UNIVERSIDADE TUIUTI DO PARANÁ

BRUNO DALL' STELLA CARDOSO
DIOGO HENRIQUE OLIVEIRA SOUZA

MAPEAMENTO DE AMBIENTES INTERNOS COM *ROBOT OPERATING SYSTEM* E VISÃO COMPUTACIONAL

CURITIBA

2022

BRUNO DALL' STELLA CARDOSO
DIOGO HENRIQUE OLIVEIRA SOUZA

MAPEAMENTO DE AMBIENTES INTERNOS COM *ROBOT OPERATING SYSTEM* E VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Faculdade de Ciências Exatas e de Tecnologia da Universidade Tuiuti do Paraná, como requisito parcial à obtenção do grau de Bacharel.

Orientador: Prof. Me. Darci Luiz Tomasi Junior

CURITIBA

2022

RESUMO

No cenário atual, onde a evolução e utilização de robôs tem sido cada vez mais presente nas indústrias e meios acadêmicos, o uso de um ambiente simulado para testes é devidamente elaborado e fundamental. Esta pesquisa busca fornecer meios para facilitar a compreensão e utilização de ambientes simulados, através da construção e configuração de um ambiente propício para testes com robôs e objetos totalmente simulados. Esse ambiente terá cômodos de uma casa simulada, onde um robô percorrerá o ambiente e irá capturar imagens para com o uso de uma Rede Neural Convolucional , consiga identificar a imagem do ambiente e, como resultado, indicar em qual cômodo o mesmo se localiza.

Palavras-chave: Ambiente simulado, Rede Neural Convolucional, Inteligência Artificial, Imagens Simuladas.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE CÓDIGOS

LISTA DE SIGLAS E ACRÔNIMOS

AGV	*Automated Guided Vehicle*
AMR	*Autonomous Mobile Robots*
APIs	*Application Programming Interface*
CNN	*Convolutional Neural Network*
GPU	*Graphics Processing Unit*
IA	Inteligência Artificial
LTS	*long-term support*
ROS	*Robot Operating System*
RVIZ	Ferramenta de visualização 3D
SO	Sistema Operacional

SUMÁRIO

1 INTRODUÇÃO

Os robôs móveis autônomos pertencem a um segmento da robótica que se expandiu rapidamente nos últimos anos. Tais robôs são de grande relevância para o desenvolvimento e aprimoramento de vários campos, sendo dispositivos capazes de se locomover dentro de um ambiente e realizar tarefas sem depender de alguém para direcioná-los. Os robôs normalmente são compostos por sensores sofisticados, tendo como objetivo realizar a identificação de objetos, localização, temperatura, entre outros, por *softwares* como o ROS (*Robot Operating System*), uma coleção de *frameworks* voltado para a robótica que disponibiliza um ambiente de criação de códigos que são responsáveis por instruir robôs em quais ações serão realizadas dentro de um espaço específico e também a Inteligência Artificial (IA), que permite ao robô tomar decisões por conta própria através de um banco de dados predefinido.

Na indústria, os robôs vêm ganhando cada vez mais espaço em diversos setores. Principalmente aqueles com tarefas que colocam um funcionário em risco, um exemplo é a mineradora Vale que pretende comprar um “cão-robô” para realizar operações de fiscalização em áreas de mineração, subir e descer escadas, transmitir imagens do ambiente, objetos e até mesmo medir a temperatura do ambiente (BÔAS, 2021). O Perceverance, é outro robô que foi enviado à Marte pela NASA com o objetivo de capturar imagens e amostra de rochas. Permitindo assim, que cientistas, através desses dados, identifiquem se o ambiente contém ou não sinais de vida microbiana ancestral (ROCHA, 2021).

Para realizar a navegação de um robô em um ambiente desconhecido, alguns recursos são utilizados para estimar o posicionamento e a localização. Quando a base está em processo de navegação e faz uso desses recursos, erros provenientes do ambiente e da base são inseridos no sistema, resultando em estimativas incorretas. Uma forma de reduzir a amplitude dos erros é com a utilização de um ambiente simulado combinado com diferentes técnicas e recursos. Compreendendo a importância dessa área, se faz necessário o desenvolvimento de uma ferramenta que contribua para o avanço operacional e técnico desse segmento, utilizando tecnologias em conjunto que serão capazes de identificar ambientes distintos em áreas simuladas.

O objetivo principal deste trabalho é avaliar o desempenho de uma CNN (rede neural convoluciona), que através de treinamentos consiga classificar e reconhecer imagens de um ambiente simulado. As imagens de entrada são coletadas do ambiente simulado, que englobam ambientes de quarto, cozinha e sala. As imagens são rotuladas e armazenadas em um banco de dados e posteriormente utilizadas como dados de entrada no treinamento da CNN, tornando possivel observar o desempenho do modelo configurado e adaptado, com uma importante contribuição na área de pesquisa

relacionada a soluções para problemas robóticos comuns.

Uma das primeiras etapas deste trabalho foi realizar a configuração de um ambiente interno simulado de uma casa, com isso três cômodos foram criados, quarto, cozinha e sala. Esses ambientes contêm objetos específicos que normalmente são encontrados em ambientes reais (cadeira, mesa, cama e armario), os cômodos quarto e cozinha contém papel de parede, já a sala um tapete, tais elementos foram atribuídos apenas para diferenciar os cômodos uns dos outros e também como forma de avaliar o comportamento da Rede Neural Convolucional.

O robô Turtlebot3 *waflle* PI foi configurado e percorreu o ambiente simulado com o objetivo de capturar e rotular imagens de cada cômodo, essa coleta foi realizada de forma manual onde 2.500 imagens de cada cômodo foram armazenadas em repositórios da plataforma do github.

Com os repositórios do github começa a etapa de configuração e criação de uma boa base de dados para usar em um modelo de treinamento de uma Rede Neural Convolucional, essa base de dados conhecida como *dataset* é muito importante pois problemas como ruídos e distorções não forem tratados de maneira correta, poderão causar problemas futuros, como o problema de *overfitting* muito comum no treinamento de redes neurais.

Uma rede neural é composta por um modelo que através de *datasets* realiza um treinamento e aprende com novos dados de entrada, com tudo para chegar em um modelo com valores satisfatórios muitas das vezes é preciso realizar alterações na estrutura do modelo pré-definido. Essas alterações normalmente são realizadas com o aumento de camadas, filtros e funções auxiliares para gerar um resultado próximo ou superior ao esperado. Com isso foi construído e explicado de maneira detalhada um modelo de CNN que identifica imagens simuladas de cômodos interno de uma casa.

Esta monografia está organizada da seguinte forma:

No Capítulo 2 é apresentada toda a fundamentação teórica utilizando conceitos inter-relacionados com as áreas de *frameworks* do ROS, Inteligência Artificial e Visão Computacional. No Capítulo 3 é apresentada a revisão da literatura correlata e no capítulo 4 é apresentada a metodologia desenvolvida e os resultados obtidos. No capítulo 6 são apresentadas as considerações finais.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentado o conteúdo referente a fundamentação teórica dos conceitos relacionados.

2.1 ROBÓTICA

2.1.1 Introdução a robótica

O significado de ’robô’ foi popularizado pelo dramaturgo theco Karel Capek em 1921 com sua peça *Robôs universais de Rossum*(RUR), todavia a palavra ’robô’ tem como resultado as palavras thecas *rabota* que significa ’trabalho obrigatório’ e *robotnik*, que significa ’servo’, atualmente a maioria dos robôs tem como objetivo realizar trabalhos obrigatórios, tais como montagem de automóveis e sequenciamento de DNA por exemplo MatariĆ (2014).

A Robótica sempre se manteve nos interesses da sociedade, cercada de expectativas retratadas principalmente através da ficção científica. O fato é que o senso comum entre os estudiosos na área da robótica é que ela assumirá um papel de grande relevância no nosso cotidiano, deixando de ficar somente na ficção científica e participando do dia a dia das pessoas.

Entretanto, ainda não existe uma definicação referente à capacidade de inteligência, interatividade e autonomia que um robô pode alcançar, nem sequer quantos anos levariam para que isso torna-se realidade, ou até mesmo se seria possível um robô alcançar níveis de capacidades cognitivas semelhantes aos dos seres humanos.

Atualmente suas principais aplicações são direcionadas para o ramo da indústria, realizando a tarefas repetitivas as quais os humanos possam ser substituídos, dentre as principais vantagens de sua utilização então a alta produtividade, flexibilidade e melhor qualidade dos produtos MatariĆ (2014).

No entanto um dos maiores desafios enfrentados pela robótica é que ela abrange várias áreas da ciência e da engenharia. Exemplos são inteligência artificial, síntese de fala, visão computacional, ciência da computação e engenharia de materiais, portanto, para progredir nessa área, é preciso que hajam avanços nessas outras áreas para que seja possível integrar todas essas tecnologias em um projeto YANG (2022).

2.1.2 Definição de robô

De acordo com MatariĆ (2014) "Um robô é um sistema autônomo que existe no mundo físico, pode sentir o seu ambiente e pode agir sobre ele para alcançar alguns objetivos". Na maioria das vezes são sistemas que possuem automação

onde realizam trabalhos e movimentações humanas, são gerenciados por humanos e dotados de sistemas eletrônicos. Robôs industriais usados em linhas de produção são a forma mais comum de robôs, são máquinas que realizam tarefas que exigem esforço repetitivo, de precisão, resistência, velocidade e força, assim podendo mover ferramentas, equipamentos especiais, peças e outros itens. Essa situação também vem sofrendo alterações nos últimos anos, com a popularidade dos robôs comerciais de limpeza de pisos e automação residencial (GAMERO, 2018) como mostra a Figura 1, o robô comercial Roomba usado para tarefas de limpeza doméstica.

FIGURA 1 – Robô Roomba da iRobot realizando mapeamento e limpeza em uma residência.

FONTE: Astor, 2017.

2.1.3 Aplicações da robótica

Com o avanço da tecnologia, a robótica tornou-se fundamental em diversos campos quando visa-se agilidade, precisão e escalabilidade. Suas aplicações vão desde a área industrial até a residencial, facilitando e aprimorando a maneira como os seres humanos interagem com as tarefas.

Na medicina, o uso de instrumentos robóticos para a realização de procedimentos cirúrgicos delicados e minimamente invasivos estão se tornando cada vez mais comuns.

Os médicos contam com tecnologia de última geração da robótica para a execução de procedimentos mais seguros, precisos, rápidos e com uma redução significativa de dores e traumas, esses são alguns dos benefícios da robótica aplicada para a medicina.

Estes instrumentos robóticos são utilizados como extensão da mão do cirurgião, obedecendo aos comandos do operador, os movimentos são realizados de forma escalonada, evitando assim possíveis erros e realizando acesso a áreas do corpo humano com maior grau de delicadeza, aplicações de robôs nesse segmento são na área cirúrgica, como na urologia, ginecologia e outros tipos de cirurgia torácica e abdominal. (WARELINE, 2015).

Na indústria, os robôs tiveram suas origens com propósitos definidos para realização de tarefas comuns, no entanto, atualmente eles são versáteis e podem ser reprogramados para desempenhar diversas funcionalidades em certos segmentos da área industrial. Algumas aplicações da robótica na indústria são: montagem, pintura, movimentação de cargas, reconhecimento de imagem e testes. (GAMERO, 2018).

Hoje em dia, os robôs articulados são os mais aplicados na indústria. Eles têm esse nome devido a sua configuração mecânica ser semelhante a um braço mecânico, apresentados na Figura 2.

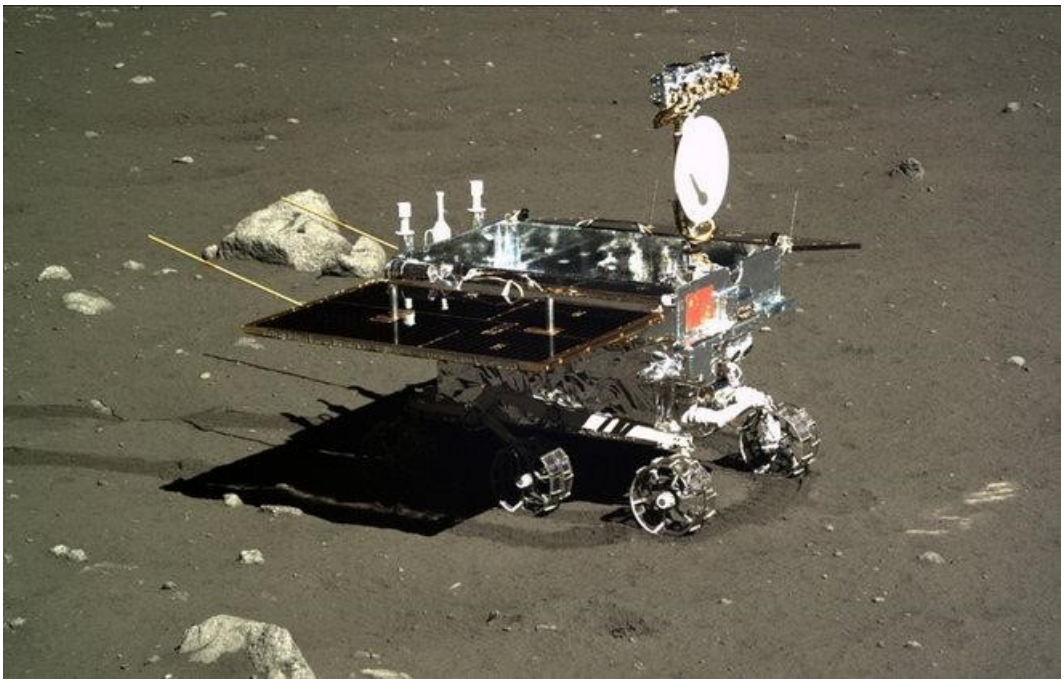
O braço é conectado a uma base com uma junta de torção que permite que o robô gire. Entre os tipos de robôs articulados, aqueles que possuem 6 juntas são os mais utilizados, seu *design* oferece máxima flexibilidade em comparação com os robôs de eixos menores (SILVEIRA, 2019).

FIGURA 2 – Tipos de Robôs Articulados.

FONTE: Silveira, 2019.

Os robôs também estão sendo utilizados para missões fora da Terra por exemplo, como o robô explorador chinês Yutu apresentado na Figura 3, que fez parte da missão Chang'e 3 com o objetivo de explorar da lua. Ele foi lançado em 2013 e permaneceu operacional na superfície lunar até final de 2016, superando consideravelmente sua expectativa de vida útil que era de apenas 90 dias. Seu principal objetivo na missão era realizar análises geológicas e contribuir com informações relevantes para futuras missões espaciais da Agência Espacial Chinesa (PEOPLE, 2017).

FIGURA 3 – Robô Yutu realizando exploração na lua.



FONTE: Reprodução: CASC/China Ministry of Defense.

2.1.4 Robôs Móveis

A robótica móvel é um tema com muita relevância nos dias atuais, com o aumento de pesquisas neste segmento. A utilização em aplicações domésticas e industriais são diversas.

Os robôs móveis são essenciais para a transferência de mão de obra humana para atividades de maior relevância, enquanto procedimentos menos complexos podem ser desempenhados pelos dispositivos.

Os robôs móveis possuem como característica principal, a capacidade de locomoção, esta ação pode ser realizada de forma semi ou completamente autônoma. Entretanto, maiores níveis de autonomia poderão ser alcançados somente com a integração de outros aspectos de maior relevância, como sensores de detecção, atuadores para realizar a locomoção e também inteligência para que possa ser possível lidar com situações complexas e obter resultados (WOLF *et al.*, 2009).

2.1.5 Robôs móveis: AGV X AMR

Os *“Automated Guided Vehicle"*(AGV) precisam de faixas, guias ou fitas magnéticas, ou até mesmo um ambiente controlado para se deslocar, como demonstra a Figura 4. Apesar de muitas indústrias utilizarem robôs móveis que possuem esta tecnologia, já existem tecnologias mais avançadas chamadas de AMR *Autonomous Mobile Robots*(AGV) (FERSILTEC, 2021).

FIGURA 4 – Robô AGV realizando transporte de carga.

FONTE: Fersiltec, 2021.

Os (AMR) são dispositivos 100% autônomos que precisam somente de sensores e digitalizadores para realizar o deslocamento. Ao contrário da tecnologia AGV, não é necessário um ambiente simulado, pois possuem câmeras embutidas e sensores. Assim eles conseguem até mesmo selecionar a melhor rota para se deslocar. Consequentemente, os robôs são mais completos em sentido de tecnologia, fácil programação e maior flexibilidade de locomoção. (FERSILTEC, 2021). A Figura 5, mostra um exemplo de robô AMR que é utilizado pela empresa Amazon.

FIGURA 5 – Robô AMR da Amazon.

FONTE: Fersiltec, 2021.

2.2 *ROBOT OPERATING SYSTEM* (ROS)

O ROS é um *framework* com um grande conjunto de ferramentas e bibliotecas de *software* de código aberto, que disponibiliza um amplo desenvolvimento em grupo, possibilitando o reúso de códigos, criado em 2000 no laboratório Willow Garage da Universidade Stanford, sendo um laboratório para pesquisa em robótica e incubadora de tecnologia dedicada no desenvolvimento de *hardware* e *software* de código aberto. O ROS originalmente foi projetado para superar alguns desafios encontrados no desenvolvimento robótico, tendo como Sistema Operacional host o Linux, suportando outras distribuições como, OS X, Debian, Ubuntu ARM, Yocto, Android, entre outros (WIKI.ROS.ORG, 2021b).

O principal objetivo do ROS é o desenvolvimento de ferramentas que possibilitam a construção de *software* para robôs, sendo esses autônomos ou não. Essa plataforma também é composta por bibliotecas com algoritmos comumente utilizados e com seu próprio blog utilizado pelos usuários como mecanismo de comunicação. Este blog permite aos usuários a troca de informações, correções de *bugs*, auxílios em pacote, bibliotecas e erros, entre outros (ARAÚJO, 2017).

As bibliotecas do ROS contêm uma coleção de códigos que facilitam a construção de *softwares*, essas bibliotecas criadas pelos usuários são conhecidas como clientes ROS e permitem escrever nós ROS, assinar e publicar tópicos, escrever e chamar serviços e utilizar servidor de parâmetros. Embora o foco seja fornecer suporte para Python e C++, essas bibliotecas podem ser implementadas em qualquer outra linguagem de programação (ROS.ORG, 2021).

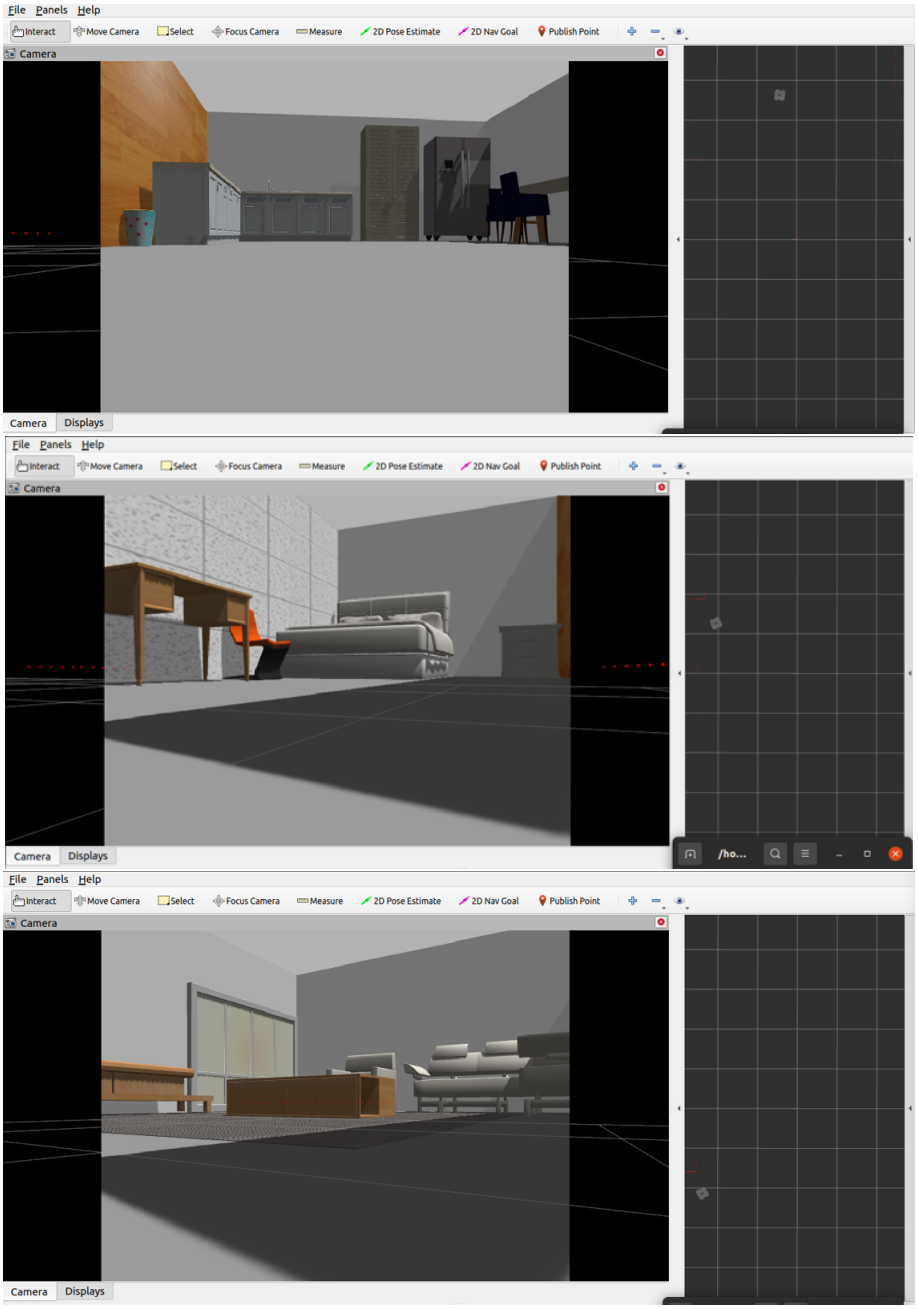
2.2.1 *Workspace Catkin*

catkin é o nome do sistema de compilação oficial do ROS e o sucessor do sistema de compilação original do ROS que era chamado de *rosbuild*, o *catkin* permite melhor distribuição de pacotes e melhor portabilidade para outros sistemas operacionais. O *Workspace* é o diretório onde são realizadas todas as instalações e modificações de pacotes relacionados ao ROS (WIKI.ROS.ORG, 2021a).

2.2.2 RVIZ

RVIZ é uma ferramenta de visualização tridimensional para ROS, sua principal aplicação é ajudar a visualizar o que o robô está vendo e fazendo. O RVIZ permite a visualização em tempo real da posição e local do robô, também conta com um mapa 3D do ambiente. Além disso é possível definir um alvo de posicionamento e fazer com que as bibliotecas de navegação do ROS movam o robô para o alvo. A Figura 6 ilustra a utilização do RVIZ (WIKI.ROS.ORG, 2021c).

FIGURA 6 – Imagem retiradas do ambiete utilizando o RVIZ.



FONTE: Os próprios autores.

2.2.3 Ambiente simulado

Um ambiente simulado agrega muito na aprendizagem, auxiliando em áreas como informática, eletrônica, química, entre outras. Além de oferecer uma experiencial real através de ambientes totalmente simulados, ele diminui muito as ocorrências de acidentes, riscos e até mesmo em custos. Um simulador robótico possibilita a criação e utilização de ferramentas que assemelham a robôs em um ambiente simulado para realização de diversas tarefas, possibilitando a execução de ações através de funções que se aproximam da experiência de um cenário real (SILVA *et al.*, 2002).

2.2.3.1 Gazebo

O *software* Gazebo surgiu em 2002 na Universidade do sul da Califórnia, com o objetivo principal suprir a necessidade de simular um robô em ambientes diversos externos. Ele é um poderoso simulador 3D de robótica, com grande comunidade e totalmente *Open Source*. Sua utilização é muito difundida no meio de indústrias e academia, grande atuante para simulação de sensores e controle de atuadores ele oferece renderização realista de ambientes em geral, com: iluminação, sombras, texturas, e permitindo modelar sensores para identificação de ambientes e objetos (POUBEL, 2019). A Figura 7 apresenta o Gazebo, com a visualização dentro do ambiente simulado. As digitalizações são exibidas pelo RViz um visualizador para tópicos do ROS.

FIGURA 7 – Ambiente do Gazebo.

FONTE: Poubel, 2019.

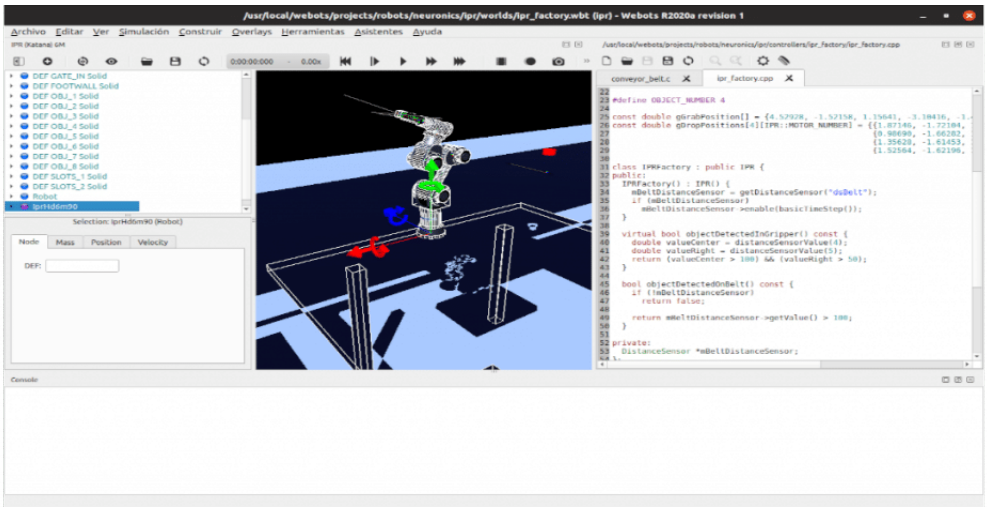
Um robô simulado muito conhecido no ambiente simulado do Gazebo é chamado de Dolly, uma ovelha robô seu objetivo é seguir um indivíduo carregando seu material pesado. Ele é composto por um *scanner* a laser para detecção de objetos e duas rodas motorizadas para que se locomova através do ambiente simulado. Este robô recebe códigos com funções que tem como finalidade instruir o robô em suas ações, ir para a frente, parar, identificar objetos, entre outras (POUBEL, 2019).

2.2.3.2 Webots

O *software* Webots é um poderoso simulador robótico 3D, utilizado na indústria, educação e pesquisa, multiplataforma suportada para Windows, MacOS e Gnu/Linux. Ele surgiu em 1996 desenvolvido pelo Dr. Oliver Michel com o objetivo de permitir o usuário a interagir durante a simulação pode modelar, programar, simular e construir

ambientes, veículos subaquáticos, drones voadores, carros, entre outros tipos de robôs em um ambiente simulado. A Figura 8 ilustra a construção e codificação do ambiente gráfico do Webots.

FIGURA 8 – Ambiente gráfico do Webots.



FONTE: Demien, 2020.

Webots utiliza *Open Dynamics Engine* um mecanismo que fornece uma simulação de alguns sistemas físicos como: a dinâmica de corpo rígido e dinâmica de corpo mole, podendo também detectar colisão (DAMIEN, 2020). Suas principais características são:

a) Criação de simulador de multiplataforma.

b) Possibilita a criação de protótipos de forma rápida e de uma vasta variedade de simulações.

c) Salva arquivos em .wtbt baseados em VRML formato que representa cenas tridimensionais.

d) As simulações realizadas no webots podem ser exportadas em forma de animações, filmes, cenas html, entre outras.

e) Disponibiliza a captura de tela, podendo exportar no formato .png, .jpeg ou então gravar simulações no formato MP4 e AVI.

f) Robôs podem ser programados em diversas linguagens de programação, sendo elas Python, C/C++, Java Matlab ou ROS com APIs simples.

2.2.3.3 RoboDK

Ao contrário dos outros simuladores, o *software* RoboDk não é gratuito, ele é voltado para a indústria e possibilita programação *off-line*, criação e utilização de

múltiplos robôs com acesso ilimitado à bibliotecas, oferecendo suporte para mesa giratória e trilhos linear sincronizando até 12 eixos para operações de manufatura.

Usuários com conhecimento mais avançado podem programar o robô simulado usando a API RodoDk, disponível em C++, C#, Python, Matlab e Visual Basic, facilitando a integração de ambientes de simulação 3D e automatização de tarefas repetitivas a projetos do usuário (AUTOMATION, 2021).

2.3 INTELIGÊNCIA ARTIFICIAL (IA)

A Inteligência Artificial (IA) é uma área da ciência da computação que busca desenvolver computadores e sistema inteligentes. Conversar pelo celular, realizar uma busca em um navegador web ou até mesmo realizar a automação de uma fábrica são características da IA com objetivo de simular o pensamento ou ação humana e com isso facilitar operações específicas no dia a dia.

A IA pode ser definida em quatro categorias, propostas por Russell e Norvig (2013, p.25) definidas como: (1) sistemas que pensam como humanos, (2) sistemas que agem como humanos, (3) sistemas que pensam racionalmente e (4) sistemas que agem racionalmente. Em suma, são sistemas que podem pensar, raciocinar e até agir (MEDEIRO, 2018).

2.3.1 Aprendizado de Máquina

Aprendizagem de Máquina é o processo de ensinar sistemas ou computadores a aprender sem serem programados, ou seja, o algoritmo pode aprender sobre seus erros e fazer previsões de dados através da construção de um modelo a partir de *inputs* (entrada de dados) consegue realizar previsões ou decisões e não simplesmente seguir instruções programadas. É um método para realizar predições, utilizado por pesquisadores e cientistas de dados que através de modelos complexos e algoritmos conseguem produzir resultados confiáveis, repetitivos e de decisões.

O Aprendizado de Máquinas é utilizado em diversas tarefas computacionais, como na criação de algoritmos que realizam diferentes aplicações: reconhecimento de escrita, processamento de linguagem natural, diagnósticos médicos, reconhecimento de fala, visão computacional e locomoção de robôs (LUGER, 2013). Estas tarefas são classificadas em três categorias:

a) Aprendizado por reforço: O algoritmo deve desempenhar objetivos recebendo *feedbacks* de acertos e erros, por exemplo, em um jogo de xadrez o algoritmo precisa ganhar a partida. Com isso, através de *feedbacks* de

novas jogadas do oponente o algoritmo consegue aprender e realiza novas movimentações para atingir o objetivo e ganhar a partida.

b) Aprendizado supervisionado: O objetivo apresentar exemplos de entradas e saídas desejadas ao algoritmo para aprender a mapear as entradas de dados para as saídas, por exemplo, o algoritmo recebe imagens rotuladas de cômodos internos de uma casa e seu objetivo é através das imagens avaliar e prever novas imagens que vão ser identificadas e mapeadas para a saída do processo de treinamento.

c) Aprendizado não supervisionado: O objetivo é deixar o algoritmo de aprendizagem descobrir sozinho novos padrões nos dados de entrada, por exemplo, o algoritmo recebe imagens de gatos e cachorros, porém, as imagens não contém nenhum tipo de rotulação ou etiqueta. Com isso o algoritmo precisa encontrar sozinho estruturas nas entradas fornecidas, descobrindo novos padrões para identificar imagens de cachorros e gatos corretamente.

2.3.2 Rede Neural Convolucional (CNN)

Uma Rede Neural Concolucional (CNN) é usada principalmente em reconhecimento de imagens e processamento de vídeo, capaz de atribui diferentes e importantes características para diferenciar aspectos e objetos presentes em imagens, realizar detecção de objetos, reconhecimento e categorização, entre outros.

Diferente de outros algoritmos de classificação de imagens, a CNN não demanda de um nível alto de pre-processamento, em outros algoritmos a aplicação de filtros são feitos de forma manual já a CNN utiliza de filtros convolucionais.

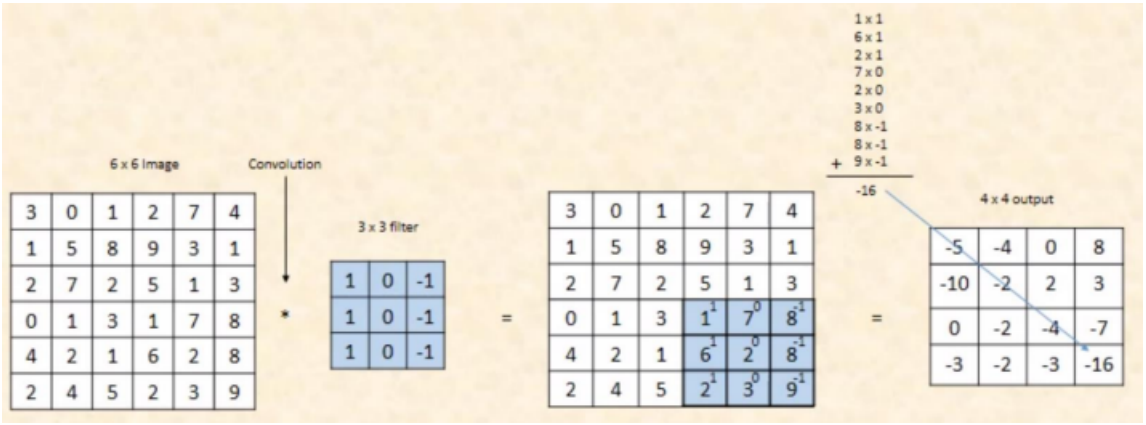
A CNN é composta por varias camadas alternadas de convolução e *pooling* e por uma ou mais camadas totalmente conectadas, cada camada contém diferentes mapas de características que possibilitam a manipulação de imagens com variação de distorção, rotação ou translação nas imagens.

A primeira camada no treinamento da CNN é a camada de redimensionamento que pega por exemplo, todas as imagens de entrada 160x160x3, 160 *pixels* de altura, 160 de largura e 3 camadas de RGB e redimensiona com o tamanho desejado e adequado para testes específicos, para realizar o redimensionamento é utilizado a função *Rescaling*.

A camada de convolução acorre multiplas vezes em diferentes camadas da rede, tem como objetivo através de duas funções produzir uma terceira função. Na CNN esse processo acontece quando a imagem de entrada recebe filtros gerando um mapa de ativação na saída (CUNHA, 2020). A Figura 9 ilustra a aplicação de um filtro de

tamanho 3x3 em uma imagem de dimensões 6x6 gerando um mapa de ativação de saída no tamanho 4x4.

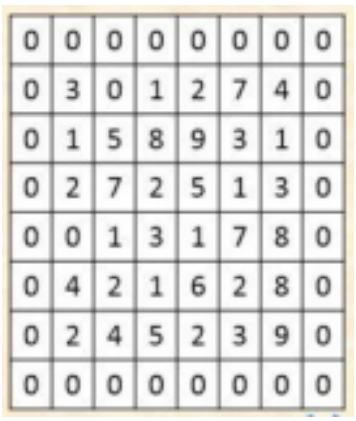
FIGURA 9 – Imagem de entrada 6x6 e filtro 3x3 gerando um mapa de ativação 4x4.



FONTE: RIZWAN, 2018.

No treinamento da CNN a camada de convolução é representada pela função Conv2D, esta função recebe normalmente quatro paramentos, o primeiro parâmetro é a quantidade de filtro convolucionais, o segundo a quantidade de *kernels* (quantidade de vezes que passa por toda a imagem, da esquerda para a direita, de cima para baixo, aplicando o produto da convolução) utilizados, o terceiro parâmetro é o *padding* com o valor *"same"*, que aplica zeros ao redor de toda a imagem. A Figura 10 apresenta a aplicação do *"same"* em uma imagem 6 x 6. E por fim como função de ativação se utiliza o *activation* com o valor *"relu"* que tem como objetivo pegar valores negativos e transforma-los em 0 ou 1, evitando que o algoritmo fique preso em mínimos locais e não consiga evoluir sua aprendizagem.

FIGURA 10 – Aplicação do *same* em uma imagem 6x6.



FONTE: RIZWAN, 2018.

A camada *pooling* é subsequente da camada de convolução com o objetivo a redução de amostragem, onde pega características importantes da matriz de imagem e cria uma matriz menor. Acelerando os cálculos computacionais da rede (CUNHA, 2020). No treinamento da CNN a função MaxPooling2D é utilizada para pegar o mapa

de ativação de saída da camada Conv2D reduzindo em um mapa menor. O objetivo é um pouco diferente da camada de convolução que extrai informações da imagem enquanto a camada *pooling* comprimi essas informações. A Figura 11 exibe a divisão de uma matriz de imagem de dimensões 4x4 em 4 partes, ilustradas nas cores azul, verde, amarela e bege aplicando a função MaxPooling que reduz a matriz original da imagem em uma matriz menor com as características importantes.

FIGURA 11 – utilização do *pooling* em uma imagem 4x4.

FONTE: RIZWAN, 2018.

Após a utilização de múltiplas camadas de convolução e *pooling* é utilizado a camada *flatten*, responsável pelo processo de achatamento dos mapas de ativação. Basicamente a camada *flatten* pega a matriz de imagem (mapas de ativação) e transforma em um único vetor para que posteriormente esses dados possam ser classificados pela camada camada totalmente conectada (CUNHA, 2020). No treinamento da CNN é utilizado a função *Flatten* para realizar o achatamento dos mapas de ativação.

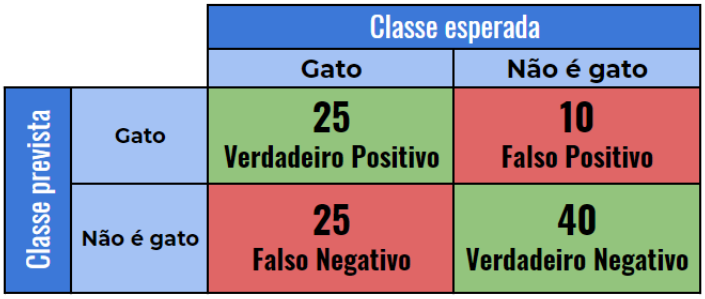
A camada totalmente conectada da CNN tem como objetivo através das extrações de características das camadas de convolução e *pooling* conseguir determinar quais classes mais se correlacionam com determinadas características. Por exemplo, se a rede está classificando uma imagem de um cômodo de uma casa, ela certamente obteve altos valores nos mapas de ativação que representam características de mesa, cadeiras, cama (CUNHA, 2020). No treinamento da CNN a camada totalmente conectada é representada pela função *Dense* composta normalmente por dois parâmetros, o primeiro funciona como os neurônios ou valores de classificação podendo ser definido com o numero de classes distintas. O segundo paramento é a função *activation* com o valor *"relu"*, *sigmoid*, entre outros. A função *sigmoid* recebe qualquer valor real como entrada produzindo valore entre 0 a 1. Quanto maior for a entrada, mais próximo a 1,0 estará o valor de saída, enquanto menor for a entrada, mais próximo a 0,0 será o valor de saída, onde valores próximos a 1,0 são considerados valores positivos e valores próximos a 0,0 valores negativos.

2.3.2.1 Avalição e previsões do modelo

Posteriormente após o treinamento, os diferentes métodos de aprendizagem deverão ser avaliados para que possam ser comparados, para realizar a avaliação algumas métricas específicas são utilizadas para esses problemas de classificação.

A matriz de confusão é uma importante ferramenta que permite a análise de forma rápida junto com o desempenho de cada sistema. Os valores que compõem a matriz são adquiridos através do fornecimento de cada segmento do conjunto de teste ao método de classificação e comparando a sua predição com a classe correta de cada segmento (UTSCH, 2018). A Figura 12 mostra um exemplo da matriz de confusão.

FIGURA 12 – Exemplo de Matriz de Confusão com 2 classes.



FONTE: Pádua, 2020.

Após a comparação os valores são classificados em quatro possíveis opções:

a) Verdadeiro Positivo (VP): segmentos que pertencem a classe positiva e foram classificados como positivos. No caso deste exemplo, são segmentos que contêm gatos e foram classificados corretamente.

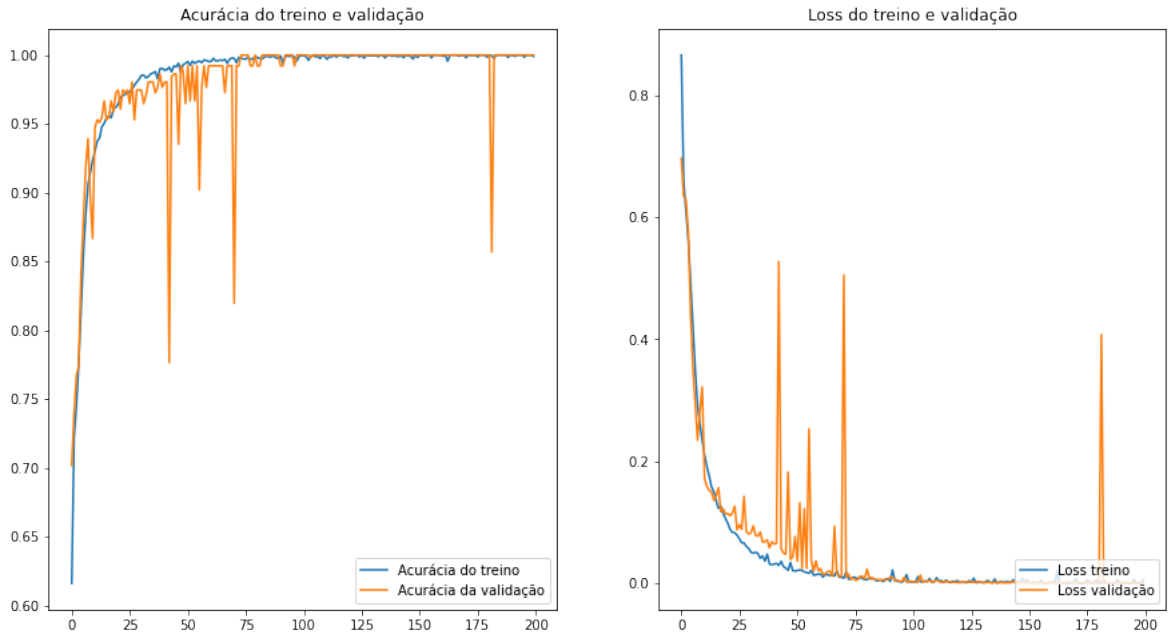
b) Falso Positivo (FP): segmentos que pertencem a classe negativa e foram classificados como positivos.

c) Falso Negativo (FN): segmentos que pertencem a classe positiva e foram classificados como negativos. Nesse exemplo são segmentos que contêm gatos, porém foram erroneamente classificados como negativos.

d) Verdadeiro Negativo (VN): segmentos que pertencem a classe negativa e foram corretamente classificados como negativos.

A métrica acurácia é popularmente utilizada na comparação de modelos de classificação. ela busca avaliar alguns aspectos do modelo como a quantidade de segmentos tanto da classe positiva quanto da negativa que foram classificados corretamente, de modo geral ela é uma boa indicação de como o modelo performou. O Gráfico 1 ilustra um exemplo de gráfico utilizando a acurácia e taxa de *loss*.

GRÁFICO 1 – Exemplo de gráfico utilizando a acurácia como métrica.



FONTE: Os próprios autores.

São utilizados para comparação os dois *datasets*, treino e validação, representados pela cor azul e laranja, os rótulos na horizontal representam o número de épocas percorridas, o rótulo vertical no gráfico à esquerda representa a acurácia que vai de 0.00 á 1.00 durante os testes (UTSCH, 2018).

O rótulo vertical no Gráfico 1 à direita representa a quantidade de *loss* (perda), o *los*s mostra quão mal o modelo está indo no treinamento e validadação, então quanto menor for o *loss*, melhor é o funcionamento do seu modelo. Neste exemplo se iniciou com o valor menor que 0.5 que seria equivalente a menos que 50% de perda.

2.3.3 Problemas comuns em Redes Neurais

Uma forma básica, porém não se pode levar como consideração final é a visualização gráfica das predições do modelo treinado. Esses gráficos podem fornecer indícios de problemas como *overfitting* e *underfitting*.

O *underfitting* é um problema fácil de ser identificado, acorre quando o erro do modelo treinado se comporta de maneira elevada tanto nos dados de treinamento como nos dados de testes.

O problema de *overfitting* por outro lado é mais difícil de ser identificado. Ocorre quando comparamos a performance do modelo de treinamento com os testes de predições.

2.3.3.1 *Overfitting*

O problema de *overfitting* ocorre quando, nos dados de treino, o modelo tem um ótimo desempenho, porém ao utilizar os dados de teste o resultado é ruim. Neste caso, o modelo acaba decorando o que deveria ser feito, e ao receber as informações dos dados de teste, o modelo aplica o que foi decorado e não generaliza bem para novos dados afetando diretamente no desempenho. Com isso, o modelo treinado não tem capacidade de generalizar para dados desconhecidos.

O *overfitting* tem algumas causas principais, que podem direcionar a solução do problema. Algoritmos muito complexos, poucos dados de entrada e ruídos nos dados de treinamento. Os ruídos em imagens são distorções e erros que podem causar problemas no treinamento da Rede Neural, em imagens simuladas o problema com ruídos é muito menor do que na utilização de imagens de ambientes reais, que normamente contém diferentes iluminações, sombras e distorções (AMAZON, 2016).

2.3.3.2 *Underfitting*

O *underfitting* acorre quando nosso modelo de treinamento não conseguiu aprender com os dados de entrada, demonstrando erros muito elevados tanto nos dados de treinamento quando nos dados de teste.

O *underfitting* tem algumas causas principais: Algoritmo inadequado, poucos dados de entrada, caracteristicas insuficientes nos dados, muitas restrições no modelo (AMAZON, 2016).

3 REVISÃO DA LITERATURA

Esse capítulo apresenta os trabalhos relacionados, com a linha de pesquisa desse trabalho.

3.1 MAPEAMENTO E LOCALIZAÇÃO SIMULTÂNEA DE ROBÔS MÓVEIS USANDO DP-SLAM E UM ÚNICO MEDIDOR LASER POR VARREDURA

Herrera (2011) apresenta o DP-SLAM (uma representação do mapa em forma de grade de ocupação), o que é um grande problema nas pesquisas da Robótica, onde o robô tem dificuldade em construir mapas e ao mesmo tempo conseguir manter sua localização em determinado ambiente sem um conhecimento prévio do mesmo. Foi desenvolvido, em MatLab um simulador para coletar dados 3D de um ambiente estruturado, juntamente com um algoritmo de verificação de correspondência, que obtém o deslocamento sem odometria do robô. Como o LRF (um medidor laser de distância por varredura) realiza uma varredura de dados 2D foram implementados algoritmos de verificação de correspondência, em que a implementação de uma plataforma rotativa no robô móvel foi capaz de fazer varreduras horizontal e verticais, processando os dados adquiridos para obter mapas 2D e 3D. Desta forma, a tese conseguiu contribuir com uma aplicação do DP-SLAM sem a utilização de informações da odometria, utilizando apenas um único LRF (HERRERA, 2011).

3.2 MAPEAMENTO DE AMBIENTES EXTERNOS UTILIZANDO ROBÔS MÓVEIS

Yukinobu (2010) apresenta o mapeamento de ambientes externos, através de um robô móvel, utilizando Redes Neurais Artificiais e sensor a laser. O trabalho descreve o desenvolvimento de um sistema de plataforma robótica equipada com sensor a laser direcionado para o solo, além dos dois algoritmos desenvolvidos, um utilizado para realizar o mapeamento de detalhes finos dos terrenos e outro para identificar regiões próprias e impróprias para que o robô possa trafegar. Os algoritmos foram treinados e validados com dados reais extraídos de quatro terrenos para testes, ilustrados na Figura 13. No cenário (a) a via contém grama nas duas laterais, o (b) é composto com grama apenas na lateral esquerda, o cenário (c) é uma via com rampa e contém grama nas duas laterais, por fim o cenário (d) é uma via de paralelepípedo com grama nas duas laterais. Esses dados foram alimentaram o algoritmo de mapeamento 3D de terrenos e para cada cenário foi gerado o mapa de nuvem de pontos no formato VRML (é um padrão de formato de arquivo para realidade virtual), apresentados na Figura 14.

FIGURA 13 – Ambientes utilizados para os experimentos.

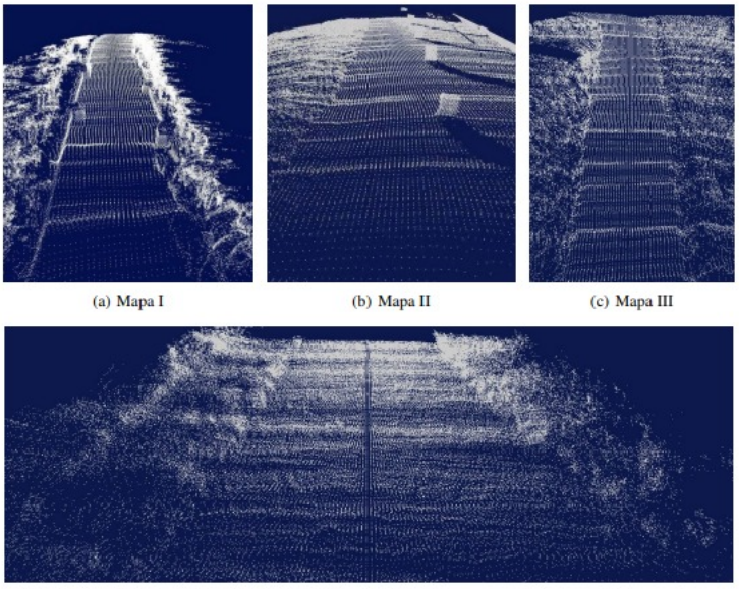
(a) Cenário I (b) Cenário II (c) Cenário III

(d) Cenário IV

FONTE: Yukinobu 2010.

Os Cenários I à IV mostram mapas de navegabilidade que representam a altura absoluta e a altura relativa. Essas alturas são representadas por meio de uma escala de cores, a cor vermelha para valores positivos, azul valores negativos e verde para valores próximos a zero. Com a análise dos mapas com altura absoluta, o autor identificou que a altura do relevo do terreno é muito compatível com o cenário real. No entanto, não se pode utilizar apenas essa referência para classificar um terreno, visto que não é detectado declives e aclives que seriam considerados obstáculos.

FIGURA 14 – Mapas 3D obtidos pelo algoritmo de mapeamento 3D.



(a) Mapa I (b) Mapa II (c) Mapa III

FONTE: Yukinobu 2010.

Os resultados mostraram que mapas de navegabilidade, classificados com RNAs, são eficientes para serem utilizados em navegação com robôs móveis em terrenos com alto nível de desestruturação (YUKINOBU, 2010).

3.3 RECONHECIMENTO DE OBJETOS NO DESENVOLVIMENTO DE UM SISTEMA DE NAVEGAÇÃO INTELIGENTE PARA ROBÔS MÓVEIS

Reinaldo (2015) apresenta o desenvolvimento de um sistema de navegação através de uma plataforma robótica móvel, que permite o robô mude seu comportamento através de imagens capturadas por um sensor RGB-D, processadas por um algoritmo conhecido como SURF, a fim de obter informações dos objetos previamente cadastrados. A Figura 15 ilustra a sala de laboratório, com 6,0 metros de comprimento e 3,90 metros de largura, composto por cadeiras brancas, bancadas, armários, entre outros objetos pequenos.

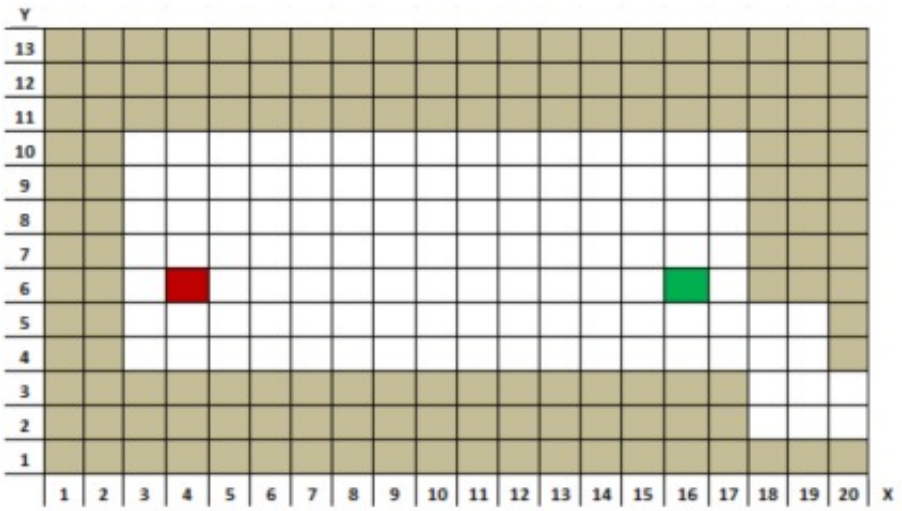
FIGURA 15 – Ambiente utilizado no experimento.

FONTE: Reinaldo, 2015.

O ambiente foi previamente mapeado com uma fita métrica, utilizando suas dimensões para a criação de um mapa com grade de ocupação, apresentado na Figura 16. As células de cor branca representam espaços vazios, podendo ser percorridos pelo robô e as células de cor marrom indicam espaços ocupados por objetos. Em todos os experimentos o robô tem por finalidade sair do ponto x = 1,05m e y = 1,65m, representado pela célula de cor vermelha, e chegar no ponto x = 4,65m e y = 1,65m, representado pela célula de cor verde.

Para percorrer da célula vermelha até a célula verde, diversos objetos foram colocados no caminho do robô. Com isso, foi observado que para concluir o percurso, após o robô identificar o objeto ele deve fazer um replanejamento do caminho e movimentação de rotação. No entanto, toda vez que o robô necessita de um novo

FIGURA 16 – Mapa com grade de ocupação.



FONTE: Reinaldo, 2015.

planejamento, o mesmo precisa parar, replanejar e só depois continuar o seu percurso. Decorrente de todos os testes, o autor verificou que o robô conseguiu remanejar sua trajetória de forma satisfatória, contornando todas as células preenchidas (células marrom) (REINALDO, 2015). A Figura 17 ilustra a plataforma robótica utilizada em todo o experimento.

FIGURA 17 – Plataforma robótica utilizada no experimento.

FONTE: Jurasildo Oliveira Reinaldo, 2015.

3.4 NAVEGAÇÃO ROBÓTICA RELACIONAL BASEADA EM EEB CONSIDERANDO INCERTEZA NA PERCEPÇÃO

Toro (2014) aborda os problemas encontrados nas percepções incertas do ambiente de navegação robótica relacional, decorrente do fato de sensores não proporcionar informações suficientes para identificação completa do ambiente de teste. O trabalho apresenta a arquitetura WRRA (Arquitetura robótica relacional baseada na Web) para navegação de robôs com o sistema KnowRob, utilizando bases de conhecimento de sentenças lógicas e aprendizagem por reforço, mostrando como , utilizar na prática recursos da *Web* semântica para lidar com as incertezas.

A arquitetura WRRA, que opera com informações semânticas, levando a navegação qualitativa, foi implementada em um ambiente simulado e só depois foi realizado em um robô real. Os experimentos realizados nesse ambiente mostraram que a navegação interna consegue se beneficiar de tal estrutura. O autor observou também que, combinando informações semânticas com os dados dos sensores dos robôs juntamente com o banco de dados, o robô conseguiu resolver o problema de observabilidade parcial dentro de um ambiente interno, uma vez que essa combinação permite trabalhar em um nível superior de abstração, podendo assim, inferir informações que o sensor do robô sozinho não consegue realizar (TORO, 2014).

3.5 UMA APLICAÇÃO DE NAVEGAÇÃO ROBÓTICA AUTÔNOMA ATRAVÉS DE VISÃO COMPUTACIONAL ESTÉREO

Espinosa (2010) descreve uma técnica de navegação autônoma utilizando imagens estereoscópicas de câmeras para estimar o movimento de um robô em um ambiente desconhecido. A estimação do movimento bidimensional do robô é calculada aproveitando a relação geométrica epipolar entre dois ou mais pontos em pares de imagens.

Os estudos realizados permitem guiar e situar um robô móvel com imagens provenientes de duas câmeras. Além dos métodos utilizados na navegação autônoma, foi permitido extrair informações com relação à distância dos objetos, informação útil que pode ser aplicada em estratégias de controle. A Figura 18 ilustra a base robótica utilizada no experimento (ESPINOSA, 2010).

FIGURA 18 – Robô Pioneer 3 utilizado para validar os procedimentos de navegação..

FONTE: Espinosa, 2010.

3.6 QUADRO DE COMPARAÇÕES DOS TRABALHOS RELACIONADOS

Nestes trabalhos foram identificados algumas *frameworks*, linguagens de programação, visualizadores e recursos utilizados. O Quadro 1 apresenta o comparativo de ferramentas utilizadas nos trabalhos relacionados:

QUADRO 1 – Comparativo dos trabalhos relacionados.

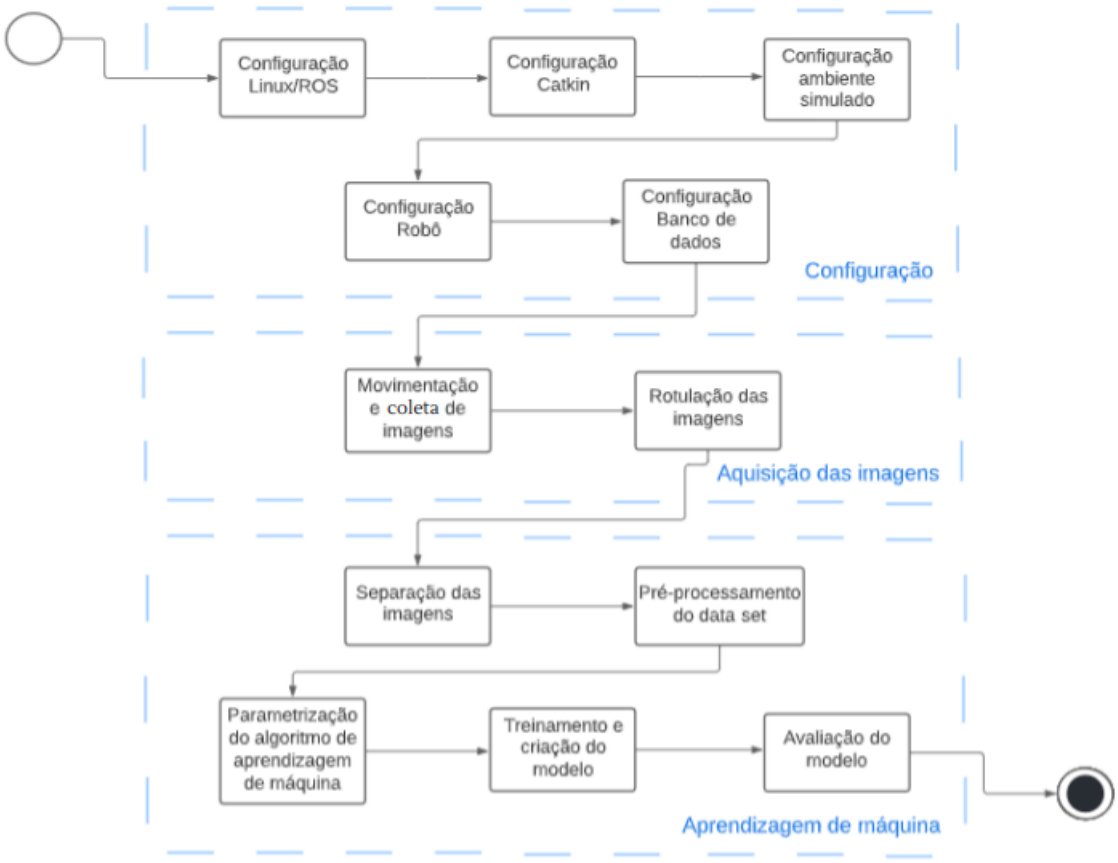
Trabalhos	**Linguagem**	***Framework***	**Visualizador**	**Recursos**
Trabalho 1	Visual C++, VRML	MatLab	2D, 3D	Ambiente virtual
Trabalho 2	Visual C++	MatLab	2D, 3D,	Simulção no MatLab
Trabalho 3	-	OpenGL Robios	3D	Desvio de Obstáculos
Trabalho 4	Prolog, Python	WRRA	2D, 3D	Simulação na WEB
Trabalho 5	-	MatLab	3D	Ambiente virtual

FONTE: Os próprios autores.

4 METODOLOGIA

Para o desenvolvimento deste trabalho de pesquisa realizado na área de robótica, algumas estapas foram realizadas para obter resultados satisfatorios na utilização de um modelo de Rede Neural Artificial. A Figura 19 apresenta o processo da metodologia proposta.

FIGURA 19 – Fluxograma da metodologia.



4.1 CONFIGURAÇÃO

Nesta seção são detalhados todos os processos de configuração do ambiente: configuração das distribuições utilizadas, simulações de ambiente, robô para percorrer e capturar imagens dos ambientes de forma manual, pacotes, repositórios e o armazenamento de imagens.

4.1.1 Configuração do Linux e ROS

Para realização dos testes e a construção da aplicação foi configurado o sistema operacional Linux com a distribuição Ubuntu, versão 20.01.1 LTS (suporte de longo prazo) Focal Fossa. Juntamente com o ROS e a distribuição *Noetic* voltado principalmente para a versão Ubuntu 20.04 (Focal). Com isso foi instalado o *Workspace Catkin*,

para poder trabalhar com os pacotes e fontes no ROS, o robô utilizado é o turtlebot3 *Waffle* PI, pois o mesmo é composto por câmeras que foram configuradas para realizar a coleta das imagens.

4.1.2 Configuração *Workspace Catkin*

Após a configuração das distribuições, foi então criado um pacote no *Workspace Catkin* titulado como *"my_pkg"*, por padrão é criado um arquivo .xml, que funciona basicamente como endereçamento, com isso as repositórios *world*, *launch* e *models* foram criados.

a) World: Contém um arquivo *"empty_world.world"*. Este arquivo armazena o mundo (ambiente) e tudo o que ele contém, como paredes, móveis, portas, entre outros. Os objetos são abstraídos do arquivo *"model.sdf'* localizado no repositório *models*.

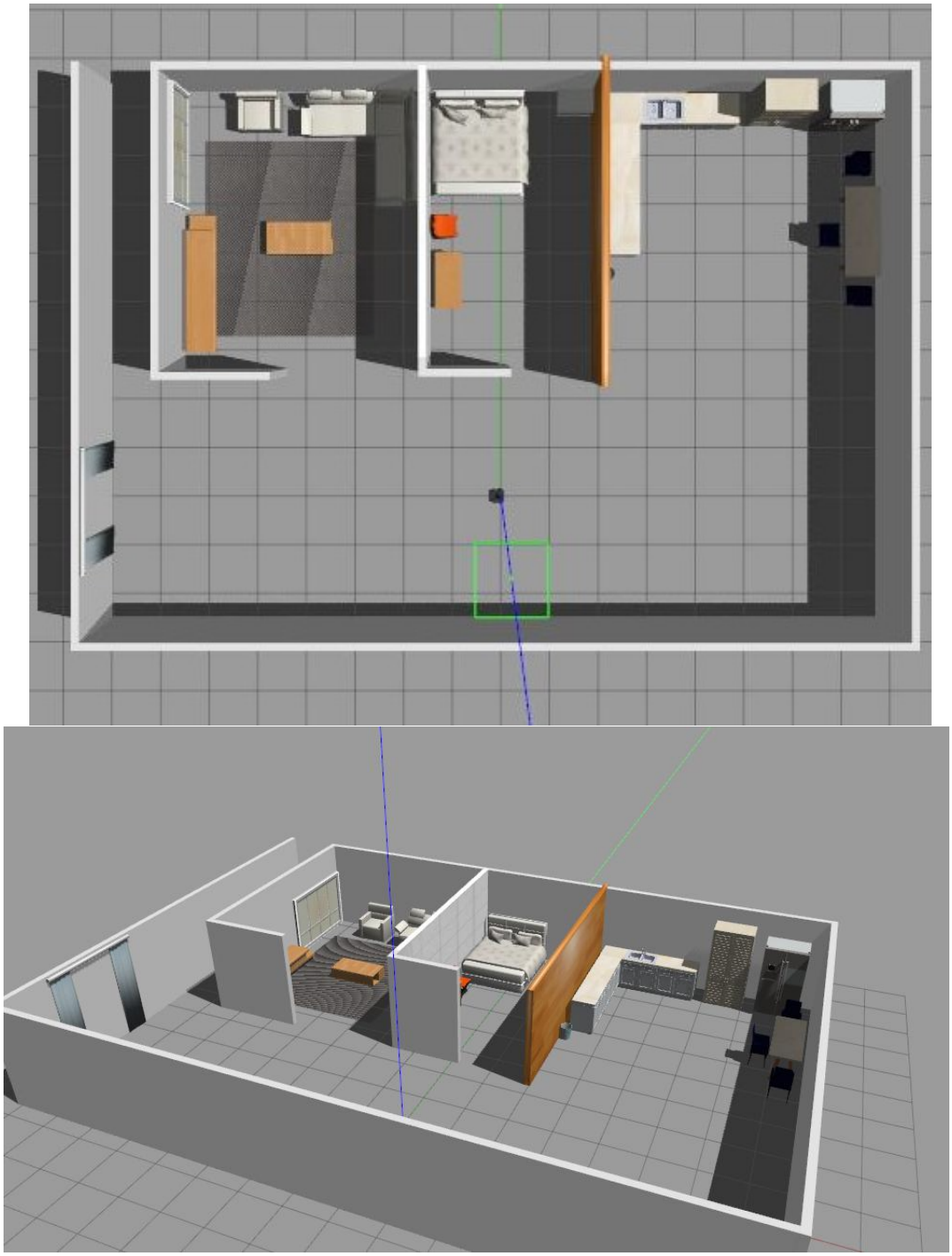
b) Launch: No repositório *launch* o arquivo *"my_world.launch"* é o arquivo que indica onde se encontra o ambiente e os parâmetros do robô utilizado na aplicação.

c) Models: Armazena o arquivo *"model.sdf'* que contém toda a parte estrutural do ambiente, como os objetos paredes e os componentes do ambiente utilizado para os testes no Gazebo.

4.1.3 Configuração do ambiente simulado

Na construção do ambiente simulado foi utilizado o Gazebo versão 11.9.0. Três ambientes foram construídos compostos por objetos e paredes para dividir o ambiente em cômodos que não possuem janelas e portas para facilitar a movimentação do robô. A Figura 20 ilustra o ambiente como um todo, no qual é possível identificar os ambientes separados por cômodos. Os ambientes são: cozinha, quarto e sala.

FIGURA 20 – Ambiente simulado Gazebo.



FONTE: Os próprios autores.

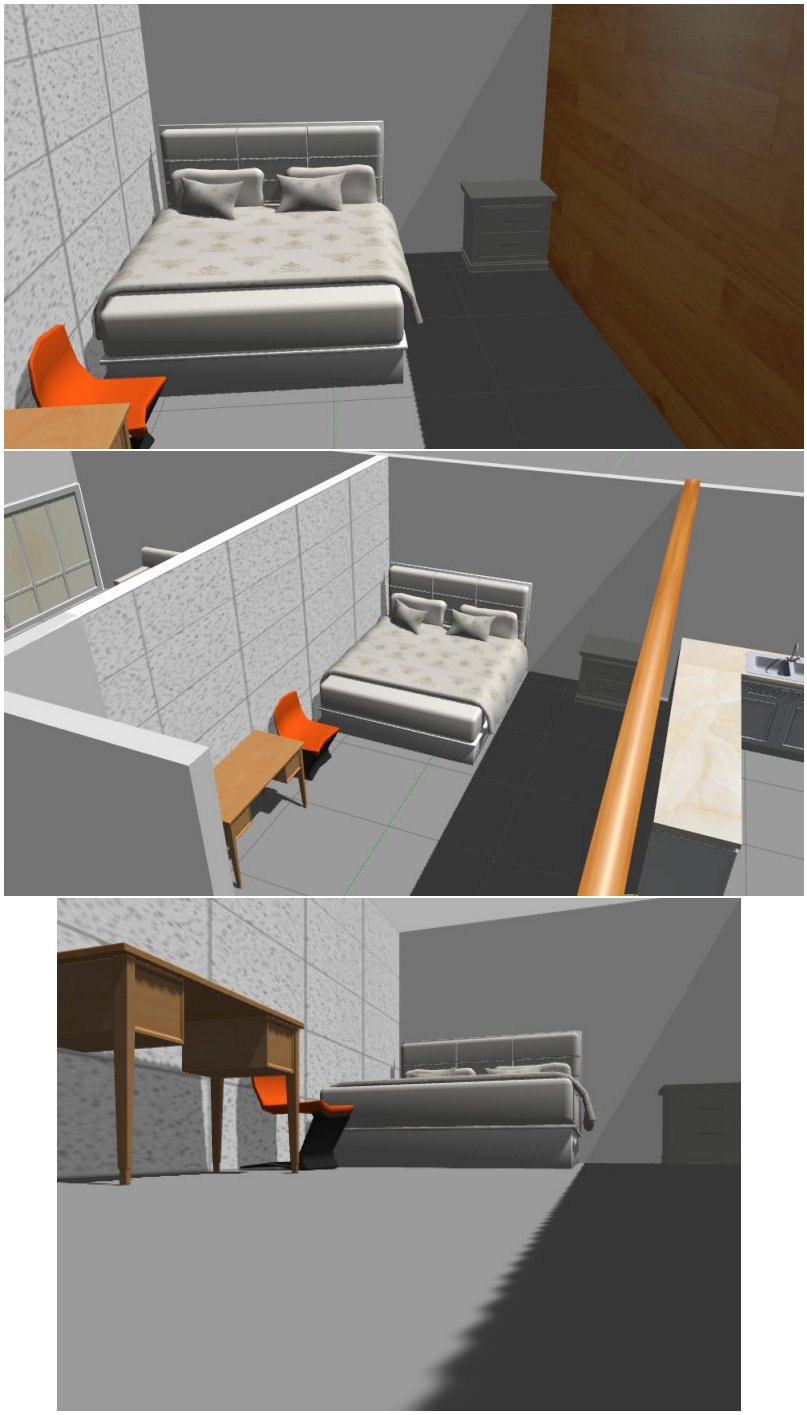
A cozinha é constituída por uma papel de parede amarelo que serve apenas para distinguir um ambiente do outro e por objetos como: mesa, cadeiras, geladeira, lixeira e armários. A Figura 21 mostra o cômodo da cozinha construído no ambiente simulado do Gazebo.

FIGURA 21 – Ambiente simulado Cozinha.

FONTE: Os próprios autores.

O Quarto contém um papel de parede branco com detalhes e um outro papel de parede amarelo e os objetos são: mesa de cabeceira, cama, cadeira e escrivaninha. A Figura 22 apresenta o cômodo do quarto construído no ambiente simulado do Gazebo.

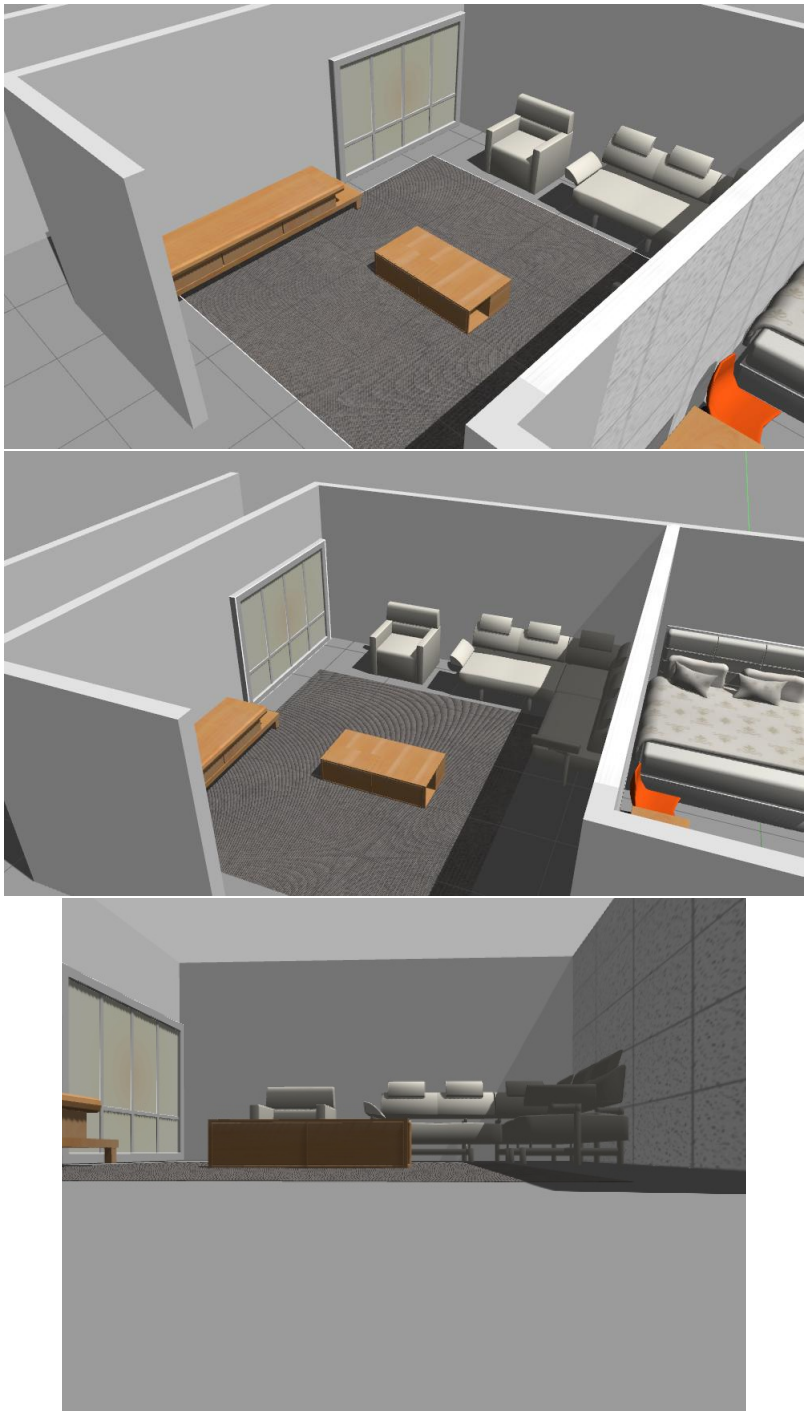
FIGURA 22 – Ambiente simulado Quarto.



FONTE: Os próprios autores.

A Sala é o único cômodo que não contém papel de parede, porém contém um tapete para diferenciar dos outros ambientes e é composta por objetos como: mesa, sofá e armário. A Figura 23 ilustra o cômodo da sala construído no ambiente simulado do Gazebo.

FIGURA 23 – Ambiente simulado Sala.



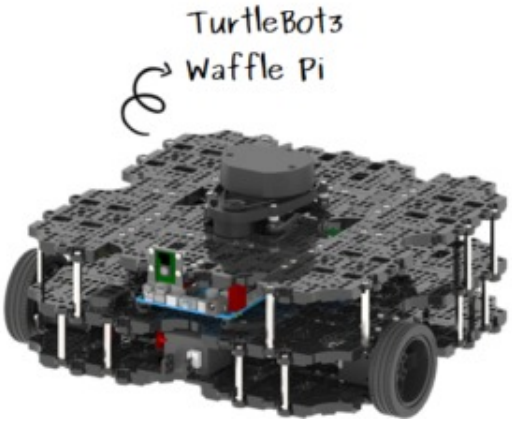
FONTE: Os próprios autores.

4.1.4 Configuração do robô

O robô utilizado é o Turtlebot3 *Waffle* PI, com isso o robô percorreu todo o ambiente simulado coletando imagens através de uma câmera que foram adicionadas na plataforma do *GitHub*. Para exportar o robô, utiliza-se como parâmetro o *"TURTLEBOT3_MODEL=waffle_pi"* que se encontra no repositório *Launch*, basicamente ele

exporta todas as configurações e componentes do robô para o ambiente simulado que foi criado, o modelo exportado é representado na Figura 24.

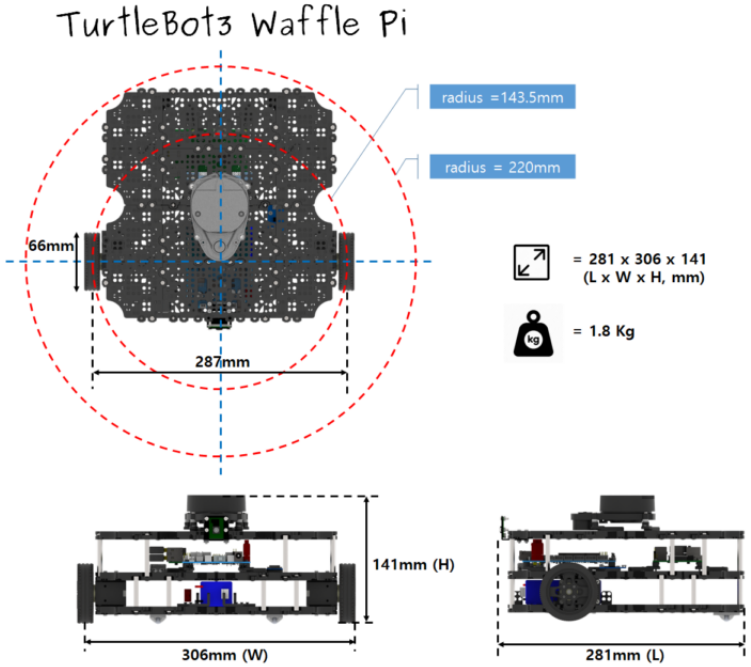
FIGURA 24 – Robô Tutlebot3 *Waffle* Pi.



FONTE: Robotis, 2022.

O Tublebot3 *Waffle* Pi é um robô real que também foi implementado para o uso em ambientes simulados, ele utiliza dos mesmos componentes para operação, como câmeras, rodas, Raspberry Pi 3 (uma placa que simula um mini computador), OpenCR 1.0 (uma placa que fornece *hardware* e *software*), *waffle-plate* (pedaços da estrutura do robô com formatos de *waffle* dando origem ao nome do modelo), entre outros. A Figura 25 descreve as dimensões do robô.

FIGURA 25 – Dimensões do Robô tutlebot3 Wiffle Pi.



FONTE: Robotis, 2022.

4.1.5 Configuração do banco de dados

Para armazenar todas as imagens coletadas dos cômodos foi utilizado o *GitHub*, uma plataforma de hospedagem de código-fonte e arquivos com controle de versão. Foram criados três repositórios: *Train*, *Validation* e *Unknown*. Estes repositórios servirão como base de dados para o treinamento da Rede Neural Convolucional ajudando em diferentes tarefas como: validação, treinamento e avaliação, onde cada um é composto por mais três repositórios que servem para organizar e separar as imagens dos respectivos cômodos: quarto, cozinha e sala.

a) *Train*: É composto por mais três repositórios: cozinha, quanto e sala. O objetivo desse repositório é fornecer a rede neural um conjunto de imagens para treinamento, ou seja, a rede neural irá se basear em 2.500 imagens de cada cômodo para aperfeiçoar seu treinamento.

b) *Validation*: Os três repositórios que o compõem, foram armazenadas 850 imagens, que irão servir para a rede como uma forma de validar as imagens utilizadas para o treinamento.

c) *Unknown*: Tem como objetivo, armazenar imagens desconhecidas da rede neural para após o treinamento da rede com os *datasets* de treinamento e validação consiga realizar uma nova avaliação da precisão em que a rede neural atingiu. Em cada repositório que o compõe foram armazenas 200 imagens, imagens que a rede neural desconhece.

4.2 APLICAÇÃO

Nesta seção toda a aplicação é descrita de forma detalhada, etapas como a realização da movimentação do robô dentro do ambiente, captura e rotulação das imagens, criação do banco de dados e implementação do modelo de CNN.

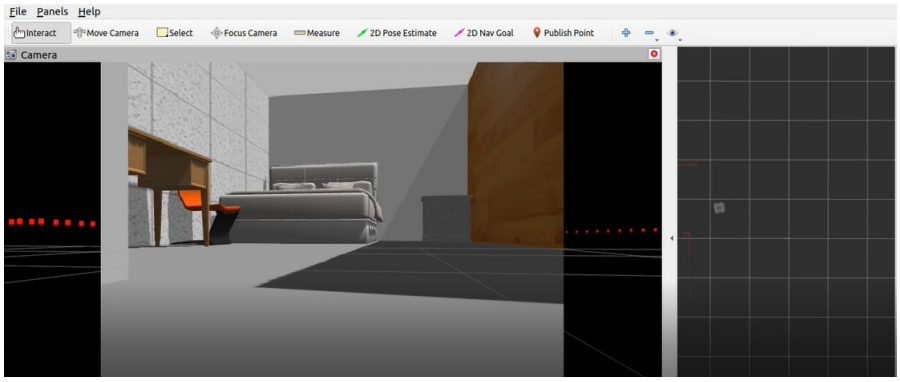
4.2.1 Movimentação do robô

Para a movimentação do robô no ambiente apresentado na Figura 20, foi utilizado o pacote *"turtlebot3_teleop"* com o arquivo *turtlebot3_teleop_key.launch*. Isto possibilitou utilizar as teclas "w", "a", "s", "d"e "x" do teclado para realizar a movimentação do robô dentro do ambiente.

O pacote *"turtlebot3_gazebo"* contém o arquivo *"turtlebot3_gazebo_rviz.launch"* basicamente é um controlador 3D do robô, que possibilita capturar imagens como por exemplo: a Figura 26 exibe a imagem retirada do cômodo quarto com o pacote do RVIZ (Ferramenta de visualização 3D para ROS). Com isso 2.500 imagens de cada

ambiente foram capturadas e separadas em repositórios para posteriormente passar por processamento de imagem.

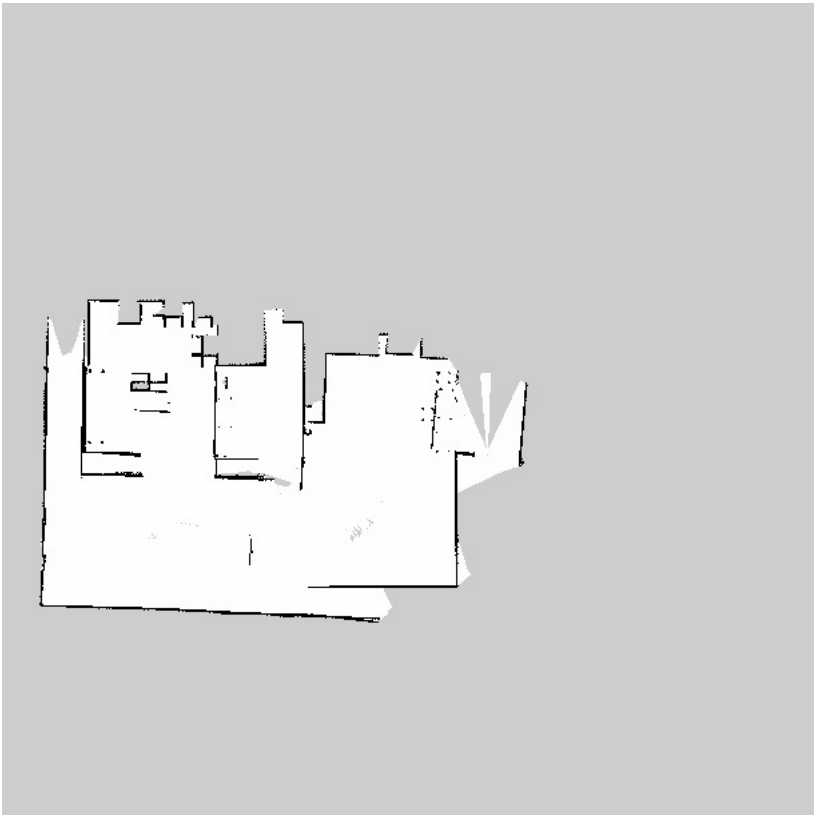
FIGURA 26 – Imagem retirada do ambiente simulado com o controlador RVIZ 3D.



FONTE: Os próprios autores.

Outro pacote utilizado para percorrer o ambiente com o robô foi o *"turtlebot3_navigation"*, que possibilitou o mapeamento de todo o ambiente, de maneira que o robô, com o uso do mouse, consegue percorrer o ambiente de forma mais dinâmica facilitando bastante a coleta das imagens, pois desenhando a trajetoria do robô com o mouse, o mesmo percorre a linha desenhada. Esse mapeamento é representado na Figura 27.

FIGURA 27 – Mapeamento do ambiente simulado.



FONTE: Os próprios autores.

4.2.2 Coleta e rotulação das imagens

Ao percorrer o ambiente simulado com robô Turtlebot3 Waffle PI foram coletadas manualmente várias imagens dos cômodos. Contudo, essas imagens passaram por um processo de rotulação que consiste em fornecer informações sobre o conteúdo das imagens, essas informações são desde os dados da imagem ou até mesmo a atribuição de uma descrição para a imagem e foram realizadas de forma manual.

Para realizar a coleta das imagens foi criado o *"turtlebot_photos"* contendo mais três repositórios: quarto, sala e cozinha. E foram armazenadas separadas por cômodos, também foi utilizado o pacote *"image_view"*, de maneira que possibilitou a captura de imagens do ambiente de forma manual, o comando *"rosrun"*, passando o parâmetro para acessar a câmera do robô e através disso gerar uma imagem com o nome e formato que podem ser especificados, o comando resultou na imagem apresentada na Figura 26.

4.2.3 Ambiente de desenvolvimento

Para todos os testes e aplicações foi utilizado o Google Colab, um serviço gratuito hospedado na nuvem pelo próprio Google, muito adequado para aprendizagem de máquina, tem como base a linguagem de programação Python. O Google Colab permite a utilização de células que podem conter códigos juntamente com imagens. As células normalmente são utilizadas para a criação de funções, aplicações e testes com imagens, gerando como saída: textos, imagens, gráficos e muitas outras funcionalidades. O Google Colab é um ambiente colaborativo, facilitando muito seu uso em equipe, disponibiliza o uso de recursos de unidade de processamento gráfico (GPUs) podendo utilizar a do próprio Colab ou configurar o da máquina local.

Para a construção da Rede Neural Artificial, carregar os conjuntos de dados (*datasets*), criação de funções e aplicação de técnicas de processamento de imagens, foi utilizado o Python, uma linguagem de programação *Open-source* de alto nível orientado a objetos e a principal linguagem de programação no uso de IA. O uso dessa linguagem é essencial pois com poucas linhas de código possibilita na criação de funções com uma boa complexidade, entre outras vantagens como: multiplataforma e múltipla possibilidades de desenvolvimento. Juntamente com o Python algumas bibliotecas são essenciais na manipulação de dados e no uso de Redes Neurais Artificialis.

a) TensorFlow: Biblioteca de código aberto para aprendizado de máquina aplicável a uma ampla variedade de tarefas. Um sistema para criação e treinamento de Redes Neurais para detectar e decifrar padrões e correlações.

b) Keras: O Keras é uma biblioteca de Rede Neural Artificial de código aberto escrita em Python. Ele é capaz de rodar em cima de TensorFlow, projetado para permitir experimentação rápida com redes neurais profundas, ele se concentra em ser fácil de usar, modular e extensível.

c) Matplotlib: Uma biblioteca de software para criação de gráficos e visualizações de dados em geral.

d) Pandas: Uma biblioteca de *software* criada para a linguagem Python para manipulação e análise de dados. Em particular, oferece estruturas e operações para manipular tabelas numéricas e séries temporais.

4.2.4 Algoritmos de aprendizagem de máquina

Uma Rede Neural Convolucional (CNN) é construída realizando algumas etapas, todas as etapas têm de grande importância para a consolidação do treinamento. A primeira etapa é a configuração do *dataset*, depois é preciso fazer um pré-processamento desse *dataset*, para ser criado funções auxiliares que ajudam ainda mais no treinamento. Com todas essas etapas concluídas pode-se criar um modelo da CNN para realizar o treinamento, com o modelo treinado são criado funções e gráficos para avaliar o desempenho do modelo.

4.2.4.1 Criação dos *Datasets*

Primeiramente precisa-se criar um *dataset* que tem como base as imagens do ambiente simulado, com essas imagens foram criados três *datasets*. O primeiro nomeado como *Train* utilizado para treinar a CNN, o segundo *Validation* responsável por validar o treinamento, e por fim o *Unknown* com imagens desconhecidas da CNN, para obter o desempenho do treinamento realizado.

4.2.4.2 Pré-processamento

Com o *dataset* já definido e configurado começa o pré-processamento, pois se as imagens utilizadas para treinamento da CNN não forem devidamente tratadas poderão trazer problemas futuros no treinamento. Por isso algumas variáveis globais foram definidas, essas são responsáveis por armazenar características distintas.

a) image_size: É composta por duas variáveis que definem as dimensões das imagens utilizadas, a *"image_width"* define a largura e a *"image_height"* a altura, ambas foram atribuído o valor 160 transformando todas as imagens do *dataset* em dimensões 160x160.

b) image_color_channel: Nesta variável foi atribuído o valor 3, definindo que as imagens utilizadas são compostas por três canais de cores denominadas RGB, podendo ir de (0,0,0) cor preta até (255,255,255) cor branca. RGB é um sistema de cores que utiliza tons de vermelho, verde e azul, criando outras tonalidades de cores.

c) image_shape: Variável que armazena o formato da imagem, a *"image_shape"* recebe os valores da *"image_size"* mais o valor da *"image_color_channel"*. Uma das variáveis mais importantes no processo de treinamento, é a entrada de dados que rede neural irá aplicar todas os filtros convolucionais.

d) batch_size: Representa a quantidade de imagens que serão usadas por vez do *dataset*, definida com o valor 200 para a variável *"batch_size"*. *Batch size* é um termo usado em aprendizado de máquina e refere-se ao número de exemplos de treinamento usados em uma iteração de uma época.

e) *epochs*: É o número de vezes que a rede neural percorre todo o *dataset* definido. Assim, cada vez que o algoritmo vê todas as amostras do *dataset*, uma época foi concluída. Após muitas tentativas foi atribuído o valor 200 para a variável *epochs* que se comportou de maneira satisfatória.

f) learning_rate: A taxa de aprendizagem é responsável por definir a velocidade em que o algoritmo irá aprender, por exemplo: ao atribuir um valor muito alto (2.0 ou 3.0), o algoritmo irá aprender rápido mas de uma forma não tão eficaz, já com valores mais baixos (0,001 ou 0,0001), o algoritmo se tornará muito mais eficaz, porém o tempo para terminar a aprendizagem será muito maior, por isso a variável learning_rate foi definica com o valor 0,00001.

4.2.4.3 Criação do modelo para treinamento da CNN

Uma Rede Neural Convolucional é composta por várias camadas, cada camada é responsável por extrair determinadas informações dos dados de entrada. No caso da entrada tem-se uma imagem, representada por uma matriz de 160x160 cujo valor não é um número e sim um vetor de tamanho três representando altura, largura e os três canais de cores. Essas informações fluem através de cada camada da rede, com a saída anterior fornecendo a entrada para a camada seguinte da rede. Nesse processo se utiliza o *Kernel* que funciona como um filtro, passando pela imagem e gerando uma resposta baseada em uma multiplicação simples, elemento á elemento e depois somando tudo. O `Código 1` representa a criação do primeiro modelo de treinamento e cada linha representa uma camada.

CÓDIGO 1 – Primeiro modelo de treinamento da CNN criado no Google Colab.

```
1   inputs = keras.Input(shape=image_shape)
2   x = tf.keras.layers.Rescaling(1./255)(x)
3   x = tf.keras.layers.Conv2D(32, 3, padding = 'same', activation =
    'relu')(x)
4   x = tf.keras.layers.MaxPooling2D()(x)
5   x = tf.keras.layers.Conv2D(64, 3, padding = 'same', activation =
    'relu')(x)
6   x = tf.keras.layers.MaxPooling2D()(x)
7   x = tf.keras.layers.Conv2D(128, 3, padding = 'same', activation
    = 'relu')(x)
8   x = tf.keras.layers.MaxPooling2D()(x)
9   x = tf.keras.layers.Flatten()(x)
10  x = tf.keras.layers.Dense(128, activation = 'relu')(x)
11  output = tf.keras.layers.Dense(3, activation = 'sigmoid')(x)
12
```

FONTE: Os próprios autores.

a) A 1° camada é a entrada dos dados, é ela que irá passar o formato do *dataset* no modelo, para isso é utilizado a variável *inputs*, que armazena o valor de entrada para posteriormente ser utilizada.

b) A 2° camada serve para realizar o redimensionamento do *dataset*, para realizar o redimensionamento se utiliza a função *Rescaling*.

c) A camada 3 representa uma camada convolucional 2D, basicamente está é responsável pela aplicação de filtros convolucionais para que durante o treinamento a rede aprenda cada vez mais filtros, basicamente ela cria novas imagens com cada filtro, tornando-a cada vez mais sólida. Na função Conv2D utiliza-se quatro parâmetros, o primeiro é a quantidade de filtros que serão utilizados, no caso foi utilizado 32 filtros, o segundo parâmetro representa o tamanho do *kernel*, 3 *kernels* foram atribuídos no modelo e como saída é construído um mapa de ativação. O terceiro parâmetro é o *padding*, ao atribuir o valor "same", estamos adicionando uma borda de números zeros ao redor da imagem. Já o último parâmetro *activation* foi atribuído o valor "relu", uma função de ativação que se realiza na saída da camada, zerando qualquer valor negativo e assim permanecendo com valores positivos.

d) Na 4° camada o MaxPooling2D é uma função subsequente da camada Conv2D, responsável por consolidar o mapa de ativação da camada Conv2D, reduzindo a resolução da entrada ao longo de suas dimensões espaciais (altura e largura).

e) As camadas Conv2D iniciais mais próximas das imagens de entrada aprendem menos filtros do que as mais profundas da rede, camadas próximas a previsão de saída, tornando assim a necessidade no aumento de filtros nas camadas subsequente, por isso o valor dos filtros do Conv2D na quinta camada foram aumentados para 64.

f) Na camada 6 pode-se observar que a função MaxPooling2D é utilizada para consolidar o mapa de ativação da camada Conv2D.

g) Na 7° camada foi atribuído mais 128 filtros a função Conv2D, esse processo é utilizado porque essas imagens resultantes podem conter elementos que facilitam a identificação da classe alvo para a rede, por isso cada camada seguinte do Conv2D é atribuído valores maiores de filtros.

h) Na 8° camada novamente a função MaxPooling2D é utilizada para consolidar o mapa de ativação da camada Conv2D.

i) A 9° contém a função *Flatten*, normalmente é utilizada para dividir a CNN em duas partes: extração de características e Rede Neural tradicional. Basicamente transforma a matriz da imagem em um vetor. Esta etapa é um preparo para entrar na camada principal da Rede Neural totalmente conectada.

j) Na 10° camada é implementada a rede neural totalmente conectada, onde deve-se informar a dimensão da saída e a função de ativação a ser utilizada. No contexto de reconhecimento de imagens, é comum se projetar esta camada densa com a função de ativação "relu". Por isso foi utilizado a função *Dense* com o valor 128 no primeiro parâmetro.

k) E por fim a 11° camada, é a saída do modelo onde o resultado é armazenado na variável *output*, nessa camanda foi atribuído o valor 3 no primeiro parâmetro da função Dense, valor referente a quantidade de classes que compõem o *dataset*, essas classes são: cozinha, quarto e sala. O segundo parâmetro é a função de ativação *activation* que recebeu o valor "sigmoid", o *sigmoid* é responsável por devolver valores entre 0, 1 e 2 correspondentes às classes.

4.2.4.4 Treinamento do modelo

Com o modelo configurado precisa-se então treinar a rede, para realizar o treinamento utiliza-se a função *fit* passando três argumentos, o primeiro é o *dataset Train*, depois o número de épocas que será utilizado para o treinamento, por último

o *dataset Validation*. O treinamento ocorre através das épocas e cada uma delas é dividida em lote, conhecido como *batch size*, onde uma época é uma passagem por todas as linhas do conjunto de testes e um lote é composto de uma ou mais amostras consideradas pelo modelo, antes que seja feita a atualização dos seus pesos.

Para melhorar a performance do primeiro modelo e diminuir problemas como de *underfitting* e *overfitting* foi realizado algumas mudanças no modelo de treinamento, O `Código` 2 apresenta o modelo de treinamento modificado e em seguida será explicado detalhadamente cada mudança realizada.

CÓDIGO 2 – Segundo modelo de treinamento da CNN criado no Google Colab.

```
1    inputs = keras.Input(shape=image_shape)
2    x = data_augmentation(inputs)
3    x = tf.keras.layers.Rescaling(1./255)(x)
4    x = tf.keras.layers.Conv2D(32, 3, padding = 'same', activation =
    'relu')(x)
5    x = tf.keras.layers.Conv2D(32, 3, padding = 'same', activation =
    'relu')(x)
6    x = tf.keras.layers.MaxPooling2D()(x)
7    x = tf.keras.layers.Conv2D(64, 3, padding = 'same', activation =
    'relu')(x)
8    x = tf.keras.layers.Conv2D(64, 3, padding = 'same', activation =
    'relu')(x)
9    x = tf.keras.layers.MaxPooling2D()(x)
10   x = tf.keras.layers.Conv2D(128, 3, padding = 'same', activation
    = 'relu')(x)
11   x = tf.keras.layers.Conv2D(128, 3, padding = 'same', activation
    = 'relu')(x)
12   x = tf.keras.layers.MaxPooling2D()(x)
13   x = tf.keras.layers.Flatten()(x)
14   x = tf.keras.layers.Dense(128, activation = 'relu')(x)
15   x = tf.keras.layers.Dense(18, activation = 'relu')(x)
16   x = tf.keras.layers.Dense(64, activation = 'relu')(x)
17   output = tf.keras.layers.Dense(3, activation = 'sigmoid')(x)
18
```

FONTE: Os próprios autores.

a) 1° Mudança: Foi aumentada a quantidade de filtros da função Conv2D, para isso em cada camada foi duplicado as funções. Podendo ser observado nas linhas 5, 8 e 11 do `Código` 2.

b) 2° Mudança: Na camada 15 foi adicionado mais uma função *Dense* e o valor do primeiro parâmetro da função foi reduzido para 18, mantendo a função de ativação como *"relu"*.

c) 3° Mudança: Na camada 16 foi adicionado outra função *Dense* e atribuído o valor 64, mantendo novamente o valor da função de ativação como *"relu"*.

d) 4° Mudança: Por fim, foi criar uma função na segunda linha do `Código 2`, conhecida como *Data Augmentation*, responsável por realizar pequenas alterações nas imagens do *dataset*, essas alterações são pequenas rotações horizontais e pequenos *zoom* nas imagens, com isso a quantidade de imagens que a rede neural desconhece aumenta sem a necessidade de atribuir mais imagens para o *dataset*, contribuindo bastante para a diminuição do *overfitting* e do *underfitting*.

4.2.4.5 Avaliação do modelo

Após o treinamento do modelo, é preciso realizar uma avaliação do seu funcionamento. Essa avaliação pode ser feita através das comparações do *dataset* de validação ou utilizando outros *datasets* onde vai dizer o quão bem o modelo foi construído utilizando o conjunto de dados de treinamento.

Para avaliar como o modelo se comportou para os dados de treinamento, basta passar os mesmos dados de entrada e saída para a função *evaluate*. Essa função retorna uma lista com a taxa de *loss* e a acurácia do modelo para o conjunto de dados. Esses dados de entrada foram passados através do *dataset Unknown*, composto por imagens desconhecidas do modelo treinado. A Figura 28 ilustra a porcentagem de acurácia que foi de 0,43 e taxa de *loss* de -953 do primeiro modelo de CNN e a Figura 29 ilustra as porcentagem de acurácia que foi de 0,992 e a taxa de *loss* de 0,023 do segundo modelo após a realização das modificações descritas na seção 4.2.4.3.

FIGURA 28 – Avaliação do primeiro modelo.

```
3/3 [==============================] - 7s 2s/step - loss: -953702.9375 - accuracy: 0.4167
Dataset Test Loss:     -953702.9375
Dataset Test Accuracy: 0.4166666567325592
```

FONTE: Os próprios autores.

FIGURA 29 – Avalidação do segundo modelo.

```
2/2 [==============================] - 1s 314ms/step - loss: 0.0235 - acc: 0.9925
Dataset Test Loss:     0.023465894162654877
Dataset Test Accuracy: 0.9925000071525574
```

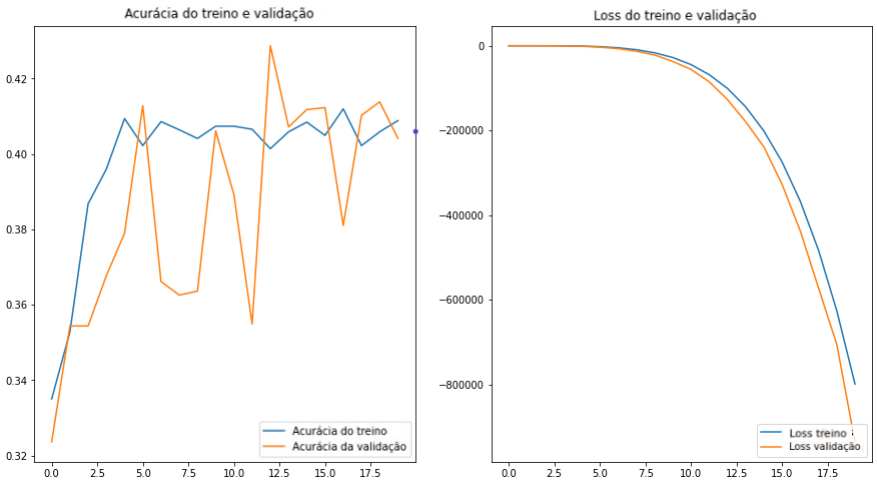
FONTE: Os próprios autores.

Ao observar a Figura 28 pode-se notar que a acurácia juntamente com a taxa de *loss* se comportaram de maneira não satisfatória, demonstrando porcentagens muito negativas em ambas as métricas. Isso ocorreu por diversos fatores, um deles é o fato do modelo do `Código 1` não ter uma complexidade para realizar o treinamento, outro fator pode ser pelo fato das imagens utilizadas no primeiro modelo foram retiradas em apenas um ângulo, podendo ocorrer também por utilizar poucas camadas de filtros convolucionas, entre muitos outros fatores.

Ao observar a Figura 29 representado pelo modelo do `Código` 2 é possível notar que os valores da acurácia e taxa de *loss* obtiveram resultados muito superiores ao primeiro modelo e muito mais satisfatórios, isso ocorreu através das mudanças aplicadas no segundo modelo, resultando em valores positivos nas métricas utilizadas (acurácia e *loss*).

O gráfico 2 apresenta os resultados de acurácia e *loss* do primeiro modelo representado no `Código` 1. No gráfico da esquerda o eixo x representa o avanço da acurácia do *dataset Train* juntamente com o *dataset Validation* e o eixo y representa as épocas em que esse avanço acontece. No gráfico da direita o eixo x representa a taxa de *loss* de ambos os *datasets* e o eixo y as épocas que ocorre o avanço da taxa de *loss*.

GRÁFICO 2 – Gráfico para verificar a acurácia e taxa de *loss* do primeiro modelo.



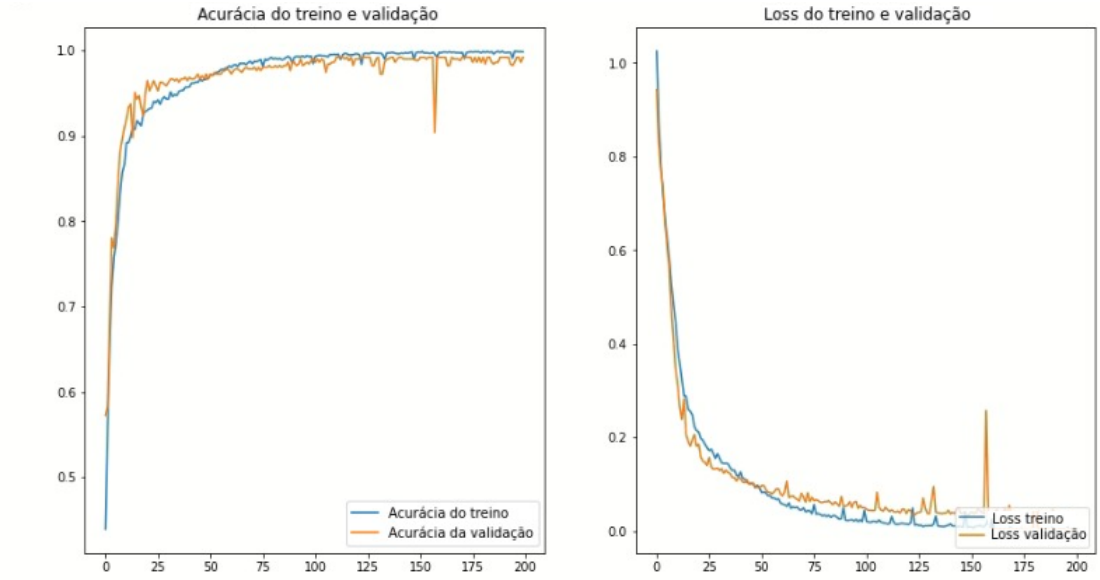
FONTE: Os próprios autores.

No gráfico 3 é apresentado o resultado do modelo após a aplicação das modificações realizadas no `Código` 2. No gráfico do lado esquerdo, o eixo x representa a avaliação do avanço da acurácia do modelo modificado e o eixo y as épocas do crecimento da acurácia, no gráfico da direta o eixo x representa o decaimento da função de perda e o eixo y as épocas em que ocorre o decaimento do eixo x.

Quando há um alto erro no treinamento com valor próximo ao erro na validação, ocorre o clássico problema de *underfitting* e quando há um baixo erro no treinamento e alto erro na validação, temos um clássico problema de *overfitting*.

Com os gráficos ilustrados no Gráfico 2 representados pela avaliação através do `Código` 1, consegue-se observar que existe uma grande diferença entre a acurácia dos *datasets* juntamente com uma queda elevada na taxa de *loss* dos *datasets* de *Train* e *Validation*, isso demonstra que o primeiro modelo não conseguiu realizar um bom treinamento por cause de diversos fatores citados anteriormente.

GRÁFICO 3 – Gráfico para verificar a acurácia e taxa de *loss* do segundo modelo.



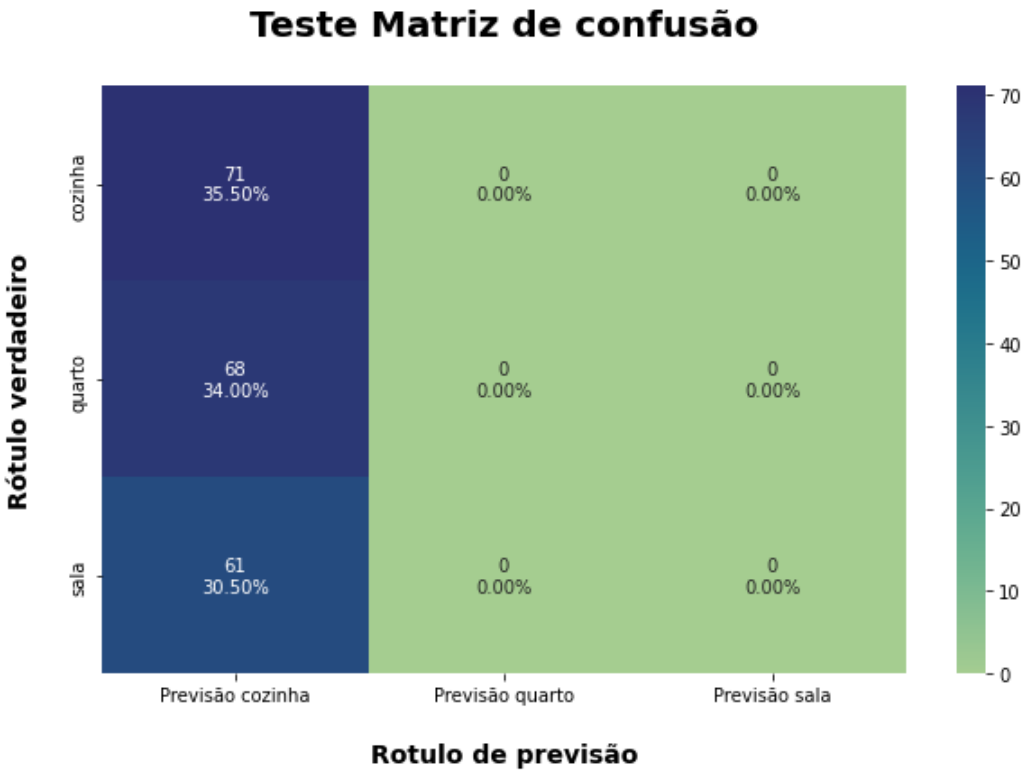
FONTE: Os próprios autores.

Ao observar o Gráfico 2 representado pela avaliação do `Código` 2, foi possível identificar que o modelo se comportou muito melhor, diminuindo os problemas de *underfitting* e *overfitting*.

Outra maneira para avaliar a CNN é a matriz de confusão, com dimensões 3x3, é uma métrica voltada para modelos de classificação que permite observar o desempenho do algoritmo. A Figura 30 ilustra a matriz de confusão do primeiro modelo e a Figura 31 exibe a matriz de confusão do segundo modelo, cada linha representa a instância de uma classe predita, enquanto cada coluna representa a instância da classe atual. os campos com tons azuis representam a acurácia e previsões que o modelo obteve em cada classe e os campos verdes represetam os erros cometidos na avaliação.

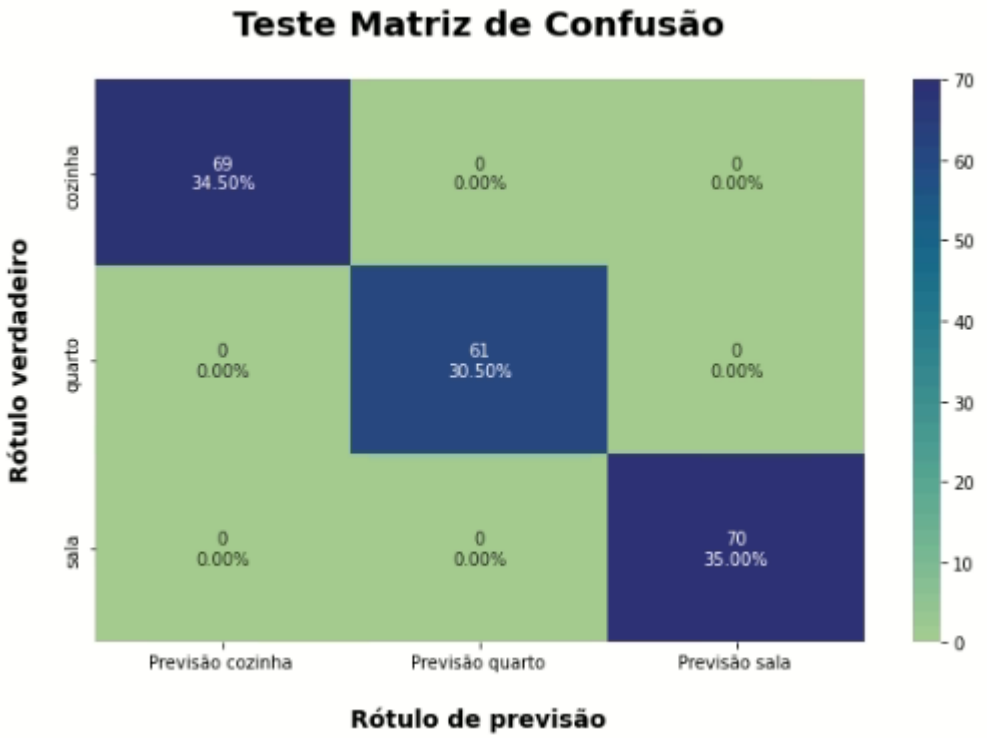
Quando observamos a matriz de confusão ilustrada na Figura 30, pode-se notar que igual as outras formas de avaliação do desempenho do treinamento da CNN, o primeiro modelo se comportou muito mal, acertando apenas as imagens do cômodo cozinha, realizando previsões negativas para os outros dois cômodos, porém quando observamos a Figura 31 é possível ver que o segundo modelo realizou todas as previsões corretamente, mostrando um desempenho muito superior ao primeiro modelo em todos os testes de avaliação realizados.

FIGURA 30 – Matriz de confusão do primeiro modelo.



FONTE: Os próprios autores.

FIGURA 31 – Matriz de confusão do segundo modelo.



FONTE: Os próprios autores.

5 CONCLUSÃO

Nos últimos anos, as simulações computacionais tem ganhado grande destaque em práticamente todas as áreas, isso se deve ao desenvolvimento de novos produtos, buscando reduzir custos, tempo e recursos consumido em ensaios e testes experimentais, que podem ser minimizados utilizando simulações.

O desenvolvimento do presente estudo possibilitou uma análise de como um algoritmo de Aprendizagem de Máquina aplicado na identificação de ambientes simulados préviamente conhecidos atráves da extração de imagens de cada cômodo do ambiente simulado, utilizando o uso de uma câmera embarcada do Turtlebot3.

Para as etapas propostas de desenvolvimento, a extração das imagens do ambiente simulado e a modelagem das camadas da CNN foram as que demandaram maior tempo e esforço, a primeira devido a quantidade elevada de imagens que seriam necessárias para que fosse possível obter os melhores resultados no treinamento e também pelos inúmeros ângulos que foram realizados as capturas, afim de diversificar o posicionamento dos objetos com relação a imagem.

Outra dificuldade enfrentada foi durante o treinamento da CNN para o reconhecimento das imagens, pois utilizando o primeiro modelo proposto e o conjunto de dados coletados, observou-se grande facilidade da rede entrar em *overfitting*, reduzindo o desempenho no teste.

Algumas técnicas como a insercão de camadas Conv2D e *Data Augmentation* para a geração de valores mais consolidados foram essenciais para elevar os resultados finais do treinamento e diminuir gradualmente o *overfitting*, para que a acurácia do modelo utilizado chegasse a valores próximos aos esperados, outro fator que contribuiu para alcançar esses valores foram as próprias imagens, que por serem capturadas de ambientes virtuais não sofreram modificações de luminosidade, ruídos ou qualquer outro tipo de interferência que pudesse ocasionar na distorção dos dados.

Sendo assim, conclui-se que o algoritmo de Aprendizagem de Máquina apresentado, conseguiu realizar a identificação de um ambiente simulados previamente conhecido, fazendo uso de imagens coletadas por uma câmera embarcada em um robô dentro desses ambientes.

No problema em questão a CNN apresentou resultados satisfatórios, evidenciando seu desempenho nos problemas envolvendo imagens digitais. Como trabalhos futuros sugere-se a inserção de modelos semelhantes em uma base robótica afim de avaliar os resultados gerados a partir ambientes reais de maior complexidade.

REFERÊNCIAS

AMAZON. *Amazon Machine Learning*. 2016. Disponível em: <https://docs.aws.amazon.compt_brmachinelearninglatestdgmachinelearning-dg.pdf#modelfitunderfittingvsoverfitting>. Acesso em: 03 Julho 2022.

ARAÚJO, M. A. Aplicação do ros no controle de movimento de robôs equipados com motores dynamixel em linux de tempo real. *Universidade Federal de Uberlândia*, 2017.

AUTOMATION, A. for A. *RoboDk*. 2021. Disponível em: <https://www.automate.org/companies/robodk>. Acesso em: 21 Novembro 2021.

BÔAS, B. V. *Robôs ganham espaço em fábricas e no campo assumindo tarefas de risco*. 2021. Disponível em: <https://www.cnnbrasil.com.br/business/robos-ganham-espaco-em-fabricas-e-no-campo-assumindo-tarefas-de-risco/>. Acesso em: 22 Outubro 2021.

CUNHA, L. C. da. *Redes Neurais Convolucionais e Segmentação de Imagens - Uma Revisão Bibliográfica*. 2020. Disponível em: <https://www.monografias.ufop.brbitstream3540000028726MONOGRAFIA_RedesNeuraisConvolucionais.pdf>. Acesso em: 01 Julho 2022.

DAMIEN. *Webots, um software de código aberto para simular robôs móveis*. 2020. Disponível em: <https://ubunlog.com/pt/webots-software-simulacion-robots-moviles/>. Acesso em: 21 Novembro 2021.

ESPINOSA, C. A. D. *Uma aplicação de navegação robótica autônoma através de visão computacional estéreo*. 00 p. Dissertação (Mestrado em Ciência da Computação) — Universidade de São Paulo - USP, São Paulo, 2010.

FERSILTEC, E. *AGV X AMR: Tudo o Que Precisa Saber sobre Robôs Móveis*. 2021. Disponível em: <https://fersiltec.com.br/blog/robotica/agv-x-amr/>. Acesso em: 05 Novembro 2021.

GAMERO, I. *Robôs Industriais: tudo o que você precisa saber!* 2018. Disponível em: <https://pollux.com.br/blog/robos-industriais-tudo-o-que-voce-precisa-saber/>. Acesso em: 22 Outubro 2021.

HERRERA, L. E. Y. Mapeamento e localização simultânea de robôs móveis usando dp-slam e um único medidor laser por varredura. *Pontifícia Universidade Católica do Rio de Janeiro*, 2011.

LUGER, G. F. *Inteligência Artificial*. 6. ed. São Paulo: Pearson, 2013.

MATARIĆ, M. J. *Introdução à robótica*. 1. ed. São Paulo: BLUCHER, 2014.

MEDEIRO, L. F. de. *Inteligência Artificial aplicada: uma abordagem introdutória*. 1. ed. Curitiba: Intersaberes, 2018.

PEOPLE, E. *Conheça 6 robôs que foram para o espaço*. 2017. Disponível em: <https://www.people.com.br/noticias/robotica/conheca-6-robos-que-foram-para-o-espaco>. Acesso em: 05 Novembro 2021.

POUBEL, L. *Robótica Open Source: Começando com Gazebo e ROS 2*. 2019. Disponível em: <https://www.infoq.com/br/articles/ros-2-gazebo-tutorial/>. Acesso em: 09 Novembro 2021.

REINALDO, J. O. *Reconhecimento de objetos no desenvolvimento de um sistema de navegação inteligente para robôs móveis*. 63 p. Dissertação (Mestrado em Ciência da Computação) — Universidade do Estado do Rio Grande do Norte, Mossoró - RN, 2015.

ROCHA, L. *Robô Perseverance da Nasa irá coletar amostras em Marte; veja imagens da região*. 2021. Disponível em: <https://www.cnnbrasil.com.br/tecnologia/robo-perseverance-da-nasa-ira-coletar-amostras-em-marte-veja-imagens-da-regiao/>. Acesso em: 22 Outubro 2021.

ROS.ORG. *ROS/Bibliotecas*. 2021. Disponível em: <http://wiki.ros.org/Client%20Libraries>. Acesso em: 25 Outubro 2021.

SILVA, J. F. R. da; GRILLO, C. A. C.; SOUSA, J. S. da S. *Ambiente de simulação para navegação robótica*. 2002. Disponível em: <https://www.aedb.br/seget/arquivos/artigos06/903_ambiente%20de %20simulacao%20de%20navegacao.pdf>. Acesso em: 09 Novembro 2021.

SILVEIRA, C. B. *Os 6 Principais Tipos de Robôs Industriais*. 2019. Disponível em: <https://www.citisystems.com.br/tipos-de-robos/>. Acesso em: 01 Novembro 2021.

TORO, W. M. M. *Navegação robótica relacional baseada em web considerando incerteza na percepção*. 74 p. Dissertação (Mestrado em Engenharia de Software) — Universidade de São Paulo - USP, São Paulo, 2014.

UTSCH, K. G. Uso de redes neurais convolucionais para classificação de imagens digitais de lesões de pele. *Universidade Federal do Espírito Santo*, 2018.

WARELINE, E. *Entenda como a Robótica na Medicina vai revolucionar os procedimentos cirúrgicos*. 2015. Disponível em: <https://www.wareline.com.br/wareline/noticias/como-funciona-a-robotica-na-medicina/>. Acesso em: 28 Outubro 2021.

WIKI.ROS.ORG. *ROS/catkin*. 2021. Disponível em: <http://wiki.ros.org/catkin>. Acesso em: 04 Julho 2022.

WIKI.ROS.ORG. *ROS/Introdução*. 2021. Disponível em: <http://wiki.ros.org/ROS/Introduction>. Acesso em: 25 Outubro 2021.

WIKI.ROS.ORG. *ROS/rviz*. 2021. Disponível em: <http://wiki.ros.org/rviz/Overview>. Acesso em: 04 Julho 2022.

WOLF, D. F. *et al. Intelligent Robotics: From Simulation to Real World Applications*. 2009. Disponível em: <http://inct-sec.icmc.usp.br/actrep/sites/default/files/highlights/Tutorial-JAI.pdf>. Acesso em: 07 Novembro 2021.

YANG, G.-Z. *The grand challenges of Science Robotics*. 2022. Disponível em: <https://www.science.org/doi/10.1126/scirobotics.aar7650>. Acesso em: 02 Julho 2022.

YUKINOBU, A. *Mapeamento de ambientes externos utilizando robôs móveis*. 122 p. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2010.