizados	11
1.4	**Estrutura do texto**	**12**
2	**REVISÃO BIBLIOGRÁFICA**	**13**
2.1	**Planejamento de Trajetórias com MoveIt**	**13**
2.2	**Manipulador de 3-DOF da Universidade de Veracruzana**	**14**
2.3	**ROSRemote**	**14**
2.4	**Controle do I-AUV Girona 500 com MoveIt**	**15**
2.5	**Controle para dois dedos robóticos independentes**	**15**
2.6	**Manipulador 5-DOF**	**16**
3	**FUNDAMENTAÇÃO TEÓRICA**	**18**
3.1	**Cinemática direta e inversa**	**18**
3.2	**ROS - Robot Operating System**	**19**
3.2.1	Nomenclatura	20
3.2.2	Roscore	20
3.3	**URDF - Formato Unificado de Descrição de um robô**	**21**
3.4	**Parâmetros de Denavit Hartenberg**	**22**
3.5	**Arquitetura e planejamento de trajetórias com o MoveIt**	**24**
3.5.1	Solucionadores de cinemática inversa	25
3.5.2	Cenário de colisão	26
3.6	**Criação de um novo projeto MoveIt**	**26**
3.6.1	URDF	26
3.6.2	SRDF	27
3.7	**Parametrização do Tempo**	**28**
3.8	**Visualização com Rviz**	**29**
3.9	**Ferramentas para exportar URDFs**	**30**
4	**PROGRAMAÇÃO DO PACOTE PARA O ROBÔ ANTROPOMÓRFICO**	**31**
4.1	**Construção do modelo 3D**	**31**
4.2	**Modelo de colisão**	**32**

4.3 Verificar hierarquia do URDF 33
4.4 Preview do arquivo URDF 34
4.5 Ponto central de atuação 35
4.6 Motion Planning 36
4.7 Criação de pacotes MoveIt Setup Assistant 37
4.8 Compilar aplicações 38

5 SIMULAÇÕES COM O PACOTE DO ROBOT ANTROPOMORFICO 39
5.1 Aplicação GoRandom 39
5.2 Aplicação GoHome 40
5.3 Aplicação TypeMoves 41
5.4 Aplicação CartesianReader 42
5.5 Aplicação StopMove 43
5.6 ROS GUI 44
5.7 Captura de pontos pelo simulador 45
5.8 Integrar software e hardware 45

6 CONCLUSÕES 47
6.1 Considerações finais 47
6.2 Trabalhos futuros 48

REFERÊNCIAS 49

APÊNDICE A 51

APÊNDICE B 52

APÊNDICE C 53

1 INTRODUÇÃO

A utilização de robôs manipuladores tem ganhado destaque nas últimas décadas, expandindo a aplicação para diversas áreas de estudo. Em especial, os robôs colaborativos lideram as pesquisas, compartilhando tarefas no mesmo espaço que seres humanos nas linhas de produção ou mesmo robôs domésticos que limpam a casa em ambientes hostis (Chitta; Sucan; Cousins, 2012).

Segundo Quigley et al. (2009), o processo criativo dos diferentes robôs era até então complexo e exaustivo. Entre os vastos problemas enfrentados destacam-se:

- Implementação de um novo *software* para cada aplicação;
- Diferentes *hardwares*, inviabilizando a reutilização de códigos;
- Limitação de uma única linguagem de programação;
- A dimensão do código pode atrapalhar o programador e a produção em equipe;
- A amplitude necessária de conhecimento está muito além das capacidades de qualquer pesquisador.

Para enfrentar esses desafios foi desenvolvido o ROS. Fundado em 2007, o ROS é uma coleção de estruturas de *software* com foco no desenvolvimento da robótica. Seu funcionamento se assemelha a um sistema operacional em recursos como abstração de *hardware*, controle de dispositivo de baixo nível, troca de mensagens e gerenciamento de pacotes (Deng; Xiong; Xia, 2017). Contudo, o fato de ser *open-Source* acelerou seu processo expansivo.

O MoveIt é um pacote adicional ao ROS, dedicado para o desenvolvimento de projetos com robôs antropomórficos de cadeia aberta. Após ser hospedado à uma plataforma *online* e de código aberto, o MoveIt teve a oportunidade de ser integrado com outras ferramentas que o tornaram robusto no planejamento de trajetória, simulação, controle de espaço e colisão, calculo de cinemática inversa, etc (Chitta; Sucan; Cousins, 2012).

Inspirado pela vasta diversidade de aplicações que o ROS proporciona, o grupo de robótica da UFSC de Joinville vem desenvolvendo seus projetos com essa ferramenta em simulações e controle de robôs. Através da contribuição entre alunos e professores, surge a oportunidade de desenvolver o projeto de simulação para um robô antropomórfico, com o uso da extensão MoveIt como piloto para o controle de trajetórias.

O trabalho também contribui para a comunidade acadêmica, apresentando técnicas e ferramentas avançadas para acelerar o desenvolvimento de projetos de robôs antrompomórficos.

1.1 Objetivo Geral

O objetivo geral deste trabalho é simular a movimentação de um robô antropomórfico com o MoveIt.

1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

a. Importar o modelo 3D do robô para a plataforma Onshape;
b. Exportar o arquivo URDF do robô através do *plug-in* onshape-to-robot;
c. Gerar o pacote de operação com o MoveIt Setup Assistant;
d. Desenvolver cinco aplicações em C++ para movimentar o robô através do MoveIt;
e. Conectar as aplicações à um módulo de interface de *hardware* para capturar o estado das juntas do simulador e mover cinco motores de passos.

1.3 Metodologia de desenvolvimento do trabalho

Esta sessão entrará em detalhes sobre a metodologia de desenvolvimento para realizar o experimento de simulação e aplicação do sistema proposto. Por se tratar de um experimento virtual, a metodologia é classificada como pesquisa experimental. Para alcançar os objetivos, foi necessário investir um tempo para compreender as ferramentas do ROS (*Robot Operating System*) e do MoveIt. A barreira de entrada para esses *softwares* requerem conhecimentos avançados de informática, programação em alto e baixo nível, eletrônica, robótica e familiaridade com sistemas operacionais Linux.

1.3.1 Estudo do ROS e MoveIt

Mesmo sendo ferramentas de código aberto, e apresentando inúmeros tutoriais na internet, o ROS e o MoveIt não são *softwares* para o usuário comum, e seu publico alvo é voltado para pesquisas e empresas. Para compreender alguns conceitos é requerido familiaridade com a linguagem de programação em C++, Python e XML, além de conceitos de robótica como rotação e translação de eixo, referência de base, ponto e TCP (*Tool Center Point*), etc.

O MoveIt é apenas um pacote adicional ao portfólio do ROS, o que justifica a necessidade de iniciar os estudos através de tutoriais do ROS. A base de estudo dessa ferramenta foram os tutorias *online* disponíveis no site oficial do ROS (2020). O livro de Joseph e Cacace (2018) complementou esse estudo, porém durante a pesquisa alguns conceitos apresentados pelo livro se tornaram obsoletos.

Os tutoriais do MoveIt são simples, rápidos e objetivos. A principal fonte de estudo se encontra no site oficial do MoveIt (2020d). O MoveIt Setup Assistant é sua principal interface de desenvolvimento. Através dela que foram geradas as principais

configurações do robô. Outros artigos científicos apresentados no capítulo 3 serviram para aprofundar os estudos de algumas ferramentas avançadas.

1.3.2 Estratégia de desenvolvimento do projeto com o MoveIt

Alcançados os objetivos de estudo deste trabalho, o próximo passo é gerar o pacote de operação do MoveIt. Aqui é proposto importar um modelo 3D de um robô para o Onshape. Através do *plug-in* onshape-to-robot foi obtido o modelo de descrição do robô para o formato URDF (Unified Robot Description Format).

Com o URDF em mãos, foi seguido os passos do MoveIt Setup Assistant conforme o estudo dos tutoriais. Ao final, teremos todas as ferramentas prontas para receber comandos das aplicações em C++.

Para atingir os objetivos do trabalho foram desenvolvidos cinco aplicações:

- Aplicação GoRandom;
- Aplicação GoHome;
- Aplicação TypeMoves;
- Aplicação CartesianReader;
- Aplicação StopMove.

As aplicações foram testadas e validadas no ambiente de simulação Rviz. Como último passo, resta ainda unir o pacote MoveIt ao *hardware*.

Para conectar as aplicações ao módulo da interface de *hardware* foi utilizado o código da Iniciação Científica de Carvalho (2020). Em seu IC, Priscila construiu um pacote ROS capaz de se comunicar com seis motores de passos. Como o ROS é modular, o *node* de *hardware* deve operar de maneira semelhante, ao substituir o robô utilizado na pesquisa pelo robô deste projeto.

1.3.3 Lista de programas utilizados

A construção deste projeto segue um roteiro linear, onde foram utilizados diferentes programas em pontos-chave para a evolução do trabalho. A lista dos principais programas e sua utilização são apresentadas a seguir:

1. GradCad: Usado para baixar o modelo 3D do robô;
2. Onshape: Plataforma CAD para a modelagem e montagem do robô;
3. Onshape-to-robot: Extensão para gerar o modelo de descrição robô em URDF;
4. OpenSCAD: Utilizado para projetar o modelo de colisão do robô;
5. MoveIt Setup Assistant: Assistente para gerar o pacote de operação do MoveIt;
6. Rviz: Interface para visualizar trajetórias e simulações.

1.4 Estrutura do texto

Esse trabalho é composto por cinco capítulos principais. No primeiro deles, é introduzido a revisão bibliográfica, que expõe alguns projetos que utilizam o ROS como piloto de seus robôs, validando objetivos do presente estudo, e justificando as motivações de desenvolvimento do mesmo.

O segundo capítulo apresenta a fundamentação teórica desse trabalho, em sua maioria formado pelo portfólio bibliográfico da comunidade *open-source* do site oficial ROS e MoveIt, e complementado por referências externas relevantes para o tema.

O terceiro capítulo apresenta o pacote MoveIt. Nesse capítulo é discutido o procedimento para exportar o modelo 3D do robô antropomórfico do Onshape para a linguagem de descrição URDF, e em seguida é apresentado ferramentas que auxiliam na construção do modelo de colisão. Para finalizar, através do MoveIt Setup Assistant, é então construído o pacote básico para o funcionamento dos *nodes* Moveit com o ROS.

O quarto capítulo apresenta a simulação de cinco aplicações C++ para movimentar o robô através do MoveIt. Ao final deste capítulo há também sugestões de ferramentas que facilitam a captura de pontos e a compreensão da interação entre os *nodes* através do pacote gráfico rqt.

Finalmente, o quinto capítulo conclui o estudo dividindo em duas partes principais: Considerações finais e Trabalhos futuros. O primeiro descreve como esse trabalho pode contribuir para o meio acadêmico e destaca as dificuldades enfrentadas. O Segundo, levanta requisitos do projeto que não tiveram espaço para seu desenvolvimento e aborda os pontos que devem ser considerados em trabalhos futuros. Em seguida, são apresentadas as referências utilizadas no desenvolvimento do trabalho.

2 REVISÃO BIBLIOGRÁFICA

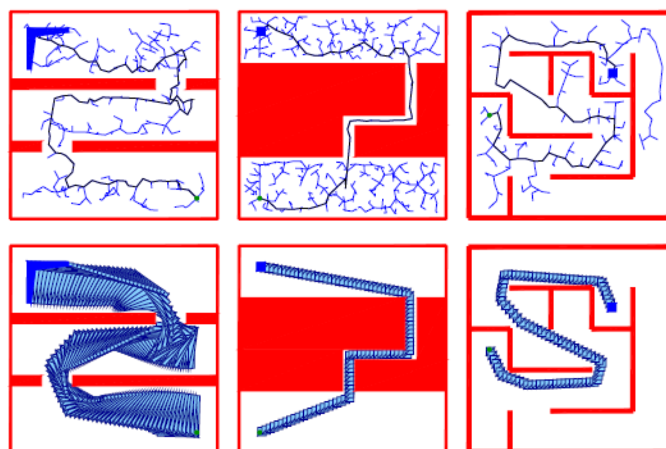
Considerando os objetivos deste trabalho, será apresentado neste capítulo, uma contextualização de projetos que utilizaram o ROS e MoveIt para operar seus robôs.

2.1 Planejamento de Trajetórias com MoveIt

Quando determinamos um ponto de destino para o robô, é necessário calcular um caminho otimizado, rápido e livre de colisões até o ponto desejado. Para traçar essa rota o MoveIt utiliza de algoritmos de busca para encontrar um caminho eficiente.

Podemos pensar nesses algoritmos como um rato preso em um labirinto, e as paredes são os obstáculos que o robô deve evitar para não colidir. A Fig. 1 é uma representação gráfica de como esses algoritmos trabalham para encontrar uma rota, as imagens superiores ilustram a árvore de busca e as inferiores o caminho encontrado.

Figura 1 – Resultado da busca de trajetórias com RRT



Fonte: Kuffner e LaValle (2000).

Existem sistemas complexos e otimizados para o planejamento de trajetórias. Contudo, requerem muito tempo de recursos e processamento. Assim, para atender a uma demanda mais aplicável, as custas da otimização, algoritmos randômicos propõe uma solução rápida e satisfatória.

O RRT (Rapidly-exploring Random Trees) é um dos algoritmos de busca randômica disponíveis na biblioteca OMPL (Open Motion Planning Library). Ele funciona cultivando duas árvores binárias enraizadas uma no início e outra no objetivo. As árvores exploram o espaço ao seu redor progredindo uma para a outra usando um método simples, porém eficiente, até convergirem, conferindo uma solução válida (Kuffner; LaValle, 2000).

O MoveIt! está integrado à biblioteca OMPL[1], que consiste em uma coleção de algoritmos de busca heurísticos para planejar movimentos baseados em amostragem (Kavraki Lab, 2012).

2.2 Manipulador de 3-DOF da Universidade de Veracruzana

O projeto da equipe Hernandez-Mendez et al. (2017a) é uma boa base de estudo para um projeto integrado. Nele são detalhados os passos necessários para controlar um braço robótico de três graus de liberdade, integrando vários módulos ROS, assim como o MoveIt. O braço foi modelado no *software* gratuito FreeCAD, e os arquivos foram usados para imprimir o robô físico, e também para gerar o formato de descrição URDF. Controladores de posição e força foram usado na garra para manipular os objetos. Neste sistema, o robô ainda possui sensores de posição e velocidade para realimentar a malha de controle.

O trabalho tinha como objetivo manipular e segurar objetos para transportá-los de um local à outro, ou em outras palavras, um simples *pick and place*. Para executar esta tarefa, um planejador RRT lida com restrições de colisão entre os elos, juntas e objetos, expostos em um ambiente virtual, provando mais uma vez a eficiência desse algoritmo de busca.

2.3 ROSRemote

O maior beneficio do ROS é ser modular, ou seja, é possível utilizar somente o necessário para a aplicação, e em caso de falha, outros módulos podem continuar trabalhando de forma independente. Além disso, sua comunidade grande e colaborativa, ajuda a compartilhar soluções e propagar brilhantes ideias à qualquer projeto.

Em sua dissertação, Pereira (2018) comenta sobre a versatilidade em desenvolver novas ferramentas para o ROS, e também comenta sobre as vantagens de escolher o ROS como piloto de projetos robóticos. Em seu trabalho, foi desenvolvido a proposta de um *framework* que permite manipular dados remotamente, como a resposta de sensores ou controlar atuadores de um robô trocando informações pela nuvem.

A ideia de operar o ROS de forma remota é interessante para projetos que pretendem evitar cabos longos e pesados que poderiam comprometer a mobilidade do robô.

[1] O OMPL é a principal biblioteca de planejamento de trajetórias do MoveIt.

2.4 Controle do I-AUV Girona 500 com MoveIt

Youakim et al. (2017) descreve em sua pesquisa uma aplicação prática do ROS e MoveIt como piloto do AUV(Autonomous underwater vehicle) Girona 500, projetado e desenvolvido pela Universidade de Girona para missões em até 500 metros abaixo da água. Neste veículo é acoplado um braço robótico 4-DOF (Degrees Of Freedom), que faz uso do RRT para calcular trajetórias. Ainda, são apresentadas as vantagens e desvantagens encontradas no decorrer da implementação com o ROS e MoveIt. Entre os pontos levantados destaca-se de forma positiva os seguintes:

- Interface amigável para a definição da cinemática através de juntas e elos;
- Descomplicada capacidade de planejar caminhos livres de colisão combinando algoritmos OMPL e a geometria de obstáculos;
- Tempo de desenvolvimento reduzido, ao permitir reutilizar códigos dos controladores disponíveis em projetos antecessores.

No entanto, durante o desenvolvimento, enfrentaram alguns desafios, sugeridos como tópicos em investigação futura.

- DOFs heterogêneos: Os DOFs do AUV usados para deslocamentos longos e movimento de baixa frequência são lentos e incertos. Enquanto os DOFs do manipulador que realizam movimentos curtos e de alta frequência, são rápidos e precisos. Seria interessante diferenciar esses DOFs no modelo, afim de promover o movimento do veículo para se aproximar quando o alvo está fora do alcance, e então movimentar o braço quando o destino está dentro de seu espaço de trabalho;
- DOFs passivos: A própria instabilidade do veículo em manter a posição pode causar perturbações, como a dos cilindros, que são DOFs passivos e sujeitos a variações de pressão e temperatura. Para contornar esse problema foi adotada uma solução de compromisso. Os DOFs passivos foram incluídos no modelo virtual com valor fixo no início de cada experimento. Ocasionando perda na precisão do efetuador final caso sofra significativa alteração. Assim, sendo necessária uma ação de replanejamento. As custas de um aumento na complexidade do sistema, é proposto o uso da hidrodinâmica para prever o comportamento do veículo. Todavia, o ROS ainda não possui suporte para esse estudo.

2.5 Controle para dois dedos robóticos independentes

Outra vantagem em desenvolver projetos modulares com o ROS é o caso do trabalho realizado por Hernandez-Mendez et al. (2017b). O artigo propõe um sistema com dois dedos independentes, equipados com sensores de posição e força.

O sistema lida com o problema de ajuste da posição dos dedos gerenciados de forma independente pela camada *middleware* do ROS, até que o objeto esteja na posição central da garra. Feito isso, o sistema aplica um controlador de força para cada um dos dedos. O sistema proposto usa dois controladores PID, um para controlar a posição do objeto, e o segundo para controlar a força aplicada. Os parâmetros PID foram obtidos pelo método da tentativa e erro até encontrar um resultado satisfatório.

Hernandez comprova a eficiência de se trabalhar em módulos separados com o ROS. O MoveIt consegue trabalhar lado a lado com outros *Nodes* do ROS, permitindo a execução paralela e simultânea do controle independente dos dedos de um manipulador enquanto o MoveIt gerencia os processos de trajetória e cinemática do braço antropomórfico.

2.6 Manipulador 5-DOF

Para validar alguns conceitos, foi desenvolvido um protótipo para fins educativos. O Five DOF Robot Arm é um manipulador de cinco graus de liberdade que foi modelado e construído baseado no manipulador Seven DOF Robot Manipulator de Joseph e Cacace (2018).

O objetivo do projeto era criar um pacote ROS com o auxílio das ferramentas do MoveIt, implementar uma interface entre *hardware* e o ROS, e por fim, desenvolver uma aplicação simples em C++ para mover o robô.

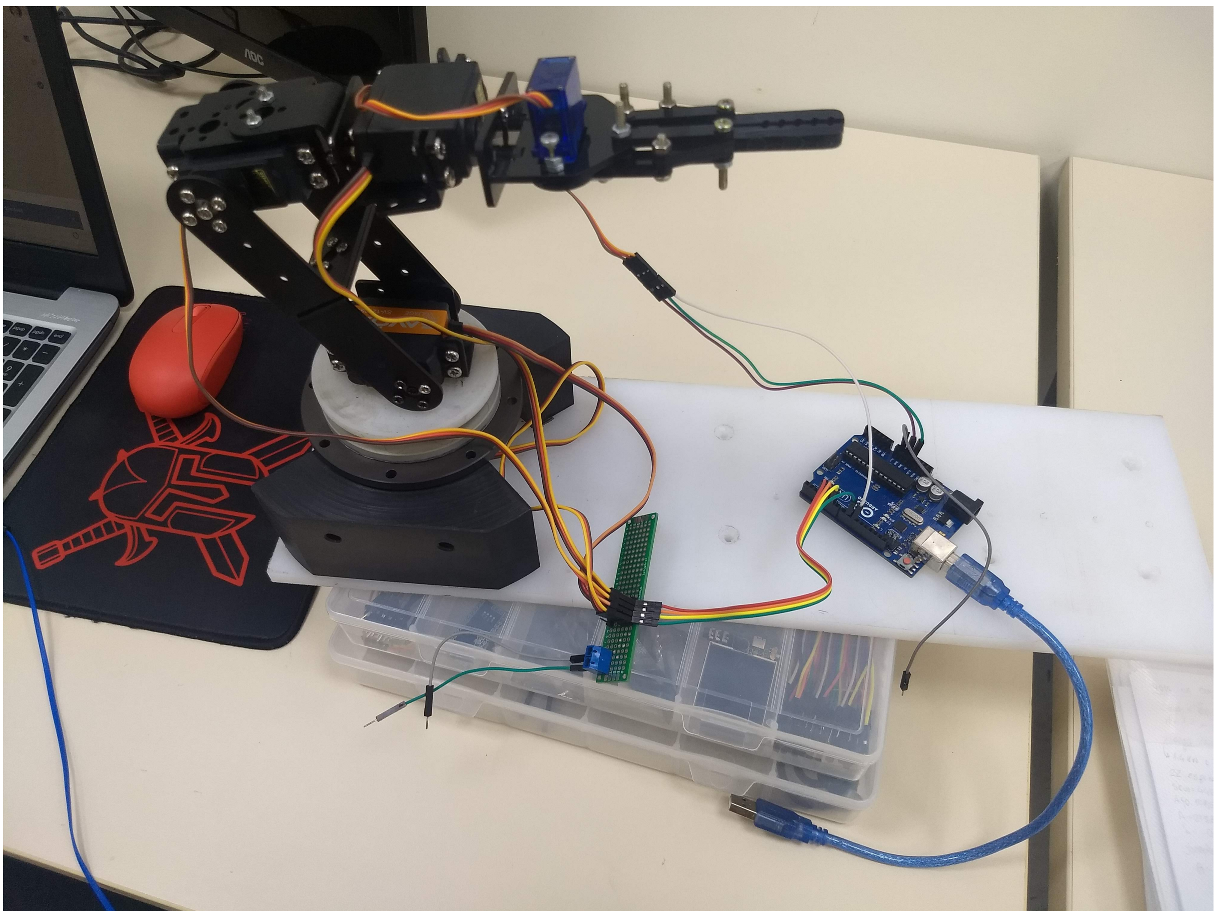
Um Arduino foi utilizado como interface de comunicação entre o computador e os motores. E uma placa foi projetada para alimentar individualmente cada motor com uma fonte externa de 6V. A Fig. 2 mostra esse circuito.

Neste projeto, é notável que o sistema ROS possui uma curva de aprendizagem lenta. Contudo, o esforço é recompensando no acelerado processo de reutilizar códigos em novos trabalhos. Transportar um programa que opera um robô para outro completamente diferente, se torna tão simples quanto trocar uma lâmpada. A estrutura permite separar o código que descreve as coordenadas do pick-and-place e a estrutura do robô. Assim, a mesma aplicação pode guiar uma infinidade de robôs, contanto que o ponto esteja dentro de um espaço de trabalho em comum.

Este projeto é apresentado em detalhes por Duarte (2019), onde também é possível explorar todo os diretórios do programa pelo endereço virtual hospedado no *site* "Github"[2].

[2] <https://github.com/paulovictor237/ROS-five-dof-arm>

Figura 2 – Five DOF Arm e Arduino



Fonte: Duarte (2019).

3 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão abordados os fundamentos teóricos necessários para o desenvolvimento deste trabalho.

3.1 Cinemática direta e inversa

No estudo da robótica, estamos sempre interessados na localização de objetos no espaço tridimensional. Esses objetos são os elos do manipulador, ferramentas anexadas ao robô, objetos manipulados, etc. Todavia, como afirma Craig (2013) tais objetos são descritos por apenas dois atributos: posição e orientação.

A fim de descrever a posição e a orientação de um corpo no espaço, sempre atrelaremos as coordenadas à um sistema de referência. Na robótica, podemos descrever a posição do manipulador de um robô em relação à sua origem através da cinemática direta e cinemática inversa.

Os robôs são compostor por uma cadeia de elos rígidos conectados por juntas rotativas ou prismáticas que por sua vez são denominados graus de liberdade ou DOFs. Essa cadeia de juntas são descritas por coordenadas lineares no caso de juntas prismáticas, e polares para o caso das juntas rotativas. Ao final da cadeia cinemática é anexado um efetuador final, como garras, pinças, ventosas, tochas de soldas e outras ferramentas. Na cinemática direta deseja-se obter a posição e orientação do efetuador final do robô no plano cartesiano a partir das coordenadas de juntas. A Fig. 3 mostra o sistema de coordenadas cartesianas e os ângulos de juntas para um robô de 4 graus de liberdade.

Segundo Sciavicco et al. (2010), uma das formas de calcular a equação que descreve a cadeia cinemática direta é através dos parâmetros de Denavit Hartenberg. O método consiste em descrever a posição e orientação de cada elo do robô em relação ao seu antecessor direto através de quatro parâmetros "d", "a", "α" e "θ". A sessão 3.4 apresenta detalhes relevantes dessa abordagem para o tema do trabalho em questão.

A cinemática inversa por sua vez, obtém os ângulos de junta a partir da posição e orientação do manipulador. Assim, uma equação matemática deve calcular todos os possíveis conjuntos de ângulos de junta para se obter a posição e orientação desejada. Contudo, o cálculo matemático para resolver o problema geométrico da cinemática inversa é complexo e ocorrem singularidades cinemáticas [1]. Laus (2012) cita três problemas a se considerar no cálculo da cinemática inversa, são eles:

- A existência de mais de um conjunto de solução;

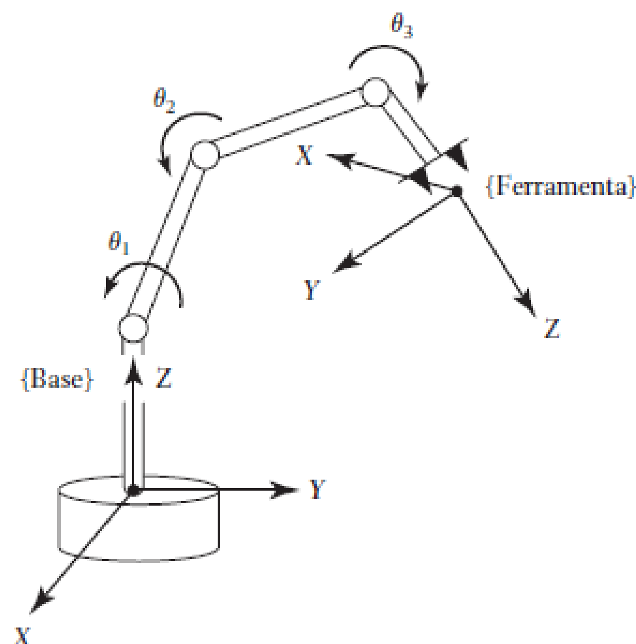
[1] Sistemas que possuem mais de uma solução de coordenadas de juntas para o mesmo ponto cartesiano

- A inexistência de qualquer solução (por falta de alcance ou destreza do robô);
- A contenção de soluções (por falta de graus de liberdade no manipulador).

A existência de mais de um conjunto de soluções normalmente é resolvido escolhendo uma entre as várias soluções. Entretanto, há casos onde podem coexistir infinitas soluções. A inexistência de qualquer solução pode ocorrer porque a posição e orientação está fora do espaço de trabalho do robô. [2]. Por fim, há manipuladores com menos de seis graus de liberdade que não podem atingir qualquer posição e orientação mesmo dentro de seu espaço de trabalho.

Vemos assim que enquanto determinar a posição de um robô através dos ângulos de juntas é uma tarefa relativamente fácil, por outro lado, o cálculo da cinemática inversa ainda é um desafio mesmo para os dias de hoje. Contudo, o MoveIt é integrado à vários pacotes de solucionadores da cinemática que abstraem esses cálculos para o usuário final.

Figura 3 – sistema de referência de coordenadas.



Fonte: Craig (2013)

3.2 ROS - Robot Operating System

O ROS foi desenvolvido inicialmente em 2007 pela Stanford Artificial Intelligence Laboratory e sua filosofia original era projetar um sistema de *software* flexível e dinâmico (Deng; Xiong; Xia, 2017). Atualmente, o ROS é executado apenas em plataformas Unix como o Ubuntu e Apple OS. Embora uma adaptação para o Microsoft Windows seja possível, ela ainda não foi totalmente explorada.

[2] Conjunto de posições cartesianas que o robô pode atingir

Normalmente, o ROS funciona como um *middleware*[3], fornecendo serviços padrão de um sistema operacional como abstração de *hardware*, controle de dispositivo de baixo nível, troca de mensagens entre processos e gerenciamento de pacotes.

3.2.1 Nomenclatura

Assim como os populares Torrents, o ROS usa a interface *peer-to-peer* como rede de comunicação entre os processos. Segundo Quigley et al. (2009), os processos fundamentais da estrutura do ROS são os *Nodes, Messeges, topics, Services* e *Actions*.

Nodes são os processos responsáveis pela execução do programa. Um sistema normalmente é composto por vários *Nodes* por exemplo, podemos pensar no projeto de um carro, um *node* fornece a informação dos sensores de distância, outro controla a direção, outro aciona as rodas do carro e assim por diante.

Messeges é o formato da informação trocada entre os *Nodes*. Uma *Messege* é uma estrutura de dados, que compreende vários campos de tipos primitivos como *integers, floats, booleans, strings*, etc.

Assim os *Nodes* podem trocar *Messeges* de três formas diferentes, através dos *topics, Services* e *Actions*. Esse formato permite que os *Nodes* sejam projetados individualmente e acoplados livremente no tempo de execução. Para facilitar o entendimento entre as diferença de cada um é apresentada a Tabela 1.

Tabela 1 – Sistema de comunicação ROS

Tipo	Fluxo dos Nodes	Sentido	Descrição
Topic	Publish $\rightarrow$ Subscribe	Unidirecional	Um Publish pode ter vários Subscribe
Service	Server $\rightarrow$ Client	Unidirecional com feedback	Um Server pode atender a apenas um Client
Action	Server $\leftrightarrow$ Client	Bidirecional	Um Server pode atender a apenas um Client

Fonte: Autor (2020).

3.2.2 Roscore

Segundo Walter (2019), o "roscore" como o próprio nome sugere, é o *core* (Núcleo) do sistema. É formado por uma coleção de *Nodes* e programas que são obrigatórios para um sistema baseado em ROS. O "roscore" deve sempre ser iniciado antes da execução de qualquer outra tarefa, pois é ele quem gerencia a troca de mensagens entre os *Nodes*.

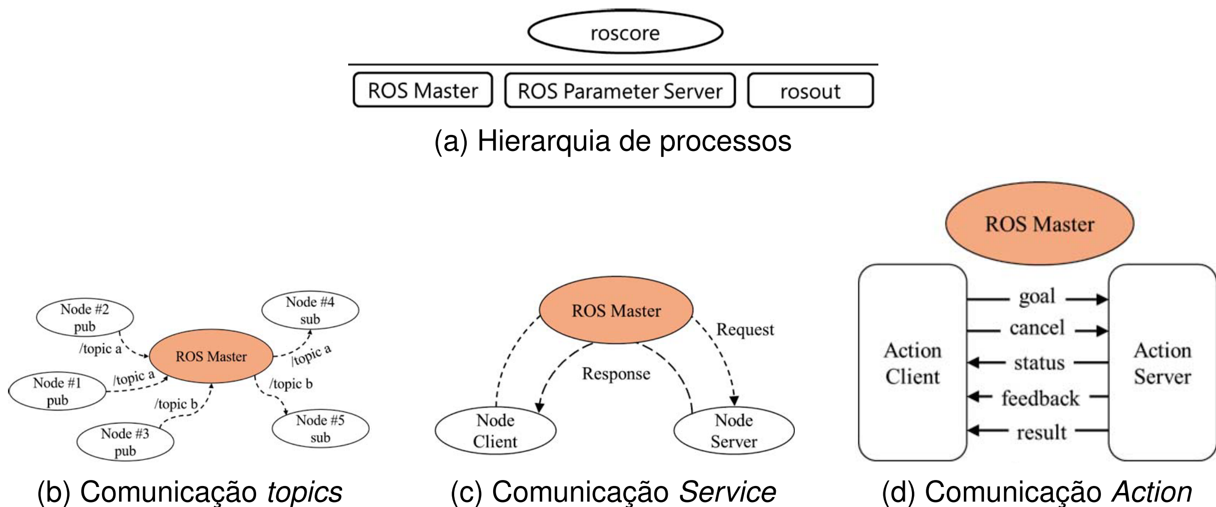
[3] *software* que se encontra na camada entre o sistema operacional e os aplicativos nele executados.

Uma forma alternativa de executar um projeto é com o uso do comando "roslaunch". Através dele é possível iniciar vários *Nodes* ao mesmo tempo e definir parâmetros inicias para a arquitetura do programa. Vale ressaltar que o "roslaunch" sempre iniciará automaticamente o "roscore" caso ainda não esteja em execução.

Atualmente, o "roscore" contém três módulos principais, como mostrado na Fig. 4a. Uma vez executado o "roscore" iniciará o "ROS Master", "ROS Parameter Server" e o "rosout logging node". Destacando "ROS Master" como o responsável por registrar e nomear cada um dos *Nodes* do sistema, além de rastrear e estabelecer a comunicação entre eles.

As subfiguras 4b, 4c e 4d, ilustram os três protocolos de comunicação entre os *Nodes*, e sua relação com o sistema centralizado pelo ROS Master.

Figura 4 – Interfaces de comunicação ROS



(a) Hierarquia de processos

(b) Comunicação *topics* (c) Comunicação *Service* (d) Comunicação *Action*

Fonte: Deng, Xiong e Xia (2017).

3.3 URDF - Formato Unificado de Descrição de um robô

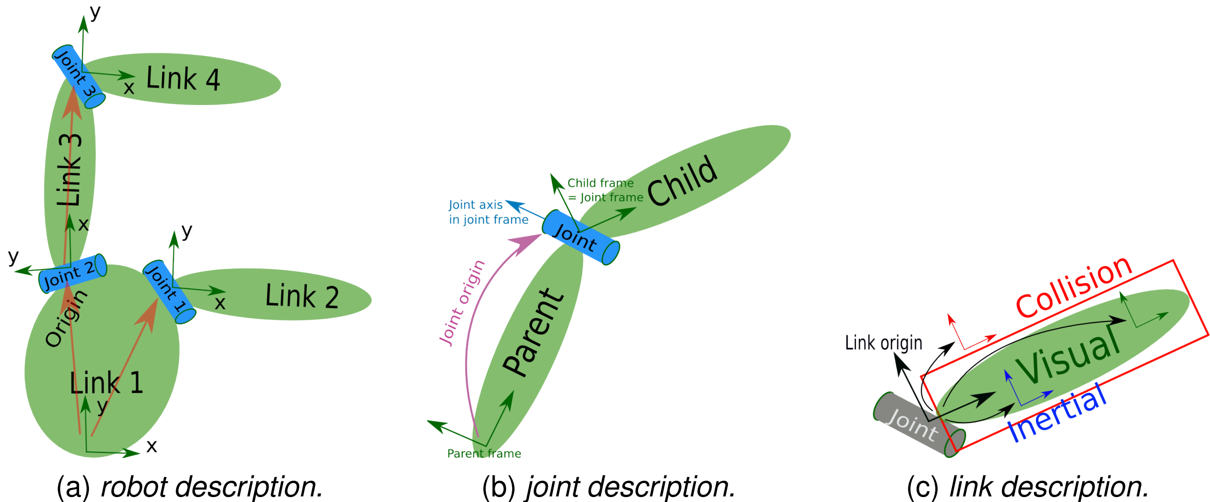
Como descreve Deng, Xiong e Xia (2017), o ROS possui um sistema para descrever a cadeia cinemática modelado em uma estruturas de árvore Fig. 5a. Tendo como berço a linguagem de marcação XML, o URDF foi projetado para registrar todas as informação físicas de elos, juntas, material, textura, geometria, área de colisão, inércia, atuadores e sensores do robô.

As duas principais estruturas de um URDF são os elos e juntas. As junta são responsável por unir os elos de um robô e determinar a classe de sua relação (fixa, rotaciona, prismática, etc). Na Fig. 5b é ilustrado essa relação com uma junta rotativa. O elo, por sua vez, armazena informações como a geometria, região de colisão, momento de inércia, etc. A Fig. 5c é um esboço visual das informações que um elo pode armazenar.

Para economizar tempo de desenvolvimento, *softwares* de modelagem 3D

como o Onshape e o Solidworks possuem extensões capazes de extrair o URDF de modelos CAD, e determinar automaticamente coordenadas, transformações de eixo, centro de massa, peso e tipos de junta.

Figura 5 – Camadas do URDF



(a) *robot description.* (b) *joint description.* (c) *link description.*

Fonte: (a) ROS URDF model [4] (b) ROS URDF joint [5] (c) ROS URDF link [6]

3.4 Parâmetros de Denavit Hartenberg

Assim como Youakim (2015) descreve em sua tese, é possível comparar o modelo URDF com o mapa dos parâmetros de Denavit Hartenberg (DH). Mas antes será apresentado uma breve contextualização do URDF. Esse arquivo apresenta dois principais pilares, a propriedade de *Links*, responsável por descrever parâmetros visuais e de colisão, e a propriedade *Joints* encarregada de armazenar as informações da cinemática do sistema. Este último pode ser segmentado nos seguintes indicadores e relacionados a parâmetros de Denavit Hartenberg:

- *Type*: Descreve o tipo de junta como fixa, rotaciona, prismática, contínua, ou flutuante;
- *Axis* (x, y, z) : Define a direção do movimento ou rotação;
- *Origin* (x, y, z) : Os parâmetros "d" e "a" da tabela DH influenciam diretamente na origem da junta;
- *Origin* (r, p, y) : O parâmetro "α" da tabela DH age sob a rotação da junta em relação ao seu antecessor direto;
- *Theta* (θ) é a quarta e última propriedade da tabela DH, esta representa o angulo de rotação da junta. Todavia, esse indicador muda seu valor durante o movimento e não é definido como um parâmetro do URDF.

[4] <http://wiki.ros.org/urdf/XML/model>
[5] <http://wiki.ros.org/urdf/XML/joint>
[6] <http://wiki.ros.org/urdf/XML/link>

Conforme foi apresentado o mapeamento de cada parâmetro da tabela DH relacionada aos indicadores do URDF, é proposto o seguinte modelo do robô antropomórfico ArmAgeddon [7], representado através dos fundamentos de Denavit Hartenberg listados na Tabela 2. Parte desses parâmetros foram obtidos pelo *site* oficial da Universal Robots em Robots (2020).

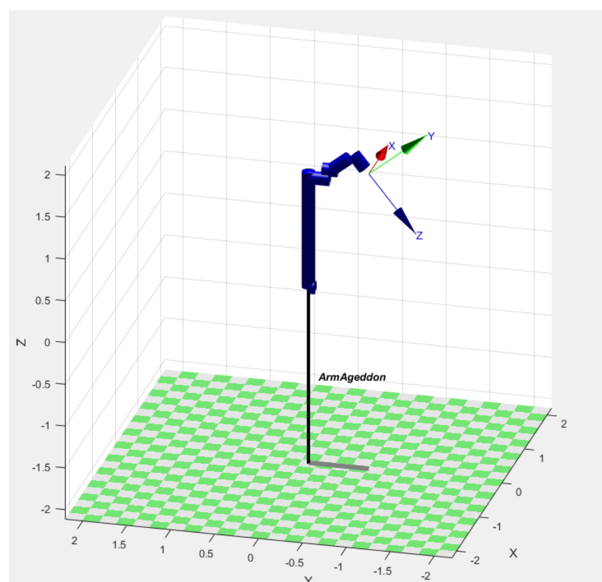
Tabela 2 – Parâmetros de Denavit Hartenberg

Junta	θ [rad]	d [m]	a [m]	α [rad]
1	$\measuredangle$	0.1273	0.0	pi/2
2	$\measuredangle$	0.0	-0.612	0.0
3	$\measuredangle$	0.0	-0.5723	0.0
4	$\measuredangle$	0.163941	0.0	pi/2
5	$\measuredangle$	0.1157	0.0	-pi/2
6	$\measuredangle$	0.0922	0.0	0.0
7	$\measuredangle$	0.430726	0.0	-2.46443

Fonte: Autor (2020).

Para validar esses modelo matemático é feito uma simulação através do Matlab, com o auxílio do pacote *Symbolic Math Toolbox* e *Robotics Toolbox (Peter Corke)*. Este último pacote foi baixado gratuitamente no *site* do desenvolvedor. O código desta simulação está anexada no Apêndice B.

Figura 6 – Simulação com o uso do Matlab e o pacote Robotics Toolbox (PeterCorke).



Fonte: Autor (2020).

[7] Nome do robô utilizado neste trabalho

3.5 Arquitetura e planejamento de trajetórias com o MoveIt

Aplicado em mais de 100 robôs industriais, O MoveIt promete ser uma das ferramentas gratuitas mais promissoras. Ele é lançado sob os termos da licença BSD e, portanto, gratuito para uso industrial, comercial e de pesquisa.

Está entre os *softwares* mais usados para a robótica de manipuladores. Ele fornece uma plataforma ágil e intuitiva para desenvolver aplicativos avançados, projetos escaláveis e integrar produtos para domínios industrial, comercial, pesquisas e outros.

O MoveIt pode ser visto como uma estrutura de *plug-ins*, que encapsula diversas bibliotecas de terceiros para trabalharem em conjunto. Suas três principais bibliotecas são:

- OMPL(Open Motion Planning Library): biblioteca de algoritmos de busca de trajetórias;
- KDL(Kinematics and Dynamics Library): armazena os solucionadores de cinemática direta e inversa;
- FCL (FLexible Collision Library): responsável pela verificação de colisões;

O "Move Group" é o pincipal *Node* do MoveIt, nele são integrados todos os componentes individuais necessários para fornecer ações e serviços ROS disponíveis para os usuários. Todas essas funcionalidades são acessadas através de uma Interface Gráfica do Usuário (GUI) nomeada Rviz, além de permitir incorporar aplicativos mais complexos com o uso de Python ou C++. Tudo é feito de maneira genérica, com abstração de componentes de baixo nível, permitindo aos usuários focarem em aplicações e tarefas de alto nível no aplicativo, como escolher um objeto ou manipulá-lo. A Fig. 7 mostra a arquitetura desse sistema. Uma descrição mas detalhada de cada um dos itens deste fluxograma é apresentado em MoveIt (2020b).

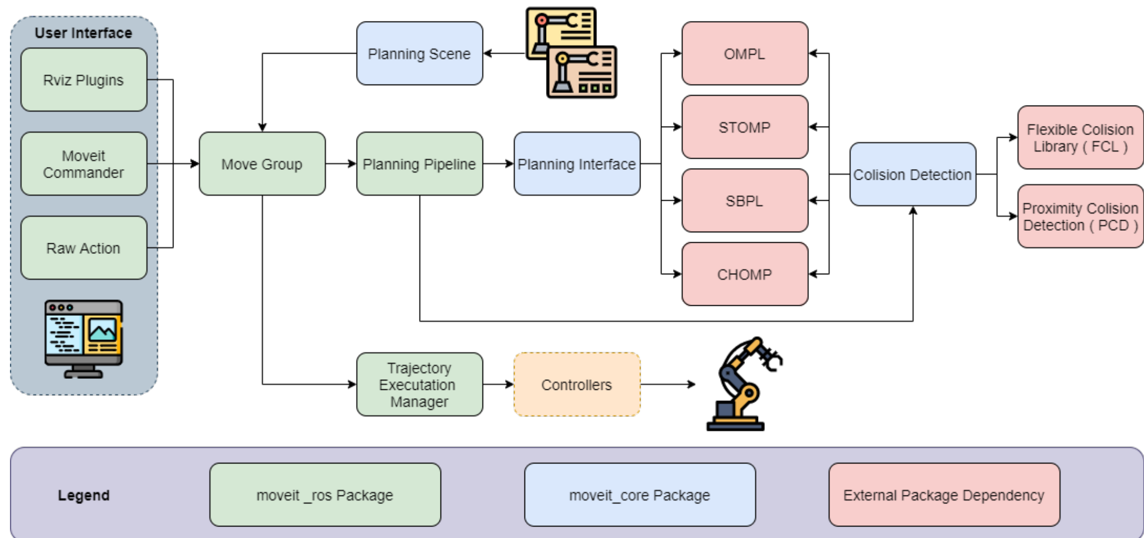
O MoveIt foi projetado para trabalhar com diferentes tipos de planejadores encapsulados através do *node* "Planinnig Interface", que faz parte do grupo de atividades"Move Group". Por consequência de sua vasta opção de planejadores, o torna ideal para aprimorar o *benchmarking* [8] de múltiplos métodos. Abaixo segue a lista de alguns dos planejadores suportados pelo MoveIt, em ordem decrescente de popularidade e suporte:

- OMPL: baseado em amostragem;
- STOMP: baseado em otimização;
- SBPL: baseado em pesquisa;
- CHOMP: baseado em amostragem.

O OMPL é uma biblioteca de planejadores de movimento de código aberto, que implementa principalmente algoritmos de busca randômica. Os planejadores OMPL

[8] Benchmarking é um processo de estudo que gera concorrência na busca das melhores práticas.

Figura 7 – Arquitetura MoveIt.



Fonte: Autor (2020).

são abstratos e não tem o conceito de um robô. Todavia, o MoveIt configura o OMPL e fornece o vínculo necessário para operar com soluções de robótica.

Assim como já apresentado na Sessão 2.1, o RRT é um dos algoritmos de busca, baseado em amostragem, disponível na biblioteca OMPL, e apresenta um bom desempenho para a proposta deste trabalho.

3.5.1 Solucionadores de cinemática inversa

Para solucionar a cinemática inversa de um robô antropomórfico é preciso recorrer a técnicas de cálculo que convertem o espaço cartesiano em uma configuração de ângulos de juntas. Por padrão os tutoriais do MoveIt recomendam utilizar o solucionador numérico KDL. Contudo, este modelo apresenta baixo desempenho e acusa erros quando usado em robôs DOF < 6 MoveIt (2020a).

Para robôs com menos de 6 graus de liberdade é possível utilizar o *plug-in* IKFast. Ao contrário do KDL, que usa solucionadores numéricos, o IKFast opera com solucionadores analíticos, que são muito mais rápidos. O resultado são operações estáveis que podem resolver a cinemática em poucos microsegundos.

O ponto negativo do IKFast é que ele exige a instalação do programa OpenRaves. Na internet há várias páginas relatando que a instalação deste é um processo árduo, custoso e complicado. Até os dias atuais, ainda é discutido a relação de tempo investido para instalar o OpenRaves, contra aceitar possíveis erros em se utilizar o KDL.

O MoveIt fornece um solucionador cinemático para cada grupo de elos e juntas definidos no arquivo *kinematics.yaml*. Contudo, o solucionador do grupo da garra deve sempre ser definido como nulo, pois não apresenta uma cadeia cinemática.

3.5.2 Cenário de colisão

Em paralelo ao planejador de movimento e os algorítimos de cinemática inversa é feita a verificação colisões. A tarefa responsável por esse serviço é o FCL. Esta biblioteca possui a classe "CollisionWorld", que por sua vez encapsula o mundo ao redor do robô como um conjunto de objetos, representados por *Meshes*, formas primitivas (como esferas, cubos, cilindros, cones, planos...) ou por objetos mais complexos, como os "octomap".

Segundo Joseph e Cacace (2018), a verificação de colisão é uma tarefa computacionalmente custosa, representando quase 90% da despesa de processamento no planejamento de trajetórias. Para reduzir esse cálculo o MoveIt fornece uma matriz chamada ACM (Matriz de colisões permitidas). Ela contém um valor binário correspondente à necessidade de verificar uma colisão entre dois pares de corpos. Se o valor da matriz for 1, significa que a verificação de colisão do par correspondente não é necessária. Podemos definir o valor como 1, nos casos onde os corpos estão sempre tão distantes que nunca colidiriam um com o outro. A otimização do ACM é fundamental para a redução do tempo de ciclo, e é conveniente atualizar esta matriz sempre que um novo objeto for adicionado ao cenário.

O assistente de configuração do MoveIt permite com apenas alguns cliques gerar a matriz de auto-colisão. O algoritmo procura por pares de elos no robô que podem ser desativados com segurança. A densidade amostral dessa verificação deve assumir um compromisso de engenharia, pois um número muito alto requer muito tempo de processamento, enquanto densidades mais baixas podem ignorar agressivas colisões.

Outro tema que precisa de atenção na hora de mapear o cenário ao qual o robô irá operar, é quando o efetuador final irá manipular algum objeto. Neste caso o objeto deve ser anexado à garra como se fossem apenas um corpo. Isso permite que o planejador de movimento considere este espaço ao traçar a trajetória.

3.6 Criação de um novo projeto MoveIt

O ponto de partida ao iniciar um novo projeto MoveIt é a descrição da modelagem e cinemática do robô. Esta etapa é dividida em duas fases, como segue:

3.6.1 URDF

Estruturado na linguagem XML, o URDF é um arquivo que unificou a forma de descrever fisicamente os robôs. Assim como descrito na Seção 3.4, os URDFs possuem duas principais propriedades, os *Links* e *Joints*. Pos sua vez, os *Links* armazenam as informações físicas dos elôs. Essas informações são classificadas da seguinte forma:

- *Visual*: Define a aparência do robô através de formas geométrica como cubos e esferas ou através arquivos *Mesh* como SLT e DAE.
- *Collision*: Define geometricamente as áreas de colisão dos elos, tal informação é usada para detectar colisão com o ambiente ou auto-colisão durante o planejamento de trajetórias. Um fato relevante a ser mencionado aqui é que as áreas de colisões são normalmente representadas com formas geométricas simples afim de reduzir o tempo de computação.
- *Inertial*: Detém informações sobre a massa e a matriz inercial rotacional representadas pelos seis atributos ixx, ixy, ixz, iyyy, iyz, izz. Quando não especificados o valor da massa e inercia são definidos como zero.

Um exemplo dessa arquitetura é mostrado a seguir.

```
 <link name="my_link">
    <visual>
     <origin xyz="0 0 0" rpy="0 0 0" />
     <geometry>
       <box size="1 1 1" />
     </geometry>
     <material name="Cyan">
       <color rgba="0 1.0 1.0 1.0"/>
     </material>
   </visual>

   <collision>
     <origin xyz="0 0 0" rpy="0 0 0"/>
     <geometry>
       <cylinder radius="2" length="0.5"/>
     </geometry>
   </collision>

   <inertial>
     <origin xyz="0 0 0.5" rpy="0 0 0"/>
     <mass value="5"/>
     <inertia ixx="100"  ixy="0"  ixz="0" iyy="100" iyz="0" izz="100" />
   </inertial>
</link>
```

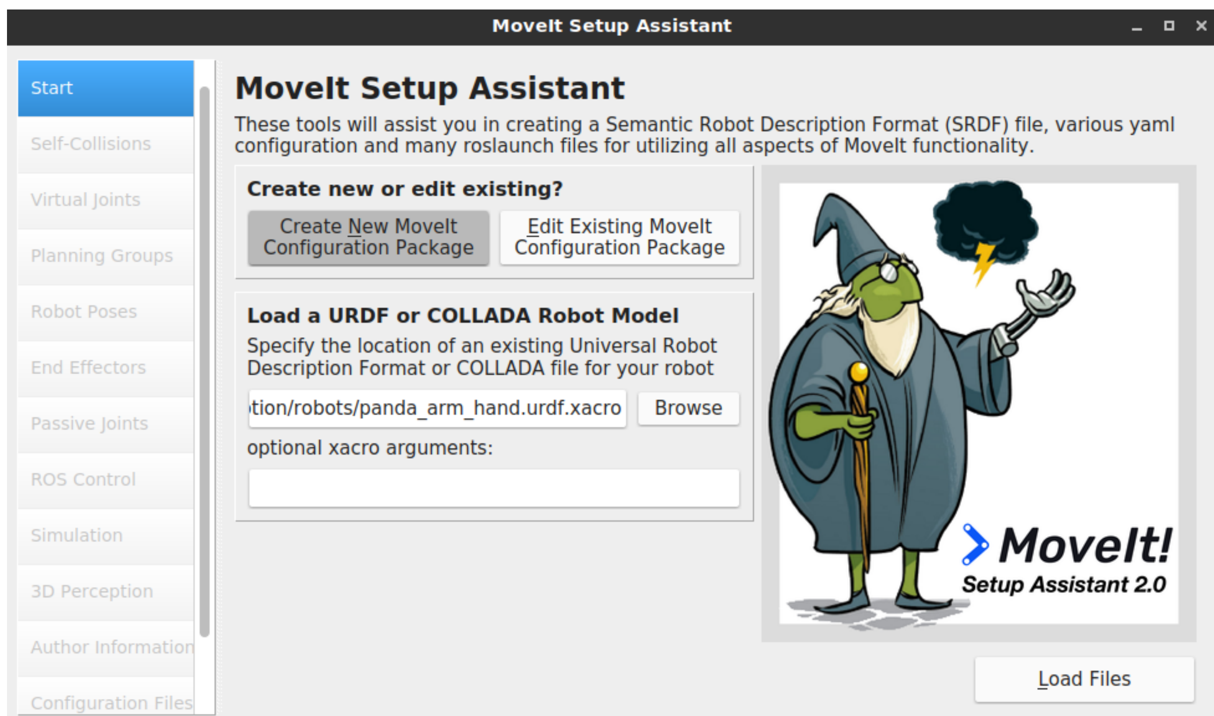
3.6.2 SRDF

O SRDF (Semantic Robot Description Format) é um complemento para o arquivo URDF. Este documento define os "planning groups", que agrupa os elos e juntas classificando as partes que pertencem a garra e o braço, para os planejadores reconhecerem a cadeia cinemática. Além disso, propriedades como o tempo limite

para o solucionador, precisão de resposta e verificação de auto-colisão também são registradas nesse arquivo.

O assistente de configuração do MoveIt mostrado na Fig 8, facilita a criação do SRDF, através de uma interface simples e intuitiva. Outros arquivos como "kinematics.yaml", "joint_limits.yaml", "controller.yaml" e "sensors.yaml" também são gerados automaticamente por esse assistente. Contudo, sempre que alguma alteração for feita no URDF, o MoveIt Setup Assistant deve ser executado para atualizar novamente os arquivos de configuração correspondentes.

Figura 8 – Assistente de configuração MoveIt.



Fonte: Autor (2020).

Ao final desta etapa, a cinemática do sistema está definida e pronta para executar os solucionadores de cinemática inversa. O robô também pode ser visualizado no Rviz usando o arquivo de inicialização padrão gerado pelo assistente de configuração. Simples consultas de trajetórias podem ser executadas no Rviz através do *plug-in* de planejamento de movimento integrado, a fim de verificar a consistência do modelo com a cinemática do sistema.

3.7 Parametrização do Tempo

O MoveIt é a principal estrutura para calcular trajetórias cinemáticas de posição para o efetuador final. Entretanto, não possui um controlador de velocidade e aceleração, em vez disso, o MoveIt oferece suporte a diferentes algoritmos para pós-processamento de uma trajetória cinemática. Neste caso, o algoritmo registra a data e hora dos valores

obtidos para parametrizar no tempo a velocidade e aceleração, com é citado em MoveIt (2020e).

Por padrão, o MoveIt define a velocidade e a aceleração das junta conforme descrito no arquivo URDF. Esses valores são copiados para o arquivo "joint_limits.yaml" ao criar um pacote com o MoveIt Setup Assistant. Através deste arquivo o usuário pode definir ajustes como a velocidade e aceleração máxima de cada junta.

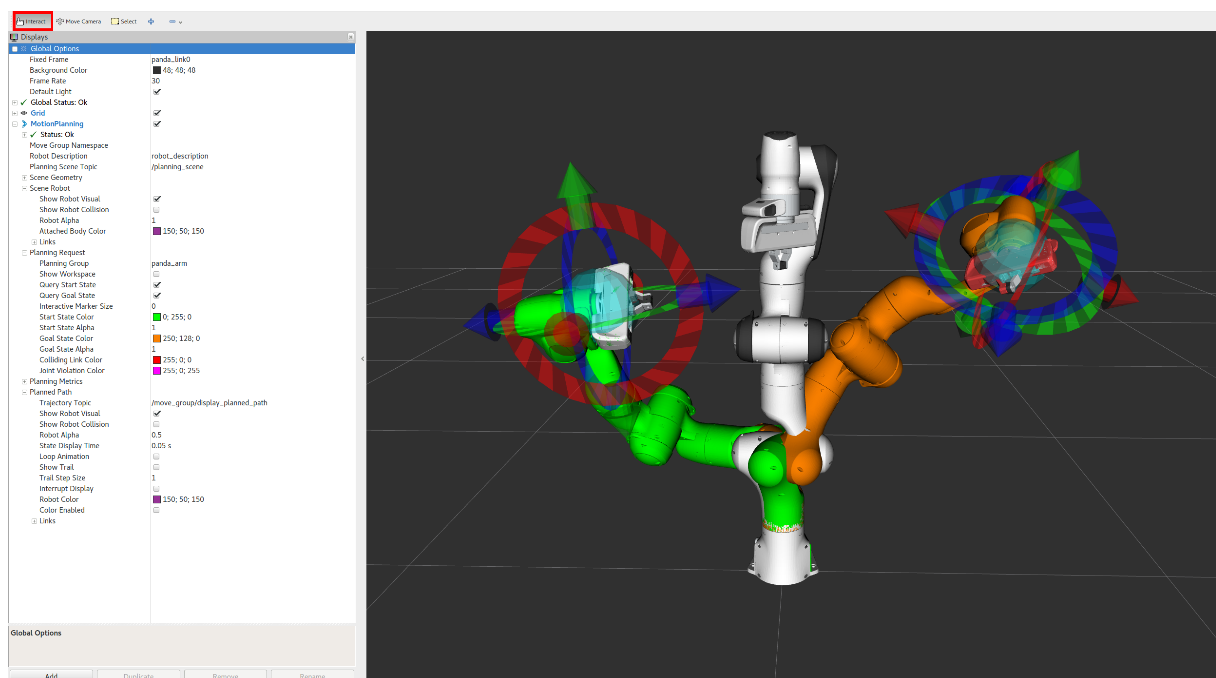
A velocidade e aceleração também podem ser modificadas durante o tempo de execução. Para isso, a velocidade e aceleração máxima das juntas são multiplicados por um fator de escala entre zero e um, por padrão esses atributos são definidos com o valor um, ou seja, estão com seu valor máximo. Dentro do programa é possível alterar essa fração antes de executar o plano de movimento.

3.8 Visualização com Rviz

O Rviz é uma ferramenta para visualizar trajetórias e simulações. Ele fornece uma GUI simples para monitorar dados do sensor, modelos do robô, mapa do ambiente, rota de trajetórias, desvio de obstáculos e determinar pontos para gerar movimentos com a cinemática inversa.

Rviz é o visualizador principal do ROS, através dele o MoveIt configura ambientes virtuais, traça interativamente estados de início e objetivo, testa diversos planejadores de movimento e representa de forma gráfica os resultados. A Fig. 9 ilustra a ferramenta operando como planejador de movimentos início-objetivo.

Figura 9 – Simulação do Panda Arm no Rviz.



Fonte: MoveIt (2020c).

3.9 Ferramentas para exportar URDFs

Atualmente existem vários repertório de *softwares* para a modelagem de objetos em 3D, que variam suas aplicações como cinema, jogos, animações ou em nosso caso, para o design de produtos. O SolidWorks é um dos programas mais conhecido e usados para modelagem técnica de produtos. Além deste, outro projeto de modelagem 3D que vem ganhando espaço no mercado é o Onshape. Este último, se destaca por renderizar a modelagem direto via *browser*, dispensando a necessidade de baixar e instalar o *software*. Por esta razão, através dele será desenvolvido o esboço tridimensional do braço robótico proposto nessa dissertação.

A carcaça virtual concebida na modelagem 3D servirá para dois propósitos. Primeiro, teremos um acelerado processo de prototipagem em um projeto futuro, com o auxílio das impressoras 3D. Segundo, e tão importante quanto, será possível importar este modelo para um ambiente virtual do ROS no qual serão feito os cálculos de trajetória e colisão.

O onshape-to-robot é uma ferramenta que permite importar modelos CAD 3D projetados no Onshape, para o formato de descrição URDF. Para o caso de robôs antropomórficos, o exportador cria uma árvore hierárquica da base até o efetuador final. Essa ferramenta pode economizar um considerável tempo de desenvolvimento, no entanto, ainda exige alguns ajustes nas propriedades do material, juntas passivas, excluir elos fictícios e substituir a área de colisão por polígonos simples para obter melhor desempenho e reduzir o tempo de processamento.

4 PROGRAMAÇÃO DO PACOTE PARA O ROBÔ ANTROPOMÓRFICO

Neste Capítulo será apresentado o diretório de pacotes e códigos Moveit desenvolvidos neste trabalho.

Este trabalho foi desenvolvido com a versão recomendada (2020) ROS1 Melodic Morenia para a distribuição Linux Ubuntu Budgie 18.04 LTS, e compilado com a extensão Catkin Tools (catkin build). Vale o registro que a última versão do ROS1 será a Noetic Ninjemys, mas no momento se encontra em fase Beta. Existe também o projeto Moveit 2 para o ROS2, porém ainda está em processo de migração.

Este projeto terá seu código aberto e disponível no site GitHub [1]. Nesta página há uma breve descrição do modo de operação para o usuário e um bash-script para instalar automaticamente todos os programas necessários para a execução do projeto. Maiores detalhes sobre a página do GitHub são descritos no Apêndice C.

Para tornar o trabalho didático e facilitar a continuação deste, as ilustrações serão acompanhadas do código de terminal que as origina.

4.1 Construção do modelo 3D

Através da comunidade de compartilhamento GrabCad[2], é possível baixar modelos CAD de forma gratuita. Por meio deste site , foi obtido o arquivo "SLDPRT" do UR10 da Universal Robots, e nele será anexado uma tocha de solda. Ambos os arquivos foram carregados para a plataforma web Onshape [3] como demonstrado na Fig. 10. Este site permite desenvolver modelos 3D e criar a relação de juntas na aba de montagem.

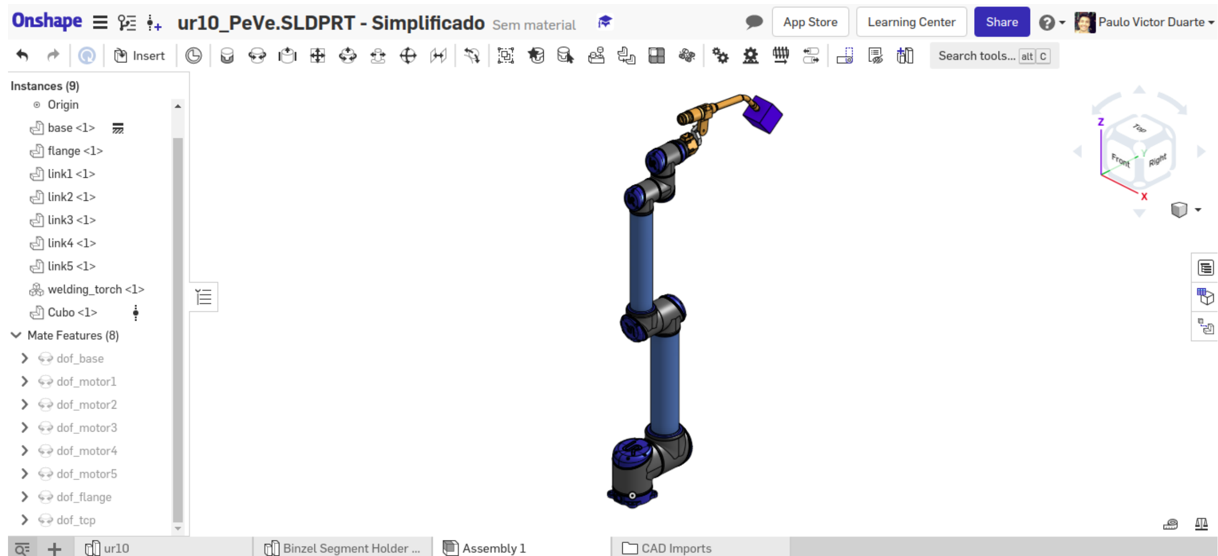
Com a extensão onshape-to-robot[4] o modelo 3D é convertido em URDF de forma automática. A extensão é bem documentada, e propõe várias soluções que aceleram o processo de criação do URDF. As juntas rotativas devem ser renomeadas com o prefixo "dof", através do prefixo que o onshape-to-robot determina o tipo de junta entre os elos do robô. Todavia, ainda é necessário alguns ajustes manuais no URDF para determinar elos passivos e juntas fixas.

A escolha de uma tocha de solda como ferramenta de trabalho servirá como um laboratório para prever possíveis desafios enfrentados pela camada de *software* em simulações. Contudo, as aplicações desenvolvidas não se comprometem com um embasamento real do movimento de solda, este pode ser um tema para trabalhos

[1] <https://github.com/paulovictor237/armageddon>
[2] <https://grabcad.com/library>
[3] <https://cad.onshape.com>
[4] <https://onshape-to-robot.readthedocs.io/en/latest>

futuros que objetivam unir conceitos de diferentes áreas da engenharia.

Figura 10 – Projeto modelado no Onshape



Fonte: Autor (2020).

4.2 Modelo de colisão

A substituição das malhas em STL por modelos geométricos primitivos, é conveniente para melhorar o tempo e desempenho dos cálculos de colisão que o MoveIt realiza. Por padrão, as malhas STL também são usadas para colisão, porém isto é versátil, mas é caro do ponto de vista computacional e pode ser numericamente instável. O onshape-to-robot propõe uma solução rápida e eficiente para criar esses modelos com o programa OpenSCAD.

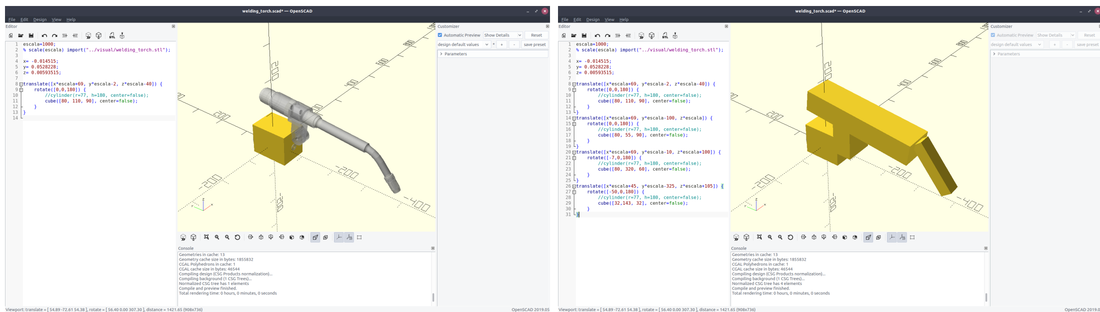
OpenSCAD é um *software* de código aberto para a criação de sólidos 3D CAD. Ao contrário da maioria dos *softwares* livres para a criação de modelos 3D como o Blender, o OpenSCAD funciona como um compilador, que lê um script de descrição do objeto, e renderiza o modelo 3D. As partes podem ser visualizadas, mas não podem ser interativamente selecionadas ou modificadas com o mouse.

O *script* especifica modelos geométricos primitivos como esferas, caixas, cilindros, etc, ideais para o propósito do aramado de colisão usado nos cálculos do MoveIt. É preciso que cada peça seja descrita manualmente assim como é ilustrado na Fig. 11.

Com os modelos OpenSCAD finalizados, o onshape-to-robot consegue criar automaticamente o URDF substituindo a geometria de colisão. Contudo, alguma imcompatibilidade do OpenSCAD com o Ubuntu 18.04 impossibilitou o uso desta função. A solução proposta para contornar esse problema foi exportar cada modelo OpenSCAD para STL, e importar manualmente para o URDF.

Devido ao limite de renderização do próprio programa, os arquivos exportados

Figura 11 – OpenSCAD

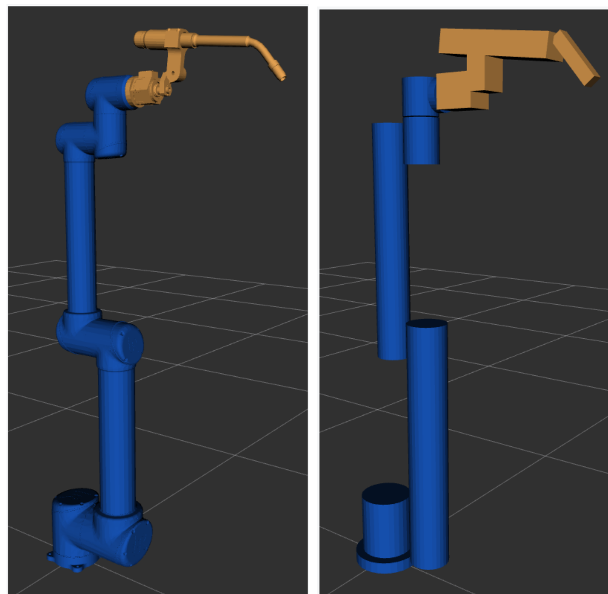


(a) Modelo em STL (b) Modelo de colisão

Fonte: Autor (2020).

possuem uma escala 1.000 maior que a original, com isso, ainda foi necessário reduzir a escala geométrica no URDF com o comando <scale="0.001 0.001 0.001">. O resultado é demonstrado na Fig. 12b, ao lado da malha STL original.

Figura 12 – Aproximação por modelos geométricos primitivos



(a) Modelo em STL (b) Modelo aproximado

Fonte: Autor (2020).

4.3 Verificar hierarquia do URDF

Para verificar se o arquivo URDF foi gerado corretamente é possível recorrer a uma função que o pacote Moveit dispõe por padrão. A verificação é feita através do seguinte comando de terminal.

```
$  check_urdf arquivo.urdf
```

Essa linha de comando verifica a hierarquia dos elos e juntas. Em caso positivo da verificação será então impresso a descrição da cadeia cinemática no terminal, como é demonstrado abaixo. Uma mensagem de erro será exibida no caso de falha.

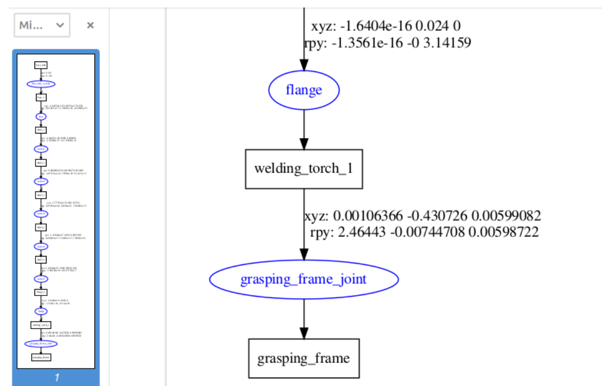
```
robot name is: ArmAgeddon
---------- Successfully Parsed XML ---------------
root Link: base_link has 1 child(ren)
    child(1):  base_1
        child(1):  link1_1
            child(1):  link2_1
                child(1):  link3_1
                    child(1):  link4_1
                        child(1):  link5_1
                            child(1):  flange_1
                                child(1):  welding_torch_1
                                    child(1):  grasping_frame
```

Outra função interessante que o Moveit dispõe é a de converter o URDF em um arquivo PDF com o desenho em árvore da cadeia cinemática. Com o comando abaixo teremos o arquivo ilustrado na Fig. 13.

Listing 4.1 – Código referetnte a Fig. 13.

```
$ urdf_to_graphiz arquivo.urdf
```

Figura 13 – Conversão de URDF para Graphiz.



Fonte: Autor (2020).

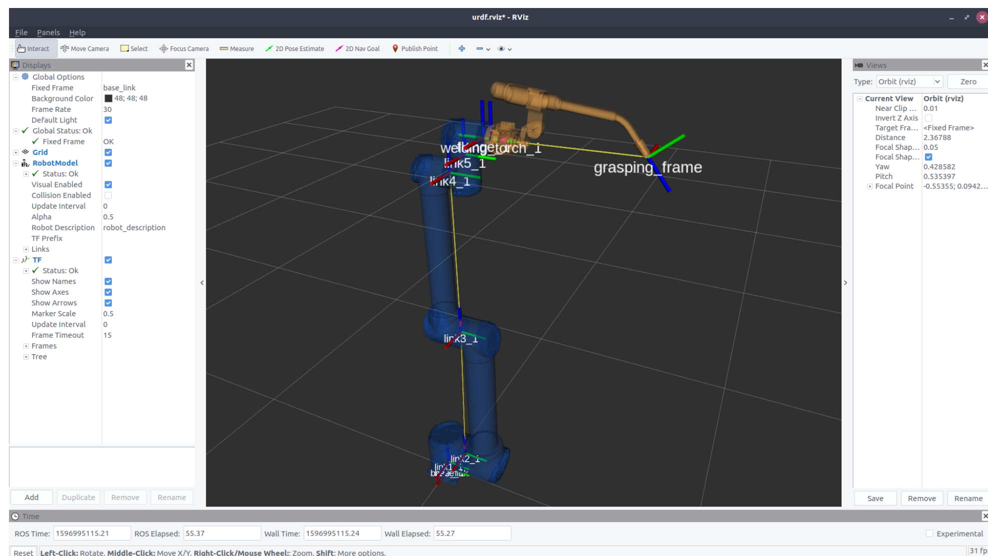
4.4 Preview do arquivo URDF

Após obter o arquivo URDF é conveniente visualizar o robô na interface do Rviz, ver Fig. 14. O comando que se segue também dá acesso a outra interface, o "joint_state_publisher", que permite controlar manualmente cada junta do robô.

Listing 4.2 – Código referetnte a Fig. 14.

```
$  roslaunch urdf_tutorial display.launch model:=arquivo.urdf
```

Figura 14 – Visualização 3D do URDF.



Fonte: Autor (2020).

4.5 Ponto central de atuação

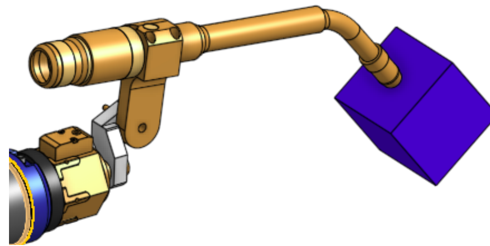
O espaço de trabalho (alcance) do robô, também é influenciado pelo ponto central de atuação ou TCP. Além do TCP, informação de momento e massa da garra também devem ser levadas em consideração quando é preciso simular com máxima precisão o comportamento real de um robô.

O TCP é o ponto de contato ou centro da garra do efetor final. Em robôs industriais este ponto é definido pela exata diferença translacional e rotacional entre a estrutura da flange[5] do robô e a ponta do efetuador final, por exemplo, uma garra de dois dedos, uma ventosa, uma ferramenta de soldagem, etc.

Para obter este ponto de forma intuitiva foi anexado o modelo de um cubo à ponta da tocha de solda, ver Fig. 15. Ao exportar o URDF com o onshape-to-robot, encontramos o TCP na origem de contato do cubo. Em um processo posterior, pode-se remover a geometria do cubo no arquivo URDF.

[5] Flange é a superfície que une o robô à garra.

Figura 15 – Grasping Frame



Fonte: Autor (2020).

4.6 Motion Planning

Para planejar e executar trajetórias o Moveit conta com uma eficiente interface chamada "Motion Planning". Com um "marcador interativo", é possível determinar uma posição inicial e final, e ver o trajeto calculado antes de realizar o movimento. Dentro do arsenal de ações que essa ferramenta dispõe há também a opção de executar trajetórias lineares quando a opção "Use Cartesian Path" é ativada.

Por padrão o "marcador interativo" é fixado na flange do robô. Todavia, há casos em que se deseja o mesmo fixado na região de pega da garra, ou em um ponto de atuação. Afim de contornar essa situação deve-se adicionar um elo auxiliar ao arquivo URDF para deslocar o "marcador interativo" até essa região. Também, pode-se aproveitar desse elo extra para adicionar um possível momento de inércia causado pelo objeto que será anexado à garra.

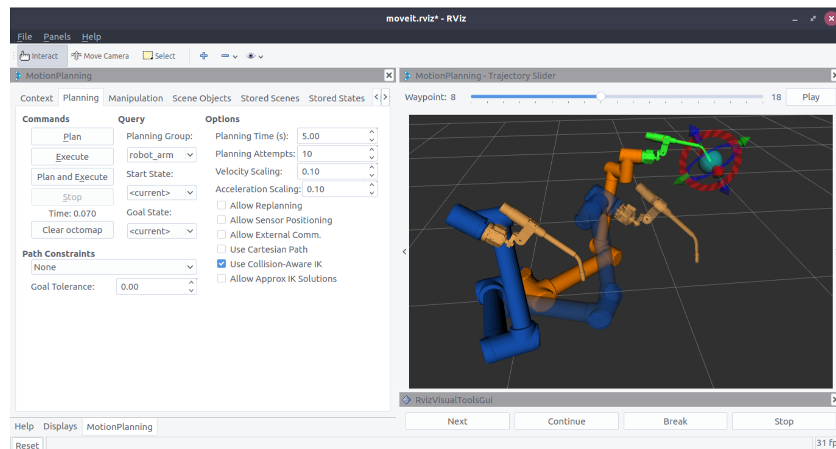
Como já mencionado na sessão anterior, é adequado deslocar o "marcador interativo" até o TCP do robô. Neste caso, bastou apenas renomear no arquivo URDF o cubo para "grasping_frame" e apagar a descrição da geometria. O código abaixo demonstra como pode ser construído esse novo elo. A Fig. 16 mostra a interface "Motion Planning" executando um cálculo de trajetória.

```
1 <link name="grasping_frame">
2 </link>
3
4 <joint name="grasping_frame_joint" type="fixed">
5 <origin xyz="0.00106366 -0.430726 0.00599082" rpy="2.46443 -0.00744708
    0.00598722" />
6 <parent link="welding_torch_1" />
7 <child link="grasping_frame" />
8 </joint>
```

Listing 4.3 – Código referetnte a Fig. 16.

```
$ roslaunch ArmAgeddon demo.launch
```

Figura 16 – Motion Planning.



Fonte: Autor (2020).

4.7 Criação de pacotes MoveIt Setup Assistant

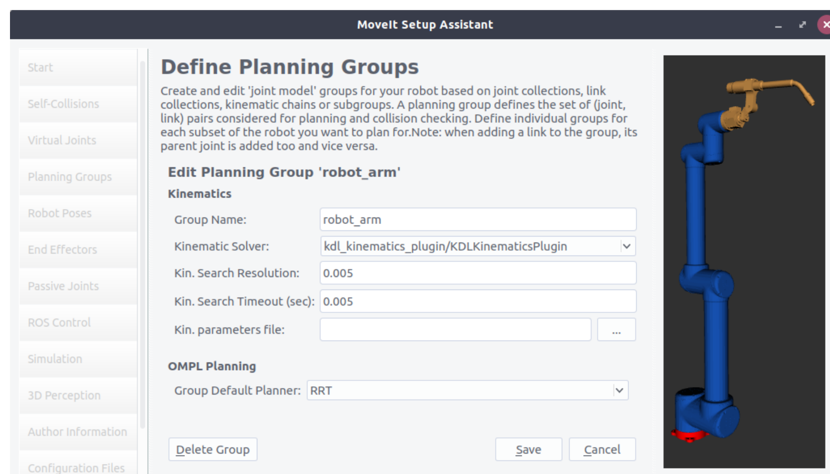
Nessa etapa é criado o pacote Moveit, onde será possível organizar toda a estrutura dos planejadores de trajetórias, identificar auto colisão, separar em grupos o braço e o efetuador, criar posição "Home" e adicionar *fake controllers* para cada junta. A Fig. 17 mostra a opção do RRT como planejador OMPL.

Neste passo é importante ressaltar que o pacote de dados gerados com o assistente MoveIt deve ser criado em um novo diretório. O assistente não permite gerar os arquivos no mesmo pacote que se encontra os arquivos de descrição URDF.

Listing 4.4 – Código referetnte a Fig. 17.

```
$ roslaunch moveit_setup_assistant setup_assistant.launch
```

Figura 17 – Moveit assistente.



Fonte: Autor (2020).

4.8 Compilar aplicações

Para aplicações em C++ que interagem com o robô é necessário adicionar algumas linhas ao arquivo "CMakeLists" e compilar o código com o "Catkin Build". Vale ressaltar que aplicações em Python não precisam desses passos. As linhas que devem ser inseridas no "CMakeLists" devem seguir o exemplo a seguir:

Listing 4.5 – CMakeLists Exemplo

```
1 add_executable(exemplo src/exemplo.cpp)
2 target_link_libraries(exemplo ${catkin_LIBRARIES} ${Boost_LIBRARIES})
3 install(TARGETS exemplo DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```

5 SIMULAÇÕES COM O PACOTE DO ROBOT ANTROPOMORFICO

Para este trabalho foram elaboradas 5 simulações que serão descritas neste capítulo bem como os resultados obtidos.

A biblioteca C++ do MoveIt possui a classe "MoveGroupInterface", a qual promove a interface entre o ROS e o *node* "move_group". As 5 simulações foram desenvolvidas através dos métodos dessa classe.

Abaixo segue os comandos para executar essas aplicações. É preciso executa-los em diferentes terminais. Também é possível controlar cada eixo do robô individualmente através de uma interface gráfica descomentando o "use_gui:=true".

Listing 5.1 – Comandos Bash.

```
$ roslaunch armageddon_moveit demo.launch #use_gui:=true
```

```
$ rosrun armageddon_robot GoRandom
$ rosrun armageddon_robot GoHome
$ rosrun armageddon_robot TypeMoves
$ rosrun armageddon_robot CartesianReader
$ rosrun armageddon_robot StopMove
```

5.1 Aplicação GoRandom

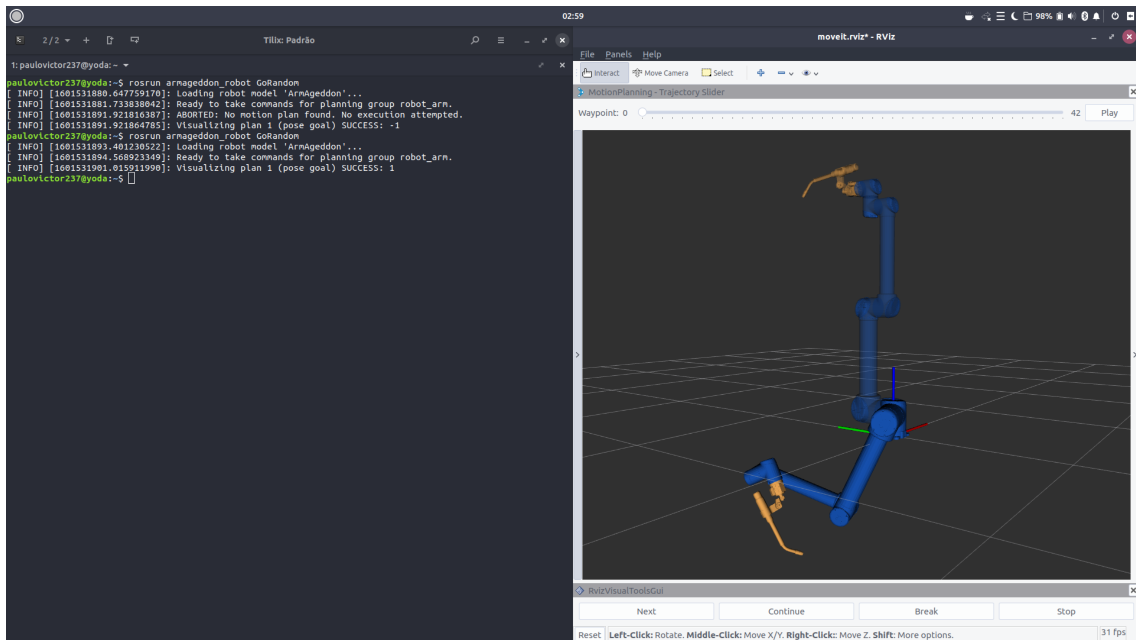
O objetivo dessa aplicação é criar um código curto e simples que apenas executa um movimento randômico. Como normalmente o ponto está muito distante da atual posição do robô, isto origina dois problemas, o tempo padrão para o MoveIt calcular a trajetória é insuficiente e o caminho é imprevisível. Para melhorar a frequência com que esse programa encontra uma trajetória de sucesso, o tempo padrão de 5 segundos foi aumentado para 10, e o algoritmo de planejamento de trajetória foi forçado para executar com o RRT, para garantir que uma trajetória seja encontrada, mesmo não sendo a ideal. Esta solução é demonstrada pelas seguintes linhas de código.

Listing 5.2 – Código GoRandom

```
1 group.setPlannerId("RRTConnectkConfigDefault");
2 group.setPlanningTime(10.0);
```

O resultado obtido é apresentado na Fig. 18, nesta simulação é possível reparar que o planejador de trajetória falhou uma vez antes de encontrar um caminho randômico possível. Todavia, mesmo encontrando o caminho, a trajetória é imprevisível e pode ser um sério problema para um robô real.

Figura 18 – Resultado da aplicação GoRandom.



Fonte: Autor (2020).

5.2 Aplicação GoHome

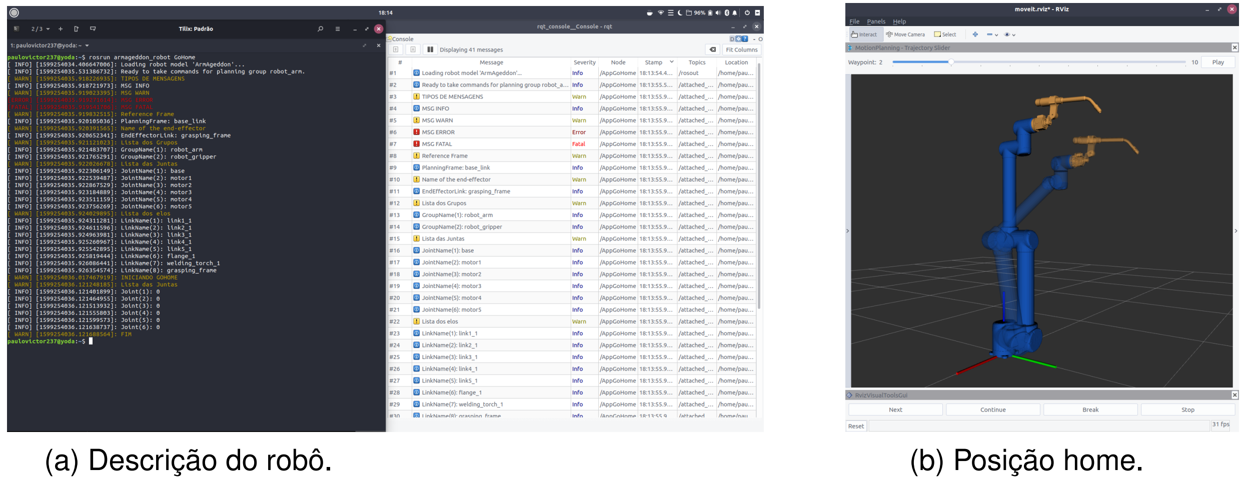
A aplicação GoHome força todas as juntas do robô para o valor zero, independente de onde ele esteja. Este código também serve como uma demonstração dos tipos de mensagens que o ROS pode emitir, e imprime na tela várias informações do robô extraídas dos próprios métodos da classe "MoveGroupInterface"

A Fig. 19a, apresenta essas mensagens através do terminal Linux em comparação à ferramenta gráfica do ROS, o "rqt_console". O resultado é apresentado na Fig. 19b, o robô translucido representa a trajetória, já o robô opaco descreve a posição final da aplicação.

Listing 5.3 – Código referetnte a Fig. 19.

```
$ rostopic echo /rosout/msg
$ rosrun rqt_console rqt_console
```

Figura 19 – Resultado da aplicação GoHome.



(a) Descrição do robô. (b) Posição home.

Fonte: Autor (2020).

5.3 Aplicação TypeMoves

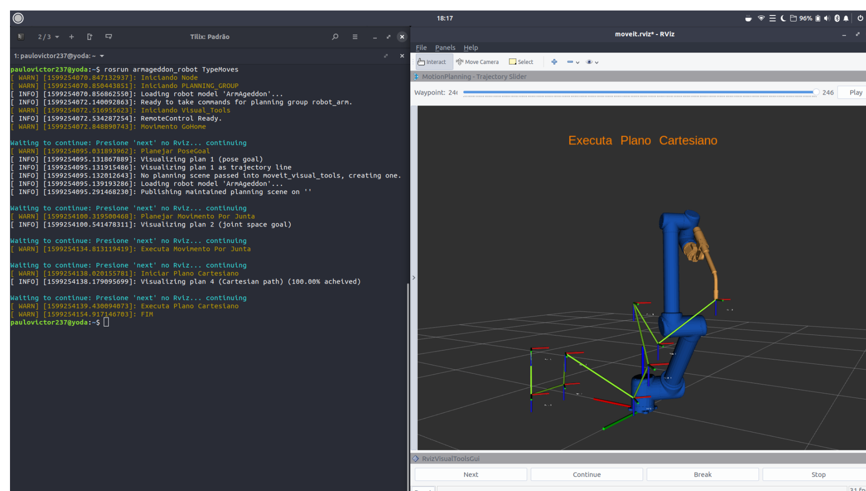
TypeMoves é uma demonstração dos tipos de movimento que o "Move Group Interface" dispõe. A aplicação planeja e executa as seguintes tarefas:

- Movimento para um ponto qualquer;
- Movimento inserindo valores de juntas;
- Movimento linear inserindo valores de posição e orientação.

Nesta prática foi necessário converter a orientação *Roll Pitch Yall* em *Quaternions* antes de atribuir os valores ao robô.

O código também faz uso da classe "MoveIt Visual Tools" para apresentar mensagens e desenhar trajetórias no Rviz. Na Fig. 20 mostra o resultado do plano cartesiano desta aplicação, o resultado dos demais métodos também foram obtidos.

Figura 20 – Resultado da aplicação Type Moves.



Fonte: Autor (2020).

5.4 Aplicação CartesianReader

O programa CartesianReader utiliza a função "CartesianPath" implementada pela classe "MoveGroupInterface" já apresentada na aplicação TypeMoves. Esta função recebe como entrada um vetor de pontos e planeja a trajetória com movimentos lineares. Outros parâmetros de inicialização da função são os seguintes:

- Fator de escala de velocidade e aceleração máxima;
- Tempo máximo para o cálculo da cinemática inversa;
- Tamanho do passo, em metros, entre os pontos da trajetória;
- Opção de ativar ou desativar a rotina que evita colisões com os objetos do ambiente.

Os parâmetros e o conjunto de pontos a serem passados para a função são declarados em um arquivo de extensão Markdown presente nos diretórios do pacote "armageddon_robot". O Programa lê esse documento e cria o planejamento de trajetória entre os pontos.

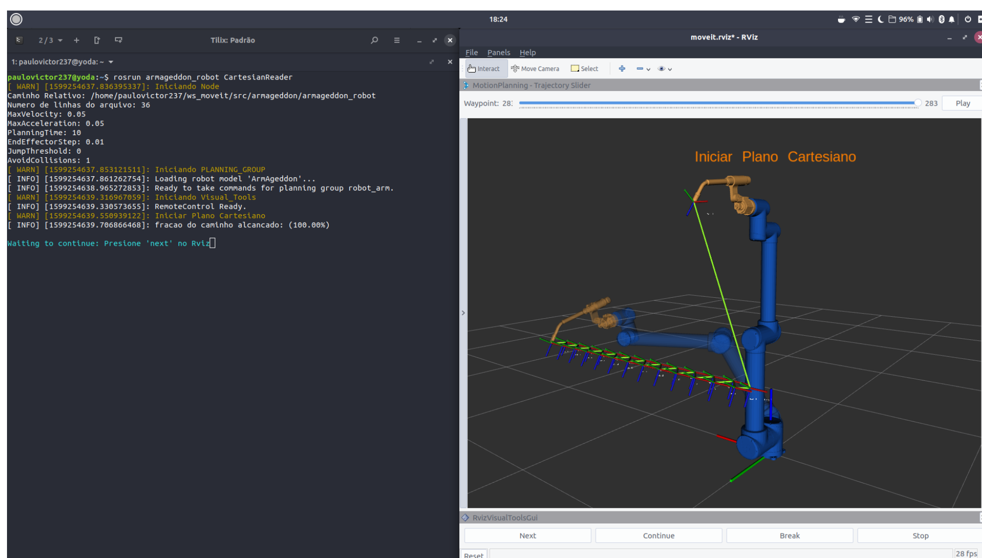
Todavia, o caminho para esse arquivo é abstrato para o programa, visto que os pacotes do ROS podem ser executados em qualquer diretório do Linux. Para resolver esse detalhe, o código captura o caminho relativo do arquivo no *workspace* onde o pacote foi instalado, como é demonstrado abaixo.

Listing 5.4 – Caminho relativo

```
1 string RelativePath=ros::package::getPath("armageddon_robot");
```

O arquivo de pontos tem como exemplo uma trajetória de solda fictícia para demonstrar a aplicação em execução, como é apresentado na Fig. 21.

Figura 21 – Resultado da aplicação Cartesian Reader.



Fonte: Autor (2020).

5.5 Aplicação StopMove

As chaves de fim de curso são um equipamento muito utilizado em projetos da robótica e da indústria. Este dispositivo tem como função indicar que um motor ou estrutura ligada ao seu eixo chegou ao limite do seu campo de movimento.

Para evitar danos ao equipamento, é desejado o total bloqueio de esforços sob o motor quando as chaves de fim de curso são acionadas. Por conseguinte, espera-se obter o mesmo comportamento espelhado à simulação.

Através deste contexto é apresentado o resultado da aplicação StopMove. Esta aplicação desenvolve a interface de *software* para o acionamento de uma chave fim de curso, e suspende qualquer movimento do robô independente da aplicação em operação.

O programa cria um tópico chamado "stop_robot", que aguarda uma mensagem com a palavra "STOP" para executar a chamada da função "MoveGroupInterface::stop".

A criação de um grupo de planejamento só é permitido na *Main*. Todavia, a construção de um tópico é através de uma *thread*, o que dificulta passar variáveis da *main* como parâmetro. A solução foi criar uma *class-thread*, nos atributos há um ponteiro que recebe por referência o grupo de planejamento criado na main.

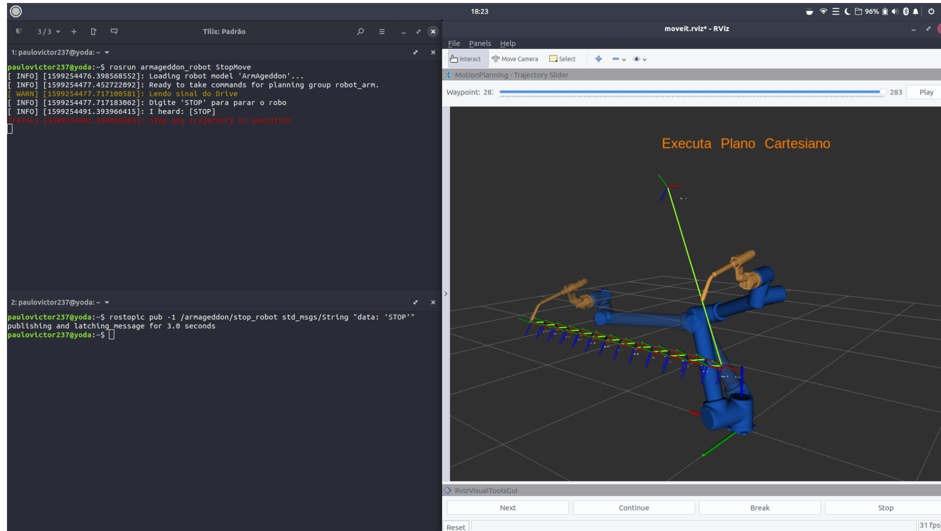
O "roscpp" se encarrega de realizar o *callback* sempre que uma nova mensagem chegar ao tópico, desta forma, a frequência de leitura sempre será síncrona à taxa de publicação no tópico. Em outras palavras, espera-se que o tópico receba a mensagem imediatamente após ela ser publicada, considerando a taxa de processamento do computador.

Por padrão o "rostopic pub" publica uma mensagem ao tópico e mantem bloqueada até que o usuário force a interrupção do programa pressionando "ctrl-c". Todavia, é possível passar como argumento dessa função o valor -1 como é mostrado na Fig. 22, assim, o "rostopic" manterá a mensagem travada por 3 segundos e encerrará automaticamente a publicação.

Listing 5.5 – Código referetnte a Fig. 22.

```
$ rosrun armageddon_robot StopMove
$ rostopic pub –1 /armageddon/stop_robot std_msgs/String "data: 'STOP'"
```

Figura 22 – Resultado da aplicação Stop Move.

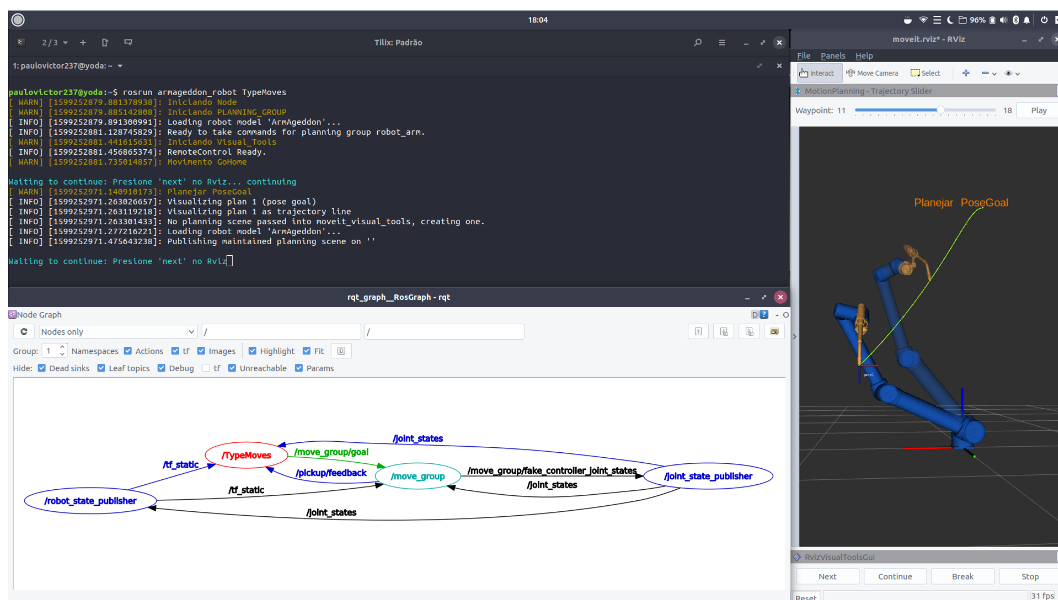


Fonte: Autor (2020).

5.6 ROS GUI

O rqt possui um arsenal completo de ferramentas gráficas que facilitam o entendimento da abstração das linhas de código. Uma das mais interessantes é o "rqt_graph". A Fig. 23 mostra este programa em execução, nele é possível mapear a relação entre os tópicos do MoveIt e as aplicações.

Figura 23 – Mapa gráfico da relação entre os *nodes* .



Fonte: Autor (2020).

5.7 Captura de pontos pelo simulador

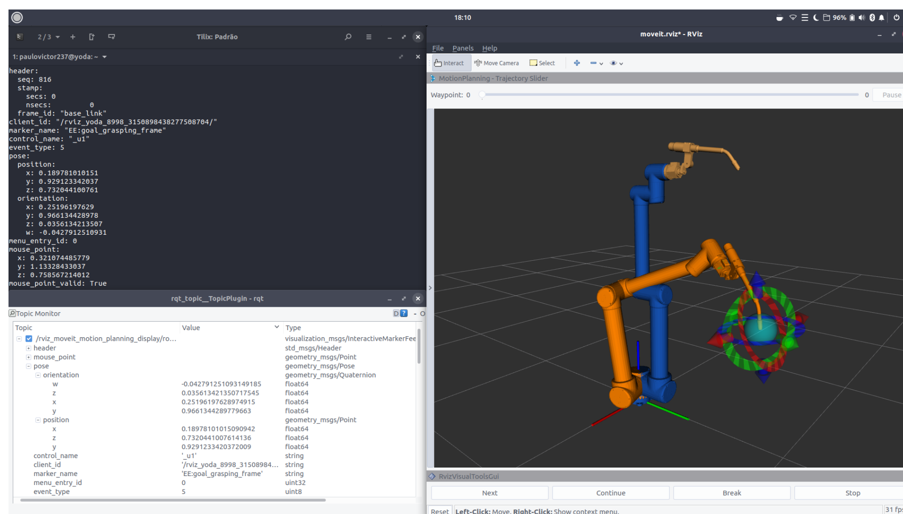
Para mapear os pontos utilizado na aplicação CartesianReader, é desejado que os pontos estejam dentro do espaço de trabalho do robô. Uma forma simples e rápida de obter a posição e orientação do robô dentro de seu espaço de trabalho pode ser feita através do "Motion Planning". A ideia é mover o robô com esta ferramenta, e através do tópico de *Feedback* obter a localização do TCP.

A informação do tópico também pode ser visualizada com a interface gráfica rqt como mostra a Fig. 24.

Listing 5.6 – Código referetnte a Fig. 22.

```
$ rostopic echo –c /rviz_moveit_motion_planning_display/
        robot_interaction_interactive_marker_topic/feedback
$ rosrun rqt_topic rqt_topic
```

Figura 24 – *Topic* Feedback.



Fonte: Autor (2020).

5.8 Integrar software e hardware

Para criar uma interface entre o ROS e o *hardware*, deve-se criar um *node* inscrito no tópico "joint_state". O MoveIt faz todo o cálculo da cinemática inversa, e publica neste tópico a posição de cada junta do robô em radianos.

O *node* "armageddon_joint_states" é quem realiza esta tarefa. Através das coordenadas polares das juntas o código deve traduzir esses valores para a linguagem do *hardware*, e espera-se ver um movimento síncrono entre o simulador e o robô real. Este *node* foi desenvolvido em uma iniciação cientifica do Laboratório de Simulação Naval (LaSiN) por Carvalho (2020) em colaboração com contribuições do autor deste trabalho para a construção do código.

O Grupo de Pesquisa do Laboratório de Simulação Naval está desenvolvendo um manipulador serial subaquático para aplicações como abertura e fechamento de válvulas ou soldagens submersas. Todavia, como experimento inicial, é proposto desenvolver uma aplicação com o ROS e o Moveit para criar um *node* capaz de gerenciar a camada de *hardware* e movimentar 6 motores de passos.

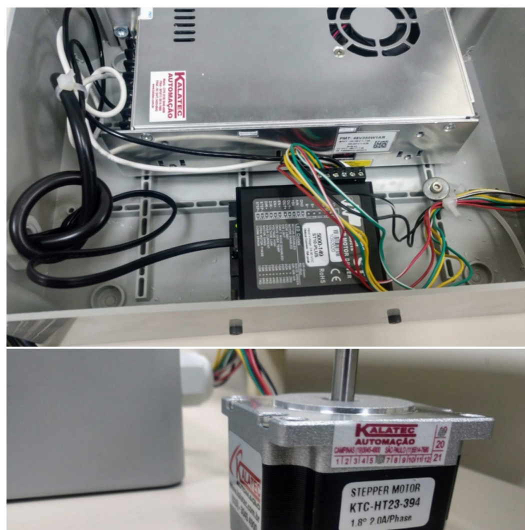
O sistema opera em malha aberta, pois não possui nenhum sensor para realimentar a malha de controle com a posição, velocidade ou aceleração dos motores. Contudo, as aplicações do MoveIt podem ajusta esses parâmetros através do pós-processamento dos dados como mencionado em 3.7.

Os motores de passos são controlados por um *drive* industrial (ST10-Plus) e conectados a um *Hub* que distribui a informação aos 6 motores através de um único cabo USB conectado ao computador. O controlador recebe parâmetros constantes de velocidade, aceleração e posição. Seu limite de precisão é de 36000 passos para uma volta completa. A Fig. 25, mostra o primeiro teste deste circuito, na figura vemos uma fonte alimentando o *drive* ST10-Plus ligado a um motor de passos.

Também foi desenvolvido um breve teste com as chaves de fim de curso. O resultado foi como esperado, e logo que a chave é acionada os motores travam instantaneamente. Entretanto, o teste foi realizado com apenas uma única chave, para um projeto futuro será necessário que cada motor tenha sua respectiva chave de fim de curso, além disso, resta também comunicar o sinal destes com a aplicação StopMove.

O IC foi testado com o robô tutorial do Moveit (Panda). Todavia, como o ROS é modular, é esperado que o *node* opere de forma semelhante ao teste com o robô deste trabalho. No entanto, por conta da situação da pandemia de Coronavírus (COVID-19) não foi possível realizar o experimento.

Figura 25 – Painel com um *drive* e um motor.



Fonte: Carvalho (2020)

6 CONCLUSÕES

Neste capítulo é apresentado as considerações finais e sugestões melhorias para trabalhos futuros.

6.1 Considerações finais

Este trabalho apresentou todo o processo para transportar o modelo 3D de um robô UR10 com uma tocha de solda acoplada em sua flange para o ambiente de programação e simulação do ROS com o pacote MoveIt, onde foi realizado a interação entre os *nodes* com aplicações em C++.

Para o que o estudo se propôs os resultados foram satisfatórios, as cinco aplicações C++ funcionaram como esperado, em harmonia com a versão 18.04 do Ubuntu e ROS Moledic Morenia. O processo de aprendizagem foi gradativo durante a construção das aplicações, onde cada uma elevou um passo a cima a complexidade teórica de seu antecessor.

O projeto contribui para a comunidade acadêmica ao exibir um roteiro completo para a criação de um pacote MoveIt, abordando desde a elaboração de um modelo 3D, exportando-o para o modelo de descrição universal URDF, contextualizando cada etapa do MoveIt Setup Assistant, apresentando cinco aplicações C++ em níveis graduais de dificuldade e complexidade, até a camada de *hardware*. E, introduzindo técnicas e dicas para aperfeiçoar trabalhos semelhantes. Todavia, por se tratar de um *software* de código aberto e em constante atualização, muitos artigos, trabalhos e métodos se tornam ultrapassado e obsoleto em um curto prazo de tempo.

O termo barreira de entrada é muito utilizado no contexto da robótica e engenharia de *software* para se referir ao tempo, esforço e conhecimento que um novo usuário deve investir para obter o domínio de um *software*. Apesar do MoveIt possuir um assistente de configuração com uma interface gráfica amigável, o ROS ainda tem muito a melhorar nesse quesito. A curva de aprendizagem do ROS é extremamente lenta e exaustiva, mesmo com os inúmeros exemplos na internet e uma comunidade open-sorce ativa, a interação com o terminal nos dias de hoje acaba afastando o usuário comum desse tipo de *software*. A incompatibilidade nativa do ROS com o Windows também afeta a popularização e a entrada de novos usuários.

Outro problema é que a medida que o Moveit se torna mais popular, novos usuários não terão a amplitude de conhecimento para personalizar todos os aspectos do conjunto de ferramentas, o que reforça a necessidade de uma constante evolução gráfica do sistema.

6.2 Trabalhos futuros

Durante o desenvolvimento, alguns parâmetros importantes para a dinâmica do robô foram deixados de lado para contextualizar melhor outros temas abordados no projeto. Um dos parâmetros é a densidade, centro de massa e momento de inércia de cada junta do robô. O modelo 3D exportado do Onshape foi considerado inteiro feito de ABS cuja densidade é de $1.052e6(kg/mm^3)$. Todavia, seu centro de massa e momento de inercia estão desfasados do modelo real de um robô UR10 bem como a tocha de solda anexada à sua flange.

Na camada de *software*, a proposta de uma sexta aplicação poderia abordar um movimento relativo ao TCP. Este tipo de aplicação é bastante comum no ambiente industrial, principalmente para robôs com ferramentas anexadas à flange.

A aplicação de captura dos pontos pode servir como base para desenvolver uma aplicação real de solda. Contudo, um embasamento teórico sobre aplicações de soldas deve ser levado em consideração na construção do código, para que o próprio programa tome as decisões de trajetória para interpolar os pontos.

Em relação a interface de *hardware*, faltou investigar no *datasheet* do *drive* quais os comandos que permitem determinar a velocidade e aceleração individual dos motores.

Por fim, seria interessante migrar o atual projeto para o ROS2, uma vez que no período de conclusão do mesmo a versão estável e recomendada do ROS passou a ser o Foxy 2.1 LTS. No site oficial do ROS2 já existe algum material para o processo de migração dos pacotes ROS1.

REFERÊNCIAS

CARVALHO, P. T. N. Simulação e controle cinemático de um manipulador serial usando o moveit. **Universidade Federal de Santa Catarina**, 2020.

Chitta, S.; Sucan, I.; Cousins, S. Moveit! [ros topics]. **IEEE Robotics Automation Magazine**, v. 19, n. 1, March 2012.

CRAIG, J. J. **Robótica**. 3. ed. : Pearson Universidades, 2013.

Deng, H.; Xiong, J.; Xia, Z. Mobile manipulation task simulation using ros with moveit. July 2017.

DUARTE, P. V. **ROS - Five Dof Robot Arm**. 2019. <https://github.com/paulovictor237/ROS-five-dof-arm>. (Acesso em: 24 de mar. de 2020).

Hernandez-Mendez, S. et al. Design and implementation of a robotic arm using ros and moveit! Nov 2017.

Hernandez-Mendez, S. et al. A switching position/force controller for two independent finger gripper over ros. 2017.

JOSEPH, L.; CACACE, J. **Mastering ROS for Robotics Programming**. 2. ed. : Packt Publishing, 2018.

Kavraki Lab. **The Open Motion Planning Library**. 2012. <https://ompl.kavrakilab.org>. (Acesso em: 23 de mar. de 2020).

Kuffner, J. J.; LaValle, S. M. Rrt-connect: An efficient approach to single-query path planning. v. 2, 2000.

LAUS, L. P. **Introdução à Robótica**. 1. ed. : Universidade Tecnológica Federal do Paraná, 2012.

MOVEIT. **IKFast Kinematics Solver**. 2020. <http://docs.ros.org/melodic/api/moveit_tutorials/html/doc/ikfast/ikfast_tutorial.html#what-is-ikfast>. (Acesso em: 29 de set. de 2020).

MOVEIT. **MoveIt! Concepts**. 2020. <https://moveit.ros.org/documentation/concepts>. (Acesso em: 18 de jun. de 2020).

MOVEIT. **MoveIt Quickstart in RViz**. 2020. <https://ros-planning.github.io/moveit_tutorials/doc/quickstart_in_rviz/quickstart_in_rviz_tutorial.html>. (Acesso em: 06 de abr. de 2020).

MOVEIT. **MoveIt Tutorials**. 2020. <https://ros-planning.github.io/moveit_tutorials>. (Acesso em: 11 de out. de 2020).

MOVEIT. **Time Parameterization**. 2020. <https://ros-planning.github.io/moveit_tutorials/doc/time_parameterization/time_parameterization_tutorial.html#speed-control>. (Acesso em: 18 de nov. de 2020).

PEREIRA, A. B. M. Rosremote: Utilizando ros para acesso remoto a robôs. **Universidade Federal de Itajuba**, março 2018.

QUIGLEY, M. et al. Ros: an open-source robot operating system. 2009.

ROBOTS, U. **Parameters for calculations of kinematics and dynamics**. 2020. <https://www.universal-robots.com/articles/ur/parameters-for-calculations-of-kinematics-and-dynamics/>. (Acesso em: 08 de ago. de 2020).

ROS. **ROS Tutorials**. 2020. <http://wiki.ros.org/ROS/Tutorials>. (Acesso em: 11 de out. de 2020).

SCIAVICCO, L. et al. **Robotics: Modelling, Planning and Control**. 1. ed. : Springer, 2010.

WALTER, L. **Roscore**. 2019. <http://wiki.ros.org/roscore>. (Acesso em: 31 de mar. de 2020).

YOUAKIM, D. **MoveIt! based Implementation of an I-AUV**. : 2015. CIRS: Underwater Robotics Research Center (VICOROB) University of Girona, 2015.

Youakim, D. et al. Moveit!: Autonomous underwater free-floating manipulation. **IEEE Robotics Automation Magazine**, v. 24, n. 3, Sep. 2017.

APÊNDICE A

Este apêndice apresenta o script de Matlab com o pacote Symbolic Math Toolbox, para obter a cadeia cinemática à partir dos parâmetros de Denavit-Hartenberg. Para verificar se este pacote está instalado no Matlab, basta digitar o comando "ver".

```
%       a alpha d theta
function [A,z,p] = DH(m)
        syms z p;
        A = eye(4,4);
        for i=1:size(m,1)
            z(1:3,i)=A(1:3,3);
            theta=m(i,4);
            d=m(i,3);
            rz = Rzz(theta,d);

            alpha=m(i,2);
            a=m(i,1);
            rx = Rxx(alpha,a);

            A = A * rz *rx;
            A = simplify (A);
            p(1:3,i+1)=A(1:3,4);
        end
        p(1:3,1)=[0; 0; 0];
end

function r = Rzz(theta,d)
    ct = cos(theta);
    st = sin(theta);
    r = [ ct  -st  0   0
          st   ct  0   0
          0    0   1   d
          0    0   0   1];
end

function r = Rxx(alpha,a)
    ct = cos(alpha);
    st = sin(alpha);
     r = [ 1   0   0    a
           0   ct -st   0
           0   st  ct   0
           0   0   0    1];
end
```

APÊNDICE B

Este apêndice apresenta a simulação da cadeia cinemática através de um script de Matlab com o pacote Robotics Toolbox (PeterCorke).

```
clear;clc;
syms theta1 theta2 theta3 theta4;
syms theta4 theta5 theta6;
syms a1 a2 a3 a4;
syms d1 d3 d3;
syms px py pz;
%Robo ArmAgeddon
%       a         alpha     d         theta
M1 =  [ 0         pi/2      1.273     theta1
        -612      0         0         theta2
        -5.723    0         0         theta3
        0         pi/2      163.941   theta4
        0         -pi/2     1.157     theta5
        0         0         922       theta6];
%CINEMATICA DIRETA
CD = DH(M1)
%SIMPLIFICAR
for i=1:size(CD,1)
    for j=1:size(CD,2)
        CD(i,j)=simplify(CD(i,j));
    end
end
CD
%SIMULACAO DO ROBO
%(necessario o pacote petercorke)
L1 = Link('a',0.0     ,'d', 0.1273     ,'alpha', pi/2     );
L2 = Link('a',-0.612  ,'d', 0.0        ,'alpha', 0.0    );
L3 = Link('a',-0.5723 ,'d', 0.0       ,'alpha', 0.0    );
L4 = Link('a',0.0     ,'d', 0.163941   ,'alpha', pi/2     );
L5 = Link('a',0.0     ,'d', 0.1157     ,'alpha', -pi/2    );
L6 = Link('a',0.0     ,'d', 0.0922     ,'alpha', 0.0    );
L7 = Link('a',0.0     ,'d', 0.430726  ,'alpha', -2.46443  );
L8 = Link('a',0.0     ,'d', 0.0       ,'alpha', -pi/2     );
bot = SerialLink([L1 L2 L3 L4 L5 L6 L7 L8], 'name', 'ArmAgeddon')
bot.teach([0 -pi/2 0 -pi/2 0 0 0 pi],'jointdiam', 0);
```

APÊNDICE C

Para contribuir com a comunidade acadêmica e auxiliar projetos futuros o pacote de desenvolvimento deste trabalho foi hospedado na plataforma web GitHub, e pode ser acessado no endereço do projeto "ArmAgeddon[1]". Esta página descreve todo o trabalho de forma objetiva, com uma linguagem direcionada para o usuário. Os seguintes tópicos são abordados:

1. Requisitos de software;
2. Instalação;
3. Aplicações;
4. Execução;
5. ROS GUI;
6. Captura de pontos pelo simulador;
7. Extra.

Infelizmente é comum para os desenvolvedores do ROS se depararem com *Bugs* e quebra de pacotes. Contudo, em alguns casos a solução mais fácil é a reinstalação completa do ROS e do MoveIt. Para contornar o exaustivo processo de instalação e ajustar problemas recorrentes, o autor deste projeto também desenvolveu um "bash-script" hospedado no site GitHub, com acesso em "BashScript Ros-Moveit[2]". Este programa executa os seguintes passos:

1. Adiciona comandos ao arquivo "bashrc";
2. Instala o ROS Melodic Morenia;
3. Instala o MoveIt Melodic 1.0 LTS;
4. Cria um *workspace* na pasta "ws_moveit";
5. Compila o *workspace* com o "catkin build".

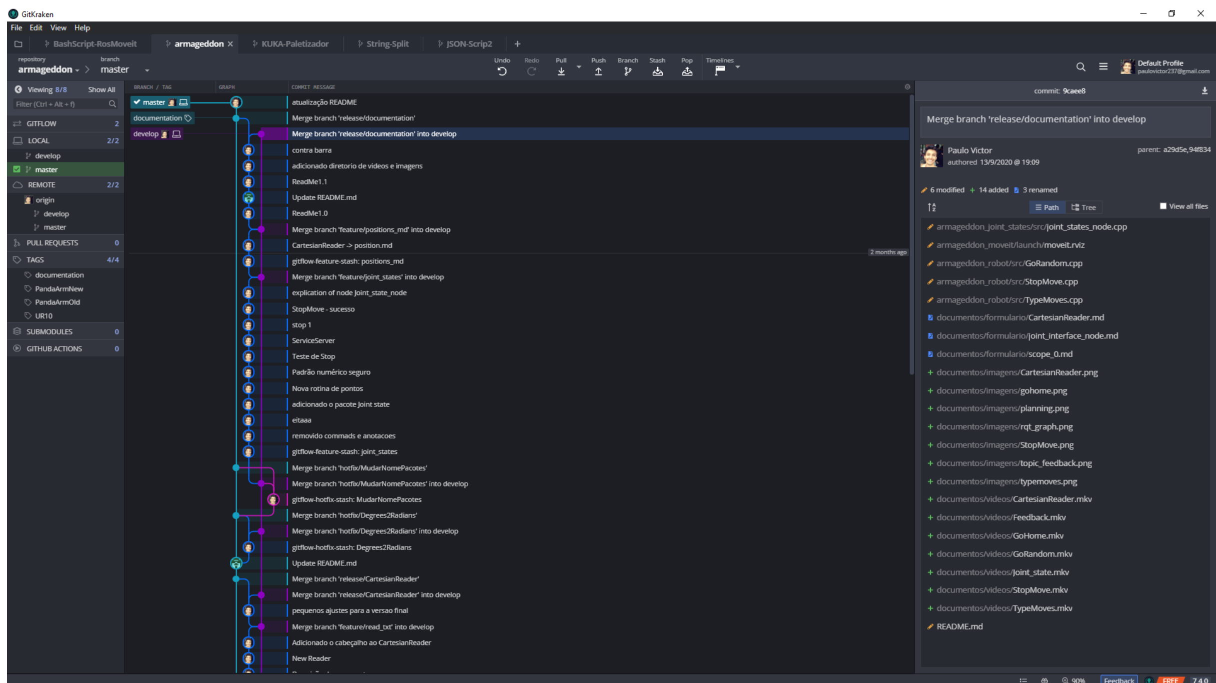
Para o desenvolvimento de todo o projeto foi essencial o uso do "GitKraken[3]" para o controle de versões de software. É recomendado baixar este programa para navegar raiadamente entre as *Branches* como demonstrado na Fig. 26.

Outro programa que contribuiu para acelerar a programação do trabalho é o "Visual Code[4]". Neste, podemos baixar extensões que adicionam suporte e destaque de sintaxe para várias linguagens de programação utilizadas no projeto como URDF, Markdown, CMake, C++, OpenSCAD, YAMAL, etc.

---

[1] <https://github.com/paulovictor237/armageddon>
[2] <https://github.com/paulovictor237/BashScript-RosMoveit>
[3] <https://www.gitkraken.com>
[4] <https://code.visualstudio.com>

Figura 26 – Painel com um *drive* e um motor.



Fonte: Autor (2020)