Universidade de São Paulo
Escola de Engenharia de São Carlos
Departamento de Engenharia Aeronáutica

LUISA MARTINS LEMOS DE SOUZA

# Geração de um ambiente virtual com Gazebo utilizando o robô KUKA LBR iiwa 14 R820™ e sensores de câmera pelo sistema ROS para simulação de etapas de manufatura na indústria aeronáutica

São Carlos
2023

LUISA MARTINS LEMOS DE SOUZA

# Geração de um ambiente virtual com Gazebo utilizando o robô KUKA LBR iiwa 14 R820$^{TM}$ e sensores de câmera pelo sistema ROS para simulação de etapas de manufatura na indústria aeronáutica

Monografia apresentada ao Curso de Engenharia Aeronáutica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Aeronáutico.

Orientador: Prof. Dr. Glauco Augusto de Paula Caurin

São Carlos

2023

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Martins Lemos de Souza, Luisa
M379g Geração de um ambiente virtual com Gazebo utilizando o robô KUKA LBR iiwa 14 R820TM e sensores de câmera pelo sistema ROS para simulação de etapas de manufatura na indústria aeronáutica / Luisa Martins Lemos de Souza; orientador Glauco Augusto de Paula Caurin. São Carlos, 2023.

Monografia (Graduação em Engenharia Aeronáutica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2023.

1. Simulações robóticas. 2. robôs colaborativos. 3. ROS. 4. sensoriamento por câmeras. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

# FOLHA DE APROVAÇÃO
## *Approval sheet*

| |
|---|
| **Candidato** / *Student*: Luisa Martins Lemos de Souza |
| **Título do TCC** / *Title* **:** Geração de um ambiente virtual com Gazebo utilizando o robô KUKA LBR iiwa 14 R820TM e sensores de câmera pelo sistema ROS para simulação de etapas de manufatura na indústria aeronáutica |
| **Data de defesa** / *Date*: 08/08/2023 |

| **Comissão Julgadora** / *Examining committee* | **Resultado** / *result* |
|---|---|
| Prof. Dr. João Paulo Eguea | APROVADO |
| Instituição / Affiliation: EESC - SAA | |
| Mestre José Otávio Savazzi | APROVADO |
| Instituição / Affiliation: Embraer | |

Presidente da Banca / Chair of the Examining Committee:

Prof. Dr. João Paulo Eguea
(assinatura / *signature*)

# AGRADECIMENTOS

Agradeço a todos as pessoas que participaram diretamente ou indiretamente da minha jornada ao longo da graduação.

Deixo o meu agradecimento especial a meus avós Cleide, Albano, Margarida e, em particular, em memória de meu avô Guido, a quem dedico esse trabalho.

A meus pais, por acreditarem no meu potencial e por me apoiarem todos os dias.

Por fim, a minha família e amigos por dividir a vida em seus momentos bons e ruins.

Obrigada por tudo.

# RESUMO

Simulações robóticas são ferramentas importantes para prever o comportamento de robôs em ambientes virtuais fidedignos e em situações não viáveis por complexidade ou custo. O uso de robôs colaborativos, como o KUKA LBR iiwa 14 R280$^{TM}$, tem impulsionado transformações nos processos de manufatura de componentes aeroespaciais, assim como o aumento de produtividade e redução de custos na indústria aeronáutica. *Software* e sistemas, como ROS e Gazebo, têm sido instrumentos importantes de prototipação ágil, fácil implementação e reprodução. Nesse contexto, esse trabalho tem por objetivo a utilização do Gazebo integrado ao ROS para criação de ambientes virtuais simulando processos de manufatura na indústria aeronáutica por meio do LBR iiwa. O trabalho parte da implementação da reutilização do código da biblioteca *"iiwa_stack"* para reproduzir a geometria, restrições cinemáticas e funcionalidades de controle do robô. Cria-se, então, um ambiente virtual com Gazebo para ser um exemplo de aplicação no formato de um pacote. A partir disso, utiliza a biblioteca do MoveIt para implementar o planejamento de trajetória acoplado ao Gazebo e ao KUKA Sunrise, permitindo a utilização em ambientes virtuais ou situações reais. Em seguida, é construído por meio da biblioteca *"gazebo_ros_link_attacher"* a funcionalidade de troca de ferramenta em tempo de execução. Por fim, é implementado o sensoriamento por câmeras visando a obtenção de imagens observadas pelo robô e externas, novamente com suporte a simulação e testes em ambientes físicos. O resultado das imagens adquiridas são disponibilizadas para aplicação em novos arquivos.

**Palavras-chave:** Simulações robóticas, robôs colaborativos, ROS e sensoriamento por câmeras.

# ABSTRACT

*Robotic simulations are important tools to predict robot behavior in realistic virtual environments and in situations not feasible due to complexity or cost. The use of collaborative robots, such as the KUKA LBR iiwa 14 R280$^{TM}$, has driven transformations in the manufacturing processes of aerospace components, as well as increased productivity and reduced costs in the aeronautical industry. Software and systems, such as ROS and Gazebo, have been important instruments for agile prototyping, easy implementation and reproduction. In this context, this work aims to use Gazebo integrated with ROS to create virtual environments simulating manufacturing processes in the aeronautical industry through LBR iiwa. The work starts from the implementation of the reuse of the "iiwa_stack" library code to reproduce the geometry, kinematic constraints and control functionalities of the robot. It then creates a virtual environment with Gazebo to be an example application in a package format. From there, it uses the MoveIt library to implement trajectory planning coupled with Gazebo and KUKA Sunrise, allowing use in virtual environments or real situations. Then, the tool change functionality at runtime is built through the "gazebo_ros_link_attacher" library. Finally, camera sensing is implemented to obtain images observed by the robot and external images, again with support for simulation and testing in physical environments. The result of the acquired images are made available for application in new files.*



**Keywords:** *Robotic simulations, collaborative robots, ROS and camera sensing.*

# LISTA DE FIGURAS

# LISTA DE ABREVIATURAS E SIGLAS

**LBR** *Leichtbauroboter*

**IIWA** *Intelligent Industrial Work Assistant*

**ROS** *Robot Operating System*

**CLP** *Controladores Lógicos Programáveis*

**JVM** *Java Virtual Machine*

**ODE** *Open Dynamics Engine*

**XML** *Extensible Markup Language*

**URDF** *Unified Robot Description*

**XACRO** *XML Macro*

**SDF** *Simulation Description Format*

**GdL** *Graus de Liberdade*

**GUI** *Graphical User Interface*

**TCP/IP** *Transmission Control Protocol/Internet Protocol*

**PID** *Proporcional-Integrativo-Derivativo*

**CAD** *Computer Aided Design*

**STL** *Standard Tessellation Language*

**VANT** *Veículo Aéreo Não Tripulado*

**V-REP** *Virtual Robot Experimentation Platform*

# CONTEÚDO

CAPÍTULO 1

# INTRODUÇÃO, JUSTIFICATIVA E OBJETIVOS

## 1.1 Motivação

A indústria aeronáutica tem se beneficiado historicamente das aplicações da robótica em processos de fabricação de peças, como corte, soldagem, pintura e montagem (Ross and Brabazon, 2009). Entretanto, devido a complexidade e especificidade, processos de menor escala continuam sendo processados de forma bastante manual.

Com o surgimento da indústria 4.0 e a constante busca pela otimização de processos, sinergias nas cadeias produtivas e de suprimentos, capacidade de customização e pela maior inteligência operacional (treinamentos), a utilização de gêmeos digitais tem trazido aumento de eficiência em processos bastante estabelecidos (Lu et al., 2017; Tao et al., 2019).

Nesse contexto, a utilização de robôs colaborativos em operações de montagem de componentes aeroespaciais tem se apresentado como uma solução eficiente, resultando em redução de custos e aumento da produtividade (Fassi et al., 2018). Em especial, a utilização do LBR iiwa na montagem de componentes aeroespaciais tem sido objeto de interesse crescente, pois a integração de robôs colaborativos nesse contexto permite a automação de tarefas complexas e repetitivas, contribuindo para o aumento da produtividade e qualidade dos produtos finais (Xu et al., 2019).

Bezzo et al. (2019) destacam que a utilização de robôs autônomos e inteligentes está emergindo como uma tendência promissora para otimizar processos logísticos em fábricas aeroespaciais, reduzindo tempo e custos.

Além disso, a robótica tem desempenhado um papel essencial na manutenção e reparo de aeronaves. A utilização de robôs em atividades de inspeção e reparo de superfícies aeroespaciais tem sido explorada para melhorar a segurança das operações, minimizar o tempo de inatividade das aeronaves e realizar verificações de qualidade (Fernandez et al., 2017; Sun et al., 2021). Novamente, o uso do LBR iiwa em operações de inspeção e manutenção de aeronaves tem se destacado pela precisão e a sensibilidade do robô para tarefas de detecção de defeitos e falhas em superfícies aeroespaciais (Vargas et al., 2020). Além disso, a capacidade do LBR iiwa de executar tarefas de forma repetitiva e consistente torna-o ideal para inspeções em ambientes de difícil acesso.

Nesse cenário, esse trabalho exemplifica a construção de uma ferramenta estruturada que permita a geração de ambientes virtuais customizáveis, visando viabilizar a utilização do LBR iiwa em novas oportunidades de aplicação voltadas a indústria aeronáutica.

## 1.2 Objetivo

Este trabalho tem como objetivo utilizar o ambiente de simulação Gazebo (um *software* dedicado a prototipagem rápida de simulações com robôs) para criar ambientes virtuais que simulem aplicações genéricas de processos de manufatura na indústria aeronáutica. Isso será realizado por meio da integração do robô KUKA LBR iiwa 14 R280$^{TM}$ com o sistema ROS (Robot Operating System). Especificamente, o foco do trabalho é estabelecer uma base para a utilização de sensoriamento por câmeras nessas aplicações.

Para alcançar os objetivos propostos, o trabalho foi dividido em diversas tarefas:

1. Definir o ambiente de aplicação que será simulado
2. Identificar e construir os arquivos de malha dos elementos da simulação
3. Desenvolver os arquivos SDF (*Simulation Description Format*) específicos para a aplicação em questão
4. Criar arquivos de extensão para adaptar o pacote *"iiwa_stack"* para a nova aplicação

5. Criar arquivos de extensão para adaptar o pacote MoveIt para a nova aplicação
6. Integrar sensores e *plugins* para câmeras ao sistema
7. Escrever *script* para extração local das imagens das câmeras
8. Disponibilizar todos os recursos e códigos desenvolvidos para a aplicação no GitHub

## 1.3 Estrutura do documento

A estrutura do presente trabalho é subdividida em cinco capítulos. O capítulo 2 apresenta a fundamentação teórica, com as referências de livros, trabalhos utilizados e o o conteúdo que foi necessário para concepção desse trabalho.

O capítulo 3 desenvolve a metodologia utilizada, a partir do planejamento da arquitetura do sistema e dos recursos utilizados para a criação de um ambiente virtual para a simulação do KUKA LBR iiwa, como o Gazebo. Segue-se com uma explicação dos recursos e estrutura de dados disponibilizados pelo ROS (Open Robotics, 2023b) e MoveIt para a simulação de funcionalidades e planejamento da trajetória do robô.

Por fim, é apresentado no capítulo 4 algumas discussões acerca de resultados, detalhes do cenário de manufatura aeronáutica que foi criado e as possibilidades de simulação que a modelagem do LBR iiwa permite no contexto da indústria aeronáutica. No último capítulo apresentam-se as conclusões e perspectivas de desenvolvimentos futuros.

CAPÍTULO 2

# FUNDAMENTAÇÃO TEÓRICA

## 2.1 Robótica

A robótica é uma área multidisciplinar da ciência e engenharia que se concentra no projeto, construção, programação e operação de robôs. Estes são dispositivos mecânicos ou sistemas automatizados capazes de realizar tarefas de forma autônoma ou controlada por humanos (Craig, 2005). A robótica abrange diversos campos, como mecânica, eletrônica, computação, inteligência artificial e controle, visando criar máquinas com habilidades que assemelham-se ou superam as capacidades humanas em determinadas atividades (Siciliano and Khatib, 2016). E, sobretudo, a robótica possui aplicação em diversas áreas e a indústria aeronáutica é uma das áreas que tem experimentado o impacto dos avanços da robótica.

### 2.1.1 Robótica aplicada na indústria aeronáutica

A robótica tem desempenhado um papel transformador na indústria aeronáutica, revolucionando processos de fabricação, manutenção e operação de aeronaves. Desde suas origens, a robótica tem sido uma área em constante evolução, impulsionada por avanços tecnológicos e inovações que têm impactado significativamente a indústria aeroespacial. Essas máquinas autônomas e inteligentes têm demonstrado um potencial notável para melhorar a eficiência, a segurança e a qualidade dos sistemas aeronáuticos.

Na indústria aeronáutica, a robótica tem sido amplamente adotada em diferentes etapas do processo de fabricação de aeronaves. Como já citado, a robótica tem se mostrado uma ferramenta valiosa para a execução de tarefas repetitivas e de alta precisão na linha de montagem (e.g. furação e inserção de pinos), como a aplicação de adesivos e rebitagem das peças com aumentos visíveis de produtividade e qualidade do produto (Santos et al., 2021). Nesse cenário, o aumento recente da capacidade computacional e o advento de novas tecnologias têm viabilizado a utilização de métodos modernos de controle em aplicações complexas. Em seu trabalho, (Lahr, 2017) demonstra a necessidade de implementação de controladores de complacência e impedância para realização de tarefas de contato (não-linearidade) como de inserção de colares roscados em pinos.

Além da fabricação, a robótica também desempenha um papel fundamental na manutenção e inspeção de aeronaves. Robôs móveis e *drones* têm sido empregados para inspecionar áreas de difícil acesso nas aeronaves, como asas e motores, permitindo a detecção precoce (manutenção preventiva) de falhas e a realização de reparos de forma mais eficiente (Chacón et al., 2020). Enquanto nos aeroportos, robôs autônomos têm sido desenvolvidos para realizar atividades como o carregamento e descarregamento de bagagens, bem como a limpeza e inspeção das pistas nos aeroportos (Garcia et al., 2019).

A robótica possui uma relação intima com a história do desenvolvimento dos VANTs. Os VANTs têm sido utilizadas para uma variedade de aplicações, desde a coleta de dados, entrega de mercadorias, atividades de pulverização e segurança. É esperado que os VANTs revolucionem a maneira áreas da indústria aeronáutica lidam com a logística, potencializando um aumento de sustentabilidade dos processos (Vose and De Silva, 2020).

Dessa forma, a interseção entre a robótica e a indústria aeronáutica apresenta um cenário promissor de avanços tecnológicos e inovações.

## 2.2 Simulação robótica

A simulação robótica é uma importante ferramenta utilizada para reproduzir o comportamento de robôs em um ambiente virtual. Por meio da simulação, é possível criar cenários controlados de teste para validar algoritmos de controle, planejamento de movimento e interações com o ambiente antes da implementação física (Cacace et al., 2017). Utilizar uma metodologia de

simulação pode proporcionar redução de custos e redução de riscos associados a testes em ambientes reais, além de possibilitar a rápida iteração no desenvolvimento de soluções robóticas mais eficientes e seguras (Ribeiro et al., 2019).

Uma das principais vantagens da simulação robótica é que viabiliza e acelera a realização de um grande número de experimentos. A simulação permite analisar o comportamento do robô em situações extremas, como colisões ou ambientes com condições climáticas adversas, o que é essencial para garantir a segurança e robustez das soluções desenvolvidas em condições difíceis ou associadas a riscos de perda do ativo (Haidu et al., 2018). Isso torna a simulação uma poderosa ferramenta para aprimorar o desempenho e a confiabilidade de robôs em aplicações críticas.

Além disso, as simulações envolvendo robótica tem ganhado aplicações em outras área. Pesquisas demonstram que a simulação é uma maneira eficiente de treinar e aperfeiçoar modelos de controle de robôs por meio da utilização de técnicas de aprendizado supervisionado e por reforço, permitindo que o robô adquira habilidades complexas sem a necessidade de muitas interações com o ambiente real (Taele et al., 2016). A simulação tem permitido que os engenheiros e pesquisadores testem diferentes estratégias de navegação e coleta de dados em missões de exploração, possibilitando o planejamento de estratégias de exploração de ambientes desconhecidos ou de baixo acesso como missões espaciais (Falco et al., 2019).

### 2.2.1 Métodos de simulação robótica

Diversas metodologias foram desenvolvidas e utilizadas para simulação robótica, cada um com suas características e aplicações específicas. Entretanto, na literatura destacam-se as metodologias de modelos de simulação física (cinemáticos e dinâmicos); e modelos de malha (Huang et al., 2020).

Na simulação baseada em física, as equações de movimento são aplicadas para modelar o comportamento dos robôs e do ambiente em que eles atuam. Dessa forma, a simulação leva em conta a dinâmica dos corpos e as forças envolvidas nas interações (e.g. gravidade e atrito). Essa abordagem se mostrou capaz de testar algoritmos de controle e movimento, avaliar cenários de colisões e outras interações entre o robô e o ambiente (Hwangbo et al., 2017; Kim et al., 2019).

Por fim, na metodologia de simulação baseada em malhas o ambiente e o robô são representados por modelos geométricos discretizados em malhas. Esse tipo de simulação é rápida

e eficiente, pois reduz a complexidade dos cálculos para verificar interação entre objetos. A simulação baseada em malhas é frequentemente utilizada em simulações de ambientes virtuais de larga escala (e.g. cidades e ambientes industriais) (Chentanez et al., 2012) ou devido às possibilidades de visualização avançadas, é utilizada em aplicações de realidade virtual e realidade aumentada (Bender et al., 2017).

A realidade virtual e a realidade aumentada têm se mostrado promissoras para aprimorar a usabilidade e a eficácia das simulações robóticas (McCall and O'Hare, 2017; Ortega et al., 2016). Além disso, a simulação robótica conta com o uso de ambientes de simulação como o Gazebo e o V-REP que possibilitam a integração com bibliotecas de controle e planejamento de movimento, como o ROS (*Robot Operating System*). Esse tipo de ambiente de simulação serve de *frameworks* para prototipagem e validação de sistemas robóticos (Ko et al., 2019).

### 2.2.2 Principais *software* e aplicações

Os *software* mais populares e ferramentas mais utilizadas para criação de simulações robóticas e que se apresentam como alternativas as ferramentas apresentadas nesse trabalho são: Gazebo; o V-REP; ROS; e Webots. Todas essas ferramentas apresentam diversas aplicações, incluindo aplicações voltadas a indústria aeronáutica

Peng et al. (2018) desenvolveram e testaram um algoritmo de navegação de enxame para múltiplos robôs móveis em um ambiente simulado com o Gazebo. Tardioli et al. (2019) utilizaram o Gazebo para simular um robô colaborativo usado na montagem de componentes de aeronaves.

O ROS é uma plataforma amplamente utilizada para o desenvolvimento e integração de sistemas robóticos e nos trabalhos de Sisbot et al. (2010) apresentam o uso do ROS para simular um robô assistente para ambientes domésticos. Analogamente, Wenzel et al. (2017) utilizou o ROS como plataforma para simular um sistema de controle de voo para VANTs em cenários complexos.

O V-REP é amplamente utilizado na indústria aeronáutica para simulação de manipuladores robóticos em linhas de montagem de aeronaves. Zhang et al. (2021) utilizaram o *software* para simular o processo de montagem de painéis de fuselagem em uma aeronave. A simulação permitiu otimizar o planejamento de trajetória do robô e identificar possíveis colisões ou interferências

antes da implementação física. Nessa lógica, com Webots, Corke et al. (2018) simularam o voo autônomo de um VANT em ambientes urbanos.

Essas pesquisas indicam a importância e eficácia do uso desses *software* para simulações robóticas, mas além disso, servem de suporte científico a realização desse trabalho.

CAPÍTULO 3

# METODOLOGIA

## 3.1 Arquitetura do sistema

O sistema foi desenhado em 5 módulos que desempenham as principais atividades do projeto enquanto são orquestrados pelo ROS:

1. Simulação utilizando Gazebo 9.0.0
2. Planejamento de trajetória utilizando MoveIt 1.0
3. Troca de ferramenta utilizando biblioteca de terceiros
4. Aquisição e processamento de imagens para *Machine Learning* utilizando Python 3.8
5. LBR iiwa utilizando KUKA Sunrise 1.11

Nessa arquitetura do sistema, conforme pode ser vista na figura 3.1, 4 módulos apresentam seu tempo de execução em uma máquina de suporte enquanto apenas o KUKA Sunrise compreende ao controlador e *software* de controle ao LBR iiwa. Para execução e organização das atividades o ROS deve estar instalado no computador de suporte.

Se por um lado cada módulo propriamente dito é apenas uma abstração da execução das atividades previstas, por outro, a nível de implementação em ROS, os módulos são descritos por unidades chamadas nós ROS, os quais executam funções específicas dentro do projeto.

O LBR iiwa (braço robótico) é composto por duas partes como pode ser visto na figura 3.2:

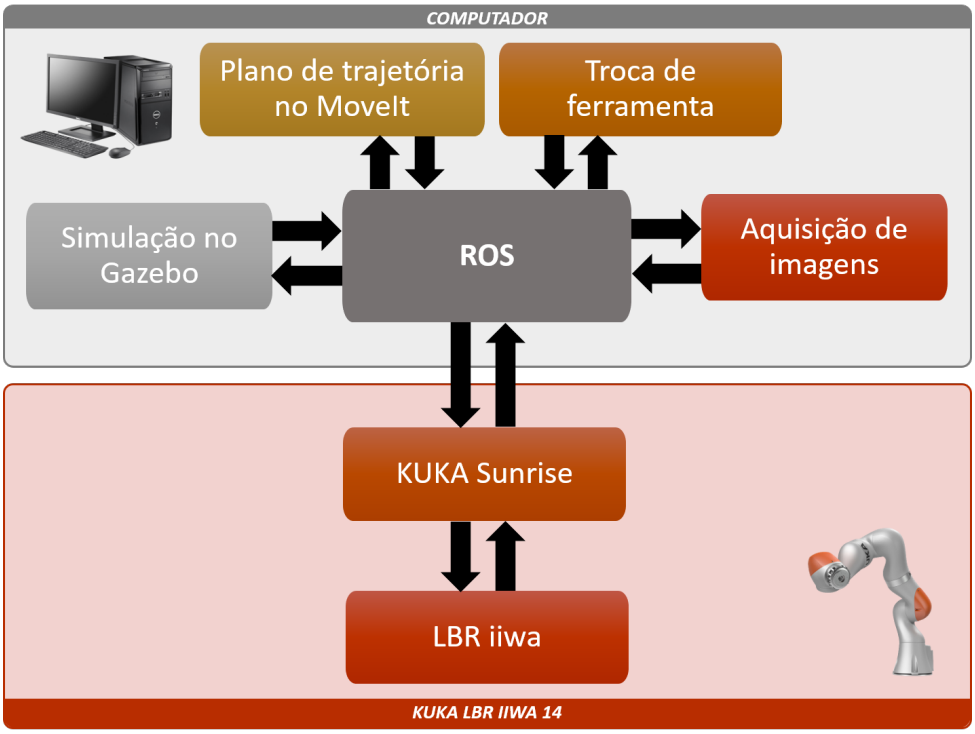



Figura 3.1: Arquitetura do sistema que engloba a operação do robô

- Estrutura mecânica que compões o braço mecânico (peças e atuadores)
- Controlador KUKA Sunrise (hardware de controle e interface, e *software* de controle)

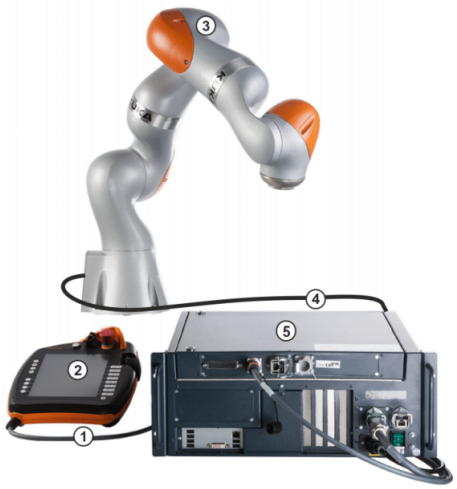



Figura 3.2: Sistemas do LBR iiwa: (1) Cabo de conexão com *SmartPad*; (2) Painel de controle do *SmartPad*; (3) LBR iiwa; (4) Cabo de conexão com KUKA Sunrise; e (5) Controlador do KUKA Sunrise (KUKA Robotics, 2023a).

O KUKA Sunrise executa todas as funções inerentes ao controle e movimentação do LBR iiwa e por meio dele se estabelece a interface de comunicação e controle externo (e.g. computador com cabo Ethernet, placa de controle, etc.).

Outro módulo é o de planejamento de trajetória que recebe informações da posição a ser atingida na simulação e, a partir da posição atual do manipulador, constrói uma trajetória possível para ser percorrida pelo braço robótico a fim de atingir o alvo.

Por sua vez, o módulo de troca de ferramenta simula a troca de uma ferramenta no ambiente da simulação em etapas análogas a realidade. O processo leva em conta a geometria e dimensões da ferramenta, tal como o posicionamento e no final anexa a ferramenta ao efetuador do robô na simulação.

O módulo de execução e visualização da simulação e de seu ambiente é executado em ambiente Gazebo, nele toda a simulação pode ser vista assim como todos os elementos que estão inclusos para tornar o ambiente mais realista. O objetivo da simulação é a previsão do posicionamento final do LBR iiwa após entradas de comandos de movimentação.

Por fim, pode-se citar o módulo de aquisição e processamento de imagens. Esse módulo se baseia na aquisição de imagens por sensores de câmera implementadas de forma adicional no robô LBR iiwa para captar a visualização das atividades. Essas imagens depois de captadas e processadas, podem ser utilizadas em algoritmos de *machine learning* para em etapas futuras melhorarem a performance das atividades exercidas, calibrações e identificações de problemas estruturais ou de montagem.

É importante destacar que os módulos da arquitetura do sistema atuam de forma independente, mas se comunicam de maneira coesa, levando em consideração as operações uns dos outros. Isso significa que as informações são compartilhadas entre os nós ROS para garantir que todo o sistema esteja ciente do estado, posicionamento ou da ferramenta acoplada. Por exemplo, é vantajoso ter as informações da ferramenta disponíveis tanto para o KUKA Sunrise quanto para a simulação no Gazebo. Para isso, existe um nó ROS dedicado à inserção, remoção, troca de ferramentas e comunicação dessas etapas, facilitando a interação e coordenação entre os diferentes componentes do sistema.

## 3.2 Robô KUKA LBR iiwa 14 R820$^{TM}$

O robô KUKA LBR iiwa 14 R820, por seu amplo campo de atuação e aplicação, pode ser descrito como um robô manipulador e foi desenvolvido pela empresa KUKA *Robotics* e desde 2013 continua ganhando inúmeros usuários na indústria aeronáutica (Xu et al., 2019). A diretriz de projeto para seu desenvolvimento está fundamentada na cooperação segura e eficiente entre humanos e robôs em ambientes industriais (KUKA Robotics, 2023a).

Dentre as principais características do LBR iiwa (*intelligent industrial work assistant*) pode-se citar:

Tabela 3.1: Características do LBR iiwa 14 R820

| **Característica** | **Descrição** |
|---|---|
| Capacidade de carga útil | 14 kg |
| Graus de liberdade | 7 |
| Segurança | Sensores integrados para detecção de colisões e forças externas |
| Peso estrutural | Utiliza materiais leves para diminuição de peso e inércia |
| Sistema compatíveis | ROS, Linux, Windows |
| Precisão e repetibilidade | Alta precisão e repetibilidade |
| Interfaces de comunicação | Ethernet, USB, CAN, Digital I/O e EtherCAT |

O termo LBR é uma sigla que deriva de *"Leichtbauroboter"*, que em alemão significa "robô de estrutura leve", que é uma característica importante do modelo e justificada pela sua natureza de interação humana. Para isso, o robô possui uma estrutura de materiais de baixa densidade específica (majoritariamente alumínio) pesando aproximadamente 29.9 kg (baixa inércia), munido de circuitos de atuação para uma resposta rápida de desaceleração em casos de colisão, o que evita acidentes em ambientes de proximidade com seres humanos.

O KUKA LBR iiwa 14 R820$^{TM}$ possui uma capacidade de carga de 14 kg, isto é, na condição mais crítica de operação é possível manipular objetos de no máximo 14 kg. E para adicionar maior dinamismo as atividades possíveis do LBR iiwa, o projeto foi concebido como um braço robótico de 7 graus de liberdade (GdL) como mostrado na figura 3.3. O que permite movimentos menos mecanizados, posicionamento mais precisos e maior manobrabilidade em diferentes direções. Devido ao número de GdL, para garantir a descrição completa da configuração de estado do robô são necessários descrever 7 posições e orientações (ângulo de redundância), além de 7 velocidades lineares e angulares. Com isso, tona-se possível determinar a posição e orientação do *Tool Center Point* (TCP).

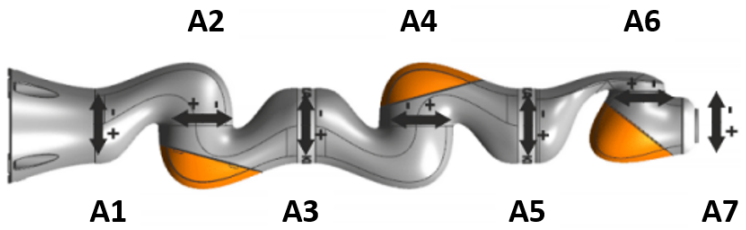



Figura 3.3: Juntas do robô LBR iiwa (KUKA Robotics, 2023a).

Vale ressaltar uma característica importante do projeto que novamente é oriundo dos requisitos necessários para a cooperação humana: o LBR iiwa é capaz de detectar e reagir a colisões ou forças externas durante a operação. Para essa atividade, o braço robótico vem acoplado de sensores de pressão e deslocamento integrados em suas juntas, bloqueando a resposta do robô em casos de colisões não previstas ou bruscas com um operador humano. Todo esse conjunto de sensores permitem ao braço robótico atingir precisão e repetibilidade de posicionamento de ±0.15 mm (KUKA Robotics, 2023a).

A utilização do KUKA LBR iiwa 14 R820$^{TM}$ como robô de suporte a pesquisa é compatível, pois apresenta vantagens de segurança, operação, popularidade e, principalmente, integração com o ROS. A utilização do LBR iiwa com ROS amplia as possibilidade de operação e programação do robô, de modo a facilitar a integração com ferramentas e aplicações não nativas ao sistema KUKA Sunrise.

### 3.2.1 KUKA Sunrise

O KUKA Sunrise, assim como todo o projeto LBR iiwa, foi pensado em uma arquitetura de *software* para controle baseada em componentes, querendo dizer que o controle do robô é dividido em módulos independentes desempenhando funções específicas (KUKA Robotics, 2023b). A ideia é fornecer de forma flexível recursos que podem ser combinados e reutilizados para criar sequências de tarefas personalizadas. Entre os principais componentes figuram:

- **KUKA Sunrise Workbench:** É a interface gráfica de usuário instalada na máquina do usuário que fornece ferramentas e possibilita a programação, simulação de tarefas robóticas complexas, definir parâmetros e configurar o comportamento do robô.

- **KUKA Sunrise OS:** É o sistema operacional que executa as tarefas de controle inicializadas pelo usuário em tempo de execução. A vantagem de utilizar um kearnel Linux para construir um sistema operacional especializado é a disponibilização de recursos avançados de controle de movimento, gerenciamento de sensores e atuadores, detecção de colisão e monitoramento de segurança.
- **KUKA Sunrise Cabinet:** É o hardware dos controladores, módulos de I/O e outras interfaces de comunicação.
- **KUKA Sunrise Connect:** É um conjunto de interfaces de comunicação para integração entre o KUKA com outros dispositivos, como sensores, atuadores, CLPs (Controladores Lógicos Programáveis) por exemplo. Trabalha com protocolos de comunicação padrões, como Ethernet/IP e OPC UA.

Na interface gráfica de programação, KUKA Sunrise Workbench, pode-se criar as aplicações utilizando a linguagem Java no computador de suporte a aplicação. Após criada a aplicação, pode-se ser transferida para o KUKA Sunrise OS executar a aplicação através do *SmartPad* do LBR iiwa, que é o sistema operacional no módulo do robô. Como dito anteriormente, muitas das capacidades apresentados pelo projeto LBR iiwa são em parte devido ao sistema operacional especializado construído sobre um *kernel* Linux. Mas a máquina de suporte pode rodar *a priori* qualquer sistema operacional. Dessa forma, a JVM (Java *Virtual Machine*) e a linguagem Java tornam-se as candidatas exatas para construção das aplicações.

A programação do robô se dá pela escolha da posição, velocidade e aceleração que devem ser atingidos. Para essa atividade, o KUKA Sunrise Workbench oferece opções de movimentação padrão (e.g. linha reta, curvado, etc.) e uma referência, geralmente estabelecida no centro da base do manipulador.

Além desses componentes principais, o KUKA Sunrise oferece uma ampla gama de bibliotecas, APIs e ferramentas de desenvolvimento que permitem a personalização e a extensão das capacidades do robô, incluindo recursos avançados de visão computacional, aprendizado de máquina e integração com sistemas de simulação.

Dentre as interfaces de comunicação possíveis, o KUKA Sunrise permite a integração com outros sistemas e dispositivos, através de conexão via Ethernet, que é a forma utilizada no Laboratório AeroTECH da EESC.

## 3.3 Gazebo

Na literatura dedicada ao tema é comum a utilização do Gazebo para simulação de aplicações robóticas (Peng et al., 2018; Tardioli et al., 2019; Ko et al., 2019). Dessa forma, o presente trabalho não visa a revolucionar em rejeitar o uso do Gazebo, mas se valer do *software* estendendo-o para novas aplicações.

O Gazebo, em essência, é um *software* que permite a simulação 3D de dinâmicas complexas para robôs e ambientes de aplicação virtualizado. Ele permite modelar robôs, ambientes e interações físicas. Esse projeto tem por objetivo a simulação de sensores, e novamente o Gazebo figura como um candidato lógico haja vista que fornece suporte a utilização de diversos sensores.

Pode-se citar algumas características do *software*:

- Permite a modelagem de objetos, terrenos, iluminação, gravidade, atrito e colisões.
- Permite a especificação de geometrias, vinculações, massa, inércia e rugosidade.
- Permite a simulação de câmeras, sensores de proximidade, giroscópios, *encoders*, sensores de força e atuadores.
- Permite simular interações físicas como colisões, detecção de colisões, dinâmica de corpos rígidos e simulação de forças e torque
- Permite integração com ROS para testagem e validação antes da implantação/operacionalização do robô
- Permite a adição de novas funcionalidades por meio de *plugins* para simular novos sensores ou dinâmicas específicas

Para a simulação é possível escolher e construir modelagens dinâmicas, tal qual *solvers* para realização dos cálculos e a esse conjunto é chamado de *physics engines*. Para o presente trabalho o motor de dinâmica utilizado foi o ODE (*Open Dynamics Engine*)

Apesar das inúmeras modelagens possíveis, um modelo em Gazebo não está livre da calibração dos parâmetros da simulação para assegurar a fidelidade em relação a um ensaio real. Assim como permanece sempre a solução de compromisso entre o esforço para um levantamento preciso de parâmetro para simulação e a complexidade em número de parâmetros e fenômenos a serem considerados sobre o impacto na precisão da simulação. Por isso, o projetista necessita entender o modelo a ser construído e a missão da simulação.

### 3.3.1 Modelagem

Para descrever robôs e ambientes virtuais é possível descrevê-los por meio de arquivos URDF (*Unified Robot Description Format*), SDF (*Simulation Description Format*) e XACRO (XML Macro). Esses arquivos são usados para descrever a estrutura, cinemática, propriedades físicas e outras informações relevantes do objeto descrito. Esses arquivos são escritos através da linguagem de marcação XML com o objetivo de descrever de forma estruturada as informações necessárias para a renderização e simulação do robô e do ambiente virtual.

A descrição de um robô ou elemento de ambiente pode ser definida de forma geral por 4 elementos:

- **Links (elos):** Os *links* representam partes físicas do robô, como o corpo, braços e pernas. Para utilizá-los é necessário o uso da *tag* <*link*> contêm informações sobre sua geometria, inércia, visualização e colisão
  - Informações sobre a geometria podem ser inseridas por meio de formas padrões (e.g. cilindro, esfera, cubo) ou arquivos de malha utilizando a *tag* <*visual*>
  - Informações sobre a massa e sobre a matriz de inércia podem ser inseridas por meio da *tag* <*inertia*>
  - Informações para descrever a geometria para colisões podem ser inseridas por meio da *tag* <*collision*>
- **Joints (juntas):** Os *joints* definem as conexões entre *links* (partes do robô) e estabelecem vinculações de posição, velocidade, força e torque na descrição do movimento relativo entre *links*. Para utilizá-los é necessário o uso da *tag* <*joint*>
  - Informações sobre o tipo de junta (e.g. revolução, rotação, deslizamento) são inseridos no *joint*
  - Informações sobre origem, orientação e eixo entre joints pode ser construído pelas *tags* <*origin*>, <*pose*> e <*axis*>
  - Informações sobre a hierarquia entre as juntas para construção do modelo completo do robô podem ser inseridas com as *tags* <*parent*> e <*child*>
- **Sensores:** Os sensores definem comportamentos específicos de sensores montados no robô, como câmeras, giroscópios e acelerômetros. Para utilizá-los é necessário o uso da *tag* <*sensor*> podem ter propriedades específicas, como tipo, posição e parâmetros de configuração.

- **Sistemas de referência:** Os sistemas de referência definem as coordenadas e eixos de referência de cada *link* e *joint*. Para a definição e transformação entre eixos de referência é utilizado o *TransForm version* 2 (tf2), o qual deve ser declarado no momento da criação de um nó ROS (3.4.2)

Vale ressaltar que o uso de uma biblioteca como tf2 se da pelo fato de que para descrever cada sistema de coordenadas em relação a outro presente no modelo, é necessário utilizar uma estrutura de dados em árvore, cuja implementação despende muito tempo de implementação.

Existem 3 formatos possíveis de arquivo de malha possível para definição da geometria e geometria de colisões: STL, OBJ e COLLADA, sendo seus sufixos *".stl"*, *".obj"* e *".dae"* respectivamente.

### 3.3.2 Arquivos SDF

Visando a construir um modelo utilizando o formato SDF, faz-se necessário a criação de uma estrutura de arquivos que deve estar organizada em uma pasta com o nome do modelo. O pacote mínimo de arquivos contidos na pasta são os arquivos de extensões SDF e CONFIG. O arquivo SDF é essencialmente um XML com *tags* específicas e o arquivo CONFIG é um arquivo de configuração estabelecendo variáveis como nome e uma documentação em texto sobre o modelo. Se existir uma geometria complexa para inclusão no modelo a ser desenvolvido, então esse pode ser descrito como uma *mesh* e imagens na mesma pasta.

As figuras 3.4, 3.5 e 3.6 demonstram a estrutura de pastas e a definição do arquivos CONFIG e SDF para a fuselagem.

Os elementos básico para a construção de arquivos SDF são:

- **<sdf>:** o elemento para indicar o início de uma arquivo SDF
- **<world>:** o elemento define propriedades físicas como gravidade e iluminação
- **<include>:** o elemento inclui novos modelos de robô no formato SDF
- **<model>:** o elemento indica o início da definição de um modelo de robô
- **<link> e <joint>:** os elementos definem as partes físicas (geometria, massa, inércia) e conexões (juntas e vínculos geométricos) entre as partes do modelo
- **<sensor>:** o elemento define sensores

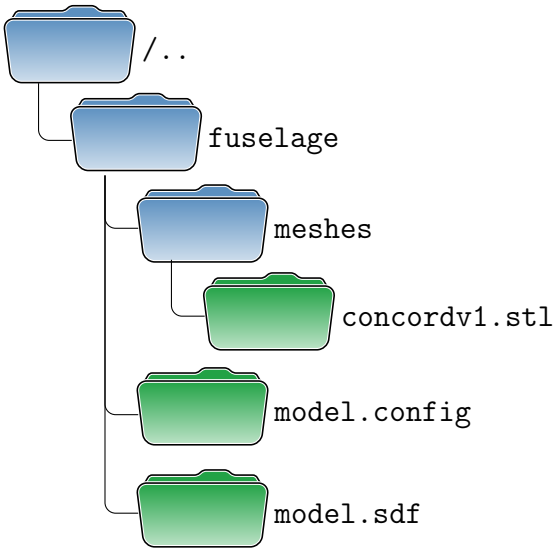



Figura 3.4: Estrutura de pastas para criação da fuselagem.

```
<model>
  <name>fuselage</name>
  <version>1.0</version>
  <sdf version="1.5">model.sdf</sdf>

  <author>
    <name>Luisa Martins Lemos de Souza</name>
    <email>luisa.martins.souza@usp.br</email>
  </author>

  <description>
       Fuselage model for Gazebo software
  </description>
</model>
```

Figura 3.5: Configuração do arquivo CONFIG.

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="fuselage">
    <static>true</static>
    <link name="fuselage">
      <inertial>
        <mass>40</mass>
      </inertial>
      <collision name="collision_fuselage">
        <geometry>
          <mesh>
            <uri>model://fuselage/meshes/concordv1.stl</uri>
            <scale>0.002 0.002 0.002</scale>
          </mesh>
        </geometry>
        <pose>0.0 0.0 0.0 0 0 0</pose>
        <surface>
        </surface>
      </collision>
      <visual name="visual_fuselage">
        <geometry>
          <mesh>
            <uri>model://fuselage/meshes/concordv1.stl</uri>
            <scale>0.002 0.002 0.002</scale>
          </mesh>
        </geometry>
        <material>
          <script>
          </script>
        </material>
      </visual>
    </link>
  </model>
</sdf>
```

Figura 3.6: Configuração do arquivo SDF.

- **<plugin>:** o elemento permite adicionar novas funcionalidades para o modelo do robô sendo desenvolvido

O Gazebo permite a construção de modelos em SDF quando incluídos seus elementos no ambiente de simulação por meio de uma exportação do modelo. Para isso, são utilizadas as ferramentas *Building Editor* e *Model Editor*. O *Model Editor* possibilita a inserção e posicionamento de modelos externos e a criação de juntas, enquanto o *Building Editor* permite a construção da geometria do modelo.

### 3.3.3 Arquivos URDF

O modo mais comum para criar um modelo de robô para uma simulação é por meio da utilização de arquivos URDF. Entretanto, dado a complexidade de escrita de um modelo em URDF, criou-se um novo tipo de arquivo chamado XACRO (Open Robotics, 2023b), que tem como missão a simplificação (não verboso), a flexibilidade (modularidade) e a reutilização (repetibilidade e escalabilidade) da descrição do robô de forma a ser semelhante a um arquivo SDF.

Um arquivo XACRO é convertido para um arquivo do tipo URDF por meio de uma biblioteca chamada xacro que pode ser instalada na distribuição do ROS.

Devido a semelhança aos arquivos SDF, o arquivo XACRO possui o mesmo conjunto de *tags* básicas apresentados pelos arquivos SDF, como descritos na seção 3.3.2

A figura 3.7 demonstram um arquivo XACRO utilizado para construir parte da descrição do LBR iiwa.

### 3.3.4 Ambiente virtual padrão e construção do *world*

Uma simulação iniciada em Gazebo parte de uma configuração padrão chamada de *empty world* (mundo vazio). Um *empty world* oferece um ambiente limpo e vazio, sem nenhum obstáculo ou objeto predefinido com apenas um piso padrão e uma fonte de iluminação. Ele fornece uma tela em branco para adicionar e posicionar os elementos que se deseja simular, como robôs, objetos, sensores e outros componentes.

```
<?xml version="1.0"?>
<robot name="iiwa14" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <!-- Import Rviz colors -->
  <xacro:include filename="$(find iiwa_description)/urdf/materials.xacro" />
  <!--Import the lbr iiwa macro -->
  <xacro:include filename="$(find iiwa_description)/urdf/iiwa14.xacro"/>
    <!--Import the lbr iiwa macro -->

  <xacro:arg name="hardware_interface" default="PositionJointInterface"/>
  <xacro:arg name="robot_name" default="iiwa"/>
  <xacro:arg name="origin_xyz" default="0 0 0"/>
  <xacro:arg name="origin_rpy" default="0 0 0"/>

  <!-- Fix to world just for testing -->
  <link name="world"/>

  <!--iiwa-->
  <xacro:iiwa14 hardware_interface="$(arg hardware_interface)" robot_name="$
(arg robot_name)" parent="world">
    <origin xyz="$(arg origin_xyz)" rpy="$(arg origin_rpy)" />
  </xacro:iiwa14>

</robot>
```

Figura 3.7: Configuração do arquivo XACRO.

A partir de um arquivo tradicionalmente chamado de *"world.world"* é possível incluir novos componentes e estabelecer uma aplicação real, que, após construído, deverá inicializar o mundo da aplicação em toda nova simulação.

Assim como os demais arquivos, um arquivo *world* é um arquivo escrito em XML contendo as etapas para construção do cenário da simulação. Na figura 3.8 é possível a construção do mundo típico da simulação utilizada nesse trabalho.

### 3.3.5 *Plugins*

Os *plugins* são implementações de funcionalidades adicionais que podem ser geralmente atribuídas a nós ROS (3.4.2) fornecendo assim mais recursos para o nó na simulação. A grande vantagem é permitir a modularidade, reutilização e intercambialidade dentro de sistemas ROS.

Essa versatilidade permitiu o surgimento de inúmeros *plugins* ao longo dos anos. Para a implementação é necessário escrever o código que cria a funcionalidade desejada definindo entradas e saídas esperadas do modelo. Geralmente, os *plugins* são escritos em C/C++, ou outras linguagens que possibilitem a compilação e a alta performance. Então, o código é compilado para uma biblioteca compartilhada (*".so"*).

```
<?xml version="1.0" ?>

<sdf version="1.5">
        <world name="default">

                <!-- Camera sensor plugin -->
                <plugin name="ros_link_attacher_plugin" filename="libgazebo_ros_link_attacher.so"/>
                <scene>
                        <ambient>0.4 0.4 0.4 1</ambient>
                        <background>0.25 0.25 0.25 1</background>
                        <shadows>false</shadows>
                </scene>
                <light type="directional" name="some_light">
                        <diffuse>0.6 0.6 0.6 0</diffuse>
                        <specular>1 1 1 0</specular>
                        <direction>-1 -1 -1</direction>
                </light>

                <!-- A ground plane -->
                <include>
                        <uri>model://ground</uri>
                </include>

                <!-- manequins ao redor do robô-->
                <include>
                        <pose>11.250382 2.703555 0.0 0.0 0.0 -1.611704</pose>
                        <uri>model://manequins</uri>
                        <name>manequins</name>
                </include>
```

Figura 3.8: Descrição do arquivo *world* utilizado no projeto.

É corriqueiro o uso de *plugins* para fornecimento de *drivers* para hardware, adicionar funcionalidades de processamento de dados e implementar algoritmos autorais. Eles são geralmente classificados em 6 tipos: *World*; *Model*; *Sensor*; *System*; *Visual*; e GUI.

Um tipo de *plugin* está atrelado a funcionalidade desejada. Por exemplo, o tipo *World* pode estender ou alterar as propriedades do cenário e do robô como o mecanismo de física e iluminação ambiente. De semelhante modo, o tipo *Model* possibilita novas vinculações e controles sobre juntas e estado de um modelo. Assim como, o tipo Sensor é necessário para captação de informações do sensor e controlar propriedades do sensor, tais como o sensor de câmera utilizado nesse trabalho.

E por fim, os *plugins* devem ser carregados dentro do escopo da aplicação. Então, o tipo *Model* deve estar associado a um arquivo que define o comportamento do modelo, assim como o tipo *World* deve estar associado a um arquivo *world*.

## 3.4 ROS

ROS (*Robot Operating System*) é um *framework* de código aberto utilizado para desenvolver *software* de controle e comunicação em robótica. Ele fornece uma coleção de bibliotecas, ferramentas e convenções que visam a simplificar o desenvolvimento de *software* para robôs (Open Robotics, 2023b).

O ROS foi inicialmente desenvolvido em 2007 pela *Willow Garage*, uma empresa de pesquisa em robótica, e agora é mantido e evoluído pela *Open Robotics*. Ele foi projetado para ser um sistema operacional flexível e distribuído, fornecendo uma estrutura para que os desenvolvedores possam criar e integrar componentes de *software* em um robô.

O ROS possui uma arquitetura modular, baseada em troca de mensagens assíncronas, que permite que os diferentes componentes do sistema (chamados de *"node"* - 3.4.2) se comuniquem de maneira eficiente. Os nós podem ser escritos em várias linguagens de programação, como C++, Python e outros, facilitando a integração de diferentes bibliotecas e pacotes.

Além disso, o ROS oferece uma ampla variedade de ferramentas para depuração, visualização e simulação de robôs, bem como uma comunidade ativa de desenvolvedores que contribuem com pacotes e recursos adicionais. Essa combinação de recursos torna o ROS uma plataforma popular e poderosa para o desenvolvimento de *software* em robótica em várias áreas, como pesquisa, indústria e educação.

Nesse trabalho, foi utilizado a distribuição **ROS Melodic** instalada em uma distribuição Linux Ubuntu 18.04.

### 3.4.1 Mensagens no ROS

As mensagens em ROS são uma forma estruturada de comunicação e troca de dados entre os nós ROS. Em essência, as mensagens são instâncias de uma classe que padroniza a informação que é trocada (Open Robotics, 2023b).

A mensagem em ROS, como uma classe, possui atributos com tipos de dados específicos e uma interface de abstração de tipo. Dessa forma, a depender da mensagem instanciada, os atributos podem possuir um tipo predeterminado, como inteiros, *floats*, *strings*, *booleans*, matrizes, imagens, ou até mesmo outras mensagens.

A princípio as mensagens no ROS carregam informações sobre o estado do sistema, comandos, leituras de sensores, dados processados, dados de saída. E todas as mensagens são alocadas em tópicos para posterior consulta.

Nesse trabalho, pode-se observar principalmente o fluxo de mensagens de geometria, mensagem de texto e imagem:

- *"geometry_msgs/Pose.msg"*
- *"geometry_msgs/Point.msg"*
- *"geometry_msgs/Twist.msg"*
- *"geometry_msgs/Quaternion.msg"*
- *"std_msgs/String.msg"*
- *"sensor_msgs/Image"*

### 3.4.2 Nós ROS

Um nó (*node*) ROS é uma unidade de execução responsável pela realização de uma tarefa específica associado a um processo. Por exemplo, um nó ROS realiza aquisição de dados de sensores, processamento de informações, controle de atuadores e execução de algoritmos.

Os nós ROS foram projetados para serem modularizados, o que permite a construção de diversos nós, cada um executando uma tarefa, e que em conjunto desenvolvam uma aplicação completa.

Como dito anteriormente, os nós podem se comunicar, e para isso podem utilizar mensagens, tópicos, serviços e ações e parte dessa estrutura de comunicação pode ser vista na figura 3.9.


Figura 3.9: Elementos em ROS: (vermelho) Nó ROS; (verde) Tópico ROS; (azul) *Namespace*; e (ciano) Mensagem ROS.

### 3.4.3 Tópicos ROS

Os tópicos ROS são as interfaces que os nós do sistemas possuem para realizar a transferência de informação.

Em um tópico ROS, existe o nó ROS que publica mensagens e outros nós que se inscrevem para receber essas mensagens. O nó ROS que publica as mensagens é chamado de *publisher*, enquanto os nós ROS que se inscrevem para receber as mensagens são chamados de *subscribers*.

Sendo assim, toda vez que o nó ROS responsável pela publicação, publicada em um tópico ROS, a mensagem é transmitida para todos os nós ROS inscritos nesse tópico ROS. Vale ressaltar que não existe a obrigatoriedade de existir um *publisher* nem um *subscriber* para existência do tópico ROS, assim como um nó ROS pode receber mensagens de diversos tópicos ROS para se aquisitar as informações do sistema, tal qual mais de um nó ROS pode publicar mensagens em um tópico ROS.

Os tópicos ROS possuem uma organização em pasta (uso de *"/"*) e podem ser acessados por meio do comando *"rostopic"*.

### 3.4.4 Serviços ROS

Para realizar uma comunicação entre nós em ROS pode-se utilizar além das mensagens um outro recurso chamado de serviço.

Diferentemente das mensagens que seguem um paradigma orientado a eventos, no qual existem nós que se inscrevem para receber uma mensagem publicada por um nó a qualquer momento e sem nenhum compromisso de estabelecer uma resposta (comunicação assíncrona), os serviços estabelecem uma forma de comunicação entre nós que demanda uma resposta do nó que fornece o serviço para o nó solicita o serviço para que a comunicação seja efetivada (comunicação síncrona).

Uma comunicação síncrona significa que o nó solicitante aguarda a resposta do nó solicitado para assim continuar a execução da tarefa. Esse tipo de comunicação a princípio estabelece uma comunicação semelhante às mensagens, entretanto é utilizada quando a informação a ser trocada é sensível, exigem uma interação complexa, necessita ser priorizada e garantida a troca

da informação (e.g. estado de um robô, valor de um sensor e coordenação entre processos diferentes nós).

Um serviço no ROS precisa ser executado a cada vez que é demandado a troca de informação entre nós para a execução de um processo. Ele pode demandar parâmetros para ser executado ou mesmo não retornar nenhuma mensagem, apenas executando uma tarefa.

Para a criação de um serviço no ROS é necessário a existência de três elementos:

- **Service Provider:** É o nó que executa as tarefas lógicas implementadas nele e envia a resposta solicitada
- **Service Client:** É o nó que solicita e aguarda a resposta do serviço
- **Service Message:** É a resposta transmitida no formato que foi especificado pelo nó cliente do serviço. Assim como nas mensagens do ROS, a mensagem de serviço é definida em um arquivo em formato *".srv"*.

Nesse trabalho, pode-se encontrar o uso de alguns serviços típicos como: para pausar a simulação do Gazebo (*"gazebo/pause_physics"*); retomar a simulação no Gazebo (*"gazebo/unpause_physics"*); solicitar as informações do ambiente simulado (*"gazebo/get_world_properties"*) como nome da ferramenta instanciada na simulação.

### 3.4.5 Ações no ROS

As ações no ROS são uma terceira forma de comunicação que se assemelham as mensagens no ROS, uma vez que possuem a mesma característica de serem assíncronas. Entretanto, enquanto uma mensagem envia, quando solicitada uma mensagem pontual, momentânea, e após o encerramento da execução da atividade, as ações em ROS iniciam um processo de comunicação contínua e duradoura, com o envio de informações de forma assíncrona, contínua, em tempo de execução e por longos períodos de duração.

Outra característica desse tipo de comunicação é quanto a existência da bidirecionalidade. Esse meio de comunicação é utilizado em casos que a execução de tarefas requerem o uso de informações continuamente e atualizadas com maior frequência por um nó (*feedback* contínuo) para assegurar o controle ideal (e.g. navegação autônoma de um robô e controle de precisão de um braço robótico). Para isso, a ação no ROS implementa um *goal* (objetivo), *feedback* (retorno)

e *result* (resultado). de forma que, o nó que inicia a ação envia um *goal* contendo os parâmetros e as instruções necessárias para a tarefa. Durante a execução, o nó que realiza a ação envia *feedback* para informar sobre o progresso ou estado atual da tarefa. Por fim, quando a tarefa é concluída, o resultado final é enviado de volta para o nó inicial.

Por fim, os elementos *feedback* e *result* são definidas em um arquivo de extensão *".action"*.

Essa estrutura de ações no ROS pode ser vista por meio dos conjuntos de tópicos ROS que definem as ações ROS mostradas na figura 3.10.

### 3.4.6 Comando *roslaunch*

O comando *"roslaunch*"é uma ferramenta do ROS que possibilita a inicialização e gerenciamento dos modelos que compõem o sistema simulado.

O *"roslaunch"* é usado para executar de forma centralizada a inicialização dos nós ROS que estavam previstos para o modelo. Tradicionalmente, pode ser indicado os parâmetros a serem definidos e as configurações a serem aplicadas. O arquivo de lançamento é escrito em XML e possui o formato *".launch"* como mostrado na 3.11.

A fim de garantir a automatização da inicialização de modelos complexos compostos de diversos robôs, é possível encadear diversos arquivos *"roslaunch"* de maneira lógica e realizar todas as etapas da inicialização de um modelo em apenas uma chamada.

### 3.4.7 Parâmetros

Os parâmetros no ROS são variáveis configuráveis em tempo de execução que definem características específicas do sistema ou personalizam o comportamento dos componentes do ROS. Eles podem ser ajustados em diferentes níveis e permitem criar sistemas mais flexíveis e adaptáveis. Os parâmetros podem ser configurados antes ou durante a execução do sistema e podem ser acessados e modificados por outros nós. Eles podem ser definidos manualmente ou carregados a partir de arquivos de configuração, como arquivos YAML (*Yet Another Markup Language*) ou XML, durante o lançamento do sistema.

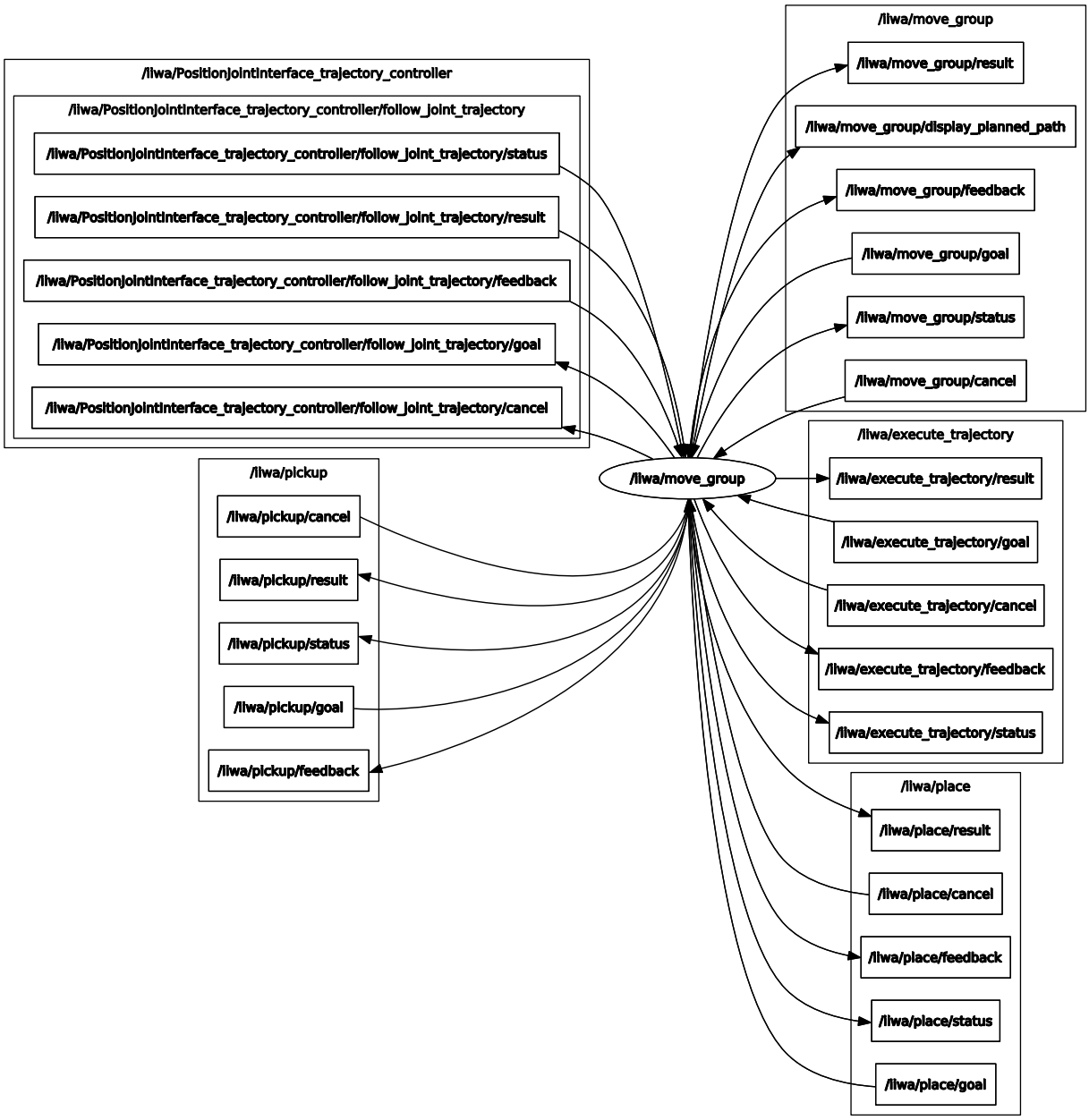



Figura 3.10: Conjuntos de tópicos ROS definindo as ações ROS executadas pelo nó ROS *move_group*.

### 3.4.8 *Namespaces*

Os *namespaces* no ROS são utilizados para organizar e agrupar os nomes de tópicos, serviços e parâmetros. Eles evitam conflitos de nomes, fornecem uma estrutura hierárquica para a comunicação e configuração do sistema e facilitam a seleção e organização de componentes em sistemas complexos. Ao utilizar *namespaces*, é possível criar uma estrutura de nomeação

```
<?xml version="1.0"?>
<launch>

  <!-- ================================================= -->
  <!-- |    Lauch file to start Gazebo with an IIWA     | -->
  <!-- |    and a simulated Sunrise controller.         | -->
  <!-- ================================================= -->

  <arg name="hardware_interface" default="PositionJointInterface" />
  <arg name="robot_name" default="iiwa"/>
  <arg name="model" default="iiwa14"/>
  <arg name="trajectory" default="true"/>

  <include file="$(find iiwa_gazebo_airplane_manufacturing)/launch/iiwa_gazebo.launch">
    <arg name="hardware_interface" value="$(arg hardware_interface)" />
    <arg name="robot_name" value="$(arg robot_name)" />
    <arg name="model" value="$(arg model)" />
    <arg name="trajectory" value="$(arg trajectory)" />
  </include>

  <include file="$(find iiwa_control)/launch/iiwa_sunrise.launch">
    <param name="hardware_interface" value="$(arg hardware_interface)"/>
    <param name="robot_name" value="$(arg robot_name)"/>
    <param name="model" value="$(arg model)"/>
    <param name="tool_length" value="0.0"/>
  </include>

</launch>
```

Figura 3.11: Descrição do arquivo *launch* utilizado no projeto.

clara e organizada para os componentes do sistema. Isso é particularmente útil em sistemas com múltiplos nós, onde diferentes partes do sistema podem compartilhar nomes sem ambiguidade. Os *namespaces* são representados como partes do nome precedidas por uma barra (*"/"*) e separadas por barras adicionais. Por exemplo, *"/iiwa/camera_sensor/image_raw"* é um exemplo de um tópico que usa *namespaces*.

### 3.4.9 RVIZ

O RVIZ é uma ferramenta de visualização 3D no ROS que permite aos usuários ver e interagir com dados de sensores e modelos de robôs em tempo real. Ele é usado para análise, depuração e simulação de sistemas robóticos, oferecendo uma interface gráfica personalizável para visualização e interação com dados em um ambiente tridimensional.

### 3.4.10 Sensoriamento de câmeras

O sensoriamento de câmeras inicia-se com a correta configuração dos sensores na simulação. Para isso, deve-se fornecer a correta especificação da câmera e seus parâmetros no ROS (e.g. tipo de câmera, resolução, frequência de aquisição, etc.) e capturar as imagens da câmera e publicá-las em um tópico de imagem.

Para utilizar sensores de câmera no ROS pode-se seguir por alguns caminhos: utilizar o RVIZ; realizar a leitura do tópico ROS diretamente; e construir um nó que se inscreve no tópico que é publicado as informações oriundas da câmera e transformar essas informações em imagens.

A utilização do RVIZ envolve a interação com os dados de uma câmera em tempo real configurando o RVIZ para exibir o tópico de imagem da câmera com o uso de um *display* de imagem associado ao tópico de imagem correto

Outra maneira é utilizando a leitura do tópico ROS diretamente e assim exibir em uma tela padrão a visualização da mensagem de imagem publicada no tópico.

E por fim, pode-se utilizar construir um nó que é responsável pelo processamento externo. Para isso segue as seguintes etapas:

1. Criar um nó no ROS
2. Inscrever o nó para leitura do tópico da câmera que publica imagens
3. Realizar o pós processamento das imagens por meio de bibliotecas adicionais nas linguagens de programação utilizadas no projeto
4. Salvar as imagens em um pasta local ou servidor

Nesse trabalho estão disponíveis as 3 formas de sensoriamento de câmeras, estando a última disponível para utilização em calibrações, algoritmos de otimização e aprendizado de maquina.

## 3.5 Planejamento de trajetória e a biblioteca MoveIt

O planejamento de trajetória consiste da determinação das posições e orientações que o robô deverá percorrer para realizar a mudança para um posição e orientação desejada (estado desejado).

Como citado anteriormente, o LBR iiwa possui 7 graus de liberdade de forma que para determinar as configurações de estado necessárias até o atingimento da posição final é uma tarefa com múltiplas soluções.

No painel de controle do LBR iiwa existem algumas opções padrões de movimentação como uma movimentação em linha reta entre dois pontos. Para executá-la, escolhe-se esta opção e informa-se o estado final desejado. Semelhantemente faz-se para uma movimentação circular, a qual demanda da especificação adicional de um ponto intermediário para o cálculo do arco. Há, ainda, outras opções de movimentação com estados intermediários ou que controlam também a orientação do braço durante a trajetória, não só a posição.

Para a escolha da trajetória, o KUKA Sunrise.*Operating System* realiza diversos cálculos para definir uma trajetória dentre as múltiplas soluções. Entretanto, esse processo de escolha não é disponibilizado pelo fabricante, tornando imprevisível a trajetória que o robô deverá percorrer para atingir seu objetivo (especialmente entre configurações de estado com grandes deslocamentos).

Nesse cenário, uma simulação de trajetória a ser desenvolvida pelo robô exerce uma função de planejamento de trajetória visando tornar o processo mais previsível e são consideradas recursos necessários para a operação do braço robótico LBR iiwa.

Diante dessa necessidade, esse trabalho tem por objetivo o uso da ferramenta MoveIt para o planejamento dessa trajetória

### 3.5.1 *MoveIt*

O MoveIt é um conjunto de bibliotecas, ferramentas e recursos no ROS que fornecem funcionalidades avançadas de planejamento de movimento para robôs. Ele é projetado para facilitar o controle e a execução de movimentos complexos, como manipulação de objetos e navegação autônoma de robôs.

O MoveIt oferece recursos abrangentes para o planejamento de trajetórias, cinemática inversa, detecção e colisão de objetos, além de interfaces intuitivas para controlar e monitorar o movimento do robô.

O MoveIt se baseia na definição de metas de movimento, ou seja, nas posições e orientações desejadas no final da trajetória, como alcançar uma posição específica ou executar uma sequência de movimentos.

Por meio da interface do MoveIt é possível visualizar com antecedência a movimentação que deverá ser desenvolvida, permitindo assim a validação, teste e previsibilidade dos movimentos antes de executados pelo robô real. Assim é possível prever erros e mitigar riscos relacionados a operação do robô.

Para realizar um planejamento de trajetória é possível utilizar a própria interface gráfica do MoveIt. Para isso, deve-se arrastar a posição do efetuador e/ou rotacionar a orientação do efetuador até a configuração desejada ao final do movimento. Assim, clica-se em *"Plan"*, então a trajetória é planejada e exibida; e ao clicar em *"Execute"*, a trajetória é então executada.

## 3.6 Pacote *iiwa_stack*: descrição do LBR iiwa

Para a realização desse trabalho foi utilizado o pacote chamado *"iiwa_stack"* na versão 1.3.0 que implementa a lógica de operacionalização do robô LBR iiwa e a interface de comunicação entre ROS e o LBR iiwa. O pacote utiliza as características de reaproveitamento de código viabilizadas pelo ROS, de forma que foi implementado visando a reutilização do pacote para uso em novas aplicações que estendem suas funcionalidades. Nesse sentido, esse trabalho busca estender a utilização do pacote *"iiwa_stack"* para uma aplicação no cenário de manufatura na indústria aeronáutica. Devido as facilidades proporcionadas pelo MoveIt e do Gazebo, o pacote *"iiwa_stack"* tornou-se uma opção lógica, uma vez que possui integração com ambas as ferramentas. A utilização do pacote instanciado no Gazebo é exemplificado na figura 3.12.

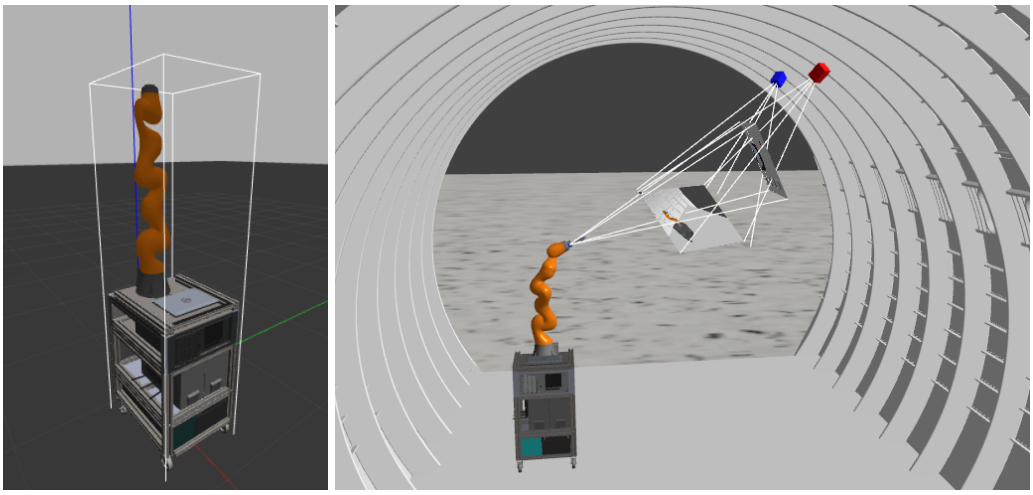


Figura 3.12: Aplicação do pacote *iiwa_stack* para simulações (Boer, 2022).

O pacote *"iiwa_stack"* utilizado nesse trabalho é resultado dos esforços de pesquisa e desenvolvimento do Laboratório de Pesquisa Interdisciplinar em Procedimentos Médicos Auxiliados por Computador (IFL-CAMP) da Universidade Técnica de Munique (TUM), entretanto foi inicialmente implementado pelo fabricante KUKA *Robotics*, estando disponível no Github do IFL-CAMP (*https://github.com/IFL-CAMP/iiwa_stack/*). De acordo com Hennersperger et al. (2016), o pacote fornece uma interface de comunicação entre ROS e o LBR iiwa que possibilita o controle e monitoramento do robô através do ROS. O objetivo é utilizar todas as vantagens presentes na plataforma ROS, para maximizar as funcionalidades e facilitar o trabalho de operacionalização do LBR iiwa. Com o pacote *"iiwa_stack"*, os usuários desfrutam de *drivers* que permitem implementar tarefas de manipulação, programar movimentos complexos e de baixo nível como o movimento dos eixos, leitura de sensores e até mesmo criar aplicações de controle em tempo real para o robô LBR iiwa (Hennersperger et al., 2016). Além disso, o pacote inclui módulos para simulação em Gazebo e um emulador do KUKA Sunrise.OS (simulando a interface de comandos). A organização do pacote mínimos necessários do LBR iiwa está descrita a seguir:

- **"iiwa_description":** contém a definição do modelo dinâmico e visual do robô no formato URDF
- **"iiwa_driver":** contém os *drivers* para comunicação cliente-servidor com o controlador instalado no LBR iiwa
- **"iiwa_hw":** funciona como uma camada de abstração entre o ROS e o controlador do LBR iiwa. Se traduz em um hardware específico para uso do ROS *Control*, fornecendo os recursos necessários para a comunicação entre *"iiwa_driver"* e o ROS *Control*.

- **"iiwa_control":** contém os arquivos de configuração dos controladores para uso na simulação.
- **"iiwa_moveit":** contém os arquivos necessário para realizar a integração com o MoveIt.

Como citado anteriormente, o objetivo do pacote é possibilitar a comunicação entre o ROS e o LBR iiwa. Para isso, é necessário definir a interface de hardware entre ROS e o hardware de controle do LBR iiwa (KUKA Sunrise Cabinet).

Inicialmente, é enviado pelo ROS a orientação das juntas (um vetor com orientações desejadas para cada junta) como uma mensagem, que deve ser comunicada ao controlador do LBR iiwa.

Essa mensagem é enviada para o KUKA Sunrise Cabinet através da interface de hardware chamada *"PositionJointInterface"* (implementado no pacote *"iiwa_hw"*). Em posse dessas informações o controlador calcula a trajetória e as forças necessárias para mover as juntas do robô para as posições desejadas.

Vale reforçar que não é conhecido o código fonte do *firmware* do KUKA Sunrise Cabinet e da estratégia de controle que é executado no LBR iiwa para determinar e executar a trajetória.

A interface de hardware *"PositionJointInterface"* utiliza para a comunicação um protocolo TCP/IP através de uma conexão Ethernet que é estabelecida pelo pacote *"iiwa_driver"*.

A execução dessas tarefas demanda um controlador. O controlador configurado é o *"PositionJointInterface_trajectory_controller"*, que está implementado no pacote *"iiwa_hw"* e tem por objetivo o controle de trajetória. O pacote possui integração com o MoveIt por meio do *"iiwa_moveit"*. Portanto a trajetória uma vez configurada no MoveIt, é transmitida ao *"PositionJointInterface_trajectory_controller"*, que envia mensagens para o *"PositionJointInterface"* para realizar o controle de posição do LBR iiwa. O *"PositionJointInterface_trajectory_controller"* é um controlador de posição do PID, definindo os ganhos proporcional, derivativo e integrativo e recebendo *feedback* do controlador físico implementado no LBR iiwa (KUKA Sunrise Cabinet). Por fim, a estrutura de nós e tópicos com as respectivas descrições de *subscribers* e *publishers* pode ser encontrada no **Apêndice B** (6.2).

## 3.7 Pacote *"gazebo_ros_link_attacher"*: manipulação de objetos em tempo de execução

O pacote *"gazebo_ros_link_attacher"* é responsável pela criação de junções para fixação e desafixação de *links* (partes do robô simulado) em tempo de execução por meio da publicação de mensagens em tópicos ROS.

O pacote opera atualizando as restrições físicas e dinâmicas das junções para garantir as partes do robô permaneçam conectados ou se desconectem. De forma que, a remoção e inserção de objetos durante a simulação pode ser realizada por meio de um serviço ROS disponibilizado pelo Gazebo. Entretanto, é necessário que o objeto inserido mova-se junto com o robô, e, para isso, após sua inserção, uma junção fixa entre o objeto e o manipulador do robô deveria ser criada, o que só foi possível com o uso do *plugin "gazebo_ros_link_attacher"* (Open Robotics, 2023a).

O início da execução ocorre após o envio de uma mensagem de texto com os nomes dos elementos a serem fixados ou desafixados e, a partir disso, um nó ROS é responsável pela execução da atividade (e.g. troca de ferramenta) requisitando a execução de dois serviços. E, é através dos serviços *"/link_attacher_node/attach"* e *"/link_attacher_node/detach"* que é possível criar e excluir junções (Open Robotics, 2023a).

Para o funcionamento correto, o pacote necessita ser incluído como um *plugin* no arquivo de extensão *world* 3.3.4, para que a funcionalidade seja iniciada em conjunto com o ambiente simulado.

CAPÍTULO 4

# RESULTADOS E DISCUSSÃO

## 4.1 Pacote *"airplane_manufacturing_stack"*: aplicação do LBR iiwa para processos de manufatura aeronáutica

Visando a utilização do robô LBR iiwa em aplicações voltadas a processos de manufatura na indústria aeronáutica, buscou-se por meio desse trabalho desenvolver um pacote construído em ROS utilizando como ambiente de simulação Gazebo e que estendesse as funcionalidades dos pacotes *"iiwa_stack"* e *"gazebo_ros_link_attacher"*.

Esse pacote foi nomeado de *"airplane_manufacturing_stack"* e os arquivos e códigos utilizados no desenvolvimento estão organizados nos sub pacotes mostrados na figura 4.1.

- **"airplane_manufacturing_setup":** contém os arquivos que descrevem os elementos incluídos no ambiente da simulação
- **"iiwa_airplane_manufacturing_moveit":** contém os arquivos de configuração do MoveIt
- **"iiwa_airplane_manufacturing_description":** contém os arquivos que invocam, incluem e sobrescrevem características do robô LBR iiwa provenientes do pacote *"iiwa_stack"*
- **"iiwa_gazebo_airplane_manufacturing":** contém os arquivos para inicialização da simulação no Gazebo, além de incluir os outros elementos da simulação

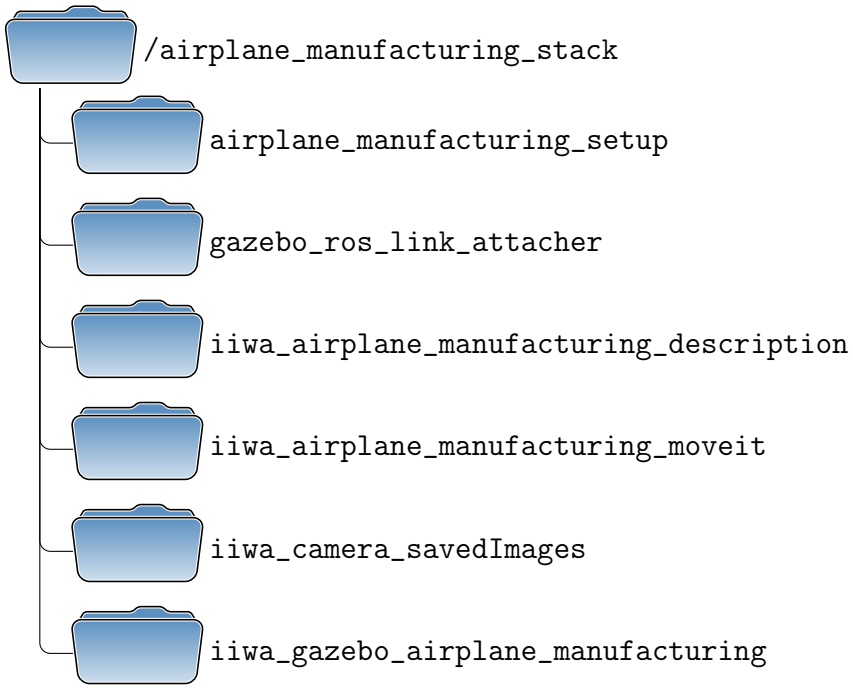



Figura 4.1: Organização dos sub pacotes da aplicação *airplane_manufacturing_stack*.

- **"gazebo_ros_link_attacher":** contém os arquivos que criam e excluem junções entre elementos da simulação em tempo de execução
- **"iiwa_camera_savedImage":** contém as imagens capturas pelos sensores de câmera ao longo da simulação

Nas seções 4.1.1, 4.1.2 e 4.1.5 descreve-se em detalhes a o desenvolvimento dos sub pacotes.

### 4.1.1 Descrição do ambiente: *airplane_manufacturing_setup*

O pacote ROS chamado *"airplane_manufacturing_setup"* foi desenvolvido para criação de um ambiente especializado de uma aplicação do robô LBR iiwa contexto da indústria aeronáutica. O contexto dessa aplicação é um ambiente de uma etapa da manufatura de uma seção central de fuselagem em um hangar utilizando o LBR iiwa. Nesse pacote podem ser encontrados os arquivos indicados na figura 4.2.

Dentro da pasta *"launch"*, existe o arquivo específico que inicializa o Gazebo apenas com o ambiente com as especificações prescritas no arquivo *"airplane_manufacturing_world.world"* (localizado na pasta *"world"*).

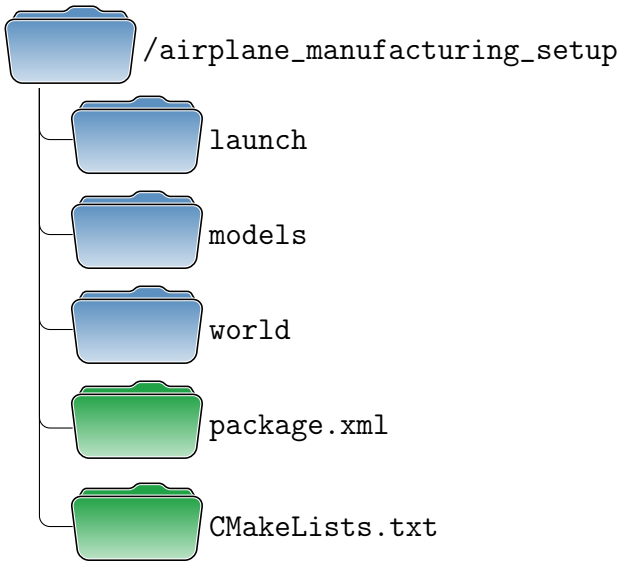



Figura 4.2: Organização das sub pastas dentro do pacote *airplane_manufacturing_setup*.

A pasta *"models"*, contém outras pastas, uma para a descrição de cada elemento da simulação. Dentro de cada subpasta é onde estão as definições dos modelo em arquivos SDF (3.3.2). Essencialmente são arquivos XML com *tags* para descrever a geometria, geometria de colisões, material, inércia, massa, cinemática, arquivos de malha da geometria e escala. Nesses arquivos não são demandadas muitas informações adicionais uma vez que são elementos que completam o cenário e não possuem uma dinâmica.

Além disso, o pacote *"airplane_manufacturing_setup"* também contém os dois arquivos comuns a todos os pacotes ROS: *"CMakeLists.txt"* e *"package.xml"*. Esses arquivos são usados para a compilação do pacote e fornecem informações sobre as dependências, configurações de compilação e outras informações relevantes.

A inclusão de elementos ocorreu de acordo com as necessidades e todos os elementos incluídos utilizaram arquivos STL desenvolvidos em CAD para inclusão de suas geometria. Arquivos STL são arquivos de malha e se possuírem muitos detalhes podem reduzir o desempenho da simulação. Por isso, foram incluídos apenas os elementos mais importantes para a realização de uma simulação exemplo como mostrado na figura 4.3:

- Fuselagem
- Suporte da fuselagem
- Mesa de suporte do LBR iiwa

- Operadores do LBR iiwa
- Hangar
- Solo

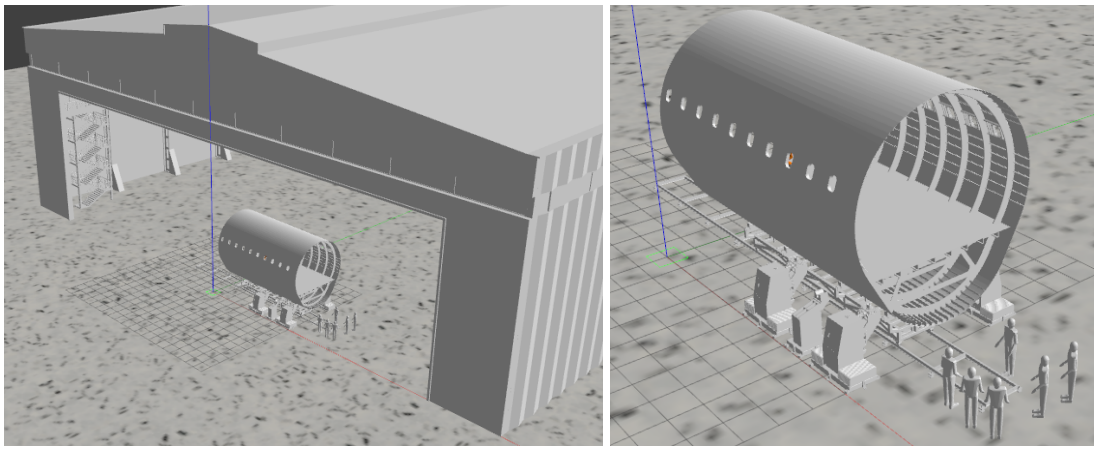


Figura 4.3: Elementos implementados para simulação do ambiente de aplicação.

Um detalhe importante na definição deste pacote é a definição, no arquivo *"package.xml"*. Esse arquivo indica a busca e inclusão de novos modelos para o ROS para o Gazebo como mostrado na figura 4.4.

### 4.1.2 Descrição do LBR iiwa: *iiwa_airplane_manufacturing_description*

No pacote chamado *"airplane_manufacturing_description"*, encontra-se a extensão da descrição do modelo do robô LBR iiwa. Essa extensão inclui um suporte móvel (*trolley*) e a utilização de sensores de câmera como deverá ser descrita na seção 4.1.4.

O suporte foi projetado para acomodar os componentes utilizados para executar os 5 módulos que compõem esse projeto como descrito na seção 3.1, isto é, o computador, roteador, baterias o suporte e o próprio robô LBR iiwa.

### 4.1.3 Inclusão do suporte ao robô LBR iiwa

Para criar a descrição especificada com a reutilização do pacote *"iiwa_stack"* foi necessário criar um arquivo XACRO. Nesse arquivo, A descrição do LBR iiwa é oriunda do pacote *"iiwa_stack"*, de forma a referenciar os arquivos URDF do pacote *"iiwa_stack"*. Nesse mesmo

```
<?xml version="1.0"?>
<package format="2">
  <name>airplane_manufacturing_setup</name>
  <version>0.0.0</version>
  <description>Airplane manufacturing setup package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->

  <maintainer email="luisa.martins.souza@usp.br">luisamls</maintainer>

  <buildtool_depend>catkin</buildtool_depend>

  <depend>gazebo_ros</depend>

  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <gazebo_ros gazebo_model_path="${prefix}/models"/>
  </export>
</package>
```

Figura 4.4: Definição do arquivo package.xml.

arquivo XACRO, é adicionado um elo para descrever o suporte e uma junção conectando o suporte ao robô como pode ser visto na figura 4.5.

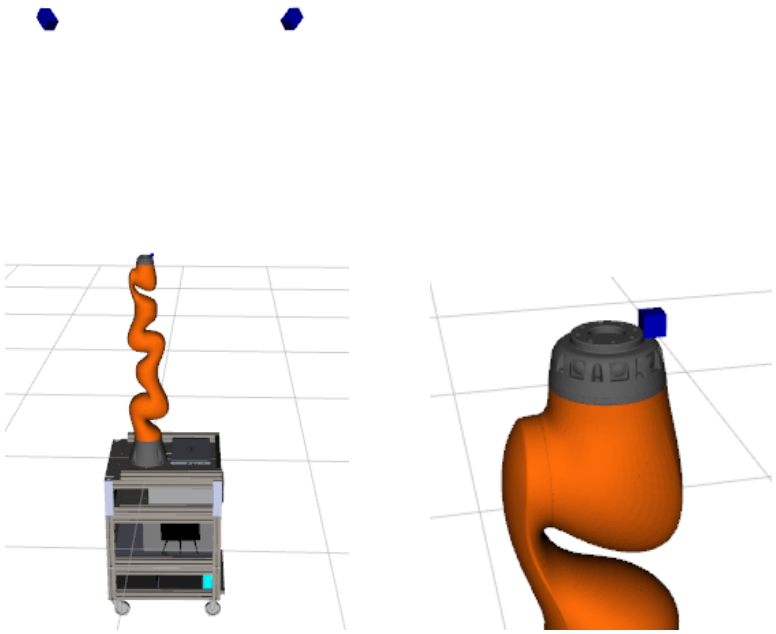


Figura 4.5: Extensão do LBR iiwa para a aplicação desse trabalho (inclusão de suporte e câmeras).

Uma simplificação foi realizada em relação a geometria de colisões implementada para o

suporte visando não perder desempenho da aplicação. O arquivo STL do suporte contempla uma riqueza de detalhes que podem sobrecarregar computacionalmente os cálculos de colisões. Portanto, optou-se por modelar a geometria de colisões do suporte como uma composição de paralelepípedos disjuntos como mostrado na figura 4.6.

Figura 4.6: Composição de paralelepípedos para a geometria de colisões do suporte.

### 4.1.4 Inclusão de câmeras

Para aprimorar as funcionalidade disponíveis nesse trabalho, foram adicionadas três câmeras ao sistema. Duas dessas câmeras foram projetadas para proporcionar uma visualização externa, enquanto a terceira foi especificamente instalada próxima ao efetuador do LBR iiwa para permitir uma visualização local do ponto de vista do robô. A inclusão dessas câmeras envolveu a atualização do arquivo XACRO definido na seção 4.1.3, que importa a estrutura do robô, o suporte utilizado e a adição de câmera vinculadas ao suporte do LBR iiwa. Para isso, cada câmera foi estabelecida os devidos elos e junções conforme descrito na seção 3.3.2. As câmeras externas foram vinculadas a base do LBR iiwa, enquanto o a câmera interna está vinculada ao centro do efetuador. Por fim, as câmeras foram modeladas como paralelepípedos de massa e inércia desprezíveis para evitar alterações na dinâmica do robô.

Para configuração das câmeras, foi adicionado como *plugin* a implementação de sensor de câmera pela biblioteca *"libgazebo_ros_camera.so"*. A biblioteca *"libgazebo_ros_camera.so"* é compatível com câmeras RGB (*Red Green Blue*), câmeras monocromáticas e câmeras infraver-

melhas. A biblioteca é projetada para se integrar com câmeras reais e fornecer funcionalidades de simulação.

Foi definido o tópico ROS *"/iiwa/camera_sensor"*, que servirá como canal para publicação das imagens capturadas pelas câmeras. A biblioteca *"libgazebo_ros_camera.so"* instância um nó ROS responsável pela publicação das mensagens do tipo *"sensor_msgs/Image"*.

A fim de visualizar as imagens transmitidas pelo tópico ROS *"/iiwa/camera_sensor"*, foi configurado uma visualização no RVIZ. Especificamente, o subpacote *"rviz"* foi implementado e configurado para permitir a visualização adequada das imagens das câmeras. Isso foi feito no arquivo *"mybot.rviz"*, que contém as configurações necessárias para executar o RVIZ. Além disso, um nó ROS chamado *"rviz"* foi implementado para executar as tarefas implementadas no arquivo *"mybot.rviz"*. As figuras 4.7 e 4.8 mostram as visualizações em tempo de execução das imagens das câmeras pelo RVIZ.

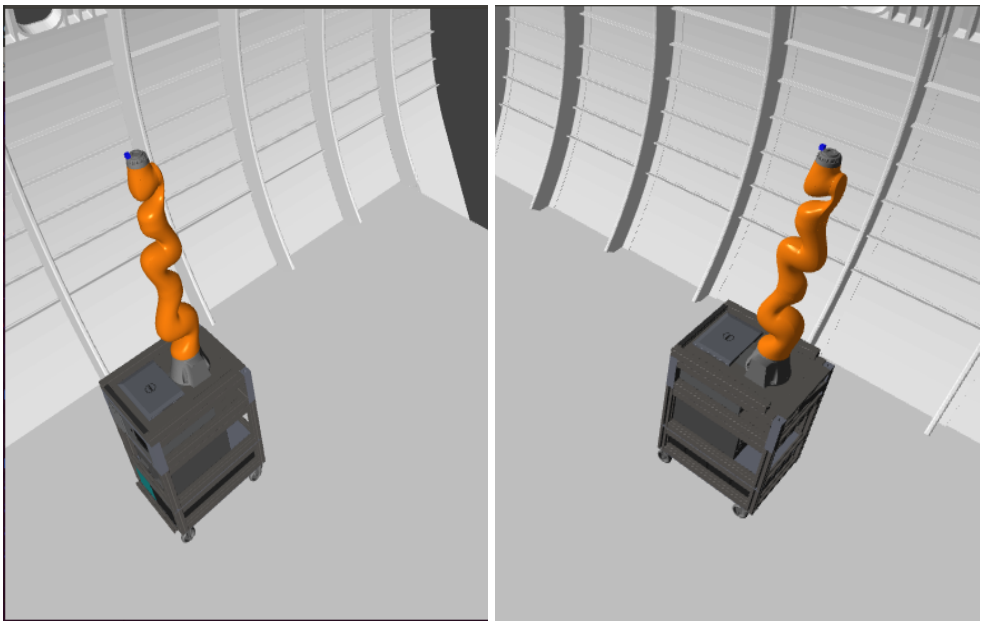


Figura 4.7: Visualização das câmera externas: esquerda e direita.

Na seção 4.2 pode ser conferido outros maneiras de visualizar as imagens publicadas no tópico *"/iiwa/camera_sensor"*.

### 4.1.5 Descrição da movimentação: *iiwa_airplane_manufacturing_moveit*

Nesse pacote estão contidos os arquivos de configuração para uso do LBR iiwa dentro do MoveIt para planejamento e execução de trajetória. Esse pacote pode ser usado para o planejamento

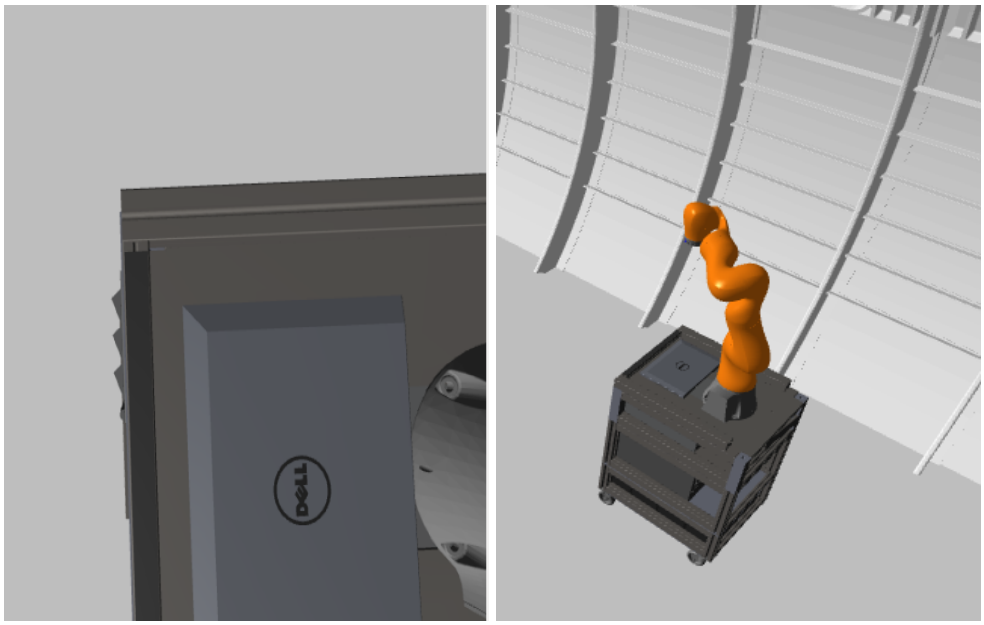


Figura 4.8: Visualização da câmera interna.

de trajetória no ambiente de simulação do Gazebo, mas também para o robô físico, alterando as propriedades do arquivo de inicialização do *"moveit_planning_execution_airplane_manufacturing.launch"*.

O MoveIt faz uso da ferramenta de visualização RVIZ do ROS como interface gráfica, como mostrado na figura 4.9.

O propósito desse projeto é a construção de uma plataforma que permita a simulação de movimentação, trajetória ou operação que pode ser definida no MoveIt de acordo com as necessidades do usuário. O funcionamento do MoveIt ocorre nas seguintes etapas: mover o LBR iiwa para uma nova posição por meio de mover e girar a esfera em sua extremidade (figura 4.10); clicar em *"Plan"* para calcular uma trajetória entre a posição atual e a posição selecionada; e clicar em *"Execute"* e observar o movimento no Gazebo (figura 4.11).

A localização do LBR iiwa pode ser proveniente da simulação ou diretamente do robô físico.

### 4.1.6 Interface para comunicação de coordenadas

Além da interface do MoveIt para o planejamento de trajetória, existe uma segunda interface de comunicação para o planejamento de trajetória com o objetivo de servir de interface para

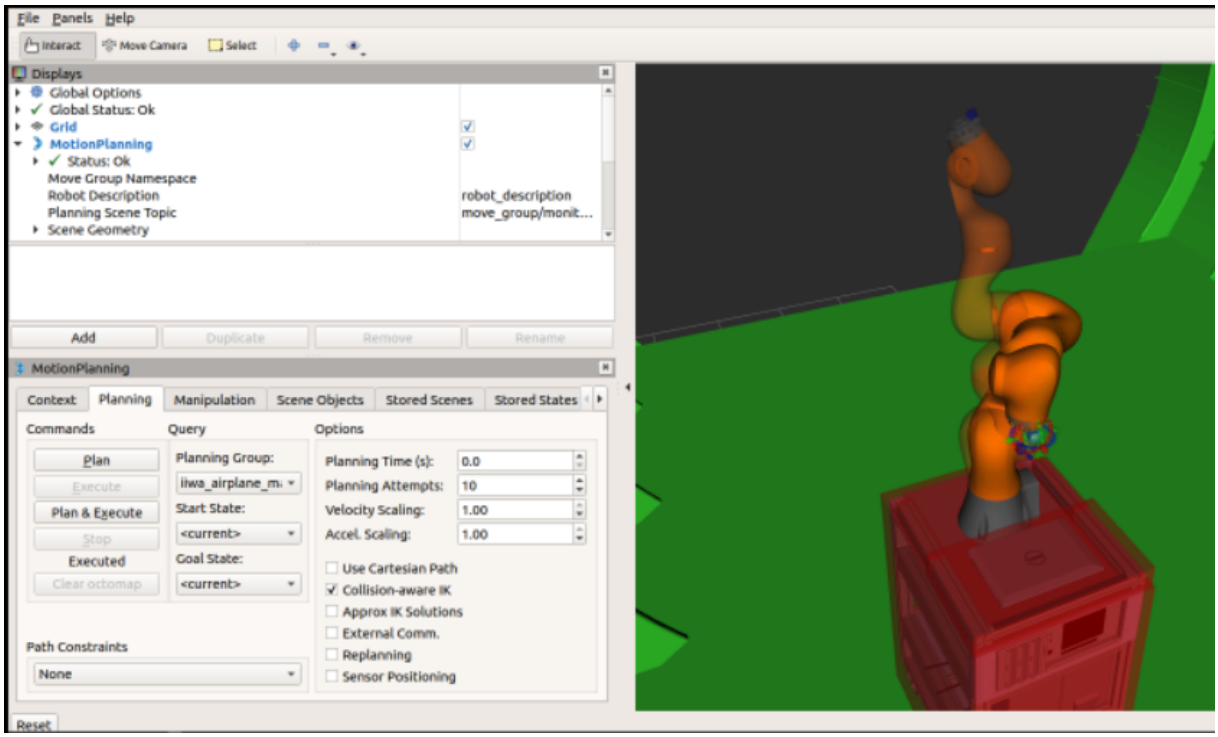



Figura 4.9: Interface gráfica do MoveIt proporcionada pelo RVIZ.

processos automatizados. É usual para programas de automatização de tarefas com o LBR iiwa a capacidade de comunicar coordenadas por meio de *software* como MATLAB e PyKUKA.

Para construção da segunda interface de comunicação, foi criado um nó ROS chamado *"airplane_manufacturing_moveit"* e um tópico ROS *"HeadCoordinates"*, ao qual o nó está inscrito.

Para cada mensagem contendo as coordenadas para a movimentação do braço robótico que é registrada no tópico ROS, o nó ROS executa um planejamento de trajetória. Após o cálculo da trajetória, o usuário está apto a autorizar a execução da movimentação. As figuras 4.12 e 4.13 mostram o fluxograma de processos desta tarefa. Vale ressaltar que o nó ROS utiliza as funções da biblioteca *"moveit_commander"* para implementar os cálculos de trajetória da ponta do robô sem colidir com os objetos da simulação e para executar as atividades de movimentação.

A interface para as atividades de automatização foi definida como o envio de mensagens do tipo *Twist* (3.4.1) para o tópico *"/HeadCoordinates"*.

Esse tipo mensagem é definida pelas posições espaciais cartesianas em metros (x, y e z) e pelas orientação descrita por meio de ângulos de Euler em radianos (*yaw*, *pitch* e *roll*) com o

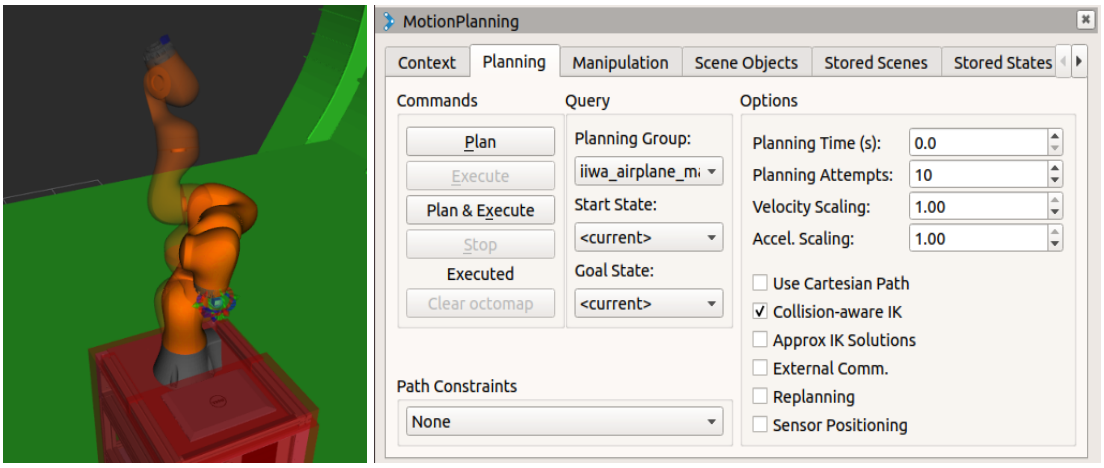



Figura 4.10: Planejamento de movimentação no MoveIt.

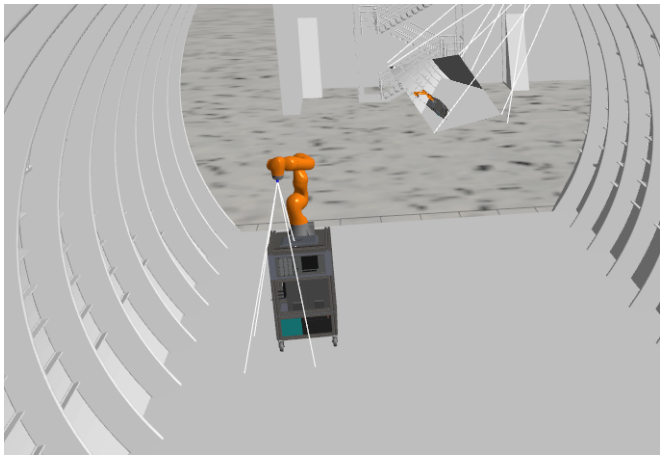


Figura 4.11: Posição final após movimentação executada.

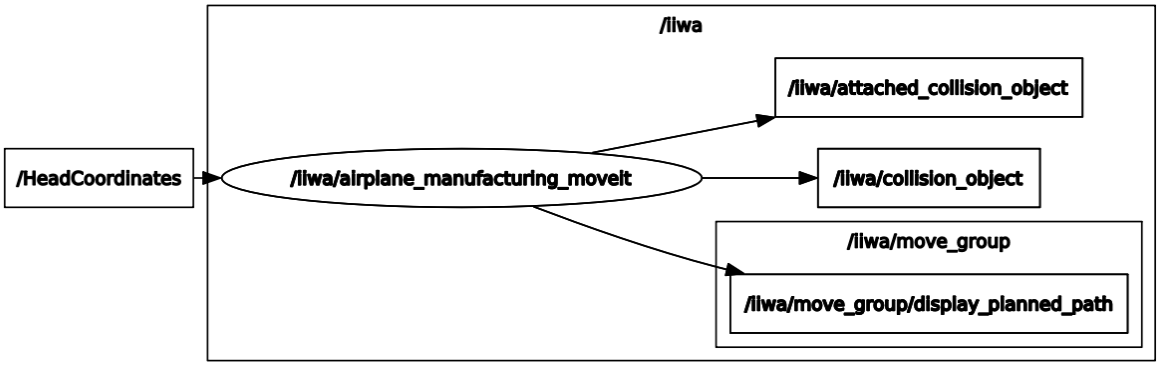



Figura 4.12: Nós e tópicos na interface de comunicação de coordenadas.

sistema de referência utilizado considera a base do robô como origem. A forma de realizar essa comunicação manual pode ser vista na seção 4.2.

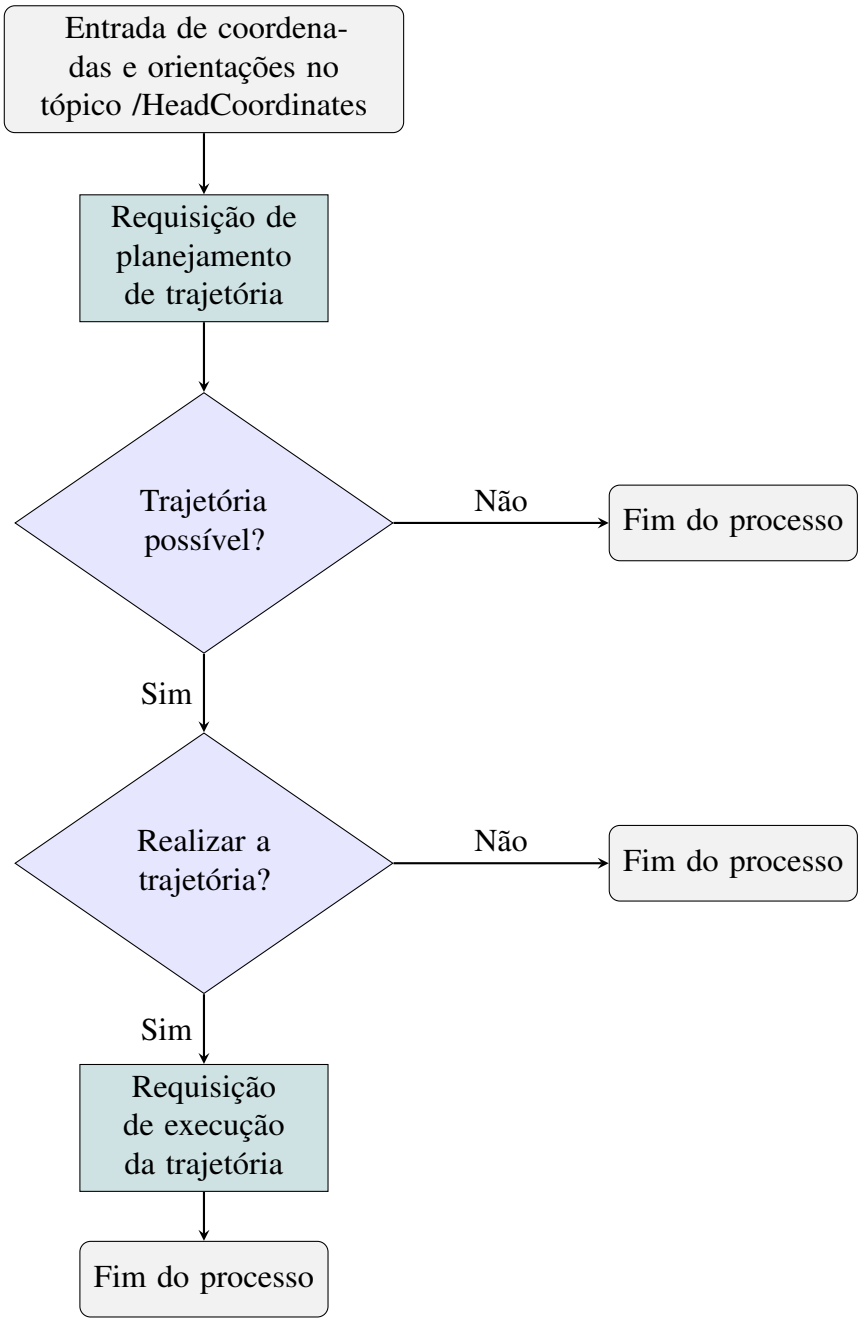



Figura 4.13: Fluxograma da utilização da interface de comunicação de coordenadas.

Após a publicação da mensagem, o nó converte a informação de orientação em *Quaternions*. Após a conversão, uma mensagem do tipo *"geometry_msgs/Pose.msg"* é enviada para as funções da biblioteca *"moveit_commander"*.

### 4.1.7 Simulação no Gazebo: *iiwa_gazebo_airplane_manufacturing*

O pacote *"iiwa_gazebo_airplane_manufacturing"* contém os arquivos que inicializam a simulação. O primeiro arquivo a ser chamado utilizando o comando *"roslaunch"* (3.4.6) é *"iiwa_gazebo_with_sunrise.launch"*. A partir desse arquivo, de forma encadeada, os demais arquivos são invocados, instanciando toda a estrutura de nós ROS, tópicos ROS, serviços ROS e ações ROS como evidenciado na figura 4.14.

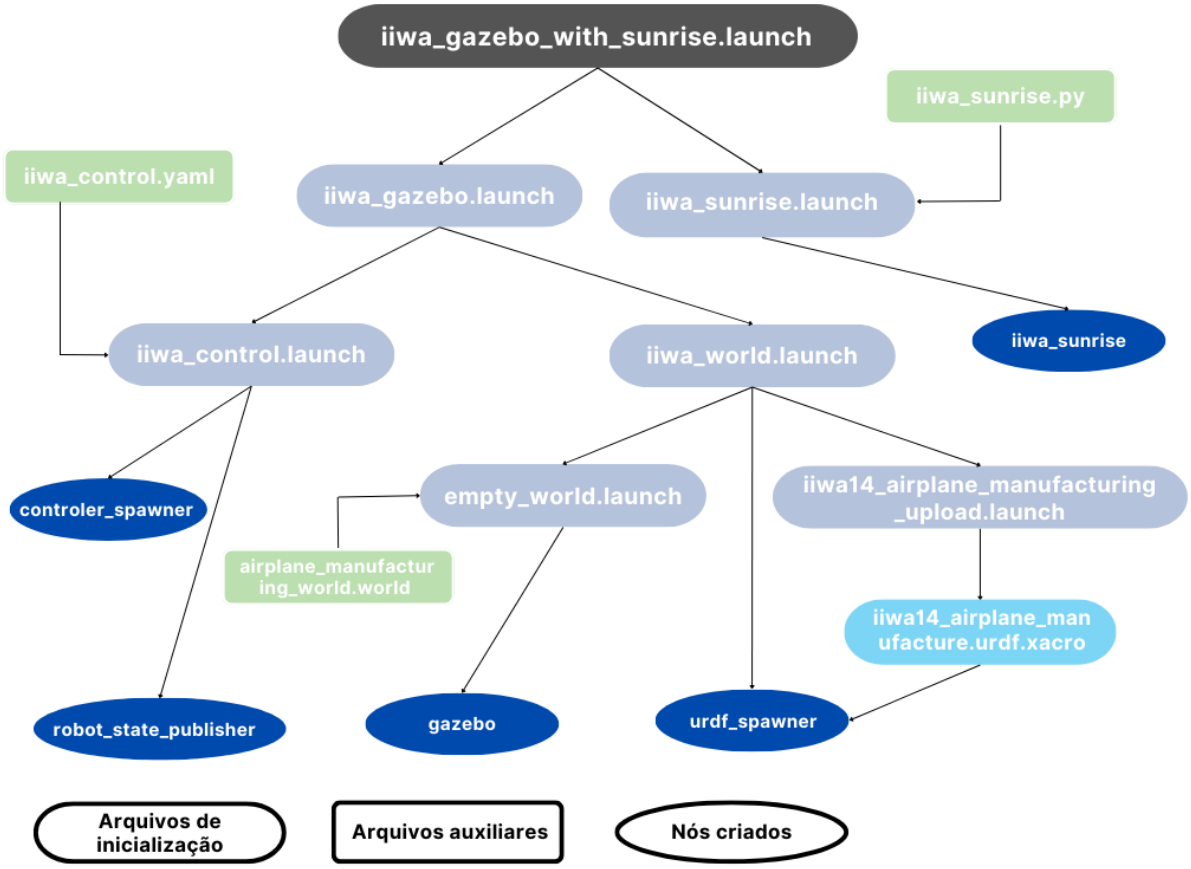



Figura 4.14: Fluxograma de inicialização da aplicação no ROS.

Os retângulos arredondados são arquivos *".launch"* que iniciam nós ROS e carregam arquivos de configuração. Os nós ROS são representados como formas ovais. Os retângulos representam arquivos de configuração usados pelos nós ou pelos arquivos *".launch"*.

Por exemplo, O arquivo *"airplane_manufacturing_world.world"* configura o ambiente e é usado pelo *"empty_world.launch"* ao iniciar o nó do Gazebo. O arquivo *"iiwa14_airplane_manufacturing.urdf.xacro"* é carregado pelo *"iiwa14_airplane_manufacturing_upload.launch"* no servidor de

parâmetros e suas informações são utilizadas pelo nó *"urdf_spawner"*, responsável por carregar o modelo do robô na simulação.

### 4.1.8 Inicialização do planejamento de movimentação no MoveIt

Analogamente a inicialização do MoveIt possui uma lógica de implementação. Na figura 4.15 é apresentado o fluxograma da inicialização do MoveIt na aplicação. O processo é iniciado pela chamado do arquivo arquivo *"airplane_manufacturing_moveit.launch"* utilizando o comando *"roslaunch"*(3.4.6) em uma segunda instância do terminal. Entretanto, a lógica de inicialização dos nós e tópicos, tal como a inclusão dos elementos no MoveIt é implementada em Python no arquivo *"airplane_manufacturing_moveit.py"*. A utilização do *script* para a construção dos nós é uma alternativa flexível a alterações com a inclusão de novos elementos e conta a bem documentada API do ROS para Python.

### 4.1.9 Troca de ferramenta: *gazebo_ros_link_attacher*

O pacote *"gazebo_ros_link_attacher"* é externo e foi incluído na simulação para realizar as etapas de fixação e desafixação de objetos.

Na proposta de aplicação desse trabalho, a tarefa a ser executada é a troca de ferramenta no efetuador do LBR iiwa. O fluxograma da figura 4.16 descreve o processo de fixação da ferramenta ao robô. Para isso, é necessário criar o nó ROS *"spawn_tool"* que deverá executar o chamado dos serviços de anexar ou desanexar a ferramenta. O nó ROS *"spawn_tool"* se inscreve ao tópico ROS *"/iiwa_tool"* que receberá a mensagem da ferramenta a ser vinculada ao robô, conforme descrito na seção 3.7. A instanciação do nó ROS *"spawn_tool"* e do tópico ROS *"/iiwa_tool"* ocorre por meio de um *script* em Python chamado *"spawn_tool.py"*.

Para requisitar-se um serviço, é necessário informar uma mensagem contendo quatro informações no formato de texto no terminal: o modelo e seu *link* que será o *parent*; e o modelo e seu *link* que será o *child*. Neste caso, o efetuador do robô será o *link parent* e a ferramenta será o *link child*. O efetuador do robô está associado ao modelo do LBR iiwa e o *link* da ferramenta está associado ao modelo da ferramenta.

Para ser possível a troca de ferramentas, é necessário que o modelo da ferramenta tenha sido previamente criado e incluído no *script* em Python *"spawn_tool.py"*. A inclusão de um modelo

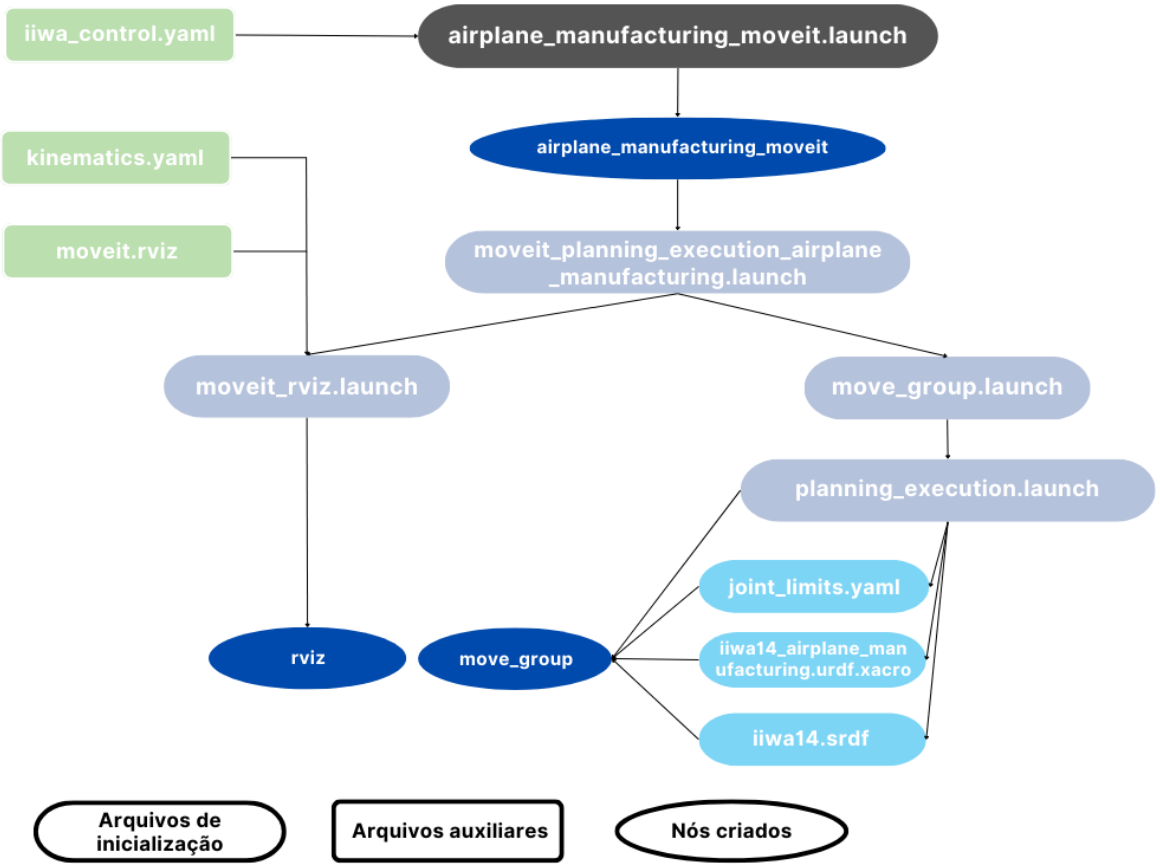



Figura 4.15: Fluxograma de inicialização do MoveIt.

de ferramenta pode ser realizado no formato SDF com descrito na seção 3.3.2 (nome, massa, posição do centro de massa e geometria) dentro do pacote *"airplane_manufacturing_setup"*.

Nesse trabalho foram incluídos os seguintes modelos de ferramenta:

- **Drill:** ferramenta de furação
- **TMS:** ferramenta de fresamento
- **Thread-tool:** ferramenta de rosqueamento

Ao publicar o texto *"thread_tool"* no tópico *"iiwa_tool"* (comando para troca de ferramenta consta na seção 4.2), qualquer ferramenta existente é removida da simulação e uma a ferramenta solicitada é incluída e anexada ao efetuador do LBR iiwa, no caso a ferramenta *Thread tool*, como demonstrado na figura 4.17.

### 4.1.10 Captura de imagens das câmeras: *iiwa_camera_savedImage*

Para atender à necessidade de captura e disponibilização das imagens das câmeras, foi decidido utilizar um *script* em Python. Essa escolha foi motivada pelo desejo de integração do conteúdo das imagens com outras bibliotecas, como aquelas voltadas para aprendizado de máquina.

No *script* Python, foi utilizado o OpenCV para converter as mensagens do tipo *"sensor_msgs/Image"* do ROS para a estrutura de dados que nativamente é utilizada no OpenCV. Em seguida, essas imagens convertidas são salvas na pasta *"iiwa_camera_savedImages"*, permitindo que sejam acessadas posteriormente para análise ou processamento.

Para realizar todas essas tarefas, um nó chamado *"read_images_ML_algorithms"* foi criado por meio do *script* Python. Esse nó é responsável pela conversão das mensagens de imagem e disponibilização das imagens para uso posterior de acordo com a frequência de aquisição estabelecida pelo sensor. A figura 4.18 ilustra a pasta das imagens aquisitadas.

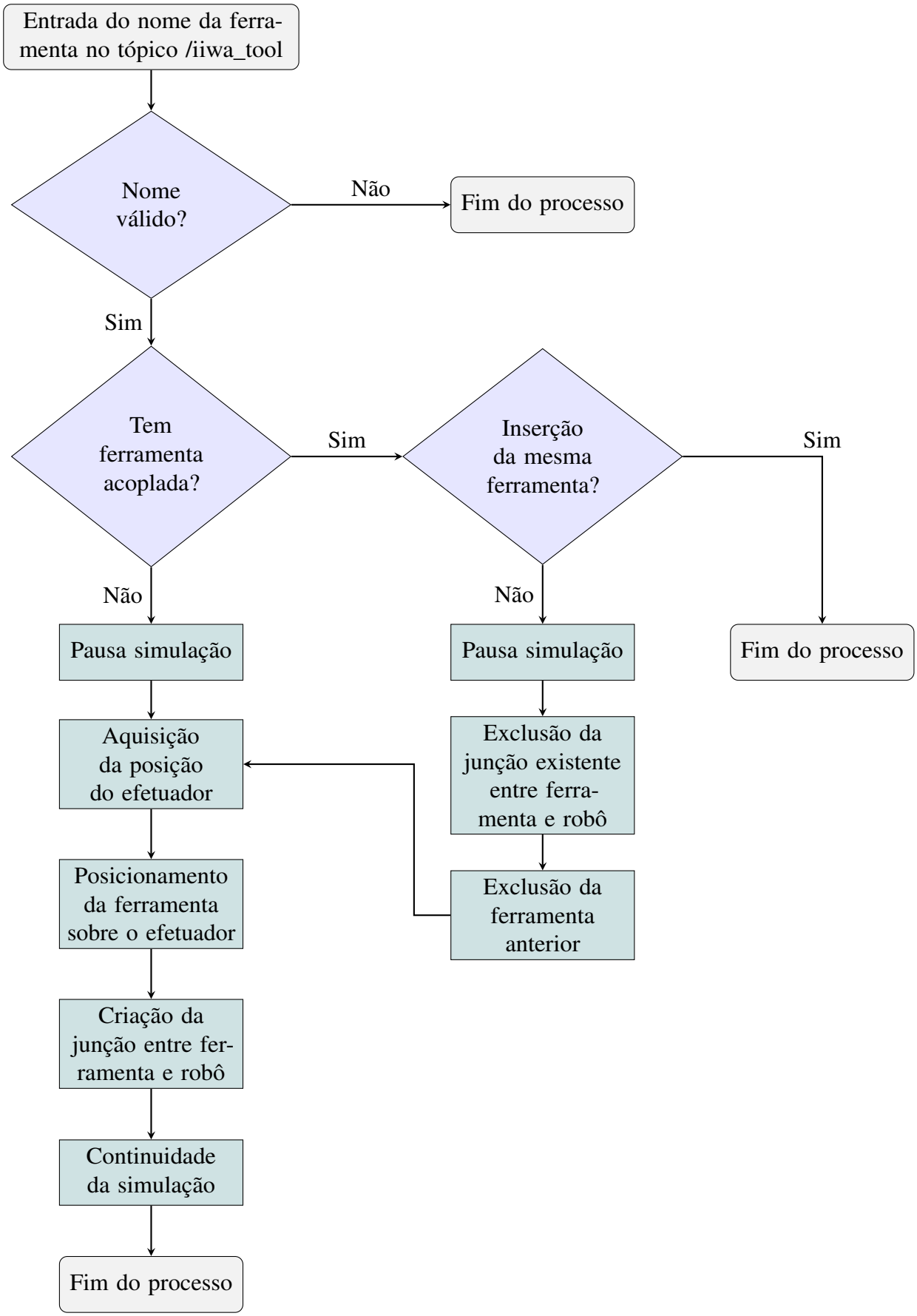



Figura 4.16: Fluxograma do processo de troca de ferramenta no Gazebo.

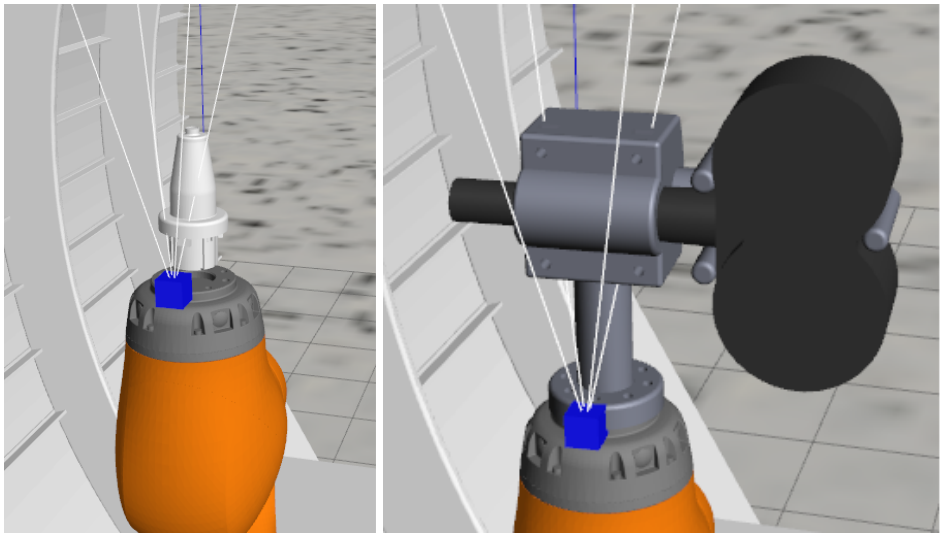


Figura 4.17: Resultado da inclusão do *Thread tool* e TMS no LBR iiwa.

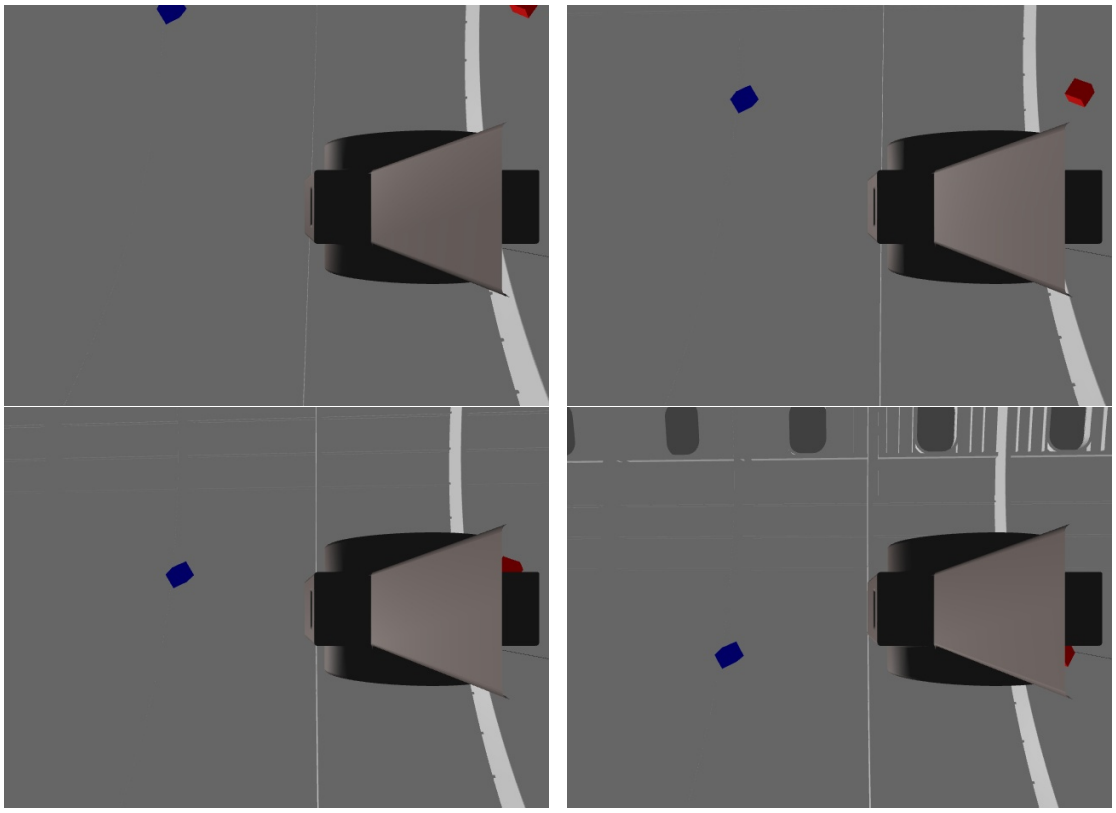


Figura 4.18: Imagens aquisitadas pela câmera interna do LBR iiwa durante uma movimentação.

## 4.2 Fluxo de execução da simulação

Para preparar a simulação, deve-se inicialmente possuir os arquivos organizados corretamente. Depois, deve-se executar a sequência de procedimentos mostrada no algoritmo 4.1. Os comandos devem ser executados em um terminal para instalar as dependências dos pacotes, compilar o *workspace* e fazer com que os códigos possam ser encontrados

```
cd iiwa_airplane_manufacturing_ws
rosdep install --from-paths src --ignore-src -r -y
sudo apt-get install ros-melodic-gazebo-ros-pkgs ros-melodic-gazebo-ros-
    control
sudo apt install ros-melodic-compressed-image-transport ros-melodic-
    camera-info-manager ros-melodic-diagnostic-updater
sudo apt-get install ros-$ROS_DISTRO-moveit-visual-tools
sudo apt install ros-melodic-moveit
catkin_make
source devel/setup.bash
```

Algoritmo 4.1: Primeiro conjunto de algoritmos.

Um terminal é necessário para inicializar a simulação e a cada novo comando deve-se abrir uma nova instância do terminal. Os comando a serem executados para realização de cada tarefa podem ser visto no algoritmo 4.2.

```
roslaunch iiwa_gazebo_airplane_manufacturing
iiwa_gazebo_with_sunrise.launch # run robot (launch iiwa and world)

roslaunch iiwa_gazebo_airplane_manufacturing
    airplane_manufacturing_moveit.launch # run moveIt (launch moveIt
    interface)

roslaunch iiwa_gazebo_airplane_manufacturing spawn_tool.launch # run
    spawn tool module (launch spawn tool code)

rostopic pub /iiwa_tool std_msgs/String "data: 'tool_name'" # spawn tool
     using a mensage in rostopic

rostopic pub /HeadCoordinates geometry_msgs/Twist -- '[linear.x, linear.
    y, linear.z]' '[angular.x, angular.y, angular.z]' # change the iiwa
    position and orientation to the specified ones
```

```
11
12 roslaunch iiwa_gazebo_airplane_manufacturing image_preprocessor.launch #
       run image preprocessor (reader for saving images from camera sensor
   for applying in machine learning applications)
13
14 rosrun image_view image_view image:="rostopic path" # run image viewer
   for seeing the images published in a rostopic
```

Algoritmo 4.2: Segundo conjunto de algoritmos.

# CAPÍTULO 5

# CONCLUSÕES E PERSPECTIVAS

Após o planejamento e execução das etapas previstas no projeto foi possível gerar um ambiente de simulação genérico para representar etapas no processo de manufatura na indústria aeronáutica. O modelo criado possui a característica de ser um *framework* para futuras aplicações, que assim como esse trabalho, tem por objetivo se valer do uso do LBR iiwa como robô principal.

O planejamento de trajetória foi configurado com sucesso pela utilização do MoveIt. E, por meio da utilização da biblioteca *"gazebo_ros_link_attacher"* foi possível realizar a fixação e desafixação de ferramentas em tempo de execução.

Além disso, vale ressaltar que foi possível construir um modelo que contempla o uso de sensoriamento por câmeras e abre o caminho para extensão de uma nova gama de aplicações através do uso desses dados.

Espera-se que esse trabalho sirva de suporte e ferramenta para facilitar, viabilizar e servir de base para a criação de novas aplicações do LBR iiwa na indústria aeronáutica. Com impacto a curto prazo, espera-se que esse trabalho sirva como suporte a aplicações do grupo de pesquisa de robótica da EESC.

O escopo do trabalho se limitou a configuração em *software* da simulação, entretanto as funcionalidades implementadas contemplam a integração com o robô físico. Portanto, os próximos passos para o projeto tem por objetivo acoplar o robô com a simulação, assim como incluir associar as câmeras físicas com as câmeras virtuais da simulação.

Em perspectiva de desenvolvimentos futuros, esse trabalho pode incluir a utilização de algoritmos de *machine learning* para auxiliar a automatização e tomada de decisão do LBR iiwa em processo associados a indústria aeronáutica, como inspeções visuais. Os mesmo algoritmos podem ser usados visando ao aumento da segurança de operação do robô para evitar colisões usando análises em tempo real. Finalmente, é possível vislumbrar associação da ferramenta desenvolvida com tecnologias de realidade virtual ou realidade aumentada. Dessa forma, a continuidade direta desse trabalho poderia ser a configuração do equipamento e a criação de um módulo de execução para aplicações de realidade virtual ou aumentada.

CAPÍTULO 6

# APÊNDICES

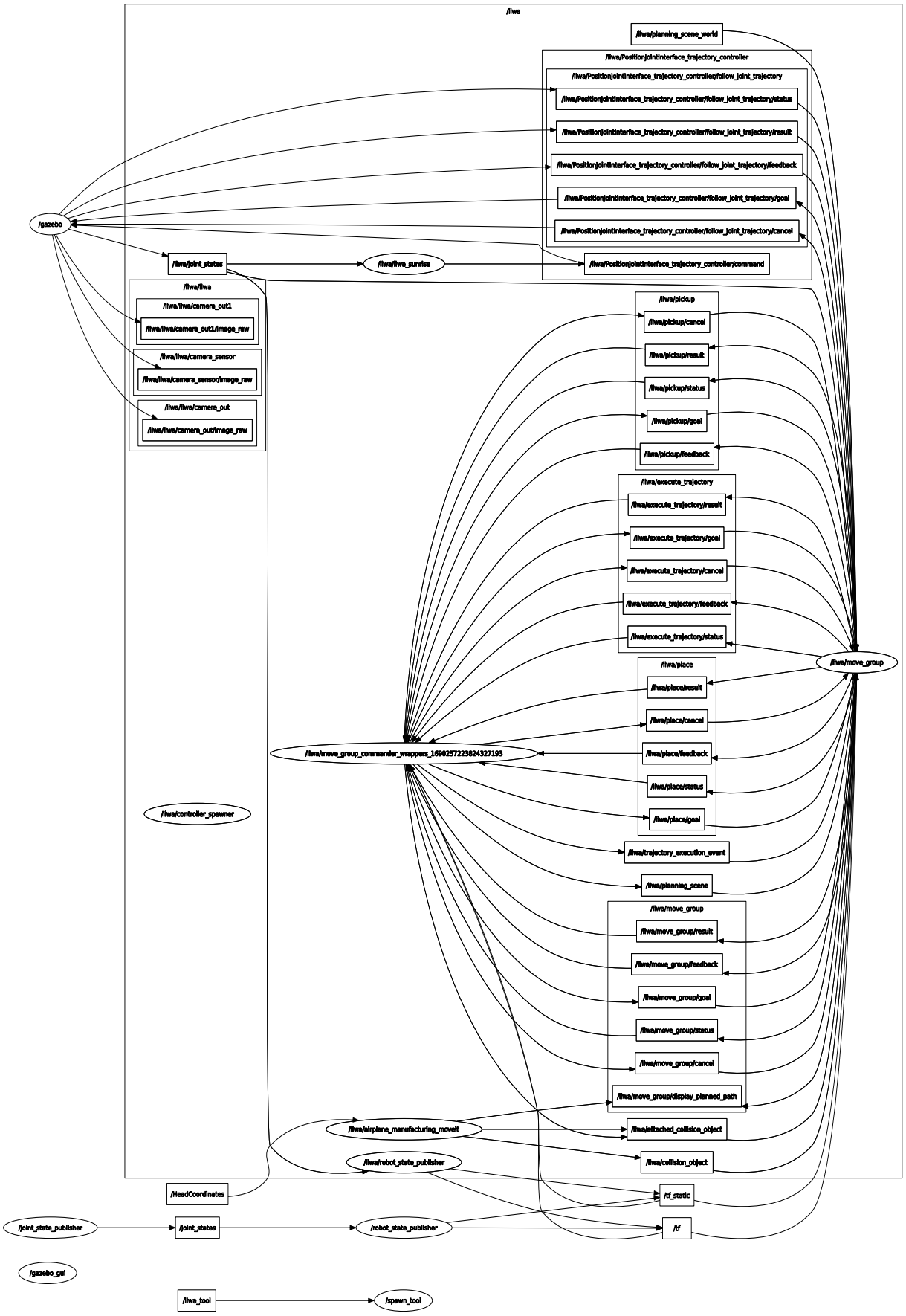
## 6.1 Apêndice A: GitHub

Foi utilizado GitHub visando a favorecer a extensibilidade e o compartilhamento do código para criação e implementação de novas *branchs* do projeto, possibilitando ainda se tornar um repositório de discussões para novas aplicações.

O GitHub tornou-se a fonte de acesso padrão aos códigos do projeto, bem como outros códigos abertos que contribuíram significativamente para o progresso desse trabalho.

Os arquivos podem ser clonados do repositório *"https://github.com/luisamartins/airplane-_manufacturing_stack"*.

## 6.2 Apêndice B: Conexões entre nós ROS e tópicos ROS da aplicação


/iiwa
/iiwa/planning_scene_world
/iiwa/PositionJointInterface_trajectory_controller
/iiwa/PositionJointInterface_trajectory_controller/follow_joint_trajectory
/iiwa/PositionJointInterface_trajectory_controller/follow_joint_trajectory/status
/iiwa/PositionJointInterface_trajectory_controller/follow_joint_trajectory/result
/iiwa/PositionJointInterface_trajectory_controller/follow_joint_trajectory/feedback
/iiwa/PositionJointInterface_trajectory_controller/follow_joint_trajectory/goal
/iiwa/PositionJointInterface_trajectory_controller/follow_joint_trajectory/cancel
/iiwa/PositionJointInterface_trajectory_controller/command
/gazebo
/iiwa/joint_states
/iiwa/iiwa_sunrise
/iiwa/iiwa
/iiwa/iiwa/camera_out1
/iiwa/iiwa/camera_out1/image_raw
/iiwa/iiwa/camera_sensor
/iiwa/iiwa/camera_sensor/image_raw
/iiwa/iiwa/camera_out
/iiwa/iiwa/camera_out/image_raw
/iiwa/pickup
/iiwa/pickup/cancel
/iiwa/pickup/result
/iiwa/pickup/status
/iiwa/pickup/goal
/iiwa/pickup/feedback
/iiwa/execute_trajectory
/iiwa/execute_trajectory/result
/iiwa/execute_trajectory/goal
/iiwa/execute_trajectory/cancel
/iiwa/execute_trajectory/feedback
/iiwa/execute_trajectory/status
/iiwa/move_group
/iiwa/place
/iiwa/place/result
/iiwa/place/cancel
/iiwa/place/feedback
/iiwa/place/status
/iiwa/place/goal
/iiwa/move_group_commander_wrappers_1690257223824327193
/iiwa/controller_spawner
/iiwa/trajectory_execution_event
/iiwa/planning_scene
/iiwa/move_group
/iiwa/move_group/result
/iiwa/move_group/feedback
/iiwa/move_group/goal
/iiwa/move_group/status
/iiwa/move_group/cancel
/iiwa/move_group/display_planned_path
/iiwa/airplane_manufacturing_moveit
/iiwa/attached_collision_object
/iiwa/collision_object
/iiwa/robot_state_publisher
/HeadCoordinates
/tf_static
/joint_state_publisher
/joint_states
/robot_state_publisher
/tf
/gazebo_gui
/iiwa_tool
/spawn_tool

# BIBLIOGRAFIA


Bender, J., Weber, M., Kolb, A., and Wacker, M. (2017). A framework for data-driven visual simulation of robot-controlled environments. *Journal of Simulation*, 11(1):31–42.

Bezzo, N., Faccio, M., Gasparri, A., and Meneghello, R. (2019). Autonomous intelligent vehicles in aeronautic manufacturing: A review. *Robotics and Computer-Integrated Manufacturing*, 55:23–37.

Boer, L. (2022). Simulação de robô neurocirúrgico. *EESC- Escola de Engenharia de São Carlos*.

Cacace, J., Siciliano, B., and Villani, L. (2017). Real-time control of robot manipulators in a 3d virtual environment. *Robotics and Autonomous Systems*, 96:57–68.

Chacón, L. et al. (2020). Robotics and autonomous systems for aircraft inspection: A review. *Aerospace*, 7(7):1–19.

Chentanez, N., Müller, M., Kim, T.-Y., and Fatahalian, K. (2012). Real-time simulation of large elastoplastic deformation with shape matching. *ACM Transactions on Graphics (TOG)*, 31(4):42.

Corke, P., Buchli, J., Hamel, T., and Mahony, R. (2018). *Design and control of autonomous flying vehicles*. Springer International Publishing.

Craig, J. J. (2005). *Introduction to Robotics: Mechanics and Control*. Pearson Prentice Hall, 3rd edition.

Falco, P., Finzi, A., Siciliano, B., and Siciliano, G. (2019). Simultaneous planning and control for redundant manipulators based on multiple representations. *Autonomous Robots*, 43(3):703–720.

Fassi, I., Ferrari, S., Colledani, M., and Sardini, E. (2018). A collaborative robotic solution for wing structure assembly in the aerospace industry. *Robotics and Computer-Integrated Manufacturing*, 51:99–111.

Fernandez, M., Varela, A., del Pino, J., and Haber Guerra, R. (2017). Inspection and repair of aerospace components by autonomous mobile robots. *Robotics and Autonomous Systems*, 88:215–228.

Garcia, R. et al. (2019). Robotic automation in airports: A review. *Journal of Airport Management*, 13(3):328–342.

Haidu, A., Kornatowski, P., and Beetz, M. (2018). Integrated simulation of robots and humans in task and motion planning. *The International Journal of Robotics Research*, 37(9):1066–1085.

Hennersperger, C., Fuerst, B., Virga, S., Zettinig, O., Frisch, B., Neff, T., and Navab, N. (2016). Towards mri-based autonomous robotic us acquisitions: A first feasibility study. *IEEE Transactions on Medical Imaging*, PP.

Huang, H., Liu, Y., and Wu, Q. (2020). A survey on robot simulation methods. *Frontiers in Robotics and AI*, 7:109.

Hwangbo, J., Lee, J., and Kanade, T. (2017). Control of a quadrotor with reinforcement learning. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3726–3733.

Kim, H., Kim, C., and Oh, J. (2019). Simulation-based real-time control of robotic manipulators using open-source libraries. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3906–3912.

Ko, S., Lee, D., Yoon, S., and Jung, J. (2019). Gazebo-based robotic simulation environment for ros-based multi-robot system. In *Proceedings of the 8th International Conference on Robot Intelligence Technology and Applications (RiTA)*, pages 132–137.

KUKA Robotics (2023a). *KUKA LBR iiwa 14 User Manual*. KUKA Robotics Corporation, Augsburg, Germany.

KUKA Robotics (2023b). *KUKA Sunrise Workbench API Reference*. KUKA Robotics Corporation, Augsburg, Germany.

Lahr, G. J. L. (2017). Síntese de movimentos em tempo real para tarefas dinâmicas de montagem de precisão utilizando robôs. *EESC- Escola de Engenharia de São Carlos*.

Lu, Y., Zhou, L., and Wang, Z. (2017). The digital twin technology in the application of manufacturing simulation. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 1399–1403. IEEE.

McCall, R. and O'Hare, G. (2017). A review of virtual reality technologies in training. *Simulation & Gaming*, 48(3):308–327.

Open Robotics (2023a). Gazebo ROS Link Attacher. `https://github.com/pal-robotics/gazebo_ros_link_attacher`. Acesso em: 17 de Julho de 2023.

Open Robotics (2023b). ROS Documentation. `http://wiki.ros.org/`. Acesso em: 17 de Julho de 2023.

Ortega, F. R., Chang, C., Fuentes, D. A., and Lindeman, R. W. (2016). Augmented reality for robotic teleoperation with a depth camera. *Journal of Field Robotics*, 33(8):1103–1119.

Peng, Y., Tan, S., and Zhang, Y. (2018). A swarm navigation algorithm based on improved potential field for multi-robots in gazebo simulation environment. In *Proceedings of the 9th International Conference on Intelligent Control and Information Processing (ICICIP)*, pages 375–379.

Ribeiro, A., Lima, P., and Correia, L. (2019). Simulation framework for mobile robots with ros integration. *Robotics and Autonomous Systems*, 120:1–11.

Ross, I. and Brabazon, P. (2009). An analysis of the impact of robotic technology in aerospace manufacturing. *Robotics and Computer-Integrated Manufacturing*, 25(4-5):693–699.

Santos, A. et al. (2021). Robotic applications in aircraft manufacturing: A comprehensive review. *Journal of Aerospace Engineering*, 34(2):1–15.

Siciliano, B. and Khatib, O., editors (2016). *Springer Handbook of Robotics*. Springer, 2nd edition.

Sisbot, E. A., Knepper, R. A., and Siddhartha, S. (2010). Safe and dependable physical human-robot interaction in anthropic domains: State of the art and challenges. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 25–30.

Sun, Y., Luo, M., Li, H., and Yu, L. (2021). An autonomous intelligent robotic system for aircraft assembly quality inspection. *Robotics and Computer-Integrated Manufacturing*, 67:102070.

Taele, P., Albu-Schäffer, A., and Hirzinger, G. (2016). Interactive simulation of a human-arm-sized robotic manipulator with a novel two-way coupling approach. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4583–4589.

Tao, F., Qi, Q., Liu, A., Kusiak, A., and Zhou, L. (2019). Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0: correlation and comparison. *Journal of Manufacturing Science and Engineering*, 141(6):061004.

Tardioli, D., Bruzzone, L., and Muradore, R. (2019). A feasibility study on the application of collaborative robots for aircraft assembly. In *Procedia CIRP*, volume 79, pages 180–185.

Vargas, J. E. L., Bergamasco, L. A. V., and Pedroso, M. M. M. (2020). A robotic system for non-destructive testing in aeronautical applications. *Journal of Aerospace Technology and Management*, 12:e12199.

Vose, G. and De Silva, C. (2020). *Robotics: Principles and Practice*. Cambridge University Press.

Wenzel, K., Dornhege, C., and Stachniss, C. (2017). Ros-based indoor and outdoor navigation of micro uavs. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1954–1961.

Xu, Y., Zhang, X., Ma, C., Liu, X., and Yang, Y. (2019). Collaborative robots in assembly automation: A review. *Robotics and Computer-Integrated Manufacturing*, 58:200–213.

Zhang, Z., Li, J., Li, S., and Wang, L. (2021). Simulation and optimization of robotic drilling process for aerospace manufacturing. *Journal of Manufacturing Systems*, 58:190–200.