INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
Bahia

Campus
Vitória da Conquista

COEEL
Coordenação de
Engenharia Elétrica

COORDENAÇÃO DE ENGENHARIA ELÉTRICA - COEEL

PROJETO FINAL DE CURSO - PFC

Navegação Social com Restrições Baseada em uma nova Percepção Robocêntrica

FÁBIO ÁBDON DE ALMEIDA LEITE

Vitória da Conquista-BA

06 de Dezembro de 2023

FÁBIO ÁBDON DE ALMEIDA LEITE

Navegação Social com Restrições Baseada em uma nova Percepção Robocêntrica

Projeto Final de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Instituto Federal de Educação, Ciência e Tecnologia da Bahia, *campus* Vitória da Conquista, como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Dr. José Alberto Diaz Amado

Coorientador: Dr. João Erivandro Soares Marques

Vitória da Conquista-BA

06 de Dezembro de 2023

FICHA CATALOGRÁFICA ELABORADA PELO SISTEMA DE BIBLIOTECAS DO IFBA, COM OS DADOS FORNECIDOS PELO(A) AUTOR(A)

L533 Leite, Fábio Ábdon Almeida

Navegação Social com Restrições Baseada em uma nova Percepção Robocêntrica: / Fábio Ábdon Almeida Leite; orientador José Alberto Diaz Amado; coorientador João Erivandro Soares Marques -- Vitória da Conquista : IFBA, 2023.

76 p.

Trabalho de Conclusão de Curso (Engenharia Elétrica) -- Instituto Federal da Bahia, 2023.

1. Automação. 2. Zona proxêmica. 3. Detecção de Objetos. I. Diaz Amado, José Alberto, orient. II. Soares Marques, João Erivandro, coorient. III. TÍTULO.

CDD/CDU

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA BAHIA
Av. Sérgio Vieira de Mello, 3150 - Bairro Zabelê - CEP 45078-900 - Vitória da Conquista - BA - www.portal.ifba.edu.br

ATA

ATA DE APROVAÇÃO PARA FINS DE REGISTRO ESCOLAR

NAVEGAÇÃO SOCIAL COM RESTRIÇÕES BASEADA EM UMA NOVA PERCEPÇÃO ROBOCÊNTRICA

FÁBIO ÁBDON DE ALMEIDA

A presente monografia de Projeto Final de Curso (PFC), apresentada em sessão realizada no laboratório G1, horário 14:00, na data de 06 **do mês de Dezembro de 2023**, foi avaliada como adequada para a obtenção do Grau de Bacharel em Engenharia Elétrica e julgada **APROVADA com média final 9,5 (nove, cinco)** pela banca examinadora.

Banca examinadora,

Prof. Dr. José Alberto Diaz Amado (Orientador)IFBA,
campus Vitória da Conquista

Prof. Dr. João Erivando Soares Marques (Coorientador)
IFBA, *campus* Vitória da Conquista

Prof. Me. Cléia Libarino Santos IFBA,
campus Vitória da Conquista

Prof. Dr. Marcelo Mendonça dos Santos
IFBA, *campus* Vitória da Conquista

Vitória da Conquista - Bahia

Documento assinado eletronicamente por **JOSE ALBERTO DIAZ AMADO**, **Membro da Unidade**, em 13/02/2024, às 11:37, conforme decreto nº 8.539/2015.

Documento assinado eletronicamente por **JOAO ERIVANDO SOARES MARQUES**, **Membro da Unidade**, em 13/02/2024, às 11:49, conforme decreto nº 8.539/2015.

Documento assinado eletronicamente por **MARCELO MENDONCA DOS SANTOS**, **Professor(a) do Ensino Básico, Técnico e Tecnológico - EBTT**, em 14/02/2024, às 20:19, conforme decreto nº 8.539/2015.

Documento assinado eletronicamente por **CLEIA SANTOS LIBARINO**, **Membro da Unidade**, em 15/02/2024, às 09:28, conforme decreto nº 8.539/2015.

A autenticidade do documento pode ser conferida no site http://sei.ifba.edu.br/sei/controlador_externo.php?acao=documento_conferir&acao_origem=documento_conferir&id_orgao_acesso_externo=0 informando o código verificador **3376056** e o código CRC **83E215C2**.

Dedico esta obra à Deus, à minha família e aos meus professores.

"O próprio Senhor irá à sua frente e estará com você; ele nunca o deixará, nunca o abandonará. Não tenha medo! Não se desanime!" [Deuteronômio 31:8]

AGRADECIMENTOS

Agradeço primeiramente a Deus, por me dar perseverança em toda essa caminhada até aqui. A minha namorada Mariana que me acompanhou nos momentos mais difíceis e também nas conquistas. Ao meu pai Rômulo, minha mãe Vanusia e meu irmão Eduardo que me apoiaram e foram pacientes durante todo esse tempo.

Agradeço ao meu professor orientador Dr. José Alberto Diaz Amado que me acompanhou neste trabalho por três anos, dando sugestões e ajudando diretamente no desenvolvimento desse trabalho.

Agradeço aos demais professores da instituição que compartilharam o conhecimento de forma que eu conseguisse aprender.

Agradeço ao IFBA por todas as oportunidades.

Agradeço aos meus colegas, Manoel Pedro Lessa Andrade, James Reis Silva Xavier e Jonas Machado Santana de Jesus que estiveram comigo durante toda a pesquisa, que vocês trilhem um caminho de sucesso.

Agradeço a todos os alunos do GIPAR que contribuíram com o desenvolvimento deste trabalho.

Agradeço a todos que contribuíram direta ou indiretamente para que esse sonho se torne realidade.

RESUMO

O avanço de tecnologias capazes de operar autonomamente com níveis de precisão próximos ao desejado está em ascensão. A aplicação de técnicas de aprendizado de máquina introduz uma nova dimensão à computação e à robótica, ambas diretamente beneficiadas pela presença de máquinas inteligentes. Neste projeto, promovemos uma adaptação na detecção de objetos para aprimorar a navegação social em sistemas de robótica autônoma. Para isso, categorizamos os objetos identificados com base em suas características físicas (ser vivo dinâmico, ser não vivo dinâmico, ser vivo estático e ser não vivo estático) a fim de classificá-los em um contexto social. Dessa forma, é gerada uma área de segurança em torno desses objetos, que deve ser respeitada durante suas interações. Os resultados dos métodos de validação foram apresentados. Assim, pôde-se concluir que a estrutura apresentada é viável para ser implementada com foco na navegação social em robôs móveis.

Palavras-chave: Detecção de Objetos, Zona de Segurança, Navegação Social, Interação Homem-Máquina

ABSTRACT

The advancement of technologies capable of operating autonomously with precision levels close to the desired is on the rise. The application of machine learning techniques introduces a new dimension to computing and robotics, both directly benefiting from the presence of intelligent machines. In this project, we promoted an adaptation in object detection to enhance social navigation in autonomous robotic systems. To achieve this, we categorized identified objects based on their physical characteristics (living dynamic, inanimate dynamic, living static, and inanimate static) in order to classify them in a social context. Consequently, a safety area is generated around these objects, which must be respected during their interactions. The results of validation methods were presented. Thus, it was concluded that the presented framework is feasible to be implemented with a focus on social navigation in mobile robots.

Keywords: Object Detection, Safety Zone, Social Navigation, Human-Machine Interaction

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Glossário: Símbolos e Siglas

Notação	Descrição	Páginas
X_{med}	Ponto médio da Bouding Box no eixo X	19, 21, 22
X_{min}	Ponto mínimo da Bouding Box no eixo X	18, 19
$X_{o.m}$	Coordenada Y do objeto em relação ao mapa	24, 27, 28
$X_{o.r.2}$	Nova coordenada X do objeto em relação ao robô	26, 27
$X_{o.r}$	Coordenada X do objeto em relação ao robô	ix, 22, 23, 25, 26
X_r	Coordenada X do robô em relação ao mapa	23, 27
Y_{max}	Ponto maxim da Bouding Box no eixo X	18, 19
Y_{max}	Ponto máximo da Bouding Box no eixo Y	18, 19
Y_{med}	Ponto médio da Bouding Box no eixo Y	19, 21, 22
Y_{min}	Ponto mínimo da Bouding Box no eixo Y	18, 19
$Y_{o.m}$	Coordenada Y do objeto em relação ao mapa	24, 27, 28
$Y_{o.r.2}$	Nova coordenada Y do objeto em relação ao robô	26, 27

Notação	Descrição	Páginas
$Y_{o.r}$	Coordenada Y do objeto em relação ao robô	ix, 22, 23, 25, 26
Y_r	Nova coordenada Y do robô em relação ao mapa	23, 27
α	Alfa, primeria letra do alfabeto grego	25, 27
β	Beta, segunda letra do alfabeto grego	25
γ	Gama, terceira letra do alfabeto grego	27
COEEL	*Coordenação do Curso de Engenharia Elétrica do IFBA campus Vitória da Conquista*	i
cx	Ponto central da resolução horizontal da câmera	22
cy	Ponto central da resolução vertical da câmera	22
d	Distância entre a câmera e o objeto, em polegada	21
df	Disntância focal da câmera	21, 22
dist	Distância entre o objeto e a câmera, informado pela câmera de profundidade	21, 22, 25–27
GIPAR	Grupo de Inovação e Pesquisa em Automação e Robótica, IFBA Vitória da Conquista	13
IHM	Interação Homem-Máquina	2
p	Comprimento do objeto na imagem, em pixels	21

Notação	Descrição	Páginas
RGB-D	Câmera que fornece dados de profundidade (D) e cor (RGB) com saída em tempo real	13
ROS	Robot Operating System	11, 13
RViz	Ferramenta de vizualização que permite visualizar os sensores dos robôs e os estados internos.	32, 33
w	Comprimento do objeto, em polegada	21
X	Eixo X do plano cartesiano	21
Y	Eixo Y do plano cartesiano	21
YOLO	You Only Look Once	6–8, 13, 15, 16, 28, 31, 32
ZED	Câmera estéreo com função RGB-D	13, 15, 16

Sumário

Capítulo 1

Introdução

Com o progresso da humanidade em um mundo globalizado, a configuração das máquinas acelerou-se após a primeira revolução industrial, de acordo com as demandas e o suporte tecnológico em termos de materiais e conhecimento. A incorporação de sistemas na rotina diária das pessoas ajusta-se às perspectivas adotadas em cada geração.

Podemos destacar essa ideia através de diversas teorias, como a Lei de Moore (SHALF, 2020), formulada por Gordon E. Moore. Esta lei estabelece uma conexão histórica entre microchips e processadores, proporcionando uma base para a compreensão e previsão da evolução de máquinas e sistemas. Consequentemente, antecipamos uma mudança cíclica a cada dois anos nos microchips, acompanhada por um aumento de 100% em sua capacidade em comparação com os modelos anteriores.

Assim, observamos uma tendência e como ela deve ser ajustada para garantir que o crescimento do processamento seja equilibrado para atividades não complicadas e tarefas simples do cotidiano humano. O uso e o avanço de técnicas de Aprendizado de Máquina (ELBADAWI; GAISFORD; BASIT, 2021; SHUJA et al., 2021) estão inaugurando um novo cenário em nosso mercado de sistemas. Essa nova compreensão de arquiteturas inteligentes, com o sistema aprendendo por si só, trouxe uma perspectiva renovada à nossa compreensão do mundo.

No século XXI, testemunhamos a efetiva inserção desses sistemas no cotidiano de qualquer pessoa que possua ao menos um smartphone conectado à internet, viabilizando a exploração de informações por meio de interações muitas vezes intuitivas. A análise dessa interação emerge como uma preocupação central

para os desenvolvedores, impulsionando a realização de novas pesquisas com o objetivo de aperfeiçoar a integração desses sistemas aos dispositivos.

Atualmente, estamos começando a considerar a relevância da robótica social(HENSCHEL; LABAN; CROSS, 2021) e sua aplicação direta na IHM (SØRAA et al., 2021) com exemplos já aplicados em ambientes como shopping centers, hotéis, aeroportos e hospitais (BLINDHEIM et al., 2022). Esse novo serviço desempenha um papel crucial ao criar arquiteturas robóticas com informações e treinamentos essenciais, incorporando inteligência artificial (GUPTA et al., 2021) para aprimorar e facilitar as atividades designadas.

Assim, é imperativo que os desenvolvedores reflitam não apenas sobre a construção de uma máquina capaz de desempenhar suas atribuições, mas que o façam da melhor maneira possível, levando em consideração questões que não podem ser racionalizadas ou mensuradas por meio de código, incluindo emoções e suas interações (GRATEROL et al., 2021; HEREDIA et al., 2022). Nesse sentido, foi elaborado um trabalho que busca oferecer praticidade assertiva, aliada a uma nova perspectiva robocêntrica (QUIROZ et al., 2022).

Dessa forma, é essencial que os desenvolvedores ponderem não apenas sobre a criação de uma máquina capaz de executar suas tarefas atribuídas, mas que o façam de maneira exemplar, levando em consideração questões que não podem ser racionalizadas ou mensuradas por meio de código, como emoções e suas complexas interações (GRATEROL et al., 2021; HEREDIA et al., 2022). Com esse propósito, foi desenvolvido um trabalho que busca eficiência prática aliada a uma nova abordagem robocêntrica (QUIROZ et al., 2022).

Este trabalho introduz uma arquitetura que realiza classificações objetivas em ambientes compartilhados por robôs e humanos. Essas classificações são divididas em quatro categorias relevantes para a mobilidade do robô, distinguindo entre objetos dinâmicos e não dinâmicos. Isso permite um monitoramento dos movimentos futuros do robô para evitar restrições à movimentação das pessoas próximas.

Para essa nova classificação, definimos os seguintes grupos: objetos dinâmicos - seres inanimados (por exemplo, carro), objetos dinâmicos - seres vivos (por exemplo, pessoas), objetos estáticos - seres inanimados (por exemplo, cadeira), e objetos estáticos - seres vivos (por exemplo, flor). A criação de um sistema simplificado pode desempenhar um papel de extrema importância no desenvolvimento de sistemas com hardware, facilitando a execução de suas tarefas.

Outro ponto que influenciou a construção da arquitetura foi a navegação em relação às zonas proxêmicas (BILIUS; VATAVU; MARQUARDT, 2021), que precisam ser ajustadas conforme diferentes distinções de objetos, especialmente no que diz respeito às pessoas. Essas zonas foram estudadas pelo antropólogo Edward Hall (BROWN, 2001) e desempenham um papel significativo na interação entre robôs e humanos, sobretudo no processo de navegação robótica (BILIUS; VATAVU; MARQUARDT, 2021). Considerar essas zonas pode resultar em uma integração mais eficaz de máquinas inteligentes em nosso cotidiano.

Dessa forma, essas técnicas foram integradas para operar em conjunto de maneira otimizada, proporcionando contribuições recíprocas para a mobilidade do robô social. A praticidade da detecção de objetos (FANG; WANG; REN, 2019) introduz uma abordagem mais leve e objetiva à navegação, direcionando o olhar de forma mais direta aos objetos para a construção de Zonas Proxêmicas naturais e garantindo que sejam respeitadas em suas interações.

Assim, podemos simplificar a relação entre várias classes identificadas por meio da classificação de redes neurais. Essa contribuição é crucial para a robótica, considerando a necessidade de um menor processamento computacional em comparação com a abordagem convencional para o desenvolvimento de diversas tarefas de forma isolada. Além disso, ela pode desempenhar um papel importante na robótica social ao criar zonas proxêmicas em ambientes compartilhados com humanos, promovendo maior fluidez nas interações sociais.

A implementação desta pesquisa foi realizada em uma cadeira de rodas motorizada adaptada para navegar de forma autônoma, representando um avanço significativo na integração de tecnologias inovadoras para melhorar a mobilidade e a autonomia de pessoas com dificuldades de locomoção. Este projeto visa proporcionar uma experiência mais independente e acessível, abrindo novas possibilidades para aqueles que enfrentam desafios na mobilidade diária.

1.1 Objetivos

1.1.1 Objetivo Geral

Trabalhar na percepção robótica da cadeira de rodas, focada na detecção e classificação de obstáculos em 3D no ambiente onde está inserida.

1.1.2 Objetivos Específicos

1) Pesquisar e treinar um algoritmo de percepção robótica com enfoque na detecção de objetos;

2) Desenvolver algoritmos capazes de gerar zonas de segurança ao redor de cada classe detectada;

3) Implementar algoritmos de percepção em ambientes simulados da cadeira de rodas;

4) Realizar a integração do sistema de detecção de objetos à cadeira de rodas autônoma.

1.2 Justificativa

Este trabalho se justifica pela crescente importância da interação entre robôs e seres humanos em ambientes sociais e de serviço. À medida que a robótica móvel se torna mais presente em nosso cotidiano, é essencial que os robôs sejam capazes de navegar e interagir de forma segura e eficiente com as pessoas ao seu redor.

A abordagem proposta visa preencher uma lacuna na área de robótica móvel ao se concentrar na previsão de posição e na navegação social. Atualmente, os sistemas de navegação autônoma muitas vezes priorizam a otimização de rotas e evitam obstáculos de maneira geral, sem considerar a natureza das interações humanas e os espaços sociais.

Ao criar um software que utilize dados de câmera RGB-D e técnicas de detecção de objetos, o projeto pretende permitir que os robôs não apenas evitem obstáculos, mas também entendam a presença e a intenção das pessoas ao seu redor. Isso é fundamental para que os robôs possam agir de maneira natural e previsível em ambientes sociais, garantindo segurança tanto para os seres humanos quanto para os próprios robôs.

Além disso, ao se concentrar em robôs de serviço, o trabalho se alinha com a tendência crescente de aplicação de robôs em setores como hospitalar, assistência domiciliar e varejo. A capacidade dos robôs de navegar de maneira autônoma e socialmente inteligente é um diferencial competitivo nesses contextos, uma vez que pode melhorar a experiência do cliente, a eficiência operacional e a aceitação

geral da tecnologia robótica.

Portanto, este trabalho é justificado pela sua relevância em contribuir para o avanço da robótica móvel, treinando os robôs a interagir com seres humanos de maneira segura, eficaz e socialmente consciente, abrindo portas para uma ampla gama de aplicações práticas e impactando positivamente nossa relação com a tecnologia robótica.

Capítulo 2

Referencial Teórico

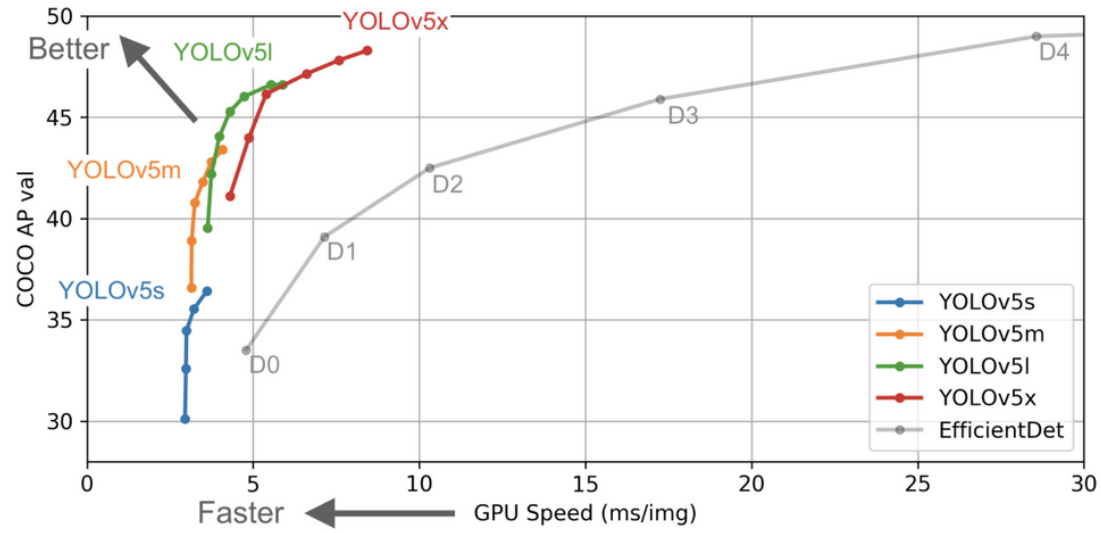
2.1 Detecção de Obstáculos

O algoritmo YOLO representa uma inovadora arquitetura de rede neural convolucional na área de visão computacional. Diferenciando-se por sua capacidade de realizar detecção de objetos em tempo real, o YOLO opera de maneira eficiente ao analisar imagens em uma única passagem, proporcionando resultados rápidos e precisos. Essa arquitetura é treinada utilizando técnicas avançadas de aprendizado profundo, o que aprimora sua capacidade de reconhecer e localizar objetos em diversas situações, destacando-se como uma ferramenta valiosa para aplicações que exigem detecção ágil e precisa de elementos visuais.

Segundo (REDMON; FARHADI, 2018), o YOLO é um sistema de detecção de objetos em tempo real de última geração, conhecido por sua extrema rapidez e precisão, sendo até quatro vezes mais veloz que seu equivalente, o Focal Loss. Também é possível treinar o algoritmo para detecções customizadas.

De acordo (SOLAWETZ, 2020), o YOLO versão 5, possui 4 versões pré-treinada disponibilizadas para os usuários. A Figura 2.1 mostra a comparação entre as quatro variantes disponibilizados. Cada uma delas possui taxas de precisão distintas e, conseguentemente, demandam de maior processamento.

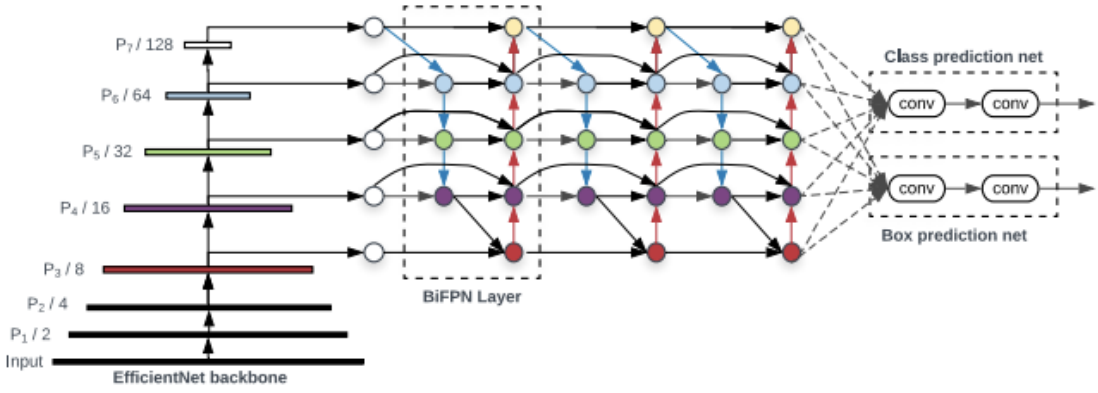
A detecção de objetos implica na extração de características a partir de imagens de entrada, chamadas imagens de treino. Essas características são posteriormente processadas por um sistema de previsão para delinear *Bounding Box* ao redor dos objetos e antecipar suas classes, (SOLAWETZ, 2020). Essas *Bounding Box*



FONTE: (JOCHER, 2020) Descrição da fonte da figura

Figura 2.1 – *Gráfico comparativo entre as versões de treinamento do YOLOv5*

é gerada utilizando uma lista de *Anchor Box Dimensions* para prever e desenha-las. A Figura 2.2 exemplifica como é a anatomia de um detector de objetos.



FONTE: (JOCHER, 2020)

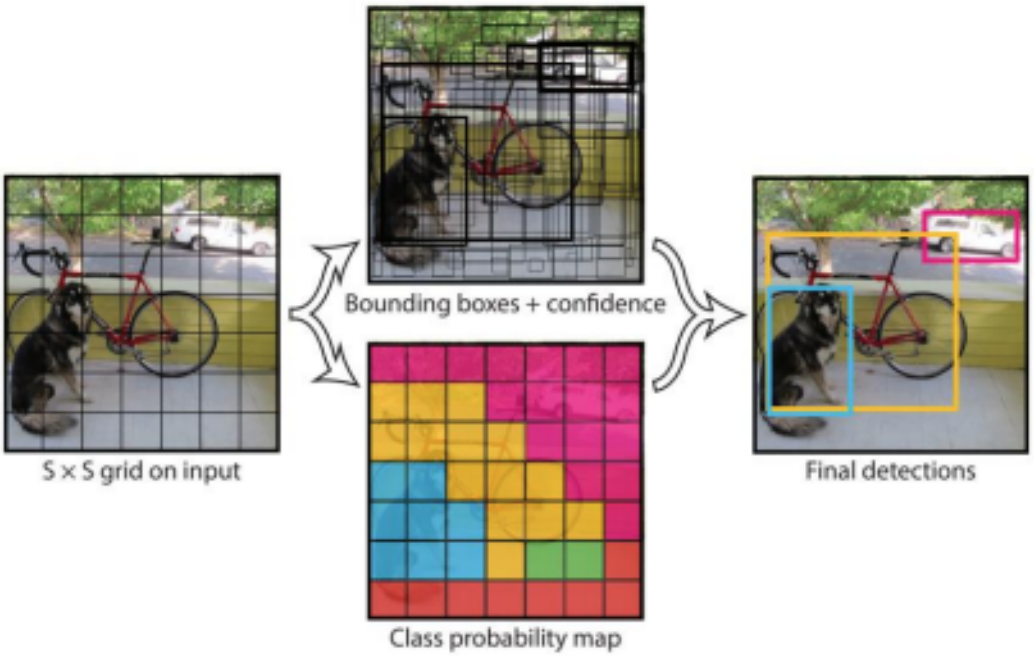
Figura 2.2 – *Anatomia de um detector de obstáculos*

Conforme (REDMON; FARHADI, 2018), O YOLO emprega uma única rede neural para analisar a imagem como um todo. Essa rede divide a imagem em regiões, realizando previsões de caixas delimitadoras e suas probabilidades para cada região demarcada. Adicionalmente, as caixas delimitadoras são refinadas com base nas probabilidades determinadas pela rede neural, utilizando uma abordagem de regressão logística para ponderação.

Segundo (**??**), regressão logística é uma técnica de análise estatística utilizada para prever um resultado binário com base em observações anteriores de um conjunto de dados. Desse modo, diversos critérios de entrada podem ser levados em

consideração para classificar dados, utilizando o histórico de resultados como suporte.

A Figura 2.3 ilustra de maneira simplificada o funcionamento do processo de regressão logística implementado na arquitetura do YOLO.



FONTE: (JOCHER, 2020)

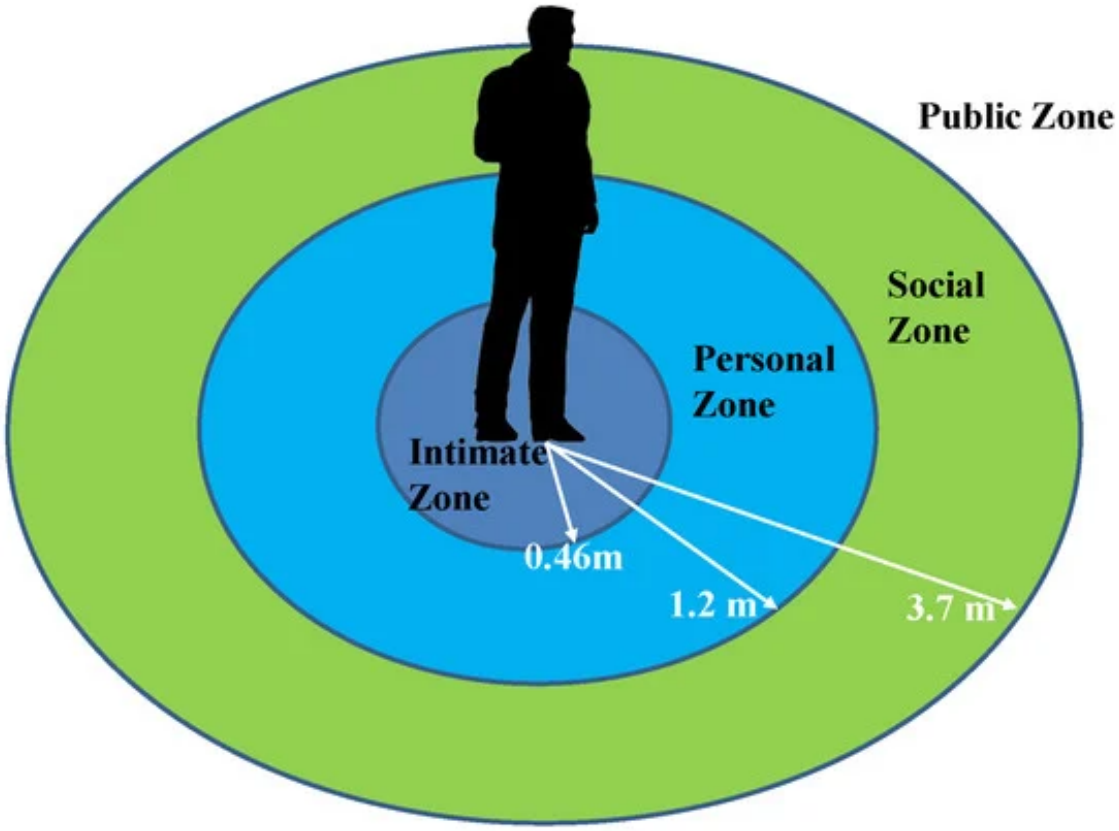
Figura 2.3 – *Funcionamento da regressão logística no YOLO*

Levando em conta esses aspectos, o YOLO apresenta várias vantagens sobre sistemas fundamentados em classificadores. Isso ocorre porque, ao analisar a imagem como um todo durante o tempo de teste, as previsões são feitas considerando todo o contexto presente. Além disso, o YOLO realiza suas previsões com uma única avaliação de rede, ao contrário de outros sistemas, tornando o processo até mil vezes mais rápido (REDMON; FARHADI, 2018).

2.2 Proxêmica Robótica

Segundo o antropólogo (HALL, 1966), o termo proxêmica é utilizado para descrever o espaço pessoal em meio social, considerando os limites individuais com base na distância entre cada ser humano. Esse estudo adota a concepção de criar zonas de conforto e espaços que devem ser intuitivamente respeitados. Em linhas gerais, observa-se que as pessoas tendem a se sentir mais à vontade onde seus limites pessoais são respeitados, enquanto o oposto ocorre quando esses limites são desconsiderados e invadidos.

A seguir, na Figura 2.4, é apresentada uma representação simplificada da percepção das zonas proxêmicas por parte de um indivíduo na sociedade. Segundo (HALL, 1966), essa zona é dividade em quatro áreas diferentes e cada uma representa um nível de intimidade entre o usuário e o próximo.



FONTE: (HALL, 1966)

Figura 2.4 – *Zonas proxêmicas*

Na perspectiva da robótica móvel, a interação homem-máquina demanda o respeito pelas zonas individuais de cada pessoa (HALL, 1966). Para integrar adequadamente esses novos dispositivos no convívio humano e manter um equilíbrio, que é intuitivo para os seres humanos, mas não para os robôs, existem várias maneiras de determinar as zonas proxêmicas das pessoas em um ambiente. (AGHAEI et al., 2021) utilizaram visão computacional e a técnica de *homografia* para estimar esses espaços teóricos.

Considerando a perspectiva humana diante da presença de robôs em ambientes sociais, (LEHMANN; ROJIK; HOFFMANN, 2020) conduziram um estudo para compreender como as pessoas percebem essas dimensões. Segundo eles, máquinas de menor porte são mais aceitas, entretanto, a maioria das pessoas ainda mantém uma distância padrão, independente do tamanho dos robôs utilizados.

Pensando no nível de confiança que os robôs passam para as pessoas e também levando em consideração de que o projeto será implementado em uma cadeira de rodas autônoma, desenvolvemos um modelo onde os diferentes tipos de obstáculos influenciam na determinação da zona de segurança gerada ao entorno da cadeira de rodas, trazendo uma maior segurança ao usuário. Essa nova classificação é dividida em quatro classes segundo as características físicas dos obstáculos, conforme será detalhado na Seção 3.2.

2.3 Navegação Social

A navegação é facilmente compreensível, pois opera lendo informações de odometria e do fluxo dos sensores para gerar comandos de velocidade para uma base móvel.

Diante do que foi mencionado, a navegação autônoma fundamenta-se na criação de um sistema de orientação para robôs em ambientes familiares, fazendo uso de um *Voxel Grid* e modelagem de espaços desconhecidos. Segundo (MARDER-EPPSTEIN et al., 2010), essa abordagem visa viabilizar um comportamento seguro em ambientes complexos.

Assim, a representação desse espaço contribui para uma abordagem que estabelece a velocidade máxima segura de operação para o robô em uso. O sistema opera detectando e evitando a maioria dos obstáculos no ambiente. Isso se deve ao fato de que, uma vez que a primeira rota é estabelecida, é necessário que o robô seja capaz de criar e executar novas rotas para evitar colisões com os obstáculos presentes no cenário, conforme indicado por (MARDER-EPPSTEIN et al., 2010).

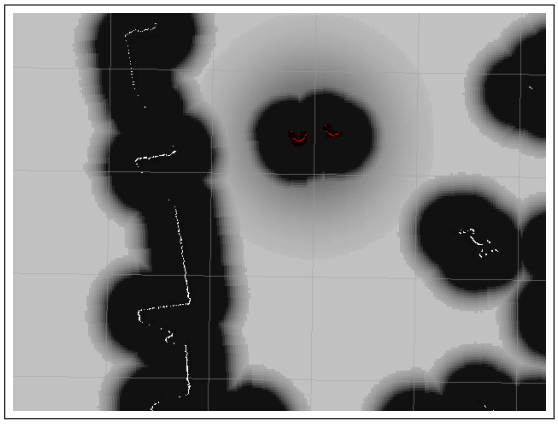
À medida que os robôs demandam crescentemente comportamentos sociais, torna-se necessário desenvolver modos de navegação adaptados a essas restrições. Isso ocorre porque diversas métricas precisam ser consideradas na definição das rotas de movimento, nem sempre priorizando a distância mais curta como a situação ideal.

Levando isso em consideração, a navegação incorpora diversos plug-ins nos quais geram restrições para a navegação social, como, por exemplo, a distância proxêmica. Segundo (LU, 2014), existem duas camadas de navegação social baseadas na distribuição gaussiana: a camada proxêmica e a camada de passagem.

A Figura 2.5 mostra uma pessoa (pontos vermelhos), parada, sendo detec-

tada pelo robô. Ao seu redor é possível ver a zona de segurança, em preto. Essa área é onde o robô não pode passar.

Já na Figura 2.6, quando a pessoa entra em movimento, uma nova área é mostrada em cinza-escuro. Neste caso, o robô não poderá trafegar por aquele espaço, pois é ali o provável percurso da pessoa.



FONTE: (LU, 2014) Descrição da fonte da Figura 1

Figura 2.5 – *Comportametno da gaussiana com a pessoa parada*

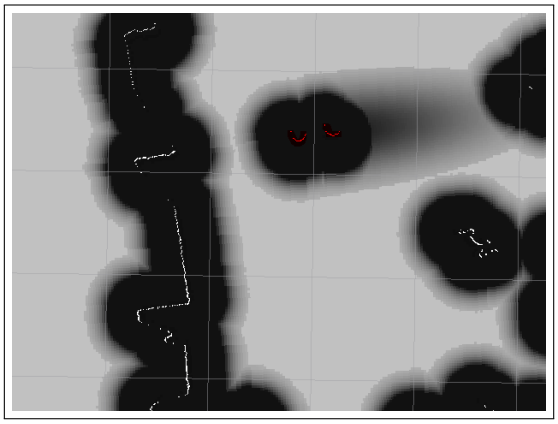


Figura 2.6 – *Comportametno da gaussiana com a pessoa em movimento*

FONTE: (LU, 2014) Descrição da fonte da Figura 2

Com isso, fica evidente como o plug-in da distância proxêmica interfere diretamente no comportamento da gaussiana na navegação social.

2.3.1 Simulador de Ambiente Virtual

Segundo o site (GAZEBOSIM, 2002), o GazeboSim é um simulador de robótica tridimensional de código aberto que permite a simulação de ambientes e sistemas robóticos. Ele oferece uma plataforma para simulação realista de robôs para vários ambientes. Também inclui recursos como simulação de física, suporte a sensores e atuadores, modelagem de ambientes, entre outros.

Esse simulador possui uma capacidade de simulação valiosa para desenvolvedores e pesquisadores que desejam testar e validar algoritmos, controladores e designs de robôs em um ambiente virtual antes de implementá-los no mundo real.

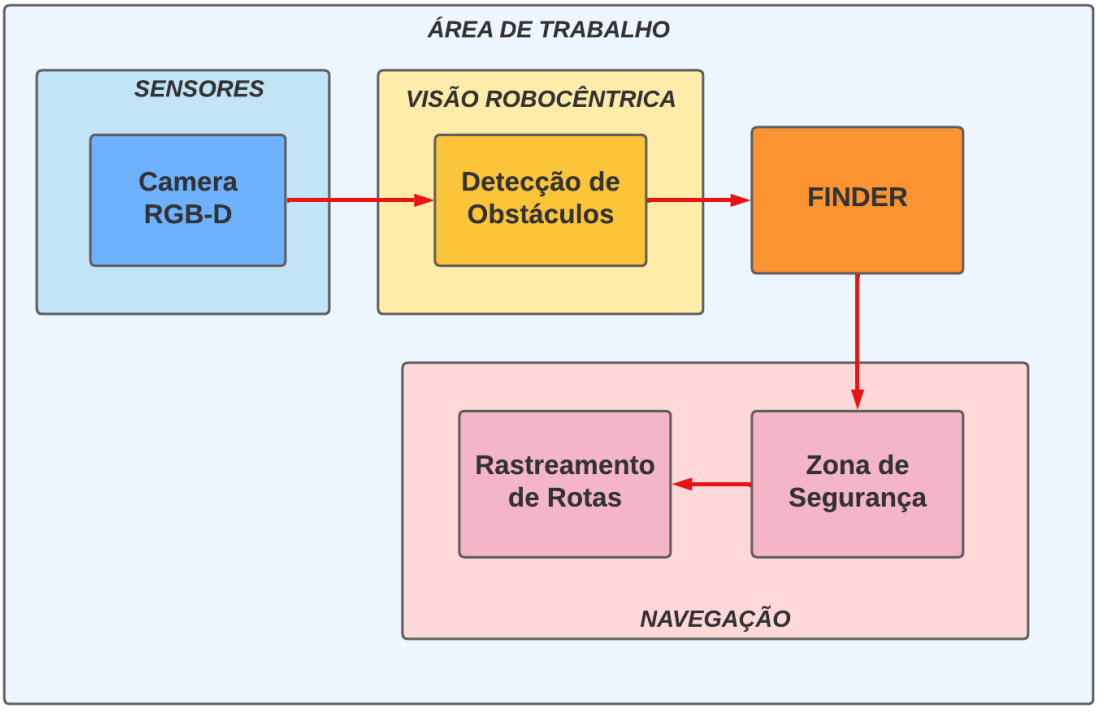
Nele também é possível integrar sua funcionalidade com o ROS (Robot Operating System), um framework contendo um conjunto de bibliotecas e ferramentas, onde é possível desenvolver diversas aplicações para robótica, conforme demostra o próprio fabricante (WIKI, 2007).

Capítulo 3

Metodologia

A metodologia deste trabalho envolve a criação de um conjunto de funções utilizando a linguagem de programação Python e fazendo uso das bibliotecas do ROS (Robot Operating System).

A Figura 3.1 ilustra o esquema que detalha a organização e o funcionamento deste trabalho.



FONTE: De autoria própria

Figura 3.1 – *Mapa esquemático do processo para gerar zonas proxêmicas com base nas classes de objetos detectadas.*

A primeira etapa, caixa em azul, envolve a aquisição de dados por meio de tópicos, um mecanismo utilizado pelo ROS para a comunicação entre os diversos serviços, já publicados pelo sensor disponível no robô. Em seguida, a visão robocêntrica, através do YOLO, identifica os obstáculos e encaminha as informações para o algoritmo FINDER, onde esses dados são processados e as zonas proxêmicas são modeladas. Já na navegação, são traçados novas trajetórias respeitando as zonas proxêmicas geradas pelo FINDER.

Os testes foram conduzidos em um laptop equipado com um processador Intel Core i7-10750H, placa de vídeo NVIDIA GeForce GTX 1650 4GB, 8GB de RAM e SSD de 512GB. O sistema operacional utilizado foi o Linux Ubuntu 20.04, e o ROS Noetic, versão compatível com o Ubuntu. Para os testes virtuais, empregou-se o software Gazebo para criar o ambiente virtual.

Vale ressaltar que o FINDER não é responsável pela navegação autônoma, ele é uma ferramenta para auxiliar na navegação social, por isso, a navegação do robô já deve está funcional, uma vez que utiliza a posição e orientação fornecidas pela navegação autônoma.

3.1 Workspace e Sensoriamento

Para a execução deste trabalho, contamos com o uso de uma cadeira de rodas motorizada adaptada pelo GIPAR para realizar a navegação de maneira autônoma, conforme ilustrado na Figura 3.2. Os testes foram conduzidos tanto em ambiente virtual, usando o Gazebo, quanto em ambiente real, nas instalações do Bloco G do IFBA Campus Vitória da Conquista.

Para a aquisição de dados, utilizamos a câmera estéreo ZED modelo 2i, uma câmera RGB-D, ou seja, uma câmera colorida com função de profundidade. Possui um amplo campo de visão com uma lente grande angular de até 120º. A resolução máxima é de 1080p com uma taxa de quadros máxima de 30Hz. A câmera de profundidade atinge uma taxa de até 100Hz, cobrindo uma faixa de detecção que varia de 0,3m a 20m. Além disso, a câmera está equipada com acelerômetro, giroscópio, barômetro e magnetômetro.

FONTE: De autoria própria

Figura 3.2 – *Cadeira motorizada modificada para navegação autônoma*

3.2 Detecção de Obstáculos

A visão computacional, uma parte fundamental da inteligência artificial, tem como objetivo treinar máquinas a extrair informações do mundo real por meio do processamento de imagens. Sua aplicação é vasta, abrangendo setores como indústria e medicina, sendo utilizada, por exemplo, na prevenção de acidentes industriais, vigilância e detecção de doenças em imagens médicas.

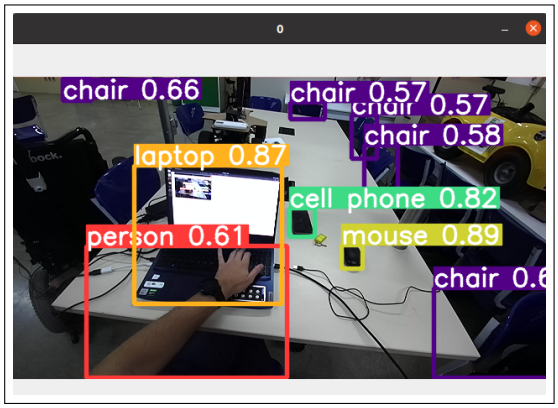
Na robótica, em especial em veículos autônomos, a detecção de objetos é um desafio crucial, permitindo a identificação de elementos como pedestres e veícu-

FONTE: De autoria própria

Figura 3.3 – *Câmera ZED versão 2i*

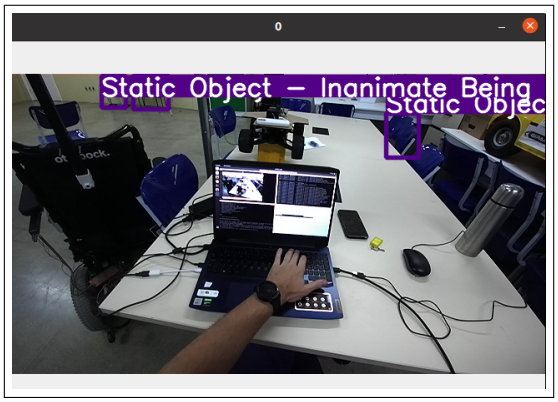
los. Nesse contexto, empregamos uma câmera ZED para a captura de imagens e o YOLO para a identificação de obstáculos.

O YOLO, por padrão, já é treinado para reconhecer 80 classes diferentes, mas muitas delas são insignificantes para este trabalho, como cadernos e lápis, representando um consumo desnecessário de processamento. Diante disso, propomos um novo treinamento dividido em quatro categorias de obstáculos, agrupadas por semelhanças entre seres vivos ou não vivos e estáticos ou dinâmicos. Nesse novo treinamento, selecionamos apenas objetos que verdadeiramente representam obstáculos para a cadeira de rodas. As Figuras 3.4 e 3.5 mostram a detecção de objetos utilizando o treinamento original e o treinamento customizado, respectivamente.



FONTE: De autoria própria

Figura 3.4 – *Detecção de objetos com treinamento original*



FONTE: De autoria própria

Figura 3.5 – *Detecção de objetos com treinamento customizado*

A organização dessas classes e seus respectivos exemplos está apresentada na Tabela 3.1.

Tabela 3.1 – *Nova classificação e seus respectivos exemplos*

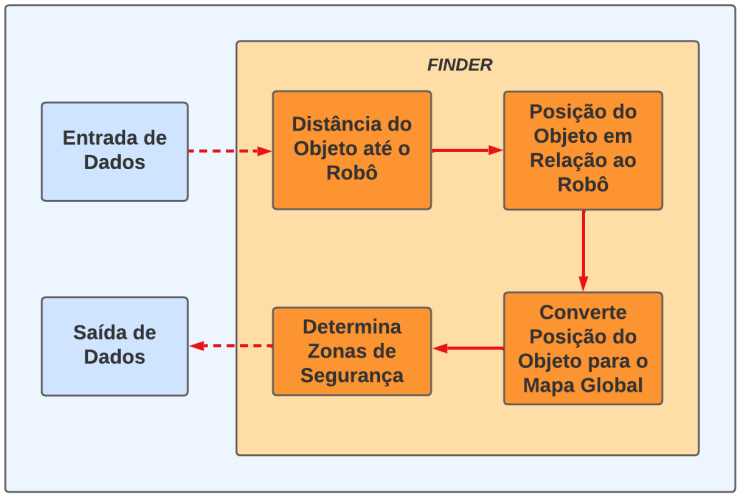
Classes	Exemplos
Dynamic Object - Living Being	Pessoa, animais
Dynamic Object - Inanimate Being	Veículos, robôs
Static Object - Living Being	Plantas
Static Object - Inanimate Being	Cadeira, mesa, armário

FONTE: De autoria própria

3.3 Finder

O Finder é um algoritmo desenvolvido neste trabalho para localização de obstáculos e geração de suas respectivas zonas de segurança. Com isso, as zonas de segurança, que anteriormente eram padronizadas para todo o mapa, agora podem ter áreas específicas para cada obstáculo detectado.

O algoritmo é dividido em quatro partes, conforme apresentado na Figura 3.6.



FONTE: De autoria própria

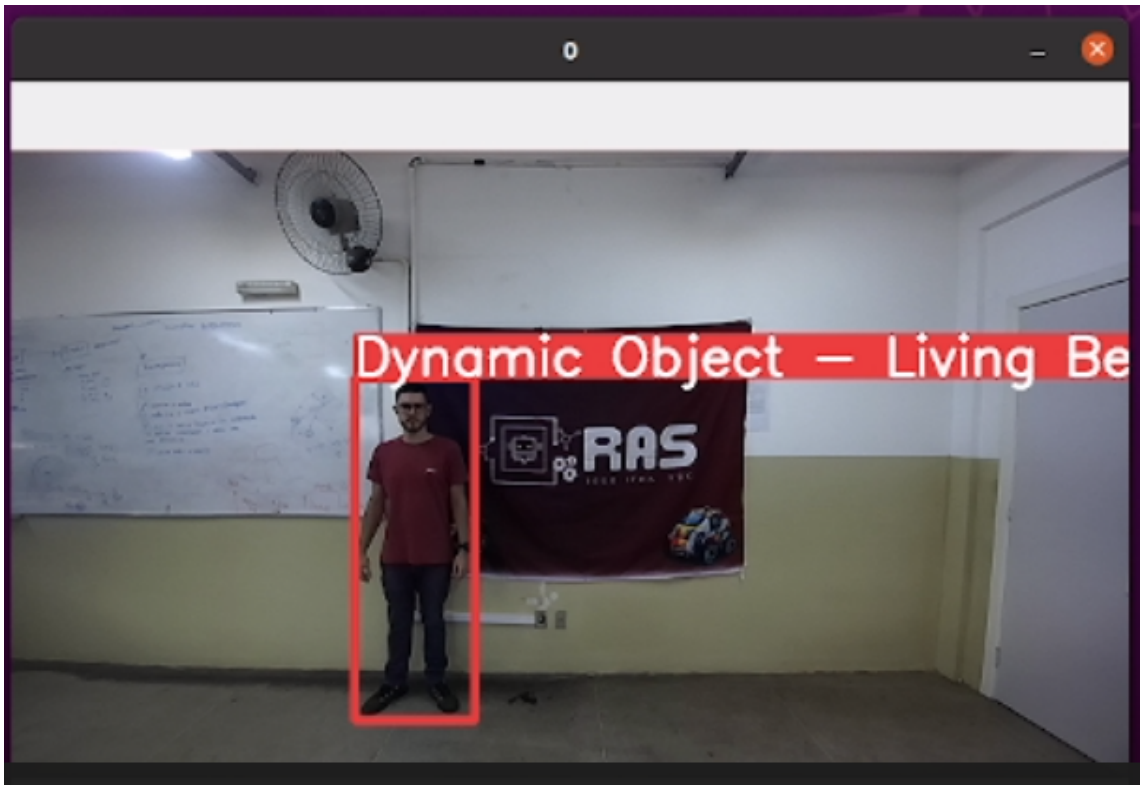
Figura 3.6 – *Mapa esquemático do funcionamento do algoritmo FINDER*

3.3.1 Entrada de Dados

Para o correto funcionamento do Finder, é necessário obter alguns dados da câmera de profundidade, do YOLO e a posição do robô no mapa.

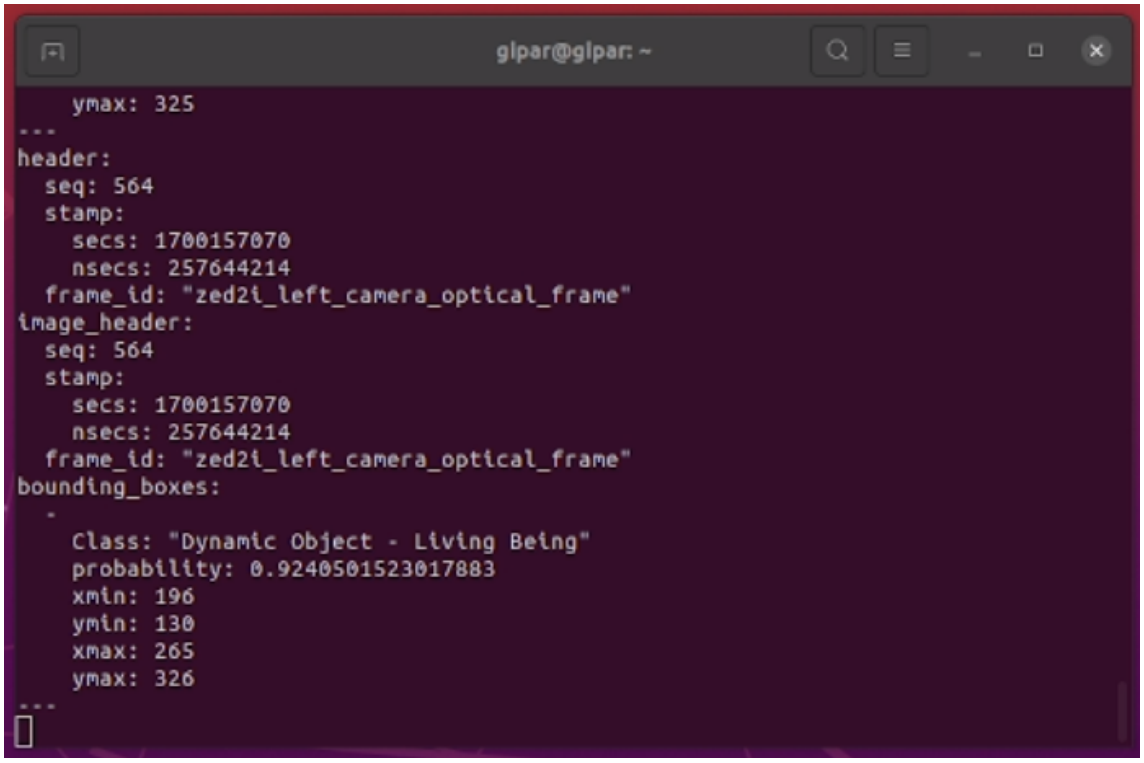
A câmera de profundidade opera medindo a distância entre a câmera e os objetos, gerando assim uma matriz de pixels. Essa matriz permite obter a distância de um pixel específico ao conhecer sua posição. As câmeras ZED publicam essas informações em um tópico.

O YOLO, após detectar os objetos na cena (conforme ilustrado na Figura 3.7), gera um tópico contendo informações sobre a classe e a bounding box de cada objeto, como exemplificado na Figura 3.8



FONTE: De autoria própria

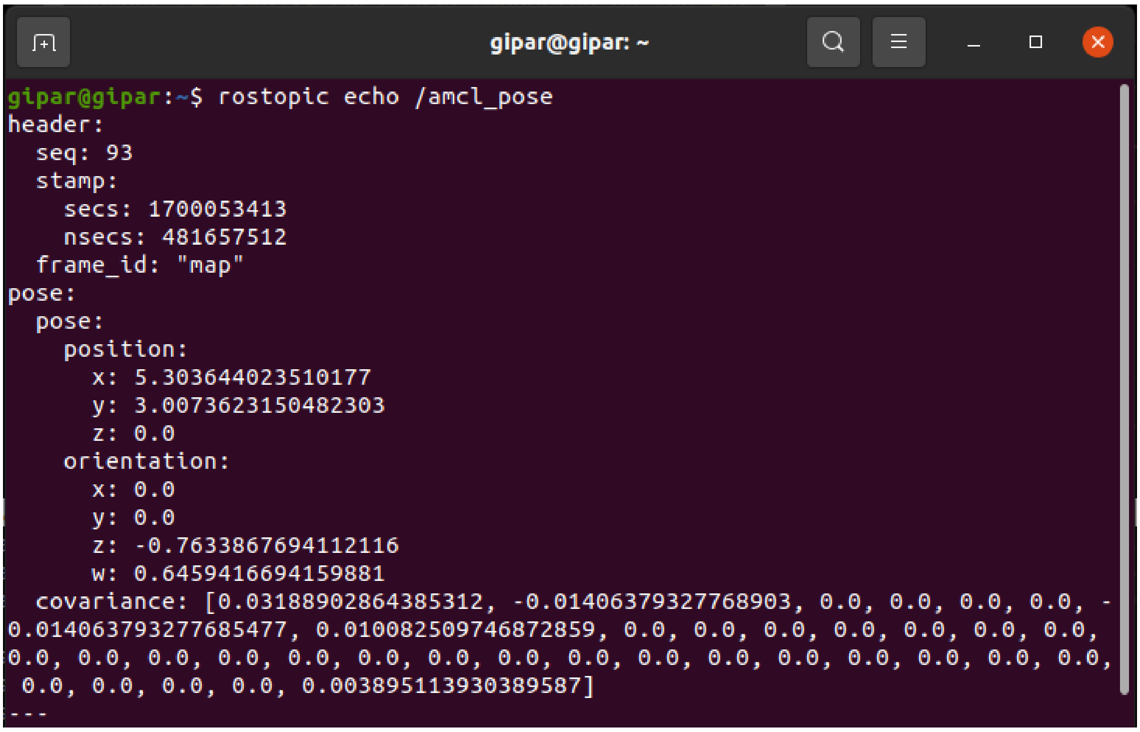
Figura 3.7 – *Obstáculo sendo detectado pelo YOLO*



FONTE: De autoria própria

Figura 3.8 – *Saída do tópico /yolov5/detections com as informações da bouding box do objeto detectado*

Por último, são necessárias as informações de posição e orientação do robô em relação ao mapa, conforme exemplificado na Figura 3.9. Essas informações são geradas pela navegação autônoma e publicados em um tópico, chamado /amcl_pose.



```
gipar@gipar: ~
gipar@gipar:~$ rostopic echo /amcl_pose
header:
  seq: 93
  stamp:
    secs: 1700053413
    nsecs: 481657512
  frame_id: "map"
pose:
  pose:
    position:
      x: 5.303644023510177
      y: 3.0073623150482303
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: -0.7633867694112116
      w: 0.6459416694159881
  covariance: [0.03188902864385312, -0.014063793277768903, 0.0, 0.0, 0.0, 0.0, -0.014063793277685477, 0.010082509746872859, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0038951139303895871]
---
```

FONTE: De autoria própria

Figura 3.9 – *Informações de posição e orientação do robô*

3.3.2 Distância do Objeto para o Robô

É fundamental que o robô avalie com precisão a distância entre si e os objetos detectados para determinar sua posição em relação às zonas de segurança estabelecidas. Neste trabalho, para alcançar esse objetivo, adotou-se uma abordagem que envolve a estimativa do ponto central da caixa delimitadora gerada pelo algoritmo de detecção de objetos. Essa estimativa é realizada com base nas coordenadas Y_{max}, X_{min}, Y_{max}, e Y_{min} fornecidas pelo próprio YOLO. As equações 3.1 e 3.2 são, então, aplicadas para calcular o ponto médio da caixa delimitadora a partir dessas coordenadas.

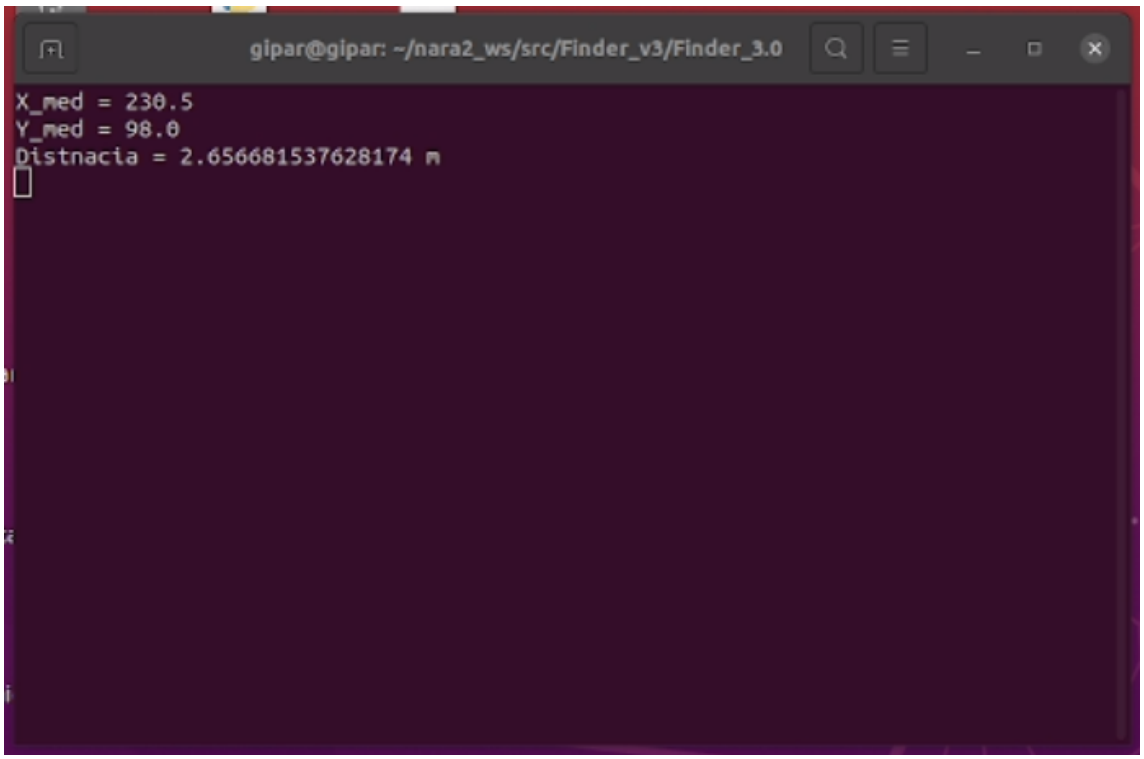
$$X_{med} = \frac{X_{max} - X_{min}}{2} + X_m in \tag{3.1}$$

$$Y_{med} = \frac{Y_{max} - Y_{min}}{2} + Y_{min} \tag{3.2}$$

Sendo que:

X_{med} ⇒ Ponto médio da Bouding Box no eixo X;
Y_{max} ⇒ Ponto maxim da Bouding Box no eixo X;
X_{min} ⇒ Ponto mínimo da Bouding Box no eixo X;
Y_{med} ⇒ Ponto médio da Bouding Box no eixo Y;
Y_{max} ⇒ Ponto maximo da Bouding Box no eixo Y;
Y_{min} ⇒ Ponto mínimo da Bouding Box no eixo Y;

Com as coordenadas do ponto central, conforme ilustrado na Figura 3.10, buscamos nos dados da câmera de profundidade a informação referente a essas coordenadas (ver linha 91, Apêndice B).



FONTE: De autoria própria

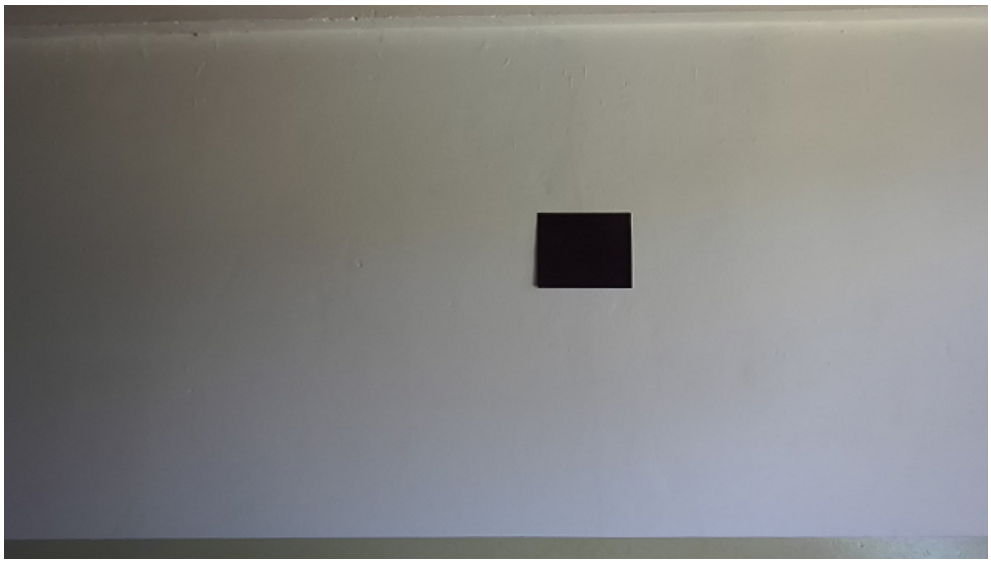
Figura 3.10 – *Saída do tópico /finder com as informações do ponto médio da bouding box*

3.3.3 Posição do Objeto em Relação ao Robô

Com as informações da distância do objeto em relação ao robô, é necessário determinar a posição X e Y do objeto no plano cartesiano. Para realizar esse cálculo, é essencial calcular a distância focal, que representa a distância entre o sensor da câmera e o centro óptico da lente.

3.3.3.1 Distância Focal

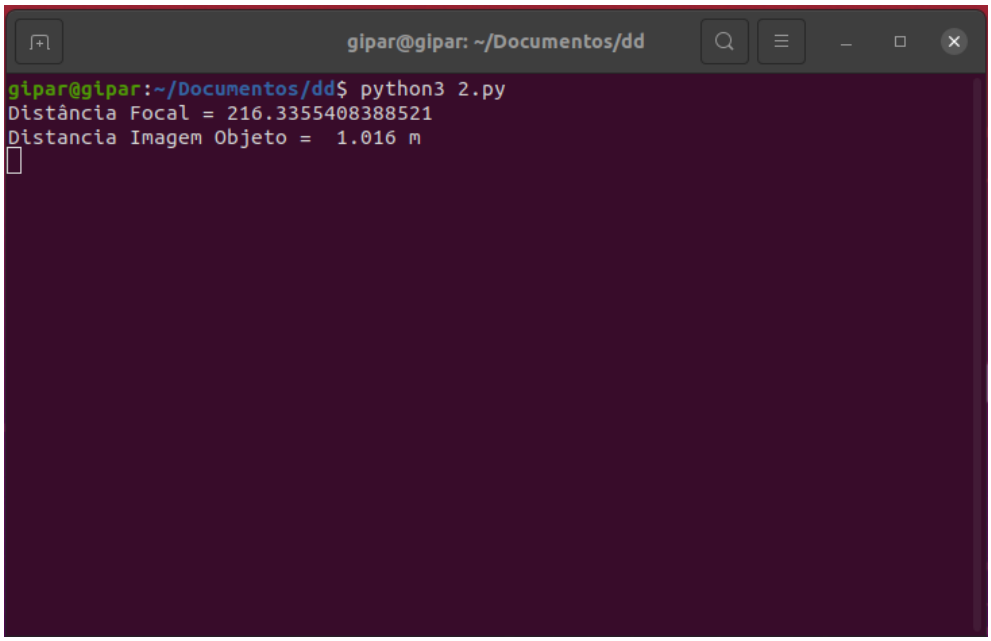
Para determinar o valor da distância focal, utilizamos um objeto, neste caso, uma folha com dimensões de 23cm x 19cm, fixado na parede, conforme ilustrado na Figura 3.11. Com a câmera posicionada a 100cm do objeto, capturamos uma imagem.



FONTE: De autoria própria

Figura 3.11 – *Folha de papel colada na parede para cálculo da distância focal*

Em seguida, calculamos o valor da distância focal (Apêndice A.1). É importante observar que algumas modificações são necessárias no código. As linhas 28 e 31 devem ser ajustadas conforme os valores mencionados no parágrafo anterior. Além disso, é necessário informar o nome do arquivo de imagem. A Figura 3.12 apresenta a saída do terminal com os valores da Distância Focal e a distância do objeto até a câmera.



FONTE: De autoria própria

Figura 3.12 – *Valores da Distância Focal e a distância da câmera até o objeto*

O nosso código (Apêndice A.1) utiliza a Fórmula 3.3 para encontrar o valor da distância focal.

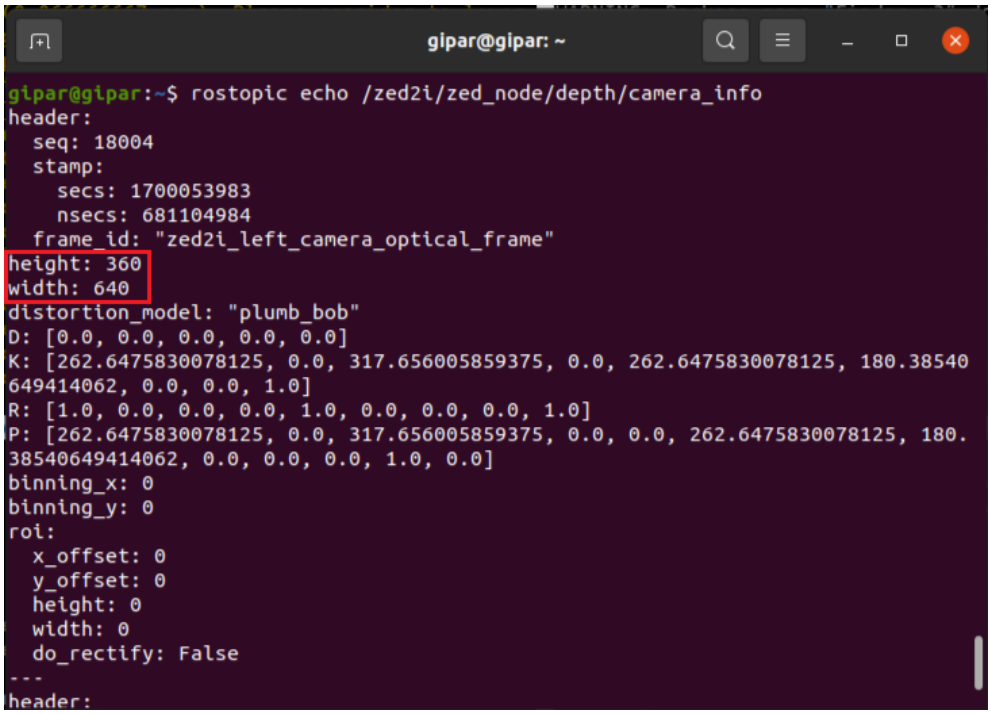
$$df = \frac{p * d}{w} \tag{3.3}$$

Sendo que:

df ⇒ Distância focal;
p ⇒ Comprimento do objeto na imagem, em pixels;
d ⇒ Distância entre a câmera e o objeto, em metros;
w ⇒ Comprimento do objeto, em metros;

3.3.3.2 Posição X e Y do Objeto em Relação ao Robô

Após calcular a distância focal, agora podemos determinar a posição X e Y do objeto em relação à câmera. Essas coordenadas podem ser calculadas pelas Fórmulas 3.4 e 3.5. Para isso, utilizaremos a distância (dist) encontrada com base nos dados obtidos pela câmera de profundidade, além dos pontos X_{med} e Y_{med} da *bounding box* e da resolução da câmera (640 x 480), obtida através do tópico de informações, conforme mostrado na Figura 3.13.



FONTE: De autoria própria

Figura 3.13 – *Informações da resolução da câmera destacada em vermelho*

No caso da resolução da câmera, é importante considerar que o pixel central da imagem é o ponto de referência do plano cartesiano. Em outras palavras, o pixel (320,240) corresponde ao ponto (0,0) do plano cartesiano. Portanto, as variáveis cx e cy assumem os valores de 320 e 240, respectivamente.

$$X_{o.r} = dist * \frac{x_{med} - cx}{df} \tag{3.4}$$

$$Y_{o.r} = dist * \frac{y_{med} - cy}{df} \tag{3.5}$$

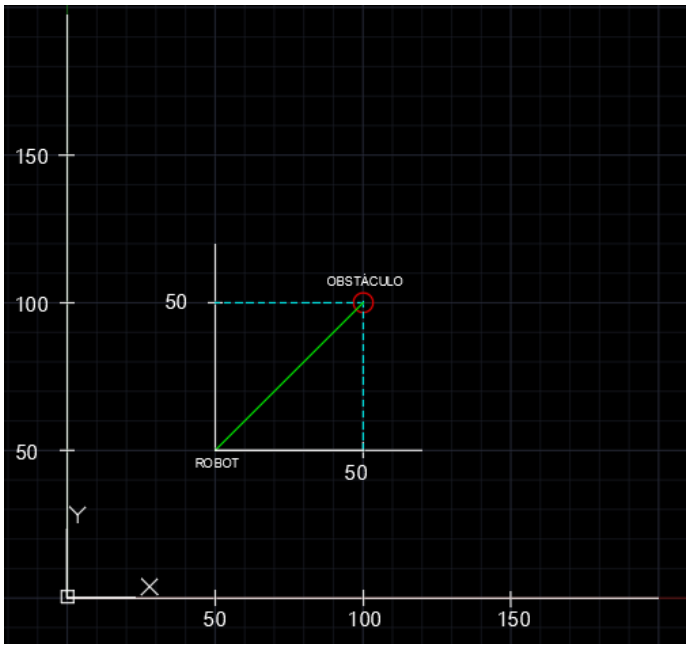
Sendo que:

$X_{o.r}$ ⇒ Posição do objeto em relação ao eixo X;
dist ⇒ Distancia encontrada pela câmera de profundidade;
X_{med} ⇒ Ponto médio da Bouding Box no eixo X, em pixels;
cx ⇒ Ponto central da imagem, neste caso é 320 pixels;
df ⇒ Distância focal;
$Y_{o.r}$ ⇒ Posição do objeto em relação ao eixo Y;
Y_{med} ⇒ Ponto médio da Bouding Box no eixo Y, em pixels;
cy ⇒ Ponto central da imagem, neste caso é 240 pixels;

3.3.4 Conversão da Posição do Objeto para o Mapa Global

As coordenadas obtidas na Seção 3.3.3.2 correspondem à posição do objeto em relação ao robô. Entretanto, é crucial encontrar as coordenadas do objeto em relação ao mapa, uma vez que o centro do robô nem sempre coincide com o ponto de referência do mapa.

Na Figura 3.14, é apresentado um exemplo em que os eixos do plano cartesiano do robô estão alinhados com os do mapa. Nesse caso, é suficiente somar as coordenadas do obstáculo em relação ao robô, $X_{o.r}$ e $Y_{o.r}$, com as coordenadas do robô no mapa, X_r e Y_r.



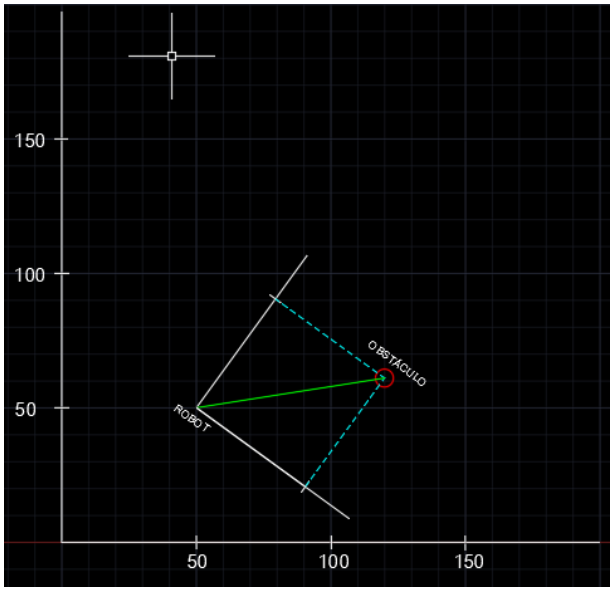
FONTE: De autoria própria

Figura 3.14 – *Plano cartesiano do robô em paralelo com o plano cartesiano do mapa*

No entanto, devido à mobilidade do robô, é pouco provável que seu plano cartesiano esteja perfeitamente alinhado com o plano do mapa. Portanto, é necessário recorrer a fórmulas geométricas para converter as coordenadas $X_{o.r}$ e $Y_{o.r}$ para as coordenadas do mapa.

A Figura 3.15 ilustra um segundo exemplo em que o plano cartesiano do robô está desalinhado em relação ao do mapa. Nesse caso, torna-se imperativo alinhar

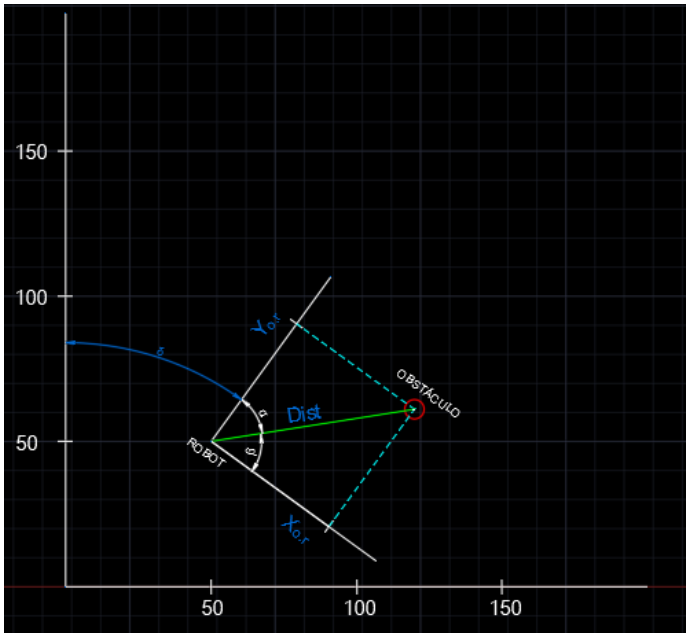
esses eixos para determinar as coordenadas do robô no mapa global, $X_{o.m}$ e $Y_{o.m}$.



FONTE: De autoria própria

Figura 3.15 – *Plano cartesiano do robô desalinhado com o plano cartesiano do mapa*

Esse alinhamento é viável devido ao conhecimento prévio das informações de posição e orientação fornecidas pelo robô, juntamente com os dados da distância (dist) e da posição $X_{o.r}$ e $Y_{o.r}$, como detalhado nas Seções 3.3.2 e 3.3.3.2, respectivamente. Essas informações estão representadas em azul na Figura 3.16.



FONTE: De autoria própria

Figura 3.16 – *Calculando os ângulos α e β utilizando a fórmula do triângulo retângulo*

Para determinar os valores de α e β, empregamos as relações trigonométricas do triângulo retângulo, conforme expressas nas Fórmulas 3.6 e 3.7.

$$cos(\alpha) = \frac{X_{o.r}}{dist} \tag{3.6}$$

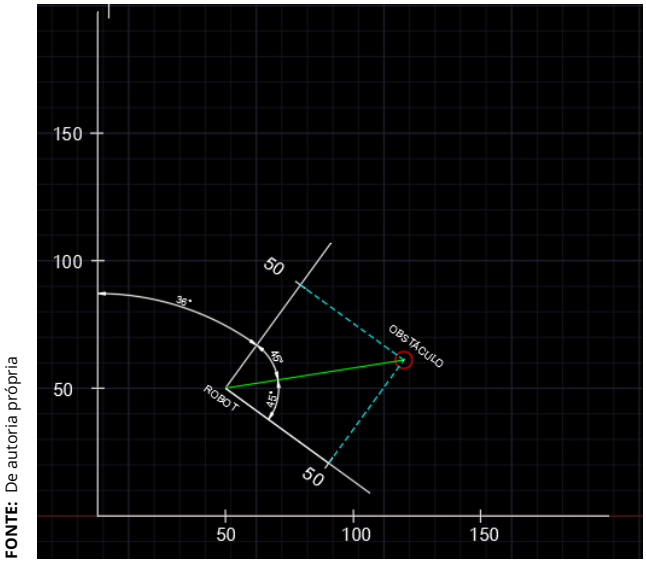
$$sen(\alpha) = \frac{Y_{o.r}}{dist} \tag{3.7}$$

Sendo que:

$X_{o.r}$ ⇒ Posição do objeto em relação ao eixo X;
$Y_{o.r}$ ⇒ Posição do objeto em relação ao eixo Y;
dist ⇒ Distância do objeto até o robô;

Na Figura 3.17, já dispomos das informações essenciais para o alinhamento do robô com o plano cartesiano do mapa. Nesta etapa, a orientação do robô é cru-

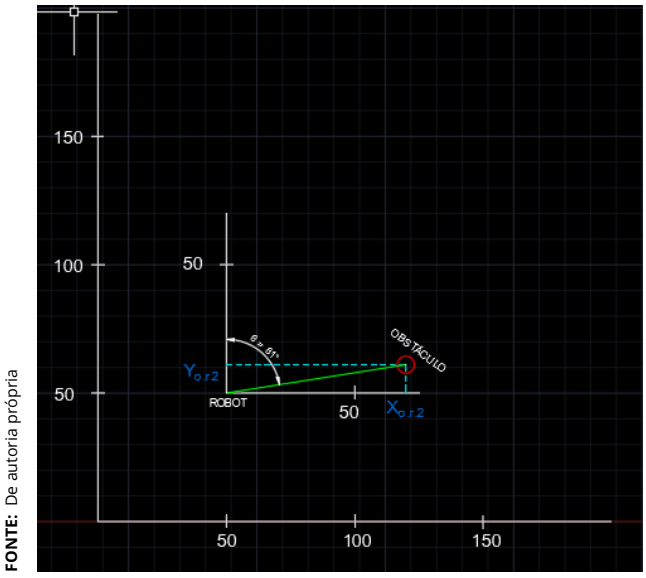
cial, pois indica a quantidade necessária de rotação para alcançar o alinhamento desejado.



FONTE: De autoria própria

Figura 3.17 – *Orientação do robô em relação ao mapa (γ) e a posição do objeto em relação ao robô*

Vale destacar que a distância (dist) permanece inalterada; a única modificação ocorreu nas variáveis $X_{o.r}$ e $Y_{o.r}$, as quais foram atualizadas para as novas coordenadas $X_{o.r.2}$ e $Y_{o.r.2}$, conforme ilustrado na Figura 3.18.



FONTE: De autoria própria

Figura 3.18 – *Alinhamento entre os planos cartesianos do robô e do mapa. Nota-se que as coordenadas $X_{o.r}$ e $Y_{o.r}$ modificaram*

Os novos valores, $X_{o.r.2}$ e $Y_{o.r.2}$, podem ser determinados através das Fórmulas 3.8 e 3.9.

$$Y_{o.r.2} = sen(\gamma) * dist \tag{3.8}$$

$$X_{o.r.2} = cos(\gamma) * dist \tag{3.9}$$

Sendo que:

$X_{o.r.2}$ ⇒ Nova posição do objeto em relação ao robô no eixo X;
$Y_{o.r.2}$ ⇒ Nova posição do objeto em relação ao robô no eixo Y;
dist ⇒ Distância do objeto até o robô;

Observa-se, ainda, a geração de um novo ângulo, resultante da soma do ângulo α com o ângulo de orientação do robô, γ, conforme expresso na Fórmula 3.10.

$$\theta = \alpha + \gamma \tag{3.10}$$

A seguir, realizamos a soma das coordenadas $X_{o.r.2}$ e $Y_{o.r.2}$ com as coordenadas X_r e Y_r, obtendo assim a posição real do objeto em relação ao mapa, conforme demonstrado nas Fórmulas 3.11 e 3.12.

$$X_{o.m} = X_{o.r.2} + X_r \tag{3.11}$$

$$Y_{o.m} = Y_{o.r.2} + Y_r \tag{3.12}$$

Sendo que:

$X_{o.m}$ ⇒ Posição do objeto em relação ao mapa no eixo X;
$Y_{o.m}$ ⇒ Posição do objeto em relação ao mapa no eixo Y;
$X_{o.r.2}$ ⇒ Nova posição do objeto em relação ao robô no eixo X;
$Y_{o.r.2}$ ⇒ Nova posição do objeto em relação ao robô no eixo Y;
X_r ⇒ Posição do objeto em relação ao eixo X;
Y_r ⇒ Posição do objeto em relação ao eixo Y;

Em seguida, chegamos às Fórmulas 3.13 e 3.14, as quais apresentam, de maneira simplificada, todos esses cálculos.

$$X_{o.m} = \{sen[\gamma + sen^{-1}(\frac{Y_{o.r}}{dist})] * dist\} + X_r \tag{3.13}$$

$$Y_{o.m} = \{sen[\gamma + sen^{-1}(\frac{Y_{o.r}}{dist})] * dist\} + Y_r \tag{3.14}$$

3.3.5 Geração das Zonas de Segurança

Após identificar as coordenadas $X_{o.m}$ e $Y_{o.m}$ e a classe do objeto por meio do YOLO, procedemos à representação no mapa da zona de segurança correspondente a essa classe, utilizando o algoritmo Finder (conforme Apêndice B) em conjunto com um Plug-in de geração de áreas Gaussianas (ver Anexo I). Esse processo nos permitiu gerar áreas de segurança específicas para cada classe identificada.

Após a identificação das coordenadas $X_{o.m}$ e $Y_{o.m}$ e a classe do objeto por meio do YOLO, o próximo passo consiste na representação no mapa da zona de segurança correspondente a essa classe. Isso é realizado através da aplicação do nosso algoritmo Finder, conforme descrito no Apêndice B, em conjunto com um Plug-in de geração de áreas Gaussianas, conforme detalhado no Anexo I.

O algoritmo Finder é essencial para mapear as coordenadas identificadas e determinar a área de segurança associada a cada classe de objeto. Ele utiliza informações contextuais, como a topologia do ambiente e a disposição dos objetos, para gerar uma representação coerente das zonas de segurança. Este processo é crucial para garantir que as áreas de segurança geradas sejam adaptadas às características específicas de cada classe identificada.

Além disso, a utilização de um Plug-in de geração de áreas Gaussianas aprimora a precisão na definição das zonas de segurança. As áreas Gaussianas são modeladas de acordo com a distribuição de probabilidade dos dados, permitindo uma representação mais suave e contínua das áreas seguras no ambiente. Isso contribui para evitar resultados abruptos e melhora a transição entre diferentes classes de objetos, proporcionando uma representação mais natural e adaptável às variações do ambiente.

O processo descrito não apenas visa gerar áreas de segurança específicas

para cada classe identificada, mas também busca otimizar a eficácia do sistema em garantir a segurança do usuário. A abordagem conjunta do algoritmo Finder e do Plug-in de áreas Gaussianas é fundamental para criar um mapa de zonas seguras que seja preciso, confiável e capaz de se adaptar dinamicamente a diferentes cenários e contextos. Isso contribui significativamente para a aplicação bem-sucedida de sistemas autônomos em ambientes diversos, proporcionando uma maior segurança e confiança no seu funcionamento.

Capítulo 4

Resultados

Nesta seção, serão apresentados e discutidos os resultados do projeto, os quais estão intrinsicamente ligados à capacidade do robô de perceber o ambiente. Isso envolve a identificação e classificação dos obstáculos ao seu redor, suas respectivas zonas de segurança, bem como a interação com a navegação autônoma, diretamente relacionada à fase de percepção. A navegação autônoma é responsável por deslocar o robô de um ponto inicial até um ponto final, seguindo o caminho mais lógico e evitando ultrapassar os limites das zonas de segurança.

Ao considerar essa subdivisão do trabalho, os resultados são divididos em dois tópicos. O primeiro trata da percepção do ambiente, abordando a detecção e classificação de obstáculos, assim como a determinação das zonas próximas aos objetos identificados. O segundo tópico refere-se à navegação autônoma, abrangendo o estabelecimento da rota mais curta, a prevenção de colisões com objetos e o respeito às áreas próximas estabelecidas.

4.1 Percepção

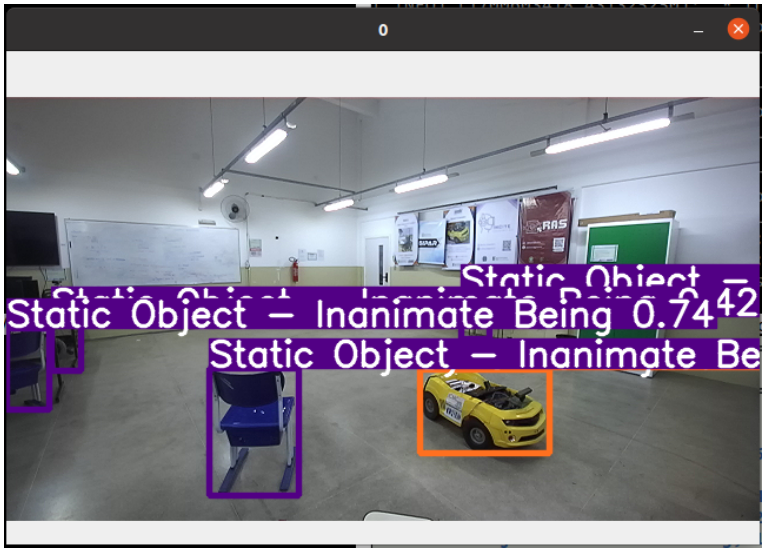
O projeto visa treinar o robô a navegar de forma autônoma e segura em um ambiente dinâmico. Isso é alcançado através da análise dos dados de percepção do ambiente, permitindo ao robô tomar decisões informadas em tempo real. Em particular, o foco recai sobre a capacidade do robô de se deslocar entre dois pontos, identificando obstáculos e desviando-se deles a uma distância segura.

Uma contribuição significativa para este projeto é a implementação de um

novo método de classificação de objetos. Esse método categoriza os objetos em quatro classes distintas: Objeto Dinâmico - Ser Vivo, Objeto Dinâmico - Inanimado, Objeto Estático - Ser Vivo e Objeto Estático - Inanimado. Essas categorias foram projetadas para abranger uma ampla variedade de objetos que possuem características semelhantes, facilitando a interpretação dos dados de percepção.

Ao categorizar os objetos dessa maneira, o objetivo é obter informações mais precisas para gerar a zona de segurança ao redor do robô. Isso significa que o robô pode distinguir entre objetos que são móveis, como seres vivos e outros robores, e objetos estáticos, como móveis e plantas. Essa distinção é crucial para garantir que o robô possa reagir adequadamente a diferentes tipos de obstáculos, ajustando seu comportamento de desvio conforme necessário.

A Figura 4.1 destaca a capacidade do sistema de detecção, utilizando o YOLO, em identificar e classificar dois obstáculos distintos: uma cadeira e um carro. Ele classifica esses objetos como Objeto Estático - Ser Inanimado e Objeto Dinâmico - Ser Inanimado, respectivamente. Essa classificação precisa é essencial para que o robô tome decisões apropriadas, considerando a natureza e o movimento potencial de cada obstáculo em seu caminho.



FONTE: De autoria própria

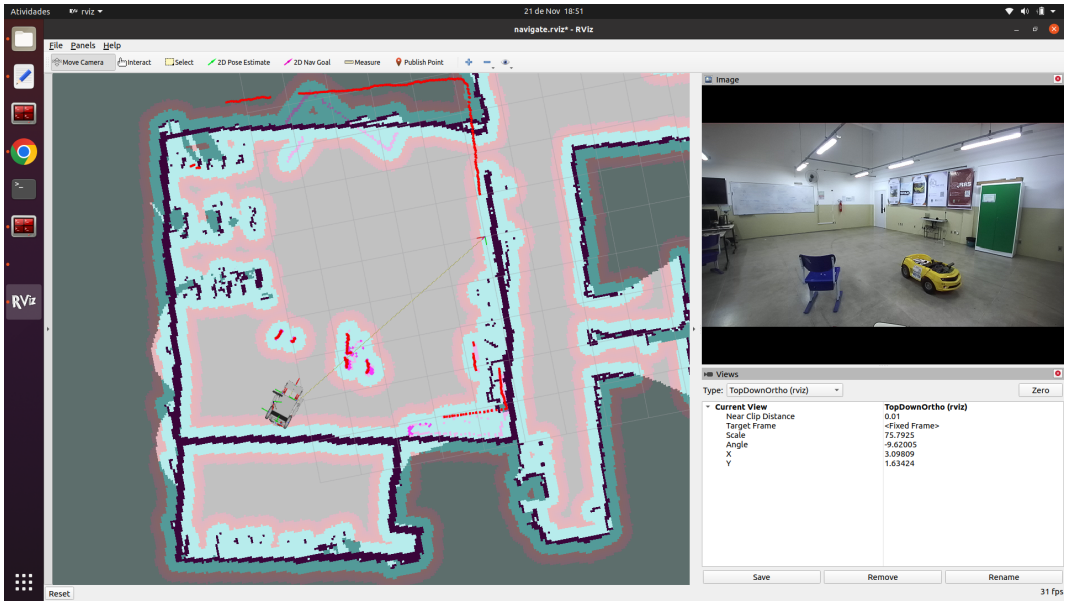
Figura 4.1 – *Visualização dos obstáculos através do YOLO*

4.2 Navegação

Na navegação autônoma, é crucial que o robô tenha a habilidade de reconhecer e evitar obstáculos. Além disso, é necessário que ele se desloque a uma

distância segura desses obstáculos, garantindo a segurança tanto do usuário da cadeira de rodas quanto do público em geral. Essa necessidade conduz à criação de áreas de segurança, as quais estabelecem a distância mínima que o robô pode manter durante seu percurso.

De forma geral, a navegação já possui uma área ao redor dos obstáculos onde o robô não pode entrar. No entanto, essa área possui o mesmo raio de inflação, conforme mostrado na Figura 4.2.



FONTE: De autoria própria

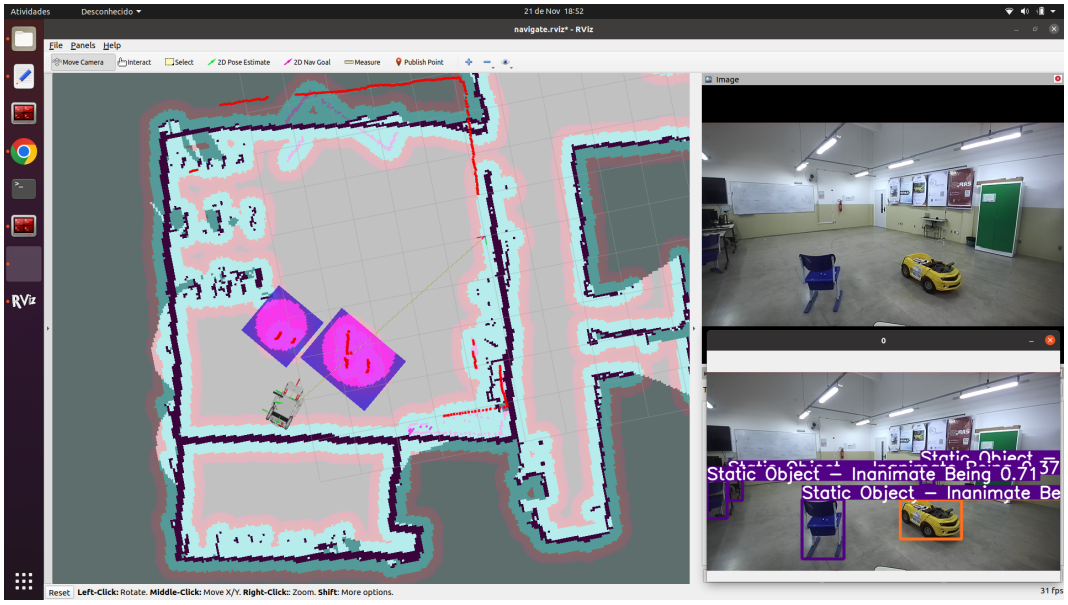
Figura 4.2 – *Visualização do Rviz com o raio de inflação pre-estabelecidos*

Com esta pesquisa, cada classe (ou objeto) detectado terá uma área de segurança específica. Na Figura 4.3, é possível perceber que a partir do momento que um obstáculo é detectado pelo YOLO, uma nova área é gerada no RViz. A imagem mostra dois obstáculos diferentes sendo detectados, uma cadeira e um carro, e ambos possuem áreas de segurança distintas, sendo a do carro um pouco maior em comparação à cadeira.

Nesta pesquisa, a abordagem adotada é a personalização das áreas de segurança com base na classe específica de cada objeto detectado. Cada classe (ou tipo de objeto) identificado pelo sistema de visão computacional, no caso, o YOLO, terá uma área de segurança designada. Isso permite que o robô ajuste dinamicamente as zonas de segurança conforme a natureza e as características específicas de cada obstáculo.

A Figura 4.3 ilustra esse conceito de maneira visual. Quando um obstáculo é detectado pelo YOLO, uma nova área de segurança é gerada e apresentada no

ambiente de visualização tridimensional, representado pelo RViz. A imagem destaca dois obstáculos distintos: uma cadeira e um carro. Notavelmente, as áreas de segurança associadas a cada objeto são claramente distintas, refletindo a natureza específica de cada classe.



FONTE: De autoria própria

Figura 4.3 – *Visualização do Rviz com a zona de segurança gerada pelo Finder*

Essa diferenciação nas áreas de segurança é crucial para otimizar a navegação do robô. Por exemplo, pode ser necessário atribuir uma zona de segurança maior para objetos potencialmente mais perigosos, como um carro, em comparação com objetos menos impactantes, como uma cadeira. Essa abordagem adaptativa leva em consideração não apenas a presença do obstáculo, mas também suas características intrínsecas.

Além disso, a personalização das áreas de segurança contribui para a eficiência operacional do robô, permitindo-lhe navegar de maneira mais ágil e segura no ambiente. Essa abordagem dinâmica é particularmente útil em ambientes dinâmicos nos quais diferentes tipos de obstáculos podem coexistir e requerem respostas distintas.

Assim, utilizando a nova classificação, estabelecemos um novo raio de inflação para cada classe, como apresentado na Tabela 4.1.

Tabela 4.1 – *Nova classificação e seus respectivos raios*

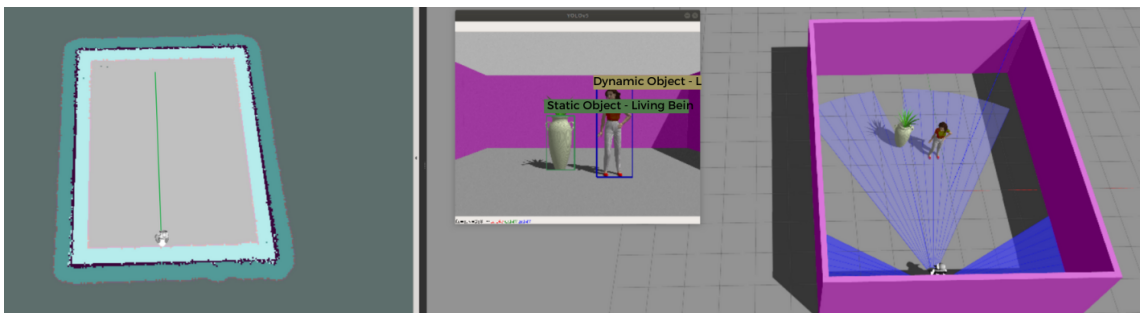
Classes	Raio (m)
Dynamic Object - Living Being	0,8
Dynamic Object - Inanimate Being	1,0
Static Object - Living Being	0,6
Static Object - Inanimate Being	0,7

FONTE: De autoria própria

Para verificar a eficácia dessas zonas, foram realizados dois testes. O primeiro teste ocorreu em um ambiente virtual criado no simulador Gazebo, utilizando um robô virtual. Já o segundo teste foi realizado em um ambiente real, utilizando uma cadeira de rodas autônoma. Em ambos os testes, o robô identifica os obstáculos e, com base nessa informação, gera a zona de segurança correspondente à sua respectiva classe.

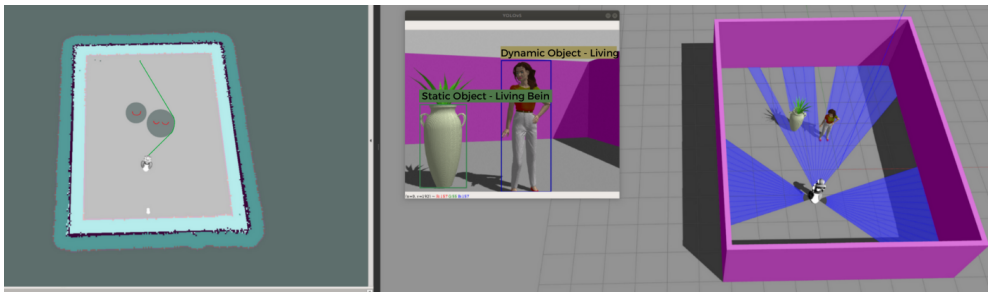
4.2.1 Teste 1: ambiente virtual

Neste primeiro teste em um ambiente virtual, o robô foi posicionado em uma sala contendo dois objetos de classes diferentes. O desafio era percorrer de um lado a outro da sala, passando por esses objetos e respeitando suas respectivas zonas de segurança. Na Figura 4.4, o robô inicia seu percurso olhando para o caminho mais curto (linha verde). É possível notar que o robô já é capaz de identificar os obstáculos à sua frente, mas, devido à distância considerável, eles não aparecem com precisão, e a zona de segurança não é exibida. À medida que o robô se aproxima dos objetos, como mostrado na Figura 4.5, a distância é mais precisa, resultando na geração da zona de segurança no mapa. Com isso, a trajetória do robô (linha verde) foi atualizada, respeitando as zonas de segurança.



FONTE: De autoria própria

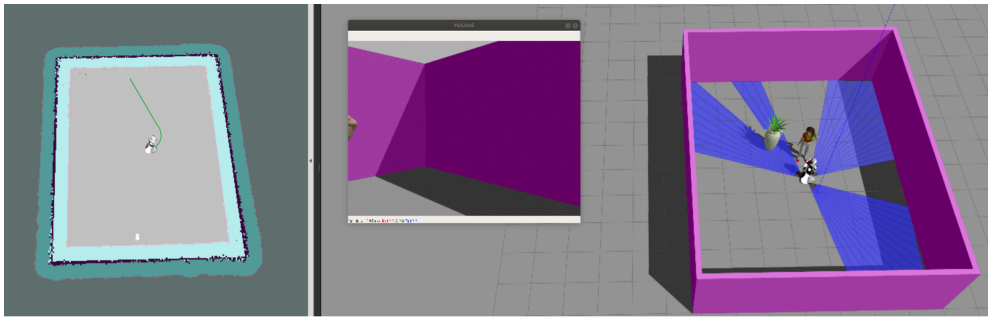
Figura 4.4 – *Ponto de partida do robô. Da esquerda para a direita: visualização do mapa no RVIZ, identificação de objetos pelo YOLO e visualização do ambiente pelo Gazebo*



FONTE: De autoria própria

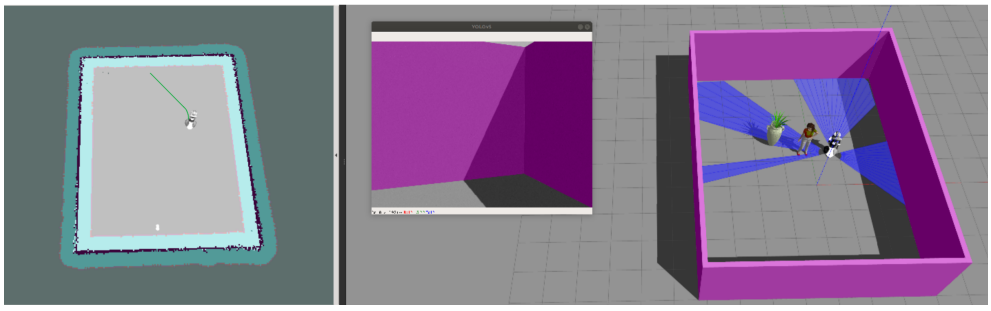
Figura 4.5 – *Zonas de segurança visíveis. Da esquerda para a direita: visualização do mapa no RVIZ, identificação de objetos pelo YOLO, e visualização do ambiente no Gazebo*

As Figuras 4.6 e 4.7 mostram o robô em sua fase final de trajetória, contornando o objeto à sua esquerda. Mesmo que não detecte mais os objetos, sua trajetória não é alterada, permanecendo dentro das zonas de segurança criadas.



FONTE: De autoria própria

Figura 4.6 – *Robô contornando o objeto à sua esquerda. Da esquerda para a direita: visualização do mapa no RVIZ, identificação de objetos pelo YOLO e visualização do meio ambiente no Gazebo*



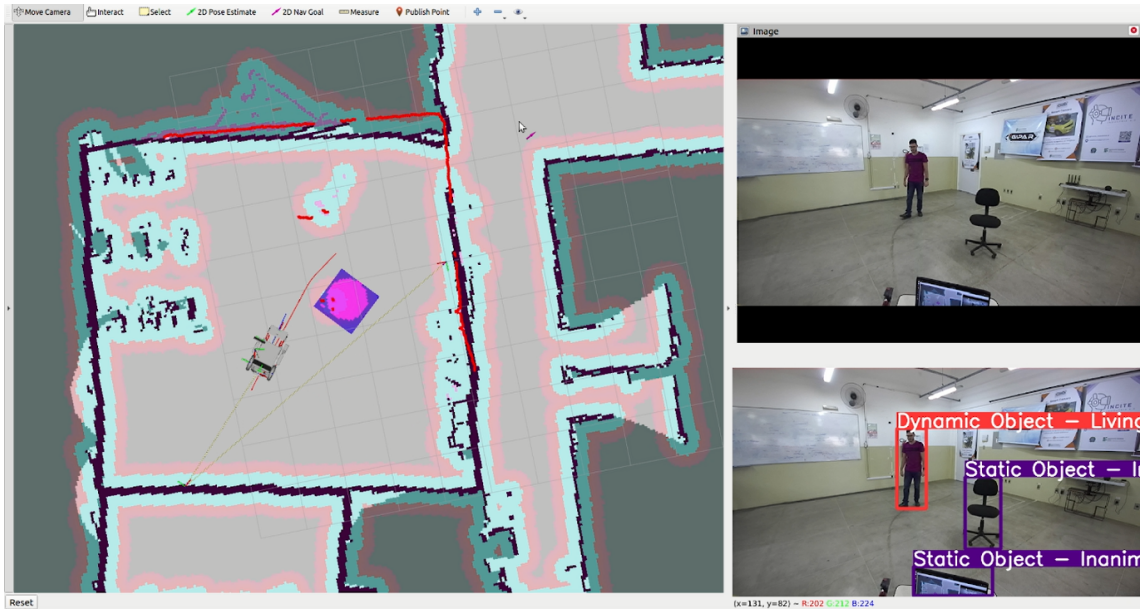
FONTE: De autoria própria

Figura 4.7 – *Etapa final da trajetória. Da esquerda para a direita: visualização do mapa no RVIZ, identificação de objetos pelo YOLO e visualização do ambiente no Gazebo*

4.2.2 Teste 2: ambiente real

No segundo teste, a validação do Finder foi realizada em um ambiente do mundo real, utilizando um robô físico. Nesse cenário prático, o robô foi colocado em uma sala que continha diversos objetos, cada um pertencente a uma classe específica. O principal desafio consistia em navegar de um lado para o outro da sala, atravessando os objetos presentes, enquanto respeitava as zonas de segurança designadas para cada tipo de obstáculo.

A Figura 4.8 fornece uma visão ilustrativa desse teste específico. Nesse exemplo, o robô detecta a presença de uma cadeira e, com base nessa detecção, gera dinamicamente a sua zona de segurança associada. Essa zona de segurança é essencial para garantir que o robô permaneça a uma distância segura do objeto durante a navegação.



FONTE: De autoria própria

Figura 4.8 – *Robô contornando o objeto à sua direita. Da esquerda para a direita: visualização do mapa no RVIZ e visualização do ambiente em tempo real*

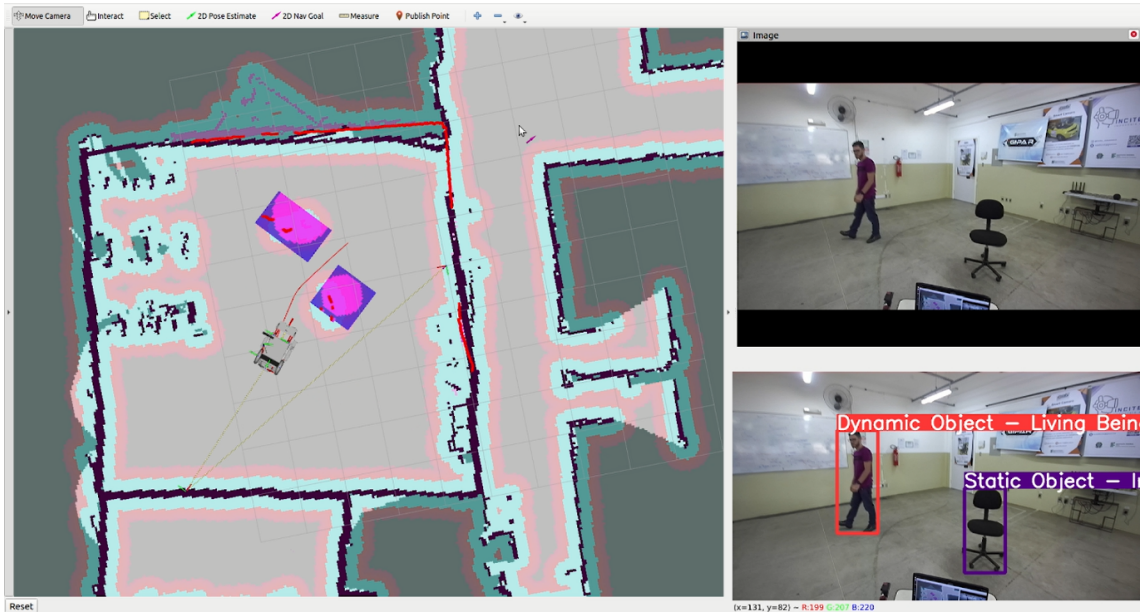
Ao se deparar com a cadeira, o sistema de controle do robô ajusta sua trajetória para contornar o obstáculo de maneira segura, evitando colisões. A representação visual na Figura 4.8 mostra claramente como a trajetória original do robô (linha vermelha) é modificada para contornar a cadeira, escolhendo um caminho alternativo que respeita a zona de segurança estabelecida para esse tipo específico de objeto.

Esse comportamento adaptativo demonstra a eficácia do Finder em lidar com

situações do mundo real, onde a presença de obstáculos de diferentes classes exige respostas específicas. A capacidade do robô de modificar sua trajetória com base nas zonas de segurança definidas dinamicamente ilustra a utilidade prática dessa abordagem em ambientes dinâmicos e complexos.

Esses resultados validam a robustez do Finder em ambientes reais, destacando sua capacidade de navegar com sucesso, contornando obstáculos e respeitando zonas de segurança específicas para cada classe de objeto, contribuindo assim para uma navegação autônoma, eficiente e segura.

Na Figura 4.9, o robô gera uma segunda zona de segurança após identificar uma pessoa. É possível perceber que a trajetória do robô já é atualizada, permitindo que ele passe entre os dois obstáculos sem invadir suas áreas de segurança.



FONTE: De autoria própria

Figura 4.9 – *Etapa final da trajetória. Da esquerda para a direita: visualização do mapa no RVIZ e visualização do ambiente em tempo real*

Os resultados da navegação em ambiente real podem ser verificados por meio do vídeo disponibilizado na plataforma do YouTube, acessível pelo seguinte link: <https://youtu.be/IwYOkD_M0IQ>

Capítulo 5

Considerações Finais

Após realizar os testes, verificamos a viabilidade do sistema, destacando a importância da percepção robocêntrica, que proporciona uma noção tridimensional do ambiente.

De maneira abrangente, foi possível desenvolver um sistema em robótica móvel que contribui para a navegação social, com ênfase na perspectiva centrada no robô, por meio da identificação de classes de obstáculos.

No âmbito da navegação, as informações produzidas pelo sistema foram utilizadas para alimentar um modelo de navegação preexistente, visando aprimorar seu desempenho em ambientes de trabalho sociais. Ao final, observou-se uma alteração no comportamento das máquinas com a implementação da navegação e geração das zonas de segurança.

Entretanto, é importante ressaltar que essa percepção pode variar dependendo de diversos aspectos, como as vibrações do robô, luminosidade, condições ambientais e de locomoção, leituras de distância, entre outros.

Para uma navegação autônoma mais sociável, é crucial que o robô respeite algumas áreas, denominadas zonas de segurança. Essas zonas são determinadas com base nas quatro classes propostas neste trabalho, permitindo ao robô determinar a distância mínima que pode se aproximar do obstáculo. A proposta foi testada em ambientes virtuais e reais contendo objetos diversos, nos quais o robô precisou identificar o obstáculo e traçar sua rota com base na zona de segurança determinada pela classificação.

Com base nos resultados obtidos a partir do comportamento do robô diante

da geração da zona de segurança e da trajetória traçada, podemos concluir que a pesquisa alcançou os resultados esperados.

5.0.1 Trabalhos Futuros

Para futuros desenvolvimentos deste projeto, sugerem-se as seguintes direções:

- Aprimoramento da Estimativa de Posição: Analisar mais de um ponto da Bouding Box para estimar melhor a distância e posição do objeto.
- Testes em Ambientes Mais Desafiadores: Realizar testes em ambientes mais desafiadores e complexos, considerando condições adversas, diferentes tipos de obstáculos e variações no ambiente, para avaliar a robustez do sistema em cenários do mundo real.
- Validação em Cenários de Uso Mais Amplos: Expandir a validação do sistema em cenários de uso mais amplos e diversificados, considerando diferentes layouts de ambientes, configurações de obstáculos e demandas específicas de navegação.

Essas sugestões podem contribuir para a evolução e aprimoramento contínuo do sistema de navegação autônoma proposto.

REFERÊNCIAS

AGHAEI, M. et al. Single image human proxemics estimation for visual social distancing. In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. [S.l.: s.n.], 2021. p. 2785–2795. (Citado na página 9.)

BILIUS, L.-B.; VATAVU, R.-D.; MARQUARDT, N. Smart vehicle proxemics: a conceptual framework operationalizing proxemics in the context of outside-the-vehicle interactions. In: SPRINGER. *IFIP Conference on Human-Computer Interaction*. [S.l.], 2021. p. 150–171. (Citado na página 3.)

BLINDHEIM, K. et al. Promoting activity in long-term care facilities with the social robot pepper: a pilot study. *Informatics for Health and Social Care*, Taylor & Francis, p. 1–15, 2022. (Citado na página 2.)

BROWN, N. Edward t. hall: Proxemic theory, 1966. *Center for Spatially Integrated Social Science. University of California, Santa Barbara. http://www. csiss. org/classics/content/13 Read*, v. 18, p. 2007, 2001. (Citado na página 3.)

ELBADAWI, M.; GAISFORD, S.; BASIT, A. W. Advanced machine-learning techniques in drug discovery. *Drug Discovery Today*, Elsevier, v. 26, n. 3, p. 769–777, 2021. (Citado na página 1.)

FANG, W.; WANG, L.; REN, P. Tinier-yolo: A real-time object detection method for constrained environments. *IEEE Access*, IEEE, v. 8, p. 1935–1944, 2019. (Citado na página 3.)

GAZEBOSIM. Sobre o gazebo. <https://gazebosim.org/about> [Acessado em 29 de Novembro de 2023]. 2002. (Citado na página 11.)

GRATEROL, W. et al. Emotion detection for social robots based on nlp transformers and an emotion ontology. *Sensors*, MDPI, v. 21, n. 4, p. 1322, 2021. (Citado na página 2.)

GUPTA, R. et al. Artificial intelligence to deep learning: machine intelligence approach for drug discovery. *Molecular Diversity*, Springer, v. 25, n. 3, p. 1315–1360, 2021. (Citado na página 2.)

HALL, E. T. *The hidden dimension*. [S.l.]: Anchor, 1966. v. 609. (Citado 2 vezes nas páginas 8 and 9.)

HENSCHEL, A.; LABAN, G.; CROSS, E. S. What makes a robot social? a review of social robots from science fiction to a home or hospital near you. *Current Robotics Reports*, Springer, v. 2, n. 1, p. 9–19, 2021. (Citado na página 2.)

HEREDIA, J. et al. Adaptive multimodal emotion detection architecture for social robots. *IEEE Access*, IEEE, v. 10, p. 20727–20744, 2022. (Citado na página 2.)

JOCHER, G. *YOLOv5 by Ultralytics*. 2020. Disponível em: <https://github.com/ultralytics/yolov5>. (Citado 2 vezes nas páginas 7 and 8.)

LEHMANN, H.; ROJIK, A.; HOFFMANN, M. Should a small robot have a small personal space? investigating personal spatial zones and proxemic behavior in human-robot interaction. *arXiv preprint arXiv:2009.01818*, 2020. (Citado na página 9.)

LU, D. V. social$_{n}avigation_{l}ayers$.<> [Acessado em 10 de Novembro de 2023]. 2014. (Citado 2 vezes nas páginas 10 and 11.)

MARDER-EPPSTEIN, E. et al. The office marathon: Robust navigation in an indoor office environment. In: IEEE. *2010 IEEE international conference on robotics and automation*. [S.l.], 2010. p. 300–307. (Citado na página 10.)

QUIROZ, M. et al. Group emotion detection based on social robot perception. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 22, n. 10, p. 3749, 2022. (Citado na página 2.)

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. (Citado 3 vezes nas páginas 6, 7, and 8.)

SHALF, J. The future of computing beyond moore's law. *Philosophical Transactions of the Royal Society A*, The Royal Society Publishing, v. 378, n. 2166, p. 20190061, 2020. (Citado na página 1.)

SHUJA, J. et al. Applying machine learning techniques for caching in next-generation edge networks: A comprehensive survey. *Journal of Network and Computer Applications*, Elsevier, v. 181, p. 103005, 2021. (Citado na página 1.)

SOLAWETZ, J. O que é yolov5? um guia para iniciantes. <https://blog.roboflow.com/yolov5-improvements-and-evaluation/> [Acessado em 09 de Novembro de 2023]. 2020. (Citado na página 6.)

SØRAA, R. A. et al. Children's perceptions of social robots: a study of the robots pepper, av1 and tessa at norwegian research fairs. *AI & society*, Springer, v. 36, n. 1, p. 205–216, 2021. (Citado na página 2.)

WIKI, R. Ros/introduction - ros wiki. <wiki.ros.org> [Acessado em 29 de Novembro de 2023]. 2007. (Citado na página 11.)

APÊNDICE A

Algoritmo para Cálculo da Distância Focal

Código A.1 – *Algoritmo para Cálculo da Distância Focal*

```
1
2 # Importa os pacotes necessários
3 from imutils import paths
4 import numpy as np
5 import imutils
6 import cv2
7
8
9 def find_marker(image):
10 # Converte a imagem em tons de cinza e detecta bordas
11     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
12     gray = cv2.GaussianBlur(gray, (5, 5), 0)
13     edged = cv2.Canny(gray, 35, 125)
14
15 # Encontra os contornos na imagem com bordas e mantêm a maior;
16     cnts = cv2.findContours(edged.copy(), cv2.RETR_LIST,
17 cv2.CHAIN_APPROX_SIMPLE)
18     cnts = imutils.grab_contours(cnts)
19     c = max(cnts, key = cv2.contourArea)
20
21 # Imprime as referências das bordas
```

```
22      print("{}" .format(cv2.minAreaRect(c)))
23      return cv2.minAreaRect(c)
24
25  def distance_to_camera(knownWidth, focalLength, perWidth):
26      # Calcula e retornar a distância do objeto até a câmera
27      return (knownWidth * focalLength) / perWidth
28
29  # Modifique para a distância medida entre o objeto e a
30  # câmera, neste caso é 1 metro
31  KNOWN_DISTANCE = 40.0
32
33  # Modifique para o comprimento do objeto, neste caso é um
34  #caderno de 29 cm
35  KNOWN_WIDTH = 0.29
36
37  # Carrega a imagem, modifique para o caminho dela
38  image = cv2.imread("1.jpg")
39
40  # Chama a função onde encontra o contor da imagem
41  marker = find_marker(image)
42
43  # Calcula distancia focal
44  focalLength = (marker[1][0] * KNOWN_DISTANCE) / KNOWN_WIDTH
45
46  # Imprime a Distancia Focal
47  print(f"Ditancia Focal = {focalLength }")
48
49  inches = distance_to_camera(KNOWN_WIDTH, focalLength,
50  marker[1][0])
51
52  # Desenha uma caixa delimitadora ao redor do objeto
53  box = cv2.cv.BoxPoints(marker) if imutils.is_cv2()
54  else cv2.boxPoints(marker)
55
56  box = np.int0(box)
57  cv2.drawContours(image, [box], -1, (0, 255, 0), 2)
58
```

```
59 # Insere a distância da câmera para o objeto
60 cv2.putText(image, "%.2fm" % (inches),
61   (image.shape[1] - 200, image.shape[0] - 20),
62   cv2.FONT_HERSHEY_SIMPLEX, 2.0, (0, 255, 0), 3)
63
64 # Plota a imagem
65 cv2.imshow("image", image)
66 cv2.waitKey(0)
```

APÊNDICE B

Algoritmo Finder

Código B.1 – *Algoritmo Finder*

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import time, os
import math, os, os.path, rospy
from cv_bridge import CvBridge
from detection_msgs.msg import BoundingBoxes
from geometry_msgs.msg import *
from math import *
from sensor_msgs.msg import LaserScan
from sympy import *
from time import sleep
from visualization_msgs.msg import MarkerArray, Marker
from sensor_msgs.msg import Image as ImageMsg
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.Qt import *
from msgs.msg import People, Person
from geometry_msgs.msg import PoseWithCovarianceStamped
from dynamic_reconfigure.parameter_generator_catkin import *
from visualization_msgs.msg import MarkerArray, Marker

exis = Symbol('exis')
bridge = CvBridge()
id_person = 0
shadow_pos_x = []
```

```
shadow_pos_y = []

robot_pose = [0, 0, 0]
robot_orientation = [0, 0, 0, 0]
detected_objects = []

class finder_v2dot0:

  def __init__(self):
    amcl = rospy.Subscriber('/amcl_pose', PoseWithCovarianceStamped ,
                            self.callback_amcl, queue_size=10)
    yolo_sub = rospy.Subscriber('/yolov5/detections', BoundingBoxes ,
                                self.callback_box, queue_size=10)
    self.pub_goal = rospy.Publisher('/move_base_simple/goal',
                                    PoseStamped, queue_size=10)
    scan_sub = rospy.Subscriber('/scan', LaserScan, self.callback_laser
                                queue_size=10)

  def callback_amcl(self, data):
    global robot_pose, robot_orientation
    robot_pose = [data.pose.pose.position.x, data.pose.pose.position.y,
                  data.pose.pose.position.z]
    robot_orientation = [data.pose.pose.orientation.x,
                         data.pose.pose.orientation.y,
                         data.pose.pose.orientation.z,
                         data.pose.pose.orientation.w]

  def callback_count(self, msg):
    self.counter = msg.count

  def callback_box(self, data):
    os.system('cls' if os.name == 'nt' else 'clear')
    print('\033[1;30;47m=' * 84)
    print("\033[1;30;47m                              FINDER  v 3  ")
    print('\033[1;30;47m=' * 84)
    print("\n\033[1;34;47mYOLOV3 is detecting {} object(s)..."
          .format(len(data.bounding_boxes)))

    self.lmax_person = []
    self.lmin_person = []
    self.x_center = []
    self.y_center = []
    self.classe = []
    self.dist_final = []
    self.dist_x = []
```

```
73      self.dist_y = []
74      self.X_map = []
75      self.Y_map = []
76      dist = []
77      X = []
78      Y = []
79      Z = []
80      Y_robot = []
81      X_robot = []
82
83      numero = len(data.bounding_boxes) - 1
84
85      for f in range(len(data.bounding_boxes)):
86          self.classe.append(data.bounding_boxes[f].Class)
87
88          if (self.classe[f] == "Static Object - Inanimate Being"):
89            self.x_center.append((data.bounding_boxes[f].xmax -
90                                  data.bounding_boxes[f].xmin)/2 +
91                                  data.bounding_boxes[f].xmin)
92
93            self.y_center.append((data.bounding_boxes[f].ymax -
94                                  data.bounding_boxes[f].ymin)/2 +
95                                  data.bounding_boxes[f].ymin)
96
97          else:
98            self.x_center.append((data.bounding_boxes[f].xmax -
99                                  data.bounding_boxes[f].xmin)/2 +
100                                 data.bounding_boxes[f].xmin)
101
102           self.y_center.append((data.bounding_boxes[f].ymax -
103                                 data.bounding_boxes[f].ymin)/2 +
104                                 data.bounding_boxes[f].ymin)
105
106         self.df = 216.3355 # Distância Focal
107         self.cx = 320      # Resolução da imagem em x / 2
108         self.cy = 240      # Resolução da imagem em y / 2
109
110         dist.append(depth_frame[int(self.y_center[f]),
111                                 int(self.x_center[f])])
112
113         X.append(dist[f]*(self.x_center[f] - self.cx)/self.df)
114         Y.append(dist[f]*(self.y_center[f] - self.cy)/self.df)
115         Z.append(dist[f])
116         print(f" : {data.bounding_boxes[f].Class}, {Z[f]}")
117
```

```
118         # CONTROLA O ANGULO DA CAMERA
119         camera_angle = 0 # -3.14 a 3.14
120 
121         # CALCULA A POSICAO DO ROBO
122         q0 = robot_orientation[0]
123         q1 = robot_orientation[1]
124         q2 = robot_orientation[2]
125         q3 = robot_orientation[3]
126 
127         yaw = math.atan2(2*(q0*q1 + q2*q3),
128                          (q0**2 - q1**2 - q2**2 + q3**2))
129         yaw = -yaw - math.pi/2
130 
131         # CONVERTE PARA COORDENADAS DO ROBO
132         Y_robot.append(-(X[f] *math.cos(camera_angle) -
133                          Z[f] *math.sin(camera_angle)))
134 
135         X_robot.append(X[f] *math.sin(camera_angle) +
136                        Z[f] *math.cos(camera_angle))
137 
138         # CONVERTE PARA COORDENADAS DO MAPA
139         yaw_robot = -math.pi/2 - yaw
140 
141         self.X_map.append(X_robot[f] *math.cos(yaw_robot) -
142                           Y_robot[f] *math.sin(yaw_robot) +
143                           robot_pose[0])
144 
145         self.Y_map.append(X_robot[f] *math.sin(yaw_robot) +
146                           Y_robot[f] *math.cos(yaw_robot) +
147                           robot_pose[1])
148 
149   def callback_laser(self, msg):
150 
151     def people_func(nome, X_map, Y_map, xx):
152 
153       person = Person()
154       person.name = nome
155 
156       person.pose.position.x = X_map
157       person.pose.position.y = Y_map
158       person.pose.position.z = 0
159 
160       person.proxemic.spread.x = xx
161       person.proxemic.spread.y = xx
162       person.proxemic.spread.z = 0
```

```
163         person.pose.orientation.w = 1
164
165         person.proxemic.freeBorder = 30
166         person.proxemic.lethalBorder = 68
167
168         people.people.append(person)
169
170       def marker_func(object_type, red, green, blue, lmax, lmin, id_type)
171         marker = Marker()
172         marker.header.stamp = rospy.Time.now()
173         marker.header.frame_id = "odom"
174         marker.ns = object_type
175         marker.type = 3
176         marker.action = marker.ADD
177         marker.scale.x = 0.50
178         marker.scale.y = 0.50
179         marker.scale.z = 0.50
180         marker.color.a = 0.85
181         marker.color.r = red
182         marker.color.g = green
183         marker.color.b = blue
184         marker.lifetime = rospy.Duration(0.30)
185         marker.pose.orientation.w = 0
186         marker.pose.position.x = lmax
187         marker.pose.position.y = lmin
188         marker.pose.position.z = 0.0
189         global id_person
190         for o in markerArray.markers:
191             o.id = id_type
192         markerArray.markers.append(marker)
193
194       marker_pub = rospy.Publisher("/marker_loc", MarkerArray,
195                                    queue_size=10)
196
197       people_pub = rospy.Publisher("/people", People, queue_size=5)
198       rate = rospy.Rate(2) # 10h
199
200       markerArray = MarkerArray()
201       people = People()
202       people.header.frame_id = 'map'
203       people.header.stamp = rospy.Time.now()
204
205       for l in range(len(self.classe)):
206         global id_person
207         id_person += 2
```

```
208
209         if (self.classe[l] == "Dynamic Object - Living Being"):
210           people_func(self.classe[l], self.X_map[l], self.Y_map[l], 0.7)
211           marker_func('object_person', 1.00, 0.00, 0.00, self.X_map[l],
212                       self.Y_map[l], id_person)
213
214         if (self.classe[l] == "Static Object - Inanimate Being"):
215           people_func(self.classe[l], self.X_map[l], self.Y_map[l], 0.5)
216           marker_func('object_person', 1.00, 1.00, 0.00, self.X_map[l],
217                       self.Y_map[l], id_person)
218
219         if (self.classe[l] == "Dynamic Object - Inanimate Being"):
220           people_func(self.classe[l], self.X_map[l], self.Y_map[l], 1)
221           marker_func('object_person', 0.00, 1.00, 0.00, self.X_map[l],
222                       self.Y_map[l], id_person)
223
224         if (self.classe[l] == "Static Object - Living Being"):
225           people_func(self.classe[l], self.X_map[l], self.Y_map[l], 0.3)
226           marker_func('object_person', 0.00, 1.00, 0.00, self.X_map[l],
227                       self.Y_map[l], id_person)
228
229       end_time = time.time() + 1
230       countTimer = 0.00
231       sleepTime = 0.50
232       while time.time() < end_time:
233         time.sleep(sleepTime)
234         people_pub.publish(people)
235         marker_pub.publish(markerArray)
236         rospy.sleep(0.0001)
237
238 def callback_depth(data):
239   global depth_frame
240   depth_frame = bridge.imgmsg_to_cv2(data, "passthrough")
241
242 if __name__ == '__main__':
243   rospy.init_node('finder_alpha')
244   os.system('cls' if os.name == 'nt' else 'clear')
245   print('\033[1;30;47m=' * 84)
246   print("\033[1;30;47m                                FINDER v3.0    ")
247   print('\033[1;30;47m=' * 84)
248   sleep(1.00)
249
250   finder_v2dot0()
251   rospy.Subscriber("/zed2i/zed_node/depth/depth_registered", ImageMsg,
252                    callback_depth)
```

```
253
254   try:
255     rospy.spin()
256   except KeyboardInterrupt:
257     print("Shutting down...")
```

ANEXO I

Plug-in Gaussiana

Código I.1 – *Plug-in Gaussiana*

```
/**
 * @file proxemic_layer.cpp
 * Implementation of the proxemic functionality
 * @author <a href="mailto:sebastian.hoose@iml.fraunhofer.de">
 * Sebastian Hoose</a>
 *
 * (c) all rights reserved
 */

#include "proxemic_layer.h"
#include <pluginlib/class_list_macros.h>

PLUGINLIB_EXPORT_CLASS(proxemic_layer::ProxemicLayer, costmap_2d::Layer)

namespace proxemic_layer
{

  ProxemicLayer::ProxemicLayer() {}


  void ProxemicLayer::peopleCallback(const msgs::People& msg)
  {
    currentPeople = msg;
//     ROS_INFO("[Costmap Proxemics Plugin] received new people msg");
  }


```

```
28  void ProxemicLayer::onInitialize()
29  {
30    ros::NodeHandle nh("~/" + name_);
31    current_ = true;
32
33    //Receive current dynamic parameters
34    dynamic_reconfigure::Server<ProxemicConfig> server;
35    dynamic_reconfigure::Server<ProxemicConfig>::CallbackType callback;
36    callback = boost::bind(&ProxemicLayer::reconfigureCB, this,_1, _2);
37    server.setCallback(callback);
38
39    //Advertise people subscriber
40    peopleSub = nh.subscribe("/finder", 10,
41                             &ProxemicLayer::peopleCallback,this);
42
43    //Initialize members
44    int localMaxTimePassed = 0;
45    nh.param("max_time_passed", localMaxTimePassed, int(60));
46    maxTimePassed_ = ros::Duration(localMaxTimePassed);
47    nh.param("gaussian_renorming", gaussian_renorming_, int(150));
48    ROS_INFO("[Costmap Proxemics Plugin] recieved new people");
49  }
50
51
52  void ProxemicLayer::reconfigureCB(ProxemicConfig &config,
53                                    uint32_t level)
54  {
55    enabled_ = config.enabled;
56    maxTimePassed_ = ros::Duration(config.max_time_passed);
57    gaussian_renorming_ = config.gaussian_renorming;
58    ROS_INFO("[Costmap Proxemics Plugin] set new dynamic parameters");
59  }
60
61
62
63  void ProxemicLayer::updateBounds(double robot_x, double robot_y,
64                                   double robot_yaw, double* min_x,
65                                   double* min_y, double* max_x,
66                                   double* max_y)
67  {
68    //If people msg is too old, do nothing
69    if (!enabled_ || ros::Time::now() - currentPeople.header.stamp >
70        maxTimePassed_)
71     return;
72
```

```
73      geometry_msgs::PoseStamped personPoseInCostmapFrame;
74      geometry_msgs::PoseStamped personPoseInOtherLink;
75
76      //Set all min and max borders to max/min for safe
77      //minimization/maximization
78      for(std::vector<msgs::Person>::iterator personIterator =
79          currentPeople.people.begin();
80          personIterator != currentPeople.people.end();
81          ++personIterator)
82      {
83        //Transform current person to robot coordinatesystem using tf
84        try
85        {
86          personPoseInOtherLink.header.frame_id =
87          currentPeople.header.frame_id;
88
89          personPoseInOtherLink.header.stamp = ros::Time(0);
90          personPoseInOtherLink.pose = personIterator->pose;
91
92          listener.transformPose(layered_costmap_->
93                                 getGlobalFrameID(),
94                                 personPoseInOtherLink,
95                                 personPoseInCostmapFrame);
96        }
97        catch( tf::TransformException ex)
98        {
99          ROS_ERROR("[Costmap Proxemics Plugin] transfrom exception :
100                   %s",ex.what());
101         continue;
102       }
103
104       //Calc max radius of current person
105       double maxRadius = (personIterator->proxemic.spread.x >
106                           personIterator->proxemic.spread.y)
107                           ? personIterator->proxemic.spread.x :
108                           personIterator->proxemic.spread.y;
109
110       //Set bounds of updated area
111       *min_x = std::min(*min_x,
112                         personPoseInCostmapFrame.pose.position.x -
113                         maxRadius +
114                         personIterator->proxemic.centerShift.x);
115
116       *min_y = std::min(*min_y,
117                         personPoseInCostmapFrame.pose.position.y -
```

```
118                         maxRadius +
119                         personIterator->proxemic.centerShift.y);
120
121       *max_x = std::max(*max_x,
122                         personPoseInCostmapFrame.pose.position.x +
123                         maxRadius +
124                         personIterator->proxemic.centerShift.x);
125
126       *max_y = std::max(*max_y,
127                         personPoseInCostmapFrame.pose.position.y +
128                         maxRadius +
129                         personIterator->proxemic.centerShift.y);
130     }
131   }
132
133
134   void ProxemicLayer::updateCosts(costmap_2d::Costmap2D& master_grid,
135                                   int min_i, int min_j, int max_i,
136                                   int max_j)
137   {
138     //if people msg is too old, do nothing
139     if (!enabled_ || ros::Time::now() -
140         currentPeople.header.stamp > maxTimePassed_)
141      return;
142
143     geometry_msgs::PoseStamped personPoseInCostmapFrame;
144     geometry_msgs::PoseStamped personPoseInOtherLink;
145
146     for(std::vector<msgs::Person>::iterator personIterator =
147         currentPeople.people.begin();
148         personIterator != currentPeople.people.end();
149         ++personIterator)
150     {
151       //transform current person to robot coordinatesystem using tf
152       try
153       {
154         personPoseInOtherLink.header.frame_id =
155                   currentPeople.header.frame_id;
156         personPoseInOtherLink.header.stamp = ros::Time(0);
157         personPoseInOtherLink.pose = personIterator->pose;
158         listener.transformPose(layered_costmap_->getGlobalFrameID(),
159                                personPoseInOtherLink,
160                                personPoseInCostmapFrame);
161       }
162       catch( tf::TransformException ex)
```

```
163         {
164           ROS_ERROR("[Costmap Proxemics Plugin] Transfrom exception :
165                     %s",ex.what());
166           continue;
167         }
168 
169         //calc max radius of current person
170         double maxRadius = (personIterator->proxemic.spread.x >
171                             personIterator->proxemic.spread.y) ?
172                             personIterator->proxemic.spread.x :
173                             personIterator->proxemic.spread.y;
174 
175         //set bounds of update area
176         double minX = (personPoseInCostmapFrame.pose.position.x -
177                        maxRadius + personIterator->proxemic.centerShift.x)
178 
179         double minY = (personPoseInCostmapFrame.pose.position.y -
180                        maxRadius + personIterator->proxemic.centerShift.y)
181 
182         double maxX = (personPoseInCostmapFrame.pose.position.x +
183                        maxRadius + personIterator->proxemic.centerShift.x)
184 
185         double maxY = (personPoseInCostmapFrame.pose.position.y +
186                        maxRadius + personIterator->proxemic.centerShift.y)
187 
188         //transform coordinates to costmap coordinates
189         int minXMap, minYMap, maxXMap, maxYMap;
190 
191         master_grid.worldToMapEnforceBounds(minX, minY, minXMap, minYMap)
192         master_grid.worldToMapEnforceBounds(maxX, maxY, maxXMap, maxYMap)
193 
194         for(int y = minYMap; y < maxYMap; y++)
195         {
196           for(int x = minXMap; x < maxXMap;x++)
197           {
198             //calc gaussian
199             double z = 0.0;
200             double xWorld = 0.0;
201             double yWorld = 0.0;
202             master_grid.mapToWorld(x,y,xWorld,yWorld);
203 
204             bool calcOfGaussianSuccess = getGausian(*personIterator,
205                                                     personPoseInCostmapFrame,
206                                                     xWorld,yWorld,z);
207             if(!calcOfGaussianSuccess)
```

```
208           {
209             ROS_ERROR("[Costmap Proxemics Plugin] Gaussian of given
210                       proxemic could not be calculated.");
211             return;
212           }
213 
214           //re-norm gaussian result, see:
215           // http://wiki.ros.org/costmap_2d#Inflation
216           z *= gaussian_renorming_; // e.g. 253.0;
217           uint8_t zInt = static_cast<uint8_t>(z);
218 
219           //some clipping (needs to be parametrized)
220           zInt = (zInt < static_cast<uint8_t>
221                   (personIterator->proxemic.lethalBorder))
222                    ? zInt : costmap_2d::LETHAL_OBSTACLE;
223           zInt = (zInt > static_cast<uint8_t>
224                     (personIterator->proxemic.freeBorder))
225                     ? zInt : costmap_2d::FREE_SPACE;
226 
227           //make sure, other obstacles are not getting overwritten
228           unsigned char maxVal = (zInt > master_grid.getCost(x,y))
229                                   ? zInt : master_grid.getCost(x,y);
230 
231           //setting costs in costmap
232           master_grid.setCost(x, y, maxVal);
233         }
234       }
235     }
236   }
237 
238 
239   bool ProxemicLayer::getGausian(msgs::Person& person,
240                                  geometry_msgs::PoseStamped&
241                                  personPoseInCostmapFrame,
242                                  double x, double y, double& z)
243   {
244     msgs::Proxemic proxemic = person.proxemic;
245 
246     if(proxemic.spread.x <= 0 || proxemic.spread.y <= 0)
247       return false;
248 
249     //rotation corrected by pose of person
250     tf::Pose personPose;
251     tf::poseMsgToTF(personPoseInCostmapFrame.pose, personPose);
252     double personYaw = tf::getYaw(personPose.getRotation());
```

```
253     double theta = proxemic.rotation + static_cast<double>(personYaw);
254     double spreadX = proxemic.spread.x;
255     double spreadY = proxemic.spread.y;
256 
257     //calc shift of gaussian normal distribution
258     double shiftX = (proxemic.centerShift.x +
259                      personPoseInCostmapFrame.pose.position.x);
260     double shiftY = (proxemic.centerShift.y +
261                      personPoseInCostmapFrame.pose.position.y);
262 
263     //calc gaussian
264     double a = ((std::cos(theta)*std::cos(theta))/(2.0*spreadX*spreadX))
265                  + ((std::sin(theta)*std::sin(theta))/
266                   (2.0*spreadY*spreadY));
267 
268 
269     double b = -(std::sin(2.0*theta)/(4.0*spreadX*spreadX))
270                  + (std::sin(2.0*theta)/(4.0*spreadY*spreadY));
271     double c = (std::sin(theta)*std::sin(theta))/(2.0*spreadX*spreadX)
272                  + ((std::cos(theta)*std::cos(theta))/
273                  (2.0*spreadY*spreadY));
274 
275     z = std::exp(-((a*(x-shiftX)*(x-shiftX))+(2.0*b*(x-shiftX)*
276                  (y-shiftY)) + (c*(y-shiftY)*(y-shiftY))));
277 
278     return true;
279   }
280 }
```