

**UNIVERSIDADE FEDERAL DE JUIZ DE FORA
FACULDADE DE ENGENHARIA ELÉTRICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

Ariéle Teixeira Ferraz

**Controle de Atuador Robótico Através do Rastreamento Ocular por Câmera
para Assistência de Pessoas com Paralisia Muscular**

Juiz de Fora

2024

Ariéle Teixeira Ferraz

**Controle de Atuador Robótico Através do Rastreamento Ocular por Câmera
para Assistência de Pessoas com Paralisia Muscular**

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Sistemas de Energia

Orientador: Prof. Dr. André Luís Marques Marcato

Coorientador: Prof. Dr. Vinícius Barbosa Schettino

Juiz de Fora

2024

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Ferraz, Ariéle Teixeira.

Controle de Atuador Robótico Através do Rastreamento Ocular por Câmera para Assistência de Pessoas com Paralisia Muscular / Ariéle Teixeira Ferraz. – 2024.

82 f. : il.

Orientador: André Luís Marques Marcato

Coorientador: Vinícius Barbosa Schettino

Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Faculdade de Engenharia Elétrica. Programa de Pós-Graduação em Engenharia Elétrica, 2024.

1. Tecnologias assistivas. 2. Robótica. 3. Manipuladores. 4. *Eye-tracking*. 5. *Pick and place* I. Marcato, André L. M. II. Schettino, Vinícius B. . III. Controle de atuador robótico através do rastreamento ocular por câmera para assistência a pessoas com paralisia muscular.

Ariéle Teixeira Ferraz

**Controle de Atuador Robótico Através do Rastreamento Ocular por Câmera
para Assistência de Pessoas com Paralisia Muscular**

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Sistemas de Energia

Aprovada em 15 de março de 2024

BANCA EXAMINADORA

Prof. Dr. André Luís Marques Marcato - Orientador
Universidade Federal de Juiz de Fora

Prof. Dr. Vinícius Barbosa Schettino - Coorientador
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Murillo Ferreira dos Santos
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Ivo Chaves da Silva Jr
Universidade Federal de Juiz de Fora

Dr. Iago Zanuti Biundini
Universidade Federal de Juiz de Fora

Juiz de Fora, 06/12/2023.



Documento assinado eletronicamente por **Andre Luis Marques Marcato, Professor(a)**, em 15/03/2024, às 11:49, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Iago Zanuti Biundini, Usuário Externo**, em 15/03/2024, às 11:50, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Murillo Ferreira dos Santos, Usuário Externo**, em 15/03/2024, às 11:50, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Ivo Chaves da Silva Junior, Professor(a)**, em 15/03/2024, às 11:52, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Vinicius Barbosa Schettino, Usuário Externo**, em 15/03/2024, às 11:54, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **1612411** e o código CRC **9AD2A42B**.

Foi pensando nas pessoas com necessidades especiais, e nas possibilidades de melhoria de qualidade de vida que o avanço tecnológico proporciona à estes indivíduos, que executei este projeto, por isso dedico este trabalho a todos aqueles a quem esta pesquisa possa ajudar de alguma forma.

AGRADECIMENTOS

Agradeço primeiramente à Universidade Federal de Juiz de Fora, essencial para o desenvolvimento da pesquisa que possibilitou a realização deste trabalho. Agradeço também a quem colaborou diretamente comigo: meu orientador, o Professor Dr. André Luís Marques Marcato, e meu co-orientador, Professor Dr. Vinícius Barbosa Schettino, pelas correções e ensinamentos sem os quais eu não teria concluído este projeto. À minha família, em especial meus pais, o Professor Dr. Roberto Lopes Ferraz e Maria da Consolação Teixeira Ferraz, pois é graças ao seu esforço que hoje posso concluir meu mestrado. Aos profissionais da saúde, o médico psiquiatra Fernando Januário da Silva e a psicóloga Luciene Soares Vilela, que me ajudaram em meus momentos mais difíceis e me auxiliaram psicologicamente pra que eu consiga finalizar minha pesquisa com sucesso. À Alexandra Elbakyan, cujo trabalho auxilia tantos pesquisadores ao redor do mundo. Finalmente, aos meus colegas do Centro Espacial ITA e do Laboratório CONCEPTIO, que me acompanharam e apoiaram durante o final da concepção deste trabalho.

Comunicação aberta é propriedade fundamental da ciência e torna o progresso científico possível. (ALEXANDRA ELBAKYAN, 2021)

RESUMO

O avanço tecnológico permite desenvolver métodos de assistência cada vez mais avançados. Estas tecnologias assistivas tem potencial para aumentar o conforto de pessoas com deficiências físicas. Dentro deste contexto, o desenvolvimento de métodos capazes de identificar comandos do usuário e realizar o controle de atuadores robóticos se mostra um campo interessante. Uma grande parcela das pesquisas nesta área utilizam tanto métodos com equipamentos de prateleira, ou COTS (do inglês *Commercial Off-The-Shelf*), quanto equipamentos especificamente projetados para o propósito, como eletroencefalogramas (EEG). Neste trabalho propõe-se um método que seja simples de se utilizar, não seja invasiva (sem EEG) e de menor custo possível. Logo, é apresentada uma metodologia de identificação de comandos do usuário a partir do movimento dos olhos utilizando de uma *webcam* e realizado o controle de um atuador robótico a partir destes comandos. Embora tenha sido identificado na literatura outros trabalhos com propósito similar, busca-se, aqui, uma alternativa de baixo custo de desenvolvimento. O trabalho foi confeccionado utilizando o sistema operacional Ubuntu 20.04 com Noetic ROS e os pacotes RT-GENE, utilizado para a obtenção do rastreamento ocular e detecção de piscadas, e YOLO, visando detectar os objetos disponíveis para o usuário. O pacote python Tkinter foi utilizado para desenvolver a interface mostrada para o usuário. Por fim, os pacotes *MoveIt*, Gazebo e *franka_ros* foram utilizados para realizar as simulações de *pick and place* do manipulador. São apresentados ainda resultados de simulação utilizando a metodologia proposta, sendo possível verificar que a mesma possui potencial de continuidade na pesquisa, além da possibilidade de avaliação em ambiente real. Os testes com o robô Youbot revelaram desafios devido à necessidade de precisão e ao acúmulo de erros ao longo do tempo. No entanto, os testes com o manipulador Panda mostram que a resolução de uma webcam comum permitiu determinar a posição do objeto no mundo, embora o robô tenha apresentado dificuldades em executar sua tarefa para algumas posições. Apesar dos desafios, foi possível contornar o problema com uma implementação simples de tentativa e erro. Trabalhos futuros pretendem implementar outros modelos de robôs e realizar testes com robôs reais, além do acréscimo de sensoriamento no robô para garantir maior precisão em suas tarefas e evitar acidentes com pessoas que estejam em sua área de trabalho.

Palavras-chave: Tecnologias assistivas. Robótica. Manipuladores. Rastreamento Ocular. Pegar e mover.

ABSTRACT

Technological advancements have paved the way for the development of increasingly sophisticated assistive methods. These technologies hold the potential to significantly improve the comfort of individuals with physical disabilities. In this context, the creation of methods that can identify user commands and control robotic actuators has emerged as an intriguing field of study. A substantial portion of research in this area employs both Commercial Off-The-Shelf (COTS) equipment and devices specifically designed for the purpose, such as electroencephalograms (EEG). This study proposes a user-friendly, non-invasive, and cost-effective method. Consequently, a methodology for identifying user commands through eye movement using a webcam and controlling a robotic actuator based on these commands is introduced. While similar works have been identified in the literature, this study seeks a low-cost development alternative. The work was conducted using the Ubuntu 20.04 operating system with Noetic ROS, and the RT-GENE packages, which were used to obtain eye tracking and blink detection, and YOLO, which aimed to detect the objects available to the user. The Python Tkinter package was employed to develop the user interface. Finally, MoveIt, Gazebo and franka_ros packages were utilized to perform the pick and place simulations of the manipulator. Simulation results using the proposed methodology are also presented, demonstrating its potential for ongoing research and real-world evaluation. Tests with the Youbot robot revealed challenges due to the need for precision and the accumulation of errors over time. However, tests with the Panda manipulator showed that the resolution of a standard webcam allowed for the determination of the object's position in the world, despite the robot encountering difficulties in performing its task for some positions. Despite these challenges, it was possible to overcome the problem with a simple trial and error implementation. Future work aims to implement other robot models and conduct tests with real robots, in addition to adding sensing to the robot to ensure greater precision in its tasks and prevent accidents with people who are in its work area.

Keywords: Assistive technologies. Robotics. Manipulator. Eye-tracking. Pick and place.

LISTA DE ILUSTRAÇÕES

Figura 1 - Representação de um manipulador com três graus de liberdade	18
Figura 2 -Exemplo de Mínimos Quadrados	21
Figura 3 - Mecanismos de locomoção utilizado em sistemas biológicos	22
Figura 4 - Representação cinemática de um robô omnidirecional com quatro rodas suecas de 45 graus	25
Figura 5 - Resumo da arquitetura do modelo RCNN	26
Figura 6 - Resumo da arquitetura do modelo YOLO	27
Figura 7 - Exemplo de rastreador ocular baseado em tela	29
Figura 8 - Exemplo de rastreador ocular usável	29
Figura 9 - Funcionamento do ROScore para comunicação em tópicos	37
Figura 10 - Esquema de comunicação publisher/subscriber	38
Figura 11 - Esquema de comunicação cliente/servidor	38
Figura 12 - Robô Youbot da empresa KUKA, já descontinuado	40
Figura 13 - Robô Panda produzido pela empresa Franka Emika	41
Figura 14 - Imagem de referência da detecção do YOLO contendo a identificação de 3 itens: um cachorro, uma bicicleta e um caminhão (truck)	42
Figura 15 - Saída da detecção realizada na imagem 14 contendo os valores de confiança para cada item identificado	43
Figura 16 - Visão geral do funcionamento do RT-GENE	44
Figura 17 - Remoção dos óculos pelo RT-GENE <i>Inpainting</i>	45
Figura 18 - Subconjuntos do dataset para o RT-BENE	46
Figura 19 - Esquema da montagem das duas câmeras utilizadas no trabalho	47
Figura 20 - Fluxo simplificado da metodologia	48
Figura 21 - Setup completo do aplicativo desenvolvido no trabalho em execução	49
Figura 22 - Fluxograma de execução dos <i>scripts</i> para a implementação do <i>setup</i> completo	50
Figura 23 - Esquema para obtenção da posição do item	53
Figura 24 - Disposição dos blocos para realizar a calibração manual	54
Figura 25 - Tela inicial do aplicativo desenvolvido no trabalho	56
Figura 26 - Exemplo de posicionamento do robô Youbot para largar um item em sua base móvel	58
Figura 27 - Sequência de imagens mostrando a ação de <i>pick and place</i>	62
Figura 28 - Planejamento de caminho com restrição planar	64
Figura 29 - Planejamento de caminho com restrição em linha	64
Figura 30 - Resultado do YOLO considerando a disposição dos três itens na mesa	67
Figura 31 - Cenário utilizado para avaliar o desempenho dos três algoritmos utilizados pna ação de <i>pick and place</i>	69
Figura 32 - Imagem do YOLO com a disposição dos três itens na mesa	72

LISTA DE TABELAS

Tabela 1 – Diferentes tipos de rodas e seus graus de liberdade	24
Tabela 2 – Trabalhos Relacionados	34
Tabela 3 – Valores e erros dos eixos x e y para as simulações de <i>pick and place</i> utilizando o mesmo item.	67
Tabela 4 – Valores e erros dos eixos x e y para as simulações de <i>pick and place</i> utilizando itens diferentes.	67
Tabela 5 – Desempenho dos algoritmos <i>pick and place</i> e seu tempo de execução médio e desvio padrão para 50 simulações	70
Tabela 6 – Valores e erros dos eixos x e y para as simulações de <i>pick and place</i> utilizando itens diferentes	72
Tabela 7 – Tempo de execução do planejamento para obtenção de 100% do trajeto	73
Tabela 8 – Tempo de execução da seleção dos botoes do aplicativo para 25 simulações	75

LISTA DE ABREVIATURAS E SIGLAS

CNN	<i>Convolutional Neural Network</i>
COTS	<i>Commercial off-the-shelf</i>
DoF	<i>Degree of Freedom</i>
EEG	Eletroencefalograma
GUI	<i>Graphical User Interface</i>
IHR	Interface Humano-Robô
IR	Infravermelho
MTCNN	<i>Multi-Task Cascaded Convolutional Networks</i>
ROS	<i>Robot Operating System</i>
SDK	<i>Software development kit</i>
YOLO	<i>You Only Look Once</i>

LISTA DE SÍMBOLOS

:	Definido Como
\rightarrow	De ... Para
\mathbb{R}	Números Reais
\pm	Mais ou Menos
\in	Pertence a

SUMÁRIO

1	INTRODUÇÃO	16
1.1	CONTRIBUIÇÕES E ORGANIZAÇÃO DO TRABALHO	16
2	FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS	18
2.1	MANIPULADORES ROBÓTICOS	18
2.1.1	Cinemática Direta e Inversa	19
2.1.2	Singularidade Cinemática	20
2.2	ROBÓTICA MÓVEL	21
2.2.1	Controlabilidade e Manobrabilidade	23
2.2.2	Robôs Holonômicos	23
2.3	DETECÇÃO DE OBJETOS	25
2.4	RASTREAMENTO OCULAR	28
2.5	TRABALHOS RELACIONADOS	31
3	INTEGRAÇÃO DE FERRAMENTAS	36
3.1	ROS	36
3.2	GAZEBO	38
3.3	MOVEIT	39
3.4	YOUBOT	39
3.5	FRANKA PANDA	40
3.6	DARKNET YOLO	41
3.7	RT-GENE E RT-BENE	43
3.7.1	RT-GENE	44
3.7.2	RT-BENE	45
4	METODOLOGIA	47
4.1	OBTENÇÃO DOS OBJETOS EM CENA	50
4.2	OBTENÇÃO DA POSIÇÃO DO OLHAR	51
4.2.1	Calibração	51
4.2.2	Rastreamento Ocular	52
4.3	OBTENÇÃO DAS POSIÇÕES DOS OBJETOS EM CENA	52
4.3.1	Seleção de Objeto	54
4.4	APLICATIVO TKINTER	55
4.5	SIMULAÇÃO NO GAZEBO	55
4.6	ALGORITMO PICK AND PLACE	57
4.6.1	Planejamento básico com o <i>MoveIt</i>	57
4.6.2	Planejamento de caminhos cartesianos	59
<i>4.6.2.1</i>	<i>Ação pick</i>	60
<i>4.6.2.2</i>	<i>Ação place</i>	60

4.6.3	Planejamento Restritivo	61
5	RESULTADOS E DISCUSSÕES	66
5.1	YOUBOT	66
5.1.1	Mesmo item em posições diferentes	66
5.1.2	Mais de um item	67
5.2	PANDA	68
5.2.1	Desempenho dos Algoritmos Pick and Place	69
5.2.2	Efeito do posicionamento do objeto a ser pego	72
5.2.3	Planejamento sucessivo do trajeto	73
5.3	EFICIÊNCIA DAS DIFERENTES INTERFACES DE SELEÇÃO	74
6	CONCLUSÕES	76
6.1	TRABALHOS FUTUROS	77
	REFERÊNCIAS	78
	APÊNDICE A – Vídeos	82

1 INTRODUÇÃO

Tecnologias assistivas são produtos, recursos, metodologias, práticas e serviços que objetivam promover a funcionalidade relacionada à atividade e à participação de pessoas com deficiência ou mobilidade reduzida, visando sua autonomia, qualidade de vida e inclusão social (MANJARI; VERMA; SINGAL, 2020). Uma destas limitações físicas é a tetraplegia, uma paralisia que afeta todas as quatro extremidades do corpo, superiores e inferiores, juntamente à musculatura do tronco (GEWEHR; ANTONIUCCI; RODRIGUES, 2018).

Sabe-se o quanto a tetraplegia secundária à lesão medular compromete a qualidade de vida dos indivíduos afetados, tanto por sua instalação súbita quanto pela grave redução das habilidades pessoais (MUNCE et al., 2013). A perda de funcionalidade motora gera vulnerabilidades específicas, afetando a percepção de corporeidade, independência física e autonomia das pessoas, além de provocar privação social, cultural, psicológica e física (BALDASSIN; LORENZO; SHIMIZU, 2018).

Avanços recentes no campo da robótica oferecem novas promessas para auxiliar pessoas com diferentes habilidades, embora ainda seja um desafio criar uma interface entre humano e máquina para pessoas com deficiências graves (SHARMA et al., 2020). Um desses avanços é o rastreamento ocular, do inglês *eye tracking*. Esta é uma técnica que detecta o movimento do olhar em relação à posição da cabeça, ou seja, o ponto para onde uma pessoa está olhando (FISCHER; CHANG; DEMIRIS, 2018). Ultimamente, técnicas como o *eye tracking* têm sido implementadas para auxiliar o usuário a controlar diretamente uma aplicação. As interfaces controladas pelo olhar têm sido utilizadas para pessoas com deficiência motora grave, que não podem usar modalidades tradicionais de interação de um indivíduo com uma interface gráfica, ou seja, periféricos de computador já existentes, como *mouses*, *touchpads* ou teclados (SHARMA et al., 2020).

O uso de modalidades não tradicionais para interação entre pessoas e robô através de uma Interface Humano-Robô (IHR) não é um conceito novo, e embora seja usado com o intuito de auxiliar pessoas com deficiência, continua sendo um desafio. Um exemplo de IHR é o trabalho de Bannat et al. (2009) em que utilizaram comandos de voz, rastreamento ocular e botões virtuais para controlar um robô em um processo de montagem. Apesar de utilizarem uma interação humano-máquina para comandar um manipulador para fazer tarefas em uma mesa, o trabalho ainda utiliza outros tipos de comando além do rastreamento ocular e orientação de cabeça.

1.1 CONTRIBUIÇÕES E ORGANIZAÇÃO DO TRABALHO

A principal contribuição desta pesquisa é o uso da identificação de comandos do usuário a partir do movimento dos olhos ou orientação de cabeça utilizando duas

câmeras comuns para possibilitar o controle de um braço robótico. A tecnologia *eye tracking* implementada, já estabelecida em Fischer, Chang e Demiris (2018), utiliza a *webcam* comum de um notebook para identificar os comandos do usuário, o possibilitando comandar o aplicativo proposto neste trabalho para a realização de atividades de pegar e posicionar (*pick and place*).

O controle de um atuador robótico, neste caso o manipulador Panda da empresa Franka Emika, mostrado na Figura 13, necessita do conhecimento das posições dos objetos disponíveis em cena. Deste modo, um algoritmo de identificação de objetos bastante conhecido na literatura (YOLO) utiliza-se de uma câmera RGB comum montada no ambiente para extrair características dos itens, obtendo assim suas respectivas posições para que o manipulador possa agarrá-los. As contribuições desta pesquisa podem ser resumidas em:

- Propor uma metodologia e desenvolvimento de um novo algoritmo para, através da identificação da posição dos olhos, realizar controle de um atuador a partir desta posição;
- Organizar um ambiente de simulação que permita a avaliação da mesma;
- Realizar simulações de maneira a determinar vantagens e desvantagens do método em relação à literatura.

Em outubro de 2023, a LARS e SBR (*Latin American Robotics Symposium e Brazilian Robotics Symposium*), simpósios visando promover a divulgação de robótica inteligente, ocorreram no Centro de Convenções de Salvador (CCS) em Salvador - Bahia. Neste evento, um artigo referente ao trabalho desenvolvido nesta dissertação foi apresentado oralmente e publicado. Tal publicação descreve brevemente o desenvolvimento desta pesquisa, realizando, também, uma análise mais sucinta dos resultados. O artigo está disponível de forma paga no *IEEE Xplore* (FERRAZ et al., 2023), podendo ser acessado através do link <<https://doi.org/10.1109/LARS/SBR/WRE59448.2023.10333033>>.

No restante desta dissertação são apresentadas cada etapa do desenvolvimento. Uma breve apresentação do estado da arte é apresentado no Capítulo 2. A metodologia e as ferramentas utilizadas durante a confecção do trabalho estão disponíveis no Capítulo 3. O Capítulo 4 descreve o desenvolvimento e funcionamento dos algoritmos feitos para realizar as simulações. No Capítulo 5 serão apresentados os resultados observados e realizada uma breve discussão sobre os mesmos. Ao final, as conclusões e trabalhos futuros serão apresentadas no Capítulo 6.

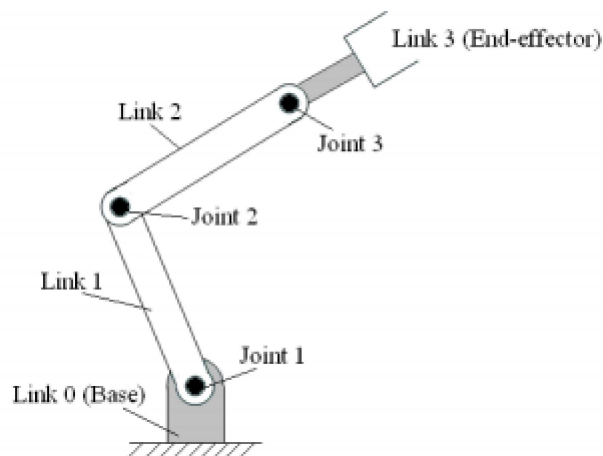
2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

A aplicação proposta nesta dissertação exige a implementação de diversos conceitos técnicos sobre robótica móvel, manipuladores industriais e processamentos de imagens, como as técnicas de identificação de objetos e rastreamento ocular. Este capítulo pretende estabelecer a fundamentação teórica necessária para possibilitar a realização deste trabalho, além de comparar trabalhos relacionados à implementação de técnicas de rastreamento ocular para controlar um robô, especialmente braços robóticos.

2.1 MANIPULADORES ROBÓTICOS

Os manipuladores robóticos são robôs versáteis concebidos e programados para operar uma variedade de ferramentas ou dispositivos, a fim de desempenhar tarefas específicas. Estes robôs têm uma base fixa e se assemelham ao braço humano, permitindo que interajam com o ambiente em que estão inseridos por meio de ferramentas localizadas em suas extremidades, conhecidas como efetadores ou *end-effectors* em inglês. Um exemplo de manipulador e sua constituição física é ilustrado na Figura 1.

Figura 1 - Representação de um manipulador com três graus de liberdade



Fonte: Reddy, A. Chennakesava (REDDY, 2014).

Esses equipamentos consistem em elos e juntas. Os elos ou *links* representam as partes rígidas do manipulador robótico destinadas a conectar as juntas. Devido sua rigidez, não realizam nenhum tipo de movimento, mas como interligam duas juntas, sua posição influencia no movimento destas articulações, de modo que os elos unidos por juntas formem o corpo da cadeia cinemática do robô.

As juntas, do inglês *joints*, são as partes móveis do robô. São elas que estabelecem conexões móveis entre os elos e realizam os movimentos programados para cada tarefa de modo a atingir um determinado ponto no espaço conforme a limitação do manipulador, modificando sua posição. As duas principais classificações de juntas são rotação/revolução

(R) ou prismática/de translação (P), de modo que outros tipos de juntas podem ser descritas como combinações dessas duas.

Os Graus de Liberdade, também conhecidos como *Degrees of Freedom* (DoF) em inglês, presentes em um manipulador, determinam a quantidade de variáveis independentes necessárias para definir ações específicas, as quais devem ser precisamente determinadas para obter as posições de todas as partes do mecanismo. O número de juntas existentes diretamente influencia nos Graus de Liberdade que o manipulador possui.

O termo em questão (Graus de Liberdade) pode ser aplicado a qualquer tipo de mecanismo. Em relação a robôs industriais, ou manipuladores robóticos, que representam uma cadeia cinemática aberta, ou seja, o dispositivo que realiza o movimento é livre no espaço e o posicionamento de cada junta pode ser especificado por uma única variável, o número de juntas equivale ao número de graus de liberdade (CRAIG, 2005). Na Figura 1, é apresentado um exemplo de manipulador com três juntas, cada uma contribuindo com um grau de liberdade, categorizando o robô como tendo três DoF.

Para que um manipulador possa posicionar sua ferramenta em qualquer ponto dentro do seu espaço de trabalho tridimensional e mantê-la com a orientação desejada, são necessários, no mínimo, seis graus de liberdade. Dessas seis juntas, três são responsáveis exclusivamente por posicionar o efetuador, enquanto as outras três são empregadas para orientar o objeto conforme seu próprio sistema de coordenadas. Manipuladores que apresentam mais de seis graus de liberdade são considerados cinematicamente redundantes (SICILIANO et al., 2010).

2.1.1 Cinemática Direta e Inversa

Cinemática é a ciência que descreve os movimentos dos corpos sem levar em conta as forças que originaram esses movimentos. O estudo da cinemática dos manipuladores refere-se a todas as propriedades geométricas e baseadas no tempo do movimento (CRAIG, 2005). Para os estudos com manipuladores robóticos, a cinemática direta é utilizada para obter a posição e orientação final do efetuador em função dos ângulos, já conhecidos, de cada uma de suas juntas.

A cinemática inversa, por sua vez, realiza a operação contrária. Por meio da configuração cartesiana final pré-determinada (desejada) para o efetuador, realizam-se cálculos para obter os parâmetros necessários para cada junta, de modo a alcançar seu objetivo.

A solução destes problemas é de fundamental importância a fim de possibilitar a execução do movimento desejado. Na cinemática direta, os valores dos ângulos das juntas já são conhecidos, bastando apenas aplicar as transformações homogêneas de translação e rotação. Para a cinemática inversa, a situação é bem mais complexa (SICILIANO et al., 2010), pois:

- As equações normalmente são não-lineares;
- Podem existir múltiplas ou infinitas (em caso de manipuladores redundantes) soluções;
- Pode não haver solução para o objetivo escolhido.

A existência de soluções são garantidas apenas para posições que se encontram no espaço de trabalho do manipulador escolhido (SICILIANO et al., 2010). Algumas abordagens para a solução da cinemática inversa se dão por meio de métodos analíticos, apesar de não serem métodos amplamente utilizados devido à complexidade de sua modelagem e equacionamento, e o fato de que não há uma garantia de seja possível aplicá-lo para todos os tipos de manipuladores.

Por outro lado, os métodos numéricos iterativos convergem para uma possível solução, mas apesar de serem mais generalistas, são computacionalmente mais pesados, exigindo uma capacidade maior de processamento. Os métodos iterativos utilizam cálculos baseados na cinemática direta e na matriz Jacobiana inversa para obter uma solução (OLIVI, 2019a).

Uma técnica numérica para a obtenção uma solução do problema da cinemática inversa é o Gradiente Descendente. Este é um método de otimização de funções objetivo, funções matemáticas de uma ou mais variáveis que se deseja otimizar, minimizando ou maximizando seu valor conforme o objetivo esperado. Para o caso de manipuladores robóticos, deseja-se encontrar seus parâmetros, assim, determina-se uma função objetivo que nos forneça a resposta desejada.

Uma boa alternativa para a função objetivo é a utilização de Mínimos Quadrados aplicado na função de erro, exibida na Equação 2.1, pois esta função de erro quadrático é convexa, possuindo um único mínimo, conforme a Figura 2. Assim, a mesma pode ser otimizada através do método de Gradiente Descendente sem que se incorra em problemas de convergência:

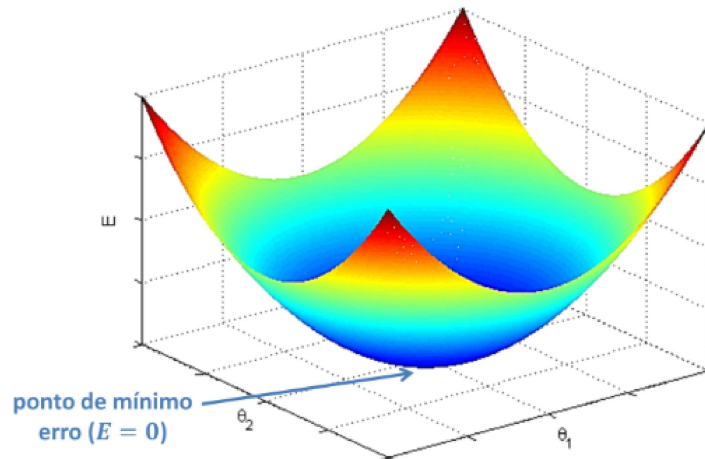
$$E = (\theta_{desejado} - \theta_{calculado})^2. \quad (2.1)$$

Onde $\theta_{desejado}$ representa a configuração desejada, ou seja, a posição final do efetuador e, caso necessário, sua orientação esperada. Em contrapartida, $\theta_{calculado}$ representa a posição e orientação calculada pelo processo de cinemática inversa.

2.1.2 Singularidade Cinemática

Conforme descrito na Seção 2.1.1, normalmente utilizam-se métodos iterativos para o cálculo da cinemática inversa sendo estes, por sua vez, baseados na formulação da cinemática direta e na matriz jacobiana. Geralmente esta matriz Jacobiana é uma função referente à configuração do manipulador, com suas equações cinemáticas linearmente

Figura 2 -Exemplo de Mínimos Quadrados



Fonte: Notas de aula (OLIVI, 2019a).

independentes, e de tamanho $m \times n$, onde m representa o grau de liberdade do efetuador do manipulador e n reflete o número de juntas do robô. Desta forma, em casos no qual a jacobiana não tem posto completo, o efetuador perde um ou mais graus de liberdade, caracterizando assim como uma configuração singular.

As posições singulares, por perderem graus de liberdade, devem sempre ser evitadas, pois representam configurações em que a mobilidade do braço se reduz, inviabilizando a livre movimentação do manipulador. Além da configuração singular em si, sua vizinhança também deve ser evitada, pois, como a matriz jacobiana transforma vetores de deslocamento infinitesimal de um sistema de coordenadas para outro, sua inversão pode resultar em valores de velocidade de juntas grandes, muitas vezes fisicamente impossíveis para o deslocamento do manipulador.

Existem na literatura alguns métodos para se contornar tal problema, como o Jacobiano pseudo-inverso (BUSS, 2004) e o método da inversa filtrada (VARGAS; LEITE; COSTA, 2014), mas como este não é foco da dissertação, tais procedimentos não serão abordados.

2.2 ROBÓTICA MÓVEL

A robótica móvel é uma área da robótica dedicada ao desenvolvimento de robôs que não possuem base fixa, ou seja, com capacidade de movimento conforme o ambiente em que se encontra, podendo este ser um veículo terrestre, aéreo ou aquático. Tais máquinas apresentam um certo nível de autonomia, ou seja, possuem capacidade de executar ações estipuladas sem intervenções humanas.

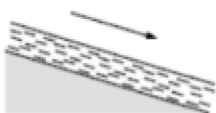




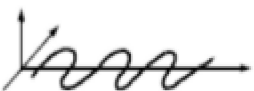



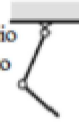


Uma das tarefas mais importantes de um sistema autônomo é a obtenção de informações do meio em que se encontra. O sensoriamento do robô se faz necessário para a

medição de grandezas presentes no ambiente, extraindo dados importantes de tais medidas (SIEGWART; NOURBAKHS; SCARAMUZZA, 2011). Além da utilização de sensores, algumas técnicas podem ser empregadas para se obter comportamentos mais complexos a serem executados pelo robô como aprendizado de máquinas, visando o autoajuste do robô durante a realização da tarefa e cooperação (OLIVI, 2019b).

Como o foco desta área é a locomoção dos robôs, a cinemática se apresenta como um tópico bastante importante, uma vez que visa estudar o comportamento de sistemas mecânicos, conforme descrito na Seção 2.1.1.

Existem diversos modos possíveis de se realizar a locomoção. Estes modelos são majoritariamente baseados em movimentações e comportamentos da própria natureza como deslizar, caminhar, galopar, correr e pular. Tais comportamentos são denominados bio-inspirados e exibidos na Figura 3.

Figura 3 - Mecanismos de locomoção utilizado em sistemas biológicos

Tipo de movimento	Resistência ao movimento	Cinemática básica do movimento
Fluxo por um canal 	Forças hidrodinâmicas	Turbilhão 
Rastejar 	Forças de fricção	Vibração longitudinal (efeito sanfona) 
Deslizar 	Forças de fricção	Vibração transversal 
Correr 	Perda de energia cinética	Movimento oscilatório de pêndulo composto 
Pular 	Perda de energia cinética	Movimento oscilatório de pêndulo composto 
Caminhar 	Forças gravitacionais	Rolamento de um polígono 

Fonte: Adaptado de *Autonomous Mobile Robots* (SIEGWART; NOURBAKHS; SCARAMUZZA, 2011)

Diferentemente dos movimentos presentes na natureza, as rodas são invenções humanas, mas representam meios de deslocamento bastante eficazes em terrenos planos e

compactos. Deste modo, são amplamente utilizadas em aplicações na robótica devido sua eficiência. Uma relação entre alguns tipos de rodas e seus respectivos DoFs são exibidos na Tabela 1.

2.2.1 Controlabilidade e Manobrabilidade

A capacidade de um robô móvel se movimentar pelo ambiente que se encontra é chamada de mobilidade cinética. Tal habilidade é proporcionada por suas rodas e capacidade de alterar sua direção, podendo esta ser limitada de acordo com sua construção física e escolha de rodas. Esta facilidade de executar movimentos ao longo do ambiente é denominada manobrabilidade (SIEGWART; NOURBAKHS; SCARAMUZZA, 2011).

A manobrabilidade (δ_M) é definida por meio do Grau de Mobilidade (δ_m) e Grau de Esterçamento (δ_s), nos fornecendo a equação:

$$\delta_M = \delta_m + \delta_s. \quad (2.2)$$

O grau de mobilidade se relaciona com a manipulação direta da movimentação de um robô, em outras palavras, os graus de liberdade que o robô manipula diretamente através das velocidades das rodas. Seu valor varia entre 0 e 3, de forma que, um robô com 0 grau de mobilidade é completamente restrito, o impossibilitando de se movimentar, e um robô com 3 graus de mobilidade consegue manipular diretamente todos os 3 possíveis graus de liberdade, o considerando um robô omnidirecional.

Grau de esterçamento, em contrapartida, são os graus de liberdade controlados indiretamente, em que se altera a configuração de direção do robô para movê-lo. Apesar de aumentar o grau de manobrabilidade do robô, o controle deste ângulo de esterçamento impõe uma restrição cinemática ao sistema.

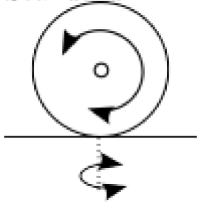
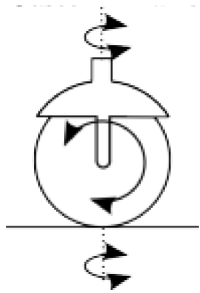

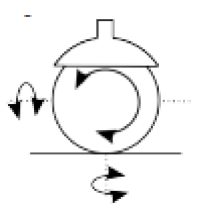
Controlabilidade é a facilidade de se gerar um sinal de controle que transfira o sistema de um estado inicial x_i a um estado final x_o . Em geral, existe uma correlação inversa entre a controlabilidade e manobrabilidade, de modo que, quanto maior a manobrabilidade de um robô móvel, maior é a dificuldade de se gerar um sinal de controle para executar a tarefa desejada (LYNCH; PARK, 2017).

2.2.2 Robôs Holonômicos

Os robôs holonômicos são aqueles que possuem capacidade de se movimentar para qualquer direção (x, y, θ) a qualquer momento e sem restrições. Este tipo de movimentação é bastante interessante devido sua grande capacidade de realizar manobras (SIEGWART; NOURBAKHS; SCARAMUZZA, 2011).

Existem diversas configurações para a construção de robôs holonômicos, dentre elas se encontra o robô omnidirecional com quatro rodas suetas, como o YouBot. Esta

Tabela 1 – Diferentes tipos de rodas e seus graus de liberdade

Tipo de Roda	Exemplo	Graus de Liberdade
Roda Padrão 	Roda dianteira de um carrinho de mão	Dois: <ul style="list-style-type: none"> • Rotação ao redor do eixo da roda • Rotação ao redor do seu ponto de contato com o chão
Roda Caster 	Cadeira de escritório	Três: <ul style="list-style-type: none"> • Rotação ao redor do eixo da roda • Rotação ao redor do seu ponto de contato com o chão • Rotação ao redor do eixo caster
Roda Sueca 	Roda padrão com rolos não acionados em torno de sua circunferência	Três: <ul style="list-style-type: none"> • Rotação ao redor do eixo da roda • Rotação ao redor do seu ponto de contato com o chão • Rotação ao redor do eixo dos rolos
Roda Esférica 	Rolamento esférico	Três: <ul style="list-style-type: none"> • Rotação em qualquer direção do plano do chão • Rotação ao redor do seu ponto de contato com o chão

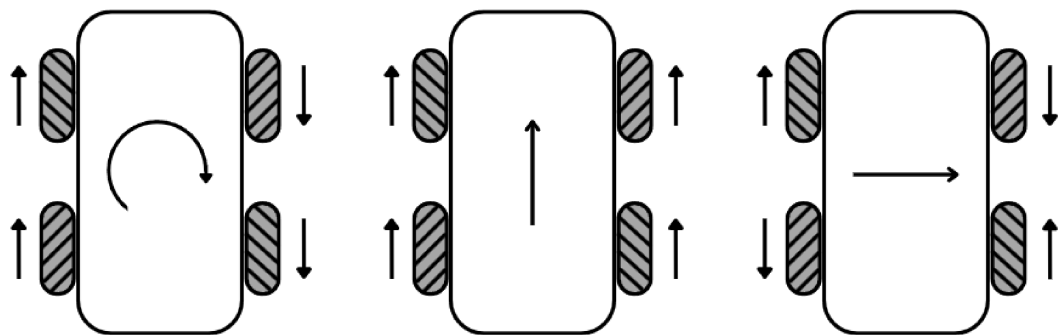
Fonte: Adaptado de *Autonomous Mobile Robots* (SIEGWART; NOURBAKHS; SCARAMUZZA, 2011)

configuração consiste em quatro rodas suecas de 45 graus conforme a Figura 4, cada uma controlada individualmente por um motor.

Com este tipo de robô é possível variar a direção de rotação e a velocidade relativa individualmente das quatro rodas, o robô pode ser movido ao longo de qualquer trajetória no plano, podendo girar simultaneamente em torno de seu eixo vertical enquanto se desloca (SIEGWART; NOURBAKHS; SCARAMUZZA, 2011).

Considerando uma velocidade de rotação igual para as quatro rodas do robô, a Figura 4a representa a configuração de rotação das rodas para que o robô gire em seu próprio eixo, a Figura 4b representa a configuração para que o robô se mova para frente e, por fim, a Figura 4c representa a configuração para que o robô se mova para a direita.

Figura 4 - Representação cinemática de um robô omnidirecional com quatro rodas suecas de 45 graus



(a) Movimentação de rotação em seu eixo. (b) Movimentação de para frente. (c) Movimentação para a direita.

Fonte: Elaborada pela autora (2024).

Apesar de possuir tamanha liberdade de movimento, conforme descrito na Seção 2.2.1, a dificuldade da controlabilidade de um robô é maior para valores de manobrabilidade maiores. Deste modo, controlar este tipo de robô por meio da velocidade de suas rodas se torna uma tarefa praticamente impossível, sendo necessário a utilização de outros sensores como GPS, acelerômetros, giroscópios, câmeras, entre outros.

2.3 DETECÇÃO DE OBJETOS

Detecção de objetos é uma técnica crucial na área de visão computacional, onde o objetivo é identificar e delimitar objetos em imagens ou vídeos. Para executar tal técnica é necessário o desenvolvimento e aprimoramento de algoritmos e modelos capazes de automaticamente reconhecer diferentes classes de objetos em um ambiente visual.

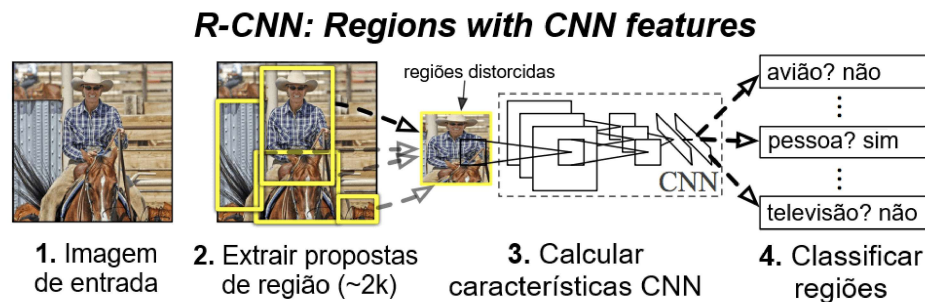
A detecção de objetos vai além da simples classificação, pois também fornece informações sobre a localização espacial dos objetos na imagem, geralmente expressa por

meio de caixas delimitadoras (ou do inglês, *bounding boxes*). Essas caixas indicam a posição aproximada dos objetos (BROWNLEE, 2021).

Geralmente utiliza-se redes neurais convolucionais (Ou CNN, do inglês *Convolutional Neural Networks*) e outras arquiteturas de *deep learning* para extrair características das imagens. Durante o treinamento, o modelo é exposto a conjuntos de dados anotados, nos quais as localizações e classes dos objetos estão previamente marcadas. Esta é uma metodologia que emprega uma técnica de janela deslizante em cada *pixel* de uma imagem para adquirir informações sobre a mesma (GALLAGHER, 2023).

A informação convolucional derivada deste processo é posteriormente submetida ao processamento por uma rede neural, de modo a ajustar seus parâmetros para aprender padrões que permitam a identificação eficiente de objetos em novas imagens não vistas durante o treinamento (GALLAGHER, 2023). Implementações notáveis de CNNs abrangem vários modelos, como R-CNN, *Mask R-CNN* e *Fast R-CNN*. A Figura 5 resume o funcionamento da RCNN.

Figura 5 - Resumo da arquitetura do modelo RCNN



Fonte: Adaptado de *Rich feature hierarchies for accurate object detection and semantic segmentation* (GIRSHICK et al., 2014).

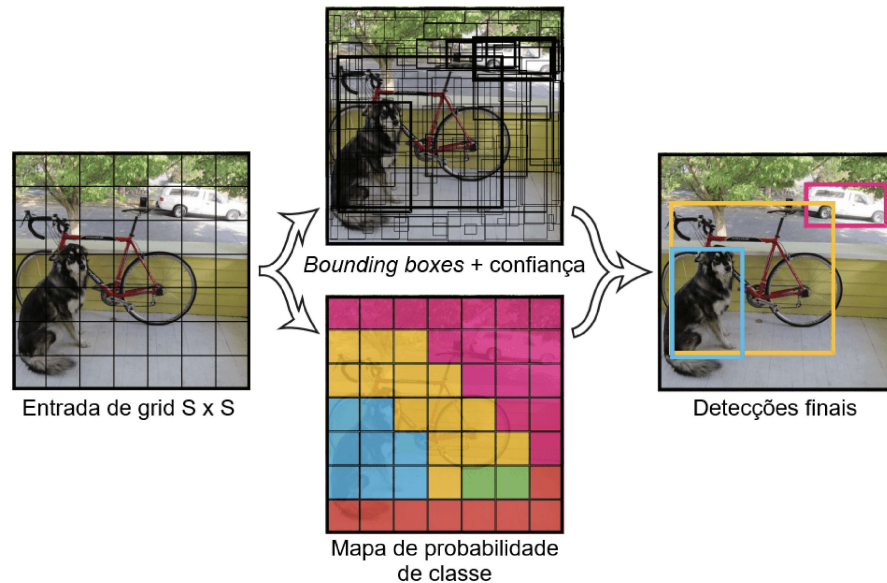
A detecção de objetos em imagens via R-CNN funciona em três etapas principais: primeiro, um método baseado em regiões, como o *Selective Search*, é usado para extrair possíveis objetos da imagem de entrada. Em seguida, uma CNN é usada para extrair as características dessas regiões selecionadas. Finalmente, cada região extraída é classificada para determinar a presença de um objeto.

Outro método de detecção de objetos é a família YOLO (sigla derivada do inglês *You Only Look Once*), desenvolvida inicialmente em (REDMON et al., 2016), sendo a "primeira metodologia de detecção de objetos a combinar o problema de desenhar caixas delimitadoras e identificar rótulos de classe em uma rede diferenciável ponta a ponta" (NELSON, 2021).

Considerando uma imagem dividida em uma grade $S \times S$, esta técnica utiliza uma única rede neural treinada visando prever, paralelamente, caixas delimitadoras (*bounding boxes*) e rótulos de classe diretamente para cada caixa da grade da imagem de entrada

(BROWNLEE, 2021). A partir dessas previsões, cada caixa é associada com as classes extraídas para obter o resultado da detecção de objetos. Estes métodos são mais rápidos que os procedimentos baseados em CNN por utilizarem uma única rede, mas perdem acurácia nesse processo, podendo apresentar dificuldades para detectar objetos pequenos. Um resumo de sua arquitetura pode ser observado na Figura 6.

Figura 6 - Resumo da arquitetura do modelo YOLO



Fonte: Adaptado de *You Only Look Once: Unified, Real-Time Object Detection* (REDMON et al., 2016).

Existem diversas versões da técnica presentes na literatura, entretanto, apenas as três primeiras: YOLO, YOLOv2, YOLOv3, foram desenvolvidas pelo seu criador Joseph Redmon. Após sua última criação, Redmon se afastou de pesquisas em visão computacional devido ao potencial de suas pesquisas integrarem "aplicações militares e as preocupações com a privacidade, que acabaram se tornando impossíveis de ignorar", segundo próprio autor.

Pesquisa e desenvolvimento utilizando técnicas YOLO tem sido uma área ativa devido sua ampla adoção pela comunidade científica, bem como novas implementações por muitos desenvolvedores e pesquisadores (BROWNLEE, 2021). A técnica YOLOv3, a última desenvolvida por Joseph Redmon, foi utilizada nesta dissertação e é melhor abordada na Seção 3.6.

A área de detecção de objetos está em constante desenvolvimento em busca de algoritmos inovadores, na avaliação de desempenho em diversos cenários e na contribuição para avanços tecnológicos que impactam uma ampla gama de aplicações práticas, buscando sempre melhorar a precisão e eficiência dos métodos de detecção de objetos. A exploração técnicas avançadas como detecção de objetos em tempo real, detecção de objetos em vídeos

e adaptação a diferentes condições de iluminação e escala são exemplos de temas bastante abordados em pesquisas recentes.

2.4 RASTREAMENTO OCULAR

O rastreamento ocular, do inglês *eye tracking*, é uma metodologia de pesquisa que envolve monitorar e registrar os movimentos dos olhos para coletar informações sobre atenção visual, foco e padrões de olhar de um indivíduo. A Reflexão da Córnea no Centro da Pupila, conhecida como PCCR em inglês (*pupil center corneal reflection*), é uma das mais empregadas atualmente devido à sua não intrusividade. (TOBII, 2024)

O método se baseia em Câmeras Infravermelhas (IR) para capturar os movimentos dos olhos com maior precisão. Neste método, uma luz infravermelha próxima do indivíduo é direcionada para o centro dos olhos (pupila), possibilitando que o reflexo da córnea (o brilho ou destaque na córnea) e o centro da pupila sejam rastreados simultaneamente. O rastreamento ocorre através do vetor que conecta o reflexo da córnea e o centro da pupila, representando a linha de visão (FARNSWORTH, 2019).

Este processo envolve a captura de imagens ou vídeos do olho usando um rastreador ocular e, em seguida, a identificação e o rastreamento do reflexo da córnea e do centro da pupila em cada quadro, conforme descrito anteriormente. Este método não é intrusivo e, portanto, é comumente usado em diversas aplicações, incluindo estudos de interação humano-computador, testes de usabilidade e certas pesquisas neurocientíficas.

Rastreadores oculares são normalmente classificados em dois tipos: rastreadores baseados em tela ou de mesa e os rastreadores usáveis. Os rastreadores baseados em tela realizam o *eye tracking* a distância e requerem uma tela para que o usuário interaja, são normalmente compostos por câmeras, IR e uma unidade de processamento com todos os algoritmos e técnicas utilizados para realizar o rastreamento (TOBII, 2024). Um exemplo de *eye tracking* baseado em tela é exibido na Figura 7.

Figura 7 - Exemplo de rastreador ocular baseado em tela



Fonte: *Tobii Pro X2 Eye Tracker Installation and Configuration Video* (TOBII, 2015).

Os rastreadores usáveis (*wearable*), por sua vez, são normalmente compostos por estruturas semelhante a óculos. Normalmente contém os mesmos componentes dos rastreadores baseados em tela (TOBII, 2024), mas como são vestíveis, garante maior liberdade de movimento para o usuário. Um exemplo de *eye tracking* vestível é exibido na Figura 8.

Figura 8 - Exemplo de rastreador ocular usável



Fonte: *Tobii Pro Glasses 3* (EYE, 2020).

Além dos dois rastreadores comerciais citados anteriormente, pesquisadores recentemente vêm desenvolvendo novas técnicas de *eye tracking* que não necessitam nenhum equipamento extra para a realização da técnica, apenas uma câmera comum, como a *webcam* de um computador. Apesar de mais baratas, tais técnicas normalmente não são

tão acuradas quanto as que utilizam equipamentos comercializados, mas ainda possuem grande potencial de desenvolvimento, como o trabalho de Fischer, Chang e Demiris (2018) que realiza o rastreamento ocular apenas com o uso de uma *webcam* comum.

O preço de um rastreador ocular pode variar bastante dependendo do modelo, marca, qualidade e das funcionalidades. É difícil estabelecer um preço médio para um equipamento de rastreamento ocular, com preços variando de pouco menos de R\$5.000 a mais de R\$350.000 (FARNSWORTH, 2022). Dispositivos mais simples, como aqueles projetados para uso em pesquisa acadêmica, podem ser mais acessíveis, enquanto sistemas mais avançados, destinados a aplicações industriais ou médicas, tendem a ser mais caros. Os rastreadores oculares baseados em tela normalmente possuem menor custo, enquanto e os óculos de rastreamento ocular geralmente estão no topo da faixa.

Além da grande variação no preço do equipamento, esses valores dificilmente são disponíveis facilmente ao público. A falta dessa transparência na precificação se deve a vários motivos, incluindo disponibilidade pública, preços elegíveis para descontos para estudantes e flutuações gerais de preços (FARNSWORTH, 2022). Um estudo elaborado por Rakhmatulin (2020) comparou diversos rastreadores mais baratos do período de 2010 a 2020. Apesar de não representar os preços atuais dos produtos e alguns destes estarem indisponíveis no mercado ou possuírem uma versão atualizada, é possível ter uma noção da precificação destes equipamentos sem a necessidade de consultar individualmente cada fornecedor.

No entanto, outras opções estão surgindo, especificamente o rastreamento ocular baseado em *webcam*. O rastreamento ocular típico normalmente possui um hardware específico que possui IR para realizar a tarefa. Em contrapartida, *webcams* não possuem IR, realizando o rastreamento empregando um processo dividido em quatro passos:

1. Detectar a localização do rosto;
2. Detectar a localização dos olhos no rosto;
3. Detectar a orientação dos olhos esquerdo e direito;
4. Mapear a orientação dos olhos no sistema de coordenadas da tela.

Esta técnica é mais econômica em comparação com os COTS, pois não há a necessidade de um hardware caro para o rastreamento ocular, funcionando com todas as *webcams* disponíveis no mercado. Também possuem grande potencial de escalabilidade, possibilitando realizar pesquisas online, independentemente de onde os participantes estejam no mundo, sem necessidade de um laboratório, levando a uma redução significativa no tempo de coleta de dados e permitindo que a análise subsequente comece mais rapidamente. Entretanto, possuem menor acurácia em comparação com os rastreadores baseados em IR,

sendo também mais sensíveis à mudança de luminosidade do ambiente e movimentação do usuário (JENSEN, 2022).

O rastreamento ocular é utilizado em vários campos, incluindo psicologia, neurociência, interação humano-computador, marketing e estudos de usabilidade. Em resumo, o rastreamento ocular é uma ferramenta de pesquisa poderosa e versátil que fornece percepções detalhadas sobre o comportamento visual e a cognição humana, contribuindo para uma compreensão mais profunda de como os indivíduos interagem com o mundo visual. Também representam um grande potencial para o desenvolvimento de novas estratégias de interação de pessoas com algum tipo de limitação física com o ambiente ao seu redor.

2.5 TRABALHOS RELACIONADOS

Pessoas com deficiência física frequentemente enfrentam desafios para se comunicar e controlar o meio ao seu redor, necessitando do desenvolvimento de soluções que os possibilitem interagir com o ambiente conforme suas capacidades e necessidades. Ao aproveitar o desenvolvimento de visão computacional e do rastreamento ocular, diversos pesquisadores desenvolveram possíveis soluções por meio de robôs assistivos que possam ser controlados com precisão e facilidade, mostrando novas possibilidades para usuários com mobilidade limitada. Este capítulo visa citar e comentar brevemente alguns destes trabalhos que se relacionam com a proposta desta dissertação, assim como seus pontos fortes e fracos que motivaram a confecção desta pesquisa.

Stiefelhagen et al. (2007) investigaram comandos de voz, gestos de apontar e orientação da cabeça e relataram resultados sobre a precisão de cada modalidade individualmente. Este trabalho, apesar de apresentar métodos de interação humano-máquina, possui foco maior no aspecto comunicativo deste contato do que em uma aplicação em si.

Palinko et al. (2016) compararam o olhar e o movimento da cabeça com base IHR para tarefas de construção de torre e relataram uma redução significativa no tempo de conclusão da tarefa e aumento na preferência subjetiva pelo sistema de rastreamento ocular em comparação com o sistema de rastreamento de cabeça, ambos os sistemas de rastreamento são abordados neste trabalho, mas com o acréscimo de comandos via movimentos verticais da cabeça e fechar os olhos.

Pasarica et al. (2016) apresentaram uma plataforma robótica controlada através do rastreamento ocular em que uma câmera foi montada em uma plataforma robótica para transmitir um vídeo ao vivo para a tela do usuário de modo a determinar a curva de aprendizado necessária para usar tal sistema e o comportamento do sistema no tempo de processamento e precisão de movimento do robô. As imagens para o rastreamento foram adquiridas de uma câmera infravermelha montada em uma armação de óculos ao invés de uma *webcam* comum, permitindo a movimentação do cursor na tela com maior precisão. O trabalho possui foco maior em avaliar a taxa de aprendizado do usuário e a precisão e

acurácia do rastreamento ocular desenvolvido.

Alsharif, Kuzmicheva e Graeser (2016) configuraram comandos através do movimento do olhar, olhos fechados e piscando para controlar os 7 graus de liberdade de um braço robótico e o desempenho do sistema foi avaliado com 10 participantes incluindo uma pessoa com deficiência motora para uma tarefa de rearranjo de blocos. Entretanto, a interface, por possuir diversos modos de operação, requisita inúmeros comandos para mudar de um modo para outro.

Sharma et al. (2020) desenvolveram uma interface gráfica controlada através do olhar para então controlar um braço robótico de modo realizar a tarefa de *pick and place*, avaliando o sistema através de 18 usuários (9 fisicamente aptos, 9 com limitações). Este trabalho não utiliza imagens de câmera para realizar o *eye-tracking*, mas um rastreador de prateleira, ou COTS do inglês *Commercial off-the-shelf*, do tipo montado em tela.

Scalera et al. (2021) apresentaram o controle de um robô industrial por meio de *eye tracking* para uso artístico, possibilitando novas formas de expressões artísticas tanto para pessoas com deficiências físicas quanto para pessoas não-deficientes. Realizaram testes experimentais utilizando um rastreador ocular comercial montado em tela para o *eye tracking*, possibilitando o controle de um braço robótico de 6 DoF para desenhar. Os autores concluíram que a qualidade dos resultados depende principalmente da calibração do rastreador ocular e da capacidade e habilidade do usuário em utilizar a aplicação desenvolvida, sem mover a cabeça e introduzir movimentos oculares involuntários durante o processo de pintura.

Sunny et al. (2021) visaram ajudar pessoas com deficiência a melhorar o controle de cadeiras de rodas e assim, realizar tarefas diárias. Para isso, desenvolveram uma GUI que comandaria um sistema robótico assistivo através de *eye tracking*. Este sistema é composto por uma cadeira de rodas motorizada, um braço robótico de 6 graus de liberdade e um rastreador ocular comercial da empresa Tobii. A interface gráfica desenvolvida foi testada com sujeitos saudáveis e inicialmente apresentou problemas devido ao grande número de botões e complexidade elevada, fazendo-se necessária uma simplificação desenvolvida pelos autores.

Abbasimoshaei, Winkel e Kern (2022) compararam o uso de duas linguagens de programação (MATLAB e Python) para a implementação do controle de um manipulador planar por meio de rastreamento ocular para auxiliar pessoas paralisadas. Ambas as rotinas utilizaram uma *webcam* para obter as imagens para processamento. Testes com um manipulador de três graus de liberdade foram realizados para comparar o desempenho de ambos os algoritmos para cinco posições distintas de olhar fixo: centro, cima, baixo, esquerda e direita. Ambos os algoritmos se mostraram promissores para uma implementação código aberto que auxilie pessoas paralisadas a controlar um braço robótico simples em duas dimensões.

Abbasimoshaei, Knudsen e Kern (2022) analisaram a possibilidade de controlar um braço robótico por meio de *eye tracking* com propósitos médicos, em particular, para auxiliar pessoas com algum tipo de paralisia física. Os autores propuseram, em adição ao *eye tracking*, um sistema de medição inercial para captar a movimentações de cabeça do usuário. Testes realizados com o sistema proposto mostraram resultados promissores para o controle de 2 DoF, entretanto, para 3 DoF esperam-se erros em seu controle por rastreamento ocular.

Xu et al. (2022) utilizaram uma combinação de sinais de EEG, visão computacional, detecção de olhar e orientação parcialmente autônoma para controlar um braço robótico, que demonstraram uma melhora drástica na precisão das tarefas e reduziram a carga cerebral causada por atividades mentais de longo prazo. O sistema foi constituído por um dispositivo de registro EEG, um rastreador ocular, um braço robótico DoF 5, uma câmera USB, Kinect e dois computadores, o que pode ser financeiramente custoso para alguns usuários.

Kong et al. (2022) desenvolveram um sistema que controle um braço robótico de 6 DoF utilizando *eye tracking*. Uma interface gráfica simples com oito botões foi criada para o usuário conseguir mover o braço e interagir com uma caixa no ambiente. Os resultados deste estudo, apesar de mostrarem que é possível controlar o manipulador utilizando a GUI por meio de *eye tracking*, o foco dos experimentos deste trabalho é avaliar o espaço de trabalho do robô e a acurácia do rastreamento ocular desenvolvido.

Ban et al. (2023) introduziram o TCES (*Two-Camera Eye-Tracking System*, um sistema de *eye tracking* utilizando uma *webcam* e um rastreador ocular comercial, monitorando os movimentos oculares por uma Rede Neural Convolutiva. Os autores introduziram também uma interface *all-in-one* que possibilita, em conjunto com o TCES, o controle de um braço robótico por meio de uma GUI. O controle do braço é realizado via comandos identificados pela movimentação ocular como: clique duplo (cima), parar (pisar), agarrar (esquerda) e soltar (direita). Este trabalho foca mais no desenvolvimento de um algoritmo para executar o rastreamento ocular e em seu potencial uso na robótica do que em uma aplicação em si.

Para melhor ilustrar os trabalhos relacionados e ressaltar suas diferenças e similaridades, as principais informações relacionadas com o desenvolvimento deste trabalho são apresentadas na Tabela 2.

Tabela 2 – Trabalhos Relacionados

Ano	Referência	Título	Técnicas	COTS	Aplicação
2007	(STIEFELHAGEN et al., 2007)	Enabling Multimodal Human-Robot Interaction for the Karlsruhe Humanoid Robot	Voz, gestos e Câmera	Não	Desenvolvimento de tecnologias <i>eyetracking</i> para interação humano-robô
2016	(PALINKO et al., 2016)	Robot Reading Human Gaze: Why Eye Tracking is Better than Head Tracking for Human-Robot Collaboration	Câmera	Não	Comparação de rastreamento ocular e de cabeça em interação humano-robô
2016	(PASARICA et al., 2016)	Remote Control of an Autonomous Robotic Platform Based on Eye Tracking	Óculos Comercial	Sim	Implementação de um sistema de controle de uma plataforma robótica por <i>eyetracking</i> para pacientes com deficiência neuromotora
2016	(ALSHARIF; KUZMICHEVA; GRAESER, 2016)	Gaze gesture-based human robot interface	Óculos Comercial	Sim	Desenvolvimento de interface humano-robô controlada por <i>eyetracking</i> para controlar um braço robótico de 7 DoF
2020	(SHARMA et al., 2020)	Eye Gaze Controlled Robotic Arm for Persons with Severe Speech and Motor Impairment	Rastreador Ocular Comercial	Sim	Desenvolvimento de interface controlada pelo olhar para usuários com graves deficiências motoras e de fala para manipular um braço robótico
2021	(SCALERA et al., 2021)	Human-Robot Interaction through Eye Tracking for Artistic Drawing	Rastreador Ocular Comercial	Sim	Desenvolvimento de arquitetura para controlar um manipulador através de uma interface de <i>eyetracking</i> com fins artístico
2021	(SUNNY et al., 2021)	Eye-gaze control of a wheelchair mounted 6 DOF assistive robot for activities of daily living	Rastreador Ocular Comercial	Sim	Projeto um sistema de controle de robô através de <i>eyetracking</i> para auxiliar indivíduos com deficiência
2022	(ABBASIMOSHAEI; WINKEL; KERN, 2022)	Design, simulation and evaluation of two coding programming languages for an eye-tracking controlling system for a three degrees of freedom robot useful for paralyzed people	Câmera	Não	Uso de <i>eyetracking</i> para controlar um dispositivo robótico de 3 DoF projetado de forma a movimentar o braço de pessoas paralisadas
2022	(ABBASIMOSHAEI; KNUDSEN; KERN, 2022)	Application of Eye-Tracking for a 3-DOF Robot Assisted Medical System	Câmera	Não	Deteção de pontos de olhar em superfícies para determinar o interesse dos indivíduos de modo a controlar robôs por <i>eyetracking</i> em um contexto médico
2022	(XU et al., 2022)	Continuous Hybrid BCI Control for Robotic Arm Using Noninvasive Electroencephalogram, Computer Vision, and Eye Tracking	EEG, Kinect, Câmera	Sim	desenvolvimento de uma interface cérebro-computador não invasivo baseado em EEG para um sistema de controle de braço robótico de 5 DoF que permita aos usuários completar tarefas utilizando um braço robótico
2022	(KONG et al., 2022)	Eye-tracking-based robotic arm control system	Rastreador Ocular Comercial	Sim	desenvolvimento de um sistema de controle de braço robótico não invasivo baseado em rastreamento ocular
2023	(BAN et al., 2023)	Persistent Human-Machine Interfaces for Robotic Arm Control Via Gaze and Eye Direction Tracking	Câmera e Rastreador Ocular Comercial	Sim	Apresentação de um sistema de rastreamento ocular de duas câmeras e um método de classificação de dados para interfaces homem-máquina

Fonte: Elaborado pela autora (2024).

Um problema identificado em trabalhos envolvendo robótica assistiva para auxiliar pessoas com deficiências motoras é a complexidade da interface desenvolvida, notado em Alsharif, Kuzmicheva e Graeser (2016), pois uma vez que o usuário utiliza apenas o movimento ocular para controlá-la, a confecção da aplicação se torna bastante limitada, sendo necessária criatividade para contornar tal adversidade. A presença de métodos que necessitam outros tipos de equipamentos específicos, como o uso de eletroencefalograma em Xu et al. (2022), apesar de oferecem bons resultados, podem ser um impeditivo para algumas pessoas. Entretanto, a principal barreira do desenvolvimento da área é o custo de equipamentos, e até mesmo softwares, para realizar a tarefa de *eye tracking*.

Uma grande parcela das pesquisas utilizam tais métodos, como visto sua presença na maioria dos trabalhos citados anteriormente, mas essa abordagem pode ser bastante custosa para o usuário.

O alto custo de implementação pode inviabilizar o acesso a essas técnicas para uma grande parcela da população, que não possui poder financeiro para adquirir os equipamentos específicos necessários. Desta forma, propõe-se uma implementação de uma interface que permita o usuário controlar um braço robótico para realizar uma ação de *pick and place* que seja simples de se utilizar e de menor custo possível. Em virtude do elevado gasto necessário em computadores que possibilitem o alto poder computacional, imprescindível para processamentos de imagem, o baixo custo se refere aos equipamentos necessários para a realização das técnicas de *eye tracking* e rastreamento de posição de cabeça, utilizados para comandar o aplicativo proposto nesta dissertação.

3 INTEGRAÇÃO DE FERRAMENTAS

Ao combinar diferentes tipos de tecnologias, desenvolveu-se uma plataforma robótica com um sistema de controle de rastreamento ocular para oferecer às pessoas com deficiências neuromotoras a possibilidade de controlar um braço robótico, possibilitando-as pegar um determinado item e movê-lo para outra posição, uma interação conhecida como *pick-and-place*.

Para isso, foi desenvolvida uma interface gráfica que, utilizando informações fornecidas por um sistema de detecção de objetos em tempo real, mostra uma tela contendo os itens que o usuário pode pegar. Este, por sua vez, utiliza o controle de rastreamento ocular ou de cabeça para selecionar o objeto desejado, fazendo com que o programa envie a posição deste item para o robô pegá-lo e trazê-lo para próximo do usuário.

Como o trabalho envolve a manipulação de um braço robótico em conjunto com aplicações de visão computacional como *eye tracking* e identificação de objetos, o uso de uma ferramenta que facilite a integração de tais recursos se mostra bastante interessante. O ROS é uma ferramenta bastante útil nessa situação, uma vez que possui pacotes prontos para realizar essas atividades cruciais para a execução do aplicativo proposto.

Visando a construção de um ambiente de simulação, o simulador Gazebo foi escolhido devido à presença de variáveis físicas no ambiente, como atrito e inércia, de modo que, para controlar os robôs YouBot e Panda na simulação, utilizou-se a biblioteca *MoveIt*. A detecção dos objetos em cena foi realizada através do pacote YOLO, já a detecção do *eye tracking* se deu através do pacote RT-GENE.

3.1 ROS

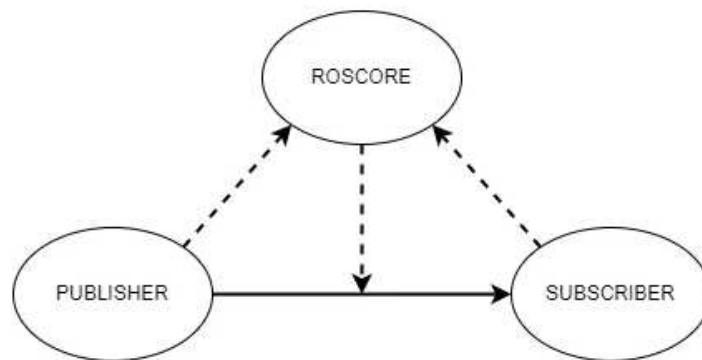
ROS, uma abreviação de *Robot Operating System* (Sistema Operacional de Robôs), é um conjunto de bibliotecas de software e ferramentas utilizadas majoritariamente para a construção e programação de robôs. Apesar de seu nome, ROS não é de fato um sistema operacional, mas sim uma SDK (*software development kit*) que providencia as ferramentas necessárias para criar aplicações robóticas (ROBOTICS, 2013–2016). É desenvolvido em código aberto e amplamente utilizado por pesquisadores e desenvolvedores na área de robótica. O *framework* fornece uma plataforma flexível e modular para projetar, desenvolver e controlar sistemas robóticos de maneira eficaz.

O ROS opera com base na concepção de nós, que são essencialmente componentes executáveis contidos nos pacotes ROS. Cada nó pode ser um sensor, um controlador, um algoritmo de processamento, etc. Os nós podem ser executados em diferentes máquinas além da possibilidade de se comunicar através do ROS *Master*.

O nó *master* do ROS, conhecido como ROS *Master* ou ROScore, constitui-se

de uma coleção de nós e programas que atuam como pré-requisitos essenciais para o funcionamento pleno do ROS (ROBOTICS, 2013–2016), sendo sua presença contínua de operação um requisito indispensável para garantir o correto funcionamento do sistema. O ROScore desempenha o papel crucial de estabelecer e facilitar a troca de informações entre os nós, possibilitando a transmissão de mensagens entre eles (TERAOKA, 2022) (BACK-END, 2022).

Figura 9 - Funcionamento do ROScore para comunicação em tópicos



Fonte: Blog Eletrogate: Introdução ao Robot Operating System (ROS) (TERAOKA, 2022)

Os tópicos são a forma básica de comunicação dos nós. Podem atuar como publicadores (*publishers*) ou inscriteiros (*subscribers*) de modo que, a cada inicialização de um nó, é estabelecida uma conexão com o ROScore para registrar detalhes dos fluxos de mensagens que publica e dos fluxos aos quais deseja se inscrever, ilustrado na Figura 9. Os publicadores têm a responsabilidade de divulgar mensagens, enquanto os inscriteiros podem acessá-las, conforme a Figura 10. Importante notar que é possível a criação de um tópico com um único publicador, enquanto inúmeros inscriteiros a este podem coexistir, reforçando a ideia de uma relação flexível e escalável entre eles (TERAOKA, 2022).

Além das mensagens em tópicos, ROS também oferece serviços, os quais são solicitações de um nó para outro nó executar uma ação específica e fornecer uma resposta. Estes serviços permitem criar uma comunicação cliente/servidor síncrona simples entre nós, conforme esquematizado na Figura 11, muito útil para alterar uma configuração de um robô, ou solicitar uma ação específica como ativar o modo *freedrive* (modo que possibilita a movimentação manual do manipulador), solicitar dados específicos, etc (BACK-END, 2022).

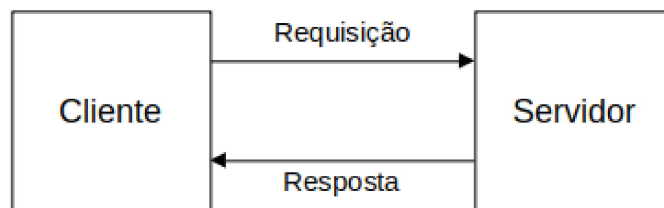
No geral, o ROS é uma plataforma poderosa que ajuda pesquisadores a concentrarem-se na pesquisa em robótica em vez de gastar tempo no desenvolvimento de infraestrutura básica. Esta ferramenta é amplamente usado em projetos de pesquisa, desde robótica móvel até robótica industrial, e tem uma comunidade ativa que contribui com pacotes e soluções para diversas aplicações.

Figura 10 - Esquema de comunicação publisher/subscriber



Fonte: Blog Eletrogate: Introdução ao Robot Operating System (ROS) (TERAOKA, 2022)

Figura 11 - Esquema de comunicação cliente/servidor



Fonte: Elaborado pela autora (2023).

3.2 GAZEBO

O Gazebo é um simulador de código aberto amplamente utilizado na pesquisa e desenvolvimento de robótica. Tal software fornece um ambiente virtual onde é possível modelar e simular sistemas robóticos em 3D. O simulador permite criar modelos precisos de robôs, sensores, atuadores e ambientes, reproduzindo fielmente as interações e comportamentos reais.

O Gazebo é particularmente valioso para pesquisadores, uma vez que oferece várias funcionalidades poderosas. Ele permite criar ambientes personalizados com detalhes como terrenos, edifícios e objetos. Além disso, é possível definir físicas realistas, como dinâmica de corpos rígidos, efeitos de fricção e gravidade, para simular o movimento e as interações entre os componentes do robô (ROBOTICS, 2022).

Uma característica notável do Gazebo é sua capacidade de simular sensores, como câmeras, sensores de proximidade e giroscópios, permitindo que os pesquisadores testem algoritmos de percepção e controle em um ambiente seguro e controlado. Isso é

particularmente útil para aprimorar algoritmos antes de aplicá-los a robôs reais.

Além disso, o Gazebo oferece suporte para diferentes bibliotecas de controle e linguagens de programação, permitindo que os pesquisadores implementem e testem seus algoritmos de controle diretamente no simulador.

Como resultado, o Gazebo é uma ferramenta valiosa para acelerar o ciclo de desenvolvimento de robôs e sistemas autônomos. Esta ferramenta permite que os pesquisadores testem e ajustem algoritmos, validem estratégias de controle e avaliem o desempenho de maneira segura e econômica. Sua flexibilidade e recursos avançados o tornam uma escolha popular entre a comunidade de pesquisa em robótica, contribuindo para avanços significativos nesse campo.

3.3 MOVEIT

O *MoveIt* é uma biblioteca e conjunto de ferramentas de planejamento de movimento para robôs e manipuladores robóticos que operam sob o sistema operacional ROS. Essa ferramenta oferece uma ampla gama de recursos para o planejamento e execução de movimentos de robôs e manipuladores robóticos, permitindo que os desenvolvedores criem trajetórias precisas e seguras para os robôs executarem suas tarefas.

A biblioteca simplifica a tarefa de gerar trajetórias complexas, considerando as restrições do ambiente, cinemática do robô e possíveis obstáculos, garantindo assim um movimento suave e eficiente (COLEMAN et al., 2014). Além disso, o *MoveIt* também permite a simulação e visualização desses movimentos, facilitando a depuração e otimização do comportamento do robô. Com sua integração ao ROS, o *MoveIt* se tornou uma ferramenta fundamental para o desenvolvimento de aplicações robóticas avançadas e interativas (ROBOTICS, 2013).

3.4 YOUBOT

A parte móvel do YouBot é um robô holonômico (Figura 12), e conforme descrito na Seção 2.2.2, este tipo de robô possui grau de liberdade máxima (3 para ambientes bidimensionais) e pode alcançar qualquer coordenada no espaço cartesiano em qualquer orientação em z . Este robô possui velocidade linear tanto no eixo- x quanto em y , além da velocidade angular em z , respeitando a regra de Fleming.

Apesar da cinemática inversa sozinha parecer ser eficaz para o deslocamento do robô, em determinados percursos o caminho percorrido passa a divergir significativamente do destino pré-determinado. Contornar este problema normalmente é possível por meio do uso de controladores, sendo o Proporcional-Integral-Derivativo (PID) um dos mais utilizados tanto na academia quanto na indústria.

Figura 12 - Robô Youbot da empresa KUKA, já descontinuado



Fonte: KUKA youBot: Research & Application Development in Mobile Robotics (GMBH, 2010)

Como os robôs holonômicos possuem complexidade muito superior, por exemplo, a robôs diferenciais, o controle de posição através das velocidades de suas rodas é uma tarefa árdua. As rodas suecas utilizadas em robôs omnidirecionais funcionam com base no deslizamento dos pequenos rolos externos, logo, a utilização de *encoders* se apresenta ineficaz (ROBOTS, 2014).

Um possível método para o controle de posicionamento de um robô como o YouBot se dá através de sensores que detectam seus arredores para rastrear seu movimento, como giroscópios, câmeras ou GPS. Como as velocidades linear e angular de um robô omnidirecional são independentes entre si, é necessário fazer dois controladores independentes para cada uma delas.

Apesar de estar disponível o robô YouBot, Figura 12, no GRIN (Grupo de Robótica Inteligente da UFJF), optou-se por utilizar o robô manipulador Panda no simulador Gazebo. O robô Panda aparentou um comportamento mais estável que o YouBot. Em trabalhos futuros com simulações em ambientes reais, pretende-se utilizar o YouBot.

3.5 FRANKA PANDA

O Panda, exibido na Figura 13 é um robô colaborativo com 7 graus de liberdade desenvolvido pela empresa alemã FRANKA EMIKA. A versão de pesquisa deste robô

permite controle direto e a possibilidade de programá-lo e conectá-lo com sensores externos através de pacotes e bibliotecas para C++, ROS e *MoveIt*.

Figura 13 - Robô Panda produzido pela empresa Franka Emika



Fonte: Mybotshop: Introdução ao Robot Operating System (ROS) (TERAOKA, 2022)

Neste trabalho utilizou-se o manipulador Panda por possuir 7 DoF, e como mencionado anteriormente, um manipulador necessita de, pelo menos, 6 DoF para chegar em qualquer ponto de sua área de trabalho com qualquer orientação. Outra motivação de sua escolha é a vasta contribuição e suporte da comunidade do ROS, uma vez que esse robô é utilizado em seus tutoriais do *MoveIt*.

3.6 DARKNET YOLO

Para tornar os objetos visíveis para o usuário na interface, é necessário que os itens sejam detectados nas imagens capturadas pela câmera. Essa etapa foi realizada utilizando uma rede neural pré-treinada: a *Darknet YOLOv3* disponível no site do desenvolvedor em (REDMON, 2021).

YOLO (*You Only Look Once*) é um algoritmo de detecção de objetos que se destaca por sua capacidade de detectar múltiplos objetos em uma imagem em uma única passagem, em vez de exigir várias passagens como outros métodos. O YOLOv3 é uma versão do YOLO que utiliza uma arquitetura mais profunda e complexa para melhorar ainda mais a precisão da detecção. Esse incorpora várias estratégias para melhorar o treinamento

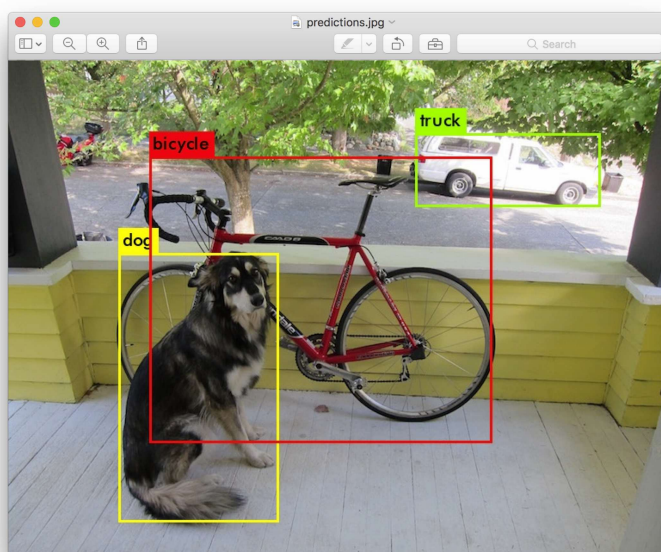
e o desempenho, englobando previsões em várias escalas, um classificador de *backbone* aprimorado e otimizações adicionais (REDMON, 2021).

Darknet-53 é uma rede neural convolucional com 53 camadas que atua como base da detecção de objetos no YOLOv3 (REDMON; FARHADI, 2018). O termo "Darknet" se refere à estrutura de rede neural usada para treinar e executar o YOLOv3. É chamada de darknet porque foi desenvolvida como um *framework* de aprendizado profundo de código aberto sem muita publicidade ou atenção em comparação com outros projetos populares.

A estrutura de rede neural *Darknet* é uma arquitetura de *deep learning* que foi projetada para ser eficiente e rápida, permitindo o treinamento de modelos de detecção de objetos em grande escala (REDMON, 2013–2016). O YOLOv3 Darknet combina essencialmente o algoritmo YOLOv3 com a estrutura de rede neural darknet.

O funcionamento da técnica ocorre dividindo uma imagem em uma grade e, para cada célula da grade, prevê caixas delimitadoras (*bounding boxes*) que envolvem objetos detectados. Cada caixa delimitadora é associada a uma classe de objeto específica (como carro, pessoa, bicicleta, etc.) e probabilidades para cada região. O modelo também gera pontuações de confiança para cada classe de objeto detectado em cada célula da grade. Considera-se como detecção de uma classe apenas as regiões da imagem com alta pontuação. Um exemplo de detecção fornecido pelo próprio autor pode ser visto nas Figuras 14 e 15, contendo as caixas delimitadoras da detecção e os valores de confiança para cada uma das bounding boxes, respectivamente.

Figura 14 - Imagem de referência da detecção do YOLO contendo a identificação de 3 itens: um cachorro, uma bicicleta e um caminhão (truck)



Fonte: YOLO: Real-Time Object Detection (REDMON, 2021)

Figura 15 - Saída da detecção realizada na imagem 14 contendo os valores de confiança para cada item identificado

```

layer      filters  size      input          output
  0 conv     32  3 x 3 / 1  416 x 416 x 3  ->  416 x 416 x 32  0.299 BFLOPs
  1 conv     64  3 x 3 / 2  416 x 416 x 32  ->  208 x 208 x 64  1.595 BFLOPs
-----
 105 conv    255  1 x 1 / 1   52 x 52 x 256  ->  52 x 52 x 255  0.353 BFLOPs
 106 detection
truth_thresh: Using default '1.000000'
Loading weights from yolov3.weights...Done!
data/dog.jpg: Predicted in 0.029329 seconds.
dog: 99%
truck: 93%
bicycle: 99%

```

Fonte: YOLO: Real-Time Object Detection (REDMON, 2021)

Além dos modelos pré-treinados para a utilização do algoritmo, também é possível treinar os próprios modelos. Para treinar o YOLOv3 Darknet, são necessárias imagens rotuladas com caixas delimitadoras e as classes dos objetos presentes. O algoritmo é então treinado para ajustar os pesos da rede neural para minimizar a diferença entre as caixas delimitadoras previstas e as caixas reais nos dados de treinamento.

Em resumo, o YOLOv3 Darknet é uma versão avançada do algoritmo YOLO de detecção de objetos, que utiliza a estrutura de rede neural *darknet* para treinamento e inferência. É conhecido por sua eficiência e rapidez em detectar múltiplos objetos, tornando-o uma ferramenta poderosa para aplicações de visão computacional e reconhecimento de objetos em tempo real, apesar de menor acurácia em comparação com métodos como o R-CNN e apresentar dificuldades em detectar objetos pequenos no ambiente.

3.7 RT-GENE E RT-BENE

Rastreamento ocular, do inglês *eye tracking*, é um método para estudar a atenção visual do usuário. Com ele, é possível investigar a direção e a duração dos movimentos oculares, fornecendo informações valiosas sobre a atenção visual, o processamento de informações visuais e as estratégias de busca visual empregadas pelo participante.

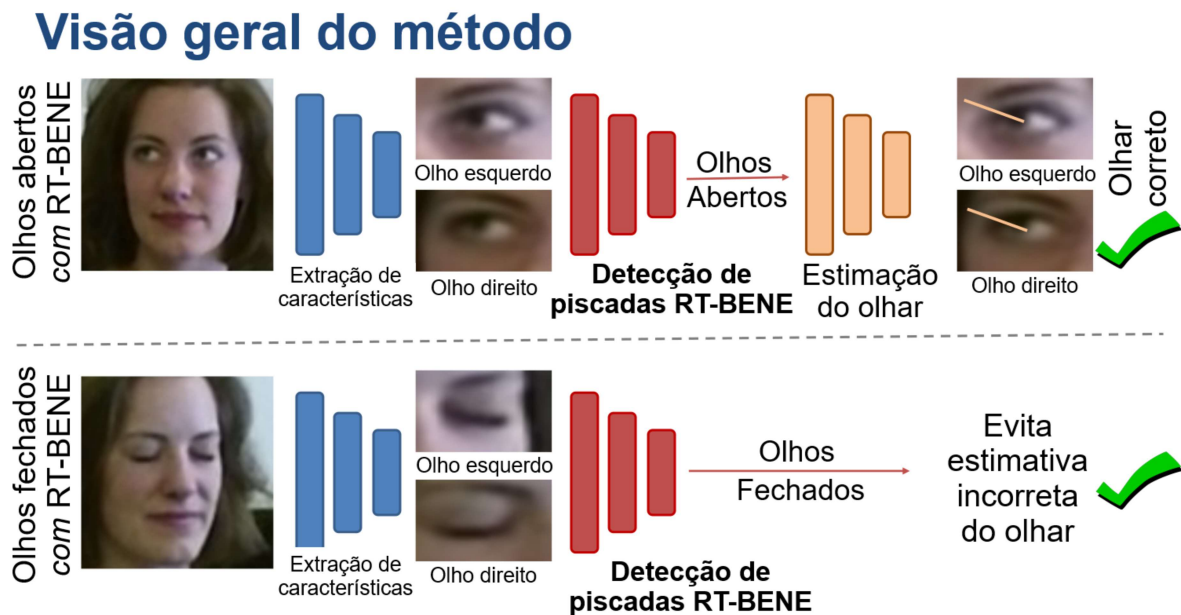
O rastreamento ocular pode ser realizado por diversos dispositivos, como câmeras de infravermelho ou acessórios usáveis em formato semelhante a um óculos. Esses dispositivos capturam os movimentos dos olhos em tempo real, permitindo a criação de trajetórias dos movimentos oculares.

É possível realizar diversas pesquisas com rastreamento ocular, podendo incluir o uso de estímulos visuais específicos, como imagens, vídeos ou interfaces de usuário, para avaliar as respostas visuais dos participantes, o que permite analisar como a atenção é

direcionada dentro desses estímulos e como os padrões de movimento ocular podem variar em diferentes contextos, ou comandar uma determinada interface homem máquina.

Pacotes do ROS foram utilizados neste trabalho para estabelecer a posição do olhar ou cabeça (RT-GENE) e identificar o piscar dos olhos (RT-BENE). Uma visão geral do funcionamento do pacote é mostrado na Figura 16.

Figura 16 - Visão geral do funcionamento do RT-GENE



Fonte: Adaptado de *RT-GENE & RT-BENE: Real-Time Eye Gaze and Blink Estimation in Natural Environments* (Git repository) (FISCHER, 2021).

A detecção do olhar começa pela utilização do RT-BENE. Este se inicia detectando a cabeça do indivíduo em cena e extraíndo, separadamente, os olhos esquerdo e direito para determinar se os estão abertos ou fechados. Caso os olhos estejam abertos, o RT-GENE estima a direção do olhar, caso estejam fechados, considera-se como uma piscada do usuário.

3.7.1 RT-GENE

O RT-GENE combina um sistema de captura de movimento para detecção de pose de cabeça, com um vetor de olhar obtido de forma automática sob condições de visualização livre (ou seja, sem especificar um alvo explícito do olhar), que permite o registro rápido do conjunto de dados (FISCHER; CHANG; DEMIRIS, 2018).

A estimativa do olhar no RT-GENE é realizada usando múltiplas redes. Para treinar estas redes, criaram um *dataset* RT-GENE que consiste em duas partes: uma em que os óculos de rastreamento ocular são utilizados, logo é possível identificar o posicionamento

da cabeça e o rastreo ocular, e outra sem tais óculos. Essa parte do *dataset* sem os óculos foi utilizada para treinar uma rede adversária generativa que seria posteriormente implementada para remover os óculos das imagens com o acessório, ilustrada na Figura 17.

Figura 17 - Remoção dos óculos pelo RT-GENE *Inpainting*



Fonte: RT-GENE & RT-BENE: Real-Time Eye Gaze and Blink Estimation in Natural Environments (Git repository) (FISCHER, 2021).

Os autores usaram Redes Convolucionais Cascadeadas de Múltiplas Tarefas (MTCNN, do inglês *Multi-Task Cascaded Convolutional Networks*) para detectar rostos, bem como pontos de referência dos ângulos dos olhos, nariz e boca. Em seguida, o recorte da face resultante é girado e dimensionado para minimizar a distância entre os pontos de referência alinhados e a posição predefinida dos pontos médios da face, obtendo assim uma imagem da face normalizada através do algoritmo *accelerated iterative closest point algorithm* proposto por Besl e McKay (1992) (FISCHER; CHANG; DEMIRIS, 2018).

Em seguida, os recortes dos olhos são extraídos da imagem facial normalizada como um retângulo de tamanho fixo centrado nos pontos de referência dos olhos. Por fim, a orientação da cabeça do sujeito é obtida aplicando o método avançado apresentado por Patacchiola e Cangelosi (2017) (FISCHER; CHANG; DEMIRIS, 2018).

3.7.2 RT-BENE

Levando em conta que os *datasets* necessários para realizar a detecção do olhar e de piscadas possui uma certa semelhança, os autores utilizaram um *framework* conjunto para a estimativa de piscadas e olhar proposto inicialmente para o RT-GENE (CORTACERO; FISCHER; DEMIRIS, 2019).

Para determinar se o usuário está piscando ou não, os autores manualmente selecionaram as imagens listadas como "sem óculos" do *dataset* RT-GENE mencionado anteriormente. O subconjunto de dados escolhido foi dividido em três categorias, conforme a imagem 18: 1) olhos abertos, 2) olhos fechados e 3) incertos. Os olhos abertos foram classificados como imagens em que pelo menos uma parte da esclera (parte branca do olho) ou da pupila é visível. Olhos fechados é o estado em que as pálpebras ficam

completamente fechadas. A categoria incerta é usada quando uma imagem não pode ser atribuída inequivocamente a uma das outras categorias, por exemplo, quando a posição da cabeça é bastante ruim.

Figura 18 - Subconjuntos do dataset para o RT-BENE



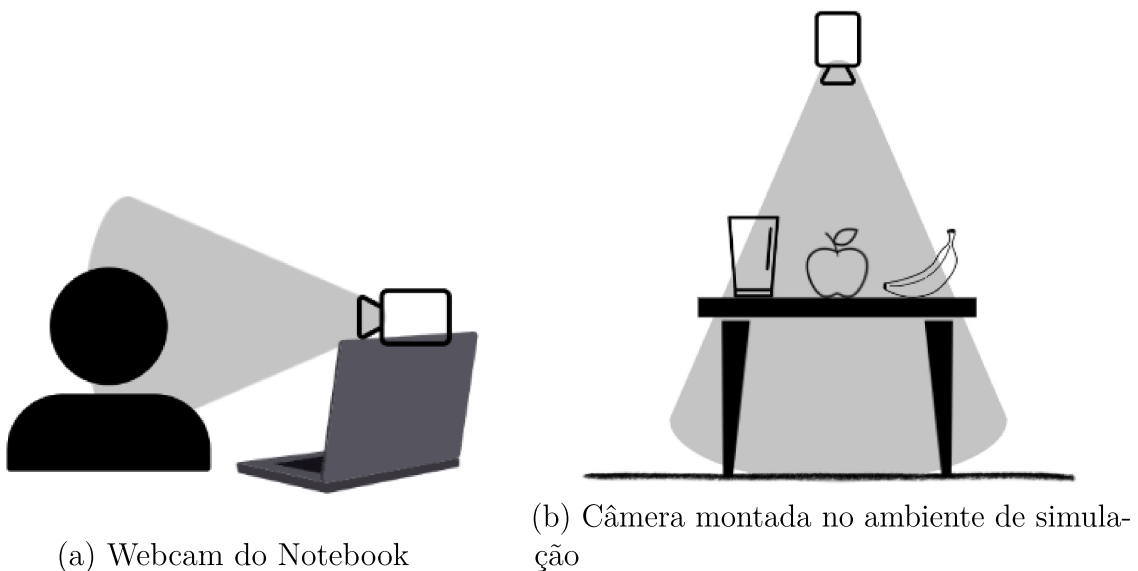
Fonte: RT-GENE & RT-BENE: Real-Time Eye Gaze and Blink Estimation in Natural Environments (Git repository) (FISCHER, 2021).

Essas imagens são usadas para treinar as redes neurais convolucionais. Os autores forneceram um conjunto de CNNs de referência para estimativa de piscada. Imagens do olho esquerdo e do olho direito são fornecidas como entrada.

4 METODOLOGIA

Este trabalho tem como objetivo permitir o comando de um manipulador robótico por meio de duas câmeras (uma webcam e outra câmera montada no ambiente) para auxiliar pessoas com limitações físicas. A webcam tem o propósito de captar as imagens para obter a direção do olhar do usuário para que este controle o aplicativo. Considerando que as simulações e testes foram realizados apenas de forma virtual, neste trabalho utilizou-se uma câmera montada no ambiente do Gazebo, sendo posicionada na vertical e apontada para baixo, onde se encontra uma mesa com três objetos distintos dispostos, uma vez que seu intuito é a identificação dos objetos em cena. Um esquema da montagem das câmeras é ilustrado pela Figura 19, onde a Figura 19a representa a webcam e a Figura 19b a câmera montada no ambiente simulado.

Figura 19 - Esquema da montagem das duas câmeras utilizadas no trabalho



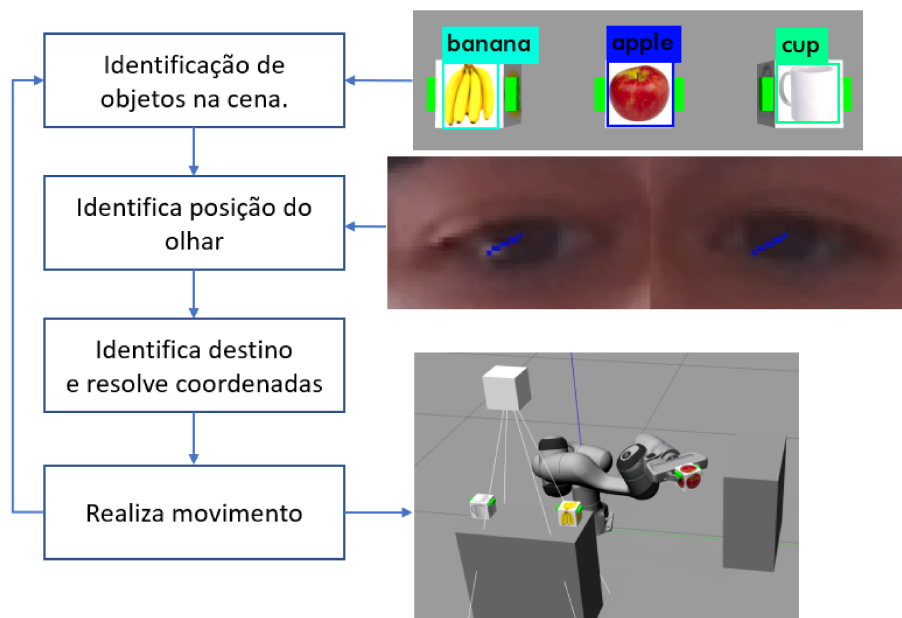
Fonte: Elaborado pela autora (2024).

O método proposto baseia-se em quatro processos principais, conforme resumido na Figura 20. O primeiro processo consiste na identificação dos objetos presentes na cena utilizando a rede YOLO (REDMON, 2021) com a câmera montada no ambiente, como descrito na Seção 4.1. Esse procedimento desenrolou-se em um ambiente simulado utilizando o Gazebo, um simulador de robótica 3D de código aberto bastante empregado com o ROS. Os objetos identificados ficam expostos em uma primeira mesa posicionada na frente do manipulador, ao lado deste, há uma segunda bancada vazia para o robô depositar o item. A câmera montada no ambiente foi posicionada com sua lente para baixo, acima da primeira mesa, na frente do robô. A identificação do YOLO pode ser vista na primeira

imagem da Figura 20, e o ambiente de simulação é mostrado na última imagem da mesma figura.

O segundo processo envolve a identificação da direção do olhar do usuário usando a biblioteca RT-GENE para selecionar um dos itens possivelmente identificados por meio de uma webcam, apresentado na Seção 4.2 (FISCHER; CHANG; DEMIRIS, 2018). O terceiro processo trata da extração das informações fornecidas pelo YOLO para obter a posição dos objetos na cena, conforme descrito na Seção 4.3. O quarto processo é o controle do manipulador visando realizar as ações de *pick and place*, explicado na Seção 4.6. O quarto e último processo transcorreu no mesmo ambiente simulado no Gazebo, já descrito anteriormente para o primeiro processo.

Figura 20 - Fluxo simplificado da metodologia



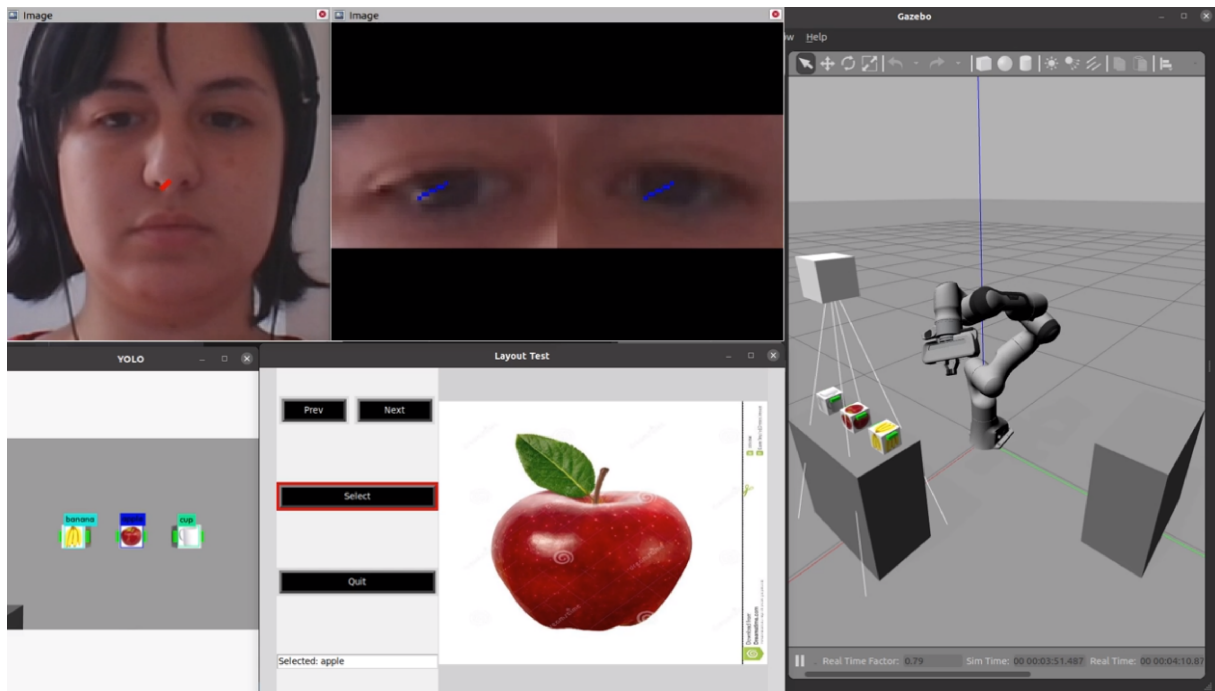
Fonte: Elaborado pela autora (2023).

Para implementar os algoritmos neste trabalho utilizou-se um computador com processador 11th Gen Intel® Core™ i7-11800H @ 2.30GHz × 16, 8 GB de RAM, GPU Nvidia Geforce RTX 3070, sistema operacional Ubuntu 20.04 com Noetic ROS e os pacotes RT-GENE, YOLO, Tkinter, *MoveIt*, Gazebo e franka_ros. Cada uma das ferramentas principais utilizadas será detalhada nas seções subsequentes.

O código do projeto é aberto e está disponibilizado em Git em Ferraz (2023) e diversos vídeos mostrando etapas da execução do trabalho, assim como a execução da aplicação completa estão disponíveis no Apêndice A. O *setup* completo do aplicativo desenvolvido em execução é ilustrado na Figura 21.

Conforme mencionado anteriormente, o trabalho foi desenvolvido utilizando diversos pacotes do ROS Noetic, entre eles o YOLO, Gazebo e RT-GENE, em conjunto com uma

Figura 21 - Setup completo do aplicativo desenvolvido no trabalho em execução



Fonte: Elaborado pela autora (2023).

aplicação empregando a ferramenta de interface gráfica do python Tkinter. A execução da aplicação completa foi dividida em quatro partes: primeira é uma classe que inicia um nó ROS para a obtenção da posição do olhar ou cabeça do usuário através das informações publicadas pelo pacote RT-GENE e receber os eventos de piscada fornecidos pelo RT-BENE.

A segunda é uma classe que inicia um nó ROS para a obtenção dos objetos em cena utilizando a câmera montada no ambiente. Essa classe utiliza o pacote YOLO para extrair os dados de nomes dos itens identificados e suas respectivas posições em *pixel* na imagem fornecida pela câmera.

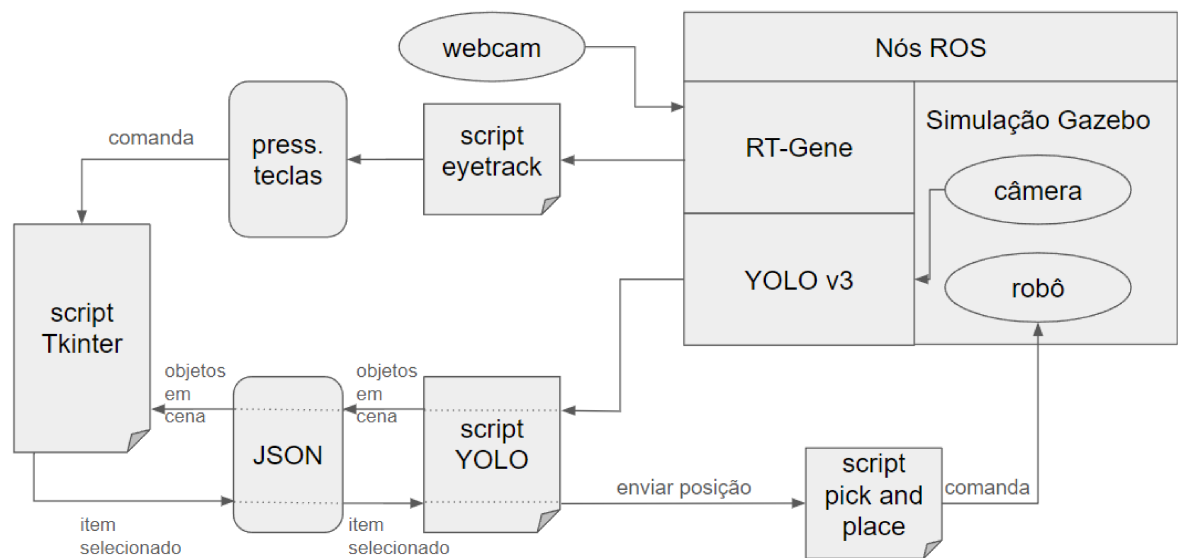
A terceira é um nó ROS criado para a simulação da ação *pick and place* no cenário do Gazebo. Este cenário é composto por duas mesas bem simples: a primeira foi posicionada na frente do braço robótico e contém três itens distintos (banana, maçã e caneca, respectivamente), a segunda se localiza na lateral do manipulador e apenas uma mesa vazia para que o robô possa posicionar livremente o objeto escolhido.

Por fim, utilizou-se o pacote Tkinter do python para a construção da GUI, assim como a lógica para a mudança de botões conforme a vontade do usuário. Não existe uma correlação de ordem entre as quatro partes, esta apenas foi definida para facilitar o entendimento do algoritmo completo.

Sabe-se que cada nó ROS deve rodar individualmente em um *script*, nesse caso um

arquivo .py individual. De modo análogo, o pacote Tkinter também necessita de seu *script* próprio, fazendo-se necessária a implementação de *threads* que executem os programas necessários. Entretanto, estes programas precisam trocar informações entre si. Entre os nós ROS é estabelecida uma comunicação apenas via *subscribers* nos tópicos necessários, mas o tkinker não possui acesso a esta troca de informações. Deste modo, um arquivo .json foi utilizado para armazenar os dados importantes e permitir tal comunicação entre as *threads*. O fluxograma na Figura 22 ilustra a interação do *script* do Tkinter com os nós ROS através do arquivo JSON.

Figura 22 - Fluxograma de execução dos *scripts* para a implementação do *setup* completo



Fonte: Elaborado pela autora (2024).

4.1 OBTENÇÃO DOS OBJETOS EM CENA

A rotina desenvolvida para a identificação dos itens na cena utiliza o algoritmo YOLOv3, descrito anteriormente, em sua versão adequada para o ROS Noetic. Tal algoritmo consiste em uma classe que se divide em três funções primordiais: uma função *getItemPosition* que visa a obtenção das posições dos objetos em cena, uma função chamada de *getBoxParam* para atualizar a lista de itens detectada e uma função chamada *selectedItem* que trata da lógica por trás da manipulação dos dados quando um objeto é selecionado para o *pick and place*.

Para se obter os nomes dos objetos dispostos na cena, apenas os dois primeiros métodos são necessários, *getBoxParam* e *selectedItem*, a função *selectedItem* é descrita na Seção 4.3.1. Estes métodos lidam com manipulação dos dados coletados através do YOLO

para permitir uma troca de mensagens entre a detecção dos objetos com o aplicativo desenvolvido.

Com a possibilidade de se remover ou modificar os itens disponíveis, há a necessidade de manter o algoritmo sempre atualizado com as informações dos objetos de fato presentes em cena. Para isso, define-se inicialmente duas listas vazias para serem posteriormente preenchidas com os dados do YOLO. Uma dessas listas contém os itens habilitados para realizar a ação *pick and place*. A lista é gerada através dos nomes dos arquivos de imagens que representam os objetos, utilizadas para a confecção da GUI proposta. Nota-se que há uma certa limitação neste método, uma vez que as imagens devem ter a nomenclatura exata com correspondência com o pacote YOLO, de modo a evitar erros de detecção.

Para cada caixa delimitada pelo YOLO, verifica-se se o item correspondente dessa caixa está presente na lista de objetos habilitados no aplicativo através dos dados em arquivo JSON. Se a classe estiver habilitada, o algoritmo realiza algumas ações: adiciona o item da caixa à lista de objetos detectados; e em seguida constrói um dicionário chamado *item* contendo o nome, tipo e posição da caixa obtidos de um método chamado. Esse dicionário é então adicionado à segunda lista criada. Após iterar por todas as caixas, atualiza-se o dicionário *data*, contendo os objetos habilitados, obtido do arquivo JSON. Substituindo-o no mesmo arquivo json com um formato indentado.

4.2 OBTENÇÃO DA POSIÇÃO DO OLHAR

O pacote RT-GENE oferece tanto a detecção da posição do olhar quanto a movimentação da cabeça. Os dois métodos foram implementados neste trabalho, sendo possível escolher qual opção de seleção o usuário deseja através da edição de uma variável no arquivo JSON. A execução de ambas as metodologias simultaneamente também é possível.

Apesar da calibração da câmera necessária para utilizar as bibliotecas do RT-GENE e RT-BENE, é preciso calibrar também os valores limites de olhar e movimento de cabeça para o usuário poder utilizar o aplicativo confortavelmente, além de garantir uma execução correta da aplicação.

4.2.1 Calibração

Considerando que essa calibração apenas armazena os devidos valores de movimentação dos olhos/cabeça para quatro direções: cima, baixo, esquerda e direita, sua programação acaba sendo bem simples. O código é composto por três funções: uma função *acquire_data* que visa a aquisição de dados do RT-GENE, uma função *calibrate* para auxiliar o usuário durante a calibração e a função *mainTF* de inicialização do processo e armazenamento dos valores calculados.

A função com o propósito de obter os dados de rastreamento ocular fornecidos pelo

pacote RT-GENE é a que realiza de fato a calibração. Essa recebe uma variável contendo a informação de qual tipo de detecção (olhos ou cabeça) será calibrado. Após inicializar algumas variáveis necessárias, a função entra em um laço em que se obtém o vetor de rotação, em seguida transforma-se esses valores de quatérnions para Euler, e somam-se seus valores atuais aos valores obtidos. Após um determinado número de iterações, neste caso 25, o laço finaliza, retornando um vetor final contendo a média dos valores do vetor de rotação calculado.

Considerando que são quatro direções possíveis de movimentação: cima, baixo, esquerda e direita, a segunda função, que visa direcionar o usuário durante o processo de calibração, exhibe, para cada uma das quatro possíveis posições de cabeça, um texto requisitando que o usuário olhe ou movimente para a direção apontada e então pressionar a tecla enter para iniciar a calibração daquela configuração. Em seguida chama-se a função anterior, para obter o valor médio da direção a calibrar. Após o processo de calibração, retorna-se um vetor contendo os valores calculados para as quatro direções para que o programa os armazene corretamente no arquivo JSON.

4.2.2 Rastreamento Ocular

Ao estabelecer o valor para o método de detecção no arquivo *data.json*, o programa que traduz o movimento do olhar para a seleção dos botões verifica quais destes métodos estão habilitados, e caso estejam, procede para a obtenção das matrizes de transformação de translação e rotação de seus respectivos sistemas de coordenadas, disponibilizadas neste mesmo arquivo, sendo que para a transformação de rotação é necessário uma passar os dados de quatérnions para Euler.

Este novo vetor de transformação é utilizado como parâmetro de uma função de nome *defMovement* para realizar a detecção da direção, com os valores limites de cada eixo, calculados durante a calibração e armazenados em *data.json* para o respectivo método de detecção. Esta função apenas verifica se o ângulo de movimentação do olhar/cabeça ultrapassou um limite estabelecido pela calibração.

A verificação de piscada ocorre paralelamente neste programa. Um *subscriber* lê constantemente os valores do respectivo tópico publicado pelo pacote RT-BENE através de uma função *callback*, e caso ela receba um determinado número N de piscadas ininterruptamente, considera-se como olhos fechados. Em caso afirmativo, a função virtualmente precisa a tecla espaço, considerada como tecla para simular um clique em um botão no aplicativo Tkinter.

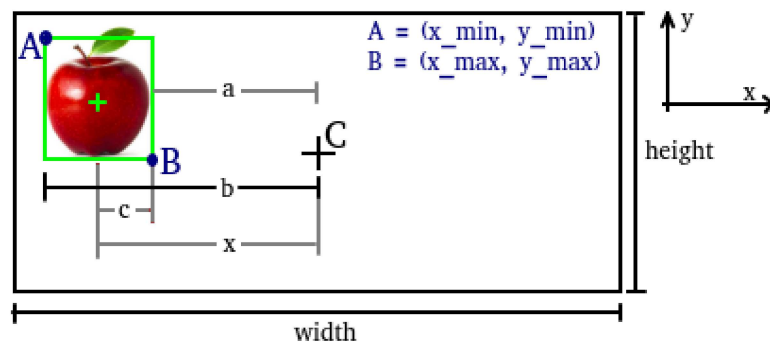
4.3 OBTENÇÃO DAS POSIÇÕES DOS OBJETOS EM CENA

Considerando que objetos diferentes possuem diversas formas e tamanhos, seria necessário que a garra se adaptasse ao item, seja mediante implementações de sensores

de toque, *bumpers* ou outras técnicas. Como o objetivo deste trabalho não engloba tal aplicação de sensoriamento para auxiliar o manipulador, para fins de simplificação, considera-se que os itens a serem capturados estão em caixas padronizadas com pequenas alças para auxiliar o manipulador a segurar o objeto.

A obtenção das posições dos objetos em cena realizada apenas obtendo um vetor (x, y, z) contendo a posição do centro do objeto detectado pelo YOLO. Devido à padronização das caixas, combinado com as coordenadas conhecidas de itens do cenário e as informações extraídas através das *bounding boxes* do YOLO, e tendo o conhecimento prévio das posições de alguns objetos fixos no ambiente, como a câmera e a mesa onde os itens ficam dispostos, é possível calcular a posição do objeto no mundo, conforme exposto para o eixo x na Figura 23.

Figura 23 - Esquema para obtenção da posição do item



Fonte: Elaborado pela autora (2023).

Através da extração de *features* da imagem, o YOLO nos fornece os valores de x_{\min} , y_{\min} , x_{\max} e y_{\max} . Como o tamanho da imagem é conhecido, conseqüentemente seu centro também é, logo é possível obter a distância do centro do objeto até o centro da imagem, ou seja, o valor de x na Figura 23:

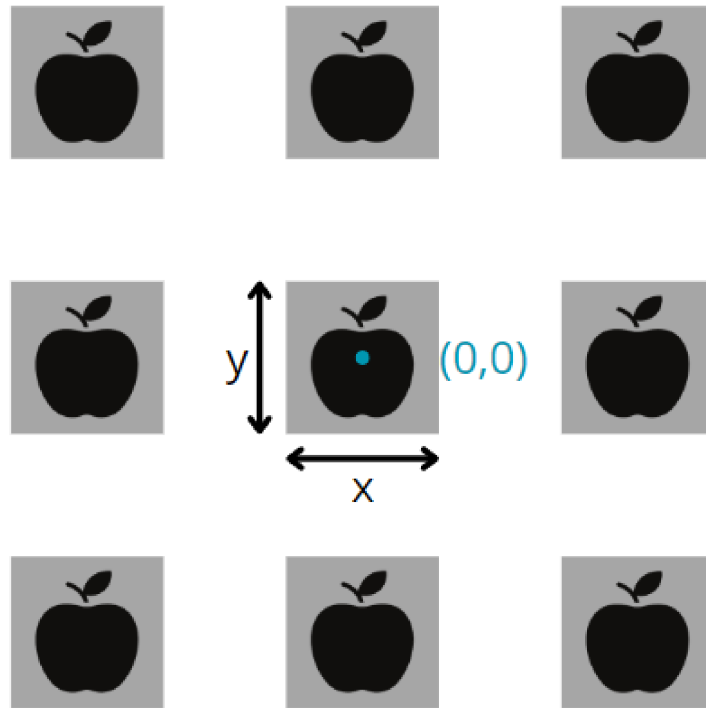
$$x = a + c \quad , \quad c = \frac{b - a}{2} \quad e \quad x = \frac{a + b}{2}. \quad (4.1)$$

Lembrando que o ponto $(0,0)$ da imagem se localiza no superior esquerdo, temos:

$$x_i = \frac{width}{2} - \frac{x_{\min} + x_{\max}}{2} \quad e \quad y_i = \frac{height}{2} - \frac{y_{\min} + y_{\max}}{2}. \quad (4.2)$$

Sendo x_i e y_i em *pixels*. Para comandar o robô as medidas devem estar em metros, logo é necessário converter os valores. Tal conversão se deu por meio de uma calibragem utilizando blocos de $0.1m$. A calibração dos blocos é realizada manualmente utilizando imagens contendo nove blocos idênticos em posições diferentes dispostos de forma matricial 3×3 , ilustrado na Figura 24.

Figura 24 - Disposição dos blocos para realizar a calibração manual



Fonte: Elaborado pela autora (2024).

Os blocos são dispostos de modo que o centro do bloco central fique localizado exatamente baixo da câmera, centralizando-o na imagem. A largura e comprimento (valores x e y) de cada bloco são contados, em *pixels*, para se obter a relação *pixel-cm* nesta configuração de ambiente. Como nesta aplicação os blocos estão bastante próximos, os valores de largura e comprimento de todas as nove posições foram praticamente iguais, portanto, aproximaram-se os cálculos das distâncias dos blocos considerando o caso da caixa centralizada, que representa uma conta mais simples.

Por fim, para adquirir a posição do objeto, basta acrescentar a posição (x,y) da câmera aos valores obtidos de x_i e y_i . O valor de z_i é facilmente determinado com as medidas da mesa e caixas padronizadas.

4.3.1 Seleção de Objeto

Considerando o propósito da aplicação desenvolvida nesta dissertação, é importante lidar com a seleção de um item específico e realizar algumas ações com base nessa seleção. A leitura e carregamento dos dados armazenados anteriormente é de fundamental importância nesta etapa. Ao "clique" no botão *select*, o algoritmo que executa o aplicativo Tkinter identifica qual item é exibido na tela do usuário, o guardando seu nome no arquivo json para posterior comparação com os objetos em cena.

Tal chave contendo a seleção de objeto, sendo essa associada ao aplicativo Tkinter (`["app"]["selected"]`), é utilizada para que o *script* que processa as informações adquiridas do YOLO verifique constantemente se o valor associado à chave do arquivo json, que contém a informação a respeito da seleção de um item pelo usuário, é classificado como verdadeiro, ou seja, contém o nome de algum objeto escrito. Se uma seleção estiver presente, O algoritmo deve prosseguir para a obtenção dos dados daquele item para realizar a ação de *pick and place*, mas ao executar tal operação diretamente, o processo se repete até que o valor da chave mude, representando um problema para a boa execução da aplicação.

Deste modo, para contornar tal inconveniência, o *script* registra o nome do item selecionado, dessa vez no dicionário associado à aplicação do YOLO (`["yolo"]["selected"]`), e então atualiza o valor dessa chave para Falso no dicionário *data*, evitando que o método execute os passos seguintes novamente.

Finalmente, após as operações relacionadas ao JSON, há um bloco *try-except* onde uma tentativa é feita para executar algumas ações com base no nome do item selecionado (se ele existir). Essas ações envolvem a chamada de uma função para realizar a ação *pick and place*, descrita na Seção 4.6.

4.4 APLICATIVO TKINTER

Uma vez que a posição do olhar é identificada, é preciso transformar isto em um comando do usuário. Para isso, foi desenvolvida uma interface gráfica com auxílio da biblioteca Tkinter, mostrado na Figura 25.

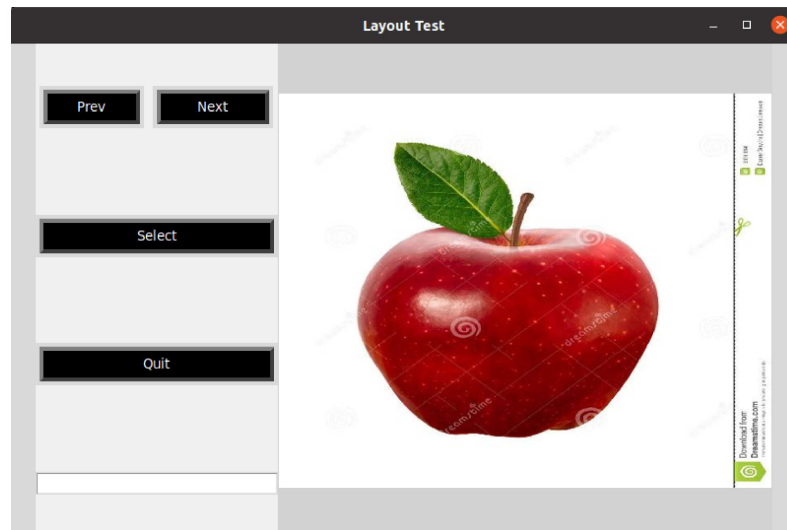
Um menu de opções é atualizado continuamente com os objetos detectados pelo YOLO que constantemente renova o arquivo JSON com os objetos em cena, conforme descrito na Seção 4.3.1.

Devido à proximidade dos botões em conjunto com a acurácia e precisão obtidas da detecção de olhar, nota-se uma certa dificuldade em determinar com precisão o botão escolhido pelo usuário. Tendo isso em mente, adotou-se um método de seleção semelhante a um *joystick*, em que a direção do olhar (cima, baixo, esquerda ou direita) altera o botão a ser pressionado. Utilizado o movimento dos olhos, o usuário irá então escolher um dos botões disponíveis no menu e, fechando o olho por um determinado período, selecionar o botão marcado, possibilitando assim, comandar o robô a pegar o item.

4.5 SIMULAÇÃO NO GAZEBO

A partir da posição do objeto selecionado na imagem é possível realizar a ação de *pick and place*, que consiste em pegar um item em uma posição e levá-lo até outra, neste caso, aproximando o objeto do usuário. Para pegar o item, simulações com dois robôs foram realizadas neste trabalho: o robô YouBot da Kuka, uma base móvel omnidirecional

Figura 25 - Tela inicial do aplicativo desenvolvido no trabalho



Fonte: Elaborado pela autora (2023).

com um manipulador de 5 DoF acoplado, e um braço robótico Panda de base fixa e 7 DoF, desenvolvido pela Franka Emika.

Uma ferramenta amplamente utilizada no ambiente de manipulação robótica é o *MoveIt*. Dentre as várias funcionalidades que a plataforma oferece, duas funções permitem a execução de uma ação *pick and place*: a função *pick*, que executa a atividade de pegar o objeto, e *place* que coloca o item em um ponto escolhido pelo usuário.

As funções padrões *pick* e *place* do *MoveIt* utilizam a ação *grasp* para agarrar e soltar o objeto. Tal processo funciona perfeitamente com o robô Youbot, mas apresenta um problema com o manipulador Panda no ROS Noetic no qual o manipulador não consegue abrir sua garra ao finalizar a ação de *pick and place* devido ao plugin *FrankaGripperSim* de modo que o robô não concluirá a ação e nem a abortará caso o *gripper* permaneça aberto ao finalizar o comando, o que normalmente acontece após pegar o item.

Isso ocorre devido a um problema em um plugin do *FrankaGripperSim* que interfere na interface dos limites das articulações da garra, que permite que o dedo oscile em seus limites, logo, possibilitando que tais juntas ultrapassem seus valores máximos, o que não é bem recebido pelo manipulador. Apenas o fechamento da garra conclui com sucesso ao utilizar este método. Uma solução amplamente utilizada por desenvolvedores e difundida pela própria documentação da Franka é a utilização da função *grasp* do pacote franka para o desenvolvimento de funções *pick and place* próprias conforme a vontade do usuário. Por conseguinte, um novo algoritmo se faz necessário para possibilitar o uso adequado do agarrador do robô.

4.6 ALGORITMO PICK AND PLACE

Considerando um possível cenário em que o usuário utiliza o programa para pegar um objeto que precisa permanecer em uma determinada orientação, como um copo contendo água, implementou-se uma restrição para a orientação do manipulador. Duas soluções para este problema foram propostas. O primeiro programa apenas realiza a movimentação do robô por meio de planejamento de caminhos cartesianos utilizando a função *compute_cartesian_path* do *MoveIt*, melhor descrito na Seção 4.6.2.

O planejamento de caminhos cartesianos apenas computa trajetórias retas de um ponto a outro conforme a sequência de posições. Nesta dissertação, fora as duas movimentações iniciais, que pega e levanta o item da mesa, e as duas últimas que o posiciona na segunda mesa e afasta sua garra do mesmo, a movimentação do braço para levar o objeto de uma mesa à outra não necessita de um planejamento em linha reta, apenas que o braço mantenha a orientação do ser efetuator, deste modo, realizou-se uma busca por alguma técnica que possibilite um planejamento que considera a restrição de orientação e que seja um pouco mais livre para o planejamento deste percurso, como é o caso de implementações de restrições através da biblioteca *Open Motion Planning Library*, em especial a restrição em caixa proposta em (MAEYER'S, 2020).

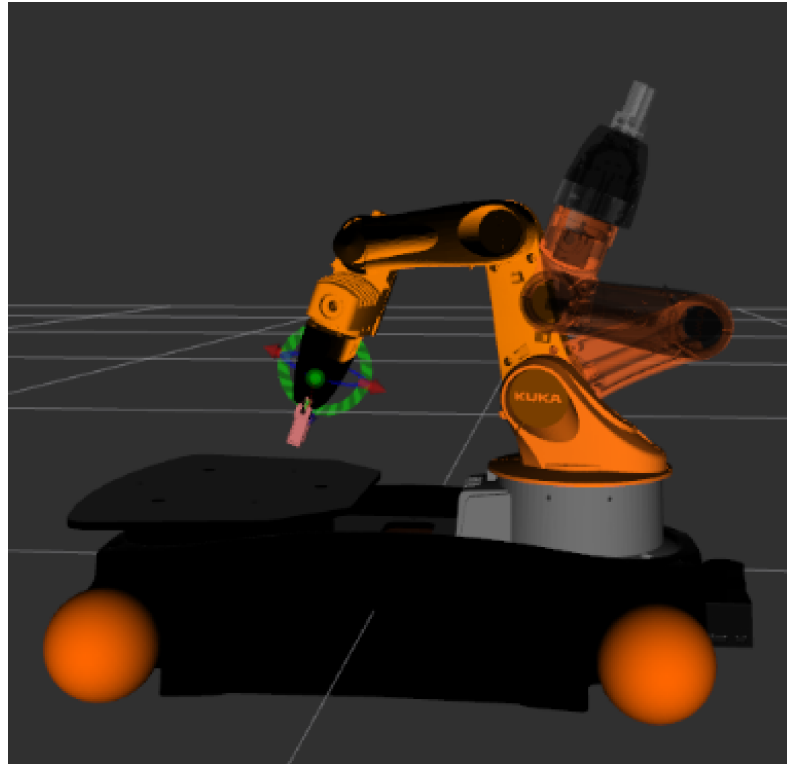
O algoritmo que implementa as restrições utiliza *path constraints*, para restringir a movimentação do *end-effector* do manipulador a um plano ou linha. Este método utiliza um projeto ainda em desenvolvimento por Jeroen De Maeyer baseado em “Cartesian Planning improvements” e “TrajOpt Integration”(MAEYER'S, 2020) para auxiliar as restrições de planejamento da *Open Motion Planning Library* (OMPL), descrito na Seção 4.6.3.

4.6.1 Planejamento básico com o *MoveIt*

Este código foi implementado inicialmente com o robô YouBot em mente. Como este robô possui uma base móvel com uma pequena superfície, a proposta, inicialmente, seria mover o Youbot até a mesa contendo os itens para então selecionar o objeto para que o robô o pegue e deposite em sua base móvel, sendo um exemplo de posicionamento do robô Youbot para largar um item ilustrado na Figura 26.

Evitar colisões é um aspecto fundamental na robótica. Manipular itens utilizando um braço robótico necessita o acréscimo dos objetos presentes no ambiente, a serem evitados, e o item agarrado, que também não deve sofrer uma colisão. A biblioteca do *MoveIt* dá suporte a essas interações ao possibilitar o acréscimo de objetos de colisão. Deste modo, para iniciar o processo de *pick and place*, deve-se adicionar os objetos presentes no ambiente. Para o caso do Youbot, como o destino do item selecionado é ser colocado na base móvel do próprio robô, só há a necessidade da criação de objetos de colisão que representem uma mesa, os itens dispostos na mesma e a câmera no ambiente e os itens

Figura 26 - Exemplo de posicionamento do robô Youbot para largar um item em sua base móvel



Fonte: Elaborado pela autora (2022).

dispostos. No caso do Panda, além dos itens anteriormente citados, necessita-se criar uma segunda mesa, onde os objetos serão posicionados. Tais itens são apenas representações dos objetos "reais" (nesse caso, simulados no ambiente gazebo) com o motivo de evitar colisões. Sua criação pode ser visualizada através do programa Rviz, uma ferramenta de visualização 3D para ROS.

Após a criação do cenário, as etapas do processo de *pick and place* podem ser executadas. No *MoveIt*, o processo de agarrar é executado por meio da interface *MoveGroup*. Para agarrar um objeto em cena, é necessário gerar uma mensagem do tipo *moveit_msgs::Grasp*. Essa mensagem facilita a especificação de posicionamento e orientações essenciais para uma operação de *pick* bem-sucedida. Deste modo, inicialmente deve-se informar as informações necessárias para agarrar e soltar o item, sendo as principais:

- 1 Posição (x, y, z) da ação *grasp*;
- 2 Orientação da ação;
- 3 Direção da ação: Como o manipulador se aproxima ou se afasta da posição escolhida.

Tais informações também são utilizadas para informar quando o manipulador deve abrir ou fechar sua garra. A partir da ação *Grasp*, é possível realizar facilmente as ações

pick e *place*. Para agarrar o item escolhido:

- 1 Fornecer a posição (x, y, z) do objeto e escolher uma orientação do efetuador para pegar o item;
- 2 Definir abordagem antes de pegar: direção e a distância deslocada do efetuador;
- 3 Definir abordagem após pegar: direção e a distância deslocada do efetuador;
- 4 Informar a posição do *end effector* a ser executada antes da abordagem "antes de pegar";
- 5 Informar a posição do *end effector* a ser executada antes da abordagem "após pegar".

Todas essas informações são enviadas para a própria função *pick* do *MoveIt*, junto com o nome do objeto a ser pego, para o algoritmo poder incluí-lo na cinemática do robô durante o processo. A ação *place* é executada bastantemente semelhante à *pick*:

- 1 Fornecer a posição (x, y, z) do objeto e escolher uma orientação do efetuador para largar o item;
- 2 Definir abordagem antes de largar: direção e a distância deslocada do efetuador;
- 3 Definir abordagem após largar: direção e a distância deslocada do efetuador;
- 5 Informar a posição do *end effector* a ser executada antes da abordagem "após largar".

Assim como em *pick*, tais informações também são enviadas para a função *place* juntamente com o nome do objeto, dessa vez para o item ser separado do braço com sucesso.

4.6.2 Planejamento de caminhos cartesianos

O planejamento utilizando o método de caminhos cartesianos se dá pela definição de sequência de percursos com suas respectivas posições iniciais e finais. Como a geração desses percursos ocorre através da interpolação da trajetória com uma resolução escolhida pelo usuário, tal trajeto ocorrerá em linha reta, de modo que, dependendo das escolhas dos pontos finais e iniciais do trajeto, em conjunto com suas respectivas orientações, seja possível a geração de um percurso de modo que o manipulador mantenha a orientação de um de seus eixos sem a necessidade de uma restrição.

Para ser feita a ação de *pick and place*, oito pontos principais foram definidos para o posicionamento do manipulador robótico. Utiliza-se os quatro primeiros para pegar o objeto (realizar a ação *pick*), os dois pontos seguintes movimentam o item de uma mesa para outra, enquanto os dois últimos pontos tem o objetivo de largar a caixa na mesa (ação *place*).

4.6.2.1 Ação *pick*

Considerando o manipulador em uma posição inicial, para comandá-lo a agarrar um item, inicialmente é necessário o mover para uma posição que o possibilite pegar o objeto, abrir sua garra, aproximá-lo do objeto, fechar a garra e, por fim, levantar seu efetuador para retirar o item da superfície em que este se encontra. Considerando que existe a necessidade de abertura e fechamento da garra do robô durante a execução da ação, não é possível planejar o trajeto completo utilizando o *compute_cartesian_path*.

Uma função que realize a ação *pick* foi então desenvolvida para a ação ser executada com sucesso. A sequência de ações executadas pelo algoritmo é:

- Posicionar o efetuador na frente do item com orientação fixa;
- Abrir a garra;
- Aproximar do item;
- Fechar a garra;
- Levantar o item em um valor de z fixo.

Parte da ação *pick* é ilustrado nas Figuras 27a, 27b e 27c. A Figura 27a mostra o manipulador em sua posição inicial, uma configuração de juntas padrão do manipulador Panda. As Figuras 27b e 27c ilustram de fato o processo de *pick* descrito: A Figura 27b mostra o processo de aproximação do manipulador para agarrar o item, enquanto a Figura 27c exibe a posição final deste processo, após o robô levantar o objeto em uma altura pré-definida.

4.6.2.2 Ação *place*

A ação *place* é realizada em apenas dois passos: A computação do caminho cartesiano até a segunda mesa e a abertura da garra para largar o item. Como a posição final do objeto não altera significativamente a pesquisa em si, suas coordenadas finais na segunda mesa são fixas e pré-definidas.

O algoritmo que executa a ação de levar e colocar o item na segunda mesa se inicia criando uma lista vazia, chamada *waypoints*, onde as posições e orientações desejadas serão armazenadas e em seguida o trecho obtém a *pose* (posição e orientação) atual do manipulador. Como as mesas estão próximas do robô, uma movimentação em linha reta de uma mesa à outra pode fazer com que o manipulador apresente dificuldades de realizar sua movimentação, devido a uma maior chance de cair em uma singularidade cinemática, uma vez que tais pontos se encontram muito próximos ao limite de sua área de trabalho. Deste modo, acrescenta-se um ponto intermediário no percurso.

Em seguida, define-se uma sequência de movimentos do robô, ajustando as coordenadas x , y e z da posição, bem como os valores x , y , z e w da orientação em quatérnio. Cada posição e orientação são adicionadas à lista *waypoints*. Levando em conta que este algoritmo se inicia a partir do fim da sequência de ações descrita anteriormente na Seção 4.6.2.1, o desenrolar do algoritmo *place* se dá conforme:

- Mover o efetuador em linha reta até um ponto médio, não variando o valor de z ;
- Mover o efetuador até a posição da mesa, ainda não variando o valor de z ;
- Descer o item até a mesa;
- Abrir a garra;
- Recuar o manipulador para liberar o objeto.

Como não há necessidade de abertura ou fechamento da garra entre a primeira e terceira etapa, o código calcula um caminho cartesiano entre os três primeiros *waypoints* descritos anteriormente, até a etapa anterior à abertura da garra. Especifica-se um passo de 0,01 para a interpolação da trajetória e um limite de salto de um ponto a outro de 5,0 (*jump_threshold*). Isso significa que o robô tentará seguir um caminho suave entre os pontos, evitando movimentos bruscos. Após a abertura da garra, um novo caminho é gerado apenas para recuar o robô.

As Figuras 27d, 27e e 27f ilustram esse processo. A Figura 27d mostra o manipulador após chegar no ponto intermediário definido. A Figura 27e, por sua vez, exibe o robô ao chegar na posição pré-definida da mesa, mas mantendo o mesmo valor de z . Por fim, a Figura 27c mostra o fim da ação, após o manipulador descer o item, abrir a garra e se afastar da mesa.

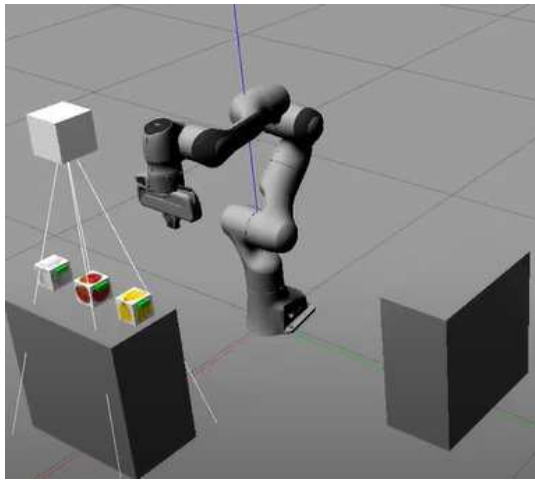
A função *compute_cartesian_path()* do *MoveIt* retorna os valores *plan* e *fraction*. A variável *plan* é o plano de movimento calculado, a trajetória em si. O valor de *fraction* representa, em porcentagem, fração do caminho que foi realmente alcançada, seu valor varia de 0 a 1. A Figura 27 mostra um exemplo da ação de *pick and place* completa, utilizando o método de planejamento de trajetórias cartesianas.

Apesar de funcionar em alguns casos, como nos sucedidos nesta dissertação, tal método não garante a permanência de uma orientação fixa, devido à ausência de restrições impostas ao planejamento. Nessa situação, recomenda-se uma metodologia que considere tais restrições, conforme a proposta na Seção 4.6.3 seguinte.

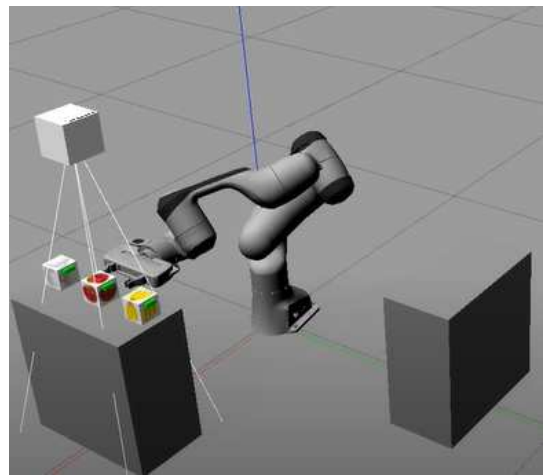
4.6.3 Planejamento Restritivo

A Biblioteca de Planejamento de Movimento Aberto (OMPL, do inglês *Open Motion Planning Library*), conhecida por sua sigla em inglês, é bastante utilizada pelo *framework*

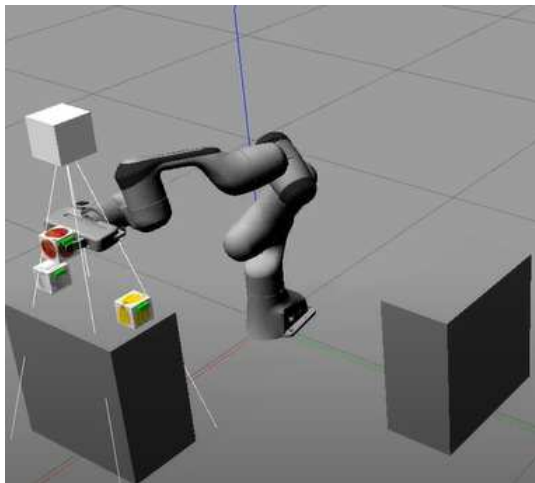
Figura 27 - Sequência de imagens mostrando a ação de *pick and place*



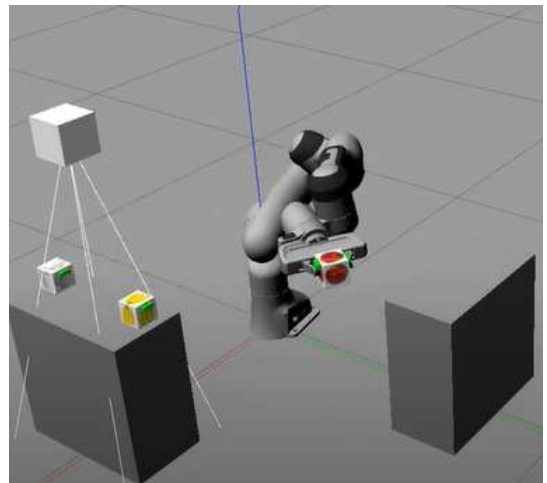
(a) Posição inicial do robô



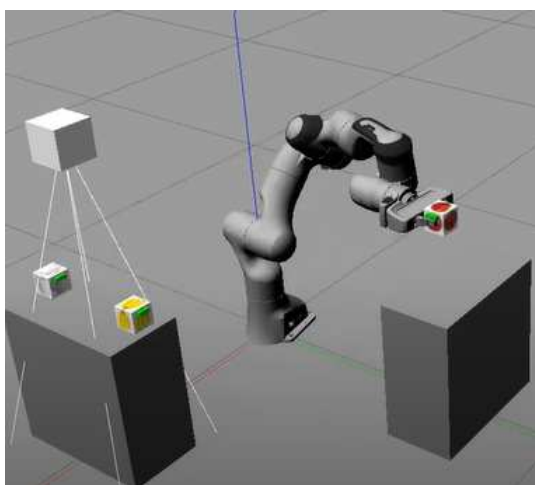
(b) Posição da garra para a ação de agarrar



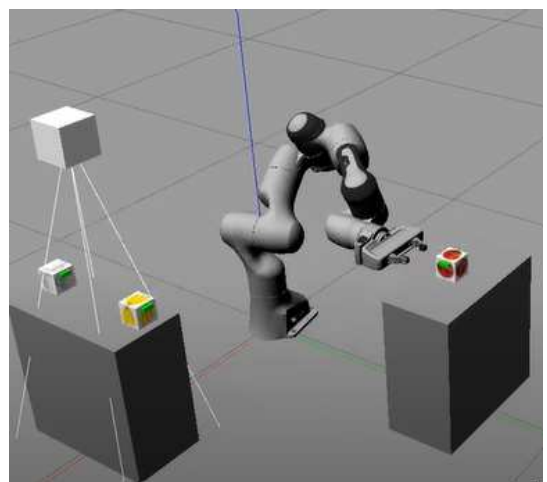
(c) Pegando o item



(d) Movimentação para a segunda mesa



(e) Chegada ao ponto final do caminho



(f) Recuar a garra após deixar o objeto em sua posição final

Fonte: Elaborado pela autora (2023).

MoveIt para a elaboração de trajetórias que sejam imunes a colisões. A movimentação de um robô pode ser restrita devido alguma limitação de seu ambiente, de sua tarefa, como se movimentar com um copo de água cheio sem o derramar, ou uma limitação física própria, como no caso de singularidades cinemáticas que ocorrem quando manipuladores robóticos ficam "presos" no meio de seu percurso por atingir o limite de alguma de suas juntas.

O OMPL incorporou funcionalidades que permitem planejar trajetórias considerando restrições genéricas. Como as restrições abordadas no OMPL não são apenas o desvio de obstáculos ou a otimização de uma função de custo, mas também engloba as restrições de movimentação do robô, os autores da biblioteca utilizaram restrições que utilizam apenas as configurações atuais do robô.

Tais restrições são dissociadas dos algoritmos de planejamento, uma vez que se introduz um "espaço de estados com restrições" em virtude da "restrição no movimento do robô ser definida por alguma função, $f(q) : \mathcal{Q} \rightarrow \mathbb{R}^n$, que mapeia o espaço de estado \mathcal{Q} do robô em um valor vetorial real \mathbb{R}^n " (KAVRAKI, 2018), sendo essa restrição satisfeita quando $f(q) = 0$. Ao realizar amostragens neste espaço de estados com restrições, obtêm-se amostras que satisfazem tais restrições, permitindo um planejamento de caminhos como se as restrições não existissem.

A implementação de (MAEYER'S, 2020) estende o suporte do OMLP para abordar também restrições de tipo caixa, planos e linhas. Entretanto, essas novas implementações de restrições no planejamento do posicionamento do *end-effector* ocasionalmente resultam em mudanças muito bruscas nos valores das juntas, comprometendo a viabilidade da trajetória em um robô real. Tal problema é bastante comum em planejamentos de caminhos com restrições utilizando a biblioteca OMPL, isso é causado pela discretização no cálculo da cinemática inversa do manipulador, onde cada ponto amostrado é individualmente bom e válido, mas a interpolação entre eles leva a esses "saltos" significativos em seus valores de juntas.

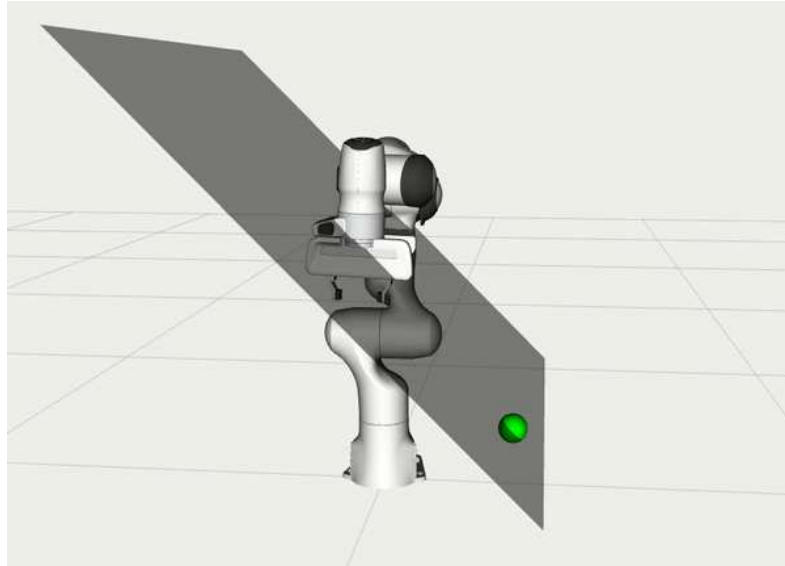
A construção das restrições no código pode ser realizada via funções simples que retornam a região a ser considerada restritiva, utilizando caixas simples, planos ou linhas. O código é projetado para interagir com o *framework MoveIt*.

O processo começa pela criação de uma restrição de posição, neste trabalho chamada pcm, do tipo *MoveIt_msgs.msg.PositionConstraint*. Essa restrição é definida no contexto de um determinado elo de referência e um elo final efetuador.

A restrição é modelada usando uma caixa sólida (BOX) tridimensional (cbox). As dimensões da caixa são especificadas como um vetor de três posições [a, b, c] no eixo x, y e z, respectivamente. No caso de uma restrição planar, mostrada na Figura 28, considera-se uma caixa em que uma de suas dimensões é bem pequena, por exemplo, uma região de dimensão [1.0, 0.0005, 1.0], sendo seus valores dados em metros. Considerando uma restrição em linha, mostrada na Figura 29, duas dessas dimensões seriam diminuídas, por

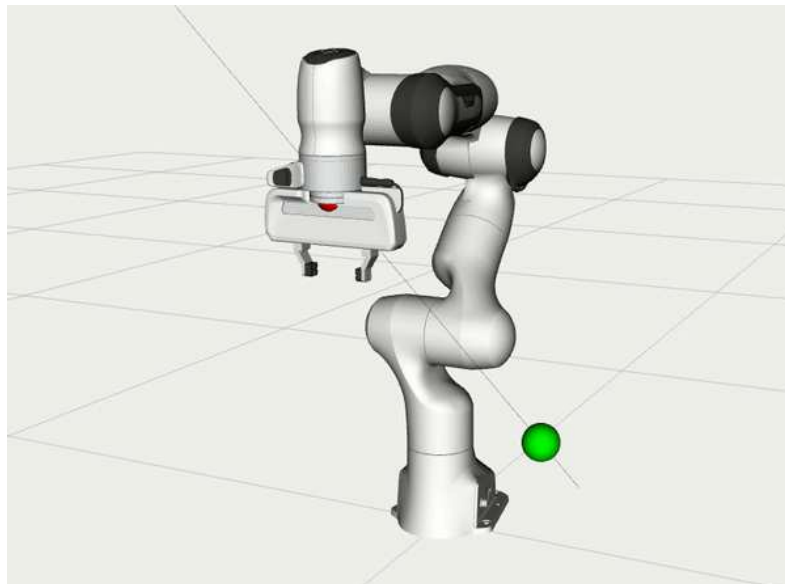
exemplo, uma região com dimensões $[0.0005, 0.0005, 1.0]$, que representa uma linha no eixo z. A região de restrição é definida pela combinação de uma caixa sólida e uma *pose* (*cbbox_pose*) que define a posição e orientação da caixa no espaço.

Figura 28 - Planejamento de caminho com restrição planar



Fonte: *A new approach for planning with path constraints in MoveIt* (MAEYER'S, 2020).

Figura 29 - Planejamento de caminho com restrição em linha



Fonte: *A new approach for planning with path constraints in MoveIt* (MAEYER'S, 2020).

A posição da região de restrição é determinada pela posição atual do efetuador em relação ao link de referência. No final, a função retorna a restrição de posição pcm, que encapsula as informações sobre a caixa sólida e a *pose* associada, prontas para serem usadas no planejamento de trajetórias considerando essas restrições. Com as restrições em mãos, é possível determinar as posições iniciais e finais do *end-effector* do manipulador e

realizar o planejamento da trajetória com as funções próprias do *MoveIt*, respectivamente *set_start_state*, *set_pose_target* e *plan*.

Para executar a ação de *pick and place* utilizando essa metodologia, um algoritmo bastante semelhante ao anteriormente descrito na Seção **4.6.2** foi implementado. Acrescentam-se as restrições de movimentação em linha para levantar o objeto da primeira mesa na ação *pick* (último item da sequência de ações, descrito conforme a Seção 4.6.2.1) e para descê-lo na segunda mesa ao final da ação *place* (terceiro item da sequência de ações, descrito conforme a Seção 4.6.2.2). Uma restrição de movimento em plano também foi implementada no lugar de um ponto intermediário durante a movimentação de uma mesa a outra, no processo da ação *place* (primeiro item da sequência de ações, descrito conforme a Seção 4.6.2.2).

5 RESULTADOS E DISCUSSÕES

Considerando a implementação de dois manipuladores distintos nesta dissertação, a seção de resultados foi dividida em três partes:

- Simulações com o robô Youbot;
- Simulações com o manipulador Panda;
- Eficiência das diferentes interfaces de seleção.

Todos os testes desta dissertação foram realizados apenas com um indivíduo saudável devida pandemia de COVID-19 ocorrida durante a confecção deste trabalho.

5.1 YOUBOT

Inicialmente realizaram-se simulações considerando ambientes diferentes para avaliar alguns aspectos do algoritmo, como acurácia da obtenção da posição do item, comportamento do robô em situações diferentes e comparação da conduta do aplicativo para itens diferentes. Cada posição foi simulada 50 vezes.

- Mesmo item em posições diferentes. Nesse teste será avaliado a capacidade do método de pegar o objeto em diferentes posições;
- Mais de um item disposto na mesa. Nesse teste será avaliado a capacidade do método de distinguir os objetos e evitar impacto com outros.

5.1.1 Mesmo item em posições diferentes

De modo a validar os cálculos obtidos na Equação 4.2 realizaram-se algumas simulações de *pick and place* com um mesmo item (banana) em diversos lugares diferentes da mesa. Os resultados obtidos estão dispostos na Tabela 3.

Observa-se que os menores e maiores erros para o eixo x são respectivamente $0.002m$ obtido na simulação 3 e $0.012m$ da simulação 5, sendo a média do erro dada por $0.007m$. Analogamente, para o eixo y os menores e maiores valores são, respectivamente, $0.003m$ da primeira simulação e $0.011m$ obtido na quinta. A média do erro do eixo y foi $0.008m$. O manipulador se comportou como esperado em todas as simulações no ambiente RViz, que representa um ambiente ideal, não considerando variáveis físicas, falhando apenas na quinta, coincidentemente a que possui erros mais altos. Apesar de obter sucesso em quase todas as simulações no RViz, no ambiente Gazebo, em contrapartida, o manipulador não obteve sucesso para pegar nenhum dos objetos.

Tabela 3 – Valores e erros dos eixos x e y para as simulações de *pick and place* utilizando o mesmo item.

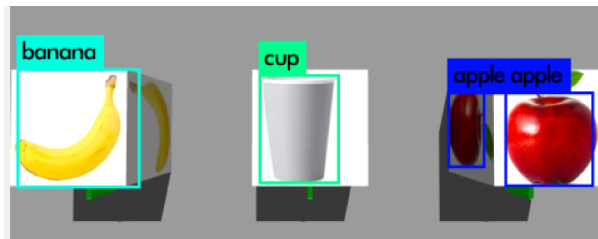
Posição	Eixo	Posição original(m)	Valores adquiridos(m)	Erro (m)
1	x	0.55	0.541	0.009
	y	0.0	0.003	0.003
2	x	0.55	0.543	0.007
	y	0.23	0.236	0.006
3	x	0.65	0.647	0.002
	y	0.25	0.242	0.008
4	x	0.65	0.647	0.003
	y	-0.20	-0.209	0.009
5	x	0.45	0.438	0.012
	y	-0.20	-0.211	0.011
6	x	0.45	0.441	0.009
	y	0.25	0.240	0.010

Fonte: Elaborada pela autora (2022).

5.1.2 Mais de um item

Selecionando os itens maçã, banana e copo, dispostos conforme a Figura 30, realizou-se a tentativa de *pick and place* para cada um dos itens citados. Os resultados das simulações exibem-se na Tabela 4.

Figura 30 - Resultado do YOLO considerando a disposição dos três itens na mesa



Fonte: Elaborada pela autora (2022).

Tabela 4 – Valores e erros dos eixos x e y para as simulações de *pick and place* utilizando itens diferentes.

Item	Eixo	Posição original(m)	Valores adquiridos (m)	Erro (m)
maçã	x	0.55	0.531	0.019
	y	-0.25	-0.261	0.011
banana	x	0.55	0.543	0.007
	y	0.25	0.254	0.006
copo	x	0.55	0.542	0.008
	y	0.0	0.012	0.012

Fonte: Elaborada pela autora (2022).

Observa-se que o item que apresentou maior erro foi a maçã e, embora estejam à mesma distância do centro, a banana obteve os menores erros. Tal ocorrido impossibilitou o manipulador a pegar a maçã. Nota-se também que, apesar do copo estar disposto mais à esquerda do centro do bloco, sua detecção ainda forneceu um valor possível para

que o manipulador o pegue na simulação do RViz. Novamente, no ambiente Gazebo, o manipulador não obteve sucesso para pegar nenhum dos objetos.

Apesar de parecer um valor baixo para uma implementação simples, uma diferença de milímetros é suficiente para representar uma dificuldade para o robô YouBot, uma vez que este possui a abertura de sua garra muito pequena e sua cadeia cinemática, neste trabalho, inclui a base móvel, que acumula consideravelmente mais erros devido o atrito com o chão, o que o impossibilita de pegar o item com sucesso.

Dada a importância de um controle maior do posicionamento da garra deste robô no ambiente para completar as tarefas propostas, há a necessidade de se avaliar técnicas que permitam que a garra sofra um ajuste fino de posição conforme a mesma se aproxima do objeto. O uso de sensores posicionados no braço pode permitir esse ajuste de maneira a garantir que o erro seja sempre o mínimo possível, garantindo a pegada segura. A implementação de sensoriamento e técnicas de fusão de sensores para obter a posição do robô no meio, como lidars, algoritmos de localização e mapeamento simultâneos (SLAM, do inglês *Simultaneous Localization And Mapping*) e filtros de Kalman também podem ser efetivas para mitigar o problema.

A motivação para o desenvolvimento desta pesquisa é o desenvolvimento da aplicação completa: obter dados de rastreamento ocular com o intuito de controlar uma interface de modo que um usuário comande um braço robótico para realizar uma ação *depick and place*, deste modo, tais implementações para corrigir e melhorar o desempenho do YouBot são postergadas para trabalhos futuros e o robô utilizado no projeto substituído pelo manipulador de base fixa e 7 DoF Panda da empresa Franka Emika.

5.2 PANDA

Para qualificar o desempenho do programa, realizaram-se algumas simulações utilizando o manipulador Panda, considerando ambientes diferentes de modo a avaliar alguns aspectos do algoritmo, como a acurácia da obtenção da posição do item, comportamento do robô em situações diferentes e comparação da conduta do aplicativo para itens diferentes. Considerando também que o usuário pode optar por pegar um objeto que precisa permanecer em uma orientação específica, como copos cheios de água, adotou-se uma restrição de orientação para a movimentação do manipulador, analisando-se também o comportamento do mesmo nesse tipo de situação. Três análises foram executadas:

- Desempenho dos Algoritmos Pick and Place;
- Efeito do posicionamento do objeto a ser pego;
- Planejamento sucessivo do trajeto.

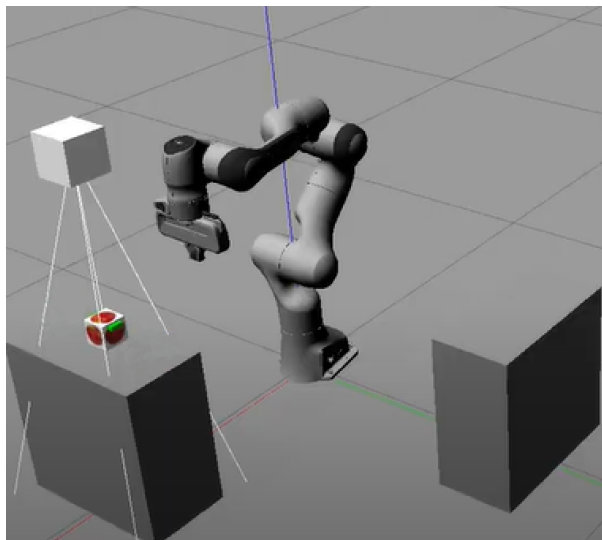
5.2.1 Desempenho dos Algoritmos Pick and Place

Como descrito na Seção 4.6, três algoritmos foram desenvolvidos para essa tarefa: o primeiro (básico) utiliza as funções *pick* e *place* já disponibilizadas pelo *MoveIt*, o segundo (cartesiano) emprega o planejamento do caminho por meio de caminhos cartesianos e o terceiro (restritivo) usa o projeto de Jeroen De Maeyer, descrito na Seção 4.6.3, para implementar restrições de movimento.

- **Básico** - Por apresentar problemas com a garra do Panda e não considerar a orientação do efetuador durante a trajetória, foi utilizado apenas por motivos de comparação;
- **Cartesiano** - Utilizado para implementar a ação de *pick and place* considerando abertura e fechamento da garra e analisar uma possibilidade de orientação constante;
- **Restritivo** - Utilizado para aumentar a liberdade de movimentação do braço de uma mesa à outra, implementando de fato as restrições de orientação do efetuador.

De modo a verificar o desempenho dos algoritmos desenvolvidos para realizar a ação de *pick and place*, para cada método implementado executou-se cinquenta simulações com um mesmo item (maçã) posicionado no centro da mesa, como mostrado na Figura 31. O desempenho dos algoritmos foi classificado em três parâmetros:

Figura 31 - Cenário utilizado para avaliar o desempenho dos três algoritmos utilizados pna ação de *pick and place*



Fonte: Elaborada pela autora (2023).

- **Acerto:** o braço executou 100% da trajetória como esperado. Um exemplo em vídeo é exibido no vídeo "*pick_and_place_action_example.mp4*" disponibilizado no Apêndice A;

- **Erros:** o braço não conseguiu terminar o planejamento ou apresentou grande dificuldades no planejamento da trajetória. Um exemplo em vídeo é exibido no vídeo "*pick_and_place_bad_plan.mp4*" disponibilizado no Apêndice A;
- **Parcial:** o braço completou quase todo o percurso ou completou sem obedecer à restrição de orientação do *end-effector*. Um exemplo em vídeo é exibido no vídeo "*pick_and_place_parcial.mp4*" disponibilizado no Apêndice A.

Apesar de não apresentar nenhuma dificuldade ao concluir a tarefa e executá-la com média de tempo de 31.339969 ± 4.238 s, o algoritmo básico do *MoveIt* não colocou o objeto na mesa em nenhuma das tentativas. Como já citado anteriormente, o *MoveIt* não consegue abrir a garra do Panda, apenas a fechar, fazendo com que, independentemente de se iniciar o trajeto já com a garra aberta, o robô não deixa o item na mesa, condicionando-o a falhas, mesmo que seu planejamento completo seja executado mantendo a orientação do efetuador. Em suas simulações parciais, o programa não respeitou a restrição de orientação para o *end-effector* mostrando que, mesmo que outro robô contemple a execução completa das funções *pick* e *place* do próprio *MoveIt*, este não garante uma orientação fixa, caso necessário.

Os resultados obtidos para os algoritmos cartesiano e restritivo estão dispostos na Tabela 5.

Tabela 5 – Desempenho dos algoritmos *pick and place* e seu tempo de execução médio e desvio padrão para 50 simulações

Algoritmo	Acertos (%)	Erros (%)	Parcial (%)	Tempo de execução (s)
cartesiano	66	12	22	50.7049 ± 6.228
restritivo	28	56	16	161.994044 ± 90.929

Fonte: Elaborada pela autora (2023).

O algoritmo cartesiano obteve uma taxa de sucesso próxima à do básico (desconsiderando o problema com a garra do manipulador), sendo seus erros devido ao processo de planejamento da trajetória calculada em oito partes (as cinco primeiras realizam a ação *pick*, descritas na Seção 4.6.2.1, as três últimas realizam a ação *place*, descritas na Seção 4.6.2.2), sendo cada parte planejada após o término da etapa anterior, de modo que o início de um dos trajetos é a posição final do manipulador gerada pelo percurso anterior.

Desta forma, é possível que a posição final da trajetória calculada anteriormente possa resultar em um planejamento que caia em uma singularidade cinemática, fazendo com que o manipulador, apesar de conseguir realizar a etapa precedente com sucesso, acabe parando no meio do trajeto devido a restrições físicas do robô.

As execuções parciais são frutos de eventuais dificuldades do manipulador recuar a garra após soltar o item no final do processo pelo mesmo motivo. Apesar de uma taxa de sucesso próxima a do básico, o tempo de execução aumentou pouco mais de 60% para o método cartesiano em comparação com as simulações de funções básicas da biblioteca, mesmo considerando que parte dessas simulações não se completam devido singularidades cinemáticas encontradas. Tal acréscimo no tempo provavelmente se deve à mudança na construção das funções *pick* e *place*.

A biblioteca *MoveIt* é programada em C++, uma linguagem de programação significativamente mais rápida que Python, a linguagem utilizada neste trabalho. Como as funções de *pick* e *place* do *MoveIt* são construídas em C++, e as funções desenvolvidas neste trabalho, apesar de utilizarem as funções, também em C++, da biblioteca, elas são constantemente chamadas em um *script* programado em Python, o que pode diminuir drasticamente o desempenho do algoritmo, como observado na Tabela 5. O acréscimo no desvio padrão entre os métodos básico e cartesiano, por sua vez, se deve às trajetórias que caem em singularidades cinemáticas no caso cartesiano, representadas pelos erros e execuções parciais, que acabam sendo mais rápidas em comparação com as execuções resultadas em sucesso.

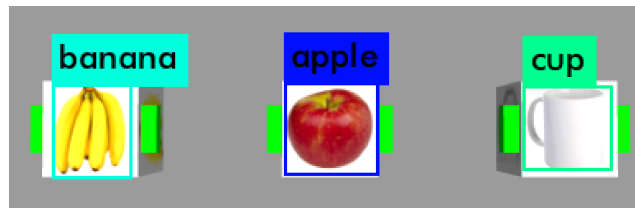
Os erros do modelo restritivo se devem às restrições impostas ao robô que podem fazer com que o valor das juntas variem bruscamente ao passar de um ponto a outro, sem respeitar o limite físico do próprio robô, de modo que o braço não consiga concluir o caminho determinado pelo algoritmo, necessitando assim de um novo planejamento em tempo real para se posicionar conforme lhe foi comandado, o que também aumentou consideravelmente o tempo de execução deste método. A possibilidade de grandes "saltos" nos valores das juntas também é visto no valor obtido para o desvio padrão, significativamente maior em comparação aos outros métodos, uma vez que o algoritmo pode funcionar como esperado, ou precisar reposicionar suas juntas, demandando um tempo considerável nesta tarefa. Tais problemas já são de conhecimento do desenvolvedor e algumas soluções já foram sugeridas, entretanto, como o projeto continua em desenvolvimento, podem não funcionar em todos os casos.

O fato de que a função do *MoveIt* necessária para largar o objeto apresenta um erro que impossibilita o manipulador de largar o item na mesa, zerando o sucesso do método, e as possíveis variações bruscas nos valores dos ângulos das juntas do manipulador proporcionadas pelo modelo restritivo inviabilizam a utilização desses métodos. Desta forma, considerando os resultados obtidos e por motivos de simplificação da avaliação do desempenho da aplicação, para as próximas simulações adotou-se apenas o algoritmo cartesiano.

5.2.2 Efeito do posicionamento do objeto a ser pego

Tendo em mente os resultados do experimento anterior, e sua limitação sobre possíveis percursos contendo singularidades cinemáticas, é interessante avaliar o comportamento do algoritmo para posições diferentes do item na mesa. Posicionando os itens maçã, banana e copo, dispostos conforme a Figura 32, realizou-se a tentativa de *pick and place* com o algoritmo cartesiano para cada um dos itens citados. Os resultados de 50 simulações para cada um dos três objetos exibem-se na Tabela 6.

Figura 32 - Imagem do YOLO com a disposição dos três itens na mesa



Fonte: Elaborada pela autora (2023).

Tabela 6 – Valores e erros dos eixos x e y para as simulações de *pick and place* utilizando itens diferentes

Item	Acertos (%)	Erros (%)	Parcial (%)	Tempo de execução (s)	Erro de posição (x,y) (m)
maçã	62	24	14	35.224347 ± 6.971	-0.006148 -0.000512
banana	30	48	22	36.534726 ± 7.143	-0.007684 0.148566
copo	52	36	12	38.101397 ± 6.212	-0.005635 -0.147541

Fonte: Elaborada pela autora (2023).

Apesar do manipulador possuir 7 DoF, ainda é possível que, conforme se mova, acabe caindo em uma singularidade cinemática, o impossibilitando concluir o percurso, uma vez que um possível requisito para que o braço se depare com tal problema é a trajetória que o manipulador escolhe nos primeiros passos da ação de *pick and place*.

Todas as simulações neste trabalho foram supervisionadas. Analisando os resultados obtidos em conjunto com uma análise qualitativa das simulações, notou-se que o manipulador se depara com singularidades cinemáticas mais frequentemente ao iniciar sua movimentação para a esquerda, onde se encontra a banana. É possível então presumir que objetos à esquerda do robô podem representar maior dificuldade para a conclusão da ação, por possuir maiores chances de iniciar o planejamento com uma configuração de juntas que o comprometa no futuro, como é o exemplo da banana que obteve metade do número

de acertos da maçã, localizada no centro da mesa. A xícara, apesar de estar a uma mesma distância do centro da mesa que a banana, se manteve com valores intermediários, mais próximos ao da maçã, apenas por estar posicionada à direita. Entretanto, análises mais detalhadas devem ser realizadas para concluir tais limitações do braço.

Ao analisar os tempos de execução, nota-se uma grande proximidade entre os valores tanto das médias quanto dos desvios padrões. Entretanto, devido o aumento considerável de tentativas fracassadas, principalmente com a banana, esperava-se um tempo de execução menor e maior desvio padrão para a banana e copo em comparação com a maçã, mas não é o que se observa nos resultados. Uma provável motivação de tal ocorrência seria um maior tempo gasto para o manipulador se posicionar durante o processo de *pick*, o que aumentaria o tempo médio de execução.

Como o algoritmo passa consideravelmente mais tempo se posicionando para pegar o item em comparação com o tempo total gasto no restante do percurso, como é possível ver no vídeo "*pick_and_place_action_example.mp4*" disponibilizado no Apêndice A, ao considerar o possível aumento no tempo conforme descrito anteriormente, explica-se então os valores de desvio padrão mais próximos que o esperado.

5.2.3 Planejamento sucessivo do trajeto

O planejamento escolhido retorna dois valores: a trajetória do braço e a porcentagem desta concluída com sucesso pelo manipulador. Deste modo, é possível implementar um código que, ao verificar que o robô falhou ao tentar completar sua trajetória por completo, o algoritmo a recalcula por inteiro, até que a simulação retorne uma taxa de 100% de sucesso. O processo de replanejamento e as tentativas de movimentação do manipulador no Rviz podem ser visto no vídeo "*pick_and_place_cartesian_complete.mp4*" disponibilizado no Apêndice A.

Para este teste foram realizadas 25 novas simulações com cada objeto disposto no centro da mesa e os resultados dispostos na Tabela 7.

Tabela 7 – Tempo de execução do planejamento para obtenção de 100% do trajeto

Item	Tempo médio de execução (s)
maçã	97.496447 ± 69.734
banana	182.636365 ± 158.763
copo	137.925552 ± 137.637

Fonte: Elaborada pela autora (2023).

Em todas as simulações para este caso obtiveram-se 100% de sucesso na conclusão da tarefa de *pick and place*, como esperado. Como os erros cometidos ao tentar pegar a banana são bem maiores em comparação com os outros itens, conforme a Tabela 6,

já era de se esperar um tempo de execução do planejamento mais elevado, conforme a Tabela 7. De modo análogo, como a maçã obteve mais acertos ao rodar o planejamento individualmente, conseqüentemente seu tempo total de execução foi menor.

Observa-se também um aumento significativo no desvio padrão do tempo para a obtenção de um caminho executável pelo manipulador. A taxa de sucesso do método utilizado não é tão expressiva quanto o desejado, desta forma, é possível que em algumas simulações o robô acerte na primeira tentativa, enquanto outras talvez demore um pouco mais para obter sucesso. Deve-se considerar também as simulações onde o braço se imobiliza em uma singularidade cinemática onde, apesar do tempo de execução da simulação diminuir em relação às tentativas concluídas com êxito, a maioria das tentativas com erro representam resultados parciais, ou seja, o manipulador apresentou dificuldades apenas no final do percurso, aumentando o desvio padrão dos resultados.

5.3 EFICIÊNCIA DAS DIFERENTES INTERFACES DE SELEÇÃO

Visando avaliar o desempenho do algoritmo que utiliza *eye-tracking* e movimentação de cabeça para selecionar os objetos, três situações diferentes foram estudadas: a seleção dos objetos utilizando as setas do teclado, a seleção através do movimento ocular e, por fim, selecionando o botão com o movimento da cabeça. Tal experimento foi realizado seguindo um roteiro em que, inicialmente, o usuário seleciona o botão "*next*", seguido do "*prev*", descendo até o botão "*quit*", sem o selecionar, para então voltar ao botão "*select*" e por fim, clicar. A sequência de ações é executada da seguinte maneira:

1 Next - Direita

2 Prev - Esquerda

3 Select - Baixo

4 Quit - Baixo

5 Select - Cima

6 Clique - Piscar

O resultado para esses testes considerando 25 simulações para cada um dos métodos propostos (rastreamento ocular, movimentação da cabeça e utilização do teclado) está disponível na Tabela 8.

Os processos alternativos demoram mais para executar as ações em comparação com o teclado, justificados pelo processamento de imagem por trás dos métodos sugeridos no trabalho. Apesar da proximidade dos valores obtidos com o olhar e movimento de cabeça para selecionar os botões no aplicativo, o movimento ocular retorna a seleção mais

Tabela 8 – Tempo de execução da seleção dos botoes do aplicativo para 25 simulações

Método	Tempo médio de execução (s)
teclado	4.17534 ± 0.709
olhos	18.383087 ± 2.524
cabeça	18.264283 ± 1.696

Fonte: Elaborada pela autora (2023).

rapidamente, mas eventualmente pode confundir o olhar para baixo com um clique, o que pode atrasar um pouco o usuário. O movimento de cabeça não possui o problema do olhar, entretanto sua detecção demora um pouco mais em comparação com o *eye-tracking*.

Deste modo, apesar do rastreamento ocular satisfazer sozinho as necessidades do usuário de maneira mais rápida, como existe a chance de confusão do olhar para baixo com um clique, é importante haver uma alternativa mais robusta a tal inconveniência, como a seleção por movimento de cabeça (caso o usuário possua esse controle motor).

Ao se comparar o método proposto com o uso de um teclado para controlar a interface, há um aumento de aproximadamente 350% no tempo médio de execução, equivalendo a cerca de 12 segundos. Assim, considerando o elevado custo computacional dos métodos de *eyetracking*, tal implementação ainda representa uma opção viável para um método alternativo de se utilizar a interface de um aplicativo, principalmente se considerar o objetivo de melhorar a acessibilidade do programa ao possibilitar que pessoas com dificuldades motoras utilizem o software.

6 CONCLUSÕES

Este trabalho apresentou uma metodologia para comando de atuadores robóticos com uso de movimento dos olhos. A mesma foi avaliada através do uso de simulações, sendo possível analisar seu potencial para futuros desenvolvimentos, bem como estabelecer um paralelo com outros métodos disponíveis na literatura.

Analisando os resultados obtidos dos testes com o robô Youbot, apesar da tarefa proposta não necessitar de grande precisão, como a precisão necessária para os robôs que performam cirurgias, um erro de alguns milímetros gerou dificuldades para o primeiro manipulador selecionado, uma vez que o acréscimo da base móvel em sua cadeia cinemática acumula bastante erro com o passar do tempo devido o atrito com o chão e a abertura das garras deste robô é muito pequena, sendo necessária a implementação de sensoriamento e localização no robô para o possibilitar agarrar bem os objetos.

Em contrapartida, os resultados demonstram que, em uma sala pequena, a resolução de uma *webcam* comum permite determinar a posição do objeto no mundo sem que seus erros afetem a ação do manipulador Panda. Entretanto, verifica-se que o robô apresenta dificuldades em executar sua tarefa para algumas posições, normalmente quando o item se localiza mais à esquerda do centro da mesa. O problema está relacionado com o método utilizado para o cálculo da trajetória do manipulador, que pode resultar em uma singularidade cinemática durante o percurso e impossibilitar sua finalização, entretanto mais análises devem ser feitas para comprovar tal problema.

Apesar da dificuldade, é possível contornar o problema via uma implementação simples de tentativa e erro. Tal resolução, apesar de efetiva, pode ser computacionalmente custosa e demorada, sendo interessante avaliar melhor o problema para possivelmente implementar uma nova metodologia ou otimizar o processo. A implementação de um novo método de planejamento de trajetória ou uma estratégia para retirar o manipulador de uma singularidade cinemática, ainda obedecendo às restrições de planejamento, se mostram como possíveis alternativas que possuem potencial de contornar o problema encontrado nesta dissertação e com menos gasto computacional em comparação com a técnica de tentativa e erro implementada.

Os testes realizados utilizando os métodos alternativos propostos neste trabalho para operar a interface gráfica mostram que, apesar do acréscimo do tempo de execução para realizar comandos no programa, a utilização de rastreamento ocular representa uma maneira viável de interação de uma pessoa com o ambiente ao seu redor. Foi possível observar também que existe uma pequena inconveniência do rastreamento ocular em comparação com a movimentação de cabeça, uma vez que é possível que o programa confunda o olhar para baixo com olhos fechados, evidenciando a necessidade de robustez do *eye-tracking*.

Apesar do RT-GENE proibir uso comercial de seu algoritmo, o desenvolvimento de rastreadores oculares baseados em webcam estão crescendo, representando uma alternativa mais barata e com maior escalabilidade em comparação com os rastreadores baseados em IR. Independentemente de suas limitações de luminosidade e menor acurácia, esta técnica mostrou grande potencial para implementações em algoritmos de *pick and place* com maior grau de complexidade por utilizar manipuladores de maior DoF (7 e 8 DoF como testados neste trabalho) com restrições de movimentos bastante limitantes, uma vez que o robô deveria manter sua orientação durante toda a trajetória.

6.1 TRABALHOS FUTUROS

Em trabalhos futuros, pretende-se a implementar outros modelos de robôs, como o YouBot (GMBH, 2010), além da realização de testes com os robôs reais. Cogita-se também avaliar o uso de visão estereoscópica para determinar a posição dos objetos no espaço, uma vez que este trabalho considera que a posição dos itens disponíveis estão posicionados em uma mesa de posição fixa com altura (eixo z) conhecida, o que não é um dado sempre conhecido. Desta maneira seria possível determinar a posição dos mesmos no espaço sem nenhuma outra informação externa, ou conhecimento prévio do ambiente.

Pretende-se ainda avaliar técnicas que permitam que a posição da garra sofra um ajuste fino de posição uma vez que a mesma se aproxime do objeto. O uso de sensores posicionados no braço pode permitir esse ajuste de maneira a garantir que o erro seja sempre o mínimo possível, garantindo a pegada segura.

Propõe-se também a realização de mais testes com maior número de participantes com diferentes capacidades de mobilidade para uma posterior avaliação subjetiva da aplicação desenvolvida, além da possível inserção de uma cadeira de rodas como base do manipulador e novas restrições no espaço de trabalho do robô, uma vez que haveria pessoas por perto.

REFERÊNCIAS

- ABBASIMOSHAELI, A.; KNUDSEN, F.; KERN, T. A. Application of eye-tracking for a 3-dof robot assisted medical system. In: *2022 10th RSI International Conference on Robotics and Mechatronics (ICRoM)*. [S.l.: s.n.], 2022. p. 503–508.
- ABBASIMOSHAELI, A.; WINKEL, M.; KERN, T. A. Design, simulation and evaluation of two coding programming languages for an eye-tracking controlling system for a three degrees of freedom robot useful for paralyzed people. *e-Prime - Advances in Electrical Engineering, Electronics and Energy*, v. 2, p. 100065, 2022. ISSN 2772-6711. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S2772671122000377>>.
- ALSHARIF, S.; KUZMICHEVA, O.; GRAESER, A. Gaze gesture-based human robot interface. In: *Technische Unterstützungssysteme, die die Menschen wirklich wollen*. [S.l.: s.n.], 2016.
- BACK-END, T. R. *What is ROS?* 2022. <<https://roboticsbackend.com/what-is-ros/>>.
- BALDASSIN, V.; LORENZO, C. F. G.; SHIMIZU, H. E. Tecnologia assistiva e qualidade de vida na tetraplegia : abordagem bioética. *Revista Bioética*, v. 26, p. 574–586, 12 2018.
- BAN, S. et al. Persistent human–machine interfaces for robotic arm control via gaze and eye direction tracking. *Advanced Intelligent Systems*, v. 5, n. 7, p. 2200408, 2023. Disponível em: <<https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202200408>>.
- BANNAT, A. et al. A multimodal human-robot-interaction scenario: Working together with an industrial robot. In: JACKO, J. A. (Ed.). *Human-Computer Interaction. Novel Interaction Methods and Techniques*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 303–311. ISBN 978-3-642-02577-8.
- BESL, P.; MCKAY, N. D. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 14, n. 2, p. 239–256, 1992.
- BROWNLEE, J. *A Gentle Introduction to Object Recognition With Deep Learning*. 2021. <<https://machinelearningmastery.com/object-recognition-with-deep-learning/>>.
- BUSS, S. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Transactions in Robotics and Automation*, v. 17, 05 2004.
- COLEMAN, D. T. et al. Reducing the barrier to entry of complex robotic software: a moveit! case study. *JOURNAL OF SOFTWARE ENGINEERING IN ROBOTICS*, 2014.
- CORTACERO, K.; FISCHER, T.; DEMIRIS, Y. Rt-bene: A dataset and baselines for real-time blink estimation in natural environments. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*. [S.l.: s.n.], 2019.
- CRAIG, J. *Introduction to Robotics: Mechanics and Control*. Pearson/Prentice Hall, 2005. (Addison-Wesley series in electrical and computer engineering: control engineering). ISBN 9780201543612. Disponível em: <<https://books.google.com.br/books?id=MqMeAQAAIAAJ>>.
- EYE, E. *Tobii Pro Glasses 3*. 2020. <<https://www.edgeeye.com.br/tobii-pro-glasses-3/>>.

FARNSWORTH, B. *What is Eye Tracking and How Does it Work?* 2019. <<https://imotions.com/blog/learning/best-practice/eye-tracking-work/>>.

FARNSWORTH, B. *Eye Tracker Prices – An Overview of 15+ products.* 2022. <<https://imotions.com/blog/learning/product-news/eye-tracker-prices/>>.

FERRAZ, A. T. *Eyetracking App for Robot Arm Manipulation (Git repository).* 2023. <https://github.com/cheopis/eyetracking_app_robot_manipulator>.

FERRAZ, A. T. et al. Robotic actuator control through eye-tracking camera to assist people with physical limitations. In: *2023 Latin American Robotics Symposium (LARS), 2023 Brazilian Symposium on Robotics (SBR), and 2023 Workshop on Robotics in Education (WRE)*. [S.l.: s.n.], 2023. p. 11–16.

FISCHER, T. *RT-GENE & RT-BENE: Real-Time Eye Gaze and Blink Estimation in Natural Environments (Git repository).* 2021. <https://github.com/Tobias-Fischer/rt_gene>.

FISCHER, T.; CHANG, H. J.; DEMIRIS, Y. Rt-gene: Real-time eye gaze estimation in natural environments. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. [S.l.: s.n.], 2018.

GALLAGHER, J. *What is Object Detection? The Ultimate Guide.* 2023. <<https://blog.roboflow.com/object-detection/>>.

GEWEHR, B.; ANTONIUCCI, J.; RODRIGUES, A. A interface recurso de tecnologia assistiva e usuário e o impacto na funcionalidade e inclusão social. In: MEDOLA, F. O.; PASCHOARELLI, L. C. (Ed.). *Tecnologia Assistiva: Pesquisa e Conhecimento - I*. [S.l.]: Canal 6 Editora, 2018. p. 15–21.

GIRSHICK, R. et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2014. p. 580–587.

GMBH, K. L. *KUKA youBot Research and Application Development in Mobile Robotics.* 2010. <<https://www.generationrobots.com/img/Kuka-YouBot-Technical-Specs.pdf>>.

JENSEN, O. B. *Webcam-Based Eye Tracking vs. an Eye Tracker [Pros & Cons].* 2022. <<https://imotions.com/blog/learning/best-practice/webcam-eye-tracking-vs-an-eye-tracker/>>.

KAVRAKI, L. *Constrained Planning.* 2018. <<https://ompl.kavrakilab.org/constrainedPlanning.html>>.

KONG, Z. et al. Eye-tracking-based robotic arm control system. In: *2022 International Conference on Computer Engineering and Artificial Intelligence (ICCEAI)*. [S.l.: s.n.], 2022. p. 663–667.

LYNCH, K.; PARK, F. *Modern Robotics: Mechanics, Planning, and Control.* Cambridge University Press, 2017. ISBN 9781316609842. Disponível em: <<https://books.google.com.br/books?id=86G0nQAACAAJ>>.

- MAEYER'S, J. D. *GSoC: Motion planning with general end-effector constraints in MoveIt*. 2020. <<https://moveit.ros.org/moveit/2020/09/10/ompl-constrained-planning-gsoc.html>>.
- MANJARI, K.; VERMA, M.; SINGAL, G. A survey on assistive technology for visually impaired. *Internet of Things*, Elsevier, v. 11, p. 100–188, 2020.
- MUNCE, S. et al. Impact of quality improvement strategies on the quality of life and well-being of individuals with spinal cord injury: a systematic review protocol. *Systematic reviews*, v. 2, p. 14, 02 2013.
- NELSON, J. *Your Comprehensive Guide to the YOLO Family of Models*. 2021. <<https://blog.roboflow.com/guide-to-yolo-models/>>.
- OLIVI, L. R. *Notas de Aula da disciplina de Manipuladores Robóticos*. [S.l.]: UFJF, 2019.
- OLIVI, L. R. *Notas de Aula da disciplina de Robótica Móvel*, UFJF. 2019.
- PALINKO, O. et al. Robot reading human gaze: Why eye tracking is better than head tracking for human-robot collaboration. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Press, 2016. p. 5048–5054. Disponível em: <<https://doi.org/10.1109/IROS.2016.7759741>>.
- PASARICA, A. et al. Remote control of an autonomous robotic platform based on eye tracking. *Advances in Electrical and Computing Engineering*, v. 16, p. 95–100, 11 2016.
- PATACCHIOLA, M.; CANGELOSI, A. Head pose estimation in the wild using convolutional neural networks and adaptive gradient methods. *Pattern Recognition*, v. 71, p. 132–143, 2017. ISSN 0031-3203. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0031320317302327>>.
- RAKHMATULIN, I. *A review of the low-cost eye-tracking systems for 2010-2020*. Zenodo, 2020. Disponível em: <<https://doi.org/10.5281/zenodo.4082164>>.
- REDDY, A. C. Difference between denavit - hartenberg (d-h) classical and modified conventions for forward kinematics of robots with case study. In: *International Conference on Advanced Materials and manufacturing Technologies (AMMT)*. [S.l.: s.n.], 2014.
- REDMON, J. *Darknet: Open Source Neural Networks in C*. 2013–2016. <<http://pjreddie.com/darknet/>>.
- REDMON, J. *YOLO: Real-Time Object Detection*, <https://pjreddie.com/darknet/yolo/>. 2021. Disponível em: <<https://pjreddie.com/darknet/yolo/>>.
- REDMON, J. et al. You only look once: Unified, real-time object detection. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, 2016. p. 779–788. ISSN 1063-6919. Disponível em: <<https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.91>>.
- REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv*, 2018.
- ROBOTICS, O. *ROS - Robot Operating System*. 2013–2016. <<https://www.ros.org>>.
- ROBOTICS, O. *Gazebo Features and Benefits*. 2022. <<https://gazebosim.org/features>>.

- ROBOTICS, P. *MoveIt Concepts*. 2013. <<https://moveit.ros.org/documentation/concepts/>>.
- ROBOTS, S. O. *OMNI-WHEEL ROBOT - FUZZY*. 2014. <https://www.societyofrobots.com/robot_omni_wheel.shtml>.
- SCALERA, L. et al. Human–robot interaction through eye tracking for artistic drawing. *Robotics*, v. 10, n. 2, 2021. ISSN 2218-6581. Disponível em: <<https://www.mdpi.com/2218-6581/10/2/54>>.
- SHARMA, V. K. et al. Eye gaze controlled robotic arm for persons with severe speech and motor impairment. In: . New York, NY, USA: Association for Computing Machinery, 2020. (ETRA '20 Full Papers). ISBN 9781450371339. Disponível em: <<https://doi.org/10.1145/3379155.3391324>>.
- SICILIANO, B. et al. *Robotics: Modelling, Planning and Control*. Springer London, 2010. (Advanced Textbooks in Control and Signal Processing). ISBN 9781846286414. Disponível em: <<https://books.google.com.br/books?id=jPCAFmE-logC>>.
- SIEGWART, R.; NOURBAKHSI, I.; SCARAMUZZA, D. *Introduction to Autonomous Mobile Robots, second edition*. MIT Press, 2011. (Intelligent Robotics and Autonomous Agents series). ISBN 9780262015356. Disponível em: <<https://books.google.com.br/books?id=4of6AQAQBAJ>>.
- STIEFELHAGEN, R. et al. Enabling multimodal human-robot interaction for the karlsruhe humanoid robot. In: *IEEE Transactions on Robotics, Special Issue on Human-Robot Interaction*. [S.l.: s.n.], 2007. v. 23, p. 840–851.
- SUNNY, M. S. H. et al. Eye-gaze control of a wheelchair mounted 6dof assistive robot for activities of daily living. *Journal of NeuroEngineering and Rehabilitation*, v. 18, 2021.
- TERAOKA, C. *Introdução ao Robot Operating System (ROS)*. 2022. <<https://blog.eletragate.com/introducao-ao-robot-operating-system-ros/>>.
- TOBII. *Tobii Pro X2 Eye Tracker Installation and Configuration Video*. 2015. <<https://tobii.23video.com/tobii-pro-x2-eye-tracker-installation>>.
- TOBII. *How do Tobii eye trackers work?* 2024. <https://connect.tobii.com/s/article/How-do-Tobii-eye-trackers-work?language=en_US>.
- VARGAS, L. V.; LEITE, A. C.; COSTA, R. R. Overcoming kinematic singularities with the filtered inverse approach. *IFAC Proceedings Volumes*, v. 47, n. 3, p. 8496–8502, 2014. ISSN 1474-6670. 19th IFAC World Congress. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S147466701642954X>>.
- XU, B. et al. Continuous hybrid bci control for robotic arm using noninvasive electroencephalogram, computer vision, and eye tracking. *Mathematics*, v. 10, n. 4, 2022. ISSN 2227-7390. Disponível em: <<https://www.mdpi.com/2227-7390/10/4/618>>.

APÊNDICE A – Vídeos

Para melhor compreensão do trabalho executado, 6 vídeos estão disponibilizados em <<https://drive.google.com/drive/folders/1JD6SqEi5FxsIbk-FjciEKdtdtmeuLm7p?usp=sharing>>. Destes, 2 vídeos mostram o funcionamento do controle do aplicativo através do rastreamento ocular (app_yolo_eye.mp4) e movimentação de cabeça (app_yolo_head.mp4).

Dos 4 vídeos restantes, 3 ilustram as possíveis ações que o manipulador pode executar durante o planejamento de sua trajetória: a primeira em que o robô fica preso logo no início do trajeto (pick_and_place_bad_plan.mp4), a opção em que o robô quase completa o trajeto, mas não consegue deixar o item na mesa (pick_and_place_parcial.mp4), e o planejamento completo com sucesso (pick_and_place_action_example.mp4).

Por fim, um vídeo é destinado para ilustrar todo o processo de execução da aplicação (full_app.mp4) simultaneamente.