Marina Oliveira Batista

Controle orientado a eventos de uma fábrica inteligente com base em realimentação visual e uma estrutura de programação distribuída

Campina Grande, Paraíba

Novembro de 2024

Marina Oliveira Batista

Controle orientado a eventos de uma fábrica inteligente com base em realimentação visual e uma estrutura de programação distribuída

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Departamento de Engenharia Elétrica

Orientador: Antonio Marcus Nogueira Lima, Dr.

Campina Grande, Paraíba

Novembro de 2024

Marina Oliveira Batista

Controle orientado a eventos de uma fábrica inteligente com base em realimentação visual e uma estrutura de programação distribuída

Trabalho de Conclusão de Curso submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, Campus Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Aprovado em: _____ / _____ / _____

Antonio Marcus Nogueira Lima, Dr.
Orientador

Kyller Costa Gorgônio, Ph.D
Convidado

Campina Grande, Paraíba
Novembro de 2024

Dedico este trabalho aos meus pais.

Agradecimentos

Agradeço primeiramente a Deus pela força e sabedoria concedidas ao longo dessa caminhada. Agradeço aos meus pais, que, em sua simplicidade, sempre apoiaram os meus estudos – este trabalho é por vocês.

Sou grata aos meus colegas do laboratório Embedded, especialmente Sabrina e Dhara, pela colaboração diária e pelas trocas valiosas no desenvolvimento deste trabalho, e também a Felipe, por todas as "consultorias grátis", Melissa e Karen, que compartilharam o mesmo ambiente de trabalho, tornando essa jornada mais leve. Agradeço a Asley pelo apoio, compreensão e incentivo ao longo deste período desafiador, sempre estando ao meu lado nos momentos mais importantes. Às minhas amigas Laura, Ianca, Emily, Beatriz e Sabrina, deixo minha sincera gratidão por sempre acreditarem em mim e na minha capacidade.

Também agradeço à Camila, que, mesmo a quilômetros de distância, sempre esteve presente como um ombro amigo, conforto e força nos momentos mais importantes que passei.

Agradeço também ao meu orientador, Professor Antonio Marcus, cuja orientação e ensinamentos, ao longo dos anos, foram fundamentais para meu crescimento acadêmico e pessoal. Seu apoio e paciência foram essenciais para que este trabalho se realizasse.

Aos funcionários e professores do Departamento de Engenharia Elétrica, expresso minha gratidão pelo trabalho e comprometimento, que foram essenciais para o meu desenvolvimento ao longo da graduação. Agradeço também aos meus colegas de turma, que, de maneira direta ou indireta, compartilharam essa jornada comigo e contribuíram para meu aprendizado e crescimento.

Cada um de vocês, de alguma forma, foi parte essencial dessa conquista.

"Confia no Senhor de todo o teu coração e não te apoies no teu próprio entendimento. Reconhece-o em todos os teus caminhos, e ele endireitará as tuas veredas."

Provérbios 3:5-6

Resumo

A transformação digital trazida pela Indústria 4.0 estabelece uma nova configuração industrial, na qual a automação inteligente esta presente. Nesse contexto, as fábricas inteligentes surgem como ambientes de produção conectados, autônomos e adaptativos, capazes de ajustar processos em tempo real para responder às condições dinâmicas de fabricação. Neste trabalho, é apresentado o desenvolvimento de um sistema de controle orientado a eventos para uma fábrica inteligente, com base em realimentação visual e uma estrutura de programação distribuída com o *Robot Operating System* (ROS). O sistema foi projetado para capturar e processar dados visuais, utilizando uma câmera Kinect e marcadores fiduciais ArUco para validação, visando proporcionar uma automação adaptativa e eficiente no laboratório *Smart Factory* da Universidade Federal de Campina Grande (UFCG), que dispõe de dispositivos capazes de replicar atividades de uma fábrica inteligente em um ambiente controlado. A arquitetura implementada inclui um modelo baseado em eventos para o controle de dispositivos de aquisição de imagem, além da integração de uma árvore de comportamento para gerenciar as operações da câmera e monitorar a posição de marcadores no ambiente. A simulação e os testes realizados no *Smart Factory* confirmaram a capacidade do sistema de responder a eventos visuais, promovendo a automação flexível necessária no contexto da Indústria 4.0.

Palavras-chave: indústria 4.0; fábrica inteligente; realimentação visual; controle orientado a eventos; ROS; árvores de comportamento.

Abstract

The digital transformation driven by Industry 4.0 establishes a new industrial configuration, characterized by the presence of intelligent automation. In this context, smart factories emerge as connected, autonomous, and adaptive production environments capable of adjusting processes in real time to respond to dynamic manufacturing conditions. This work presents the development of an event-driven control system for a smart factory, utilizing visual feedback and a distributed programming structure with the Robot Operating System (ROS). The system was designed to capture and process visual data using a Kinect camera and ArUco fiducial markers for validation, aiming to provide adaptive and efficient automation within the *Smart Factory* laboratory at the Federal University of Campina Grande (UFCG), which is equipped to replicate smart factory activities in a controlled environment. The implemented architecture includes an event-based model for controlling image acquisition devices, as well as the integration of a behavior tree to manage camera operations and monitor marker positions within the environment. Simulation and testing conducted at the *Smart Factory* confirmed the system's ability to respond to visual events, promoting the flexible automation required in the context of Industry 4.0.

Keywords: industry 4.0; smart factory; visual feedback; event-eriven eontrol; ROS; behavior tree.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IoT	*Internet of Things*
CPS	*Cyber-Physical Systems*
ROS	*Robot Operating System*
UR10	*Universal Robots*
SED	*Sistema a Eventos Discretos*
RGB	*Red, Green, Blue*
RGB-D	*Red, Green, Blue - Depth*
RVIZ	*ROS Visualization*
UR10	*Universal Robots 10*

Sumário

1 Introdução

A Indústria 4.0 representa uma transformação significativa na forma como os ambientes industriais operam, integrando tecnologias como a Internet das Coisas *(Internet of Things - IoT)* e os Sistemas Ciberfísicos *(Cyber-Physical Systems - CPS)*, com o objetivo de otimizar processos e aumentar a eficiência de produção (CHEN et al., 2017). No contexto das fábricas inteligentes, essa integração requer uma automação avançada, adaptativa e distribuída, na qual dispositivos e robôs colaboram em tempo real para gerenciar operações de maneira autônoma (ZEMLA et al., 2023).

Um dos pilares dessa automação é o uso de sistemas de visão, que permitem aos robôs ajustar suas operações com base em dados visuais obtidos por câmeras e sensores de profundidade. Esses dispositivos fornecem informações em tempo real sobre a posição e orientação dos objetos no ambiente, viabilizando o controle preciso e dinâmico das operações. Por meio da análise de dados visuais, os sistemas podem localizar, rastrear e manipular objetos de forma coordenada, promovendo uma adaptação constante às condições do ambiente (CORKE, 2023).

Para coordenar essa variedade de dispositivos, o uso do *Robot Operating System* (ROS) oferece uma arquitetura de programação distribuída, na qual cada componente do sistema industrial – desde sensores até robôs – opera de forma integrada e modular (SIMONIč et al., 2021). O ROS possibilita a troca contínua de dados e o gerenciamento de eventos, facilitando uma resposta ágil e coordenada às mudanças no ambiente fabril. Neste trabalho, proponho um sistema de controle orientado a eventos que utiliza realimentação visual e ROS para promover uma automação flexível e eficiente, aplicado ao laboratório *Smart Factory* da Universidade Federal de Campina Grande, onde a integração de robôs e sensores em um ambiente simulado reproduz o contexto de uma fábrica inteligente.

1.1 Justificativa

A rápida evolução dos processos produtivos e a demanda por ambientes industriais adaptativos e eficientes destacam a importância de tecnologias que promovam uma automação inteligente. A realimentação visual, ao permitir que sistemas captem e analisem dados visuais em tempo real, contribui para uma tomada de decisão mais informada e ágil, favorecendo a flexibilidade das operações em uma fábrica inteligente (DOE; SMITH; PEREZ, 2020). Em particular, o uso de câmeras e sensores de profundidade possibilita uma interação contínua com o ambiente, facilitando o controle e o monitoramento dos processos de produção de forma autônoma e responsiva (LIAO et al., 2017).

Além disso, a escolha do ROS como plataforma de integração e controle oferece uma estrutura modular e distribuída que facilita o gerenciamento e a adaptação de cada dispositivo envolvido no sistema. A arquitetura do ROS permite que cada elemento, como sensores e robôs, responda a eventos em tempo real, promovendo uma automação orientada a eventos, tornando o sistema mais adaptável às mudanças do ambiente (SIMONIč et al., 2021). Assim, a combinação de sistemas de visão, uma estrutura distribuída e uma abordagem orientada a eventos visa criar uma automação eficiente, capaz de atender às demandas de flexibilidade e escalabilidade das fábricas inteligentes no contexto da Indústria 4.0 (RAHMAN; ALAM; UDDIN, 2023).

1.2 Objetivo geral

Neste Trabalho de Conclusão de Curso, o objetivo geral é desenvolver um sistema de controle orientado a eventos, utilizando realimentação visual e programação distribuída com ROS, para a automação de uma fábrica inteligente no contexto da Indústria 4.0. Tem-se como objetivos específicos:

- implementar um sistema de captura de dados visuais do ambiente, utilizando uma câmera para fornecer informações em tempo real para a tomada de decisões no sistema.

- desenvolver e integrar um modelo de controle orientado a eventos para dispositivos de aquisição de imagem, permitindo a operação independente e autônoma da câmera com base em eventos específicos;
- modelar e simular o ambiente da fábrica inteligente no simulador Gazebo, replicando as condições físicas e operacionais do laboratório *Smart Factory*, para testar o sistema;
- configurar e validar o modelo baseado em eventos da câmera em uma árvore de comportamento;
- realizar testes experimentais para observar e documentar o comportamento do sistema de controle orientado a eventos com realimentação visual no laboratório *Smart Factory*;
- deixar uma contribuição para o laboratório *Smart Factory*, através da implementação do sistema de visão, que possa ser utilizado em futuras pesquisas e experimentos no ambiente de simulação.

1.3 Organização do documento

Este documento está organizado em seis capítulos. No Capítulo 2, é apresentada uma revisão teórica sobre os temas fundamentais para o desenvolvimento do projeto. No Capítulo 3, é descrita a metodologia, detalhando o desenvolvimento do sistema no ambiente *Smart Factory*, a simulação no Gazebo e a integração dos dispositivos no ROS. O Capítulo 4 trata da implementação do controle orientado a eventos com base em sistemas de visão. Em seguida, é apresentada no Capítulo 5 os resultados obtidos a partir dos experimentos realizados. Por fim, o Capítulo 6 conclui o trabalho e sugere possíveis direções para pesquisas futuras.

2 Fundamentação Teórica

Este capítulo fornece a base teórica necessária para o entendimento dos conceitos e tecnologias utilizados no desenvolvimento deste trabalho.

2.1 Indústria 4.0

A Indústria 4.0 representa uma evolução nos sistemas de produção, convertendo estruturas tradicionais em ambientes digitalmente interconectados e adaptáveis. Essa revolução incorpora tecnologias digitais avançadas para modernizar práticas industriais e transformar os processos de fabricação e distribuição, promovendo uma produção mais flexível e responsiva. De acordo com (CHEN et al., 2017), a Indústria 4.0 se diferencia das revoluções industriais anteriores — mecanização, eletrificação e automação digital — ao possibilitar uma integração entre sistemas físicos e cibernéticos, com foco em uma produção adaptável e personalizada, aliada à eficiência e sustentabilidade.

Entre essas tecnologias, a IoT permite conectar dispositivos e sensores, criando uma rede de comunicação que monitora continuamente os processos industriais. Essa interconexão possibilita a coleta de dados em tempo real, proporcionando uma visão integrada das operações e facilitando a automação e otimização dos processos (HANSON; SMITH, 2021).

Além disso, os *CPS* promovem a integração entre os ambientes físico e digital, permitindo o monitoramento e controle em tempo real. Os *CPS* combinam sensores e atuadores que capturam informações do ambiente físico e, em resposta, ajustam parâmetros no sistema digital conforme as variações detectadas. Essa estrutura viabiliza uma produção flexível e personalizada, o que permite às fábricas inteligentes uma adaptação dinâmica a novas demandas e condições operacionais (ACKERMAN, 2016).

Por fim, a IA e o *Big Data* também desempenham papéis de relevância na Indústria 4.0. A IA permite que sistemas robóticos e autonômos processem grandes

volumes de dados e ajustem suas operações com base em aprendizado, otimizando processos de produção. Com o uso de algoritmos de aprendizado de máquina, é possível realizar análises preditivas e detectar padrões em tempo real, o que ajuda a antecipar demandas e ajustar a produção conforme necessário, o que segundo (RAHMAN et al., 2022) reduz o tempo de inatividade e promove um fluxo de produção contínuo.

2.1.1 Fábricas inteligentes

O conceito de fábricas inteligentes surge como uma evolução do processo de automação iniciado ao longo das primeiras revoluções industriais, onde máquinas foram gradualmente integradas com sistemas de controle. No entanto, com o avanço da Indústria 4.0, as fábricas inteligentes apresentam uma evolução da automação tradicional ao integrar sistemas físicos e digitais, oferecendo um ambiente de produção interconectado, autônomo e adaptativo (BENOTSMANE; KOVáCS; DUDáS, 2019). Essas fábricas utilizam tecnologias digitais para permitir uma operação descentralizada, onde cada máquina pode atuar de maneira autônoma e em coordenação com outras, ajustando-se em tempo real às demandas da produção.

As fábricas inteligentes baseiam-se em uma arquitetura distribuída que incorpora dispositivos e sensores habilitados por IoT, CPS, IA e *Big Data*, que juntos possibilitam uma produção ágil e responsiva. Segundo (ZEMLA et al., 2023), esse tipo de infraestrutura torna possível que as máquinas e sistemas se comuniquem continuamente, trocando dados sobre o desempenho de processos e o estado das operações, o que viabiliza um nível de integração e eficiência sem precedentes.

Além da integração em tempo real, outro aspecto central das fábricas inteligentes é a capacidade de aprendizado e adaptação. A IA e o *Big Data* desempenham papéis críticos, pois permitem que os sistemas analisem grandes volumes de dados, identifiquem padrões e realizem previsões sobre falhas e necessidades de manutenção, promovendo uma manutenção preditiva e um controle de qualidade avançado (LIAO et al., 2017). Conforme observado por (JASIńSKI; IWANIEC; TADEUSIEWICZ, 2020), essa análise preditiva aumenta a eficiência ao minimizar tempos de inatividade e ao ajustar a produção de acordo com variáveis internas e externas.

Outro ponto que diferencia as fábricas inteligentes das configurações industriais tradicionais é a capacidade de adaptação rápida a novos produtos e variações de produção. Utilizando CPS, essas fábricas conseguem modificar parâmetros operacionais e realocar recursos conforme necessário, possibilitando uma produção personalizada em larga escala. Essa flexibilidade estrutural permite a adaptação a um mercado volátil e exige uma coordenação de componentes digitais e físicos, com aplicações que vão desde o ajuste de processos até a reconfiguração de linhas de montagem (HANSON; SMITH, 2021).

Além dos avanços técnicos, a implementação de fábricas inteligentes também representa um desafio organizacional. Empresas precisam adotar uma visão colaborativa onde os dados de produção são compartilhados entre diferentes níveis de gestão e operacionais, promovendo uma cultura orientada por dados (BENOTSMANE; KOVáCS; DUDáS, 2019). Essa abordagem requer uma infraestrutura robusta e segura, garantindo que os dados possam ser transmitidos e analisados sem comprometer a integridade dos sistemas.

Assim, as fábricas inteligentes concretizam a aplicação dos princípios da Indústria 4.0 ao oferecer um modelo de produção responsivo e otimizado, capaz de atender às demandas modernas de personalização, eficiência e sustentabilidade. Esse avanço não apenas redefine os processos produtivos, mas também estabelece novos padrões para o setor industrial, enfatizando a importância de uma produção autônoma e conectada para a competitividade global (JASIńSKI; IWANIEC; TADEUSIEWICZ, 2020).

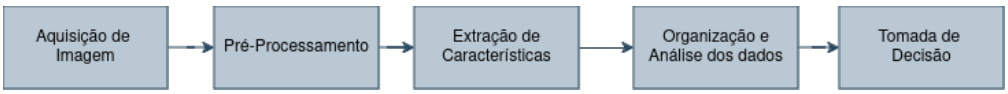
2.2 Visão computacional

A visão computacional é uma área interdisciplinar que integra inteligência artificial, processamento de imagem e aprendizado de máquina, capacitando máquinas a interpretar e processar dados visuais obtidos de diversos sensores (CORKE, 2023). Essa tecnologia permite que sistemas extraiam informações relevantes de imagens e vídeos, simulando a percepção humana e possibilitando decisões autônomas com base em dados visuais (HARTLEY; ZISSERMAN, 2004).

Para ilustrar o fluxo de processamento em um sistema de visão compu-

tacional, a Figura 1 apresenta um fluxograma que descreve as principais etapas envolvidas nesse processo.

Figura 1 – Fluxograma do processo de visão computacional



Fonte: Autoria Própria

O processo começa com a aquisição de uma imagem, geralmente obtida por meio de sensores ou câmeras, que fornecem os dados visuais iniciais ao sistema. Em seguida, a imagem passa por uma etapa de pré-processamento, onde ocorre o ajuste de aspectos como iluminação e remoção de ruídos, com o intuito de melhorar a qualidade do dado visual. Após essa preparação, o sistema realiza a extração de características, selecionando informações relevantes da imagem, como formas, texturas ou padrões específicos, de acordo com o objetivo da aplicação. Essas características são então processadas e organizadas, o que permite que o sistema realize inferências ou reconheça padrões na imagem. Finalmente, com base nas características extraídas e interpretadas, o sistema é capaz de tomar decisões automatizadas ou fornecer informações para apoio à decisão. Esse fluxo geral de processamento torna-se a base para diferentes aplicações, desde simples análises de imagem até sistemas mais complexos (SZELISKI, 2010; CORKE, 2023; HARTLEY; ZISSERMAN, 2004).

2.2.1 Aquisição de imagem

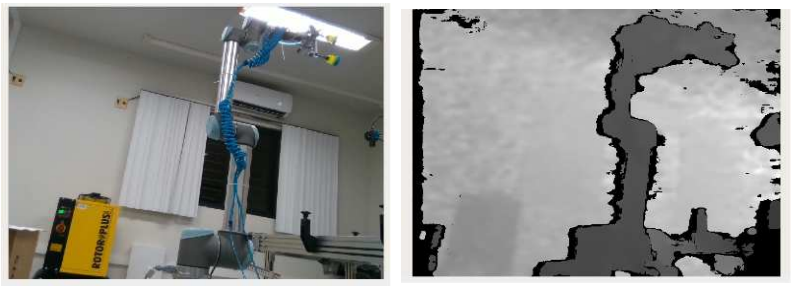
A aquisição de imagem é a primeira etapa no processo de visão computacional e constitui a base sobre a qual todos os outros processos subsequentes dependem. Este estágio envolve a captura de dados visuais utilizando dispositivos como câmeras e sensores, que fornecem ao sistema informações em formato digital para posterior processamento e análise. (SZELISKI, 2010).

Entre os dispositivos mais comuns para a aquisição de imagens, incluem-se as câmeras *RGB (Red, Green e Blue)*, câmeras de profundidade, câmeras estéreo e sensores infravermelhos. As câmeras *RGB*, capturam dados nas três cores básicas,

sendo adequadas para a maioria das aplicações convencionais de visão computacional. As câmeras de profundidade, por sua vez, oferecem dados tridimensionais ao combinar informações de profundidade e textura, importante para aplicações que exigem percepção espacial. As câmeras estéreo, geralmente posicionadas em pares, permitem a reconstrução tridimensional a partir de duas perspectivas distintas, possibilitando o cálculo de profundidade por meio da triangulação. (HARTLEY; ZISSERMAN, 2004) Abaixo, a Figura 2 ilustra um exemplo de dados capturados em formatos *RGB* e *RGB-D*.

Abaixo, estão ilustrados exemplos de dados capturados nos formatos: *RGB* Figura 2(a) e *RGB-D* Figura 2(b).

Figura 2 – Exemplos de imagens nos formatos *RGB* (a) e *RGB-D* (b).



Fonte: Autoria Própria

A qualidade e eficácia da aquisição de imagem em sistemas de visão computacional dependem diretamente das especificações e configurações da câmera utilizada. As principais características de uma câmera estão descritas na Tabela 1

Tabela 1 – Principais características de uma câmera.

Característica	**Descrição**
Resolução	Quantidade de detalhes capturados em pixels.
Tipo de Sensor	Tipo de sensor (CMOS ou CCD), influenciando qualidade e consumo de energia.
Distância Focal e Campo de Visão	Medida que define o ângulo do campo de visão.
Abertura *(f-stop)*	Controla a entrada de luz, afetando exposição e profundidade de campo.
Taxa de Quadros (fps)	Quantidade de quadros capturados por segundo.
Sensibilidade ISO	Capacidade de captar luz em baixa iluminação; valores altos aumentam o ruído.
Profundidade de Cor	Quantidade de cores capturadas, medida em bits.
Conectividade	Tipo de conexão

A escolha adequada dessas configurações é fundamental para assegurar a precisão dos dados visuais e o desempenho do sistema.

As seções 2.2.2 e 2.2.3, que serão descritas a seguir estão baseadas na teoria apresentada no capítulo 13 do livro (CORKE, 2023).

2.2.2 Calibração de câmera

A calibração de câmera é um processo fundamental em sistemas de visão computacional, pois permite a conversão precisa de coordenadas de imagem para coordenadas no espaço tridimensional. Esse procedimento busca minimizar distorções e ajustar projeções para que correspondam à realidade física. Câmeras digitais podem apresentar imprecisões causadas por fatores como as características geométricas das lentes e o alinhamento do sensor, o que torna a calibração indispensável para aplicações que exigem precisão espacial.

Esse processo de calibração considera o modelo de projeção de perspectiva, que é a transformação que ocorre ao representar um mundo tridimensional em uma imagem bidimensional, implicando na perda de informações de profundidade. Este modelo é comumente ilustrado pelo conceito de câmera *Pinhole*, que projeta pontos 3D no plano da imagem usando a perspectiva central.

Para determinar a localização dos pontos na imagem, o modelo matemático usa os parâmetros intrínsecos da câmera, que são: distância focal f e o centro óptico (u_0, v_0), que representam o centro de projeção da imagem. Assim, podemos expressar a relação entre as coordenadas no espaço 3D (X, Y, Z) e as coordenadas projetadas (x, y) no plano da imagem por meio das equações:

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z} \tag{2.1}$$

onde X, Y, Z são as coordenadas de um ponto no espaço tridimensional, e x, y são as coordenadas projetadas no plano da imagem. Esse modelo, ilustrado na Figura 4, resulta em uma imagem não invertida, com o ponto principal como a origem do sistema de coordenadas da imagem.

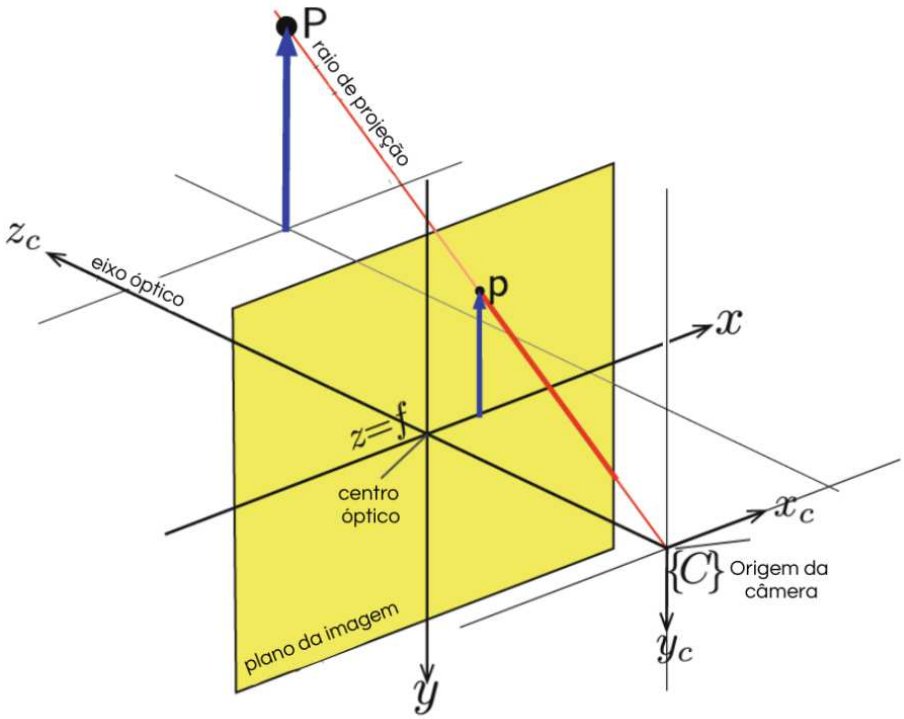
No modelo de projeção, a relação entre um ponto tridimensional $P = (X, Y, Z, 1)^T$ e suas coordenadas na imagem pode ser representada pela matriz de projeção da câmera $\mathbf{C}$, que engloba os parâmetros intrínsecos. A matriz $\mathbf{C}$ é dada por:

$$\mathbf{C} = \begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.2}$$

onde f_x e f_y representam a distância focal em pixels ao longo dos eixos x e y, e (u_0, v_0) define a posição do ponto principal. Essa matriz projeta pontos do espaço 3D para o plano da imagem, de modo que a coordenada homogênea projetada $\mathbf{p} = (x, y, w)^T$ é obtida multiplicando $\mathbf{C}$ por P:

$$\mathbf{p} = \mathbf{C} \cdot \mathbf{P} \tag{2.3}$$

Figura 3 – Modelo de projeção central. O plano da imagem está a uma distância f à frente da origem da câmera, formando uma imagem não invertida. O sistema de coordenadas da câmera segue a regra da mão direita, com o eixo z definindo o centro do campo de visão.



Fonte: Adaptado de (CORKE, 2023)

onde $x = \frac{f_x X}{Z} + u_0$ e $y = \frac{f_y Y}{Z} + v_0$.

Além dos parâmetros intrínsecos, a calibração envolve os parâmetros extrínsecos da câmera, que descrevem sua posição e orientação em relação ao sistema de coordenadas do mundo. Os parâmetros extrínsecos consistem em uma matriz de rotação $\mathbf{R}$ e um vetor de translação $\mathbf{t}$, que juntos compõem uma matriz de transformação $[\mathbf{R}|\mathbf{t}]$. Essa matriz permite converter as coordenadas de um ponto no sistema de referência do mundo para o sistema de referência da câmera. A transformação é dada por:

$$\mathbf{P}_{\text{camera}} = \mathbf{R} \cdot \mathbf{P}_{\text{mundo}} + \mathbf{t} \tag{2.4}$$

onde $\mathbf{P}_{\text{camera}}$ é a posição do ponto no sistema de coordenadas da câmera e $\mathbf{P}_{\text{mundo}}$ é a posição do ponto no sistema de coordenadas do mundo.

Para combinar os parâmetros intrínsecos e extrínsecos, a projeção completa de um ponto tridimensional no sistema de referência do mundo para o plano da imagem é dada por:

$$\mathbf{p} = \mathbf{K} \cdot [\mathbf{R}|\mathbf{t}] \cdot \mathbf{P}_{\text{mundo}} \tag{2.5}$$

onde $\mathbf{K}$ representa a matriz de parâmetros intrínsecos da câmera.

Além disso, os parâmetros intrínsecos também consideram as distorções internas da câmera, que afetam a representação das linhas e formas geométricas na imagem. Distorções, como a radial e a tangencial, são comuns em sistemas de lentes e tornam-se mais evidentes nas bordas da imagem. A distorção radial pode causar curvaturas nas linhas, resultando em efeitos como a distorção em barril (linhas curvas para fora) e a distorção em almofada (linhas curvas para dentro). A distorção tangencial, por outro lado, é causada por desalinhamentos na lente, que geram deslocamentos laterais nos pontos da imagem.

Para representar a distorção radial, podemos usar a seguinte expressão polinomial:

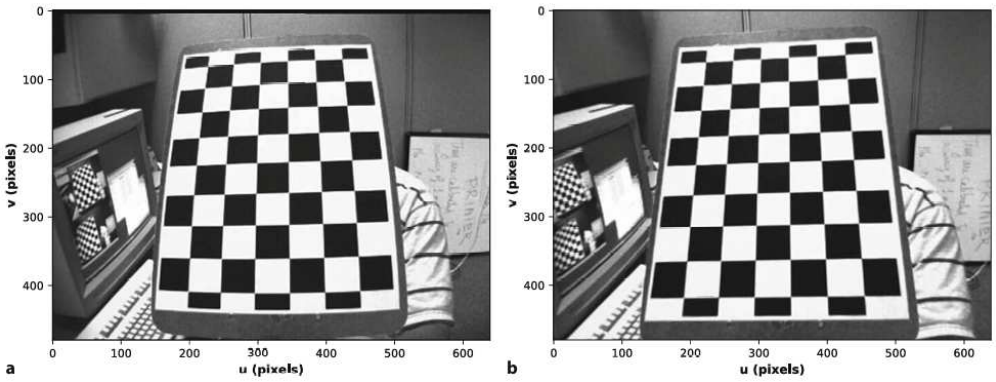
$$\delta r = k_1 r^3 + k_2 r^5 + k_3 r^7 + \cdots \tag{2.6}$$

onde r é a distância radial de um ponto ao centro óptico da imagem e (u_0, v_0), e k_1, k_2, k_3 são os coeficientes de distorção. Assim, como pode ser visto na Figura 4, essas correções são relevantes para que a imagem final seja fiel ao ambiente real e possam ser aplicadas técnicas para minimizar tais distorções nos sistemas de visão computacional.

2.2.2.1 Calibração com tabuleiro de xadrez

A calibração de câmera com um tabuleiro de xadrez utiliza as interseções dos quadrados como pontos de referência. Diversas imagens do tabuleiro são capturadas

Figura 4 – Distorção de lente. a) Imagem distorcida, com curvatura acentuada das linhas verticais no tabuleiro de xadrez; b) Imagem sem distorção.



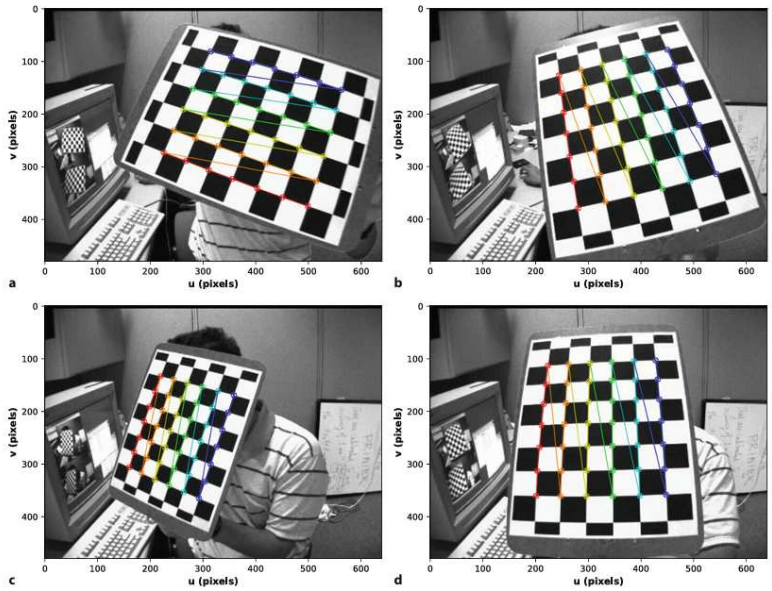
Fonte:(CORKE, 2023)

em diferentes ângulos e distâncias, e as coordenadas dos pontos no plano da imagem (u, v) são associadas às coordenadas conhecidas (X, Y, Z) no espaço tridimensional.

Esse processo permite calcular tanto os parâmetros intrínsecos quanto os extrínsecos da câmera. A calibração também corrige distorções de lente, aplicando uma função de otimização que minimiza o erro entre os pontos observados e os calculados.

A Figura 5 mostra o tabuleiro de xadrez utilizado no processo de calibração, com os pontos de interseção destacados.

Figura 5 – Exemplo de calibração de câmera utilizando um tabuleiro de xadrez. Os pontos de interseção dos quadrados servem como referências de calibração.



Fonte:(CORKE, 2023)

2.2.3 Marcadores fiduciais

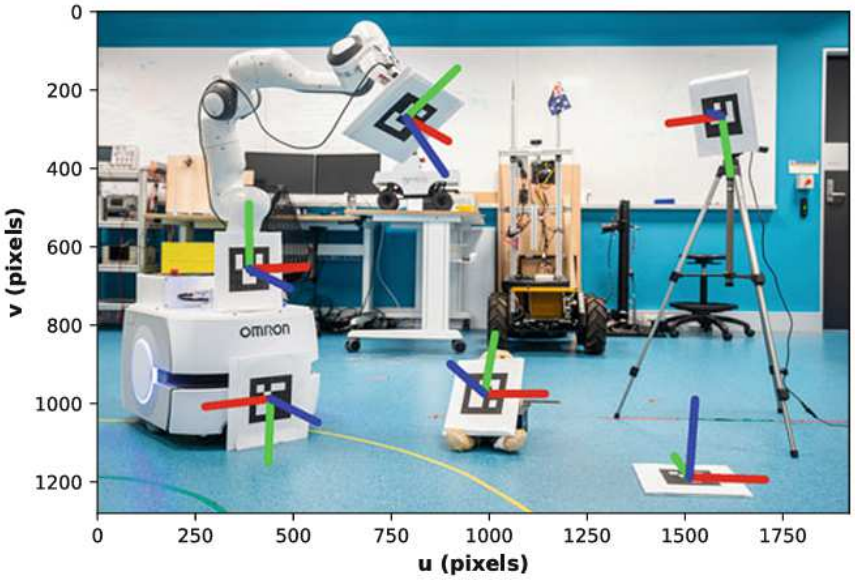
Marcadores fiduciais, como os marcadores ArUco, são um tipo de marcador visual utilizados em visão computacional para auxiliar na detecção e rastreamento de objetos. Esses marcadores consistem em padrões de alto contraste que facilitam a identificação em uma cena e permitem determinar a posição e orientação relativa da câmera em relação ao marcador.

Cada marcador possui um padrão quadrado composto por um código interno que armazena dados binários, permitindo que cada marcador seja identificado de forma única. As famílias de marcadores, como ArUco 4x4_50 e ArUco 4x4_250, indicam o tamanho do *grid* do padrão e o número de identificadores únicos disponíveis. Esse design não apenas facilita a identificação, mas também contribui para estimar a orientação do marcador na cena.

Quando os parâmetros intrínsecos da câmera e as dimensões do marcador

são conhecidos, é possível calcular a posição e orientação do marcador em relação à câmera. Bibliotecas como *OpenCV* possuem funções integradas para detectar marcadores fiduciais, extrair seus contornos e calcular sua posição e orientação. A Figura 6 ilustra uma cena com marcadores ArUco, onde cada marcador fornece um ponto de referência confiável para rastreamento.

Figura 6 – Cena contendo vários marcadores ArUco 4x4_50, usados para rastreamento e estimativa de pose.



Fonte: (CORKE, 2023)

2.3 Sistemas de eventos discretos

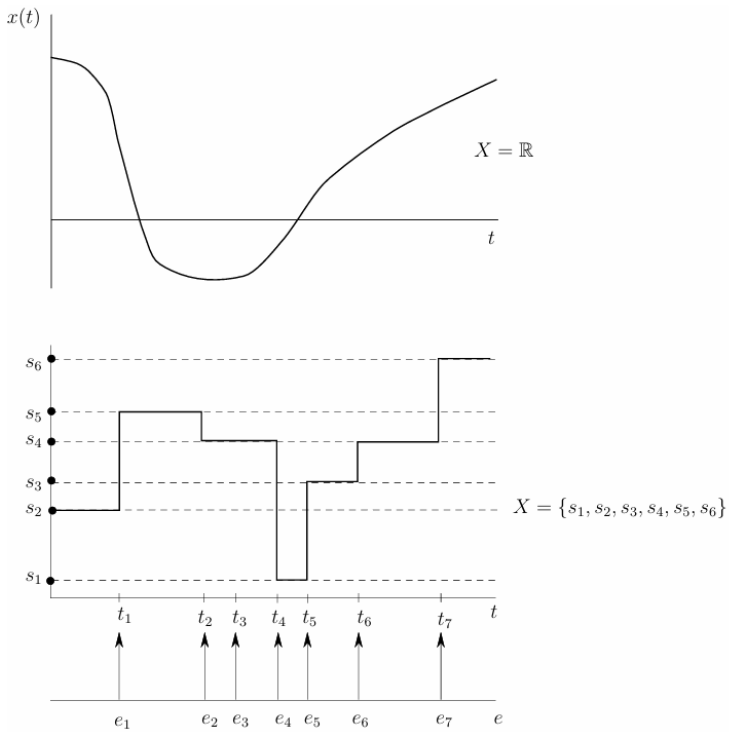
Toda esta seção foi escrita com base na teoria em (CASSANDRAS; LAFORTUNE, 2008).

A análise quantitativa e o desenvolvimento de técnicas de controle são essenciais para medir e otimizar o desempenho de sistemas segundo critérios bem definidos. Para isso, modelos matemáticos são construídos para representar o comportamento de sistemas reais, fornecendo uma base para a análise e o controle desses sistemas. Em termos simples, um modelo pode ser entendido como uma

representação que imita o comportamento do sistema original. No contexto de Sistemas a Eventos Discretos (SED), essa modelagem descreve o comportamento do sistema em função de estados discretos e eventos específicos que provocam transições entre esses estados.

SEDs são sistemas que se caracterizam por estados discretos e mudanças de estado que ocorrem em resposta a eventos específicos. Esses eventos, representados por e em um conjunto E, podem ser ações, condições ou falhas que provocam alterações no sistema. Na Figura 7 compara-se sistemas com variáveis contínuas e SEDs, onde, nos primeiros, o estado varia continuamente, enquanto, nos SEDs, ele muda em resposta a eventos discretos.

Figura 7 – Comparação de caminhos de amostra para sistemas de variáveis contínuas e SED.

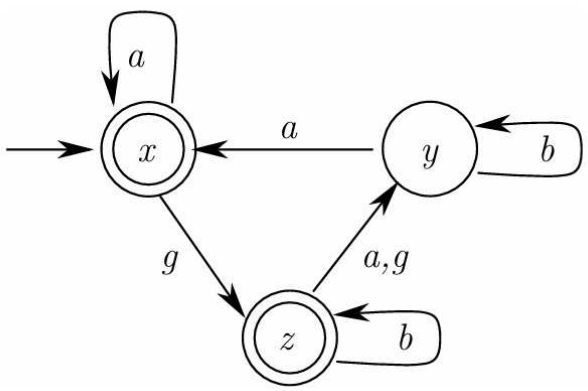


Fonte: (CASSANDRAS; LAFORTUNE, 2008).

SEDs podem ser descritos por autômatos, que modelam transições de estados com base em eventos. Um autômato determinístico é definido por uma sextupla

$G = (X, E, f, \Gamma, x_0, X_m)$, onde X é o conjunto de estados, E o conjunto de eventos, f a função de transição, Γ os eventos ativos, x_0 o estado inicial e X_m o conjunto de estados marcados. Na Figura 8, é ilustrado um diagrama típico de transição de estados, no qual os nós representam estados e os arcos indicam transições de acordo com eventos.

Figura 8 – Diagrama de transição de estados.



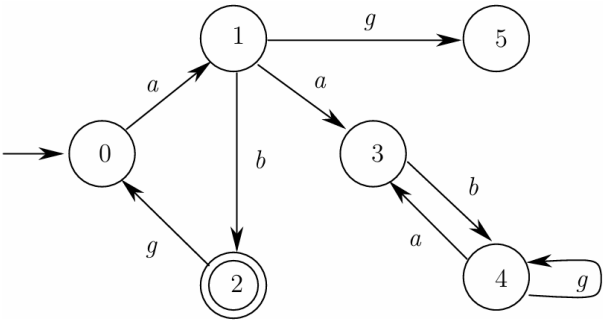
Fonte: (CASSANDRAS; LAFORTUNE, 2008).

Autômatos possuem duas linguagens principais: a linguagem gerada $L(G)$, com todas as sequências de eventos possíveis, e a linguagem marcada $L_m(G)$, que inclui apenas as sequências que levam o sistema a estados marcados específicos. O sistema é considerado bloqueante quando $L_m(G) \subset L(G)$, o que indica a possibilidade de *deadlock* (impasse) ou *livelock* (ciclo infinito sem alcançar um estado final). A Figura 9 exemplifica um autômato bloqueante, onde o estado 5 representa um *deadlock* e os estados 3 e 4 formam um *livelock*.

2.4 Árvores de comportamento

As árvores de comportamento são uma arquitetura de controle cada vez mais aplicada em sistemas de robótica, especialmente em ambientes dinâmicos. Criadas originalmente na indústria de jogos para o controle de personagens, as árvores de comportamento passaram a ser utilizadas em robótica devido à sua modularidade,

Figura 9 – Exemplo de um autômato bloqueante.



Fonte: (CASSANDRAS; LAFORTUNE, 2008).

capacidade de resposta e escalabilidade (COLLEDANCHISE; ÖGREN, 2018). Esta estrutura permite organizar o comportamento de agentes de maneira hierárquica, facilitando a adaptação a mudanças e a resposta a eventos(ÖGREN; SPRAGUE, 2022).

A árvore de comportamento é composta por nós, organizados de forma hierárquica, que representam ações ou condições específicas. Esses nós são classificados em dois tipos principais: *nós de controle* e nós de execução. Os nós de controle determinam o fluxo da árvore, especificando a ordem de execução e as condições para transição entre estados de sucesso, falha ou execução contínua. Três tipos comuns de nós de controle são: sequência, *fallback* e paralelismo. Um nó de sequência, por exemplo, exige que todos os nós filhos tenham sucesso para que o fluxo prossiga, enquanto um nó de *fallback* tenta alternativas até que uma condição de sucesso seja encontrada (GUGLIERMO et al., 2024).

Os nós de execução dividem-se em dois subtipos: nós de ação e nós de condição. Os nós de ação representam tarefas específicas, como mover-se até um ponto, capturar um objeto ou interagir com outros dispositivos. Já os nós de condição verificam o ambiente ou o estado do sistema, controlando a sequência de execução conforme as condições estabelecidas. Esse design modular permite a reutilização de blocos de comportamento e adaptações rápidas a novas situações e objetivos (GUGLIERMO et al., 2024).

A modularidade e a reatividade das Árvores de Comportamento apresentam vantagens para robôs que operam em ambientes dinâmicos e compartilhados, como hospitais e fábricas. Nesse contexto, as Árvores de Comportamento permitem que os robôs reajam em tempo real a mudanças no ambiente, como obstáculos inesperados ou alterações nos objetivos das tarefas (Quasi Robotics, 2024). A estrutura hierárquica das Árvores de Comportamento facilita o encapsulamento de comportamentos complexos em subárvores, promovendo clareza e simplicidade no controle, além de possibilitar a execução paralela de tarefas. Essa característica é especialmente útil para equipes de robôs heterogêneos, onde diferentes capacidades precisam ser coordenadas de forma distribuída.

A integração das árvores de comportamento com sistemas de visão tem mostrado resultados relevantes. Segundo (PAXTON et al., 2016), o uso de módulos de visão computacional em conjunto com árvores de comportamento é explorado para orientar robôs colaborativos em tarefas, como manipulação e montagem em ambientes colaborativos. A visão computacional é empregada para detectar e interpretar informações visuais do ambiente, que são então processadas por nós de condição na árvore. Dessa forma, o robô é capaz de monitorar a posição e o estado dos objetos ao redor, ajustando automaticamente suas ações de acordo com a situação. Essa abordagem melhora a precisão e a eficiência das operações, especialmente em atividades que requerem adaptação contínua às mudanças no ambiente (PAXTON et al., 2016).

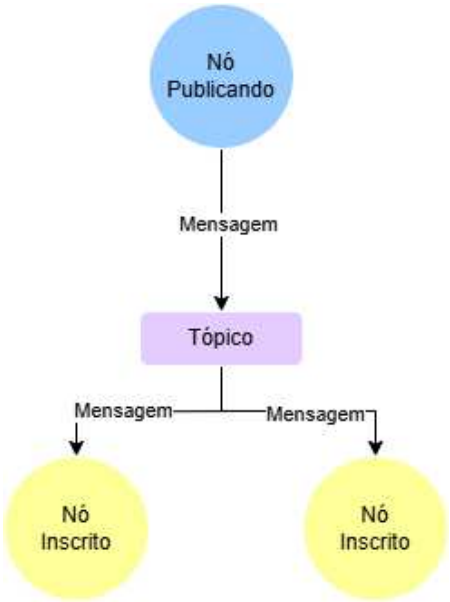
2.5 *Robot Operating System* (ROS)

O *Robot Operating System (ROS)* é uma coleção de bibliotecas de *software* de código aberto que oferece infraestrutura para o desenvolvimento de sistemas robóticos. Ele foi projetado para facilitar a integração de diferentes componentes, como sensores e atuadores, permitindo a criação de sistemas robóticos modulares e escaláveis. Sua principal característica é a arquitetura distribuída, na qual processos independentes, chamados de nós, comunicam-se de maneira eficiente por meio de um modelo de publicação e assinatura, garantindo flexibilidade e adaptabilidade ao sistema robótico (SIMONIč et al., 2021; DOE; SMITH; PEREZ, 2020).

Para configurar sistemas com múltiplos nós, o ROS utiliza pacotes, que agrupam nós, dependências, arquivos de configuração e outros recursos necessários para executar tarefas específicas. Esses pacotes organizam o desenvolvimento e a distribuição de funcionalidades em sistemas robóticos. O ROS também oferece arquivos de lançamento (*launch files*), configurados em XML ou Python, que permitem iniciar vários nós e definir parâmetros do sistema de forma automatizada.

A modularidade do ROS permite uma arquitetura distribuída, facilitando a execução de várias operações simultaneamente, como a captura de dados de sensores e o processamento de algoritmos de controle. A comunicação entre os nós é realizada por meio de tópicos, nos quais os dados podem ser publicados por um nó e recebidos por outros de maneira assíncrona. A Figura 10 ilustra esse modelo de comunicação entre nós, no qual um nó publica uma mensagem em um tópico, que é então distribuída para todos os nós inscritos.

Figura 10 – Modelo de comunicação por publicação e assinatura no ROS.



Fonte: Autoria própria

2.6 Gazebo clássico

O Gazebo é uma plataforma de simulação robótica 3D de código aberto, amplamente utilizada no desenvolvimento, teste e validação de robôs em ambientes tridimensionais realistas. Criado em 2002, o Gazebo foi projetado para oferecer simulações físicas de alta fidelidade, replicando fenômenos como gravidade, colisões e fricção, o que o torna uma ferramenta fundamental para prever e validar o comportamento de robôs antes da implementação física. Sua arquitetura modular oferece suporte a uma gama de sensores e atuadores simulados, proporcionando um ambiente para o teste de algoritmos de controle, navegação e interação (GAZEBO..., 2024).

O Gazebo é adequado para simular ambientes dinâmicos e complexos em que os robôs interagem com objetos que possuem características físicas realistas, como massa, velocidade e força. Esses atributos simulam situações como empurrar, levantar e derrubar objetos, permitindo que os desenvolvedores ajustem os robôs a partir de um ambiente seguro e controlado. As forças aplicadas aos robôs e os objetos são processadas por motores de física, como o *Open Dynamics Engine (ODE), Bullet, Simbody* e *DART*, garantindo simulações realistas e precisas (SIMONIč et al., 2021; GAZEBO..., 2024).

De acordo com a documentação do (GAZEBO..., 2024), as principais funcionalidades que o simulador oferece podem ser descritas da seguinte forma:

- **ambiente de simulação**: dispõe de uma plataforma avançada que permite simular robôs em diferentes cenários de pesquisa, desenvolvimento e testes de forma realista e controlada;
- ***plugins***: possui suporte à criação de *plugins* personalizados, possibilitando o controle adicional de dispositivos além da personalização de ambientes simulados;
- **sistema de comunicação**: oferece uma infraestrutura distribuída e assíncrona de troca de mensagens para integração entre os diferentes componentes do sistema robótico;

- **simulação de sensores**: suporta diferentes tipos de sensores virtuais, como câmeras *RGB*, sensores de profundidade, permitindo a simulação de ruídos e condições realistas de operação;
- **integração física**: o Gazebo utiliza uma interface modular para motores de física que permitem simular interações físicas detalhadas, como gravidade, colisões e outras forças externas;
- **renderização gráfica**: inclui motores de renderização gráfica que oferecem visualizações 3D detalhadas e realistas do ambiente simulado;
- **modelagem de robôs**: fornece diversos modelos prontos de robôs e permite a criação de modelos personalizados utilizando o formato *SDF (Spatial Data File)*.

A integração do Gazebo com o ROS é um dos seus maiores diferenciais. Essa integração facilita o desenvolvimento iterativo de robôs, permitindo que os sistemas sejam testados, ajustados e otimizados em um ambiente virtual antes da implementação em cenários físicos (SIMONIč et al., 2021). Isso minimiza riscos e custos, proporcionando uma transição suave entre os testes no simulador e as operações em ambientes reais.

3 Metodologia

Este Capítulo descreve o desenvolvimento de um sistema de controle orientado a eventos para uma fábrica inteligente, empregando realimentação visual e uma estrutura de programação distribuída.

O objetivo foi desenvolver um sistema capaz de responder de forma adaptativa às mudanças no ambiente de produção, reagindo de maneira autônoma a eventos detectados visualmente. O sistema foi implementado e testado no ambiente experimental Laboratório*Smart Factory*, projetado para simular as condições de uma fábrica inteligente e alinhado ao contexto da Indústria 4.0. Esse ambiente está localizado no Laboratório de Sistemas Embarcados e Computação Pervasiva (Embedded) da Universidade Federal de Campina Grande.

O desenvolvimento do sistema iniciou-se com a criação de uma cena virtual no simulador Gazebo, representando a configuração física do laboratório e a posição dos principais dispositivos. Em seguida, os dispositivos físicos foram instalados e configurados no laboratório para operar de maneira integrada ao sistema de controle.

Para adaptar a câmera ao sistema, foi desenvolvido um modelo genérico baseado em eventos para dispositivos de aquisição de imagem. Esse modelo foi implementado em uma árvore de comportamento, permitindo acoplar funcionalidades específicas, como a detecção de marcadores fiduciais e a estimação de posição e orientação.

Por fim, o sistema foi validado por meio de tarefas representativas de uma fábrica inteligente, avaliando sua capacidade de resposta a eventos visuais e sua adaptação a diferentes condições operacionais.

3.1 Ambiente de desenvolvimento

Para o desenvolvimento deste trabalho, foi utilizado o sistema operacional *Ubuntu 22.04*, devido a sua compatibilidade com o ROS e estabilidade reconhecida em aplicações robóticas. A versão utilizada do ROS foi a *Humble Hawksbill*. Para a gestão do ambiente de desenvolvimento, foi utilizado o *Docker*[1] em conjunto com o *Dev Containers*[2], garantindo que todas as dependências necessárias fossem isoladas e que o ambiente fosse replicável de forma consistente, evitando problemas de incompatibilidade de versões de bibliotecas e ferramentas. Essa abordagem também permitiu uma maior flexibilidade no gerenciamento do ciclo de desenvolvimento. Adicionalmente, o versionamento do código foi realizado utilizando *Git* com repositórios hospedados no *GitHub*, garantindo o controle das alterações realizadas ao longo do projeto.

3.1.1 Dispositivos utilizados

3.1.1.1 Câmera RGD-D Microsoft Kinect 360

Neste trabalho, foi utilizado o sensor Kinect 360 da Microsoft como dispositivo de captura de imagem e profundidade. O Kinect é um sensor comum de ser utilizado em aplicações de visão computacional e robótica devido à sua capacidade de aquisição imagens em tempo real e de fornecer informações detalhadas sobre a profundidade dos objetos no ambiente (EL-IAITHY; HUANG; YEH, 2012). Desenvolvido inicialmente para a plataforma de jogos *Xbox*, o Kinect rapidamente encontrou espaço na pesquisa e desenvolvimento de soluções de baixo custo para aplicações em robótica e sistemas de monitoramento de ambientes.

O Kinect opera utilizando um sensor RGB para capturar imagens convencionais e um sensor de profundidade baseado em infravermelho para medir a distância de objetos em uma cena. A Tabela 2 apresenta as principais especificações técnicas do Kinect, incluindo a resolução do sensor RGB, faixa de alcance do sensor de profundidade, campo de visão e precisão de medição, características que tornam o

[1] Documentação do *Docker*: <https://docs.docker.com/>
[2] Documentação do *Dev Containers*: <https://code.visualstudio.com/docs/devcontainers/containers>

Figura 11 – Sensor Microsoft Kinect 360.

Fonte: Adaptado de (COMMONS,)

dispositivo adequado para capturar dados precisos em ambientes internos.

Tabela 2 – Especificações técnicas do Microsoft kinect

Características	**Especificações**
Resolução do sensor RGB	640 x 480 pixels
Faixa de alcance do sensor de profundidade	0.8 a 4.0 metros
Campo de visão horizontal	57 graus
Campo de visão vertical	43 graus
Frequência de captura	30 FPS
Precisão de medição de profundidade	±1 cm

Fonte: Adaptado de (EL-IAITHY; HUANG; YEH, 2012)

No contexto deste trabalho, o Kinect foi selecionado tanto por sua disponibilidade em laboratório, como também devido à sua capacidade de capturar dados de profundidade e imagens de forma simultânea, com precisão satisfatória para aplicações internas. Dessa forma, o Kinect fornece uma solução eficaz e acessível para o levantamento de informações visuais e espaciais em tempo real.

3.1.1.1.1 Integração do Kinect com o ROS

A integração do Kinect com o ROS foi realizada utilizando os pacotes `libfreenect` e `kinect_ros2`[3]. O `libfreenect` fornece os *drivers* necessários para

[3] Repositórios dos drivers no GitHub: `libfreenect`: <https://github.com/OpenKinect/libfreenect.git>, `kinect_ros2`: <https://github.com/fadlio/kinect_ros2>

que o ROS possa se comunicar com o sensor Kinect, enquanto o `kinect_ros2` é responsável por publicar os dados de imagem e profundidade do Kinect em tópicos específicos do ROS, permitindo seu uso em aplicações de visão computacional.

A configuração completa do Kinect com o ROS envolve três etapas principais:

1. Instalação do `libfreenect` e do `kinect_ros2`: é necessário instalar o `libfreenect` no sistema operacional, que fornece os *drivers* para comunicação com o Kinect. Em seguida, o `kinect_ros2` é instalado para permitir que os dados do Kinect sejam publicados diretamente no ROS.

2. Conexão e verificação do dispositivo: após a instalação dos pacotes, o Kinect deve ser conectado ao computador via USB. A verificação da conexão pode ser realizada com o comando `lsusb`, para assegurar que o sistema reconheceu o Kinect como um dispositivo ativo.

3. Inicialização dos nós de publicação de dados: com o Kinect conectado e reconhecido, o próximo passo é iniciar o `kinect_ros2` com um arquivo de inicialização (*launch*), que inicia automaticamente a publicação dos dados de imagem e profundidade do Kinect em tópicos do ROS.

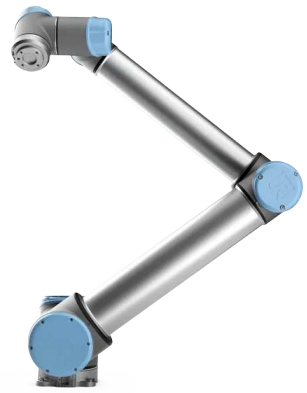
Concluídas as etapas de instalação e conexão, a configuração dos tópicos no ROS possibilita o uso dos dados do Kinect em outras aplicações.

3.1.1.2 Manipulador Robótico UR10

O UR10, desenvolvido pela *Universal Robots*, é um manipulador robótico industrial comumente utilizado em tarefas de automação, incluindo montagem, embalagem e movimentação de objetos. Esse robô possui capacidade de carga de até 10 kg e alcance máximo de 1300 mm, (Universal Robots, 2023) sendo amplamente utilizado nos setores de manufatura e logística devido à sua precisão e flexibilidade. Na aplicação deste trabalho, o UR10 está equipado com um efetuador final ventosas.

O UR10 permite a integração com sistemas de controle de automação, comunicando-se por meio de protocolos de rede como TCP/IP, Modbus TCP e Profinet. No contexto deste trabalho, ele já estava previamente configurado e integrado ao ROS, o que possibilita seu controle por comandos enviados através da interface distribuída. Na aplicação experimental, o UR10 foi utilizado para

Figura 12 – Manipulador robótico UR10



Fonte: (Universal Robots, 2023)

manipulação e movimentação de objetos na cena de teste, operando em resposta aos eventos gerados pelo sistema de visão computacional.

3.1.1.3 Robô móvel TurtleBot2i

O TurtleBot2i é um robô móvel que foi desenvolvido com o objetivo de ser utilizado em aplicações de pesquisa, educação e prototipagem em robótica (TurtleBot, 2023b). Esse robô é comumente empregado em projetos de navegação, mapeamento e manipulação, sendo uma solução flexível e modular que possibilita a adição de diferentes sensores, facilitando experimentos em robótica autônoma e colaborativa (TurtleBot, 2023c). A Figura 13 apresenta o TurtleBot2i, destacando sua estrutura modular.

No ambiente de desenvolvimento deste trabalho, o TurtleBot2i já estava configurado para operar com o ROS, dispensando ajustes adicionais para sua integração. Todos os pacotes necessários para comunicação e controle do TurtleBot2i já estavam instalados, permitindo acesso direto aos tópicos e serviços disponibilizados pelo robô. A operação do TurtleBot2i no ROS possibilita o compartilhamento de dados dos sensores e o recebimento de comandos de movimentação. Sua estrutura modular inclui sensores e uma base móvel com rodas, permitindo deslocamentos em diferentes direções.

Figura 13 – Robô Móvel TurtleBot2i.

Fonte: (TurtleBot, 2023a)

3.1.1.4 Esteira transportadora industrial

A esteira transportadora utilizada neste trabalho faz parte dos dispositivos disponíveis no laboratório e é fundamental para simular o fluxo de materiais e produtos dentro de uma fábrica inteligente. Este dispositivo permite o transporte contínuo de objetos, facilitando o teste de cenários que envolvem a movimentação e o controle de materiais ao longo de uma linha de produção. Vale ressaltar que a esteira não possui comunicação direta com o sistema ROS, operando de forma independente em relação aos demais dispositivos integrados no ambiente de simulação.

Figura 14 – Esteira transportadora industrial

Fonte: Autoria própria

3.2 Integração dos dispositivos físicos com o ROS

Para configurar os dispositivos físicos utilizados no laboratório *Smart Factory*, foi desenvolvido em Python o pacote `smartfactory_bringup`. Este pacote possui uma estrutura de diretórios para organizar os arquivos de configuração e inicialização dos dispositivos no ambiente ROS. Essa estrutura pode ser observada na Figura 15.

Figura 15 – Estrutura de diretórios do pacote `smartfactory_bringup`

```
smartfactory_ws/
|-- src/
    |-- smartfactory_bringup/
    |   |-- config/
    |   |-- launch/
    |   |-- smartfactory_bringup/
```

Fonte: Autoria própria

A seguir, uma descrição dos diretórios:

- `config/`: armazena os arquivos de configuração dos dispositivos, como parâmetros de calibração das câmeras e tópicos específicos para comunicação no ROS. Esses arquivos permitem a personalização de parâmetros para cada dispositivo, facilitando ajustes no ambiente experimental.
- `launch/`: contém os arquivos de inicialização, responsáveis por carregar as configurações e iniciar os dispositivos físicos no ambiente ROS. Assim, esses arquivos simplificam o processo de inicialização, permitindo que todos os dispositivos sejam configurados e conectados com um único comando de execução.
- `smartfactory_bringup/`: contém os nós necessários para a comunicação e controle dos dispositivos no sistema distribuído, além de códigos e arquivos auxiliares para a configuração de cada dispositivo.

Informações adicionais sobre o repositório do código e detalhes das colaborações no desenvolvimento de partes do sistema podem ser encontradas no Apêndice A.

3.3 Calibração da Câmera Kinect

Para garantir a precisão das medições visuais no sistema, foi realizado o processo de calibração da câmera Kinect, necessária para a correta aplicação de visão computacional. A câmera, fixada no teto do ambiente experimental, foi calibrada utilizando um padrão de tabuleiro de xadrez. Esse procedimento foi executado com o pacote de calibração de câmera disponível no ROS, chamado `camera_calibration`.[4]

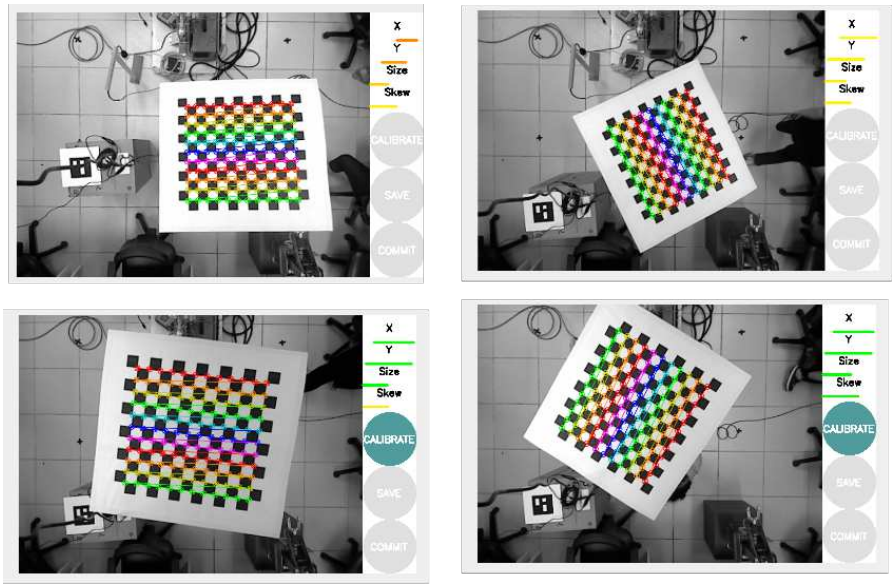
Para iniciar o processo de calibração, a câmera deve estar em modo de transmissão contínua, publicando as informações de aquisição de dados em um nó no ROS. O pacote de calibração de câmeras do ROS inscreve-se nesse nó para acessar as imagens transmitidas pela câmera. Na inicialização do pacote de calibração, foram especificados parâmetros como o nome do nó de publicação das imagens, o tamanho do tabuleiro de xadrez (11x12) e o tamanho do quadrado do padrão de calibração (0,03 m).

Durante a calibração, o pacote orienta por meio de uma interface gráfica, indicando os eixos coordenados que ainda necessitam de dados adicionais para otimizar a calibração. Com base nessas orientações, o tabuleiro deve ser movimentado conforme recomendado, assegurando a aquisição de imagens em diferentes posições e ângulos. A Figura 16 ilustra a interface do pacote de calibração em diversas etapas do processo, destacando o estado dos eixos (X, Y, *size, skew*) que precisam de mais informações para melhorar a precisão.

Quando o pacote de calibração considera que foram coletados dados suficientes em todos os eixos, o processo é finalizado, e os parâmetros resultantes são armazenados em um arquivo de configuração no formato *'.yaml'*. Esse arquivo contém as informações necessárias para corrigir distorções e alinhar a imagem da câmera com a realidade física.

[4] Repositório do pacote no GitHub: <https://wiki.ros.org/camera_calibration>

Figura 16 – Calibração do Kinect usando o pacote ROS *camera_calibration*

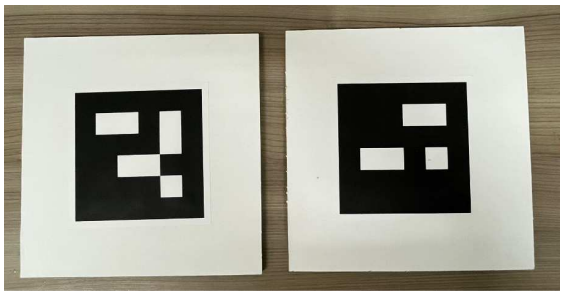


Fonte: Autoria própria

3.4 Detecção de Marcadores Fiduciais

Para a identificação de objetos e a geração de eventos visuais na cena, foram utilizados marcadores fiduciais do tipo ArUco 4x4 com tamanho de 150 mm, como ilustrado na Figura 17. Essa abordagem permite que a câmera identifique a posição e a orientação dos objetos no ambiente, facilitando o controle da cena com base nos eventos gerados pela detecção dos marcadores.

Figura 17 – Marcadores ArUco

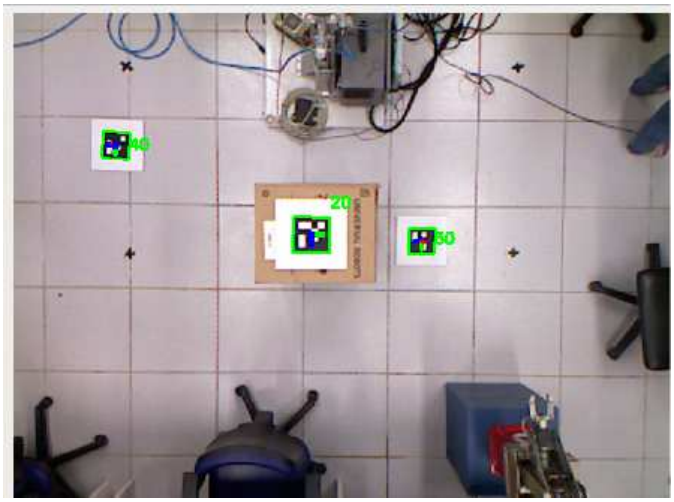


Fonte: Autoria própria

Para o processo de detecção, foi utilizado o pacote `aruco_pose_estimation`[5], disponível no ROS, que realiza a detecção dos marcadores utilizando técnicas de visão computacional e informações RGB e RGB-D da câmera para a estimação de posição e orientação.

O funcionamento do pacote requer que a câmera esteja em modo de transmissão contínua, publicando as imagens capturadas em um tópico no ROS. Após a inicialização, o pacote carrega o arquivo de calibração da câmera no formato "yaml", utilizado para corrigir distorções e garantir a precisão da detecção. Em seguida, o *RViz*, ferramenta de visualização do ROS, é aberto, exibindo a imagem da câmera em tempo real. Quando um marcador ArUco é detectado, o pacote destaca automaticamente os contornos do marcador e sobrepõe o sistema de coordenadas diretamente na transmissão exibida no *RViz*, conforme apresentado na Figura 18.

Figura 18 – Detecção e estimação de posição e orientação de Marcadores ArUco no Ambiente de Teste



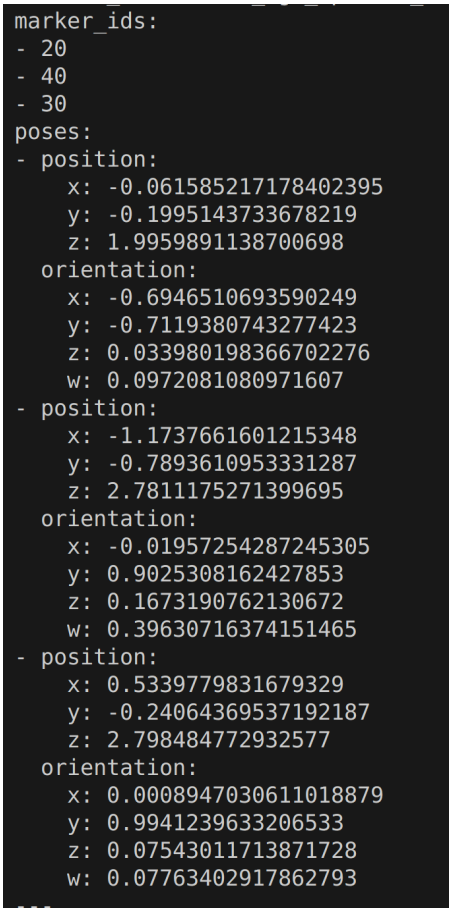
Fonte: Autoria própria

Além disso, o pacote publica em um novo tópico as informações de posição, orientação e o ID de cada marcador detectado, permitindo o monitoramento e controle dos objetos identificados na cena. Inicialmente, o sistema publica todos os IDs dos marcadores detectados; em seguida, são publicados a posição e a orientação

[5] Repositório do pacote no GitHub: *aruco_pose_estimation<https://github.com/AIRLab-POLIMI/ros2-aruco-pose-estimation>*

em *quaternions* de cada marcador, na mesma ordem em que foram exibidos os IDs, todas essas informações são relativas à câmera. O sistema também suporta a detecção de múltiplos marcadores simultaneamente, possibilitando a monitoração de diversos objetos no ambiente. A Figura 19 apresenta o ID, a posição e a orientação dos marcadores ArUco detectados na cena exibida na Figura 18.

Figura 19 – Informações publicadas em um tópico ROS de marcadores ArUco: IDs, posições e orientações

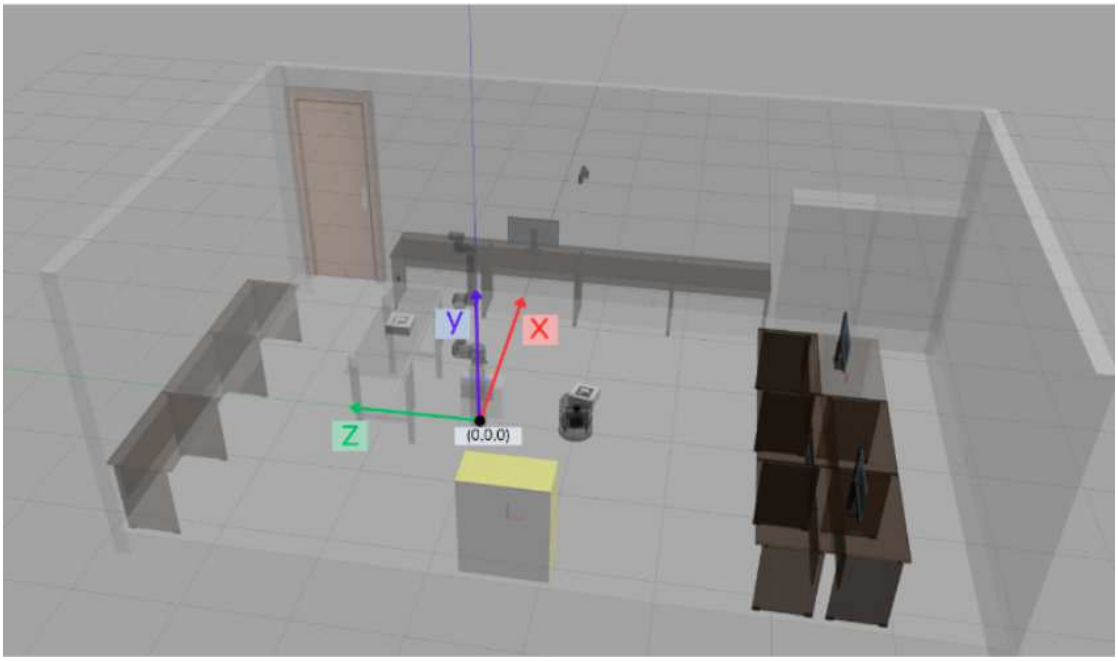


Fonte: Autoria própria

Para que fosse conhecida a posição dos marcadores detectados em relação ao sistemas de coordenadas de referência , a partir da pose publicada, foi necessário desenvolver um nó no ROS que publica em um novo tópico a pose do marcador referenciada em relação ao sistema de coordenadas origem de todo o sistema. Na

Figura 20 está ilustrada o sistemas de coordenadas de referência do sistema definido para (x, y, z).

Figura 20 – Sistemas de coordenadas de referência de todo o sistema no ambiente de simulação Gazebo

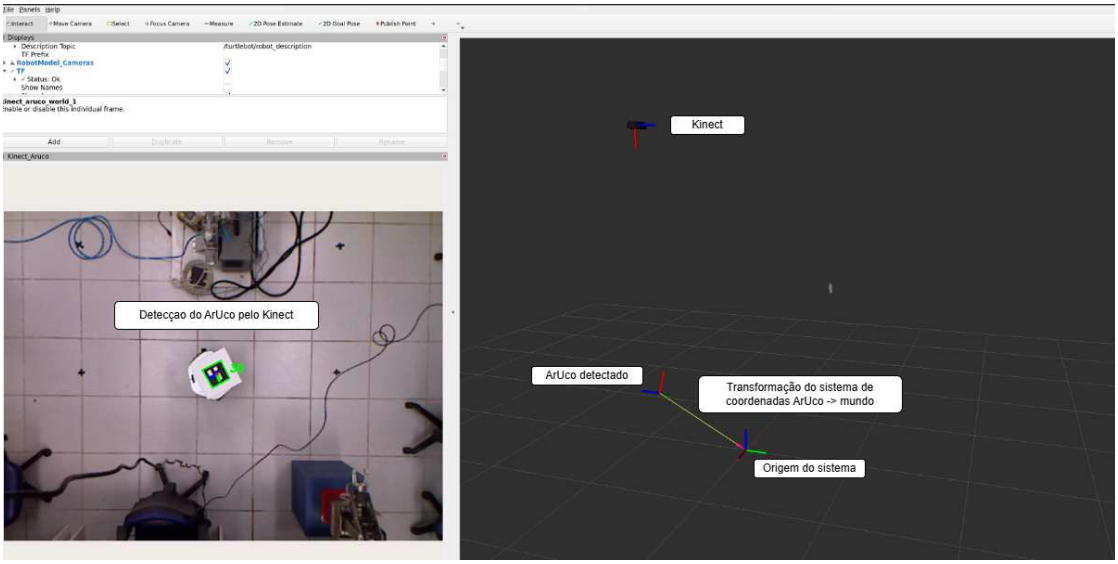


Fonte: Autoria própria

Na Figura 21 é apresentado os sistemas de coordenadas de referência e dos dispositivos físicos Kinect, UR10 e do marcador que esta sendo detectado. Esse sistema de coordenadas foi definido utilizando o pacote `TF2`[6] do ROS. Para converter as coordenadas, foi feito um código em Python utilizando o pacote que calcula as transformações de coordenadas dos dispositivos para a referência. Com essa etapa finalizada, as poses dos arucos detectados, são transformadas do sistema de coordenadas da câmera para o sistema de coordenadas de referência e são publicadas no tópico `_kinect_aruco_poses_world`

[6] Documentação da biblioteca TF2:<https://docs.ros.org/en/humble/Concepts/Intermediate/About-Tf2.html>

Figura 21 – Sistemas de coordenadas de referência do sistema, da câmera Kinect e do UR10



Fonte: Autoria própria

3.5 Simulação no Gazebo

A simulação desenvolvida neste trabalho teve como principal objetivo facilitar o desenvolvimento, integração e testes dos dispositivos utilizados no Laboratório *Smart Factory*, bem como permitir a realização de experimentos em um ambiente controlado antes da implementação no ambiente físico. Para a sua implementação, foi utilizado software Gazebo, escolhido devido à sua capacidade de integrar-se diretamente com o ROS. Essa integração permite que os dispositivos simulados no Gazebo interajam com o ambiente ROS de forma semelhante ao que ocorreria no ambiente físico, possibilitando o controle e a comunicação entre os elementos simulados e ROS.

3.5.1 Estrutura dos Pacotes de Simulação

Para organizar a simulação e garantir modularidade, foram criados dois pacotes principais: `smartfactory_description` e `smartfactory_simulation`. Cada pacote foi estruturado para cumprir uma função específica dentro do projeto.

Na Figura 22 é apresentada a estrutura dos diretórios dos pacotes, mostrando a organização dos arquivos principais que compõem o ambiente de simulação.

Figura 22 – Estrutura de diretórios dos pacotes

```
smartfactory_ws/
|-- src/
    |-- smartfactory_description/
    |   |-- meshes/
    |   |-- rviz/
    |   |-- urdf/
    |
    |-- smartfactory_simulation/
        |-- launch/
        |-- models/
        |-- world/
```

Fonte: Autoria própria

O pacote `smartfactory_description` contém a descrição de cada dispositivo e componente utilizado na simulação, organizando-os em diretórios específicos conforme mostrado na Tabela 3, que apresenta os principais diretórios, seus conteúdos e funções no contexto da simulação.

Tabela 3 – Organização do pacote `smartfactory_description`

Diretório	**Conteúdo**	**Função**
`meshes`	Malhas 3D	Define a aparência visual dos dispositivos
`rviz`	Configurações `.rviz`	Configura visualizações no RViz para análise e monitoramento da simulação
`urdf`	Arquivos `.xacro`	Descreve a estrutura física e as propriedades dos dispositivos, como *links*, *joints*, massa e inércia

O pacote `smartfactory_simulation` é responsável pela coordenação e inicialização da simulação, organizando os arquivos necessários para que os dispositivos sejam carregados e posicionados no ambiente Gazebo, garantindo a comunicação

com o ROS. Esse pacote inclui arquivos de inicialização (`launch`), modelos personalizados, scripts auxiliares e a definição do ambiente simulado. A Tabela 4 apresenta a estrutura do pacote `smartfactory_simulation`, detalhando os principais diretórios, seus conteúdos e funções.

Tabela 4 – Organização do pacote `smartfactory_simulation`

Diretório	**Conteúdo**	**Função**
`launch`	Arquivos `.launch.py`	Coordenam a inicialização dos dispositivos e da cena no Gazebo
`models`	Modelos adicionais	Contém modelos personalizados ou modificados para o ambiente simulado
`world`	Arquivo `.world`	Define a configuração espacial do ambiente simulado no Gazebo

3.5.2 Configuração da Cena e Integração com o ROS

A criação da cena simulada consistiu na modelagem do ambiente físico, a inclusão dos robôs e sensores, e a configuração dos arquivos de inicialização para permitir a execução completa da simulação.

Para reproduzir a estrutura do laboratório no ambiente virtual, foram realizadas medições detalhadas da sala com o auxílio de uma trena *laser*, assegurando que cada elemento estivesse posicionado de maneira correspondente a configuraçaõ do ambiente físico. A estrutura da sala foi montada utilizando modelos existentes da *Gazebo Models and Worlds Collection* [7], que fornece uma variedade de objetos e materiais adequados para compor ambientes no *Gazebo*. Com base nas dimensões coletadas, foi criado o arquivo `lab_smart_factory.world`, onde móveis, computadores e outros itens do laboratório foram inseridos e posicionados conforme o arranjo real.

Os modelos do robô UR10 e do TurtleBot2i já são desenvolvidos pela comunidade e portanto foram integrados ao ambiente de simulação, enquanto os demais dispositivos foram modelados especificamente para este projeto. Para o

[7] Repositório Gazebo Models and Worlds Collection: <https://github.com/leonhartyao/gazebo_models_worlds_collection>

UR10 foi utilizado o pacote *Universal Robots ROS2 Gazebo Simulation*[8] e para o TurtleBot2i foi utilizado o pacote *Turtlebot2i Simulation*[9].

As câmeras Kinect e Basler foram incluídas na simulação por meio de arquivos *xacro* desenvolvidos especificamente para este projeto. O uso de `.xacro` possibilita a parametrização e reutilização de componentes entre diferentes dispositivos, tornando a modelagem mais modular e adaptável a mudanças. Os arquivos `basler.urdf.xacro` e `kinect.urdf.xacro` permitiram a inclusão dos modelos *STL* das câmeras, definindo suas posições no ambiente virtual e configurando o *plugin* `libgazebo_ros_camera`. Esse *plugin* foi utilizado para emular as funcionalidades das câmeras no ROS, com a especificação de parâmetros como resolução de imagem, campo de visão e taxa de captura. O posicionamento das câmeras na simulação foi alinhado com suas localizações no laboratório físico.

Para o início da simulação, foi implementado um arquivo de lançamento que gerencia toda a configuração da cena no *Gazebo*, incluindo a inicialização dos dispositivos e sua integração com o ROS. Este arquivo de lançamento chama os seguintes nós:

1. Gazebo: inicializa o ambiente de simulação a partir do arquivo *world*, que representa a estrutura completa do laboratório.

2. TurtleBot2i: carrega o robô móvel, configurado para atuar conforme a posição e orientações definidas na cena, de modo a reproduzir seu comportamento no laboratório real.

3. UR10: ativa o manipulador robótico utilizando o pacote oficial do ROS, ajustando sua posição para corresponder à disposição física no laboratório.

4. Câmeras Kinect e Basler: configura os parâmetros de captura de imagem e profundidade das câmeras, assegurando a reprodução fiel de suas funções por meio do *plugin*.

[8] Repositório do pacote no GitHub:<.https://github.com/UniversalRobots/Universal_Robots_ROS2_Gazebo_Simulation>

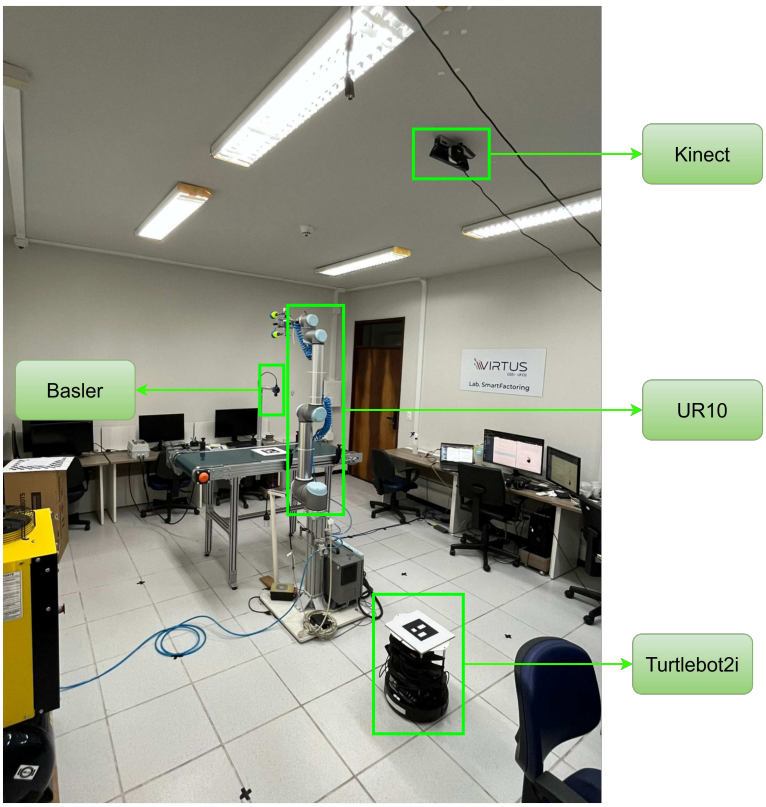
[9] Repositório Turtlebot2i Simulation: <https://github.com/SmartFactoryLab-UFCG/smartfactory_pkg/tree/main/smartfactory_ws/src/smartfactory/smartfactory_simulation>

5. *Rviz*: executa o *Rviz*, permitindo a visualização em tempo real de todos os componentes da simulação, o que facilita o monitoramento e o ajuste dos dispositivos na cena.

6. Transformações estáticas: configura os nós de transformação estática para sincronizar os sistemas de coordenadas, assegurando que todos os dispositivos da simulação compartilhem o mesmo referencial.

Na Figura 23, é apresentada a configuração do laboratório físico com os dispositivos reais, como o UR10, o TurtleBot2i, a câmera Kinect, a câmera Basler, a esteira e os móveis do laboratório.

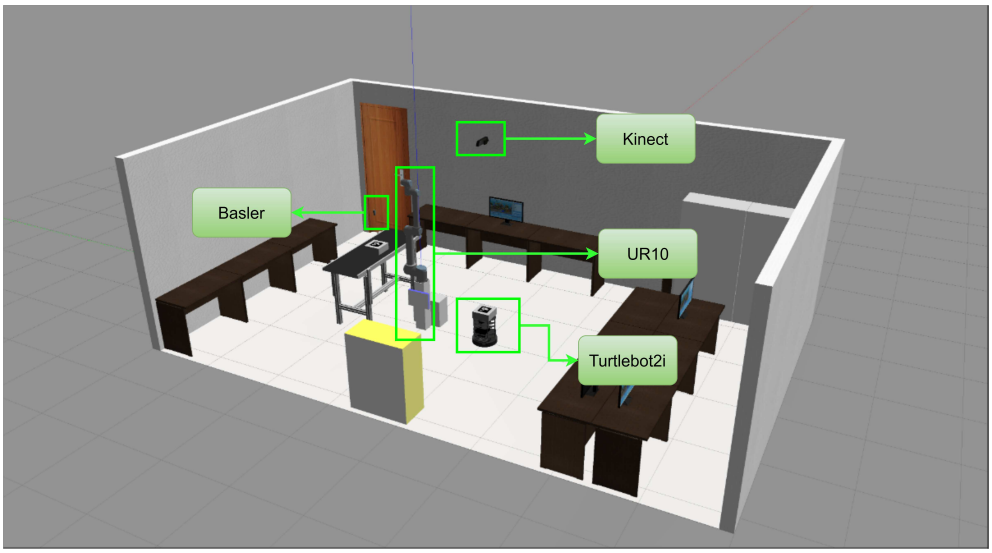
Figura 23 – Laboratório físico *Smart Factoring*.



Fonte: Autoria própria

A Figura 24 ilustra a configuração do ambiente simulado no Gazebo, destacando a disposição dos dispositivos e componentes principais da cena.

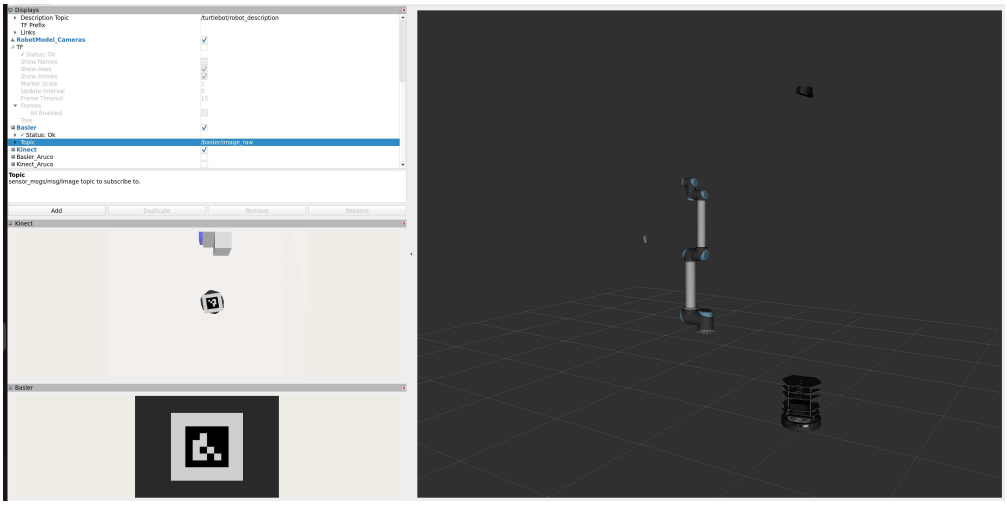
Figura 24 – Visualização da configuração do ambiente simulado no Gazebo



Fonte: Autoria própria

Por fim, a Figura 25 mostra a visualização da cena no *Rviz*.

Figura 25 – Visualização no *Rviz*.



Fonte: Autoria própria

4 Controle orientado a eventos com base em realimentação visual

Neste capítulo, é descrito o desenvolvimento teórico de um modelo de câmera em modo de transmissão contínua, baseado em estados e eventos, adequado para aplicações controladas por realimentação visual. Este modelo foi concebido especificamente para integração com o *Robot Operating System* (ROS), que oferece uma estrutura de programação distribuída ideal para o desenvolvimento de sistemas modulares e de controle orientado a eventos. A escolha do ROS permite que cada estado e evento do autômato seja mapeado hierarquicamente em uma árvore de comportamento, favorecendo um controle autônomo e distribuído. Essa arquitetura facilita a operação independente e modular da câmera, que se integra facilmente a uma estrutura de programação orientada a eventos, beneficiando-se das bibliotecas e dos mecanismos de comunicação do ROS.

4.1 Modelo da câmera baseado em estados e eventos

Esta seção aborda o desenvolvimento de um modelo teórico baseado em um SED, voltado para o controle de um dispositivo genérico de aquisição de imagem. O objetivo do desenvolvimento deste modelo é proporcionar uma abordagem metodológica para a inicialização de dispositivos de aquisição de imagem, organizando suas operações principais em uma sequência de ações e transições de estado que respondem a estímulos internos e externos.

O modelo de um dispositivo genérico de aquisição de imagem pode ser descrito pela séxtupla

$$G = (Q, \Sigma_e, \delta, \Gamma, q_0, Q_m),$$

onde $Q = \{S_1, S_2, S_3, S_4\}$ é o conjunto de estados, $\Sigma_e = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9\}$ é o conjunto de eventos e $\Sigma_{uc} = \{e_1, e_2, e_5, e_7, e_8, e_9\}$ representa os eventos não controláveis.

A função de transição $\delta : Q \times \Sigma_e \rightarrow Q$ define como o sistema transita entre estados em resposta aos eventos, enquanto $q_0 = S_1$ indica o estado inicial do sistema. O conjunto de estados marcados, $Q_m = \{S_3\}$, identifica o estado objetivo do sistema, que representa a condição desejada de operação contínua e bem-sucedida da câmera.

O sistema passa por quatro estados principais, conforme descrito na Tabela 5.

Tabela 5 – Estados do módulo de controle da câmera

Estado	**Descrição**
S_1	Câmera desligada, sem atividade operacional
S_2	Câmera ligada, pronta para operar, mas sem transmissão ativa
S_3	Câmera em operação contínua, capturando e transmitindo imagens
S_4	Estado de erro, indicando falha na transmissão ou problema de hardware

O conjunto de eventos que controla as transições entre esses estados é detalhado na Tabela 6. Eventos como iniciar a transmissão (e_3) e encerrar a transmissão (e_4) são controláveis pelo sistema ROS, pois ele consegue gerenciar essas operações diretamente no nível de *software*. Por outro lado, ligar (e_1) e desligar (e_2) a câmera – no sentido de conectar ou desconectar o fornecimento de energia, ou apertar um botão físico – são eventos não controláveis, assim como erros de inicialização (e_5) e durante a operação (e_7), que afetam o fluxo do sistema de forma autônoma.

Tabela 6 – Eventos de funcionamento da câmera

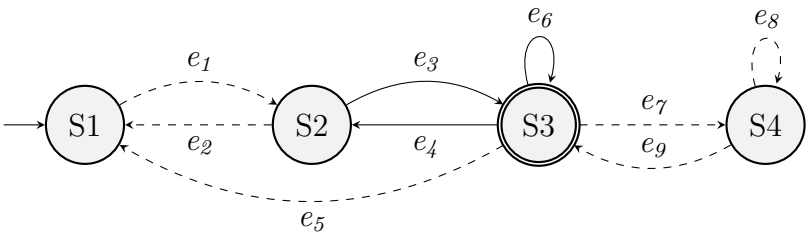
Evento	**Descrição**	**Controlável**
e_1	Ligar a câmera	×
e_2	Desligar a câmera	×
e_3	Iniciar transmissão contínua	✓
e_4	Encerrar transmissão contínua	✓
e_5	Erro durante inicialização	×
e_6	Reiniciar operação contínua	✓
e_7	Erro durante operação contínua	×
e_8	Persistência do erro	×
e_9	Transmissão reestabelecida	×

O comportamento básico de uma câmera pode ser descrito da seguinte forma:

1. inicialmente, a câmera está desligada e requer uma intervenção externa para ser ativada;
2. após ser ligada, a câmera entra em um estado de espera no qual está pronta para capturar imagens, mas aguarda um comando específico para iniciar a transmissão;
3. ao receber o comando de ativação, a câmera começa a transmitir imagens de forma contínua em um tópico do ROS;
4. durante a operação de captura, ou mesmo antes de iniciar a transmissão, podem ocorrer situações de erro, como falhas de conexão ou problemas de hardware, que interrompem a operação normal da câmera;
5. em caso de falha de conexão, a câmera retorna ao estado de espera, enquanto o ROS tenta automaticamente restabelecer a conexão (evento e_8);
6. se a câmera conseguir restabelecer a conexão, ela retorna à operação normal (evento e_9) e continua a transmissão de imagens de acordo com o comando recebido inicialmente.

A sequência de estados e transições desse ciclo operacional é formalizada no autômato ilustrado na Figura 26

Figura 26 – Autômato do modelo SED para um dispositivo de aquisição de imagem no ROS



4.2 Implementação do modelo da câmera em árvore de comportamento no ROS

Para garantir a adequada configuração e operação da câmera no ROS, foi desenvolvida uma árvore de comportamento baseada no modelo de estados e eventos, descrito na Seção 4.1, que se concentra exclusivamente na sequência de inicialização do dispositivo. Este modelo abrange desde a verificação inicial do estado de conexão da câmera até a configuração final de transmissão, assegurando que o dispositivo esteja pronto para a aplicação de visão computacional.

A escolha pela árvore de comportamento permite uma estrutura modular e flexível, favorecendo a reatividade e a recuperação de falhas, características fundamentais em sistemas de controle em tempo real (GUGLIERMO et al., 2024).

A árvore de comportamento proposta é organizada em três principais etapas hierárquicas, estruturadas em sequências e *fallbacks* (mecanismos de recuperação de falhas), como mostra a Figura 27. A estrutura da árvore inicia com o nó “Raiz”, que é um nó paralelo. Este nó permite a execução simultânea de todos os seus nós filhos e só retorna sucesso quando todos os filhos concluíram suas tarefas respectivas. Esse tipo de nó é usado para coordenar de maneira sincronizada as principais ações, garantindo que todas as etapas críticas sejam completadas antes de avançar.

No exemplo de operação deste sistema, a árvore de comportamento começa

com a "Sequência de Inicialização", onde o primeiro comportamento é a verificação do estado da câmera para confirmar se está ligada e reconhecida pelo sistema. Caso positivo, a árvore executa o pacote ROS responsável por iniciar a transmissão contínua de imagens. Cada nó é executado em sequência, de modo que, para a árvore avançar, todos devem retornar *SUCCESS*.

Após a inicialização, o sistema passa para a "Sequência de Operação de *Streaming*", que monitora o estado de transmissão contínua da câmera. Este nó verifica se o *streaming* está ativo; em caso de sucesso, o sistema continua a operação, e em caso de falha, o nó de *fallback* de "Gerenciamento de Erro"(mecanismo de recuperação de falha) é ativado para lidar com falhas de transmissão. Quando um erro é detectado, a árvore tenta uma reconexão automática ao pacote ROS, permitindo ao sistema retornar à operação normal se a conexão for restabelecida.

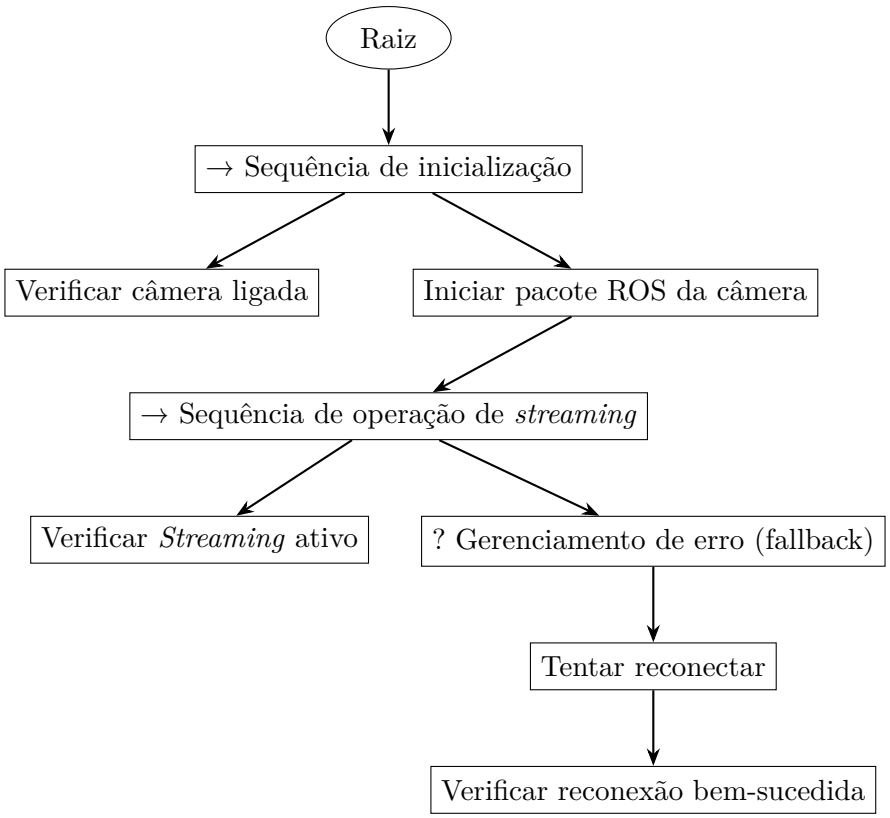


Figura 27 – Árvore de comportamento para controle de inicialização e operação de câmera no ROS

5 Resultados

Neste capítulo, são apresentados os resultados obtidos a partir da implementação do sistema de controle orientado a eventos com base em realimentação visual em uma fábrica inteligente, conforme descrito nas Seções 3 e 4. O sistema foi desenvolvido com uma estrutura de programação distribuída no ROS, permitindo que dispositivos e robôs trabalhem de forma coordenada e autônoma em resposta a eventos visuais. Para validação, foram configurados módulos de árvore de comportamento para o modelo de câmera e o sistema de detecção de marcadores ArUco, ambos operando no ambiente ROS. Uma tarefa experimental foi conduzida no laboratório *Smart Factory* para validar todo o sistema.

5.1 Módulo da câmera na árvore de comportamento

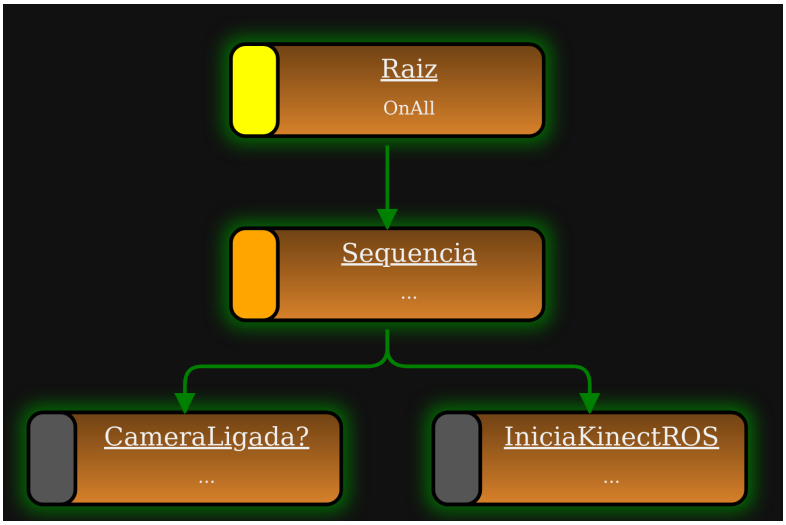
Utilizando a biblioteca `py_tree_ros`[1], foi implementada uma árvore de comportamento para o modelo de câmera. Para visualizar a árvore de comportamento em execução, utilizou-se a biblioteca adicional `py_tree_ros_viewer`[2], permitindo observar via interface gráfica a sequência e o estado de cada nó da árvore. A Figura 28 ilustra a árvore de comportamento utilizada para controlar o modelo de câmera no sistema de realimentação visual, e a Figura 29 mostra como os estados de cada nó são exibidos no terminal do sistema.

A estrutura da árvore é composta pelos seguintes nós principais:

- `Raiz (OnAll)`: representado pela cor amarela, este nó coordena a execução dos nós subsequentes. O tipo´´OnAlL" indica que todos os nós filhos devem ser ativados para que o controle prossiga.
- `Sequência`: representado pela cor laranja, este nó organiza o fluxo de execução

[1] Repositório py_tree_ros no GitHub: <https://github.com/splintered-reality/py_trees_ros>
[2] Repositório py_tree_ros_viewer no GitHub: <https://github.com/splintered-reality/py_trees_ros_viewer>

Figura 28 – Visualização na interface gráfica do modelo da câmera em árvore de comportamento aplicado ao Kinect



Fonte: Autoria própria

Figura 29 – Exibição dos estados dos nós da árvore de comportamento do módulo de câmera no terminal

```
[ INFO] CameraLigada?        : Kinect está ligado e pronto para transmitir.
[ INFO] IniciaKinectROS      : Nó do Kinect inicializado.
[INFO] [1730559462.860964850] [camera behavior tree node]: Câmera transmitindo.
```

Fonte: Autoria própria

em ordem sequencial, onde todos os nós filhos devem retornar sucesso para que o fluxo continue.

- `CameraLigada?`: nó de condição (cor cinza) que verifica se a câmera está ligada e pronta para transmitir. No contexto da árvore de comportamento, nós de condição como este verificam o estado sem alterar o sistema.

- `IniciaKinectROS`: também representado pela cor cinza, é um nó de ação que inicia o pacote ROS da câmera, permitindo que a câmera inicie a transmissão de imagens para o sistema.

5.2 Módulo de detecção de marcadores fiduciais na árvore de comportamento

Para validar que o modelo de câmera implementado em árvore de comportamento (Seção 5.1) pode ser configurado e utilizado como parte de um sistema, integrou-se um módulo para a detecção de marcadores ArUco, utilizando o pacote `aruco_pose_estimation`, conforme descrito na Seção 3. Este módulo permite que a câmera identifique e monitore a presença de marcadores ArUco no ambiente de trabalho, fornecendo informações essenciais para a execução de tarefas no laboratório com os demais dispositivos.

A configuração do módulo de detecção de ArUco incluiu uma árvore de comportamento própria, com estados de monitoramento, detecção e resposta a falhas, conforme ilustrado nas Figuras 30, 31 e 32. As bordas coloridas dos nós do módulo indicam visualmente o estado do processo de detecção:

- Borda Azul: Indica que o módulo de ArUco está ativo e monitorando o campo de visão.
- Borda Verde: Sinaliza uma detecção bem-sucedida do marcador ArUco, permitindo que o sistema transmita as informações de pose ao ROS.
- Borda Vermelha: Aponta uma falha na detecção, geralmente quando o marcador não está presente ou fora do campo de visão da câmera.

Para avaliar o tempo de detecção e publicação do sistema, foi realizada uma coleta de dados com um total de 235 amostras, representando o intervalo entre a detecção dos marcadores ArUco e a publicação das informações de pose no ROS. Os resultados revelaram uma média de aproximadamente 2,26 ms com uma variância de 0,25 ms.

Figura 30 – Estado de monitoramento do módulo de ArUco, indicado pela borda azul.

Fonte: Autoria própria

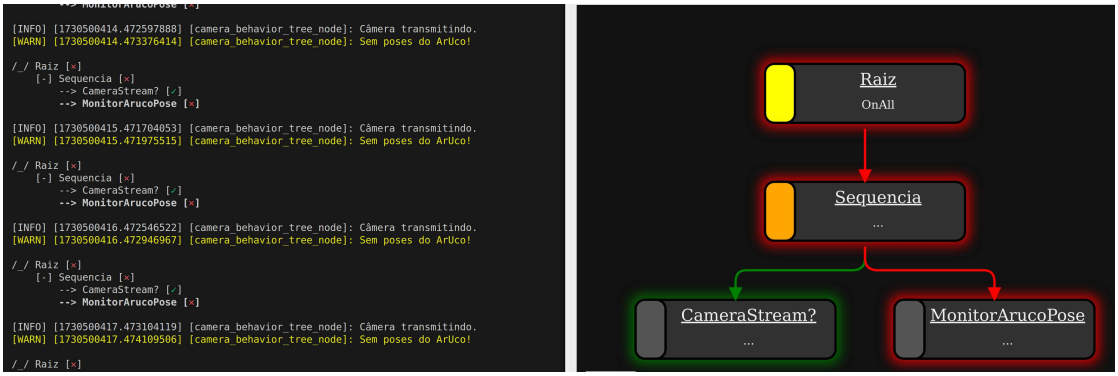
Figura 31 – Detecção bem-sucedida do marcador ArUco, indicada pela borda verde.

Fonte: Autoria própria

5.3 Validação do controle orientado a eventos com base em realimentação visual e ROS

Para validar o sistema de controle orientado a eventos com realimentação visual, utilizando o ROS e os módulos em árvore de comportamento descritos nas Seções 4.2 e 5.2, foi realizada uma tarefa experimental no laboratório *Smart Factory*. Nesta tarefa, a câmera atuou como sensor visual, detectando e identificando marcadores ArUco, cujas poses foram publicadas no ROS. A partir dessas informações, foi possível orientar o robô UR10 para manipular objetos. Vale ressaltar que o modelo

Figura 32 – Falha na detecção do marcador ArUco, indicada pela borda vermelha.



Fonte: Autoria própria

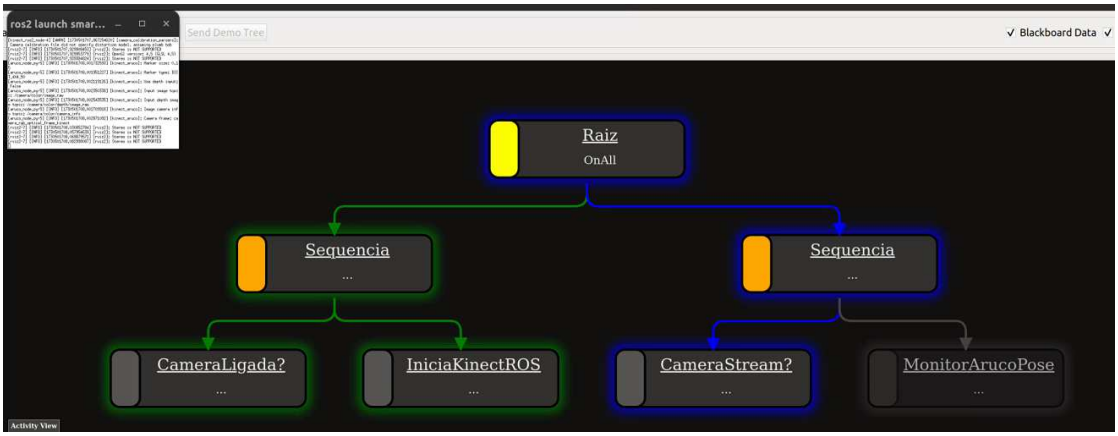
e a implementação do UR10 já estavam previamente configurados e operacionais, não estando dentro do escopo deste trabalho.

A estrutura em árvore de comportamento organizou, de forma hierárquica, os estados e ações dos módulos da câmera e do ArUco, possibilitando um controle modular e autônomo. A árvore monitora continuamente o estado da câmera para garantir que ela esteja transmitindo dados de forma contínua. Quando um marcador ArUco é detectado, o módulo de detecção registra a posição e a orientação do marcador e publica essas informações em um tópico específico no ROS. Dessa forma, a pose dos objetos é enviada para o UR10, permitindo o acesso a dados atualizados de posição e a execução de tarefas de manipulação.

Após a detecção do marcador ArUco, a pose do objeto é publicada no ROS, permitindo que o robô UR10 receba essas informações e execute a tarefa de manipulação. Com os dados de posição e orientação do marcador, o UR10 é capaz de se orientar para pegar o objeto, simulando uma operação comum em ambientes industriais automatizados.

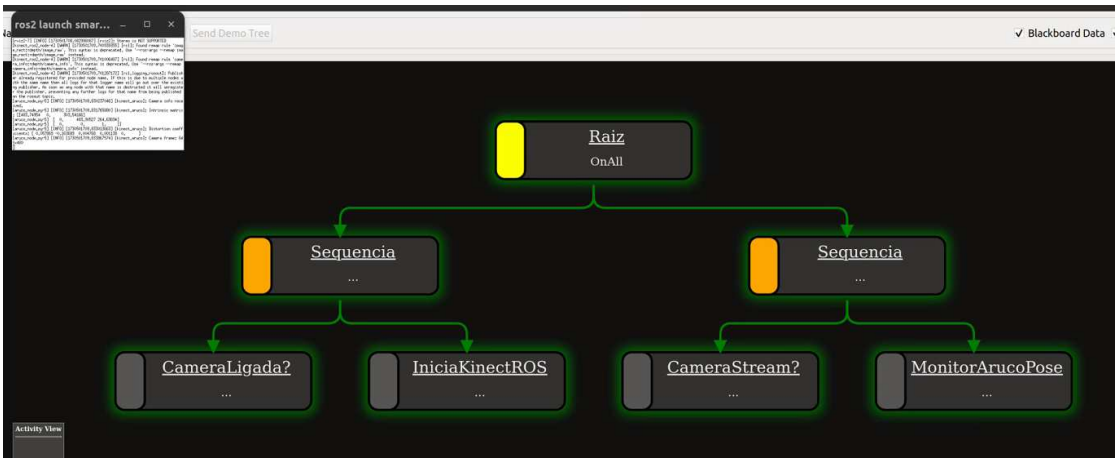
A Figura 35 mostra o momento em que o UR10 realiza a tarefa de manipulação, posicionando-se para pegar o objeto com o marcador ArUco, com base na pose identificada e recebida no ROS.

Figura 33 – Estrutura em árvore de comportamento com o estado da câmera (verde) e módulo ArUco (azul).



Fonte: Autoria própria

Figura 34 – Visualização do fluxo de controle com ambos os módulos no estado verde, indicando operação bem-sucedida.



Fonte: Autoria própria

Figura 35 – Robô UR10 executando a tarefa de manipulação com base na pose detectada do marcador ArUco.

Fonte: Autoria própria

5.4 Validação da realimentação visual e detecção de marcadores ArUco na simulação

Antes de integrar o modelo de câmera e o módulo de detecção de ArUcos no sistema final, foram realizados testes preliminares no ambiente simulado do Gazebo. Esses testes tiveram como objetivo validar a configuração da câmera e a eficácia na detecção dos marcadores ArUco.

No ambiente simulado, a câmera Kinect foi configurada para detectar e identificar marcadores ArUco em diferentes posições e orientações, replicando as condições esperadas no ambiente real. A detecção e a publicação das informações de pose dos marcadores no ROS ocorreram conforme esperado. Esse processo foi

visualizado no *Rviz*, conforme ilustrado na Figura 36.

Figura 36 – Detecção de marcador ArUco no ambiente simulado do gazebo exibido no *Rviz*

Fonte: Autoria própria

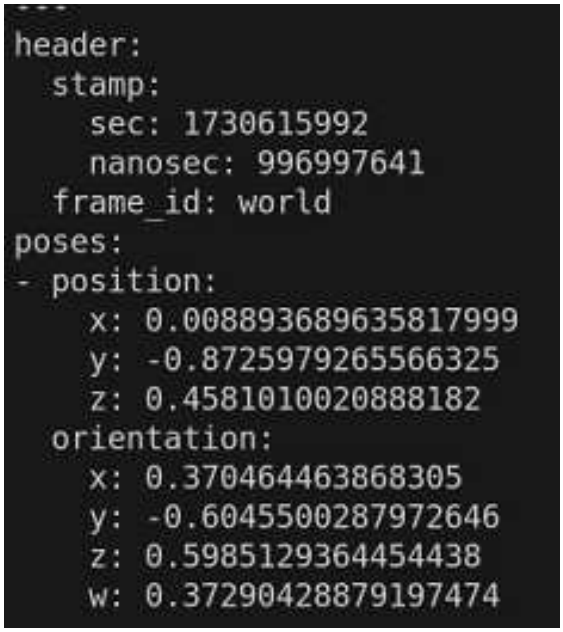
Na Figura 37 está ilustrado como o ArUco estava localizado no gazebo no momento do teste de detecção, observado na Figura 36.

Figura 37 – Detecção de Marcador ArUco no Ambiente Simulado do Gazebo

Fonte: Autoria própria

Durante a simulação, a câmera conseguiu detectar consistentemente o marcador e publicar sua pose no ROS, como mostra a Figura 38 com o resultado da detecção exibido no terminal.

Figura 38 – Resultado da detecção do marcador publicado no ROS



Fonte: Autoria própria

6 Conclusão

Neste trabalho, foi desenvolvido e implementado um sistema de controle orientado a eventos para uma fábrica inteligente, com base em realimentação visual e em uma estrutura de programação distribuída utilizando o *Robot Operating System* (ROS). O sistema, alinhado aos princípios da Indústria 4.0, foi projetado para integrar dispositivos de aquisição de imagem a sistemas robóticos, viabilizando uma resposta eficiente a eventos visuais.

A construção de modelos baseados em Sistemas a Eventos Discretos (SED) e em árvores de comportamento contribuiu para a modularidade do sistema, facilitando a integração com os demais dispositivos e permitindo que novos dispositivos de aquisição de imagem sejam adicionados de forma ágil por meio do módulo disponibilizado. Embora o foco deste trabalho esteja na integração de dispositivos de aquisição de imagens para o controle de tarefas, esse sistema faz parte de uma estrutura maior, que inclui outros dispositivos presentes no laboratório *Smart Factory*, descritos na Capítulo 3. Modelos de controle baseados em estados e eventos para esses dispositivos também foram considerados, mas seu desenvolvimento completo está fora do escopo deste trabalho.

Este sistema foi validado por meio de um experimento, no qual o Kinect monitora uma área de trabalho do robô UR10, transmitindo comandos e informações através de tópicos ROS, utilizando árvore de comportamento. Como demonstração, foi implementada na árvore, a detecção de marcadores ArUco transportados pelo TurtleBot 2i até a área de trabalho do UR10, confirmando a eficácia do sistema em responder a eventos visuais.

A simulação no Gazebo e os experimentos realizados no laboratório *Smart Factory* validaram a funcionalidade e adaptabilidade do sistema em cenários de fábrica inteligente. Além de representar um avanço no campo da automação industrial, este trabalho também contribui com uma aplicação prática para o laboratório, ao implementar e validar um modelo funcional de controle visual integrado a um ambiente de simulação e experimentação. Esse modelo serve de base para aprimo-

ramentos e novos experimentos, fortalecendo a infraestrutura tecnológica do *Smart Factory*.

6.1 Trabalhos futuros

Sugere-se que os seguintes aprimoramentos sejam considerados para continuidade e expansão do projeto:

- integração com aprendizado de máquina: incorporar técnicas de aprendizado de máquina para melhorar a precisão na detecção de objetos e na tomada de decisão do sistema, permitindo uma adaptação mais inteligente às variações do ambiente de produção;
- expansão do sistema para outros dispositivos: integrar novos sensores e dispositivos, como câmeras adicionais e sensores de proximidade, para ampliar a capacidade de monitoramento e controle em tempo real;
- otimização do modelo de árvore de comportamento: refinar o modelo de árvore de comportamento para melhorar a eficiência do controle e reduzir o tempo de resposta a eventos;
- desenvolvimento de um gêmeo digital do sistema: criar um gêmeo digital do sistema, permitindo simulações mais precisas e testes de estratégias de controle antes da aplicação no ambiente real.

A Repositório do Projeto e Colaborações

O código-fonte e as configurações desenvolvidas para este trabalho estão disponíveis em um repositório público no GitHub. Este repositório contém os módulos principais, bem como as configurações necessárias para replicação do sistema em outros ambientes.

Repositório: <https://github.com/SmartFactoryLab-UFCG/smartfactory_pkg>

Alguns módulos e funcionalidades de uso geral foram desenvolvidos em colaboração com outros membros do laboratório. As partes que não estão diretamente no escopo deste trabalho foram desenvolvidas por esses colaboradores, garantindo uma integração completa do sistema para os experimentos realizados.

Referências

ACKERMAN, E. State of the art in industry 4.0 and smart factory. *IEEE Spectrum*, v. 53, n. 8, p. 58–64, 2016. Citado na página 18.

BENOTSMANE, R.; KOVáCS, G.; DUDáS, L. Economic, social impacts and operation of smart factories in industry 4.0 focusing on simulation and artificial intelligence of collaborating robots. *Social Sciences*, v. 8, n. 5, p. 143, 2019. Citado 2 vezes nas páginas 19 e 20.

CASSANDRAS, C. G.; LAFORTUNE, S. *Introduction to discrete event systems*. [S.l.]: Springer, 2008. Citado 4 vezes nas páginas 29, 30, 31 e 32.

CHEN, B. et al. Smart factory of industry 4.0: Key technologies, application case, and challenges. *IEEE Access*, v. 6, p. 6505–6519, 2017. Citado 2 vezes nas páginas 15 e 18.

COLLEDANCHISE, M.; ÖGREN, P. *Behavior trees in robotics and AI: An introduction*. [S.l.]: CRC Press, 2018. Citado na página 32.

COMMONS, W. *Xbox 360 Kinect Standalone*. Accessed: 2024-10-31, available at https://commons.wikimedia.org/wiki/File:Xbox-360-Kinect-Standalone.png. Citado na página 39.

CORKE, P. *Robotics, Vision and Control: Fundamental Algorithms in Python*. Third edition. [S.l.]: Springer Nature Switzerland AG, 2023. v. 146. (Springer Tracts in Advanced Robotics, v. 146). ISBN 978-3-031-06468-5. Citado 8 vezes nas páginas 15, 20, 21, 23, 25, 27, 28 e 29.

DOE, J.; SMITH, J.; PEREZ, M. ROS-Industrial based robotic cell for Industry 4.0: Eye-in-hand stereo camera and visual servoing for flexible, fast, and accurate picking and hooking in the production line. *Journal of Robotics and Automation*, Elsevier, v. 35, n. 4, p. 234–245, 2020. Citado 2 vezes nas páginas 16 e 33.

EL-IAITHY, R. A.; HUANG, J.; YEH, M. Study on the use of microsoft kinect for robotics applications. *Proceedings of the IEEE Conference*, IEEE, p. 1280–1288, 2012. Citado 2 vezes nas páginas 38 e 39.

GAZEBO Sim - About. 2024. Disponível em: <https://gazebosim.org/about>. Citado na página 35.

GUGLIERMO, S. et al. Evaluating behavior trees. *Robotics and Autonomous Systems*, v. 178, p. 104714, 2024. Citado 2 vezes nas páginas 32 e 58.

HANSON, R.; SMITH, J. Digital factory technologies for robotic automation and enhanced manufacturing cell design. *Advanced Robotics and Automation*, v. 22, p. 88–99, 2021. Citado 2 vezes nas páginas 18 e 20.

HARTLEY, R.; ZISSERMAN, A. *Multiple View Geometry in Computer Vision*. 2nd. ed. Cambridge, UK: Cambridge University Press, 2004. ISBN 978-0521540513. Citado 3 vezes nas páginas 20, 21 e 22.

JASIńSKI, M.; IWANIEC, M.; TADEUSIEWICZ, R. Smart robots for smart factories. *Industry 4.0 Journal*, v. 18, p. 101–113, 2020. Citado 2 vezes nas páginas 19 e 20.

LIAO, Y. et al. Past, present and future of industry 4.0 - a systematic literature review and research agenda proposal. *International Journal of Production Research*, v. 55, p. 3609–3629, 2017. Citado 2 vezes nas páginas 16 e 19.

PAXTON, C. et al. Costar: Instructing collaborative robots with behavior trees and vision. *arXiv preprint arXiv:1611.06145*, 2016. Disponível em: <https://arxiv.org/abs/1611.06145>. Citado na página 33.

Quasi Robotics. *Behavior Trees for Autonomous Mobile Robots (AMRs)*. 2024. Acesso em: 2 nov. 2024. Disponível em: <https://www.quasi.ai/behavior-trees-for-autonomous-mobile-robots-amrs/>. Citado na página 33.

RAHMAN, M.; ALAM, S.; UDDIN, R. A review on intelligent manufacturing systems: The role of ai and iot. *Journal of Industrial Engineering*, v. 15, p. 34–47, 2023. Citado na página 16.

RAHMAN, M. T. et al. A comprehensive review of vision-based robotic applications: Current state, components, approaches, barriers, and potential solutions. *Robotics*, v. 11, n. 6, p. 139, 2022. Citado na página 19.

SIMONIč, M. et al. Modular ROS-based software architecture for reconfigurable, Industry 4.0 compatible robotic workcells. Ljubljana, Slovenia, p. 6–10, 2021. Citado 5 vezes nas páginas 15, 16, 33, 35 e 36.

SZELISKI, R. *Computer Vision: Algorithms and Applications*. [S.l.]: Springer Science & Business Media, 2010. ISBN 978-1848829343. Citado na página 21.

TurtleBot. *TurtleBot2i - Official Website*. 2023. Accessed: 2023-10-31. Disponível em: <https://turtlebot2i.weebly.com/>. Citado na página 42.

TurtleBot. *TurtleBot2i GitHub Repository*. 2023. Accessed: 2023-10-31. Disponível em: <https://github.com/turtlebot/turtlebot2i>. Citado na página 41.

TurtleBot. *TurtleBot2i Wiki - ROS Documentation*. 2023. Accessed: 2023-10-31. Disponível em: <https://wiki.ros.org/turtlebot2i>. Citado na página 41.

Universal Robots. *User Manual: CB3 Series Robots*. [S.l.], 2023. Accessed: 2023-10-31. Disponível em: <https://s3-eu-west-1.amazonaws.com/ur-support-site/18369/manual_en_1.3.pdf>. Citado 2 vezes nas páginas 40 e 41.

ZEMLA, F. et al. Modern trends in smart factories: Process optimization and data-driven solutions. *Journal of Automation and Control*, v. 45, p. 97–111, 2023. Citado 2 vezes nas páginas 15 e 19.

ÖGREN, P.; SPRAGUE, C. I. Behavior trees in robot control systems. *Annual Review of Control, Robotics, and Autonomous Systems*, v. 5, p. 1–25, 2022. Citado na página 32.