CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO

RODOLFO LOPES DUTRA

SIMULAÇÃO DE UM PROTÓTIPO DE BRAÇO ROBÓTICO TIPO SCARA COM CONTROLE DE POSICIONAMENTO DE OBJETO

Leopoldina

2026

RODOLFO LOPES DUTRA

SIMULAÇÃO DE UM PROTÓTIPO DE BRAÇO ROBÓTICO TIPO SCARA COM CONTROLE DE POSICIONAMENTO DE OBJETO

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Controle e Automação do Centro Federal de Educação Tecnológica de Minas Gerais, do Campus Leopoldina, como parte do requisito para obtenção do título de Bacharel em Engenharia de Controle e Automação.

Orientador: Fabiano Drumond Chaves

Leopoldina

2026

Dutra, Rodolfo Lopes.
D978s Simulação de um protótipo de braço robótico tipo SCARA com controle de posicionamento de objeto. / Rodolfo Lopes Dutra. – 2026.
112 f. : il., gráfs.; figs.

Orientador: Fabiano Drumond Chaves.
Bibliografia: f. 106-107.

Trabalho de Conclusão de Curso (Graduação) - Centro Federal de Educação Tecnológica de Minas Gerais, *Campus* Leopoldina, apresentada a curso de Engenharia de Controle e Automação. Departamento de Eletroeletrônica, 2026.

1. Controle. 2. Visão computacional. 3. Robôs. 4. Cinética. 5. Simulação. 6. Modelagem. I. Chaves, Fabiano Drumond. II. Título.

CDU: 681.5 :004.94

Elaboração da ficha catalográfica pela bibliotecária Luzia Adriana Damasceno, CRB 6º - 2305 / CEFET-MG *campus* Leopoldina.

MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
COORDENAÇÃO DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO - LP

FOLHA DE APROVAÇÃO DE TCC Nº 2/2026 - CECALP (11.51.20)

Nº do Protocolo: NÃO PROTOCOLADO

Leopoldina-MG, 16 de junho de 2026.

RODOLFO LOPES DUTRA

SIMULAÇÃO DE UM PROTÓTIPO DE BRAÇO ROBÓTICO TIPO SCARA COM CONTROLE DE POSICIONAMENTO DE OBJETO

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Controle e Automação do Centro Federal de Educação Tecnológica de Minas Gerais, do Campus Leopoldina, como parte do requisito para obtenção do grau de Bacharel em Engenharia de Controle e Automação.

Aprovado em 16 de junho de 2026.

Banca Examinadora:

Prof. D. Eng. Fabiano Drumond Chaves (CEFET-MG) - Presidente
Prof. D. Eng. Accacio Ferreira do Santos Neto (CEFET-MG)
Prof. D. Eng. Murillo Ferreira dos Santos (CEFET-MG)

Leopoldina
2026

(Assinado digitalmente em 16/06/2026 16:18)
ACCACIO FERREIRA DOS SANTOS NETO
PROFESSOR ENS BASICO TECN TECNOLOGICO
DEELP (11.61.04)
Matrícula: ###317#7

(Assinado digitalmente em 17/06/2026 08:05)
FABIANO DRUMOND CHAVES
PROFESSOR ENS BASICO TECN TECNOLOGICO
DEMLP (11.61.12)
Matrícula: ###090#5

(Assinado digitalmente em 16/06/2026 15:57)
MURILLO FERREIRA DOS SANTOS
PROFESSOR ENS BASICO TECN TECNOLOGICO
DEELP (11.61.04)
Matrícula: ###196#6

Processo Associado: 23062.030925/2026-01

Visualize o documento original em https://sig.cefetmg.br/public/documentos/index.jsp informando seu número: **2**, ano: **2026**, tipo: **FOLHA DE APROVAÇÃO DE TCC**, data de emissão: **16/06/2026** e o código de verificação: **9fc399d0b4**

Dedico esse trabalho a minha familia, aos professores, aos meus colegas de laboratório e a todos que estiveram comigo durante o processo desse trabalho a todos o meus sinceros agradecimentos, mas mais importante é a minha companheira que me ajudou mais que todos durante o desenvolvimento desse projeto.

AGRADECIMENTOS

Agradeço aos professores que estiveram comigo me ajudando durante o trabalho, principalmente os professores responsáveis pelo laboratório SIRO que permitiram utilizar do mesmo.

RESUMO

Este trabalho apresenta o desenvolvimento e a validação em ambiente simulado de um robô *Selective Compliance Assembly Robot Arm* (Braço Robótico para Montagem de Conformidade Seletiva) (SCARA) aplicado a uma tarefa de *pick and place*. O projeto foi desenvolvido utilizando *Robot Operating System* (Sistema Operacional de Robôs) (ROS), *Gazebo*, *MoveIt* e visão computacional com *OpenCV*. Foram realizadas a modelagem matemática do manipulador, incluindo propriedades físicas, cinemática direta e inversa, além do projeto e ajuste dos controladores Proporcional, Integral e Derivativo (PID). O sistema de visão foi implementado para identificar o objeto por meio da câmera e converter sua posição para o sistema de referência do mundo, permitindo que a cinemática inversa calculasse as posições articulares necessárias para a manipulação. Os controladores foram avaliados por ganhos calculados, empíricos e ajustados, sendo observado bom desempenho nas juntas rotacionais e necessidade de ajuste empírico na junta prismática. Ao todo, foram realizados 30 testes, nos quais falhas de trajetória, altura de deposição e controle do fuso foram identificadas e corrigidas. Após os ajustes, o robô executou corretamente a tarefa, apresentando erro cartesiano final de aproximadamente $4{,}37\,mm$. Os resultados demonstram que a integração entre simulação, controle, planejamento de movimento, cinemática inversa e visão computacional foi funcional para validar a aplicação proposta em ambiente virtual.

Palavras-chave: Visão computacional; cinemática inversa; modelagem 3D; simulação; braço robótico.

ABSTRACT

This work presents the development and validation—in a simulated environment—of a SCARA robot applied to a pick-and-place task. The project was developed using ROS, Gazebo, MoveIt, and computer vision with OpenCV. The manipulator's mathematical modeling was performed—covering physical properties as well as forward and inverse kinematics—alongside the design and tuning of PID controllers. A vision system was implemented to identify the object via camera and convert its position to the world reference frame, enabling the inverse kinematics module to calculate the joint positions required for manipulation. The controllers were evaluated using calculated, empirical, and fine-tuned gains; good performance was observed in the rotational joints, while empirical tuning was required for the prismatic joint. A total of 30 tests were conducted, during which issues regarding trajectory, deposition height, and lead screw control were identified and corrected. Following these adjustments, the robot successfully executed the task, exhibiting a final Cartesian error of approximately 4.37 mm. The results demonstrate that the integration of simulation, control, motion planning, inverse kinematics, and computer vision effectively validated the proposed application within the virtual environment.

Keywords: Computer vision; inverse kinematics; 3D modeling; simulation; robotic arm.

MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
COORDENAÇÃO DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO - LP

TERMO DE RESPONSABILIDADE Nº 131/2026 - CECALP (11.51.20)

Nº do Protocolo: NÃO PROTOCOLADO

Leopoldina-MG, 16 de junho de 2026.

Eu, **RODOLFO LOPES DUTRA**, matrícula **201723510165**, discente do Curso de Engenharia de Controle e Automação, declaro para os devidos fins, que o presente Trabalho de Conclusão de Curso, de título **SIMULAÇÃO DE UM PROTÓTIPO DE BRAÇO ROBÓTICO TIPO SCARA COM CONTROLE DE POSICIONAMENTO DE OBJETO**, é de minha autoria e que estou ciente dos:

- Artigos 297 a 299 do Código Penal;
- Decreto-Lei n.º 2.848 de 7 de dezembro de 1940;
- Lei n.º 9.610, de 19 de fevereiro de 1998, sobre os Direitos Autorais;
- Regulamento Disciplinar Discente do CEFET-MG;
- E que plágio consiste na reprodução de obra alheia e submissão dela como trabalho próprio ou na inclusão, em trabalho próprio, de ideias, textos, tabelas ou ilustrações (quadros, figuras, gráficos, fotografias, retratos, lâminas, desenhos, organogramas, fluxogramas, plantas, mapas e outros) transcritos de obras de terceiros sem a devida e correta citação da referência.

Por ser verdade, e por ter ciência do referido artigo, firmo a presente declaração, isentando o (a) professor(a) **Prof. D. Eng. Fabiano Drumond Chaves** (orientador), os membros da banca examinadora e o Centro Federal de Educação Tecnológica de Minas Gerais, Campus Leopoldina, de qualquer responsabilidade.

(Assinado digitalmente em 23/06/2026 08:39)
Rodolfo Lopes Dutra
DISCENTE
Matrícula: 2017#######5

Processo Associado: 23062.030925/2026-01

Visualize o documento original em https://sig.cefetmg.br/public/documentos/index.jsp informando seu número: **131**, ano: **2026**, tipo: **TERMO DE RESPONSABILIDADE**, data de emissão: **16/06/2026** e o código de verificação: **0d9d0ff4c7**

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CMD Prompt de Comando

PID Proporcional, Integral e Derivativo

ROS *Robot Operating System* (Sistema Operacional de Robôs)

RViz *Robot Operating System Visualization* (Visualização do Sistema Operacional de Robôs)

SCARA *Selective Compliance Assembly Robot Arm* (Braço Robótico para Montagem de Conformidade Seletiva)

URDF *Unified Robot Description Format* (Formato Unificado de Descrição de Robôs)

LISTA DE SÍMBOLOS

π	Pi
θ	Theta
ζ	Zeta
ω	Omega

SUMÁRIO

1 INTRODUÇÃO

Com o avanço das tecnologias, a automação industrial tem se tornado cada vez maior e mais fundamental na manufatura moderna. Ela tem contribuído de forma consistente para otimizar processos, trazendo maior eficiência, produtividade e padronização dos processos industriais. Os robôs vêm sendo introduzidos no meio industrial com objetivos claros: aumentar o desempenho e a segurança dos processos, consequentemente diminuindo riscos e aumentando a produtividade. O uso desses sistemas possibilita que as tarefas repetitivas, perigosas ou as que exijam uma alta precisão sejam realizadas com mais agilidade e qualidad parareduzir o esforço humano em atividades insalubres, além de melhorar o desempenho geral das operações (TAY; CHOONG; YOONG, 2022).

Com novas tecnologias sendo desenvolvidas, diversos tipos e modelos de manipuladores robóticos foram sendo desenvolvidos. Entretanto, os robôs do tipo SCARA se destacam por sua estrutura simples, composta por três eixos rotativos e um eixo prismático. A estruturação do robô SCARA o torna muito adequado para aplicações de montagem, manipulação de peças e processos do tipo *pick-and-place*, em que precisão e velocidade de operação são fatores determinantes. Por ser um robô compacto e mais simples de controlar, ele foi empregado em diversos nichos, sejam eles industriais, acadêmicos e até mesmo laboratoriais (SHARIATEE et al., 2014).

O deslocamento e posicionamento de objetos para locais predeterminados definem o desempenho de um bom controle de posicionamento; ele é um dos mais relevantes no desempenho de manipuladores robóticos. A partir dele é definida a precisão com que o sistema pode executar certas tarefas. Analisando o contexto industrial, esse tipo de controle necessita ser executado da forma correta para que assim possa garantir confiabilidade, repetibilidade e uma maior redução de desperdícios. O estudo dessa área representa um campo essencial para o avanço da robótica na automação.

Tendo em vista o contexto industrial atual e a crescente utilização de sistemas robóticos em processos de automação, o objetivo principal deste trabalho é desenvolver, desde suas etapas iniciais, uma simulação de um braço robótico com características de um modelo SCARA, voltado ao controle, posicionamento e manipulação de objetos. Mais do que apenas implementar uma simulação funcional, o projeto busca compreender o passo a passo necessário para a estruturação de um manipulador robótico industrial, abrangendo desde a modelagem da estrutura mecânica até a integração entre ferramentas de simulação, planejamento de movimento, controle

e visão computacional. Dessa forma, o trabalho também tem como finalidade aproximar o ambiente acadêmico de conceitos aplicados na indústria, possibilitando o aprendizado prático de tecnologias utilizadas no desenvolvimento de sistemas robóticos.

Para alcançar esse objetivo, foram empregados diferentes softwares e ferramentas amplamente utilizadas na área de robótica. O ROS foi adotado como plataforma principal de integração, permitindo a comunicação entre os módulos de controle, simulação e visão computacional por meio de uma arquitetura modular e distribuída. O ROS é um *middleware* de código aberto para o desenvolvimento de software robótico, executando sobre o sistema operacional Ubuntu Linux. Ele possui um conjunto de bibliotecas e ferramentas que auxiliam na construção de sistemas robóticos, conhecidos por seu design modular e escalável. Dado um modelo de ambiente, o planejamento de tarefas preocupa-se com a composição de ações estruturadas para atingir metas específicas, minimizando custos como tempo ou energia (CASHMORE et al., 2015).

O MoveIt foi utilizado como planejador de trajetórias, sendo este um *framework* de planejamento e controle e movimento para robôs manipuladores no ROS, combinando módulos de cinemática, percepção 3D e controle de movimento, permitindo gerar trajetórias livres de colisão em ambientes não estruturados. Ele se conecta a outros nós do sistema para integrar as funções de planejamento, percepção e controle, oferecendo uma interface robusta para o ROS (CHITTA; SUCAN; COUSINS, 2012).

O simulador Gazebo, desenvolvido originalmente em 2002, foi empregado para a simulação física do ambiente e validação dos movimentos do robô sob diferentes condições. Sua integração com o ROS é amplamente utilizada pela comunidade robótica por sua capacidade de simular ambientes complexos com múltiplos robôs de forma coordenada, garantindo realismo, flexibilidade e robustez nas análises (RIVERA; SIMONE; GUIDA, 2019).

O sistema de visão computacional foi desenvolvido em *Python*, permitindo a detecção dos objetos a serem manipulados no ambiente, possibilitando ao robô identificar e manipular os objetos desejados. Isso busca simular um ambiente industrial, em que o robô possa realizar operações de separação, organização ou montagem de peças conforme for estabelecido para ele. Para o controle das juntas do robô, foram utilizados controladores PID, ajustados de forma individual para cada eixo, a fim de garantir a estabilidade e precisão nos movimentos.

Com isso, o trabalho integra todas as etapas de modelagem, simulação e controle, apresentando um panorama completo para o desenvolvimento de um manipulador robótico.

Espera-se que os resultados obtidos contribuam para o avanço de soluções educacionais e experimentais no campo da robótica aplicada, reforçando a importância do controle de posicionamento e da integração entre ferramentas de software e hardware no contexto da automação industrial moderna.

1.1 Problema e Proposta

A automação industrial tem utilizado cada vez mais robôs manipuladores em tarefas de movimentação, separação e posicionamento de objetos. Entretanto, em muitos casos, esses sistemas são programados para executar movimentos em posições previamente definidas, limitando sua flexibilidade quando há variações na posição dos objetos ou alterações no ambiente de trabalho. Diante deste cenário, este trabalho propõe o desenvolvimento de um robô do tipo SCARA em ambiente simulado, integrado a um sistema com visão computacional. O objetivo é permitir que o manipulador identifique a posição de objetos no ambiente simulado e realize os cálculos necessários para sua movimentação de forma autônoma.

A proposta consiste em estruturar, desde as etapas iniciais, um projeto robótico voltado ao contexto industrial, utilizando ferramentas como ROS, MoveIt e Gazebo para modelagem, simulação, planejamento de trajetórias e controle do robô. O sistema, com o auxílio de uma câmera no ambiente simulado, deve obter as coordenadas dos objetos no ambiente do Gazebo, permitindo que o robô execute as tarefas de manipulação como pegar, posicionar e remover objetos de uma determinada área. Desta forma, é possível aproximar o ambiente acadêmico de aplicações reais industriais, demonstrando como a integração entre simulação, percepção visual e controle de movimento podem ampliar a flexibilidade de um manipulador robótico.

Além da movimentação do braço robótico, o trabalho também contempla a configuração dos grupos de juntas, a definição dos limites de movimento, o planejamento de trajetórias livres de colisão e os cálculos de cinemática inversa necessários para o posicionamento do efetuador. Assim, o projeto não se restringe apenas à criação de um modelo em simulação virtual, mas busca apresentar o passo a passo da estruturação de um sistema robótico completo em ambiente simulado, servindo como base para estudos futuros e melhorias no controle de manipuladores robóticos.

1.2 Justificativa

O desenvolvimento deste trabalho é de grande importância para o meio acadêmico, com este projeto é possível compreender todas as etapas envolvidas no desenvolvimento de um manipulador robótico utilizado em aplicações industriais. A criação de um ambiente simulado permite testar a estrutura mecânica, os algoritmos de controle, o planejamento de movimento e a integração com sensores antes de uma possível construção física, reduzindo erros de projeto e facilitando a análise do comportamento do sistema antes da construção fora da simulação.

No contexto acadêmico, o projeto contribui para o aprendizado de conceitos importantes da robótica aplicada, como modelagem de robôs, cinemática inversa, controle de juntas, visão computacional e integração entre diferentes ferramentas de software. Por ser desenvolvido em ambiente simulado, o sistema também pode ser adaptado para diferentes cenários de manipulação, permitindo que novos testes sejam realizados com alterações nos objetos, nas posições de trabalho ou nas estratégias de controle.

Dessa forma, o trabalho apresenta relevância, tanto para o ensino quanto para a pesquisa em automação e robótica, pois oferece uma base para o estudo de sistemas robóticos industriais de maneira acessível e controlada. A integração entre o robô SCARA, a câmera simulada e as ferramentas de planejamento possibilita explorar soluções aplicáveis a tarefas de movimentação, separação e posicionamento de objetos, contribuindo para o desenvolvimento de projetos futuros envolvendo técnicas de controle mais avançadas, inteligência artificial, sensores adicionais e aproximação entre simulação e aplicação real.

1.3 Objetivo Geral

O objetivo geral deste trabalho é desenvolver e validar, em ambiente virtual, um manipulador robótico do tipo SCARA integrado a um sistema de visão computacional, capaz de realizar operações de *pick-and-place* em uma simulação com física aplicada. O sistema busca representar as principais etapas de estruturação de um braço robótico industrial, contemplando a modelagem do robô, o planejamento de trajetórias, o controle dos movimentos e a obtenção das coordenadas dos objetos no ambiente simulado.

1.4 Objetivos específicos

Para alcançar o objetivo geral deste trabalho, foram definidos objetivos específicos relacionados à modelagem, simulação, controle e validação do robô SCARA. Esses objetivos orientaram as etapas de desenvolvimento do projeto, desde a construção do modelo matemático e virtual do manipulador até a implementação da tarefa de *pick and place* com auxílio de visão computacional.

A Tabela 1.1 apresenta os principais objetivos específicos estabelecidos para o desenvolvimento deste trabalho.

Tabela 1.1 – Objetivos específicos do trabalho.

Etapa	Objetivo específico
1	Desenvolver o modelo virtual do robô SCARA no ambiente ROS.
2	Calcular as propriedades físicas e inerciais dos principais componentes do manipulador.
3	Formular a cinemática direta e inversa do robô SCARA.
4	Projetar e configurar controladores PID para as juntas do robô.
5	Integrar o modelo do robô ao *Gazebo*.
6	Configurar o *MoveIt* para o projeto.
7	Implementar um sistema de visão computacional com *OpenCV* para identificação de objetos.
8	Desenvolver uma rotina de *pick and place* integrando visão computacional.
9	Realizar testes de funcionamento do sistema e registrá-los.
10	Avaliar os resultados obtidos, identificando limitações do sistema e possíveis melhorias para trabalhos futuros.

2 ESTADO DA ARTE

A evolução da tecnologia no mundo atual está diretamente relacionada ao aumento da demanda por maior eficiência e precisão nas linhas de produção devido ao avanço dos robôs industriais. Neste cenário, os robôs do tipo SCARA se destacam por sua arquitetura mecânica simplificada e satisfatoriamente eficiente, principalmente quando utilizados em tarefas de montagem, movimentação e manipulação de peças. Na literatura. A literatura sobre esses manipuladores apresenta um progresso significativo, começando pelos estudos de modelagem dinâmica e controle, passando pelo uso de ambientes de simulação e, mais recentemente, integrando sistemas de visão computacional para aumentar a autonomia em atividades industriais.

2.1 Robôs SCARA

2.1.1 Histórico e características

O robô SCARA foi desenvolvido por Makino e Furuya em 1981, inspirado no funcionamento de um biombo japonês em que sua estrutura apresenta flexibilidade no plano horizontal e rigidez vertical. A partir desse conceito, o manipulador SCARA passou a utilizar a chamada conformidade seletiva, característica que permite pequenas acomodações no plano horizontal durante operações de montagem ao mesmo tempo em que mantém rigidez suficiente no eixo vertical para garantir estabilidade e precisão. Essa combinação tornou o robô SCARA muito adequado para tarefas de inserção, montagem e manipulação de componentes, nas quais movimentos rápidos e precisos no plano $x - y$ são combinados com deslocamentos verticais controlados por uma junta prismática. Desde sua criação, o SCARA tem sido amplamente estudado em razão de suas características cinemáticas e dinâmicas, bem como de sua aplicação em processos industriais que exigem precisão e tarefas repetidas (MAKINO, 2014).

A simplicidade mecânica do robô SCARA contribuiu para sua utilização em larga escala nas linhas de produção, considerando sua capacidade de realizar movimentos rápidos e controlados. Sua estrutura permite que o manipulador execute tarefas de *pick-and-place*, montagem, separação e posicionamento de peças com boa eficiência. Dessa forma, o SCARA se tornou um dos manipuladores mais estudados no campo da robótica industrial, servindo de base para diversos estudos relacionados às práticas de controle e automação.

2.1.2 Modelagem e controle

Após sua introdução, diversos estudos passaram a investigar formas de modelagem, simulação e controle dinâmico desse tipo de manipulador. Em 1986, Calleja, Marinero e Martinez abordaram aspectos relacionados ao controle do robô, contribuindo para a compreensão de sua integração em sistemas automatizados de produção(CALLEJA; MARINERO; MARTINEZ, 1986). Posteriormente, em 1988, Lee, Lee e Lim utilizaram o método de Runge-Kutta para integrar as equações de movimento e calcular acelerações e torques em diferentes trajetórias, desenvolvendo pacotes de simulação voltados à análise dinâmica do robô(LEE; LEE; LIM, 1988). No mesmo período, Ibrahim, Cook e Tieu empregaram trajetórias polinomiais associadas ao modelo lagrangiano para simular o comportamento do manipulador(IBRAHIM; COOK; TIEU, 1988). Esses trabalhos demonstraram que a modelagem dinâmica completa do SCARA envolve elevada complexidade.

Os estudos iniciais também evidenciaram características importantes do comportamento dinâmico do SCARA. Conforme discorrido por Padhy, os torques gerados nos dois primeiros elos podem apresentar valores significativamente diferentes, sendo o torque do primeiro elo geralmente superior ao do segundo. Esse comportamento mostra que seu controle exige uma análise cuidadosa da dinâmica do sistema, apesar da aparente simplicidade estrutural do manipulador. Assim, a modelagem, a simulação e a otimização do controle permaneceram como temas relevantes para pesquisadores, principalmente com o objetivo de melhorar a precisão, a velocidade e a eficiência do robô em aplicações industriais (PADHY, 1992).

Com o avanço das tecnologias de controle e modelagem, o estudo dos robôs SCARA passou a se concentrar também na comparação entre diferentes estratégias de controle. Visioli e Legnani (2002) investigaram controladores descentralizados, como PID e modos deslizantes, além de controladores baseados em modelo, como o controle por torque computado e o controle por redes neurais. Os resultados experimentais mostraram que os controladores descentralizados podem apresentar desempenho satisfatório em diversas aplicações, enquanto estratégias baseadas em redes neurais podem melhorar o rastreamento em trajetórias de alta velocidade, reduzindo erros de posicionamento. Esse estudo contribuiu para ampliar a compreensão sobre as vantagens e limitações de diferentes técnicas de controle aplicadas a manipuladores industriais (VISIOLI; LEGNANI, 2002).

Alguns desafios estruturais e operacionais continuaram sendo observados em manipuladores do tipo SCARA e em arquiteturas relacionadas, apesar dos avanços obtidos. Entre esses desafios,

se destacam a possibilidade de colisões internas e a variação do comportamento do sistema em função da orientação da ferramenta. Com o objetivo de superar essas limitações, Krut e sua equipe propuseram, em 2003, uma nova arquitetura mecânica denominada I4, mantendo vantagens de projetos anteriores, como o H4, mas reduzindo problemas associados a colisões internas e melhorando o desempenho em movimentos de alta velocidade. Nesse contexto, a matriz Jacobiana desempenhou papel importante na análise e no desempenho da nova arquitetura, evidenciando como o projeto mecânico influencia diretamente o comportamento cinemático e dinâmico do sistema (KRUT et al., 2003).

Em 2005, Das e Dulger desenvolveram um modelo matemático completo para o robô SCARA Serpent 1, incluindo a dinâmica dos atuadores, motores de corrente contínua e controladores PD. Por meio de simulações numéricas e experimentos práticos, os autores demonstraram a eficácia do modelo na previsão do comportamento do sistema, destacando a importância do ajuste adequado dos parâmetros de controle para a melhoria do desempenho, especialmente em operações de *pick-and-place*. A boa relação entre os resultados simulados e experimentais reforçou a relevância da combinação entre modelagem matemática, simulação e validação prática no desenvolvimento de sistemas robóticos mais precisos e eficientes (DAS; DÜLGER, 2005).

2.1.3 Simulação com ROS, Gazebo e MoveIt

Os ambientes de simulação passaram a desempenhar papel fundamental no desenvolvimento de manipuladores robóticos por consequência do avanço das ferramentas computacionais. Considerando este contexto, o uso do ROS e do Gazebo se tornou relevante para a criação, teste e validação de sistemas robóticos modernos. Qian (2014), destacou a integração dessas plataformas para simular manipuladores robóticos em ambientes tridimensionais, com foco na execução de tarefas como *grasp and place*. O ROS, por meio de sua comunicação baseada em tópicos e mensagens, facilita a integração entre diferentes componentes do sistema, enquanto o Gazebo possibilita uma simulação física mais realista antes da implementação em um robô físico. A utilização do formato *Unified Robot Description Format* (Formato Unificado de Descrição de Robôs) (URDF) para a modelagem do robô e de controladores de trajetória para a execução de movimentos precisos demonstra a importância dessas ferramentas na redução do tempo de desenvolvimento e aprimoramento dos algoritmos de controle (QIAN et al., 2014).

Complementando essa abordagem, Yang, em 2017, apresentou a construção de uma plataforma de simulação para um robô SCARA utilizando o ROS e a ferramenta de visualização

tridimensional *Robot Operating System Visualization* (Visualização do Sistema Operacional de Robôs) (RViz). O estudo destacou o uso do MoveIt!, uma biblioteca de planejamento de movimento integrada ao ROS, que se tornou um recurso importante para o controle de manipuladores robóticos e para a execução de tarefas de *pick-and-place*. Essa integração permite validar algoritmos de planejamento antes de sua aplicação em sistemas físicos.

Mena, em 2020, explorou o uso do ROS no controle de um robô SCARA de código aberto, evidenciando como a combinação entre ROS, programação em Python e simulação pode tornar o processo de construção e validação mais eficiente. Esse tipo de abordagem contribui para a democratização da robótica e da automação, pois permite que sistemas robóticos sejam desenvolvidos, testados e aprimorados com menor custo e maior flexibilidade (MENA, 2020).

2.1.4 Visão computacional aplicada

A integração de sistemas de visão computacional passou a ter grande importância no avanço dos robôs SCARA, principalmente em tarefas de *pick-and-place*. Em 2016, Andhare destacou o uso de uma câmera posicionada acima da área de trabalho para detectar objetos e converter coordenadas da imagem para o espaço real por meio de transformações bidimensionais. Essa abordagem simplifica o sistema ao reduzir a necessidade de sensores adicionais e permite que o robô realize tarefas de manipulação com maior precisão e eficiência em ambientes industriais (ANDHARE; RAWAT, 2016).

Posteriormente, Huang, em 2018, demonstrou como câmeras de profundidade, combinadas com técnicas de aprendizado profundo e servo-controle visual, podem permitir que robôs autônomos realizem tarefas em tempo real com maior precisão. A utilização de câmeras para percepção tridimensional do ambiente contribui para a detecção, o reconhecimento e a estimativa da pose dos objetos, tornando o sistema mais eficiente e acessível (HUANG; MOK, 2018). De forma semelhante, Chaudhary, em 2019, propôs um sistema de servo-controle visual utilizando uma única câmera para estimar a profundidade e otimizar o planejamento de trajetórias, tornando o manipulador mais adaptável em ambientes dinâmicos (CHAUDHARY; ZAKHAMI; ROY, 2019).

Dessa forma, é possível observar que a evolução dos robôs SCARA parte de sua concepção mecânica, passa pelo desenvolvimento de modelos dinâmicos e estratégias de controle, avança para o uso de plataformas de simulação e chega à integração com sistemas de visão computacional. A combinação entre ferramentas como ROS, Gazebo, MoveIt! e técnicas de

percepção visual amplia as possibilidades de aplicação desses manipuladores, permitindo o desenvolvimento de soluções mais flexíveis, econômicas e eficientes para tarefas de manipulação e automação industrial.

3 MATERIAIS E MÉTODOS

Este capítulo apresenta os métodos utilizados no desenvolvimento da simulação e controle do robô SCARA proposto neste trabalho. Inicialmente, é descrita a preparação do ambiente computacional por meio da instalação do ROS Noetic no sistema Ubuntu 20.04 LTS, bem como a criação do espaço de trabalho Catkin e do pacote responsável por armazenar os arquivos do projeto. Essa etapa foi importante para organizar a estrutura do projeto e permitir a integrar diferentes recursos utilizados ao longo da implementação.

Após a instalação correta do ambiente, são abordados os procedimentos de modelagem do robô, incluindo a criação dos arquivos URDF e Xacro, que representam as propriedades físicas necessárias para representar o manipulador da forma correta ao ambiente de simulação. A visualização inicial do modelo foi realizada no RViz, ele permite validar a estrutura geométrica e o posicionamento das juntas antes da simulação física.

A próxima etapa demonstra a integração do robô ao Gazebo, ele é responsável por considerar propriedades como massa, inércia, gravidade e atuação dos controladores. Nessa etapa, é demonstrada a configuração dos controladores das juntas, os parâmetros PID e os recursos de monitoramento em tempo real por meio das ferramentas RQT, o que possibilita a análise da resposta do manipulador. Também é demonstrado a adição de um objeto ao ambiente de simulação e *plugins* para fixar o objeto nas garras, para que a manipulação ocorresse da melhor forma, além de ser adicionado também uma câmera virtual, permitindo ampliar os testes de manipulação e percepção do sistema.

Além da modelagem e simulação, este capítulo apresenta a configuração do MoveIt, ferramenta empregada para o planejamento de trajetórias e organização dos grupos de movimento do robô e da garra. Também são descritos os fundamentos de cinemática direta e cinemática inversa utilizados no projeto, uma vez que o controle do manipulador depende da relação entre as coordenadas desejadas no espaço de trabalho e os ângulos correspondentes das juntas. Dessa forma, os métodos apresentados estabelecem a base necessária para o funcionamento do robô SCARA, desde sua estruturação no ROS até sua movimentação controlada em ambiente virtual. Todos os códigos utilizados no projeto estão disponíveis para uso pelo link (DUTRA, 2026)

3.1 Instalação do ROS

Para o desenvolvimento deste projeto, foi utilizada a distribuição ROS 1 Noetic Ninjemys, executada em ambiente Ubuntu 20.04 LTS. A escolha dessa distribuição foi por conta de sua compatibilidade com o sistema operacional adotado, essa versão também conta com diversos pacotes disponíveis voltados à robótica e ao amplo suporte encontrado na documentação oficial e na comunidade de desenvolvedores. O ROS, é uma estrutura computacional que pode ser utilizada em diversas aplicações por disponibilizar bibliotecas, ferramentas, padrões de comunicação e recursos que facilitam a integração entre diferentes componentes de um sistema.

Apesar do nome, o ROS não é como um sistema operacional convencional, mas sim como uma camada intermediária de desenvolvimento. Essa camada permite que diferentes partes de um projeto, como sensores, atuadores, algoritmos de controle, modelos cinemáticos, ferramentas de visualização e ambientes de simulação, possam se comunicar de maneira organizada. Essa característica é fundamental em projetos robóticos, pois um robô geralmente depende da atuação conjunta de diversos módulos independentes, que precisam trocar informações em tempo real para executar uma determinada tarefa.

No contexto deste trabalho, o ROS foi utilizado como base para a estruturação do projeto do robô SCARA, permitindo organizar os arquivos de modelagem, configuração, controle e simulação em pacotes. Essa organização em pacotes facilita o desenvolvimento modular, pois cada parte do sistema pode ser criada, modificada e testada de forma separada. Assim, arquivos relacionados à descrição física do robô, configurações de juntas, controladores, visualização e planejamento de movimento podem ser armazenados de maneira padronizada, o que facilita a manutenção e evolução do projeto.

Uma das principais vantagens do uso do ROS está em seu modelo de comunicação. O sistema funciona com base em nós, tópicos, mensagens e serviços, permitindo que diferentes processos sejam executados de forma independente, mas ainda assim conectados dentro de uma mesma aplicação. Em um sistema robótico, isso significa que um nó pode ser responsável por publicar informações sobre a posição de uma junta, enquanto outro nó pode receber esses dados e utilizá-los para controle ou visualização. Essa arquitetura favorece a reutilização de códigos e a integração com diferentes ferramentas, tornando o desenvolvimento mais flexível.

A distribuição ROS Noetic foi escolhida por ser uma das versões mais consolidadas do ROS 1 e por apresentar compatibilidade direta com o Ubuntu 20.04 LTS. Além disso, essa versão

permite o uso de ferramentas importantes para este projeto, como o RViz, o Gazebo e o Moveit. Esses recursos foram essenciais para a modelagem, visualização, simulação e planejamento de movimentos do robô SCARA, permitindo que o comportamento do manipulador fosse analisado em ambiente virtual antes da realização de qualquer aplicação prática.

Dessa forma, a instalação e configuração do ROS representaram uma etapa inicial do projeto. Ao utilizar da integração de suas diferentes ferramentas em um único ambiente de desenvolvimento, foi possível modelar, a visualizar, a simular, o controlar e o planejar o movimento de maneira organizada. Isso ajuda reduz a complexidade do desenvolvimento e permite que cada etapa seja validada progressivamente.

O processo de instalação do ROS Noetic foi realizado com base nas orientações disponibilizadas na documentação oficial do ROS. Essa documentação apresenta os procedimentos necessários para configuração dos repositórios, instalação dos pacotes principais, definição das variáveis de ambiente e validação do funcionamento do sistema. Por se tratar de uma ferramenta amplamente utilizada e em constante dependência de pacotes específicos do sistema operacional, recomenda-se que a instalação seja sempre realizada a partir do site oficial, garantindo que os comandos e dependências estejam compatíveis com a versão utilizada (WIKI, 2020).

Portanto, a utilização do ROS neste trabalho forneceu a base necessária para o desenvolvimento do robô SCARA, desde a organização inicial do projeto até sua integração com ferramentas de simulação e planejamento. A adoção dessa plataforma possibilita a construção de um ambiente robusto, modular e compatível com práticas consolidadas na área de robótica, favorecendo a realização de testes, ajustes e validações ao longo do desenvolvimento do sistema.

3.2 Criação do ambiente de trabalho

Após a instalação do ROS e das dependências iniciais, foi criado o ambiente de trabalho responsável por armazenar e organizar todos os arquivos relacionados ao desenvolvimento do robô SCARA. Esse ambiente recebeu o nome de `Projeto_SCARA` e foi estruturado de acordo com o padrão utilizado pelo ROS 1, baseado em *workspaces Catkin*. Essa organização é importante para permitir que os pacotes do projeto sejam compilados e reconhecidos corretamente dentro do sistema.

No ROS, um espaço de trabalho funciona como a pasta principal do projeto. Dentro dele são armazenados os pacotes principais da aplicação robótica para a modelagem, configuração,

controle, visualização e simulação. Dessa forma, o ambiente `Projeto_SCARA` foi utilizado como base para concentrar todos os arquivos necessários à implementação do manipulador.

Dentro desse espaço de trabalho foi criada a pasta `src`, que é o diretório padrão onde os pacotes do ROS são desenvolvidos. Em seguida, foi criado o pacote principal do projeto, denominado `modelo_scara`. Esse pacote tem a função de reunir os arquivos relacionados a modelagem física e visual do robô, incluindo suas configurações de visualização, arquivos de inicialização, controladores e elementos necessários para integração com as ferramentas do ROS.

Durante a criação do pacote `modelo_scara`, foram adicionadas dependências importantes para o funcionamento do sistema, tal como a `geometry_msgs` que é responsável por fornecer tipos de mensagens geométricas, como posições, orientações e vetores, que são utilizados em aplicações robóticas. A dependência `std_msgs` foi utilizada por disponibilizar mensagens básicas do ROS, permitindo a troca de informações simples entre diferentes nós do sistema.

Também foram adicionadas as dependências `roscpp` e `rospy`, que permitem o desenvolvimento de nós em C++ e Python, respectivamente. Essas bibliotecas são importantes porque possibilitam a criação de códigos responsáveis por controlar o robô, publicar comandos, receber informações dos sensores e interagir com os demais componentes do sistema. Mesmo que parte do projeto seja desenvolvida em apenas uma dessas linguagens, manter ambas como dependências amplia a flexibilidade para futuras implementações.

A dependência `urdf` foi adicionada por ser responsável pelo suporte à descrição do modelo robótico no formato URDF. Esse formato permite representar a estrutura física do robô, incluindo elos, juntas, dimensões, massas, inércias e relações entre os componentes. A partir dessa descrição, o ROS consegue interpretar a estrutura do manipulador e disponibilizá-la para ferramentas de visualização e simulação.

Para permitir a visualização do robô, foram incluídas dependências relacionadas ao RViz, como `rviz`, `robot_state_publisher` e `joint_state_publisher_gui`. O `rviz` disponibiliza o ambiente gráfico de visualização, permitindo observar o modelo do robô, seus eixos de referência e sua estrutura cinemática. O `robot_state_publisher` é responsável por publicar as transformações entre os links do robô com base nos estados das juntas. Já o `joint_state_publisher_gui` permite alterar manualmente os valores das juntas por meio de uma interface gráfica, facilitando os primeiros testes de movimentação e validação do modelo.

Outra dependência importante é o `tf2`, utilizado para gerenciar as transformações entre os diferentes sistemas de coordenadas do robô. Em um manipulador robótico, cada elo e cada junta possui uma posição relativa em relação aos demais componentes. O uso do `tf2` permite que essas relações sejam representadas corretamente, possibilitando a visualização da árvore de transformações e a verificação da posição de cada parte do robô no espaço.

Foram adicionadas também dependências associadas ao controle das juntas, como o `controller_manager` e `joint_state_controller`. O `controller_manager` é responsável por carregar, inicializar e gerenciar os controladores utilizados no sistema, enquanto o `joint_state_controller` permite publicar os estados das juntas do robô. Essas dependências são importantes para as etapas posteriores do projeto, principalmente na integração com o Gazebo, onde o robô passa a ser controlado em um ambiente de simulação física.

Após a criação do pacote `modelo_scara`, foi criada a pasta `urdf`, destinada ao armazenamento dos arquivos de descrição do robô. Nessa pasta foram desenvolvidos os arquivos responsáveis por representar a estrutura do manipulador SCARA, iniciando pelo arquivo `scarav1.urdf`. Esse primeiro modelo foi utilizado como base para a definição inicial dos elos, juntas e dimensões do robô.

Essa organização inicial do ambiente de trabalho é necessária para estabelecer a base do projeto. A partir dela, será possível desenvolver o modelo do robô de forma organizada, capaz de integrar as ferramentas de visualização, preparar os controladores e, após sua validação, avançar para a simulação no Gazebo e para o planejamento de movimentos com o Moveit. Os comandos utilizados para a criação do ambiente de trabalho, do pacote `modelo_scara` e das pastas iniciais do projeto são apresentados no Anexo A.

3.3 Visualização e Configuração do RViz

Após a criação do ambiente de trabalho e da estrutura inicial do pacote `modelo_scara`, foi realizada a configuração do RViz para permitir a visualização do modelo robótico desenvolvido. O RViz é uma ferramenta de visualização integrada ao ROS, utilizada para representar visualmente robôs, sensores, sistemas de coordenadas, trajetórias e demais informações publicadas no ambiente de desenvolvimento. Sua utilização foi fundamental para validar a estrutura geométrica do robô SCARA antes da integração com ambientes de simulação física.

Para organizar a inicialização do modelo no RViz, foi criada uma pasta denominada `launch` dentro do pacote `modelo_scara`. Essa pasta tem o intuito de armazenar arquivos de inicialização do ROS, eles são responsáveis por executar, diferentes nós, parâmetros e configurações de forma conjunta para o funcionamento do sistema. Dessa forma, permite iniciar o modelo do robô e a ferramenta de visualização de maneira padronizada, em vez de chamar cada nó de forma individual.

Para criado adicionar o modelo do robô no RViz para foi desenvolvido o `rviz.launch`. Esse arquivo permite que o modelo seja exibido no ambiente gráfico. Porém antes de executa-lo, é necessário que o ROS reconheça corretamente os pacotes, dependências e arquivos desenvolvidos dentro do projeto para isso é preciso que o terminal esteja associado ao espaço de trabalho criado. Os comandos utilizados para essa etapa são apresentados no Anexo B.

Além da pasta `launch`, também foi criada uma pasta chamada `config`. Essa pasta foi desenvolvida para armazenar dos arquivos de configuração do projeto, incluindo as configurações visuais do RViz. Essa configuração foi realizada para tornar o projeto mais modular e organizado, tornando a estrutura do pacote mais clara e facilitando futuras alterações no projeto.

Ao iniciar o modelo no RViz, foi necessário configurar o ambiente de visualização para que o robô fosse apresentado corretamente. O primeiro passo é definir o *fixed frame*, que representa o sistema de referência fixo utilizado pela ferramenta. Essa configuração é importante porque permite ao RViz compreender qual elemento será considerado como referência principal para posicionar os demais links, juntas e sistemas de coordenadas do robô.

Em seguida, foi adicionado o recurso `RobotModel`, responsável por exibir o modelo robótico a partir da descrição definida nos arquivos URDF. Esse recurso permite visualizar a estrutura física do robô, incluindo seus elos, juntas e relações geométricas. Com isso, é possível verificar se o modelo do robô SCARA estava sendo interpretado corretamente pelo ROS e se sua geometria correspondia à estrutura planejada.

Outro recurso importante adicionado ao ambiente foi o sistema de transformadas, conhecido como *Transform* ou *TF*. Esse recurso permite visualizar os sistemas de coordenadas associados aos diferentes links do robô. Sua utilização é essencial para verificar se as juntas estão posicionadas e orientadas corretamente, uma vez que erros na definição dos eixos podem comprometer o comportamento do manipulador nas etapas posteriores de simulação e controle.

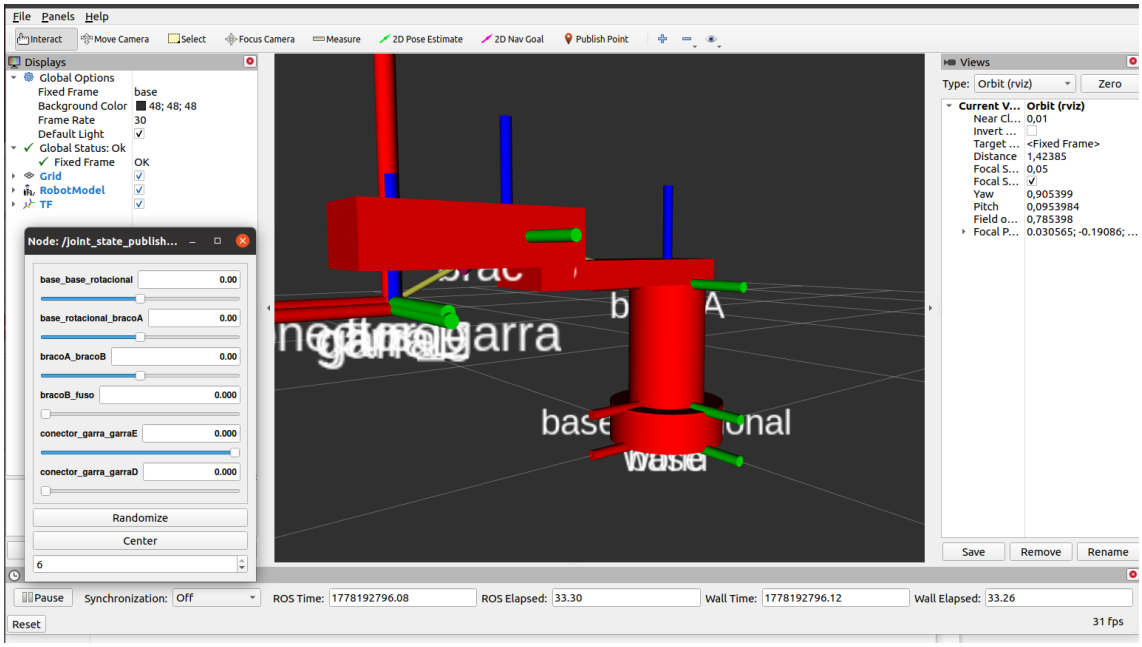
Após a configuração inicial, é necessário salvar as configurações do RViz em um arquivo

dentro da pasta `config`. Nesse projeto, as configurações foram salvas como `config.rviz`, dessa forma, ao adicionar o `config.rviz` dentro do `rviz.launch` e executar novamente o arquivo, o RViz é aberto automaticamente com o enquadramento, os recursos visuais e os parâmetros já definidos. Isso evita que a configuração precise ser refeita manualmente sempre que o programa for iniciado. Para isso, o arquivo de inicialização foi ajustado para carregar diretamente o arquivo de configuração salvo.

Com essa etapa concluída, foi possível visualizar o modelo do robô SCARA, analisar sua estrutura e verificar seus eixos de rotação. Essa validação visual foi importante para identificar possíveis erros de modelagem antes da integração com o Gazebo, que será utilizado para simular o comportamento físico do robô. Diferentemente do Gazebo, o RViz não aplica efeitos físicos, como massa, gravidade, inércia ou colisões, funcionando principalmente como uma ferramenta de visualização e validação geométrica do sistema.

A Figura 3.1 apresenta o modelo visual do robô SCARA carregado no ambiente do RViz.

Figura 3.1 – Modelo visual do robô SCARA.



3.4 Modelagem matemática e métodos aplicados ao robô

Antes de adicionar o robô ao ambiente de simulação do Gazebo, foi necessário utilizar alguns recursos matemáticos relacionados ao comportamento do manipulador SCARA. Essa etapa foi importante para que o modelo desenvolvido não fosse apenas uma representação visual,

mas também uma estrutura capaz de apresentar um comportamento próximo ao de um robô real durante os testes de movimentação, controle e simulação.

A modelagem de um robô manipulador envolve diversas etapas, entre elas a definição de suas propriedades físicas, a descrição da relação entre suas juntas e a posição do *end-effector*, além da escolha de estratégias de controle adequadas para garantir estabilidade e precisão nos movimentos. No caso deste trabalho, foram utilizadas as propriedades inerciais dos elos, a cinemática direta, a cinemática inversa e o controle das juntas por meio de controladores PID.

Com o objetivo de utilizar o Gazebo, para representar um robô com comportamentos mais realistas, foi necessário calcular as propriedades físicas e inerciais, sem elas o modelo não funciona corretamente na simulação. Para realização dos cálculos, os elos foram aproximados por geometrias simples, como cilindros e paralelepípedos, permitindo estimar suas massas e momentos de inércia. Esses valores são fundamentais para a simulação física, pois influenciam diretamente a resposta dinâmica do manipulador aos comandos aplicados.

Além das propriedades físicas, também foi necessário considerar a cinemática do robô. A cinemática direta permite determinar a posição do *end-effector* a partir dos valores conhecidos das juntas. No robô SCARA, essa relação é especialmente importante porque os dois primeiros elos realizam movimentos rotacionais no plano horizontal, enquanto a junta prismática é responsável pelo deslocamento vertical. Dessa forma, a cinemática direta permite compreender como cada junta contribui para a posição final da garra no espaço de trabalho.

A cinemática inversa, por sua vez, realiza o processo oposto. A partir de uma posição desejada para o *end-effector*, ela permite calcular os valores necessários para as juntas do robô. Essa etapa é de extrema importância para que o manipulador consiga alcançar pontos específicos no ambiente, como a posição de um objeto a ser manipulado. No desenvolvimento deste trabalho, a cinemática inversa foi utilizada para relacionar as coordenadas desejadas no espaço cartesiano aos ângulos das juntas rotacionais e ao deslocamento da junta prismática.

Outro aspecto importante considerado foi o controle das juntas. Mesmo que a cinemática indique quais posições devem ser atingidas, é necessário que o robô consiga alcançar essas referências de forma estável. Para isso, foram utilizados controladores PID, responsáveis por reduzir o erro entre a posição desejada e a posição real de cada junta. O ajuste da resposta do sistema por meio das ações proporcional, integral e derivativa, contribui para melhorar a precisão e tornar o movimento mais suave.

Dessa forma, os métodos apresentados nesta seção servem como base para a etapa de simulação no Gazebo. A definição das propriedades inerciais permite que o robô tenha comportamento físico mais adequado, enquanto os estudos de cinemática direta e inversa estabelecem a relação entre as juntas e o movimento do *end-effector*. Por fim, a configuração dos controladores possibilita que o manipulador execute os movimentos planejados de maneira controlada, tornando possível validar o funcionamento do robô SCARA em ambiente virtual.

3.4.1 Cálculo das propriedades físicas e inerciais

Para que o modelo do robô SCARA apresentasse o comportamento correto no ambiente de simulação, foi necessário definir suas propriedades físicas, principalmente a massa e o momento de inércia dos elos. Essas propriedades são fundamentais em simulações realizadas no Gazebo, pois influenciam diretamente a forma como cada componente responde aos esforços aplicados pelos controladores, à ação da gravidade e às interações com o ambiente.

O momento de inércia representa a resistência de um corpo à variação do seu movimento rotacional. Assim, quanto maior for a massa de um componente e quanto mais distante essa massa estiver do eixo de rotação, maior será o esforço necessário para movimentá-lo. Essa informação é importante porque cada elo do manipulador pode ser tratado como um corpo rígido sujeito a rotações ou translações. Dessa forma, ao definir corretamente as propriedades inerciais do sistema, faz com que a simulação apresente respostas mais próximas de um sistema físico real.

No Gazebo, os parâmetros de massa e inércia são utilizados pela simulação para calcular a movimentação dos eixos do robô. Caso esses valores sejam definidos de forma inadequada, o modelo pode apresentar comportamentos instáveis, como oscilações excessivas, movimentos bruscos, quedas inesperadas ou dificuldade para seguir os comandos enviados às juntas. Por esse motivo, antes da implementação do modelo na simulação física, foram realizados os cálculos de inércia dos principais componentes do manipulador.

Para simplificar a modelagem, os elos do robô foram aproximados por geometrias simples, como cilindros e paralelepípedos. Essa aproximação é comum em modelos robóticos, pois permite representar fisicamente os componentes do sistema sem tornar o cálculo excessivamente complexo. Além disso, considera-se que a massa está distribuída uniformemente ao longo de sua estrutura.

Para componentes cilíndricos, como bases, eixos ou elementos rotacionais, o momento

de inércia em torno do eixo central (Z) pode ser calculado por:

$$I_z = \frac{1}{2}mr^2 \tag{3.1}$$

em que I_z representa o momento de inércia em torno do eixo (Z), (m) é a massa do cilindro e (r) é o seu raio. Essa equação é aplicada quando a rotação ocorre em torno do eixo longitudinal do cilindro.

Quando a rotação ocorre em torno de um eixo perpendicular ao eixo central do cilindro, os momentos de inércia em relação aos eixos (X) e (Y) são calculados por:

$$I_x = I_y = \frac{1}{12}m(3r^2 + h^2) \tag{3.2}$$

em que (h) representa o comprimento ou altura do cilindro. Essa formulação permite representar a resistência do componente quando submetido a rotações em eixos diferentes do seu eixo principal.

Para os elementos aproximados por paralelepípedos, como os braços do manipulador, os momentos de inércia em torno dos eixos principais são determinados pelas seguintes equações:

$$I_x = \frac{1}{12}m(y^2 + z^2) \tag{3.3}$$

$$I_y = \frac{1}{12}m(x^2 + z^2) \tag{3.4}$$

$$I_z = \frac{1}{12}m(x^2 + y^2) \tag{3.5}$$

Nessas expressões, (x), (y) e (z) representam as dimensões do corpo em comprimento, largura e altura, respectivamente, enquanto (m) representa a massa do componente. Essas equações são utilizadas considerando que os eixos de rotação passam pelo centro de massa do corpo. Caso o eixo de rotação esteja deslocado em relação ao centro de massa, torna-se necessário aplicar o Teorema dos Eixos Paralelos, também conhecido como Teorema de Steiner, para corrigir o valor da inércia (HIBBELER, 2018).

Como exemplo de aplicação, a base rotacional do robô SCARA foi aproximada por um cilindro. Considerando raio de (0,08,m), comprimento de (0,25,m) e massa de (13,57,kg), o momento de inércia em torno do eixo (Z) foi calculado por:

$$I_z = \frac{1}{2} \cdot 13,57 \cdot (0,08)^2 = 0,0434, kg \cdot m^2 \tag{3.6}$$

Esse valor representa a resistência da base rotacional à variação de seu movimento angular e foi utilizado na descrição física do modelo. De modo semelhante, o braço A foi aproximado por um paralelepípedo com comprimento de (0,42,m), largura de (0,15,m), altura de (0,05,m) e massa de (8,505,kg). Para o eixo (X), o momento de inércia foi calculado por:

$$I_x = \frac{1}{12} \cdot 8,505 \cdot \left((0,15)^2 + (0,05)^2\right) = 0,01772, kg \cdot m^2 \tag{3.7}$$

Após a realizar os cálculos, os valores de massa e inércia foram adicionados no arquivo de descrição do robô. Essa etapa permitiu que o modelo deixasse de ser apenas uma representação geométrica e passasse a conter propriedades físicas necessárias para a simulação dinâmica. Dessa forma, o Gazebo pôde considerar a distribuição de massa dos componentes durante a movimentação do manipulador.

A definição adequada das propriedades inerciais também possui relação direta com o controle das juntas. Como cada junta precisa movimentar um conjunto de elos com determinada massa e inércia, os ganhos dos controladores devem ser ajustados levando em consideração a resposta física do sistema. Assim, valores inadequados de inércia podem dificultar a sintonia do controlador PID, gerar oscilações ou comprometer a estabilidade do robô durante a execução dos movimentos.

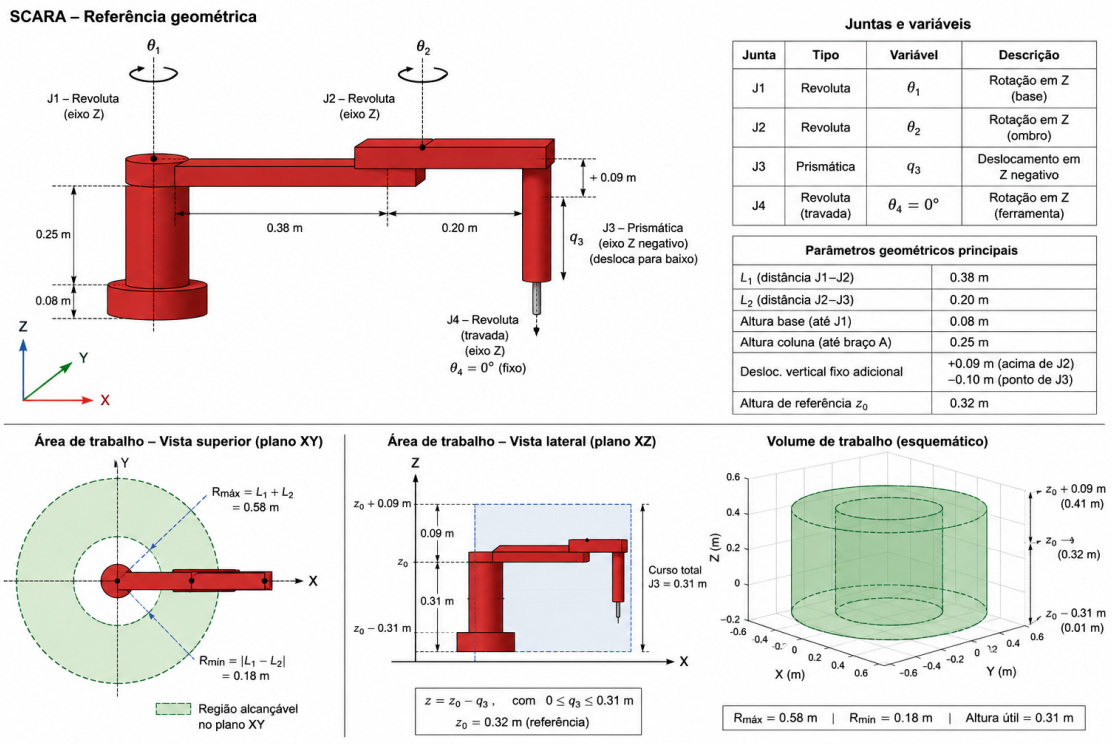
Portanto, o cálculo das propriedades físicas e inerciais constituiu uma etapa essencial da modelagem do robô SCARA. Esses parâmetros forneceram a base necessária para a simulação física no Gazebo e contribuíram para a posterior configuração dos controladores das juntas, permitindo que o manipulador apresentasse comportamento mais estável e coerente durante os testes.

3.4.2 Cinemática direta do robô SCARA

A cinemática direta tem como objetivo determinar a posição e a orientação do *end-effector* a partir dos valores conhecidos das juntas do robô. No caso do robô SCARA desenvolvido neste trabalho, a estrutura é composta por duas juntas revolutas principais, responsáveis pelo deslocamento no plano horizontal, uma junta prismática, responsável pelo movimento vertical, e uma junta final revoluta travada, associada à orientação da ferramenta.

A Figura 3.2 apresenta a referência geométrica utilizada para a modelagem cinemática do robô. Nela são indicadas as principais juntas, as variáveis associadas a cada movimento, os comprimentos dos elos e a região aproximada de trabalho do manipulador.

Figura 3.2 – Referência geométrica, juntas e espaço de trabalho do robô SCARA.



Para a formulação da cinemática direta, foram considerados os parâmetros geométricos principais do robô. O primeiro elo, representado por L_1, corresponde à distância entre a primeira junta revoluta J_1 e a segunda junta revoluta J_2, possuindo comprimento de $0{,}38\,m$. O segundo elo, representado por L_2, corresponde à distância entre J_2 e a junta prismática J_3, possuindo comprimento de $0{,}20\,m$. A altura de referência do sistema é definida por $z_0 = 0{,}32\,m$, enquanto o deslocamento vertical da junta prismática é representado por q_3.

As variáveis articulares consideradas são θ_1, associada à rotação da base, θ_2, associada à rotação do segundo elo, e q_3, associada ao deslocamento vertical da junta prismática. A quarta junta, representada por θ_4, foi considerada travada, com valor fixo de $0°$, uma vez que, nesta etapa do projeto, a orientação final da ferramenta não foi utilizada como variável ativa de controle.

Para a formulação da cinemática direta, foi utilizada uma tabela de transformações homogêneas baseada nos princípios clássicos de modelagem cinemática de manipuladores robóticos (CRAIG, 2005; SPONG; HUTCHINSON; VIDYASAGAR, 2006). Essa abordagem foi adotada porque os parâmetros geométricos específicos do projeto, como comprimentos dos

elos, deslocamentos fixos, tipos de juntas e sentidos de movimentação, já estavam definidos a partir da estrutura do robô SCARA. Dessa forma, em vez de utilizar uma tabela genérica de parâmetros, as transformações foram descritas diretamente de acordo com a sequência real de deslocamentos e rotações do modelo desenvolvido.

A Tabela 3.1 apresenta essas transformações homogêneas, utilizadas para representar a relação entre os elos do robô. Elas descrevem as translações e rotações sucessivas desde a base até o *end-effector*, permitindo obter a posição e a orientação da ferramenta em relação ao sistema de referência da base.

Tabela 3.1 – Tabela de transformações homogêneas do robô SCARA.

Etapa	Junta	Tipo	Transformação
1	`base_base_rotacional`	Revoluta	$T_z(0{,}08)R_z(\theta_1)$
2	`base_rotacional_bracoA`	Fixa	$T_z(0{,}25)$
3	`bracoA_bracoB`	Revoluta	$T_x(0{,}38)T_z(0{,}09)R_z(\theta_2)$
4	`bracoB_fuso`	Prismática	$T_x(0{,}20)T_z(-0{,}10)T_z(-q_3)$
5	`fuso_conector_garra`	Revoluta travada	$R_z(\theta_4)$

Na Tabela 3.1, T_z representa uma translação ao longo do eixo Z, T_x representa uma translação ao longo do eixo X, e R_z representa uma rotação em torno do eixo Z. A primeira transformação descreve a elevação inicial da base e a rotação θ_1. Em seguida, a transformação fixa eleva o sistema até a altura da coluna do robô. A terceira etapa representa o deslocamento ao longo do primeiro elo e a rotação θ_2, enquanto a quarta etapa posiciona o segundo elo e inclui o deslocamento vertical da junta prismática. Por fim, a quinta transformação representa a junta da ferramenta, considerada travada neste modelo.

A transformação homogênea total do sistema pode ser obtida pelo produto sucessivo das transformações individuais, conforme apresentado na Equação 3.8.

$$T_0^f = T_1 T_2 T_3 T_4 T_5 \tag{3.8}$$

Como a junta final foi considerada fixa, com $\theta_4 = 0^\circ$, a orientação final do efetuador depende principalmente da soma das rotações θ_1 e θ_2. Dessa forma, a matriz homogênea resultante que relaciona a base do robô ao *end-effector* é dada pela Equação 3.9.

$$T_0^f = \begin{bmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 & 0{,}38\cos(\theta_1)+0{,}20\cos(\theta_1+\theta_2) \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 & 0{,}38\sin(\theta_1)+0{,}20\sin(\theta_1+\theta_2) \\ 0 & 0 & 1 & 0{,}32-q_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

A partir da matriz apresentada na Equação 3.9, é possível obter diretamente a posição do *end-effector* no espaço cartesiano. As coordenadas x, y e z são extraídas da última coluna da matriz homogênea, resultando nas Equações 3.10, 3.11 e 3.12.

$$x = 0{,}38\cos(\theta_1)+0{,}20\cos(\theta_1+\theta_2) \tag{3.10}$$

$$y = 0{,}38\sin(\theta_1)+0{,}20\sin(\theta_1+\theta_2) \tag{3.11}$$

$$z = 0{,}32-q_3 \tag{3.12}$$

As Equações 3.10 e 3.11 mostram que a posição horizontal do efetuador é determinada pela combinação das rotações das duas primeiras juntas e pelos comprimentos dos elos. A primeira parcela representa a contribuição do primeiro elo, enquanto a segunda parcela representa a contribuição do segundo elo após a rotação acumulada $\theta_1+\theta_2$. Já a Equação 3.12 mostra que a posição vertical depende diretamente do deslocamento da junta prismática, considerando a altura de referência $z_0 = 0{,}32\,m$.

Com base nos comprimentos dos elos, o espaço de trabalho no plano XY pode ser estimado a partir dos alcances máximo e mínimo do manipulador. O alcance máximo ocorre quando os dois elos estão alinhados no mesmo sentido, sendo calculado por:

$$R_{max} = L_1+L_2 = 0{,}38+0{,}20 = 0{,}58\,m \tag{3.13}$$

O alcance mínimo ocorre quando os elos estão dobrados em sentidos opostos, sendo definido por:

$$R_{min} = |L_1-L_2| = |0{,}38-0{,}20| = 0{,}18\,m \tag{3.14}$$

Dessa forma, a região alcançável no plano horizontal pode ser interpretada como uma área anular, limitada internamente por $R_{min} = 0{,}18\,m$ e externamente por $R_{max} = 0{,}58\,m$, conforme indicado na Figura 3.2. Esse limite geométrico influencia diretamente os cálculos de cinemática direta e inversa, pois qualquer ponto desejado fora dessa região não pode ser alcançado apenas pelas rotações das juntas J_1 e J_2.

Além do limite no plano XY, o robô também possui restrição de movimento no eixo vertical. Considerando o deslocamento da junta prismática no intervalo $0 \leq q_3 \leq 0{,}31\,m$, a posição vertical do efetuador é dada por $z = z_0 - q_3$. Assim, quando $q_3 = 0$, o efetuador permanece na altura de referência, e quando $q_3 = 0{,}31\,m$, o efetuador atinge sua posição mais baixa.

$$z_{max} = 0{,}32\,m \tag{3.15}$$

$$z_{min} = 0{,}32 - 0{,}31 = 0{,}01\,m \tag{3.16}$$

Portanto, o espaço de trabalho do robô SCARA é definido pela combinação da região alcançável no plano horizontal com o curso vertical da junta prismática. Essa limitação é fundamental para o planejamento dos movimentos, pois garante que os pontos de destino estejam dentro da capacidade física do manipulador. Além disso, esses limites são utilizados posteriormente na cinemática inversa, uma vez que o cálculo das juntas só é válido para posições que pertençam ao espaço de trabalho do robô.

A cinemática direta, portanto, permite prever a posição do *end-effector* a partir dos valores das juntas. Essa relação é essencial para validar o modelo desenvolvido, verificar a coerência dos movimentos no ambiente de simulação e servir de base para o cálculo da cinemática inversa, que realiza o procedimento oposto: determinar os valores das juntas necessários para alcançar uma posição desejada no espaço.

3.4.3 Cinemática inversa do robô SCARA

A cinemática inversa realiza o procedimento oposto ao da cinemática direta. Enquanto a cinemática direta permite determinar a posição do *end-effector* a partir dos valores das juntas, a cinemática inversa tem como objetivo calcular os valores articulares necessários para que o robô alcance uma posição desejada no espaço cartesiano. No caso do robô SCARA desenvolvido

neste trabalho, esse cálculo é utilizado para determinar os ângulos das juntas revolutas θ_1 e θ_2, além do deslocamento da junta prismática q_3, a partir das coordenadas desejadas X, Y e Z.

A formulação da cinemática inversa foi baseada na geometria do manipulador e nas equações obtidas pela cinemática direta. Conforme apresentado anteriormente, a posição do *end-effector* no plano horizontal é dada pela combinação das rotações das duas primeiras juntas e dos comprimentos dos elos. Assim, considerando $L_1 = 0{,}38\,m$, correspondente ao braço A, e $L_2 = 0{,}20\,m$, correspondente ao braço B, as coordenadas do *end-effector* no plano XY podem ser expressas por:

$$X = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2) \tag{3.17}$$

$$Y = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2) \tag{3.18}$$

A partir dessas equações, é possível observar que o movimento no plano horizontal depende diretamente das juntas revolutas J_1 e J_2. Já o movimento vertical é controlado de forma independente pela junta prismática J_3, o que é uma característica típica de robôs do tipo SCARA. Dessa forma, o problema da cinemática inversa pode ser dividido em duas partes: o cálculo dos ângulos θ_1 e θ_2 no plano XY, e o cálculo do deslocamento q_3 no eixo vertical.

Inicialmente, calcula-se a distância radial entre a base do robô e o ponto desejado no plano horizontal. Essa distância, representada por R, é obtida por:

$$R^2 = X^2 + Y^2 \tag{3.19}$$

Essa relação permite verificar se o ponto desejado está dentro do espaço de trabalho do robô. Como apresentado na cinemática direta, o alcance máximo do manipulador no plano XY é dado por $L_1 + L_2$, enquanto o alcance mínimo é dado por $|L_1 - L_2|$. Portanto, para que o ponto seja alcançável, a distância R deve satisfazer a seguinte condição:

$$|L_1 - L_2| \leq R \leq L_1 + L_2 \tag{3.20}$$

Considerando os valores adotados neste projeto, o ponto desejado deve estar aproximadamente entre $0{,}18\,m$ e $0{,}58\,m$ em relação à base do robô. Caso essa condição não seja atendida, não existe uma configuração física das juntas θ_1 e θ_2 capaz de posicionar o *end-effector* no ponto

solicitado. Essa verificação é importante para evitar que o algoritmo tente calcular posições fora da região alcançável do manipulador.

Para determinar o ângulo θ_2, utiliza-se a Lei dos Cossenos aplicada ao triângulo formado pelos elos L_1, L_2 e pela distância R. A relação geométrica é dada por:

$$R^2 = L_1^2 + L_2^2 + 2L_1L_2\cos(\theta_2) \tag{3.21}$$

Isolando o termo $\cos(\theta_2)$, obtém-se:

$$\cos(\theta_2) = \frac{X^2 + Y^2 - L_1^2 - L_2^2}{2L_1L_2} \tag{3.22}$$

O valor obtido para $\cos(\theta_2)$ deve estar no intervalo $[-1, 1]$. Caso esteja fora desse intervalo, o ponto desejado é considerado inalcançável, pois não existe solução real para o ângulo θ_2. Essa condição também está diretamente relacionada aos limites do espaço de trabalho do robô.

Como o robô SCARA pode apresentar duas configurações geométricas para alcançar um mesmo ponto no plano, conhecidas como configuração com cotovelo para cima e cotovelo para baixo, é necessário escolher uma dessas soluções. Neste trabalho, foi adotada a configuração com seno positivo, correspondente à solução de cotovelo para cima. Assim, o seno de θ_2 é calculado por:

$$\sin(\theta_2) = \sqrt{1 - \cos^2(\theta_2)} \tag{3.23}$$

Em seguida, o ângulo θ_2 é obtido utilizando a função arco tangente de dois argumentos, que considera simultaneamente os valores de seno e cosseno e permite determinar corretamente o quadrante do ângulo:

$$\theta_2 = atan2\left(\sin(\theta_2), \cos(\theta_2)\right) \tag{3.24}$$

Após o cálculo de θ_2, determina-se o ângulo θ_1. Para isso, considera-se inicialmente o ângulo entre o eixo X da base e o ponto desejado, dado por $atan2(Y, X)$. Em seguida, subtrai-se a contribuição angular causada pelo segundo elo, obtendo o ângulo absoluto da primeira junta. Dessa forma:

$$\theta_1 = atan2(Y, X) - atan2\left(L_2 \sin(\theta_2), L_1 + L_2 \cos(\theta_2)\right) \tag{3.25}$$

Essa equação permite calcular a rotação necessária da base para posicionar o primeiro elo de forma que, combinado com o segundo elo, o *end-effector* alcance a coordenada desejada no plano XY. A primeira parcela indica a direção geral do ponto em relação à base, enquanto a segunda parcela corrige essa direção considerando a geometria formada pelo segundo elo.

O deslocamento vertical do *end-effector* é calculado de maneira independente, pois a junta prismática é responsável pelo movimento no eixo Z. Como definido na cinemática direta, a posição vertical do efetuador é dada por:

$$Z = z_0 - q_3 \tag{3.26}$$

em que $z_0 = 0{,}32\,m$ representa a altura de referência do punho. Isolando o deslocamento prismático, obtém-se:

$$q_3 = z_0 - Z \tag{3.27}$$

Assim, para uma posição vertical desejada Z, o valor de q_3 indica quanto a junta prismática deve se deslocar em relação à altura de referência. Esse cálculo também deve respeitar os limites físicos da junta prismática. Considerando o intervalo $0 \leq q_3 \leq 0{,}31\,m$, a posição desejada no eixo vertical deve permanecer dentro da faixa de trabalho definida para o manipulador.

Portanto, a cinemática inversa do robô SCARA permite converter uma posição desejada no espaço cartesiano em comandos articulares para as juntas do manipulador. As coordenadas X e Y são utilizadas para calcular os ângulos θ_1 e θ_2, enquanto a coordenada Z é utilizada para determinar o deslocamento q_3. Essa relação é fundamental para a movimentação automática do robô, pois permite que o sistema receba a posição de um objeto no espaço de trabalho e calcule as juntas necessárias para que o *end-effector* se aproxime desse ponto.

A formulação utilizada segue os princípios clássicos da cinemática de manipuladores robóticos, nos quais a cinemática inversa é obtida a partir da geometria do sistema e das relações estabelecidas pela cinemática direta (CRAIG, 2005; SPONG; HUTCHINSON; VIDYASAGAR, 2006).

3.4.4 Projeto matemático dos controladores PID das juntas

Conforme apresentado na subseção anterior, a cinemática inversa permite converter uma posição desejada do *end-effector*, definida no espaço cartesiano pelas coordenadas x_d, y_d e z_d, em referências articulares para o manipulador. Dessa forma, obtêm-se os valores desejados das juntas θ_{1d}, θ_{2d} e q_{3d}, que são utilizados como entradas de referência para os controladores.

A partir dessas referências, o controle de posição do robô SCARA foi desenvolvido no espaço das juntas, uma vez que os atuadores do manipulador atuam diretamente sobre as variáveis articulares. Assim, cada junta ativa recebe uma referência individual e o controlador atua para reduzir o erro entre a posição desejada e a posição real da junta.

As juntas consideradas no controle principal foram a junta revoluta da base, representada por θ_1, a junta revoluta entre os braços, representada por θ_2, e a junta prismática vertical, representada por q_3. A junta de rotação do *end-effector* não foi considerada no controle de posição, pois no modelo desenvolvido ela se encontra travada, apresentando limite inferior e superior iguais a zero.

Para cada junta ativa foi adotado um controlador PID independente. O erro de posição de cada junta é definido por:

$$e_i(t) = q_{i,d}(t) - q_i(t) \tag{3.28}$$

em que $q_{i,d}(t)$ representa a posição desejada da junta e $q_i(t)$ representa a posição real da junta. Para as juntas revolutas, q_i corresponde às posições angulares θ_1 e θ_2, medidas em radianos. Para a junta prismática, q_i corresponde ao deslocamento linear q_3, medido em metros.

A estratégia de controle pode ser representada de forma resumida pela seguinte sequência:

$$(x_d, y_d, z_d) \rightarrow \text{cinematica inversa} \rightarrow (\theta_{1d}, \theta_{2d}, q_{3d}) \rightarrow \text{controle PID das juntas} \tag{3.29}$$

Para a modelagem matemática das juntas revolutas, foi considerado um modelo dinâmico simplificado, composto pela inércia equivalente da junta e por um coeficiente de atrito viscoso. A representação dinâmica de juntas robóticas a partir de termos inerciais e dissipativos é compatível com a modelagem clássica de manipuladores robóticos (SPONG; HUTCHINSON; VIDYASAGAR, 2006; SICILIANO et al., 2009). Além disso, a simplificação para uma função de transferência de baixa ordem é adequada para etapas iniciais de projeto de controle, pois

permite representar a dinâmica dominante do sistema sem exigir a modelagem completa de todos os acoplamentos dinâmicos do manipulador (OGATA, 2010). Para uma junta revoluta genérica i, a dinâmica pode ser representada por:

$$\tau_i(t) = J_i\ddot{\theta}_i(t) + B_i\dot{\theta}_i(t) \tag{3.30}$$

em que $\tau_i(t)$ é o torque aplicado à junta, J_i é a inércia equivalente da junta, B_i é o coeficiente de atrito viscoso equivalente e $\theta_i(t)$ é a posição angular da junta.

Aplicando a Transformada de Laplace, considerando condições iniciais nulas, obtém-se:

$$\tau_i(s) = J_i s^2\theta_i(s) + B_i s\theta_i(s) \tag{3.31}$$

Colocando $\theta_i(s)$ em evidência:

$$\tau_i(s) = \left(J_i s^2 + B_i s\right)\theta_i(s) \tag{3.32}$$

Logo, a função de transferência que relaciona a posição angular da junta com o torque aplicado é dada por:

$$G_i(s) = \frac{\theta_i(s)}{\tau_i(s)} = \frac{1}{J_i s^2 + B_i s} \tag{3.33}$$

Esse modelo foi utilizado para representar as juntas revolutas θ_1 e θ_2. Para a junta prismática, o raciocínio é semelhante, porém a inércia rotacional é substituída pela massa equivalente deslocada no movimento linear. Assim, a dinâmica da junta prismática pode ser representada por:

$$F_3(t) = M_3\ddot{q}_3(t) + B_3\dot{q}_3(t) \tag{3.34}$$

em que $F_3(t)$ é a força aplicada na junta prismática, M_3 é a massa equivalente deslocada, B_3 é o coeficiente de atrito viscoso e $q_3(t)$ é o deslocamento linear da junta.

Aplicando a Transformada de Laplace:

$$F_3(s) = M_3 s^2 q_3(s) + B_3 s q_3(s) \tag{3.35}$$

$$F_3(s) = \left(M_3 s^2 + B_3 s\right) q_3(s) \tag{3.36}$$

Portanto, a função de transferência da junta prismática é dada por:

$$G_3(s) = \frac{q_3(s)}{F_3(s)} = \frac{1}{M_3 s^2 + B_3 s} \tag{3.37}$$

No modelo desenvolvido, a junta prismática possui eixo de atuação no sentido negativo do eixo Z. Dessa forma, o aumento do deslocamento q_3 provoca a descida do *end-effector*. Considerando os deslocamentos fixos da estrutura do robô, a altura do efetuador pode ser expressa por:

$$z = 0{,}32 - q_3 \tag{3.38}$$

Os parâmetros equivalentes das juntas foram estimados a partir das propriedades físicas definidas no modelo URDF/Xacro, como massas, momentos de inércia e posições dos centros de massa. Para as juntas revolutas, a inércia equivalente foi estimada considerando a contribuição dos corpos movimentados por cada junta. Essa estimativa pode ser representada pelo Teorema dos Eixos Paralelos, utilizado para transferir momentos de inércia entre eixos paralelos deslocados em relação ao centro de massa (HIBBELER, 2018; SPONG; HUTCHINSON; VIDYASAGAR, 2006):

$$J_i = \sum_{k=1}^{n} \left(I_{z,k} + m_k r_k^2\right) \tag{3.39}$$

em que $I_{z,k}$ é o momento de inércia do corpo k em relação ao eixo Z, m_k é a massa do corpo, r_k é a distância entre o eixo da junta e o centro de massa do corpo e n representa o número de elementos movimentados pela junta.

Para a junta prismática, a massa equivalente foi obtida pela soma das massas dos elementos deslocados verticalmente, sendo eles o fuso, o conector da garra e os dois elementos da garra:

$$M_3 = m_{fuso} + m_{conector} + m_{garraE} + m_{garraD} \tag{3.40}$$

Substituindo os valores presentes no modelo:

$$M_3 = 0{,}763 + 0{,}00378 + 0{,}00216 + 0{,}00216 \tag{3.41}$$

$$M_3 \approx 0{,}7711\,kg \tag{3.42}$$

Os parâmetros equivalentes adotados no projeto dos controladores são apresentados na Tabela 3.2.

Tabela 3.2 – Parâmetros equivalentes utilizados no projeto dos controladores.

Junta	Parâmetro equivalente	Valor utilizado
θ_1	J_1	$4{,}106\,kg \cdot m^2$
θ_2	J_2	$0{,}3435\,kg \cdot m^2$
q_3	M_3	$0{,}7711\,kg$

O controlador adotado para cada junta foi o PID na forma paralela, estrutura amplamente utilizada em sistemas de controle devido à sua simplicidade, robustez e facilidade de implementação (OGATA, 2010). Sua função de transferência é dada por:

$$C_i(s) = K_{p_i} + \frac{K_{i_i}}{s} + K_{d_i}s \tag{3.43}$$

No domínio do tempo, a ação de controle pode ser escrita como:

$$u_i(t) = K_{p_i}e_i(t) + K_{i_i}\int e_i(t)dt + K_{d_i}\frac{de_i(t)}{dt} \tag{3.44}$$

em que K_{p_i}, K_{i_i} e K_{d_i} são, respectivamente, os ganhos proporcional, integral e derivativo da junta i. O termo proporcional atua diretamente sobre o erro de posição, o termo integral reduz o erro em regime permanente e o termo derivativo contribui para o amortecimento da resposta transitória.

Considerando a função de transferência da junta revoluta e o controlador PID $C_i(s)$, a função de transferência de malha fechada com realimentação unitária é:

$$T_i(s) = \frac{C_i(s)G_i(s)}{1 + C_i(s)G_i(s)} \tag{3.45}$$

A equação característica do sistema em malha fechada é obtida a partir do denominador:

$$1 + C_i(s)G_i(s) = 0 \tag{3.46}$$

Substituindo as expressões de $C_i(s)$ e $G_i(s)$, e multiplicando a equação por $s(J_i s^2 + B_i s)$, obtém-se:

$$J_i s^3 + (B_i + K_{d_i}) s^2 + K_{p_i} s + K_{i_i} = 0 \tag{3.47}$$

Essa expressão representa a equação característica da junta revoluta controlada por PID. Para a junta prismática, a estrutura matemática é equivalente, substituindo-se a inércia J_i pela massa equivalente M_3:

$$M_3 s^3 + (B_3 + K_{d_3}) s^2 + K_{p_3} s + K_{i_3} = 0 \tag{3.48}$$

A sintonia dos controladores foi realizada pelo método de alocação de polos, técnica baseada na escolha de uma dinâmica desejada para o sistema em malha fechada (OGATA, 2010). Como a inserção do termo integral no controlador PID resulta em uma equação característica de terceira ordem, foi definido um polinômio desejado composto por um par de polos dominantes de segunda ordem e um terceiro polo real não dominante. O polinômio desejado foi definido como:

$$P_d(s) = \left(s^2 + 2\zeta\omega_n s + \omega_n^2\right)(s + p_3) \tag{3.49}$$

em que ζ é o fator de amortecimento, ω_n é a frequência natural desejada e p_3 é o terceiro polo real. A frequência natural foi determinada a partir do tempo de acomodação desejado, utilizando a aproximação clássica para sistemas de segunda ordem:

$$t_s \approx \frac{4}{\zeta\omega_n} \tag{3.50}$$

Isolando ω_n, tem-se:

$$\omega_n = \frac{4}{\zeta t_s} \tag{3.51}$$

Neste trabalho, adotou-se um fator de amortecimento $\zeta = 0{,}7$, buscando uma resposta com bom amortecimento e baixa oscilação. O tempo de acomodação desejado foi definido como $t_s = 2\,s$. Assim:

$$\omega_n = \frac{4}{0{,}7 \cdot 2} \tag{3.52}$$

$$\omega_n \approx 2{,}857\, rad/s \tag{3.53}$$

O terceiro polo foi escolhido de modo a ser mais rápido que a dinâmica dominante. Para isso, foi adotada a relação:

$$p_3 = 5\zeta\omega_n \tag{3.54}$$

Substituindo os valores:

$$p_3 = 5 \cdot 0{,}7 \cdot 2{,}857 \tag{3.55}$$

$$p_3 \approx 10 \tag{3.56}$$

Dessa forma, o terceiro polo é posicionado em $s = -10$, ficando mais à esquerda no plano complexo e apresentando uma dinâmica mais rápida que os polos dominantes.

Expandindo o polinômio desejado:

$$P_d(s) = s^3 + (2\zeta\omega_n + p_3)s^2 + (\omega_n^2 + 2\zeta\omega_n p_3)s + \omega_n^2 p_3 \tag{3.57}$$

A equação característica normalizada da junta revoluta é obtida dividindo-se a Equação 3.47 por J_i:

$$s^3 + \frac{B_i + K_{d_i}}{J_i}s^2 + \frac{K_{p_i}}{J_i}s + \frac{K_{i_i}}{J_i} = 0 \tag{3.58}$$

Comparando a Equação 3.58 com o polinômio desejado da Equação 3.57, obtêm-se as expressões dos ganhos PID:

$$K_{d_i} = J_i(2\zeta\omega_n + p_3) - B_i \tag{3.59}$$

$$K_{p_i} = J_i(\omega_n^2 + 2\zeta\omega_n p_3) \tag{3.60}$$

$$K_{i_i} = J_i \omega_n^2 p_3 \tag{3.61}$$

Para a junta prismática, substitui-se J_i por M_3, resultando em:

$$K_{d_3} = M_3(2\zeta\omega_n + p_3) - B_3 \tag{3.62}$$

$$K_{p_3} = M_3(\omega_n^2 + 2\zeta\omega_n p_3) \tag{3.63}$$

$$K_{i_3} = M_3 \omega_n^2 p_3 \tag{3.64}$$

Neste projeto, os coeficientes de atrito viscoso B_1, B_2 e B_3 foram considerados inicialmente nulos, uma vez que não foram identificados experimentalmente. Com isso, os ganhos calculados por alocação de polos são apresentados na Tabela 3.3.

Tabela 3.3 – Ganhos PID calculados por alocação de polos.

Junta	K_p	K_i	K_d
θ_1	197,7584	335,1837	57,4840
θ_2	16,5441	28,0408	4,8090
q_3	37,1387	62,9469	10,7954

Os ganhos obtidos representam os valores iniciais de projeto dos controladores PID, calculados com base no modelo dinâmico simplificado das juntas e nas especificações de desempenho adotadas. A validação desses parâmetros foi realizada por meio da simulação da resposta ao degrau das juntas, permitindo verificar características como tempo de subida, tempo de acomodação, sobrelevação e erro em regime permanente.

Dessa forma, a sintonia inicial dos controladores foi conduzida por um procedimento matemático sistemático, baseado na modelagem das juntas e na alocação de polos em malha fechada. Esse procedimento evita que os ganhos sejam definidos exclusivamente por tentativa e erro e fornece uma base inicial coerente para os ajustes posteriores realizados no ambiente de simulação.

3.5 Configuração do Gazebo

3.5.1 Reestruturação do modelo para simulação

Para tornar a integração do robô ao ambiente de simulação do Gazebo mais fácil, foi necessário modificar a estrutura inicial do modelo para uma forma mais modular. Para isso, foi criado o arquivo `juntas.xacro`, utilizando macros para concentrar os principais parâmetros e propriedades do manipulador, como dimensões, massas, inércias, juntas, motores, sensores e demais componentes necessários à representação do robô.

A utilização do formato Xacro torna a modelagem mais organizada e mais fácil de fazer alterações, pois a criação de macros simplifica a descrição do modelo. Dessa forma, alterações em dimensões, propriedades físicas ou componentes do sistema puderam ser realizadas de maneira mais direta, sem precisar modificar repetidamente diversos trechos do arquivo principal. Essa abordagem também facilitou a evolução do projeto, especialmente nas etapas que exigiram ajustes na estrutura do robô ou adicionar novos elementos.

Com base nessa organização, o modelo inicialmente desenvolvido no arquivo `scarav1.urdf` foi reorganizado em uma nova versão, nomeado `scarav2.urdf`. Essa nova versão teve como objetivo melhorar a identificação dos componentes, separar de forma mais clara as propriedades físicas e preparar o robô para a integração com os recursos necessários à simulação no Gazebo. Com essa configuração o modelo passa a incorporar elementos relacionados à simulação física como massa e inercia, além de motores para controle das juntas deixando de ser apenas uma representação visual.

As propriedades de massa e inércia inseridas no modelo foram definidas a partir dos cálculos apresentados na Subseção 3.4.1. Esses parâmetros são importantes para que o Gazebo represente o comportamento do robô da forma correta, pois influenciam diretamente a resposta dos elos aos esforços aplicados pelos atuadores e controladores. Como já foi dito na Subseção 3.4.1, quando esses valores são definidos de forma inadequada, o manipulador pode apresentar comportamentos instáveis.

Além das propriedades físicas, também foi necessário configurar os atuadores responsáveis pelo movimento das juntas. No ambiente do Gazebo, não basta apenas uma definição geométrica simples do robô, além disso, é necessário associar as juntas a mecanismos de controle capazes de receber comandos do ROS e convertê-los em movimentos dentro da simulação. Sem essa configuração, o robô poderia ser carregado no simulador, mas não responderia corretamente aos

comandos enviados, podendo apresentar instabilidade devido à ação da gravidade e à ausência de controle aplicado às juntas.

3.5.2 Inicialização do ambiente de simulação

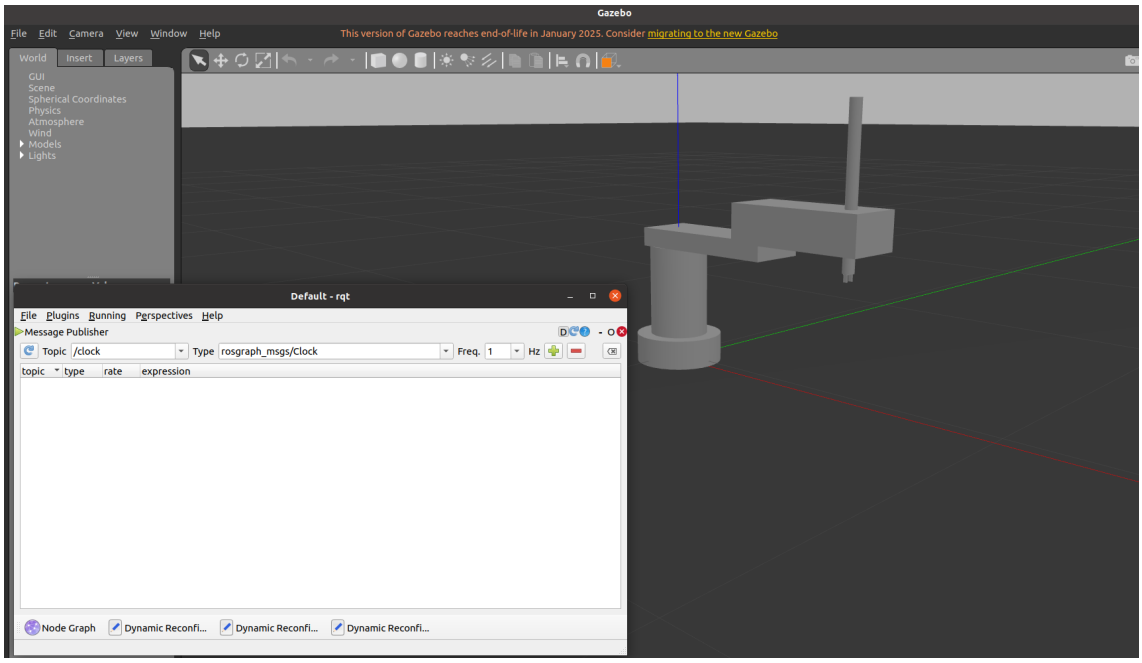
Como parte da configuração do sistema de controle, foi criado o arquivo `controle_juntas.yaml` armazenado na pasta `config` do pacote. Esse arquivo contém os parâmetros dos controladores de cada junta ativa, incluindo os ganhos do controlador PID. Os valores iniciais utilizados foram definidos a partir do projeto matemático apresentado na Subseção 3.4.4, evitando utilizar de valores aleatórios e relacionando a configuração dos controladores à dinâmica simplificada das juntas, às propriedades inerciais do manipulador e ao método de alocação de polos.

Para a junta revoluta da base, representada por θ_1, foram adotados inicialmente os ganhos $K_p = 197{,}7584$, $K_i = 335{,}1837$ e $K_d = 57{,}4840$. Para a junta revoluta entre os braços, representada por θ_2, foram utilizados os ganhos $K_p = 16{,}5441$, $K_i = 28{,}0408$ e $K_d = 4{,}8090$. Para a junta prismática, representada por q_3, foram definidos os ganhos $K_p = 37{,}1387$, $K_i = 62{,}9469$ e $K_d = 10{,}7954$. Esses valores foram inseridos como parâmetros iniciais no arquivo de configuração dos controladores.

Para adicionar o modelo do robô SCARA no ambiente de simulação do Gazebo, foi criado o arquivo `gazebo.launch`. Esse arquivo carrega o manipulador em um mundo virtual vazio e inicia os controladores necessários à movimentação das juntas. Antes de incluir objetos ou tarefas mais complexas. Ao inciar em um ambiente virtual vazio é possível avaliar melhor a estabilidade do robô, assim como a resposta das juntas aos comandos de posição e a influência dos parâmetros de controle de forma isolada.

Dessa forma, o arquivo `gazebo.launch` age de forma a integrar o modelo físico, os controladores e o simulador. Com a configuração implementada, ele permite executar a simulação com os recursos necessários para que o robô seja adicionado ao mundo da forma correta, para validar o comportamento do robô e realizar os ajustes dos controladores. Os comandos utilizados para executar essa configuração são apresentados na Seção C.1 do Anexo C. Com as configurações realizadas, foi obtido o ambiente demonstrado na Figura 3.3.

Figura 3.3 – Modelo do robô SCARA no ambiente de simulação do Gazebo.



3.5.3 Publicação de comandos para movimentação das juntas

Caso a interface do `rqt_reconfigure` não apresente inicialmente os recursos necessários para o envio de comandos às juntas, é preciso configurar o ambiente do RQT. Para isso, basta acessar o menu superior *Plugins*, selecionar a opção *Topics* e adicionar o recurso `Message Publisher`. Esse *plugin* permite publicar mensagens diretamente nos tópicos do ROS, possibilitando o envio manual de comandos de posição para os controladores das juntas.

O `Message Publisher` publica valores de posição para as juntas, além de conseguir publicar valores para que as juntas realizem movimentos cíclicos. Para isso, devem ser adicionados os tópicos correspondentes aos controladores das juntas, normalmente identificados pela estrutura `nome_da_junta/command`, permitindo observar a resposta do manipulador no ambiente do Gazebo.

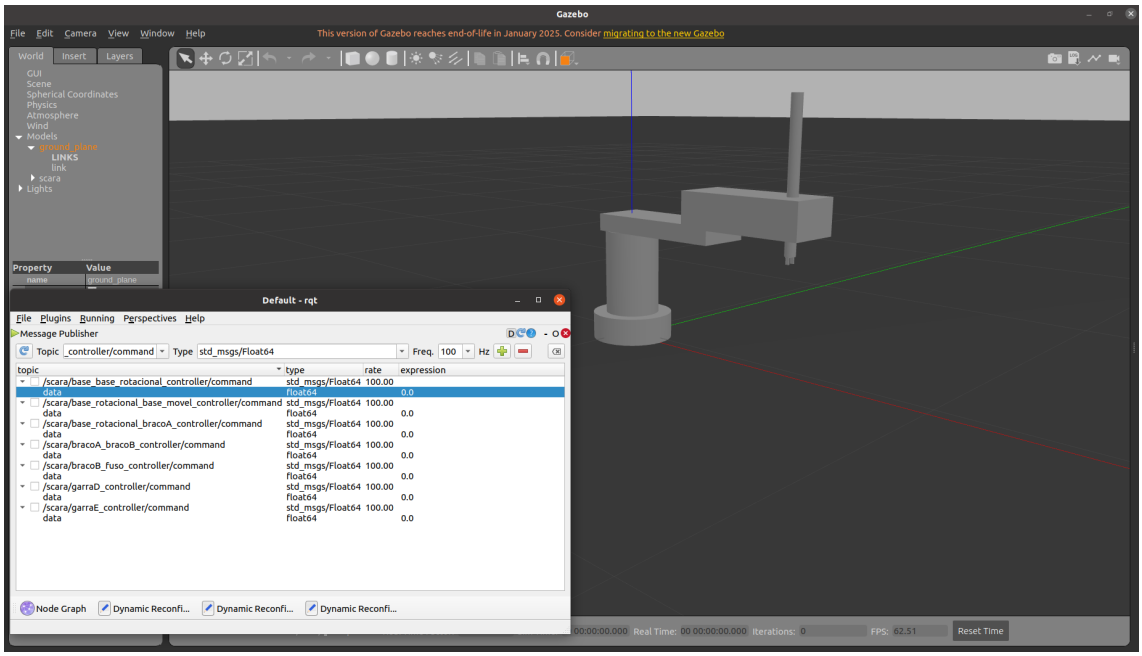
Para configurar o *plugin* também é necessário definir uma frequência de publicação das mensagens. Esse parâmetro indica quantas vezes por segundo o comando será enviado ao controlador, caso seja muito baixo o robô pode não concluir o movimento completo, o que dificulta a observação do comportamento das juntas.

Por outro lado, frequências muito altas também podem prejudicar a simulação. Embora aumentem a taxa de envio de comandos, valores excessivos podem reduzir o desempenho

da simulação e tornar os movimentos mais lento ou menos responsivos, o robô pode acabar ultrapassando os limites estabelecidos. Dessa forma, a escolha da frequência deve equilibrar continuidade de movimento e desempenho.

Neste projeto, foi adotada uma frequência de $100\,Hz$ para o envio dos comandos às juntas. Esse valor permitiu uma movimentação contínua para analisar o comportamento das juntas do robô. Após a definir o tópico e a frequência, o comando numeral é adicionado ao `Message Publisher`, ele faz com que a posição das juntas seja modificada durante os em radianos para as juntas rotacionais, e em milímetros para as prismáticas. A configuração final do *publisher* é apresentada na Figura 3.4.

Figura 3.4 – Tópicos das juntas adicionados ao *plugin Message Publisher*.



Para realizar os testes de movimentação das juntas, foi utilizada uma expressão periódica no campo *data* do `Message Publisher`. Essa expressão permite enviar comandos variáveis as juntas, fazendo com que elas executem um movimento cíclico durante a simulação. A expressão utilizada é apresentada na Equação 3.65.

$$1{,}57 \sin\left(\frac{i}{100}\right) \tag{3.65}$$

Essa função faz com que a junta selecionada alterne sua posição entre $1{,}57\,rad$ e $-1{,}57\,rad$. Dessa forma, é possível observar o comportamento do motor da base rotacional e

das demais juntas durante a execução de movimentos cíclicos. Ao utilizar esse comando em diferentes juntas, é possível verificar se o robô responde corretamente aos comandos enviados e se os controladores configurados conseguem acompanhar as referências de posição.

3.5.4 Monitoramento da resposta das juntas no RQT

Para avaliar melhor o desempenho do robô com os valores de PID calculados anteriormente, foi utilizado o recurso *Plot* do RQT. Esse *plugin* permite visualizar graficamente, em tempo real, as variáveis de controle publicadas nos tópicos do ROS. Com isso, torna-se possível comparar a posição desejada enviada ao controlador com a posição real atingida pela junta durante a simulação.

A utilização do *Plot* é importante porque a análise visual do movimento no Gazebo, embora útil, nem sempre permite identificar com precisão erros de acompanhamento, oscilações ou atrasos na resposta das juntas. Por meio dos gráficos, é possível observar de forma mais clara se os ganhos PID calculados estão proporcionando uma resposta adequada ou se será necessário realizar ajustes posteriores com o auxílio do `rqt_reconfigure`.

Para adicionar esse recurso, deve-se acessar a aba *Plugins* do RQT, selecionar a opção *Visualization* e, em seguida, escolher o *plugin Plot*. Após a abertura da interface gráfica, devem ser inseridos os tópicos correspondentes às variáveis que se deseja monitorar. No caso da junta da base rotacional, por exemplo, é possível acompanhar o comando enviado ao controlador por meio do tópico:

```
/scara/base_base_rotacional_controller/command/data
```

Esse tópico representa a posição de referência que o controlador deve seguir. Para comparar essa referência com a posição realmente atingida pela junta, também é necessário adicionar o tópico correspondente ao valor atual do processo:

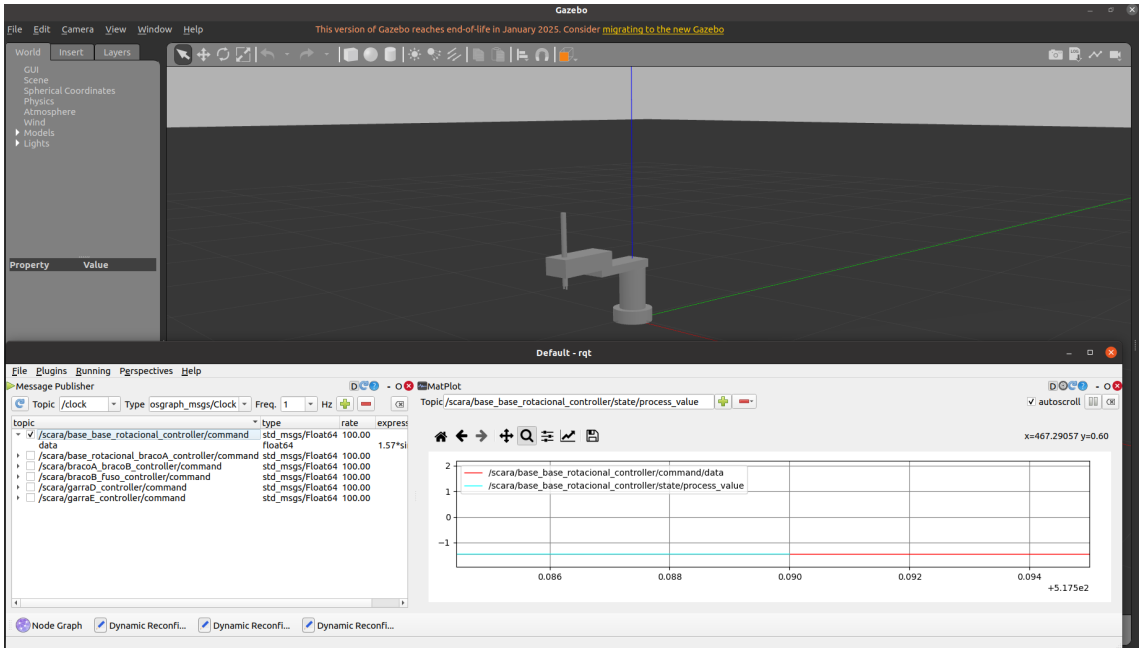
```
/scara/base_base_rotacional_controller/state/process_value
```

O primeiro tópico indica o valor desejado de posição, enquanto o segundo apresenta a posição real da junta durante a movimentação. Quando ambos são exibidos no mesmo gráfico, torna-se possível avaliar o desempenho do controlador de forma direta, observando se a junta acompanha adequadamente a referência aplicada.

Caso exista uma diferença significativa entre a posição desejada e a posição real, essa discrepância pode indicar a necessidade de ajustes nos ganhos PID, nos parâmetros físicos do modelo ou nas configurações dos controladores. Da mesma forma, oscilações excessivas, atraso elevado ou erro persistente em regime permanente podem indicar que os parâmetros calculados precisam ser refinados no ambiente de simulação.

Assim, o uso conjunto do `Message Publisher` e do *Plot* permite testar os movimentos das juntas e analisar graficamente a resposta do sistema. Essa etapa é fundamental para validar os ganhos PID obtidos no projeto matemático e preparar a comparação posterior com os valores ajustados empiricamente. A Figura 3.5 ilustra a utilização do *Plot* para monitorar a posição de referência e a posição real de uma junta específica.

Figura 3.5 – Monitoramento da posição desejada e da posição real de uma junta no RQT.



3.5.5 Ajuste empírico dos controladores

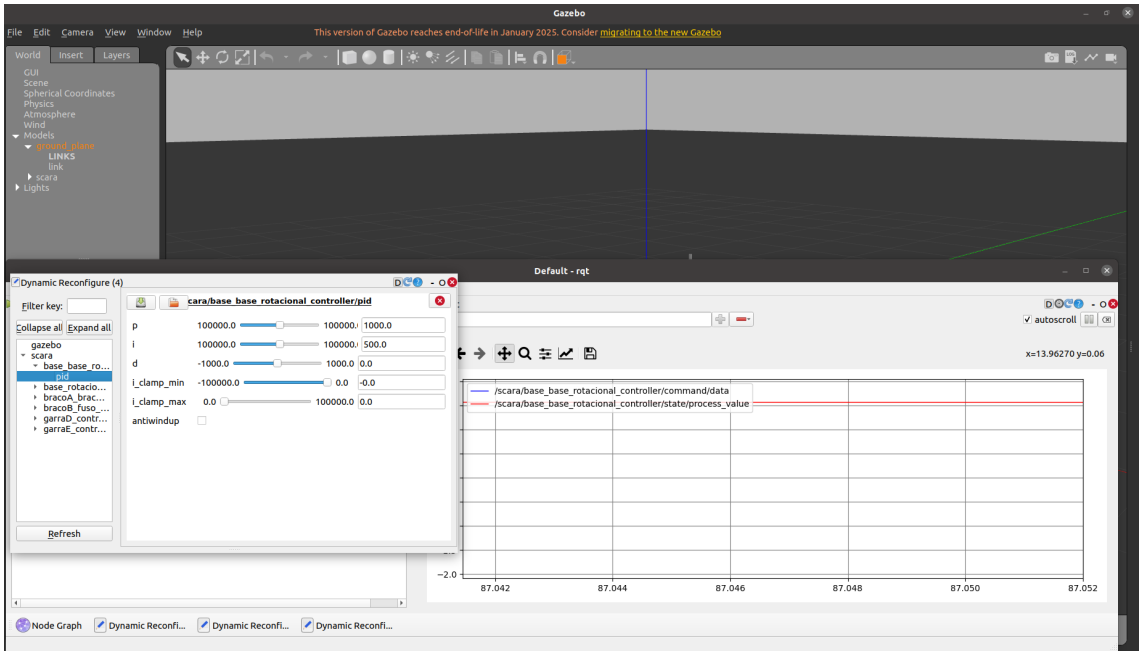
Para realizar o ajuste dos controladores no RQT, foi utilizado o *plugin* `dynamic_reconfigure`, ele permite modificar em tempo real os ganhos do controlador PID de cada junta durante a execução da simulação. Com isso é possível observar o comportamento do robô no Gazebo e alterar os parâmetros de controle sem a necessidade de alterar diretamente os arquivos de configuração, assim não sendo necessário interromper a simulação.

Ao utilizar o `dynamic_reconfigure` em conjunto com o *Plot* o ajuste fica ainda

mais eficiente, pois enquanto o `dynamic_reconfigure` permite alterar os valores de K_p, K_i e K_d, o *Plot* possibilita visualizar graficamente a posição desejada e a posição real de cada junta. Com isso, é possível identificar se há alterações indesejadas entre o comando enviado e a resposta obtida pelo robô.

Ao utilizar esse método é possível verificar o desempenho dos ganhos calculados pelo método matemático e encontrar valores mais adequados, caso o modelo matemático ainda não seja suficiente. Os valores empíricos são obtidos por meio de testes, ao observar a resposta das juntas em tempo real até que o comportamento do sistema apresente maior estabilidade, e menor oscilação ao acompanhar a referência. A Figura 3.6 apresenta os *plugins* utilizados em conjunto durante esse processo.

Figura 3.6 – *Plugins* utilizados no ajuste dos controladores.



Um ponto importante sobre o ajuste dos controladores é que as juntas do robô não devem ser analisadas apenas de forma individual, embora cada junta possua seu controlador próprio, o movimento de uma junta influência no comportamento das demais, principalmente devido a dinâmica do robô. Por isso é importante adicionar ao *Plot* os tópicos referentes a todas as juntas ativas, permitindo uma análise completa do comportamento do robô.

Para tornar essa verificação mais completa, foram aplicados comandos cíclicos às juntas, utilizando a expressão apresentada anteriormente, $1{,}57\,\mathrm{sin}(i/100)$. Esse tipo de comando permite

que as articulações realizem movimentos contínuos, facilitando a observação da resposta do sistema ao longo do tempo. Ao movimentar as juntas simultaneamente, é possível verificar se os controladores mantêm estabilidade, se existe alguma interferência entre os movimentos e se os ganhos definidos são adequados para o funcionamento do robô em conjunto.

Após a análise dos ganhos calculados e a obtenção dos valores empíricos por meio do `dynamic_reconfigure`, os parâmetros considerados mais adequados devem ser inseridos no arquivo `controle_juntas.yaml`. Dessa forma, sempre que o robô for inicializado no Gazebo por meio do arquivo `gazebo.launch`, os controladores serão carregados com os valores definidos, garantindo que o manipulador apresente um comportamento mais estável durante os testes.

Os procedimentos foram realizados a fim de validar a modelagem física do robô e avaliar o desempenho dos controladores. A combinação entre ROS, Gazebo, *Plot* e `dynamic_reconfigure` permite acompanhar o comportamento do sistema de forma visual e dinâmica, o que facilita a comparação entre os ganhos obtidos por cálculo e os ganhos ajustados empiricamente.

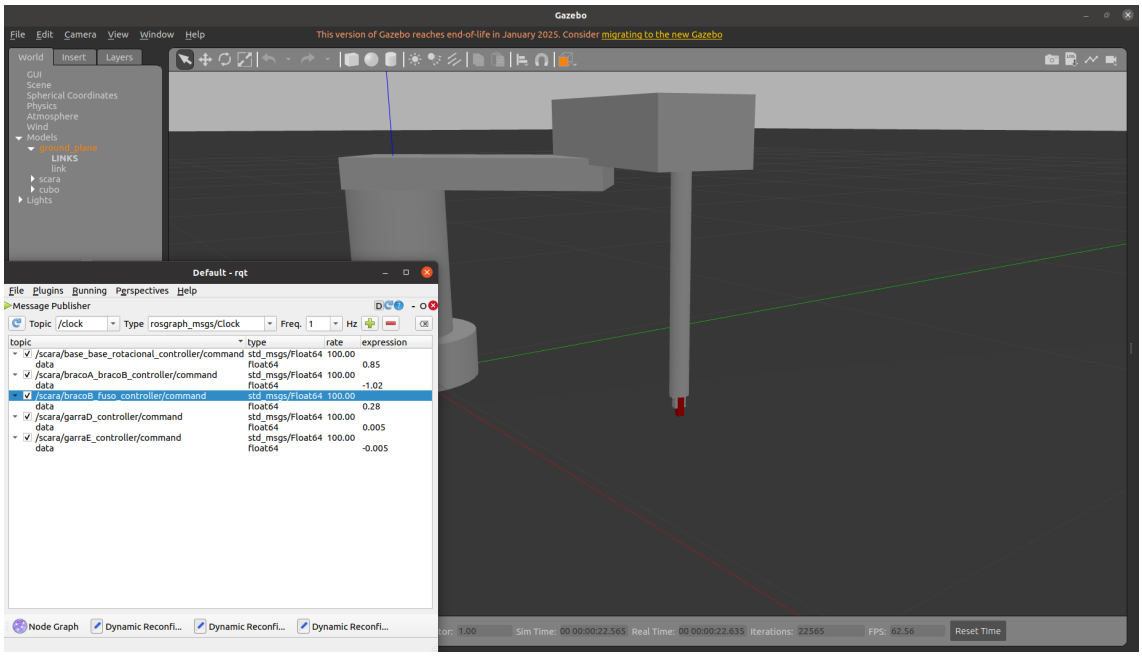
3.5.6 Inserção do objeto no ambiente de simulação

Após validar os controladores das juntas, o próximo passo foi adicionar um objeto ao ambiente de simulação. O objeto foi posicionado dentro da área de trabalho do robô a fim de verificar se ele seria capaz de alcançar e deslocar o objeto. Isso seria feito utilizando o `Message Publisher` para publicar os comandos para as juntas.

Para isso, foi desenvolvido um novo arquivo nomeado `objeto.urdf`, armazenado na pasta `urdf` do pacote, a fim de manter a organização da estrutura do projeto. Como o robô, para o objeto também foi necessário adicionar suas propriedades de massa e inércia, considerando sua representação como um corpo metálico. Após a criação do objeto no mundo, o arquivo *launch* foi alterado para carregar o objeto no mundo do Gazebo em uma posição acessível ao manipulador.

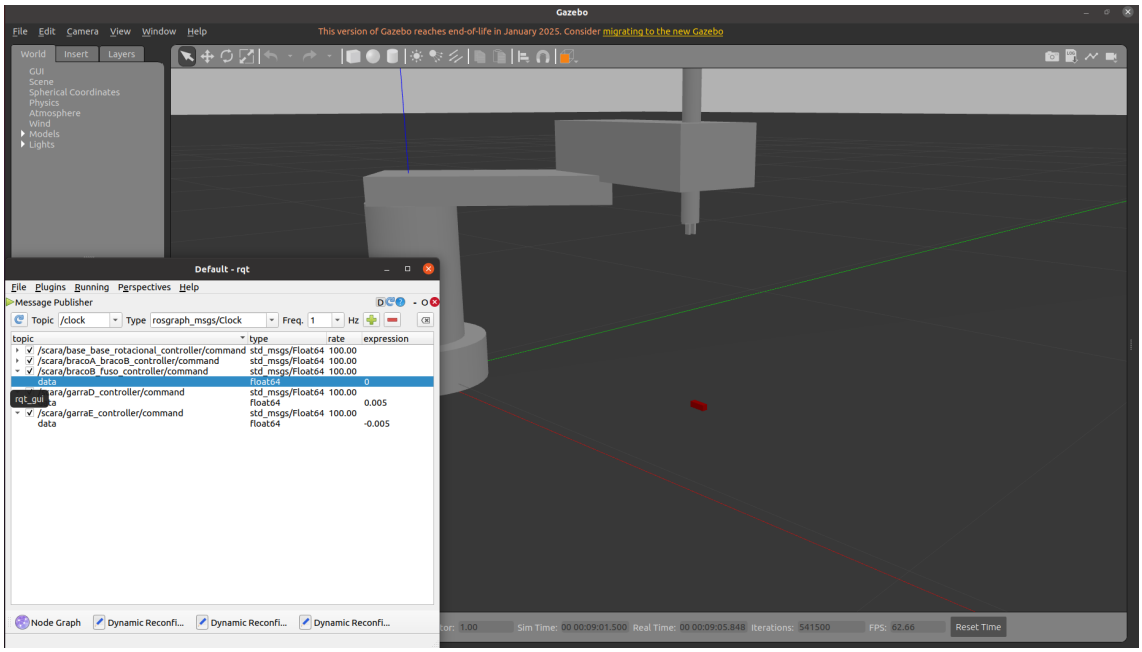
Com o objeto inserido no ambiente de simulação, foi possível utilizar o `Message Publisher` para enviar comandos às juntas e posicionar o *end-effector* próximo ao objeto. As rotações necessárias para alcançar a posição do objeto no mundo são apresentadas na Figura 3.7.

Figura 3.7 – Rotações necessárias para o manipulador alcançar o objeto.



Durante os testes iniciais de manipulação, foi possível observar que o robô conseguia alcançar o objeto e pegá-lo com a garra. Entretanto, ao final do movimento, o objeto acabava escorregando e caindo. Esse comportamento ocorreu porque, nessa etapa, o modelo ainda não possuía o *plugin* necessário para melhorar a interação de contato entre a garra e o objeto no Gazebo. A Figura 3.8 apresenta o resultado desse teste, no qual é possível observar o objeto caído após a tentativa de manipulação.

Figura 3.8 – Objeto derrubado durante o teste inicial de manipulação.

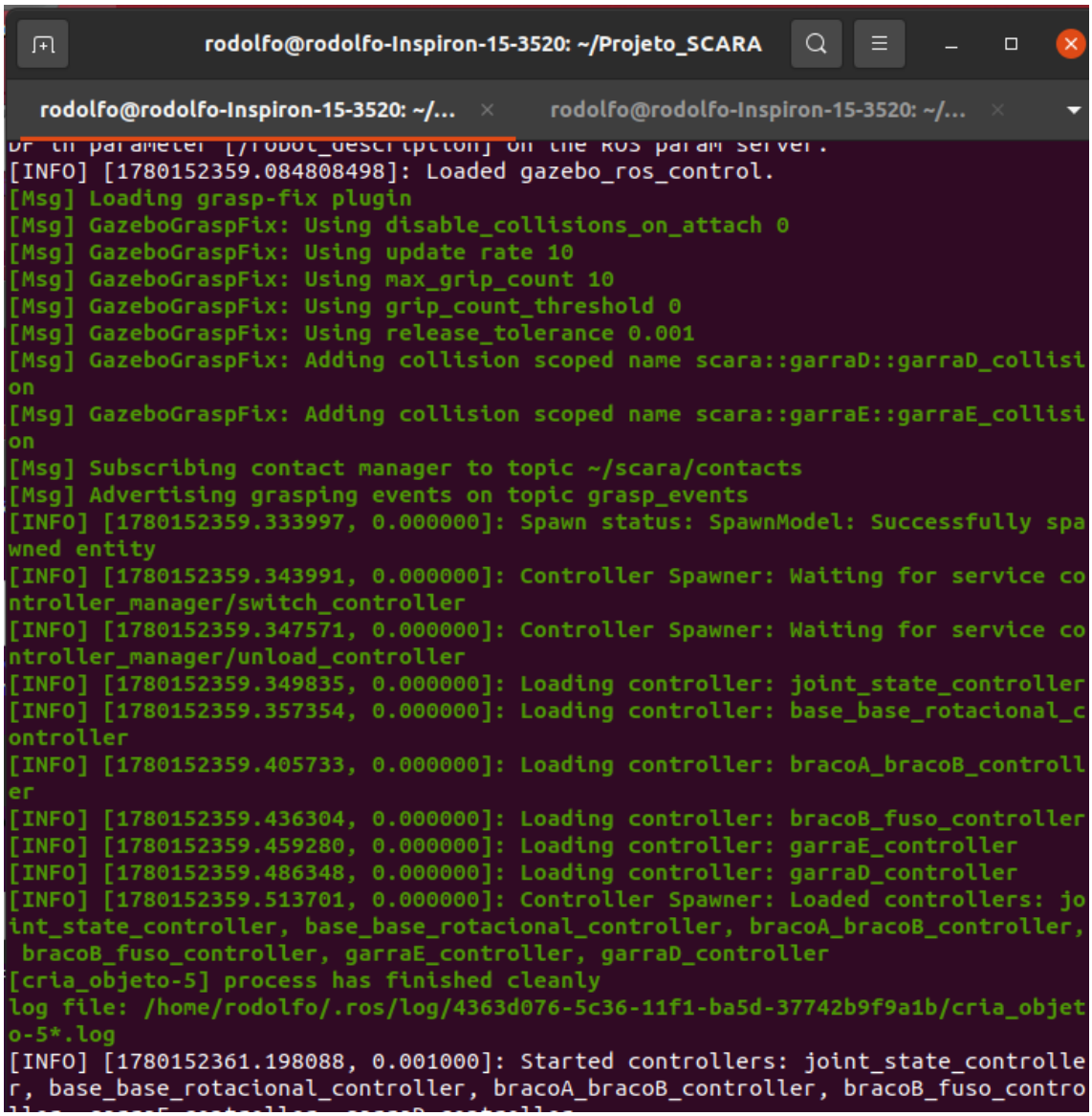


3.5.7 Plugins de preensão no Gazebo

Para que o objeto não saísse das garras do robô foram adicionados *plugins* específicos relacionados ao *grasp* no Gazebo. Esses recursos ajudam na interação entre a garra e o objeto, tornando a prenssão mais estável. Os comandos utilizados para instalar os pacotes necessários são apresentados na Seção C.2 Anexo C.

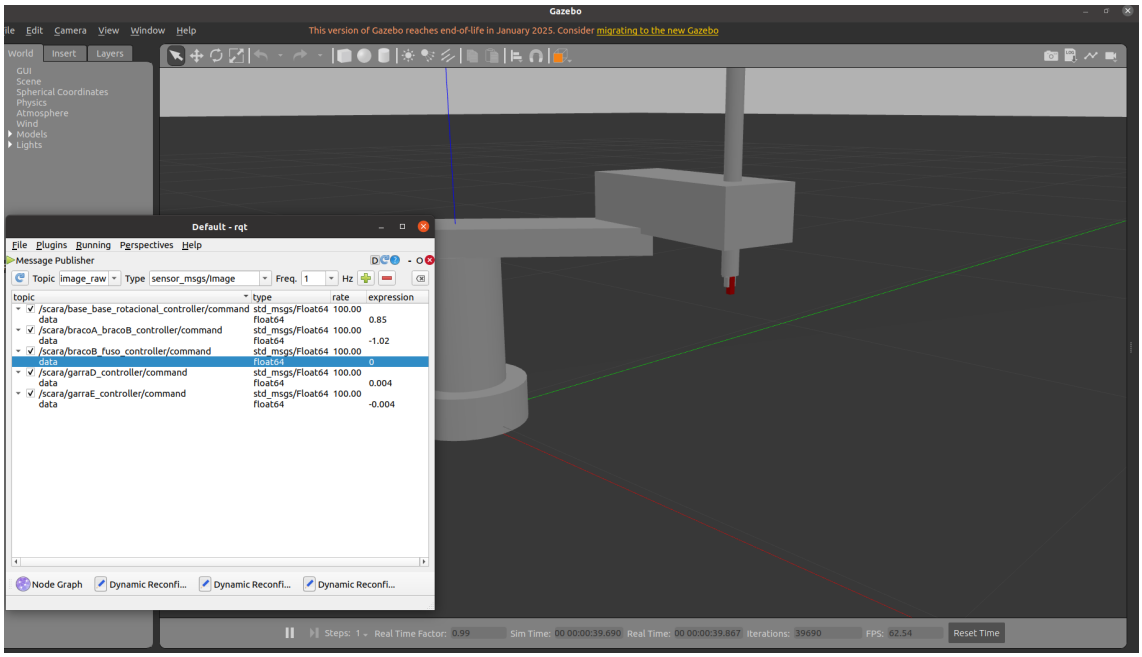
Após os *plugins* serem adicionados, foi necessário alterar o arquivo `scarav2.urdf`, para que fossem aplicados a garra. Essa configuração melhora a aderência entre a garra e o objeto durante a simulação, fazendo com que o objeto seja aderido com mais firmeza. Quando a configuração é bem sucedida o o terminal apresenta mensagens indicando o carregamento adequado dos *plugins*, conforme demonstrado na Figura 3.9.

Figura 3.9 – Mensagens de carregamento dos *plugins* de manipulação no Gazebo.



Com os *plugins* configurados, foram realizados novos testes de manipulação, com a nova configuração o robô conseguiu segurar o objeto da forma esperada, reduzindo o escorregamento observado anteriormente. A Figura 3.10 apresenta o resultado do teste após a configuração dos recursos de preensão.

Figura 3.10 – Manipulação do objeto após a configuração dos *plugins* de preensão.



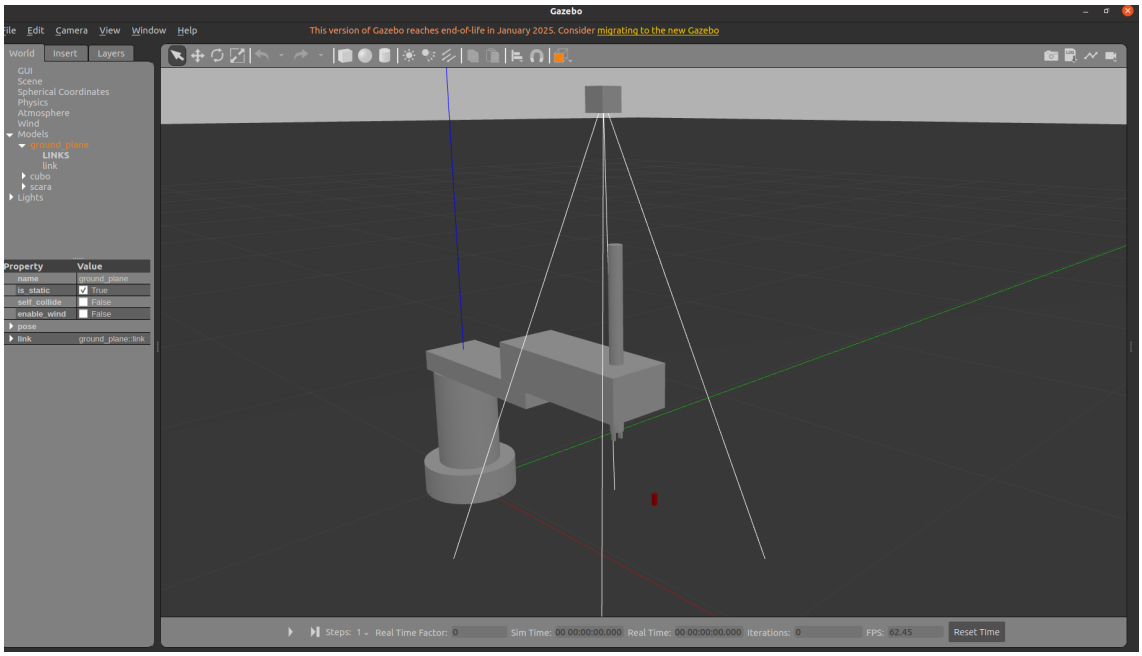
3.5.8 Inserção da câmera no ambiente de simulação

Após a validação inicial da manipulação do objeto, a etapa seguinte consistiu na inserção de uma câmera no ambiente de simulação. O objetivo dessa adição foi permitir a observação do espaço de trabalho do robô e preparar o sistema para as próximas etapas de identificação de objetos. Para isso, foram realizadas modificações tanto no arquivo `scarav2.urdf` quanto no arquivo de macros `juntas.xacro`.

No arquivo `juntas.xacro`, foi adicionada uma macro responsável pela configuração do sensor de câmera no Gazebo. Essa configuração foi responsável por definir parâmetros importantes, como características do sensor, tópicos de publicação da imagem e informações de calibração. Já no arquivo `scarav2.urdf`, foram criados o modelo físico da câmera, suas dimensões, seu posicionamento e a junta responsável por conectá-la ao mundo.

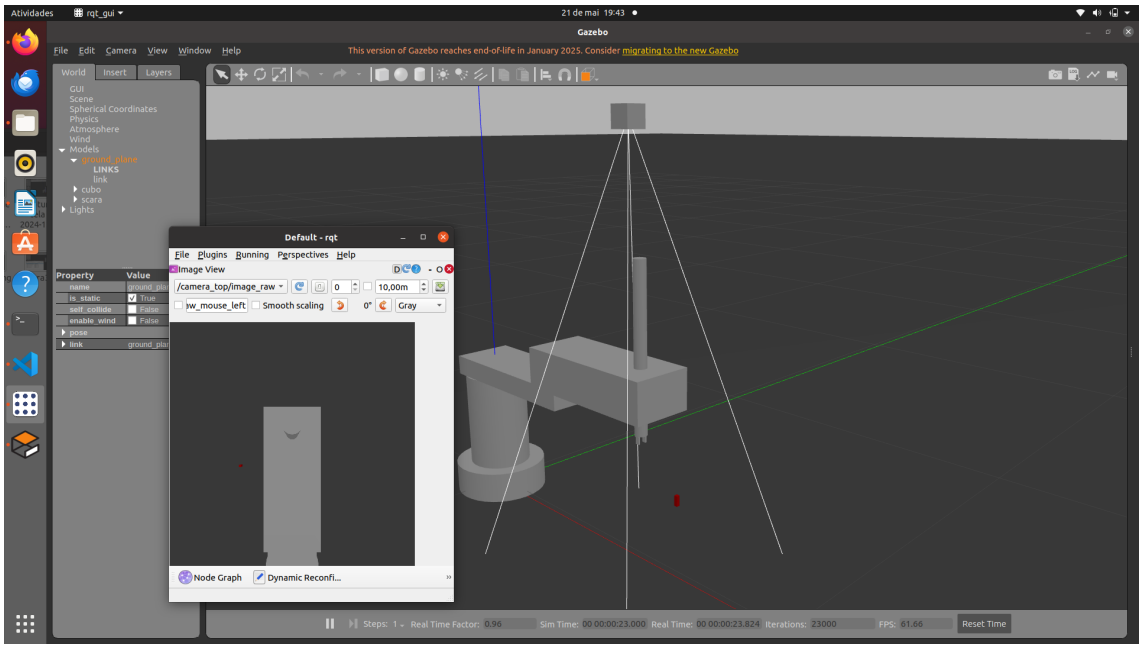
Com essa configuração, a câmera foi inserida no ambiente de simulação e fixada ao link `world`, sendo orientada para capturar a região do chão onde o robô e os objetos estão posicionados. Dessa forma, o sensor passou a fornecer uma visão superior do espaço de trabalho, permitindo acompanhar tanto a posição do manipulador quanto a localização dos objetos no ambiente. A Figura 3.11 apresenta o posicionamento da câmera e seu campo de visão no Gazebo.

Figura 3.11 – Campo de visão da câmera no ambiente de simulação.



O RQT também disponibiliza a visualização da câmera em tempo real, para isso é necessário o uso do *Image View*, disponível na seção *Visualization*. Após isso, basta selecionar o tópico correspondente a câmera `/camera_top/image_raw`, e despausar o mundo no Gazebo, com isso será possível acompanhar a imagem gerada pelo sensor, permitindo verificar se a câmera está posicionada de forma correta e se o campo de visão está adequado, A Figura 3.12 apresenta a imagem obtida pela câmera durante a simulação.

Figura 3.12 – Imagem capturada pela câmera no RQT.



3.6 Configuração do MoveIt e integração com o Gazebo

Após a configuração do robô no ambiente de simulação do *Gazebo*, será realizada a configuração do *MoveIt*. Essa ferramenta foi utilizada neste projeto para auxiliar no planejamento dos movimentos do robô SCARA, permitindo organizar os grupos de movimentação, definir o *end-effector*, configurar os controladores e verificar possíveis colisões durante a geração das trajetórias.

A utilização do *MoveIt* é importante porque ele atua como uma ferramenta de apoio ao planejamento, ele faz isso considerando a estrutura cinemática do robô, os limites das articulações, a posição da garra e a possibilidade de colisões entre os próprios elos ou com elementos do ambiente como em um manipulador robótico, isso contribui para que os movimentos sejam executados de maneira mais organizada.

Embora seja possível movimentar as juntas diretamente por meio dos tópicos dos controladores no ROS, o *MoveIt* permite estruturar o planejamento de trajetórias de forma mais completa, o uso do *MoveIt* também prepara o sistema para as próximas etapas nas quais o robô deve alcançar e manipular objetos de forma automatizada no ambiente de simulação.

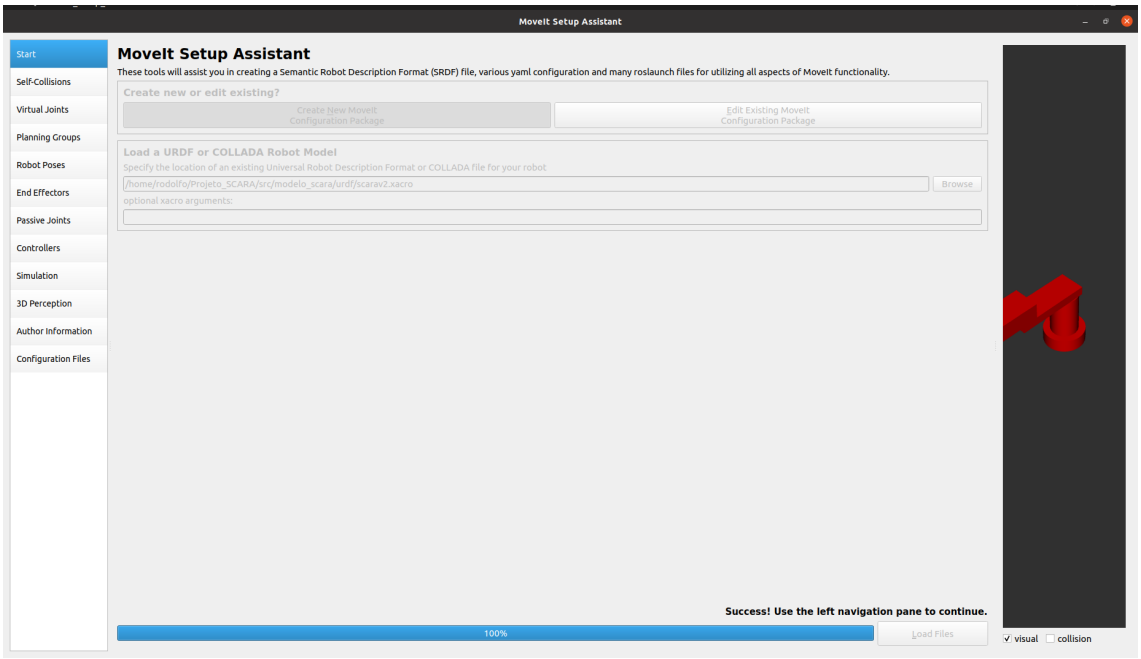
Para iniciar a configuração deve ser criada uma nova pasta dentro do projeto, essa vai ser responsável por armazenas os arquivos gerados pelo assistente do *MoveIt*, ela foi

denominada como `moveit_config`. Além disso a extensão do arquivo `scarav2.urdf`, deve ser alterado para o formato Xacro, isso faz com que o assistente consiga identificar o modelo do projeto, sendo assim o arquivo passa a ser denominado `scarav2.xacro`.

Caso o *MoveIt* não esteja instalado, basta utilizar os comandos para a instalação dos pacotes apresentados nas seção D.1 do Anexo D. Após a instalação, foi inicializado o assistente de configuração, denominado *MoveIt Setup Assistant*. Os comandos utilizados para inicialização do assistente são apresentados nas Seções D.1 e D.2 do Anexo D.

Ao abrir o *MoveIt Setup Assistant*, foi selecionada a opção de criação de um novo projeto. Em seguida, o arquivo `scarav2.xacro` foi carregado para que a ferramenta pudesse interpretar a estrutura do robô, incluindo seus elos, juntas e limites de movimentação. Esse procedimento é apresentado na Figura 3.13.

Figura 3.13 – Modelo do robô carregado no assistente de configuração do *MoveIt*.



Com o modelo carregado, o menu lateral do assistente apresenta diversas opções de configuração. Neste projeto, foram ajustados principalmente os itens relacionados às colisões internas, aos grupos de planejamento, ao *end-effector* e aos controladores. Esses elementos são essenciais para que o *MoveIt* consiga planejar movimentos coerentes com a estrutura física do robô e com os limites das juntas.

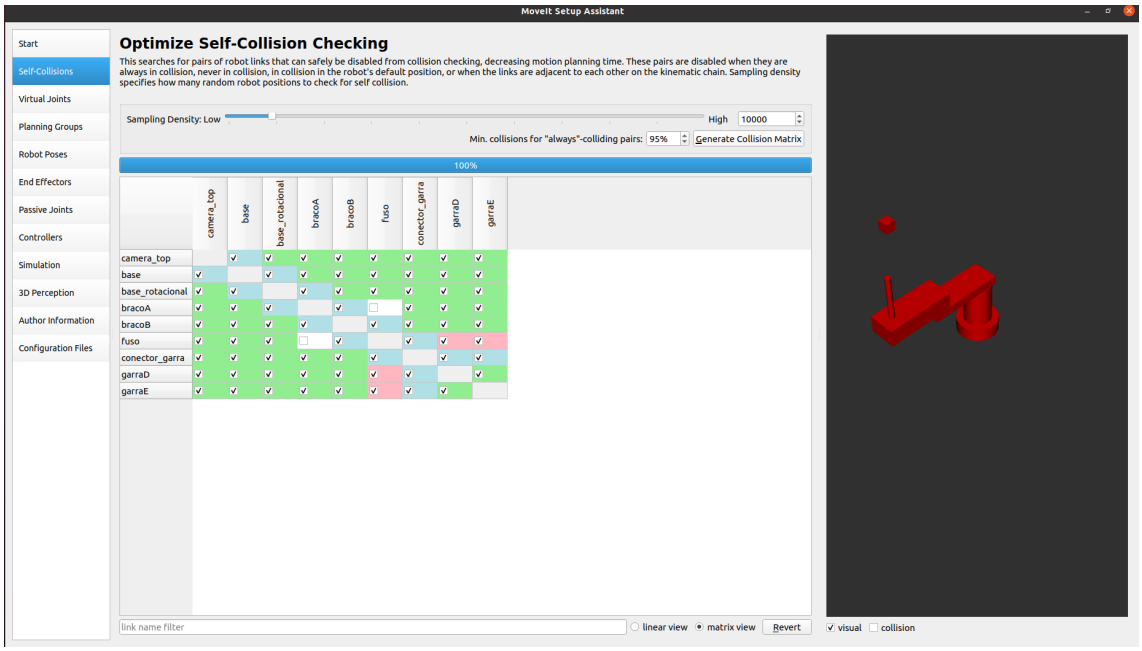
Os principais itens configurados foram:

- ***Self-collisions*:** utilizado para identificar possíveis colisões entre os próprios elos do robô durante o movimento;
- ***Planning groups*:** responsável pela criação dos grupos de movimento do robô, sendo definidos neste projeto o grupo do manipulador e o grupo da garra;
- ***End effectors*:** utilizado para definir a extremidade responsável pela interação com o ambiente, neste caso associada à garra do robô;
- ***Controllers*:** responsável pela associação dos grupos definidos aos controladores utilizados na execução dos movimentos planejados.

A primeira etapa configurada foi a de *self-collisions*. Essa configuração permite que o *MoveIt* identifique quais partes do robô podem entrar em contato entre si durante a movimentação. Embora seja possível definir essas colisões manualmente, neste projeto foi utilizada a opção de autogeração, na qual o próprio *MoveIt* analisa o modelo e determina os pares de elos relevantes para a verificação de colisões.

Essa etapa é importante porque evita que o planejador gere trajetórias em que os elos do robô atravessem sua própria estrutura. Assim, além de melhorar a coerência dos movimentos planejados, a verificação de colisões contribui para reduzir o risco de movimentos inadequados que poderiam danificar o manipulador em uma aplicação real. A configuração das colisões internas é apresentada na Figura 3.14.

Figura 3.14 – Configuração das colisões internas do robô no *MoveIt*.



3.6.1 Definição dos grupos de planejamento

Após a configurar as colisões, a próxima etapa consiste em definir os grupos de planejamento. No *MoveIt*, esses grupos representam conjuntos de juntas e elos que serão movimentados durante o planejamento das trajetórias. Neste projeto, foram definidos dois grupos principais: o grupo do robô, responsável pelo movimento completo do robô SCARA, e o grupo da garra, responsável pela abertura e fechamento do *end-effector*.

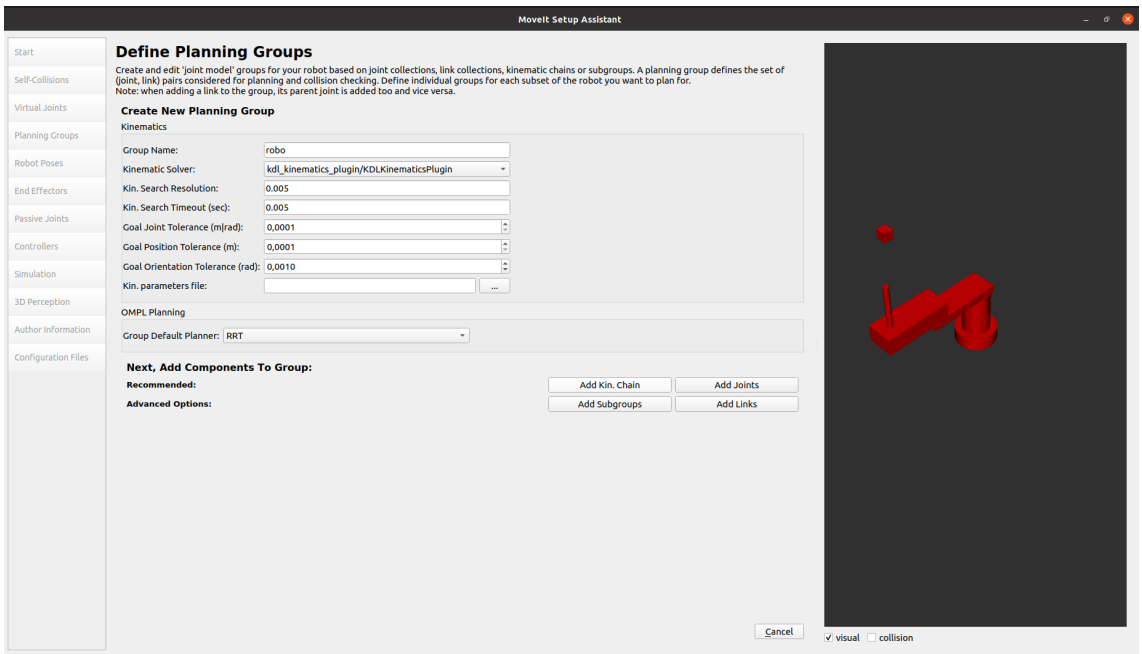
Na configuração do grupo do robô, são definidos parâmetros que influenciam diretamente o modo como o *MoveIt* calcula e avalia os movimentos planejados. A Figura 3.15 apresenta os campos utilizados nessa etapa.

- ***Group Name*:** define o nome do grupo de planejamento, permitindo identificar o conjunto de juntas que será controlado;
- ***Kinematic Solver*:** especifica o algoritmo de cinemática inversa utilizado para calcular as posições das juntas a partir da posição desejada do *end-effector*;
- ***Kin. Search Resolution*:** determina a resolução da busca realizada pelo solucionador cinemático. Valores menores aumentam a precisão da busca, porém podem elevar o tempo de processamento;

- ***Kin. Search Timeout*:** define o tempo máximo disponível para que o solucionador encontre uma solução cinemática antes de indicar falha;
- ***Goal Joint Tolerance*:** estabelece a margem de erro permitida para as posições finais das juntas;
- ***Goal Position Tolerance*:** define a distância máxima aceitável entre a posição final do *end-effector* e a posição desejada;
- ***Goal Orientation Tolerance*:** estabelece a tolerância permitida para a orientação final do *end-effector*;
- ***Group Default Planner*:** permite selecionar o planejador de movimento utilizado para gerar a trajetória, como algoritmos baseados em RRT ou PRM.

Esses parâmetros determinam como o *MoveIt* interpreta o grupo de movimento, calcula as soluções cinemáticas e avalia se a trajetória planejada atende às tolerâncias definidas. Dessa forma, eles contribuem para que o manipulador alcance o alvo com maior precisão, respeitando os limites físicos das juntas e a estrutura cinemática do robô.

Figura 3.15 – Parâmetros do grupo de planejamento do manipulador.

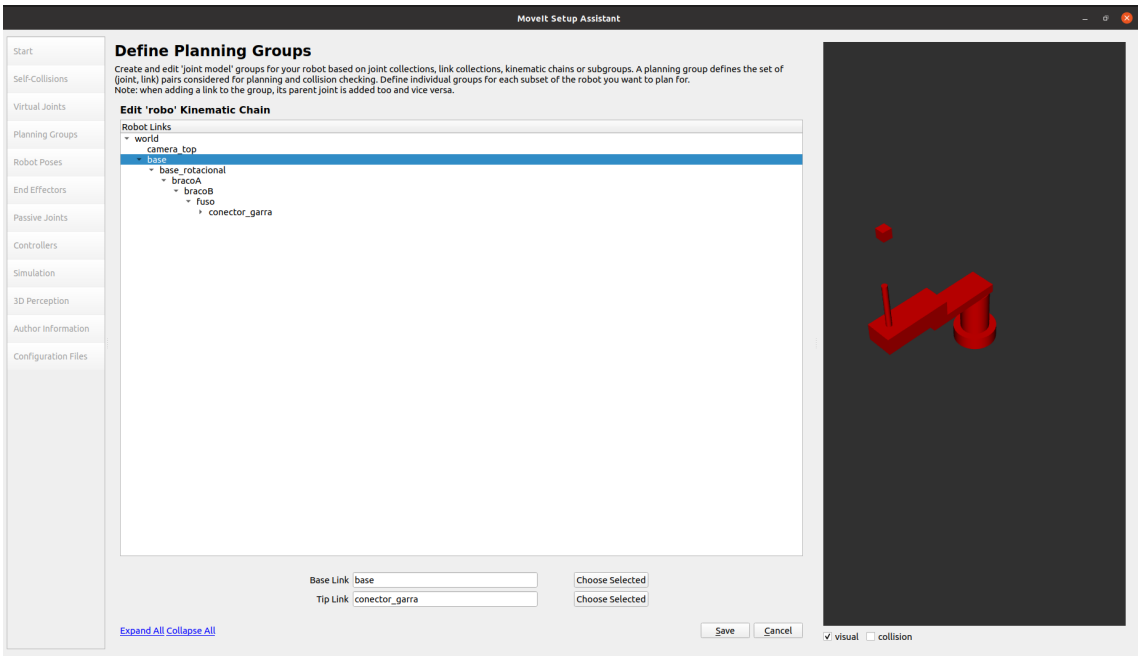


Com os parâmetros principais definidos, a próxima etapa é selecionar as juntas responsáveis pelo movimento do grupo robô. Para isso, foi utilizada a opção *Add Kin. Chain*, ela permite

adicionar uma cadeia cinemática completa a partir da escolha de um link inicial e de um link final, ela inclui automaticamente todos os elos e juntas intermediários, evitando a necessidade de adicioná-los individualmente.

No caso do robô SCARA, essa opção foi utilizada para definir o conjunto de juntas responsável pelo movimento do braço manipulador. Assim, o *MoveIt* passa a reconhecer a relação cinemática entre a base e o efetuador, possibilitando o planejamento de movimentos coordenados e contínuos do manipulador. A definição da cadeia cinemática utilizada no grupo do robô é apresentada na Figura 3.16.

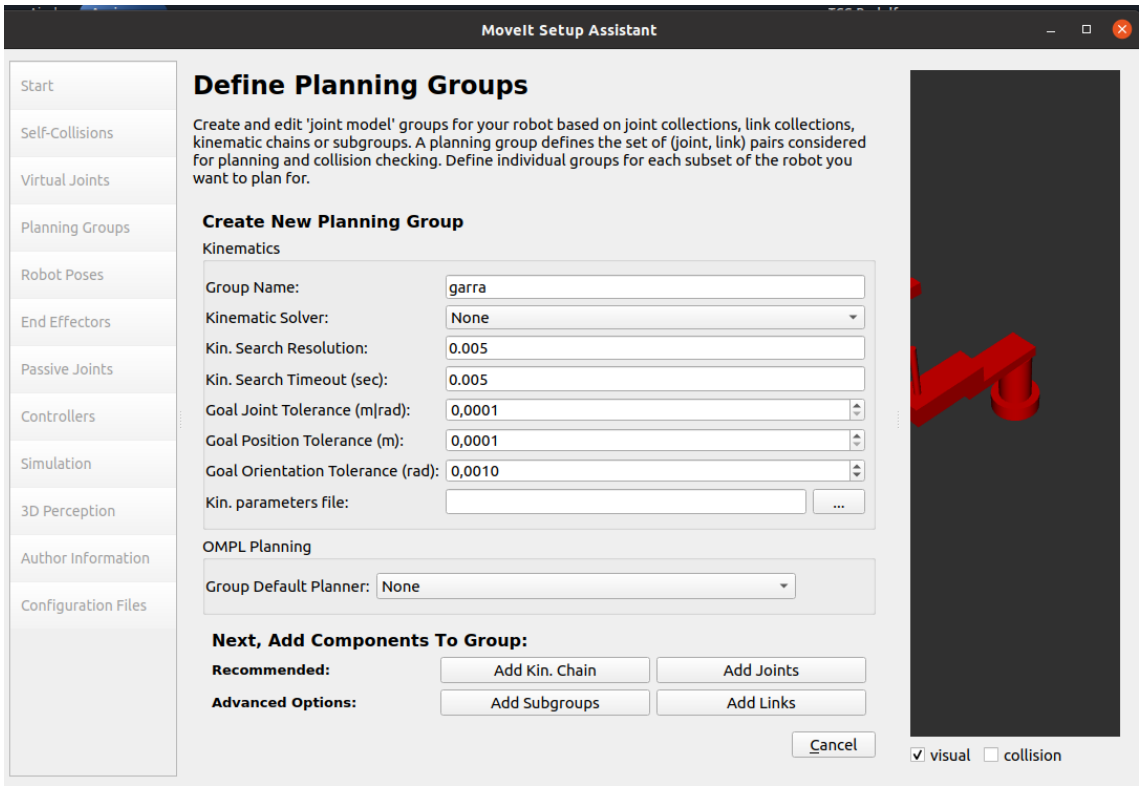
Figura 3.16 – Cadeia cinemática utilizada para definir o grupo do manipulador.



Além do grupo robô, também foi criado o grupo garra, como esse grupo possui uma estrutura mais simples, por ser composto apenas pelas juntas responsáveis pela abertura e fechamento do *end-effector*, não é necessário configurar uma cadeia cinemática completa nem utilizar planejadores de movimento mais complexos para esse grupo.

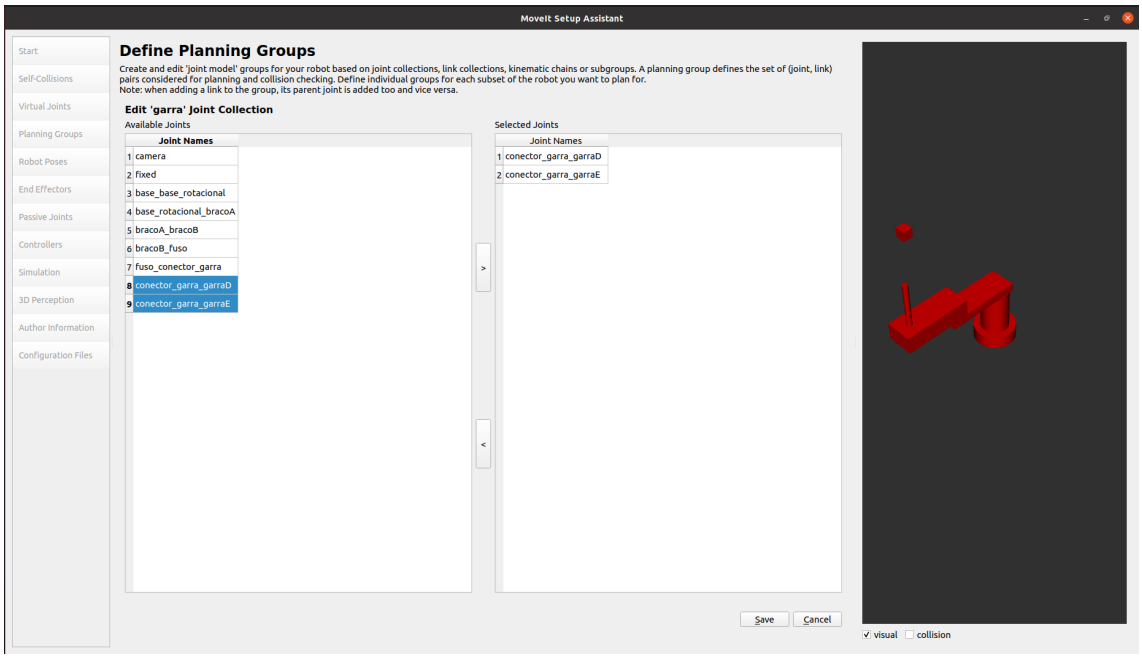
A configuração dos parâmetros do grupo da garra é apresentada na Figura 3.17.

Figura 3.17 – Parâmetros do grupo de planejamento da garra.



Como a garra apresenta uma estrutura simples, sua configuração foi realizada por meio da seleção direta das juntas que compõem o mecanismo de preensão. Essa abordagem é suficiente para permitir o controle da abertura e do fechamento da garra, sem a necessidade de definir uma cadeia cinemática entre links inicial e final. As juntas selecionadas para o grupo da garra são apresentadas na Figura 3.18.

Figura 3.18 – Juntas utilizadas para definir o grupo da garra.



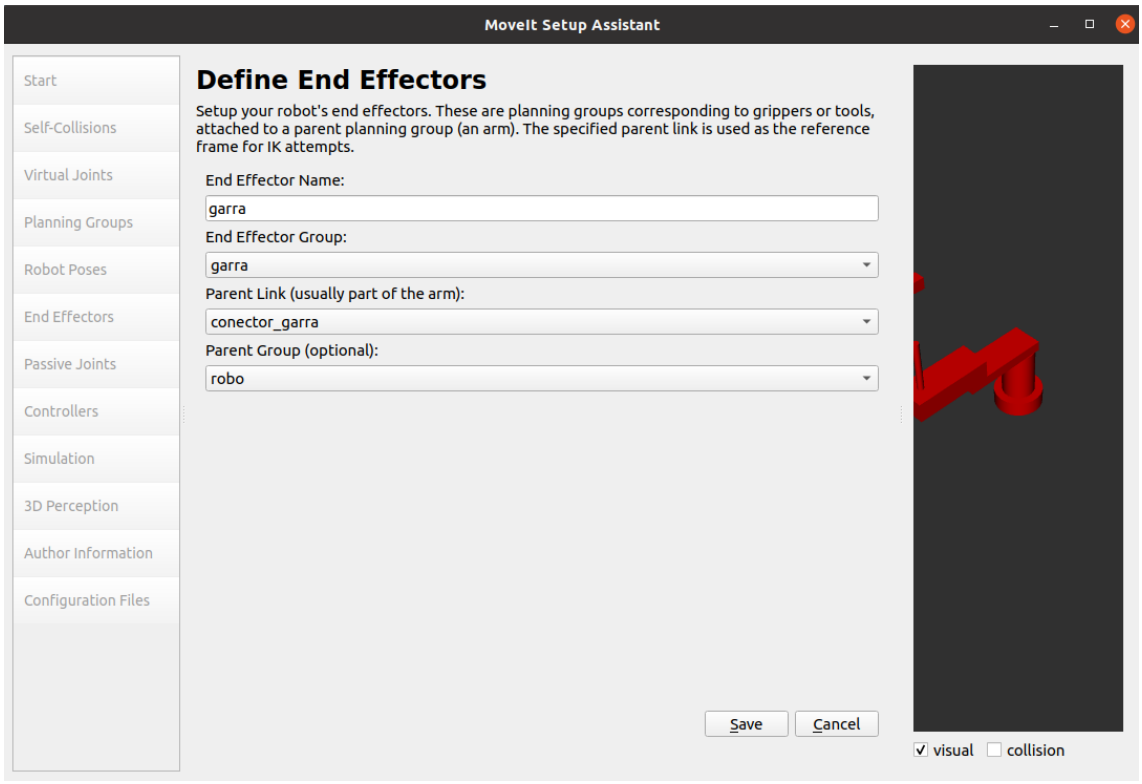
3.6.2 Definição do *end-effector*

Após a definir os grupos de planejamento, foi realizada a configuração do *end-effector*, ele representa a extremidade do robô responsável pela interação com o ambiente. No caso do robô SCARA, essa função está associada à garra, utilizada para alcançar e manipular objetos no espaço de trabalho.

A definição correta do *end-effector* é importante já que o planejamento de movimento o considera como referência para atingir uma posição desejada. Assim, quando é definido um ponto de interesse no espaço, o *MoveIt* calcula as configurações das juntas necessárias para que a garra alcance a posição enviada, respeitando a estrutura cinemática do manipulador.

Nessa etapa, foram definidos o grupo associado à garra, o link final correspondente ao efetuador e o grupo principal ao qual esse efetuador está vinculado. A partir dessa configuração o planejador irá identificar corretamente qual parte do robô deve ser posicionada durante a execução das trajetórias. A definição do *end-effector* é apresentada na Figura 3.19.

Figura 3.19 – Definição do *end-effector* no *MoveIt*.



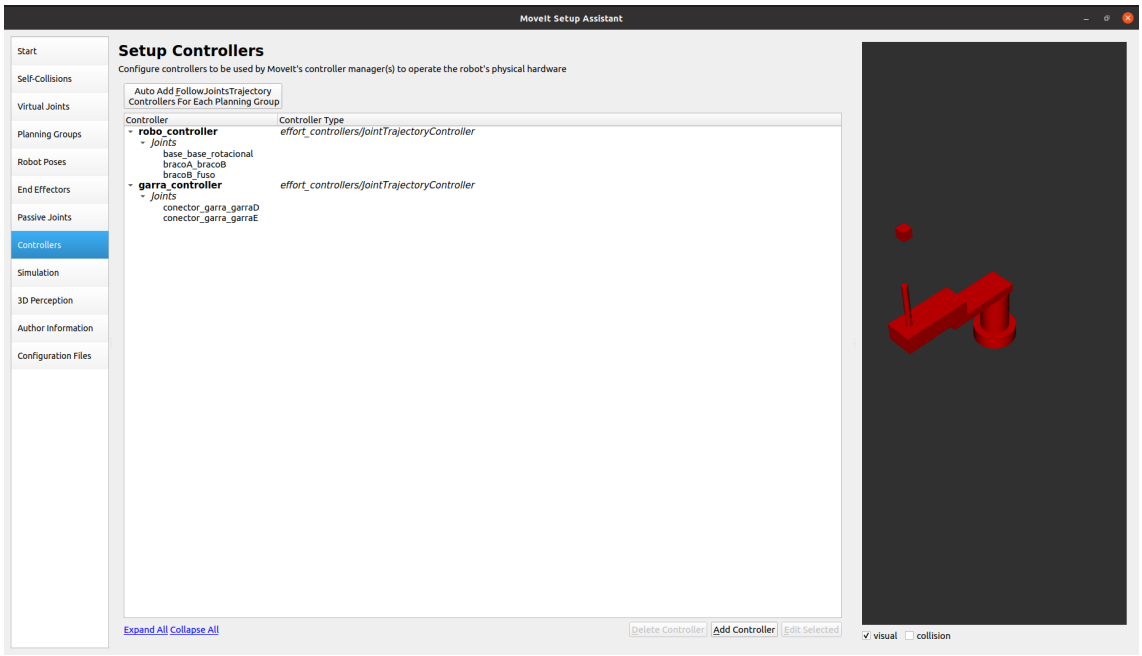
3.6.3 Configuração dos controladores

Com os grupos de planejamento e o *end-effector* definidos, a etapa seguinte consistiu na configuração dos controladores. Esses controladores são responsáveis por associar os grupos criados no *MoveIt* aos comandos que serão utilizados para executar os movimentos planejados.

No assistente de configuração, o *MoveIt* permite gerar automaticamente os controladores a partir dos grupos previamente definidos. Para isso, foi utilizada a opção *Auto Add*, que cria os controladores correspondentes ao grupo do manipulador e ao grupo da garra. Essa etapa facilita a integração entre o planejamento de movimento e a execução das trajetórias no ambiente do ROS.

A configuração dos controladores gerados para os grupos definidos é apresentada na Figura 3.20.

Figura 3.20 – Controladores associados aos grupos de planejamento definidos.

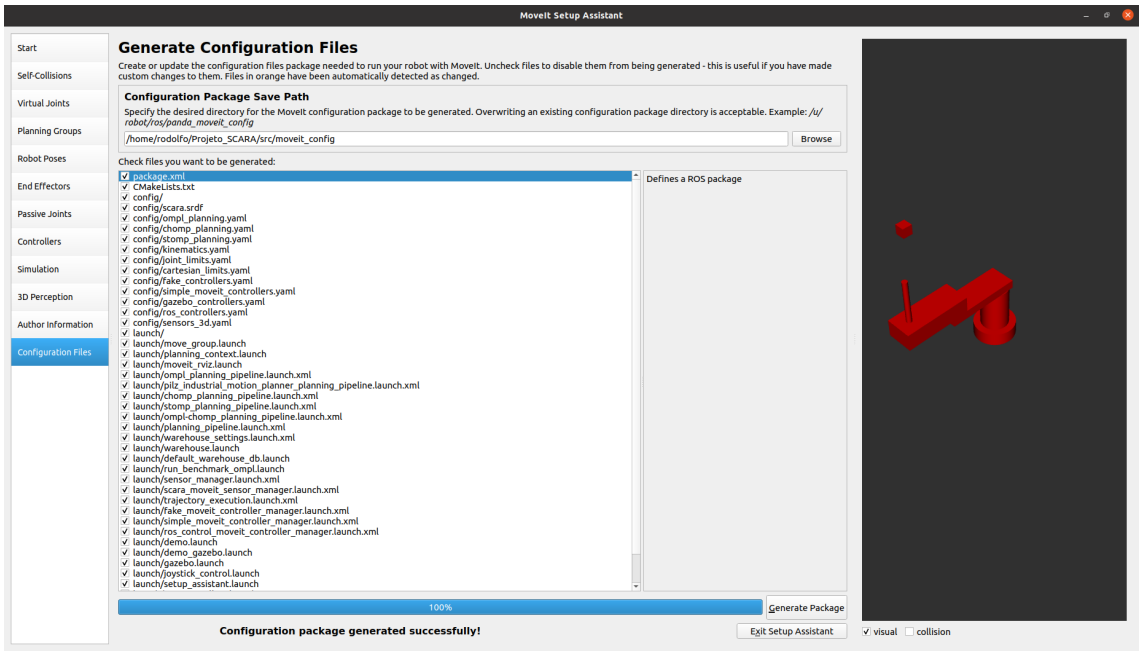


3.6.4 Geração dos arquivos de configuração

A última etapa da configuração consistiu na geração dos arquivos do pacote do *MoveIt*. Para isso, foi preenchido o campo referente ao autor do projeto e, em seguida, acessada a seção *Configuration Files*. Nessa etapa, o caminho da pasta `moveit_config`, criada anteriormente, foi indicado como local de destino dos arquivos gerados.

Concluindo esse procedimento, o assistente vai exportar todas as configurações realizadas para utilizar o ambiente de planejamento, com esses arquivos é possível carregar o modelo no *RViz* e utilizar a interface de planejamento do *MoveIt* para testar movimentos do manipulador. A etapa de salvamento do pacote é apresentada na Figura 3.21.

Figura 3.21 – Geração dos arquivos de configuração do pacote do *MoveIt*.



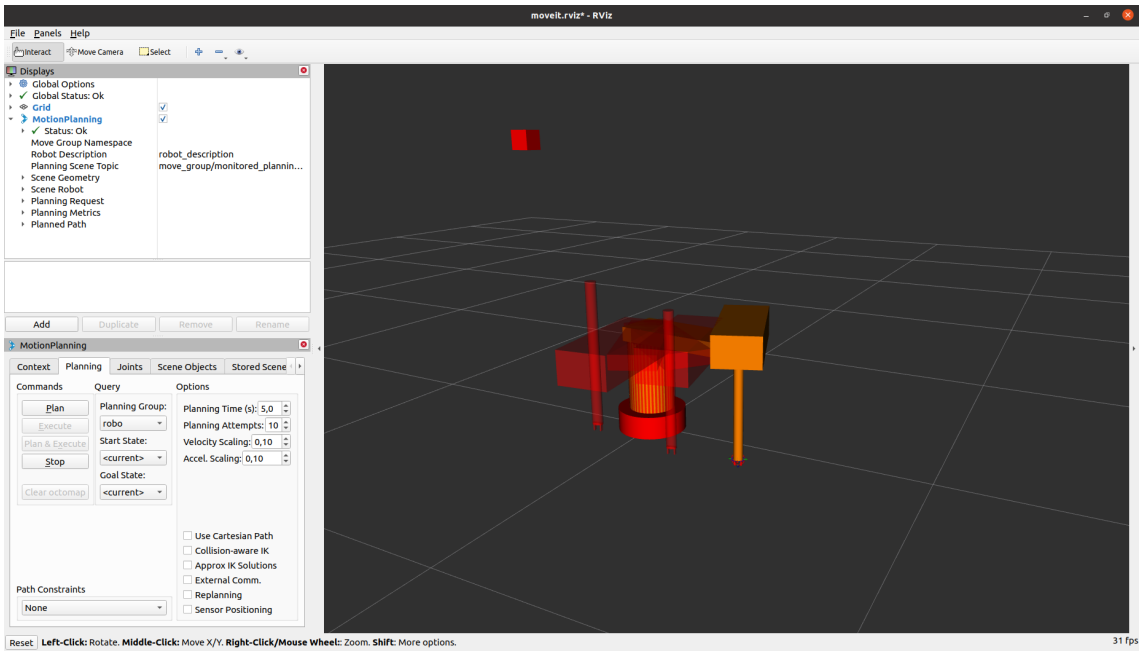
3.6.5 Validação do pacote Moveit

Após a geração dos arquivos de configuração do *MoveIt*, foi realizada uma validação inicial com o objetivo verificar se os grupos de planejamento, as juntas e os controladores haviam sido configurados corretamente, isso foi possível a partir da execução da demo do *MoveIt*. O comando utilizado para abrir a demonstração do pacote é apresentado na Seção D.3 do Anexo Anexo D.

Após a iniciar a demonstração, a interface do RViz é carregada com os recursos de planejamento do *MoveIt*, com isso, é possível selecionar os grupos criados anteriormente, como o grupo robô e o grupo garra. Ao escolher um grupo e acessar a seção *Joints*, é possível alterar os valores das juntas para definir uma posição desejada.

Ao retornar para a aba *Planning* e selecionar a opção *Plan*, o *MoveIt* calcula uma trajetória entre a posição atual do robô e a posição desejada. Essa trajetória gera uma animação de previsão do movimento antes de executá-lo, é exibida no RViz. Com isso é possível validar a coerência dos grupos definidos e observar o comportamento do robô durante o planejamento de trajetórias. A Figura 3.22 apresenta um teste de movimentação realizado para o grupo do manipulador.

Figura 3.22 – Teste de movimento do grupo do manipulador no *RViz*.



3.6.6 Integração entre MoveIt e Gazebo

Após a validação inicial no RViz, foi realizada a integração entre o *MoveIt* e o *Gazebo*. Essa etapa foi necessária para que as trajetórias planejadas no ambiente do *MoveIt* pudessem ser enviadas aos controladores responsáveis pela movimentação do robô no ambiente de simulação física.

Para isso, foram realizadas modificações em alguns arquivos do pacote `moveit_config` e do pacote principal do robô. O primeiro ajuste foi feito no arquivo `ros_controllers.yaml`, responsável por definir os controladores utilizados pelo *MoveIt*. Nesse arquivo, as configurações geradas automaticamente foram substituídas pelos parâmetros adequados aos controladores utilizados no *Gazebo*.

Além disso, foi necessário criar o arquivo `scara_moveit_controller_manager.launch` dentro da pasta `launch` do pacote do *MoveIt*. Ele é responsável por carregar o gerenciador de controladores configurado para o robô, isso permite que o *MoveIt* reconheça quais controladores devem ser acionados durante a execução das trajetórias.

Também foi criado um pacote de configuração dentro pacote `modelo_scara` nomeado como `controletrajetoria.config`, ele contém parâmetros de orientação e controle das juntas durante a execução das trajetórias, para que os movimentos planejados sejam enviados

corretamente aos controladores.

Após isso, foi criado o arquivo `moveit_gazebo.launch`, a fim de inicializar de forma conjunta o *MoveIt* e o *Gazebo*, ele é responsável por carregar os controladores e os recursos de planejamento de movimento em uma única execução, facilitando a integração entre o planejamento e simulação.

Por fim, foi necessário modificar o tipo de transmissão definido no arquivo `juntas.xacro`, alterando o modo de atuação de *effort* para *position*. Essa alteração foi realizada para permitir que os comandos de posição gerados pelo *MoveIt* fossem interpretados corretamente pelos controladores utilizados no *Gazebo*. A partir dessa modificação, os ambientes conseguem reconhecer os controladores das juntas do robô, possibilitando executar trajetórias planejadas no ambiente de simulação.

4 PACOTES PARA IDENTIFICAÇÃO E *PICK AND PLACE*

4.1 Criação do pacote para codificação e visão

Após a integração entre o *MoveIt* e o *Gazebo*, a próxima etapa é criar um pacote específico para o desenvolvimento dos códigos de movimentação e visão computacional. Esse pacote foi criado para permitir a implementação de códigos em *Python* capazes de controlar o robô para manipulação de objetos e processar as imagens da câmera no ambiente de simulação.

Para isso, foi criado o pacote `scara_move` dentro do espaço de trabalho do projeto. Para que os códigos funcionem da maneira correta é necessário criar um pacote com algumas dependências, entre elas estão verb|rospy|, utilizado para a criação de nós em *Python*; o `moveit_commander`, utilizado para comunicação com os recursos do *MoveIt*; o `sensor_msgs` e o `image_transport`, utilizados no recebimento de imagens; e o `cv_bridge`, responsável por converter imagens do ROS para um formato compatível com o *OpenCV*. Os comandos utilizados para criação e compilação do pacote são apresentados na Seção E.1 do Anexo E.

Com a criação e compilação corretado do pacote, foi desenvolvido dentro da pasta `src` o primeiro código de teste denominado `pegarlargar.py`, esse código implementa uma rotina simples de *pick and place*, onde o robô movimenta suas juntas para alcançar o objeto e posicioná-lo em outro local, esse código não faz uso da visão nem da cinemática ele é baseado em testes para alcançar o objeto em uma coordenada já estabelecida.

Como o *script* não utiliza dos cálculos da cinemática, as juntas recebem uma matriz de valores para que sejam movidas diretamente, onde cada linha representa o valor desejado de cada junta. Esse é um código mais simples que foi desenvolvido com o intuito de testar os movimentos das juntas e controladores, assim como validar a comunicação com o *MoveIt*.

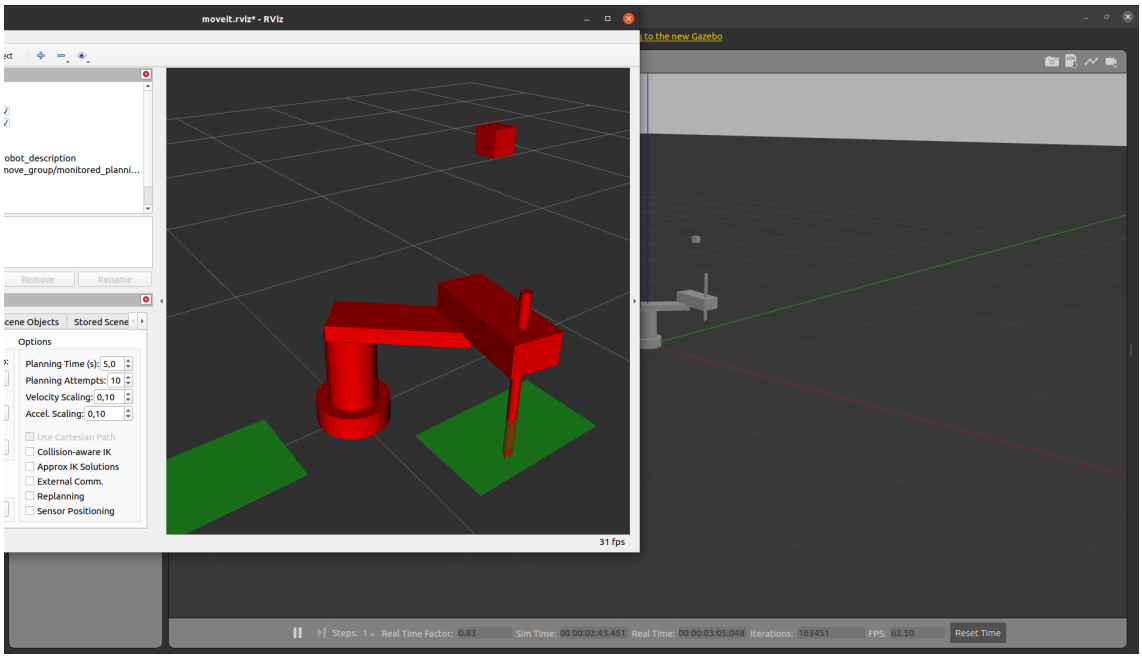
O programa, além de enviar comandos de movimentação, também cria na cena *MoveIt* objetos de colisão. Esses objetos representam o espaço onde a peça está posicionada inicialmente e aonde será depositada. Essa representação é importante porque permite que o planejamento considere elementos presentes na cena durante a movimentação.

Após a criação do código, foi necessário conceder permissão de execução ao arquivo para que ele pudesse ser executado como um nó do ROS. Os comandos utilizados para tornar o arquivo executável e para executar o programa são apresentados nas Seções E.2 e E.3 do Anexo E.

Para que o *script* funcione corretamente, é necessário primeiro rodar o launch `moveit_gazebo.la` Após isso é necessário abrir uma nova janela no terminal e instanciar o ambiente do projeto, assim será possível executá-lo para testes. É importante lembrar que a permissão de execução do código, precisa ser concedida apenas uma vez, não sendo necessário repetir após alterações no código

Com a execução do código, foi possível verificar o ambiente criado e observar a movimentação do robô durante a tarefa de *pick and place*. A Figura 4.1 apresenta o teste realizado com o grupo do manipulador.

Figura 4.1 – Teste inicial de movimentação do robô em uma tarefa de *pick and place*.



Com o robô se movimentando corretamente, a próxima etapa consiste em identificar o objeto no ambiente por meio da câmera e repassar suas coordenadas ao código de manipulação. Como a rotina inicial de *pick and place* utiliza posições articulares predefinidas, será necessário desenvolver uma versão mais completa, capaz de calcular automaticamente as posições e rotações das juntas a partir das coordenadas do objeto. Essa evolução permitirá integrar a visão computacional à cinemática inversa apresentada na Subseção 3.4.3, tornando o processo de captura e posicionamento do objeto mais automatizado.

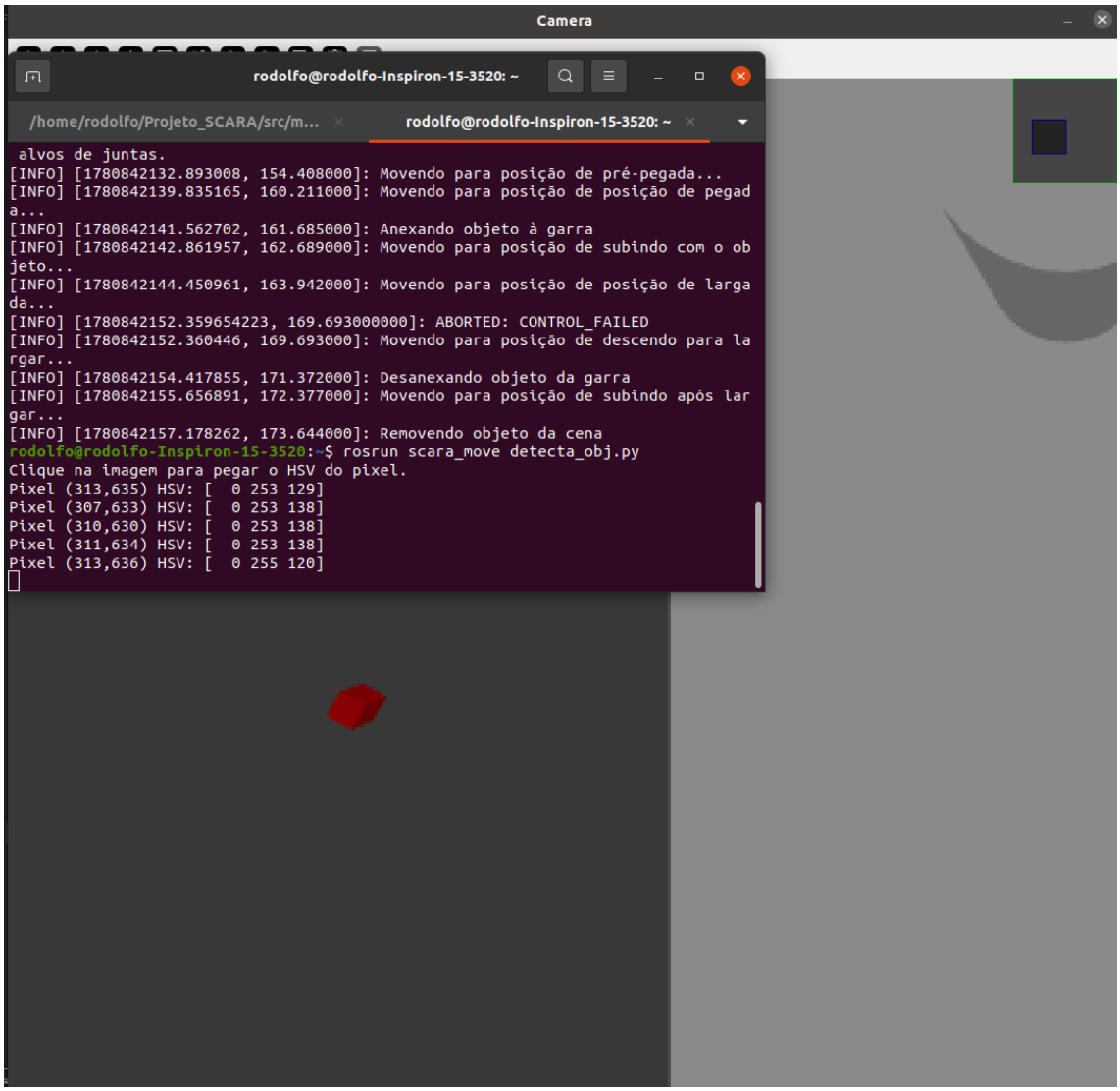
4.2 Identificação do objeto por visão computacional

Para a visão computacional foi utilizado o *Opencv*, para identificar o objeto no ambiente de simulação, para isso é necessário determinar a faixa de cor do objeto, isso é possível ao criar uma máscara que é capaz de segmentá-lo na imagem capturada pela câmera. Essa etapa é importante para que o objeto seja o ponto de interesse da cena e para que sua posição seja calculada com maior precisão.

Para realizar a criação dessa máscara, foi foi desenvolvido o código `detectahsv.py`, armazenado na pasta `src` do pacote `scara_move`. Esse código ao ser executado permite obter os valores de cor do objeto ao clicar sobre o mesmo, assim a faixa de cor é mostrada no Prompt de Comando (CMD). A partir desses valores, é possível definir a faixa de cores que serão utilizadas na máscara de identificação. Os comandos utilizados para tornar o *script* executável seguem o mesmo procedimento apresentado no Anexo E.

A Figura 4.2 apresenta a etapa de verificação da faixa de cor do objeto.

Figura 4.2 – Verificação da faixa de cor do objeto no espaço HSV.



Após a definição da faixa de cor, foi criado um novo pacote denominado `position_tracker`. Esse pacote tem como finalidade disponibilizar um serviço no ROS para enviar ao código de manipulação as coordenadas do objeto identificadas pelo sistema de visão. Dessa forma, o código responsável pelo *pick and place* pode solicitar a posição do objeto apenas quando necessário, sem depender de uma publicação contínua de coordenadas. Os comandos utilizados para criação e compilação desse pacote são apresentados na Seção E.4 do Anexo E.

Com o pacote criado, foi desenvolvido o serviço responsável por fornecer as coordenadas do objeto detectado. Para isso, foi criada a pasta `srv` dentro do pacote `position_tracker`. Dentro da mesma, foi desenvolvido o arquivo `GetPosition.srv`. Esse é um arquivo simples que define a estrutura da chamada de serviço, permitindo que outro nó do ROS solicite a posição

atual do objeto. Além disso, o arquivo `CMakeLists.txt` foi modificado para incluir a geração do serviço durante a compilação do pacote.

Em seguida, foi criado o programa `pegaposicao.py` no pacote `scara_move`. Ele é responsável por processar a imagem da câmera, identificar o objeto vermelho e armazenar suas coordenadas no sistema de referência do mundo. Para isso, o *script* utiliza a faixa HSV definida anteriormente, cria uma máscara para segmentar o objeto, identifica os contornos presentes na imagem e calcula o centro do objeto detectado.

Para converter a posição do objeto na imagem para coordenadas do mundo, foi realizada uma projeção de um raio tridimensional a partir do sistema óptico da câmera. Esse raio é gerado com base na posição do objeto na imagem e projetado no espaço 3D.

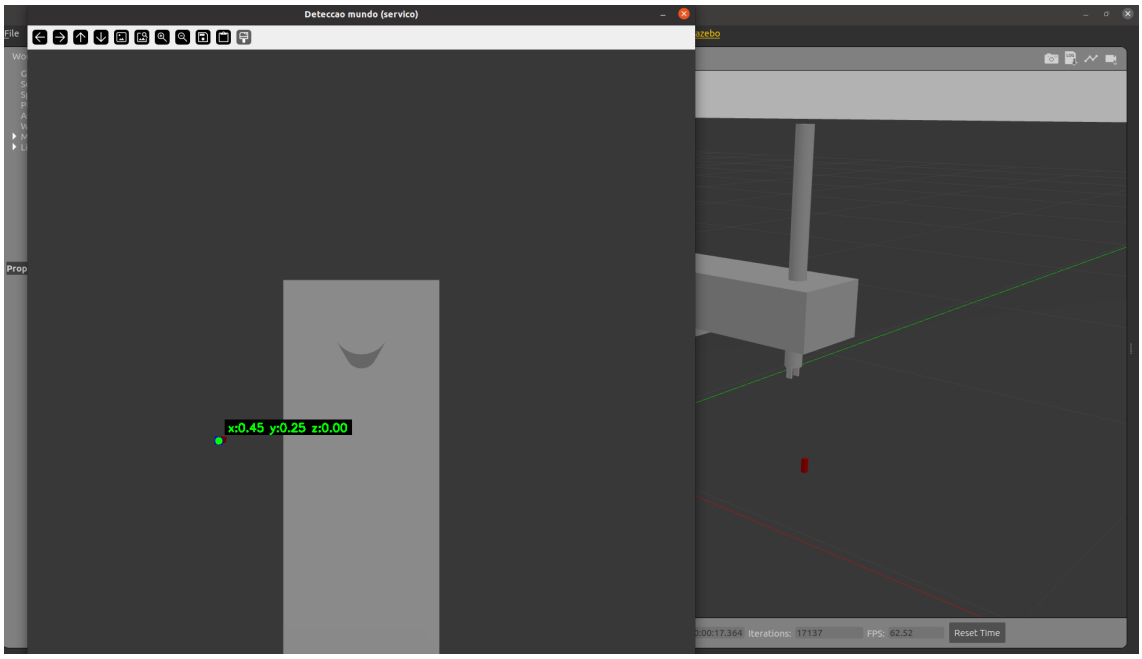
Com isso é calculado sua interseção com o chão, permitindo estimar a posição real do objeto dentro da simulação. Para que a posição seja enviada corretamente, o raio é convertido para *frame* da câmera no padrão do ROS e depois é transformada para o *frame* global por meio das transformadas disponíveis no sistema, para realizar essas conversões foram utilizados alguns *plugins* do próprio ROS.

Após esse procedimento, o *script* armazena internamente as coordenadas do objeto detectado, desde que a interseção seja válida. Caso nenhum objeto seja identificado nos *frames*, as coordenadas são redefinidas como *None*, para evitar que o sistema utilize coordenadas desatualizadas, foi aplicado um pequeno ajuste na coordenada Y, subtraindo $0{,}01\,m$, para compensar um desvio sistemático observado durante os testes.

Quando o serviço de posição é chamado, o nó retorna as últimas coordenadas válidas armazenadas. Caso um objeto tenha sido detectado, os valores correspondentes às coordenadas são enviados. Caso contrário, o serviço emite um aviso informando que nenhum objeto foi encontrado e retorna valores nulos. Isso reduz o tráfego de mensagens e mantém a capacidade de localização visual em tempo real permitindo que a posição do objeto seja fornecida apenas quando o robô precisa realizar a captura.

A Figura 4.3 apresenta a execução do código de visão e a posição do objeto identificada no sistema de referência do mundo.

Figura 4.3 – Identificação da posição do objeto no sistema de referência do mundo.



4.2.1 Rotina de pick and place com visão e cinemática inversa

Para realizar a captura do objeto a partir das coordenadas obtidas pelo sistema de visão, foi desenvolvido o arquivo `pegarlargar.py` dentro do pacote `scara_move`. Esse *script* se comunica com o serviço de visão criado anteriormente, recebendo as coordenadas do objeto no espaço de trabalho e utilizando a cinemática inversa do robô SCARA, apresentada na Subseção 3.4.3, para determinar as posições articulares necessárias ao movimento.

Ao receber as coordenadas do objeto no mundo, o programa executa uma sequência de movimentos para se aproximar do objeto, pegá-lo e levá-lo até um ponto fixo. Os ângulos θ_1 e θ_2, obtidos pela cinemática inversa, são ajustados em relação aos valores `theta1_ref` e `theta2_ref`, definidos a partir da posição de repouso do robô antes de serem enviados às juntas, isso foi realizado a fim de tornar a movimentação mais compatível com a posição de partida do manipulador.

Antes de executar qualquer movimento, o programa verifica se os valores calculados se estão dentro dos limites físicos que foram definidos para o robo no URDF. Caso os cálculos ultrapassem os limites, o movimento não é executado exibindo uma mensagem de erro, evitando assim que o robô tente executar uma trajetória que danifique sua estrutura.

A rotina de *pick and place* foi estruturada em etapas sequenciais, inicialmente o programa

adiciona as mesas que determinam o começo e o fim da rotina de movimento do *MoveIt*. Em seguida, o serviço de visão é chamado para obter as coordenadas atuais do objeto. Com essas informações, uma pequena caixa é adicionada à cena de planejamento para representar o objeto a ser manipulado dentro do *MoveIt*.

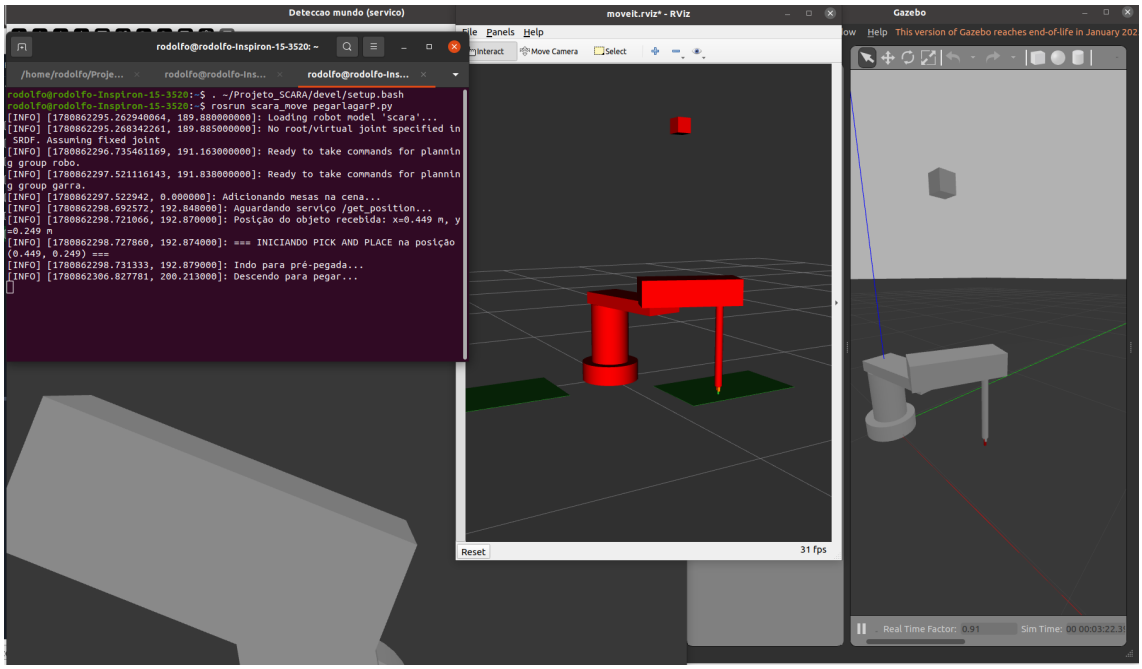
Ao objeto ser identificado pela visão, o robô começa sua rotina de movimentos, se deslocando inicialmente para uma posição de pré-pegada, ficando logo acima das coordenadas do objeto. Após isso a garra é aberta e o manipulador desce até a altura de pegada, ao alcançar a posição a garra é fechada e o objeto é associado ao *end-effector*. Com isso, o *MoveIt* passa a considerar que o objeto está preso à garra, acompanhando seus movimentos e sendo verificado as colisões. Após a preensão o braço sobe novamente e transporta o objeto.

A etapa de depósito utiliza uma posição fixa, o manipulador primeiro se desloca para uma posição de pré-largada e em seguida desce para largar o objeto. Após a liberação, o braço sobe novamente e encerra a sequência de movimentação.

a fim de garantir um movimento mais suave foi utilizada uma velocidade menor para o movimento, o braço foi configurado para operar com 30% da velocidade disponível para o moveit, enquanto que a garra foi limitada a 50%. As funções de movimentação combinam cálculos cartesianos com recursos de planejamento do *MoveIt*, fazendo com que as trajetórias sejam contínuas e compatíveis com os obstáculos da cena.

Com isso o código integra todas as etapas propostas pelo projeto: a identificação do objeto pela visão computacional, a cinemática inversa para movimentação das juntas, e o planejamento de movimento com o *MoveIt*, responsável por executar a trajetória de forma organizada. A Figura 4.4 apresenta a execução da rotina de captura e posicionamento do objeto no ambiente de simulação.

Figura 4.4 – Execução da rotina de *pick and place* a partir das coordenadas obtidas pelo sistema de visão.



5 RESULTADOS

5.1 Resultados da comparação dos controladores PID

Após a modelagem matemática das juntas e a configuração dos controladores no ambiente de simulação, foi desenvolvido um código no *MATLAB*, denominado de `PID_Juntas.m` para comparar o desempenho de três conjuntos de ganhos PID aplicados ao robô SCARA, o *script* está disponível dentro da pasta do projeto.

O objetivo desse código é verificar comportamento das juntas para os ganhos calculados por meio de alocação de polos, ganhos obtidos empiricamente no ambiente de simulação utilizando os *pluguins* do Gazebo e um terceiro conjunto de ganhos que seria a combinação ganhos calculados e os obtidos de forma empírica.

O código desenvolvido considera os parâmetros físicos da juntas como a inércia e sua massa. A partir desses parâmetros, foram montadas funções de transferência simplificadas a fim de representar o comportamento de cada junta.

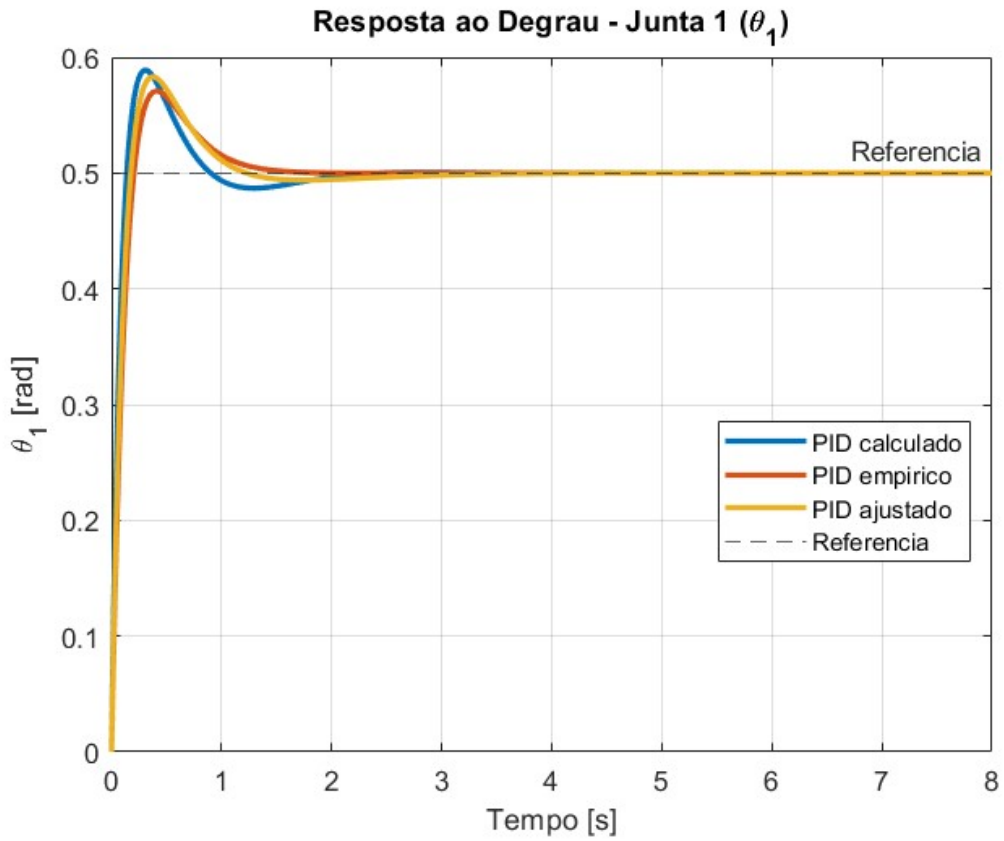
Para o projeto do controlador calculado, foi adotado um coeficiente de amortecimento $\zeta = 0{,}7$ e em um tempo de acomodação desejado de $2\,s$. A partir disso foi calculado uma frequência natural ω_n e definido um terceiro polo auxiliar, utilizado na alocação dos polos da malha fechada, assim foram obtidos os ganhos K_p, K_i e K_d para cada junta do robô.

Como foi citado, além dos ganhos calculados, foram obtidos ganhos por ajuste empírico no ambiente de simulação, esses valores foram definidos a partir do ajuste por observação dos valores desejado e real apresentados pelos plugins do Gazebo, a fim de ser obter o melhor desempenho e estabilidade. O código também implementa a combinação entre os ganhos obtidos empiricamente e os calculados.

Os dados dos ganhos foram obtidos a partir da resposta ao degrau de cada junta, para as juntas θ_1 e θ_2, foi utilizada uma referência de $0{,}5\,rad$. Para a junta prismática q_3, a referência foi de $0{,}10\,m$. As respostas foram analisadas considerando o valor final, o erro em regime permanente, o tempo de subida, o tempo de acomodação e o sobressinal.

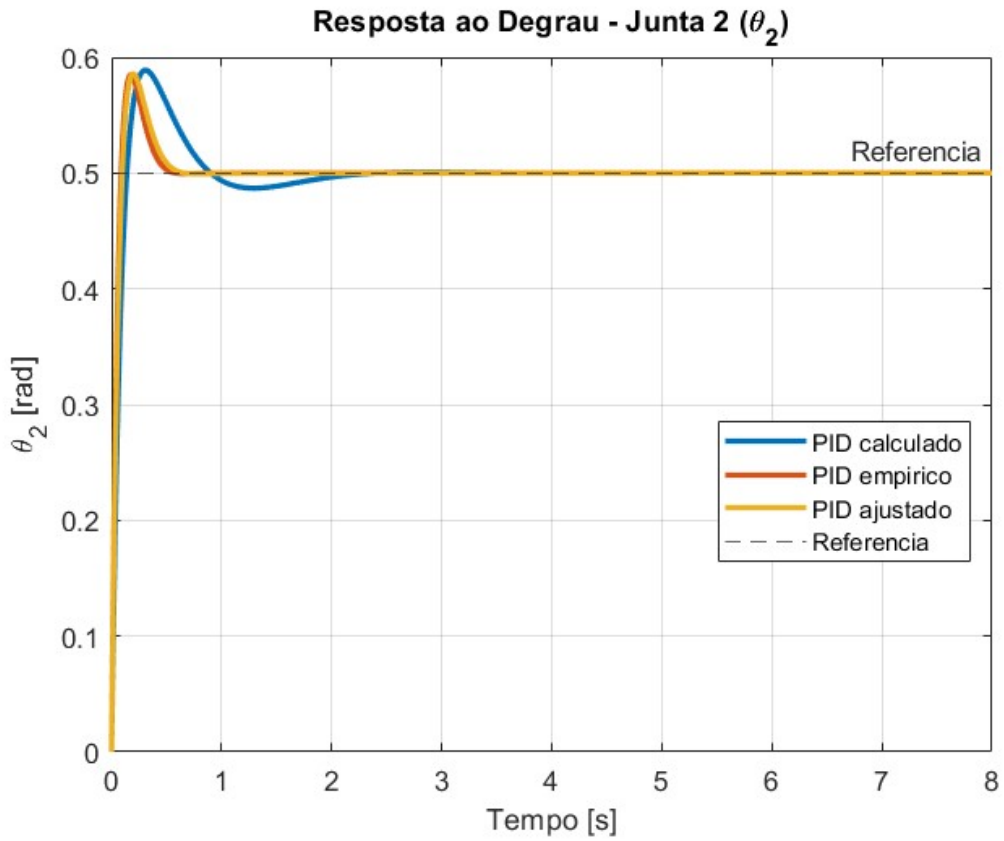
Os gráficos comparativos das respostas das juntas apresentados nas Figuras 5.1, 5.2 e 5.3 permitem avaliar os comportamentos do métodos de sintonia utilizados, a analise dos gráficos serve como uma etapa complementar à simulação, permitindo observar de maneira gráfica e quantitativa o desempenho dos controladores antes de sua aplicação final no modelo do robô.

Figura 5.1 – Resposta ao degrau da junta θ_1 para os controladores calculado, empírico e ajustado.



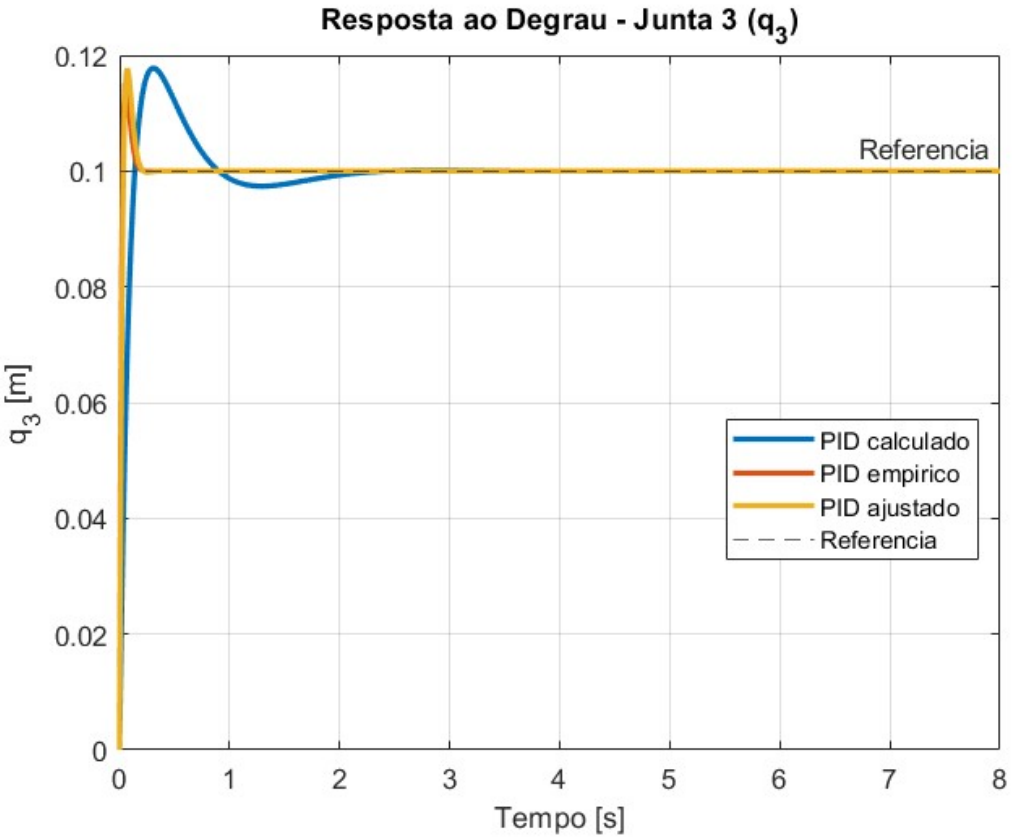
Na Figura 5.1, é possível observar que todos os controladores conseguiram alcançar a referência estabelecida. É possível notar que o controlador obtido pelo modelo matemático apresentou uma resposta rápida com o maior sobressinal inicial e uma leve oscilação antes da estabilização. Já o controlador empírico demonstra uma resposta mais suave, com menor oscilação após o pico inicial. O controlador ajustado como era de se esperar representa um comportamento intermediário, porém se aproximando mais do método empírico, mas mantendo parte da rapidez obtida pelo calculado.

Figura 5.2 – Resposta ao degrau da junta θ_2 para os controladores calculado, empírico e ajustado.



É possível observar que na Figura 5.2, que o controlador calculado apresentou uma maior oscilação, inicialmente ultrapassando a referência e logo após ficando levemente abaixo dela antes da acomodação. Os controladores empírico e ajustado apresentaram respostas mais rápidas e com menor tempo de acomodação visual. Isso indica que, para a segunda junta, os ganhos obtidos empiricamente se ajustaram melhor à dinâmica observada na simulação.

Figura 5.3 – Resposta ao degrau da junta prismática q_3 para os controladores calculado, empírico e ajustado.



Na Figura 5.3, é possível nota que o controlador calculado apresentou uma resposta mais lenta e com maiores oscilações, por outro lado, os controladores empírico e ajustado atingiram rapidamente o valor desejado, com menor tempo de estabilização. Esses resultados mostram que, para a junta prismática, os ganhos empíricos foram melhores à resposta esperada no ambiente de simulação.

De modo geral, os três conjuntos de ganhos foram capazes de conduzir as juntas às referências estabelecidas, entretanto, o controlador calculado apresentou uma resposta esperada de acordo com o modelo matemático adotado, mas com maior oscilação em algumas juntas.

Já o controlador empírico demonstrou um melhor comportamento prático em termos de acomodação, principalmente nas juntas θ_2 e q_3. O controlador ajustado, por sua vez, representou uma solução intermediária entre a teoria e o ajuste prático, mantendo estabilidade e resposta próxima à referência.

Apesar dos gráficos obtidos demonstrarem que todos os controladores conseguiram alcançar a referência, ao executar o simulador, os ganhos calculados para a junta prismática não

foram capazes de manter a junta na posição desejada, tornando inviável sua utilização prática no controle dessa articulação.

Esse comportamento indica que, para a junta prismática, o modelo matemático simplificado utilizado no cálculo dos ganhos não conseguiu representar os efeitos presentes na simulação dinâmica, como a ação da gravidade, limitações do atuador, propriedades de contato e comportamento do controlador implementado no *Gazebo*.

Dessa forma, embora o controlador calculado tenha apresentado resposta estável na simulação matemática, ele não se mostrou adequado para aplicação direta no modelo simulado do robô. Por outro lado, os controladores empírico e ajustado apresentaram melhor desempenho para a junta q_3, sendo capazes de manter a junta prismática em uma posição estável durante os testes.

Pode-se concluir que o projeto matemático dos controladores serviu como um ponto de partida para definir os ganhos, enquanto que o ajuste empírico foi importante para adequar melhor o projeto, essa comparação reforça a necessidade da validação dos parâmetros calculados, uma vez que as simplificações do modelo matemático não conseguem representar completamente os efeitos presentes na simulação dinâmica do manipulador.

5.2 Testes de funcionamento e coleta de dados

Após a integração entre o sistema de visão, a cinemática inversa, o planejamento de movimento, o PID encontrado e o ambiente de simulação, foram realizados testes com o robô SCARA executando a tarefa de *pick and place*. O objetivo dessa etapa foi verificar se o manipulador era capaz de identificar o objeto, calcular as posições articulares necessárias, realizar a aproximação, capturar o objeto e transportá-lo até o ponto de depósito definido.

Para registrar os dados dos testes, foram desenvolvidos dois códigos auxiliares. O primeiro foi criado dentro do pacote `scara_move`, com o nome `pegarlagarR.py`, ele é responsável pela criação do arquivo *rosbag*, utilizado para gravar os principais tópicos publicados durante a execução do robô. Esse registro permitiu armazenar informações relacionadas ao movimento das juntas, posição do *end-effector*, coordenadas do objeto identificado e demais variáveis relevantes para a análise posterior.

O segundo código criado na mesma pasta possui o nome `rosbag_matlab.py` foi desenvolvido para extrair os dados armazenados no *rosbag* e convertê-los para arquivos com

extensão .m. Essa etapa foi importante porque permitiu organizar os dados coletados em formato adequado para processamento no *MATLAB*, facilitando a geração de gráficos, comparação de trajetórias e avaliação do desempenho do robô durante os testes.

Ao todo, foram realizados 30 testes com o robô. Durante os 20 primeiros testes, foram identificados alguns problemas que impediram a conclusão adequada da tarefa em determinadas execuções. O principal problema observado estava relacionado a falhas de trajetória, especialmente durante a movimentação da junta prismática. Essas falhas foram associadas ao comportamento do fuso, que não apresentava uma resposta adequada em algumas situações.

Para corrigir esse problema, foi realizado um ajuste no controlador da junta prismática, aumentando o ganho proporcional do fuso para 1500. Após essa modificação, a resposta da junta ficou mais adequada, permitindo que o robô não falhasse ao calcular a trajetória de subida ao terminar o movimento de pegar o objeto.

Além das falhas de trajetória, também foram observados problemas relacionados a bugs da própria simulação. Em algumas execuções, o comportamento físico do objeto ou da garra no ambiente do *Gazebo* não ocorreu da forma esperada, interferindo no resultado final da tarefa. Outro fator identificado foi a altura utilizada ao largar o objeto. Com os testes foi possível perceber que a altura da largada não estava ideal, fazendo com que o objeto não fosse depositado corretamente ou apresentasse instabilidade ao ser solto.

Com base nos problemas identificados nos primeiros testes, foram realizados ajustes nos parâmetros de controle, na altura de posicionamento do objeto e na rotina de movimentação. Esses ajustes tiveram como objetivo tornar a tarefa mais estável e reduzir falhas relacionadas tanto à trajetória do manipulador quanto à interação entre a garra e o objeto.

Após as correções realizadas, foram executados mais 10 testes com o robô. Nessa nova etapa, o movimento do manipulador ocorreu corretamente em quase todas as execuções, indicando que os ajustes aplicados ao controle da junta prismática e à rotina de movimentação foram suficientes para corrigir as falhas de trajetória observadas nos testes iniciais.

Entretanto, em 2 dos 10 testes ocorreu um bug relacionado à interação entre o objeto e a garra, no qual o objeto permaneceu travado durante a execução da tarefa. Apesar disso, o movimento do robô foi realizado corretamente, ou seja, o manipulador executou a trajetória planejada, alcançou as posições definidas e completou a sequência de movimentação. Dessa forma, essas falhas foram associadas ao comportamento da simulação de contato entre o objeto e

a garra, e não ao planejamento ou ao controle das juntas.

Nos demais testes, o robô conseguiu realizar a sequência completa de *pick and place*, demonstrando que a integração entre visão computacional, cinemática inversa, controle das juntas e planejamento de movimento foi realizada de forma funcional. A melhora observada entre os testes iniciais e finais evidencia a importância dos ajustes realizados durante a validação experimental, principalmente no ganho proporcional da junta prismática e na definição da altura adequada para depósito do objeto.

De forma geral, os testes mostraram que o sistema desenvolvido foi capaz de executar a tarefa proposta em ambiente simulado. As falhas iniciais permitiram identificar limitações na sintonia do controlador e na configuração da rotina de manipulação, enquanto os testes finais indicaram maior estabilidade do movimento e melhor desempenho do robô durante a execução da tarefa.

Para avaliar o desempenho do sistema de controle durante a execução da tarefa de pegar e posicionar o objeto, foram registrados os valores desejados e reais das juntas do robô, bem como a posição cartesiana desejada e real do *end-effector*. Os dados foram coletados durante toda a operação por meio de arquivos ROSBag e posteriormente convertidos para o formato CSV, permitindo sua análise no MATLAB.

Inicialmente, foram analisadas as respostas das três principais juntas responsáveis pelo posicionamento do robô SCARA. As juntas 1 e 2 são rotacionais e responsáveis pelo posicionamento do robô no plano horizontal, enquanto a junta 3 é prismática e controla o deslocamento vertical do *end-effector*.

5.2.1 Gráficos dos resultados obtidos nos testes

A Figura 5.4 apresenta a comparação entre as posições desejadas e reais das três juntas durante a execução completa da tarefa de pick and place.

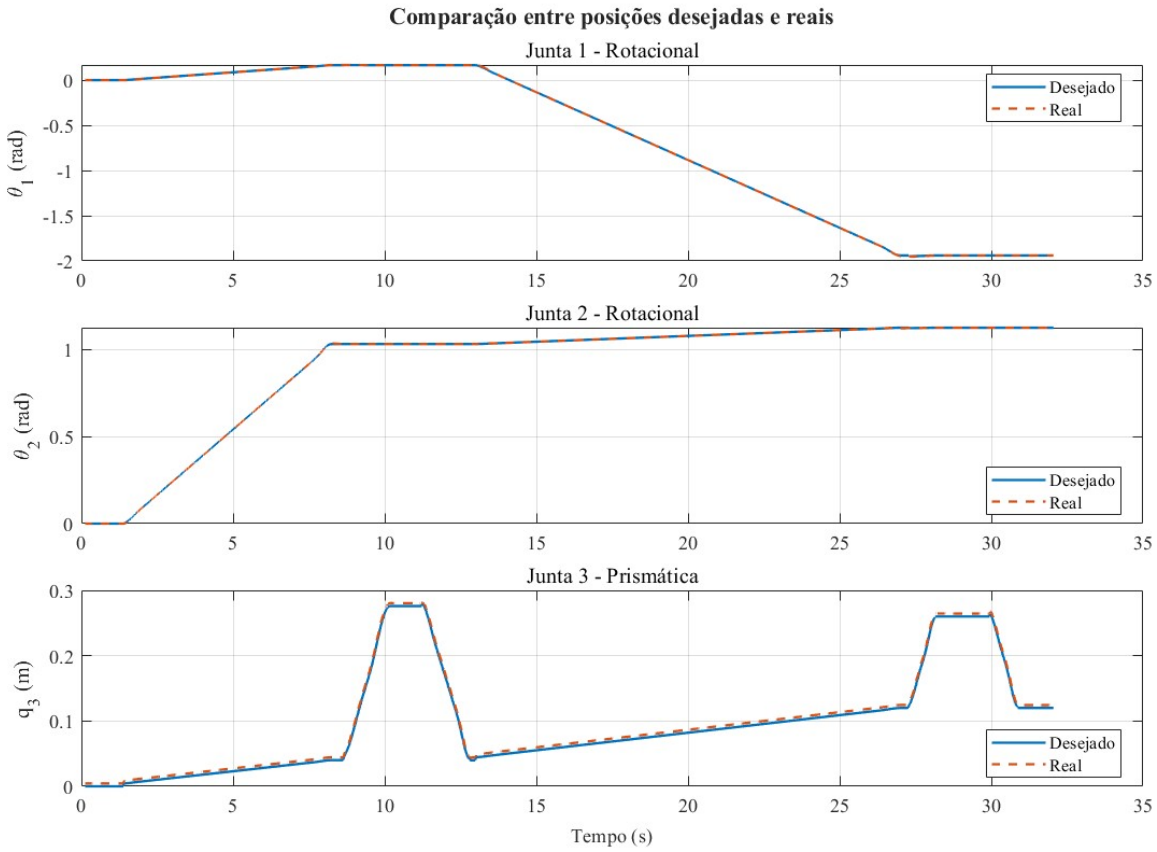


Figura 5.4 – Comparação entre as posições desejadas e reais das juntas do robô SCARA.

É possível notar que as resposta reais das juntas rotacionais executaram o movimento de maneira satisfatória dadas as referências produzidas pelo planejador de movimento. Nas juntas 1 e 2, as curvas desejada e real permanecem juntas durante a execução do movimento, apresentando pequenos erros principalmente na mudança de posição.

Essas diferenças ocorrem durante as transições entre as etapas da tarefa, como aproximação do objeto, descida da garra, transporte e posicionamento final. Nesses instantes, o controlador precisa acompanhar variações mais rápidas da referência, o que pode provocar pequenos atrasos e picos transitórios.

A junta 3 que corresponde a junta prismática, também acompanhou o movimento de referência, entretanto, foi observado uma diferença maior entre o valor desejado e o real. Diferente das juntas rotacionais, a resposta da junta prismática apresentou um erro residual que permaneceu mesmo após a finalização dos movimentos.

O comportamento observado indica que o controlador utilizado para a junta prismática apesar de produzir o movimento necessário para executar a tarefa, não eliminou completamente

o erro em regime permanente. Essa diferença pode ser associada a alguns fatores como à modelagem da junta, às características do sistema de transmissão, à ação gravitacional ou aos parâmetros de controle associados a junta.

A Figura 5.5 apresenta os erros de rastreamento calculados para cada junta. O erro foi determinado pela diferença entre o valor desejado e o valor real:

$$e_i(t) = q_{i,d}(t) - q_i(t), \tag{5.1}$$

em que $q_{i,d}(t)$ representa a posição desejada da junta i e $q_i(t)$ representa a posição real.

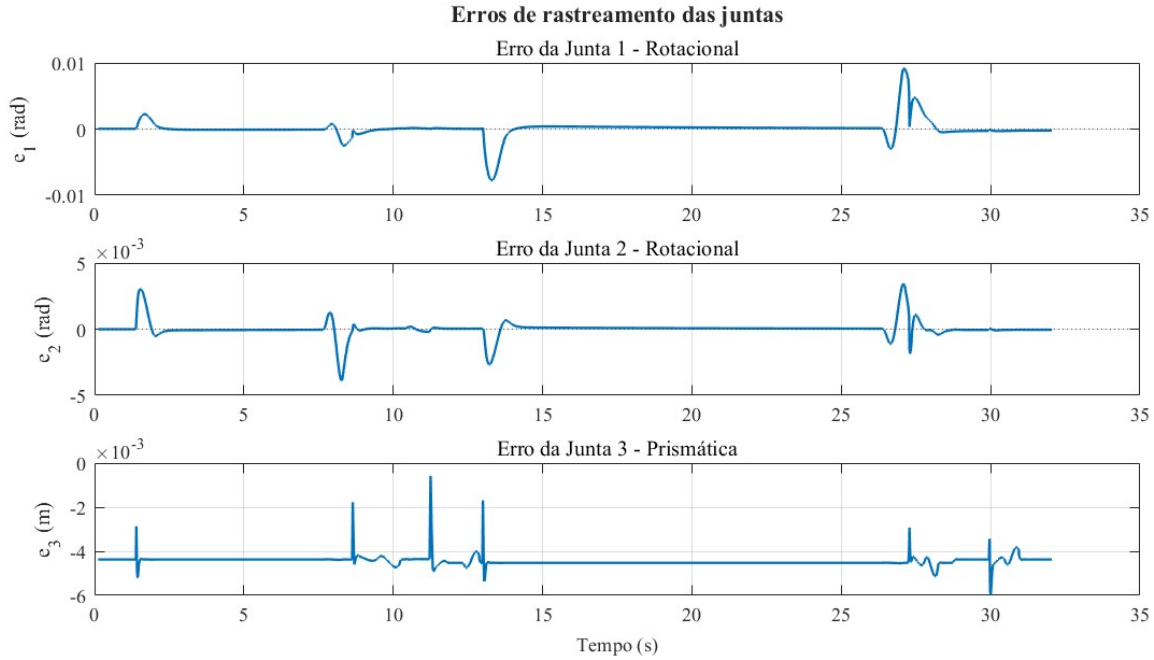


Figura 5.5 – Erros de rastreamento das juntas do robô SCARA.

Para as juntas rotacionais, os maiores valores de erro ocorrem durante as mudanças de referência. Após os movimentos, os erros diminuíram e permaneceram próximos a zero. Isso demonstra que os controladores das juntas 1 e 2 foram capazes de estabilizar o sistema nas posições desejadas.

A junta 1 apresentou durante sua movimentação picos de erro superiores aos da junta 2. Esse comportamento pode ser explicado ao analisar que a junta 1 movimenta uma parcela maior da estrutura do robô, sendo assim tendo maior inercia que a junta 2. Apesar disso, o erro final permaneceu reduzido, indicando que a posição desejada foi alcançada com uma boa precisão.

Ao analisar a junta 2 é possível notar que ela apresenta os menores erros entre as juntas. Sua resposta permanece próxima a da referência durante todo o movimento, com baixo erro médio e erro final praticamente nulo.

Já na junta prismática é possível observar um comportamento diferente. Após a realização de sua movimentação o erro não retornou para o zero, permanecendo próximo de $4{,}37$ mm ao final do teste. Esse resultado confirma a presença de um erro em regime permanente atrelado a junta.

Embora esse erro não tenha impedido a realização da tarefa de *pick and place*, ele reduziu a precisão da posição vertical do *end-effector*. Portanto, os resultados indicam a necessidade de ajustes adicionais nos ganhos do controlador da junta prismática.

Além da análise no espaço das juntas, foi realizada a comparação entre as coordenadas cartesianas desejadas e reais do *end-effector*. A Figura 5.6 apresenta os resultados para as coordenadas X, Y e Z.

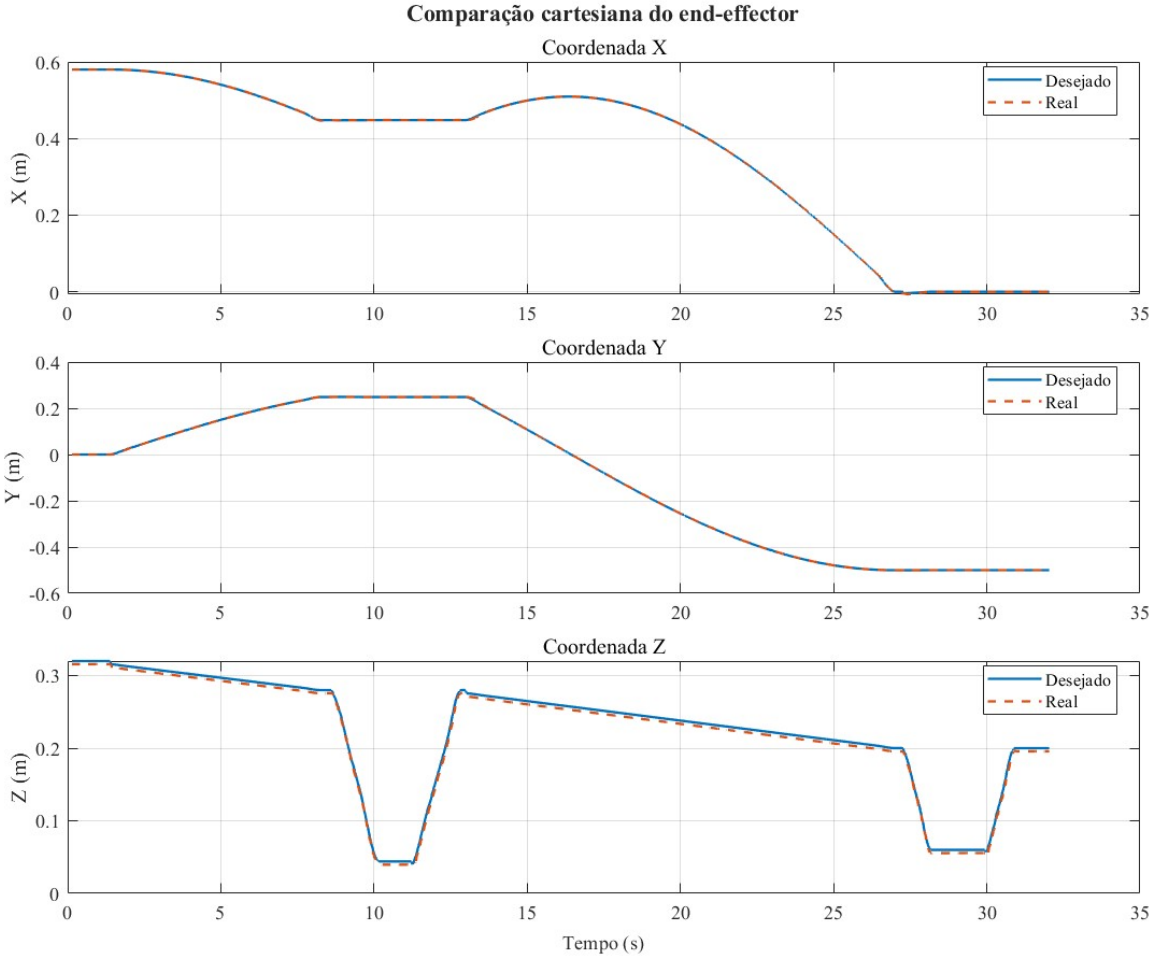


Figura 5.6 – Comparação entre as coordenadas cartesianas desejadas e reais do *end-effector*.

Como foi dito acima as juntas rotacionais, que são responsáveis pelo deslocamento em X e Y, obtiveram uma boa correspondência entre as trajetórias desejada e real, ocorrendo pequenos erros durante a troca de referências.

Já para a coordenada Z, onde a junta prismática é a responsável pela movimentação foi possível observar o movimento real não acompanhou propriamente o desejado isso devido ao erro de regime permanente que foi citado e que não conseguiu ser corrigido, com o gráfico é possível ver a relação entre o erro no espaço das juntas e o erro cartesiano do *end-effector*.

A Figura 5.7 apresenta a trajetória desejada e a trajetória real do *end-effector* no plano horizontal.

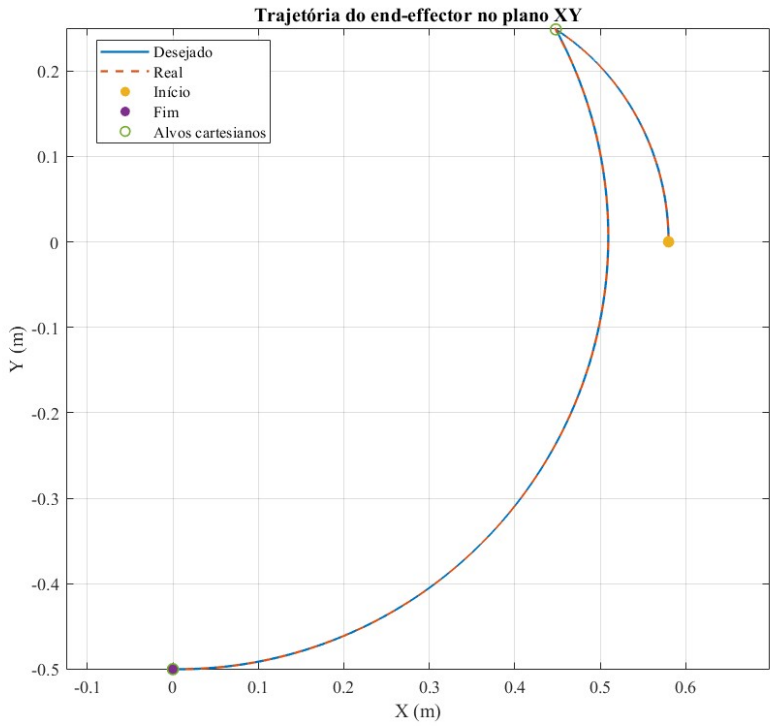


Figura 5.7 – Trajetória desejada e real do *end-effector* no plano XY.

Com o gráfico da trajetória é possível observar que a trajetória real e a desejada estiveram próximas durante todo o deslocamento do robô. Os pequenos desvios observados ocorreram durante as mudanças de direção e nas transições entre os alvos cartesianos

Isso indica que o sistema de controle foi capaz de conduzir o *end-effector* entre as posições desejadas para conclusão da tarefa. A proximidade entre as trajetórias demonstra que os pequenos erros notados nas juntas rotacionais não provocaram problemas no deslocamento horizontal do robô.

É importante observar que os comandos enviados ao robô correspondem as posições calculadas a partir dos alvos cartesianos. Entre os pontos gerados, o *MoveIt* gera uma trajetória que estima valores desconhecidos que estão dentro de um conjunto de dados conhecidos e discretos, para assim serem executados pelos controladores das juntas. Dessa forma, os pequenos erros de trajetória são a combinação do planejamento de movimento e da dinâmica dos controladores.

O erro cartesiano foi calculado a partir da distância euclidiana entre as posições desejada e real do *end-effector*:

$$e_{xyz}(t) = \sqrt{[x_d(t) - x(t)]^2 + [y_d(t) - y(t)]^2 + [z_d(t) - z(t)]^2}. \tag{5.2}$$

A Figura 5.8 apresenta a evolução desse erro durante o ensaio.

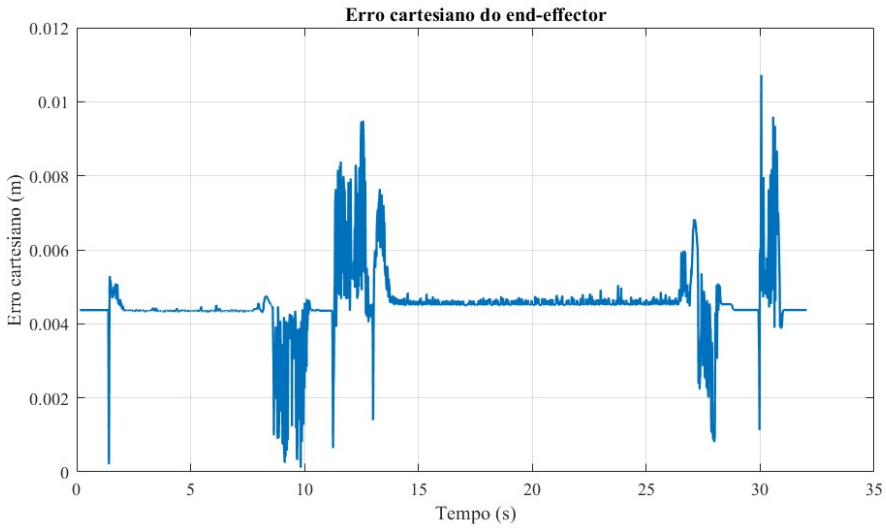


Figura 5.8 – Erro cartesiano do *end-effector* durante a execução da tarefa.

Ao observar o erro cartesiano, é obtido a mesma conclusão os maiores valores do erro cartesiano ocorrem durante as mudanças de posição, principalmente nos movimentos verticais, causando um erro final de $4{,}37$ mm, causado pelo erro da coordenada da junta prismática durante a aproximação e afastamento do objeto. Após os períodos transitórios, o erro diminuiu, mas não retornou completamente a zero.

Apesar de o robô ter executado corretamente a sequência de pegar, transportar e posicionar o objeto, a precisão cartesiana final foi limitada principalmente pelo desempenho da junta prismática.

A Figura 5.9 apresenta a trajetória registrada para o objeto durante a execução da tarefa.

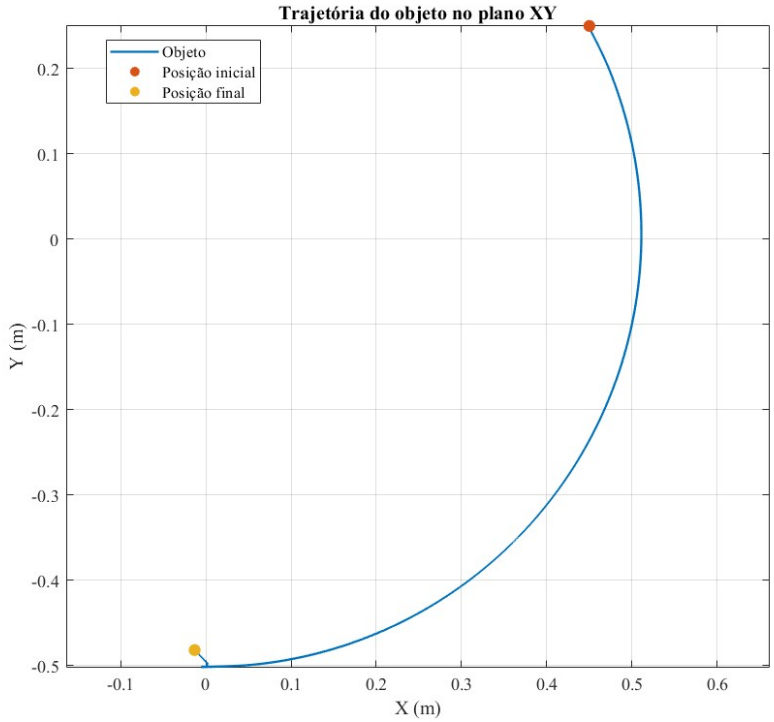


Figura 5.9 – Trajetória do objeto durante a operação de pick and place.

O gráfico da trajetória do movimento do objeto permite verificar que durante a execução do movimento do robô o objeto o acompanhou durante todo o percurso, indicando que o objeto foi anexado a garra

Após a deposição do objeto, é possível observar que o objeto está um pouco acima da posição final do *end-effector*, isso devido a altura de posicionamento do objeto, ao ser desanexado o objeto não ficou perfeitamente em pé, ele acabou caindo fazendo com que isso seja demonstrado no gráfico.

Esse resultado demonstra que o robô identificar o objeto, se mover até as suas coordenadas, anexar o objeto em suas garras, executar o movimento de deposição do objeto e deixa-lo na posição de depósito desejada.

Além da análise gráfica, também foram calculadas métricas que permitem avaliar desempenho dos controladores. A Tabela 5.1 apresenta o erro quadrático médio, o erro médio absoluto, o erro máximo e o erro final de cada junta.

Tabela 5.1 – Métricas de desempenho das juntas no teste analisado.

Junta	RMSE	MAE	Erro máximo	Erro final
θ_1	0,001366 rad	–	0,009203 rad	0,000213 rad
θ_2	0,000598 rad	–	0,003906 rad	0,000045 rad
q_3	0,004453 m	–	0,005971 m	0,004371 m

O RMSE representa o erro quadrático médio durante todo o teste, esse dado permite avaliar melhor os erros de maior amplitude, sendo útil para avaliar de forma geral o desempenho da movimentação do robô. Observando a tabela é possível perceber que a junta 2 apresentou o menor RMSE, indicando o melhor desempenho entre as juntas analisadas

O RMSE da junta 1 foi maior que o da junta 2, mas ainda foi baixo ao comparar à amplitude dos movimentos executados pela junta. O maior erro máximo da junta 1 ocorre durante as transições de referência, mas ele não permanece após a estabilização.

O RMSE da junta prismática foi de aproximadamente $4{,}45$ mm e erro final de aproximadamente $4{,}37$ mm. A proximidade entre esses dois valores indica que o erro ocorreu durante todo o teste, caracterizando assim um erro em regime permanente.

É importante lembrar que os valores das juntas rotacionais são apresentados em radianos, já que seus movimentos são rotacionais, enquanto os valores da junta prismática são apresentados em metros. Com isso em mente é necessário ter atenção, ao realizar uma comparação direta, pois representam grandezas físicas diferentes.

Para facilitar a interpretação, os valores da junta prismática podem ser convertidos para milímetros, conforme apresentado na Tabela 5.2.

Tabela 5.2 – Métricas de desempenho da junta prismática convertidas para milímetros.

Métrica	Valor
RMSE	4,453 mm
Erro máximo	5,971 mm
Erro final	4,371 mm

A Tabela 5.3 apresenta as métricas calculadas para a posição cartesiana do *end-effector*.

Tabela 5.3 – Métricas do erro cartesiano do *end-effector*.

Coordenada	RMSE	Erro máximo	Erro final
X	0,787 mm	5,098 mm	0,113 mm
Y	0,781 mm	5,424 mm	0,006 mm
Z	4,507 mm	9,593 mm	4,371 mm
XYZ	4,641 mm	9,594 mm	4,373 mm

Os resultados demonstram que as coordenadas X e Y apresentaram baixos valores de RMSE e erros finais próximos de zero. Esse comportamento confirma o bom desempenho das juntas rotacionais no posicionamento horizontal.

A coordenada Z apresentou o maior RMSE e o maior erro final entre as coordenadas cartesianas. Como o deslocamento vertical depende diretamente da junta prismática, esse resultado confirma que o erro cartesiano total foi predominantemente causado pelo controle da terceira junta.

O RMSE cartesiano tridimensional foi de aproximadamente $4{,}64$ mm, enquanto o erro cartesiano final foi de aproximadamente $4{,}37$ mm. Esses valores indicam que o robô alcançou a posição final com boa precisão no plano horizontal, mas manteve uma diferença vertical de poucos milímetros.

De modo geral, os resultados mostram que o sistema foi capaz de executar a tarefa de pick and place e acompanhar as trajetórias definidas pelo planejador. As juntas rotacionais apresentaram boa precisão e baixos erros finais. Entretanto, a junta prismática apresentou erro em regime permanente, sendo identificada como o principal ponto a ser aprimorado no sistema de controle.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou o desenvolvimento e a validação em ambiente simulado de um robô SCARA aplicado a uma tarefa de *pick and place*. Para isso, foram realizadas etapas de modelagem matemática, configuração física do manipulador, implementação dos controladores, integração com ferramentas de planejamento de movimento e desenvolvimento de um sistema de visão computacional para identificação da posição do objeto.

Inicialmente, foi realizada a modelagem do robô, considerando seus parâmetros físicos, geométricas e inerciais. Esses parâmetros foram essenciais para que o projeto se comportasse da maneira correta no ambiente simulado, assim permitindo representar o comportamento do manipulador de uma forma mais realista. Para que robô efetuasse o movimento de seu *end-effector* da maneira correta, foram desenvolvidas equações de cinemática direta e inversa para o projeto, essas foram utilizadas para relacionar as posições desejadas das juntas para a conclusão do movimento.

Também foi desenvolvido o projeto dos controladores PID das juntas do robô. Os ganhos foram calculados por alocação de polos e foi desenvolvido um *script* para obtenção dos mesmos, além do projeto dos controladores, foi realizado um controle empírico visto que durante os testes o modelo simplificado não representou completamente todos os efeitos da simulação. Para as juntas rotacionais θ_1 e θ_2, os controladores apresentaram bom desempenho, com baixos erros finais e acompanhamento satisfatório das referências, entretanto, para a junta prismática q_3, os ganhos teóricos deixaram a desejar já que não foram capazes de sustentar a posição da junta sendo necessários ajustes no ganho proporcional da junta.

Ao comparar os resultados dos ganhos calculados e empíricos, é possível notar a importância da união entre o teórico e o prático, apesar de ser um ambiente simulado ele ainda permite avaliar elementos que afetam o modelo de forma que apenas o modelo matemático não é capaz de determinar sozinho. Apesar do modelo matemático apresentar bons resultados, na prática ainda foram necessários ajustes para melhorar a estabilidade e desempenho do robô, isso demonstra que a simulação ainda pode apresentar variáveis que modificam o comportamento ideal do robô.

A integração do MoveIt permitiu determinar as possíveis colisões do modelo, organizar os grupos de planejamento do robô, definir a garra como o *end-effector* e configurar os controladores de cada grupo. Em conjunto com o *Gazebo*, foi possível testar os movimentos planejados em um

ambiente de simulação física. Além disso, foi utilizado o *OpenCV*, para identificação do objeto por meio de uma câmera, que ao identifica-lo da converte de sua posição para coordenadas de referência do mundo. Com esses fatores foi realizada a entre a visão computacional, cinemática inversa e planejamento de movimento.

Os testes realizados demonstraram que o sistema desenvolvido foi capaz de executar a tarefa proposta. Ao todo, foram realizados 30 testes com o robô. Nos 20 primeiros, foram observadas falhas relacionadas principalmente à trajetória da junta prismática, bugs da simulação e altura inadequada no momento de deposição do objeto. Após os ajustes no controlador do fuso e na rotina de movimentação, foram realizados mais 10 testes, nos quais o movimento do robô ocorreu corretamente. Em 2 desses testes, o objeto permaneceu travado na garra devido a problemas de interação na simulação, mas a trajetória do manipulador foi executada conforme planejado.

A análise dos dados coletados por meio do *ROSBag* mostrou que as juntas rotacionais foram capazes de executar os movimentos de forma aceitável, apresentando erros finais reduzidos. Entretanto, a junta prismática apresentou um erro em regime permanente de $4{,}37\,mm$, esse valor também influenciou no erro final cartesiano do efetuador. Apesar disso, o erro observado não impediu a realização da tarefa proposta, isso indica que o sistema atingiu um desempenho aceitável para a proposta.

Diante disso, é possível concluir que o objetivo principal do projeto foi alcançado, visto que foi possível desenvolver, simular e validar um robô SCARA capaz de identificar, alcançar e transportar um objeto em um ambiente virtual. O projeto demonstra a integração completa entre ROS, *Gazebo*, *MoveIt*, controle PID, cinemática inversa e visão computacional para aplicações de manipulação robótica. Além disso, os resultados obtidos permitiram identificar pontos fortes do sistema, como o bom desempenho no posicionamento horizontal e limitações a serem aprimoradas.

6.1 Trabalhos futuros

Existe uma infinidade de trabalhos futuros que possam ser sugeridos para o projeto desenvolvido, entretanto o que seria mais interessante seria o desenvolvimento físico do projeto simulado, isso permitiria avaliar o comportamento do manipulador diante de fatores reais, como atrito, folgas, limitações, ruídos e variações no ambiente. Também seria uma ótima oportunidade para melhorar os controladores, visto que em um ambiente real eles se comportariam de outras

formas, isso tornaria o trabalho ainda mais completo, podendo se tornar um sistema mais robusto e próximo de uma aplicação industrial analisado o real e o simulado.

REFERÊNCIAS

ANDHARE, P.; RAWAT, S. Pick and place industrial robot controller with computer vision. In: **2016 International Conference on Computing Communication Control and Automation (ICCUBEA)**. [S.l.]: IEEE, 2016. p. 1–4.

CALLEJA, R. G.; MARINERO, J. C. F.; MARTINEZ, R. D. O. Simulation of a scara robot. In: JAMSHIDI, M.; LUH, J. Y. S.; SHAHINPOOR, M. (Ed.). **Recent Trends in Robotics: Modelling, Control and Education**. Amsterdam: North-Holland, 1986.

CASHMORE, M. et al. Rosplan: Planning in the robot operating system. In: **Proceedings of the International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2015. p. 333–341.

CHAUDHARY, S.; ZAKHAMI, S.; ROY, S. D. Visual feedback based trajectory planning to pick an object and manipulation using deep learning. In: **Proceedings of the 2019 4th International Conference on Advances in Robotics**. [S.l.: s.n.], 2019. p. 1–6.

CHITTA, S.; SUCAN, I.; COUSINS, S. Moveit! [ros topics]. **IEEE Robotics & Automation Magazine**, v. 19, n. 1, p. 18–19, 2012.

CRAIG, J. J. **Introduction to Robotics: Mechanics and Control**. 3. ed. Upper Saddle River: Pearson Prentice Hall, 2005.

DAS, M. T.; DÜLGER, L. C. Mathematical modelling, simulation and experimental verification of a scara robot. **Simulation Modelling Practice and Theory**, v. 13, n. 3, p. 257–271, 2005.

DUTRA, R. **Arquivos do projeto**. 2026. Acessado em: 2 maio 2026. Disponível em: <https://drive.google.com/file/d/1xA3UQKRvGbkCzy_XCbRYOttzKz6-xekH/view?usp=sharing>.

HIBBELER, R. C. **Mecanica para Engenharia: Dinâmica**. 14. ed. São Paulo: Pearson, 2018.

HUANG, P.-C.; MOK, A. K. A case study of cyber-physical system design: Autonomous pick-and-place robot. In: **2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)**. [S.l.]: IEEE, 2018. p. 22–31.

IBRAHIM, M. Y.; COOK, C.; TIEU, K. Dynamic behavior of a scara robot with links subjected to different velocity and trajectories. **Robotica: International Journal of Information, Education and Research in Robotics and Artificial Intelligence**, 1988. Dados de volume, número e páginas não informados na referência original.

KRUT, S. et al. I4: A new parallel mechanism for scara motions. In: **2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)**. [S.l.]: IEEE, 2003. p. 1875–1880.

LEE, M. K.; LEE, C. N.; LIM, K. Y. Development of a fully integrated simulation package for industrial robot. In: **Proceedings of KACC**. Seoul: [s.n.], 1988.

MAKINO, H. Development of the scara. **Journal of Robotics and Mechatronics**, v. 26, n. 1, p. 5–8, 2014.

MENA, F. G. G. **Diseño e implementación de sistema de control para robot Open Source tipo SCARA**. Tese (Doutorado), 2020.

OGATA, K. **Modern Control Engineering**. 5. ed. Upper Saddle River: Prentice Hall, 2010. ISBN 978-0-13-615673-4.

PADHY, S. K. On the dynamics of scara robot. **Robotics and Autonomous Systems**, v. 10, n. 1, p. 71–78, 1992. ISSN 0921-8890. Disponível em: <https://www.sciencedirect.com/science/article/pii/092188909290016R>.

QIAN, W. et al. Manipulation task simulation using ros and gazebo. In: **2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)**. [S.l.]: IEEE, 2014. p. 2594–2598.

RIVERA, Z. B.; SIMONE, M. C. D.; GUIDA, D. Unmanned ground vehicle modelling in gazebo/ros-based environments. **Machines**, v. 7, n. 2, p. 42, 2019.

SHARIATEE, M. et al. Design of an economical scara robot for industrial applications. In: **2014 Second RSI/ISM International Conference on Robotics and Mechatronics (ICRoM)**. Tehran, Iran: [s.n.], 2014.

SICILIANO, B. et al. **Robotics: Modelling, Planning and Control**. London: Springer, 2009.

SPONG, M. W.; HUTCHINSON, S.; VIDYASAGAR, M. **Robot Modeling and Control**. Hoboken: John Wiley & Sons, 2006.

TAY, S. H.; CHOONG, W. H.; YOONG, H. P. A review of scara robot control system. In: **2022 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)**. Kota Kinabalu, Malaysia: [s.n.], 2022.

VISIOLI, A.; LEGNANI, G. On the trajectory tracking control of industrial scara robot manipulators. **IEEE Transactions on Industrial Electronics**, v. 49, n. 1, p. 224–232, 2002.

WIKI, R. **ROS Noetic Ninjemys Installation on Ubuntu**. 2020. Acesso em: 01 Mai. 2026. Disponível em: <https://wiki.ros.org/noetic/Installation/Ubuntu>.

ANEXO A – COMANDOS UTILIZADOS PARA CRIAÇÃO DO AMBIENTE DE TRABALHO

Neste anexo são apresentados os comandos utilizados para a criação do ambiente de trabalho do projeto, configuração do *workspace Catkin*, criação do pacote principal e definição da estrutura inicial de pastas utilizada na modelagem do robô SCARA.

```
mkdir Projeto_SCARA
cd Projeto_SCARA

mkdir src
catkin_make

cd src
catkin_create_pkg modelo_scara geometry_msgs
joint_state_publisher_gui roscpp rviz tf2 urdf
controller_manager joint_state_controller
robot_state_publisher std_msgs rospy

cd modelo_scara
mkdir urdf
```

ANEXO B – COMANDOS E CONFIGURAÇÕES UTILIZADOS NA VISUALIZAÇÃO DO ROBÔ NO RVIZ

Neste anexo são apresentados os comandos e trechos de configuração utilizados para inicializar o ambiente de trabalho, executar o arquivo *launch* responsável pela visualização do robô e carregar automaticamente as configurações salvas do RViz.

Para que o ROS reconheça o espaço de trabalho criado, foi utilizado o seguinte comando:

```
. ~/Projeto_SCARA/devel/setup.bash
```

Após a configuração do ambiente, o arquivo `rviz.launch` foi executado por meio do comando:

```
roslaunch modelo_scara rviz.launch
```

Para que o RViz fosse aberto automaticamente com as configurações salvas na pasta `config`, foi adicionada ao arquivo `rviz.launch` a seguinte chamada:

```
<node name="rviz" pkg="rviz" type="rviz"
args="-d $(find modelo_scara)/config/config.rviz"/>
```

ANEXO C – COMANDOS UTILIZADOS NA SIMULAÇÃO E MANIPULAÇÃO NO GAZEBO

Neste anexo são apresentados os comandos utilizados para inicializar o modelo do robô SCARA no ambiente de simulação do Gazebo e para instalar os pacotes responsáveis por melhorar o comportamento de preensão da garra durante a manipulação de objetos.

C.1 Inicialização do modelo no Gazebo

O comando a seguir foi utilizado para executar o arquivo `gazebo.launch`, responsável por carregar o modelo do robô, os controladores das juntas e as ferramentas auxiliares no ambiente de simulação.

```
roslaunch modelo_scara gazebo.launch
```

C.2 Instalação dos plugins de manipulação

Os comandos a seguir foram utilizados para instalar os pacotes relacionados ao comportamento de preensão da garra no ambiente de simulação do Gazebo.

```
cd ~/Projeto_SCARA/src
git clone https://github.com/JenniferBuehler/gazebo-pkgs
git clone https://github.com/roboticsgroup/
roboticsgroup_gazebo_plugins
cd ~/Projeto_SCARA
catkin_make
```

ANEXO D – CÓDIGOS UTILIZADOS NA CONFIGURAÇÃO DO MOVEIT

Neste anexo são apresentados os comandos e códigos utilizados para a configuração do *MoveIt* e para a movimentação do robô SCARA no ambiente de simulação. Os códigos foram organizados separadamente para facilitar a consulta e evitar a inserção de comandos diretamente no corpo do texto principal.

D.1 Instalação dos pacotes do MoveIt

O comando a seguir foi utilizado para instalar os pacotes necessários ao funcionamento do *MoveIt* no ROS Noetic.

```
sudo apt install ros-noetic-moveit
ros-noetic-moveit-ros-planning
ros-noetic-moveit-ros-visualization
ros-noetic-moveit-visual-tools
```

D.2 Inicialização do assistente de configuração do MoveIt

Após a instalação dos pacotes, o assistente de configuração do *MoveIt* foi inicializado por meio do comando apresentado a seguir.

```
roslaunch setup_moveit_assistent setup_assistant.launch
```

D.3 Inicialização da demonstração no *RViz*

```
roslaunch moveit_config demo.launch
```

ANEXO E – COMANDOS UTILIZADOS NA CRIAÇÃO DO PACOTE DE CODIFICAÇÃO E VISÃO

Neste anexo são apresentados os comandos utilizados para criação do pacote de codificação e visão computacional, bem como para execução do *script* inicial de *pick and place*.

E.1 Criação e compilação do pacote

```
cd ~/Projeto_SCARA/src
catkin_create_pkg scara_move cv_bridge geometry_msgs
image_transport moveit_commander rospy sensor_msgs
std_msgs tf message_generation
cd ~/Projeto_SCARA
catkin_make
```

E.2 Permissão de execução do *script*

```
cd ~/Projeto_SCARA/src/scara_move/src
chmod +x pegarlargar.py
```

E.3 Execução do *script* de *pick and place*

```
source ~/Projeto_SCARA/devel/setup.bash
rosrun scara_move pegarlargar.py
```

E.4 Criação e compilação do pacote de rastreamento de posição

```
cd ~/Projeto_SCARA/src
catkin_create_pkg position_tracker roscpp rospy std_msgs message_genera
cd ~/Projeto_SCARA
catkin_make
```