UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Lucas Toschi de Oliveira

Sistema de detecção de pontos de emergência em campos agrícolas com o robô TerraSentia

São Carlos

2024

Lucas Toschi de Oliveira

Sistema de detecção de pontos de emergência em campos agrícolas com o robô TerraSentia

Monografia apresentada ao Curso de Engenharia Elétrica com Ênfase em Eletrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Assoc. Marcelo Becker

São Carlos

2024

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

Oliveira, Lucas Toschi de
O48s Sistema de detecção de pontos de emergência em campos agrícolas com o robô TerraSentia / Lucas Toschi de Oliveira; orientador Marcelo Becker. São Carlos, 2024.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2024.

1. Agricultura. 2. Percepção. 3. Detecção. 4. Robótica. 5. SLAM. 6. Visão Computacional. 7. Deep Learning. 8. Sistemas Embarcados. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Lucas Toschi de Oliveira

Título: "Sistema de detecção de pontos de emergência em campos agrícolas com o robô TerraSentia"

Trabalho de Conclusão de Curso defendido e aprovado em 05/06/2024,

com NOTA 10 (Dez), pela Comissão Julgadora:

Prof. Associado Marcelo Becker - Orientador - SEM/EESC/USP

Prof. Associado Valdir Grassi Junior - SEL/EESC/USP

Prof. Titular Marco Henrique Terra - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Professor Associado José Carlos de Melo Vieira Júnior

À minha mãe, Andréa de Morais Toschi Oliveira,
ao meu pai, Sebastião de Oliveira,
ao meu irmão, Gabriel Toschi,
à minha companhia diária, Carolina

AGRADECIMENTOS

Aos meus pais e ao meu irmão, por me fornecerem a possibilidade de cursar uma graduação; por me fornecerem conselhos valiosos e companhia constante; pelo carinho e apoio incondicional.

Ao Professor Associado Marcelo Becker, por ter sido meu orientador e ter me apoiado durante toda a minha trajetória como pesquisador durante a graduação; sua disponibilidade para me apoiar quando necessário foi fundamental para o desenvolvimento deste trabalho. Agradeço também por sempre trazer oportunidades para mim e confiar nas minhas capacidades.

Ao Dr. Vitor Akihiro Hisano Higuti, pelos valiosos conselhos, sugestões e revisões que enriqueceram significativamente este trabalho. Seu apoio inestimável durante o intercâmbio foi essencial para que eu pudesse desenvolver parte deste projeto com sucesso. Sua dedicação e disponibilidade para ajudar sempre foram muito apreciadas.

Ao Dr. Girish Chowdhary, pelo apoio para viabilização do intercâmbio e pela orientação durante o desenvolvimento de parte deste projeto. Sua expertise e conselhos foram de grande importância para o aprimoramento das minhas habilidades de pesquisa e para a realização deste trabalho.

Ao Me. Mateus Valverde Gasparino, pelo apoio para viabilização do intercâmbio e pelos inestimáveis conselhos sobre este trabalho e minha carreira profissional. Suas orientações ajudaram-me a tomar decisões importantes e a enxergar novas perspectivas para o meu futuro acadêmico e profissional.

Ao Me. José Roberto Cuarán Valenzuela, pelo apoio durante o desenvolvimento do produto final deste projeto. Seus ensinamentos teóricos foram fundamentais para a compreensão dos conceitos necessários, e sua ajuda também foi crucial para viabilizar meu intercâmbio. Agradeço por sua dedicação e comprometimento.

À Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), pelo apoio financeiro através dos processos 2020/11262-0 e 2022/13040-0.

À empresa EarthSense, pelo fornecimento do robô TerraSentia e por alguns dados utilizados neste projeto. A contribuição de vocês foi essencial para o desenvolvimento e sucesso deste trabalho.

RESUMO

Oliveira, L. T. **Sistema de detecção de pontos de emergência em campos agrícolas com o robô TerraSentia**. 2024. 72p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2024.

O crescimento da população mundial exige que os métodos e tecnologias empregados na produção de alimentos sejam cada vez mais eficientes, produzindo mais com menos espaço. A robótica autônoma é uma área cada vez mais pesquisada para auxiliar nesse problema. Nesse contexto, o uso de algoritmos Localização e Mapeamento Simultâneos (LOMAS) é essencial para criar um mapa e localizar um robô nele. Porém, para ambientes dinâmicos (como o agrícola), eles ainda são um problema aberto na literatura. Este trabalho busca estudar a aplicação de algoritmos LOMAS utilizando dados do robô TerraSentia, uma plataforma robótica desenvolvida pela Universidade de São Paulo e a Universidade de Illinois (Urbana-Champaign). Além da aplicação direta de algoritmos LOMAS, este trabalho também propõe módulos auxiliares para melhorar o desempenho deles no ambiente agrícola. A aplicação direta de LOMAS com os dados obtidas não se mostrou promissora. Dentre os módulos propostos, destaca-se os resultados promissores obtidos com o detector de pontos de emergência das plantas, pontos nos quais elas emergem do solo. Trabalhos futuros também são propostos para validar em mais profundidade os módulos juntamente com os algoritmos LOMAS.

Palavras-chave: Agricultura, Percepção, Detecção, Robótica, LOMAS, Visão Computacional, Deep Learning, Sistemas Embarcados

ABSTRACT

Oliveira, L. T. **Emergency point detection system in agricultural fields with the TerraSentia robot**. 2024. 72p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2024.

The growth of the world population demands that the methods and technologies employed in food production become increasingly efficient, producing more with less space. Autonomous robotics is an area increasingly researched to assist with this problem. In this context, the use of Simultaneous Localization and Mapping (SLAM) algorithms is essential for creating a map and locating a robot within it. However, for dynamic environments (such as agriculture), they remain an open problem in the literature. This work seeks to study the application of SLAM algorithms using data from the TerraSentia robot, a robotic platform developed by the University of São Paulo and the University of Illinois (Urbana-Champaign). In addition to the direct application of SLAM algorithms, this work also proposes auxiliary modules to improve their performance in the agricultural environment. The direct application of SLAM with the obtained data did not show promising results. Among the proposed modules, the promising results obtained with the plant emergence point detector stand out, identifying points where plants emerge from the soil. Future work is also proposed to validate the modules and the SLAM algorithms in greater depth.

Keywords: Agriculture, Perception, Detection, Robotics, SLAM, Computer Vision, Deep Learning, Embedded Systems

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

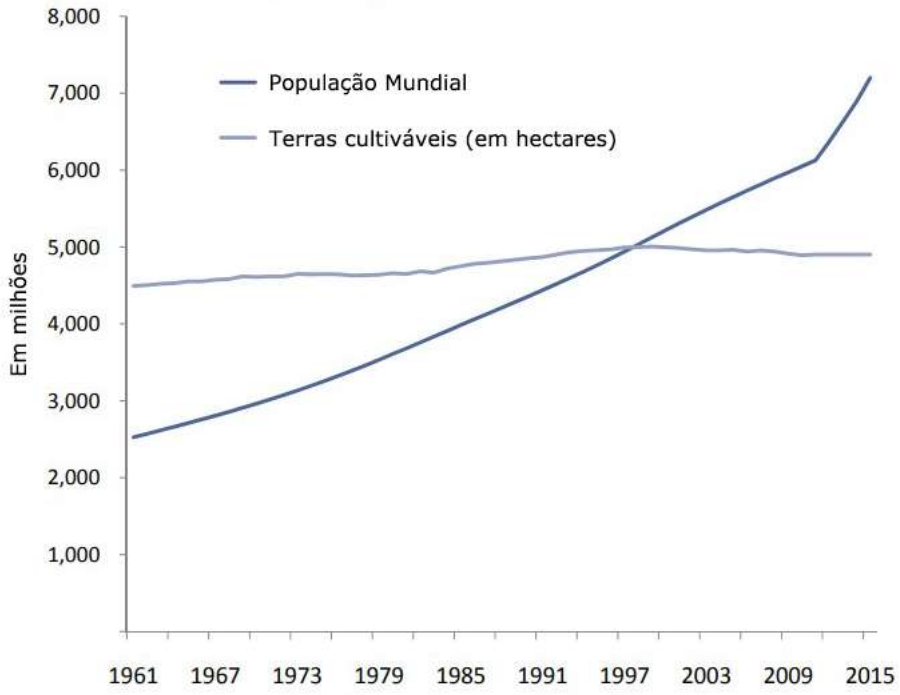
Em 1798, Thomas Malthus, intelectual iluminista, publicou uma obra chamada *"An Essay on the Principle of Population"*. Nesta obra, Malthus afirmou que o crescimento populacional tinha o formato de uma progressão geométrica, enquanto que o crescimento na produção de alimentos uma progressão aritmética (ou seja, mais lenta que a primeira). Segundo os malthusianos modernos, essa é uma dificuldade que nunca poderá ser superada. Entretanto, de acordo com o relatório econômico do grupo financeiro multinacional Goldman Sachs de 2016 (REVICH et al., 2016), essa conclusão é precipitada, apesar da análise ser compatível com a realidade.

O documento considera que a humanidade enfrenta o problema de se sustentar no futuro, diante da contínua expansão no número de indivíduos. Na Figura 1, pode-se verificar que, até 2015, a população mundial mostrou um crescimento mais acelerado que a quantidade de hectares de terra cultivável. Contudo, de acordo com o estudo, a resposta para o futuro está no avanço tecnológico, com o surgimento de veículos autônomos e sistemas administrativos inteligentes.

Nesse contexto de necessidade de implementações mais sofisticadas para sustento da vida humana, a agricultura de precisão surgiu como um novo conceito a partir dos anos 80. O paradigma objetiva o aumento na produtividade e na qualidade dos rendimentos agrícolas, reduzindo os custos. A robótica, com sua versatilidade, é considerada uma ferramenta razoável nessa conjuntura.

Entretanto, nem todas as tecnologias desenvolvidas para plataformas robóticas em ambi-

Figura 1 – Relação entre população mundial e a quantidade de terra cultivável (em hectares)



Fonte: adaptado de (REVICH et al., 2016)

entes controlados são aplicáveis no meio agrícola (Gao et al., 2018). Algumas delas se baseiam em condições específicas, que nem sempre são satisfeitas em campos e plantações. Além disso, por se tratar de um ambiente consideravelmente mais complexo, uma aplicação pode utilizar diversas tecnologias de forma concatenada para obter resultados minimamente interessantes.

1.1 Motivação

Uma das tecnologias para navegação autônoma que está em desenvolvimento nos últimos 40 anos é a chamada *"Simultaneous Localization And Mapping"* (SLAM), traduzida por *"Localização e Mapeamento Simultâneos"* em português (SMITH; CHEESEMAN, 1986). Essa ferramenta propõe a utilização de instrumentos para que o robô possa entender o ambiente que o circunda e ser capaz de estimar sua posição de acordo com seus arredores. As informações do espaço no qual o robô está inserido podem ser captadas principalmente por dispositivos como sensores acústicos ou ópticos (como a tecnologia LiDAR, *"Light Detection And Ranging"*) e câmeras (monoculares ou estéreo) (ZAFFAR et al., 2018). Quando se utilizam sensores visuais, a tecnologia é chamada especificamente de *Visual Simultaneous Localization and Mapping* (VSLAM), traduzida por *"Localização e Mapeamento Simultâneos Visual"* (Han; Xi, 2020a).

Os métodos tradicionais de SLAM levam em consideração que o ambiente não se modifica durante a extração das características, o que não pode ser assumido na maioria dos ambientes reais. Na implementação convencional, o sistema carece de um entendimento abstrato do meio para diferenciar objetos fixos daqueles que se movem, resultando em erros grosseiros de trajetória e podendo ocasionar o colapso do controle de navegação (Han; Xi, 2020a).

A aplicação dessa tecnologia em meios com objetos se movimentando começou em meados de 2003 e é chamado de *"SLAM in Dynamic Environments"* (SLAMIDE), traduzido por *"Localização e Mapeamento Simultâneos em Ambientes Dinâmicos"*. Os principais desafios são: como definir objetos como dinâmicos a partir de *pixels* planos e como identificar tais objetos. Com base nos resultados favoráveis da implementação de *Deep Learning* no processamento de imagens, esta foi incorporada por alguns pesquisadores na aplicação do SLAMIDE (XIAO et al., 2019).

A principal contribuição deste projeto é a aplicação do método descrito por Yuan et al. (2021). O detector desenvolvido obteve resultados promissores na identificação dos pontos de emergência das plantas, a partir da suposição de que sejam os pontos mais estáticos do cenário agrícola. Os pontos de emergência são definidos como os pontos em que cada uma das plantas emergem do solo e, portanto, não mudam com o tempo. Testes extensivos desse detector ainda precisam ser realizados em conjunto com algoritmos SLAM.

1.2 Objetivos

Em sua concepção, este projeto teve como objetivo principal o estudo da aplicação de SLAM em contextos agrícolas, utilizando os dados de uma plataforma robótica. Entretanto, como é mostrado, a aplicação de SLAM de forma direta com dados reais não gerou resultados satisfatórios. Nesse contexto, o objetivo final do projeto foi alterado para o desenvolvimento de

algoritmos de percepção que, futuramente, pudessem ser úteis para complementar algoritmos de SLAM. O contexto completo dessa pesquisa é descrita nesta monografia, com o intuito de justificar as decisões tomadas.

Dessa forma, tendo em vista o contexto e objetivo central do projeto, os seguintes objetivos secundários foram realizados:

- Aplicação de um algoritmo de SLAM com dados reais
- Discussão e análise dos resultados do algoritmo de SLAM
- Implementação de um detector de objetos usando *Deep Learning*
- Discussão e análise dos resultados do detector de objetos
- Implementação de uma pipeline de detecção com segmentação de máscaras usando *Deep Learning*
- Discussão e análise final dos resultados da abordagem final
- Discussão comparativa entre métodos
- Proposição de melhorias e trabalhos futuros

1.3 Organização do documento

Este documento é organizado nos seguintes capítulos:

1. **Introdução:** introduz o contexto do problema, descreve o problema em si, apresenta o fluxo de desenvolvimento e define os objetivos do projeto.
2. **Plataforma robótica:** descreve a plataforma robótica e os dados utilizados no projeto
3. **Aplicação direta de SLAM:** apresenta o desenvolvimento e os resultados da aplicação direta de SLAM brevemente
4. **Detector de objetos:** apresenta, explica e justifica sua utilização agregado a SLAM. Também apresenta brevemente os resultados obtidos.
5. **Abordagem final:** apresenta, explica e justifica sua utilização. Apresenta os resultados finais obtidos.
6. **Conclusão:** reflete sobre o trabalho realizado, propõe melhorias e passos futuros.

2 PLATAFORMA ROBÓTICA

O TerraSentia (mostrado na Fig. 2) é um robô de campo de baixo custo, leve e compacto. Ele foi desenvolvido pelo Distributed Autonomous Systems Laboratory (DASLab), localizado na Universidade de Illinois (Urbana-Champaign, EUA) e o Laboratório de Robótica Móvel (LabRoM), localizado na Universidade de São Paulo (USP, Brasil). Ele foi descrito em Kayacan, Zhang e Chowdhary (2018) e Higuti et al. (2018) e é projetado para se locomover abaixo do dossel de uma plantação. A plataforma possui um minicomputador e pode ser equipada com diversos sensores (como câmeras, sensores inerciais e LiDAR).

Figura 2 – Plataforma TerraSentia

Fonte: (EARTHSENSE, 2021)

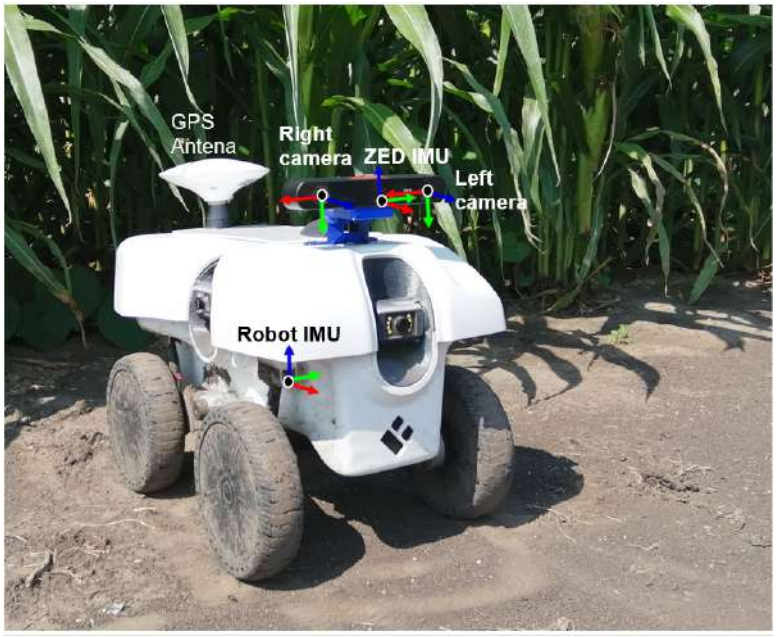
2.1 Conjunto de dados utilizado

O autor entrou em contato com o DASLab para requisitar dados coletados pela equipe utilizando o robô TerraSentia. O laboratório gentilmente cedeu um conjunto para fins de pesquisa, que foram utilizados durante todo o projeto.

Os membros do DASLab coletaram dados durante vários meses no verão de 2022. Eles customizaram o robô (mostrado na Figura 3) com uma câmera ZED2 (STEREOLABS, 2023) e um receptor GPS RTK. Além disso, mantiveram presente a *Inertial Measurement Unit* (IMU), Unidade de Medição Inercial em tradução livre. Entre outros tipos de dados, as sequências gravadas continham imagens RGB, imagens de profundidade e a posição do robô estimada com um Extended Kalman Filter (EKF) (MEEUSSEN, 2023).

A Figura 3 também mostra o posicionamento dos sensores em relação ao robô. Cada um deles está representado de acordo com um sistema de coordenadas 3D, contendo uma origem (representada por um ponto preto com contorno branco) e os eixos x, y e z, representados

Figura 3 – O robô TerraSentia customizado



Fonte: DASLab

pelas cores vermelha, verde e azul, respectivamente. Vale ressaltar que a IMU está posicionada aproximadamente no centro de massa do robô. Dessa forma, pode-se considerar o centro de coordenadas do robô como coincidente com o desse sensor.

As sequências estavam armazenadas em arquivos SVO (para compressão de alta resolução) (STEREOLABS, 2023b), um formato proprietário da empresa StereoLabs (STEREOLABS, 2023). Para descompactar os dados e obtê-los como tópicos do Robot Operating System (ROS), foi utilizado o pacote zed-ros-wrapper (STEREOLABS, 2023a). Para a utilização do pacote, é necessário uma Graphics Processing Unit (GPU) configurada. Para tanto, utilizou-se um notebook com uma NVIDIA GeForce GTX 1650 (NVIDIA, 2023b) e a API NVIDIA CUDA, na versão 12.2 (NVIDIA, 2023a).

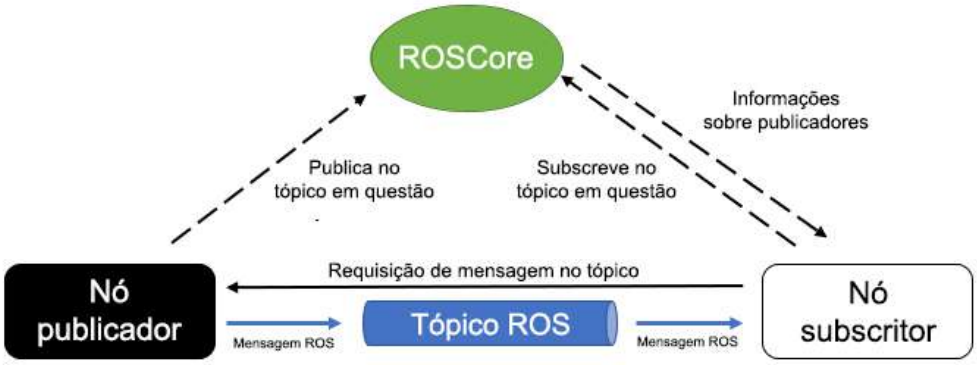
2.2 Robot Operating System (ROS)

Robot Operating System (ROS, traduzido para "Sistema Operacional Robótico") (ROBOTICS, 2021) é um conjunto de *frameworks* que auxilia o desenvolvimento de sistemas robóticos. Ele é utilizado no funcionamento da plataforma TerraSentia e também foi utilizado no desenvolvimento deste projeto em diferentes etapas.

Uma das características mais notáveis dessa ferramenta é sua capacidade de concretizar a modularização de um sistema. ROS possui um sistema de "nós", arquivos executáveis que podem estar desenvolvidos em linguagens distintas. Cada nó é responsável por uma função diferente do robô, facilitando a execução de tarefas de forma paralela. Uma estrutura de programas chamada de *roscore* é responsável por gerenciar os diferentes nós de um mesmo sistema, além de armazenar parâmetros globais, que podem ser consultados por qualquer parte da organização.

Para trocarem dados, os nós utilizam uma estrutura chamada de Tópico ROS. Ele é um canal de comunicação entre diferentes partes do robô. Exemplificando, um nó responsável por fazer a leitura de sensores se inscreve no papel de publicador de um tópico, enquanto que outro, responsável pelo tratamento desses dados, se torna assinante do tópico, recebendo os dados. A inscrição ou subscrição a um tópico é informada e gerenciada pelo *roscore*. Vale ressaltar que as informações são estruturadas em Mensagens ROS. Assim, uma mensagem pode conter informações como translação e rotação do robô em vários eixos, que são enviadas de uma só vez. Esse processo pode ser visualizado na Figura 4.

Figura 4 – Esquema de comunicação entre nós utilizando *ROS*



Fonte: (ROBERT, 2020) (adaptado pelo autor)

Nesse contexto, um nó, responsável pela recepção de dados de uma câmera, pode publicar as imagens capturadas através de um tópico para um outro, cuja função é de realizar algum tipo de processamento.

3 APLICAÇÃO DIRETA DE SLAM

A primeira etapa deste projeto consistiu na aplicação de um algoritmo de VSLAM diretamente com as imagens do robô TerraSentia. O sistema escolhido para aplicação foi o ORB-SLAM 2 (Mur-Artal; Tardós, 2017) para câmeras monoculares, por se tratar de um sistema completo e representativo. Vale ressaltar que esta tecnologia é *open-source* (Mur-Artal et al., 2022a), além de também já ser amplamente utilizada pela comunidade ROS (Mur-Artal et al., 2022b).

3.1 ORB-SLAM2

O sistema ORB-SLAM2 (Mur-Artal; Tardós, 2017) se trata de uma ferramenta completa para câmeras monocular e stereo. Em relação ao uso desse sistema para câmeras monoculares, seu predecessor ORB-SLAM (MUR-ARTAL; MONTIEL; TARDÓS, 2015) utiliza quase os mesmos princípios e técnicas. Com o objetivo de obter um entendimento de algoritmos SLAM adequado ao nível de estudo proposto, o sistema ORB-SLAM foi alvo de uma análise mais detalhada. Ele é dividido em cinco etapas diferentes: **detecção**, **localização**, **mapeamento**, **relocalização** e **fechamento de loops**. A seguir, cada uma dessas etapas será descrita brevemente.

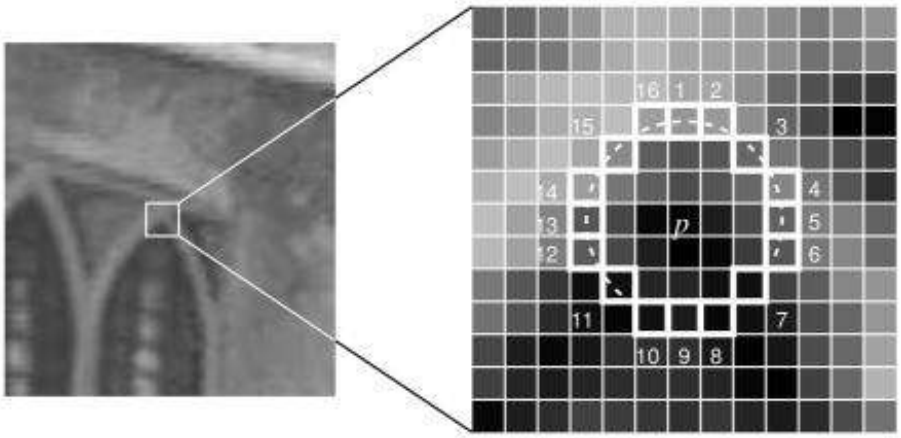
3.1.1 Detecção

A etapa de **detecção** é capaz de extrair características do ambiente através de descritores de imagem. Eles procuram identificar atributos de uma imagem, possibilitando a comparação com outras imagens. Um exemplo de característica são os "cantos" presentes no cenário. Computacionalmente, eles podem ser identificados através da análise de regiões que possuem variações abruptas em relação às regiões subjacentes (OPENCV, 2022d).

O descritor de imagem utilizado pelo ORB-SLAM é chamado de ORB. Este, por sua vez, é uma junção do detector FAST (*Features from Accelerated Segment Test*) e do descritor BRIEF (*Binary Robust Independent Elementary Features*) (OPENCV, 2022c). O detector FAST realiza a identificação de cantos através da análise da intensidade de alguns pixels localizados em uma circunferência em torno de cada ponto da imagem, como mostrado na Figura 5. Além disso, se utiliza de uma árvore de decisão e supressão dos cantos com menor pontuação para melhores resultados (OPENCV, 2022b).

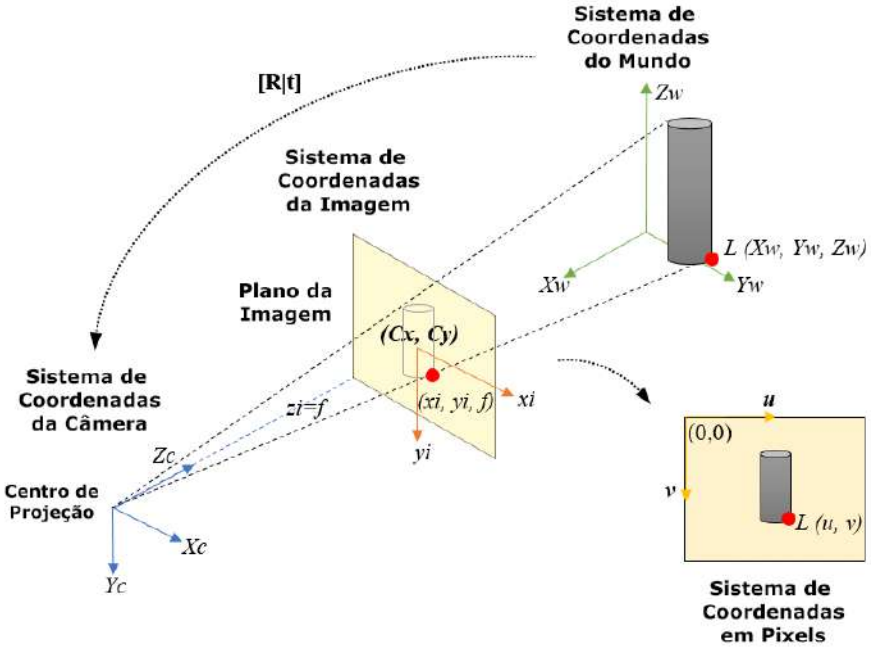
O descritor BRIEF elege alguns pares de características obtidos da imagem e realiza comparações de intensidade entre eles. Os resultados dessas comparações geram vetores de valores binários. Dessa forma, duas imagens podem ser comparadas utilizando a distância Hamming. Esta, por sua vez, se trata do número de valores diferentes entre dois vetores binários (OPENCV, 2022a).

Figura 5 – Funcionamento do descritor FAST



Para um pixel p, 16 valores dispostos circularmente ao redor de p são considerados no cálculo do descritor. Fonte: (OPENCV, 2022b)

Figura 6 – Modelo de câmera *"pinhole"*



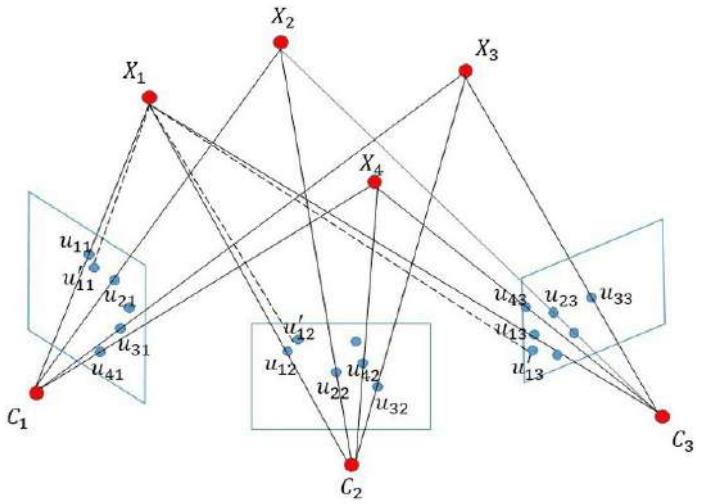
Fonte: (ORTIZ; GONçALVES; CABRERA, 2017) - tradução livre

3.1.2 Localização

A etapa de **localização** consiste na identificação da posição e orientação da câmera em relação às características extraídas do cenário. Para isso, o primeiro passo é utilizar um modelo de projeção dos pontos 3D do mundo real em relação ao plano da imagem 2D. Para isso, obtém-se as transformações entre eixos de coordenadas do mundo real, da imagem, do sistema adotado para os pixels e da própria câmera (mostrado na Figura 6). Essa relação é obtida através de semelhança de triângulos, transformações de corpo rígido (considerando coordenadas homogêneas) e correções para distorções de câmeras (CHEN; CHEN; WANG, 2019).

Um robô com um sistema de VSLAM possui um sistema de aquisição de imagens que

Figura 7 – Funcionamento do algoritmo de *Bundle Adjustment*



Legenda: u_{ij} representa observações 2D, X_i os pontos no sistema 3D, C_j o centro da câmera em diferentes momentos e u'_{ij} as re-projeções em pontos 2D. As linhas cheias representam o procedimento de projeção e as pontilhadas o de re-projeção. Fonte: (CHEN; CHEN; WANG, 2019)

funciona a uma determinada taxa. Assim, muitos pontos são obtidos, cada um sendo descritos pelo sistema ORB, mencionado anteriormente. O processo de relacionar os pontos em diferentes imagens é chamado de *Bundle Adjustment* (BA). Ele é responsável por minimizar o erro entre as observações 2D feitas na imagem capturada e a re-projeção em 2D dos pontos 3D de outras imagens capturadas anteriormente (mostrado na Figura 7).

Neste algoritmo também é considerada a distância Hamming entre os descritores dos pontos - quanto menor, mais provável os dois pontos 2D se referem a um mesmo ponto 3D.

3.1.3 Mapeamento

A etapa de **mapeamento** utiliza de algumas estruturas para seu bom funcionamento. A primeira delas é a eleição de *frames* específicos da câmera para serem *keyframes* (“quadros chave”, em tradução livre). Estes são os *frames* que possuem pontos que sustentam o mapa criado - a utilização de todos os disponíveis impossibilitaria o mapeamento de grandes áreas por problemas de memória. As condições para criá-lo exigem que:

1. Mais que 20 *frames* se passaram desde a última relocalização (algoritmo executado quando não se consegue encontrar a posição da câmera a partir dos pontos extraídos);
2. Mais que 20 frames se passaram desde a última inserção de um *keyframe*;
3. O *frame* candidato precisa ter ao menos 50 pontos detectados;
4. O *frame* candidato precisa ter 90% menos pontos do que um *frame* de referência.

Para relacionar os *keyframes*, utiliza-se um grafo chamado de *Covisibility Graph* (“grafo de co-visibilidade”, em tradução livre). Cada nó dele simboliza um *keyframe* e cada aresta é

ponderada com o número de observações que são compartilhadas entre dois *keyframes* (no mínimo 15). Como o *Covisibility Graph* pode se tornar muito denso e dificultar a execução de outros algoritmos, utiliza-se também um segundo grafo chamado de *Essential Graph* ("grafo essencial", em tradução livre). Ele possui o mesmo conceito do primeiro, porém retém nós que são conectados por arestas com peso muito maior que o anterior (em torno de 100 observações compartilhadas).

3.1.4 Relocalização

A etapa de **relocalização** é realizada transformando o *frame* atual em uma *bag of words* ("bolsa de palavras", em tradução livre) com implementação baseada na ferramenta DBoW2 (GALVEZ-LóPEZ; TARDOS, 2012). As "palavras visuais" são uma discretização do espaço de descritores (ou seja, exemplos deste), conhecido como vocabulário visual. Ele é criado previamente ao funcionamento do SLAM (*"off-line"*) e pode ser usado em diferentes ambientes, se o vocabulário criado é geral o suficiente. Para a relocalização, é feito um histograma das palavras visuais nas imagens atuais, buscando identificar qual distribuição é semelhante para *keyframes* já armazenados. Por escolha dos autores, são retornados todos os *keyframes* que possuem uma pontuação maior que 75% da melhor encontrada.

3.1.5 Fechamento de *loop*

A etapa de **fechamento de *loop*** é essencial para manter a consistência do mapa. O primeiro passo é detectar *keyframes* candidatos para este processo. Para isso, é calculado a similaridade entre a *bag of words* do *keyframe* em questão e de seus vizinhos presentes no *Covisibility Graph*. Para aceitar um *keyframe* candidato, é necessário que sejam encontrados três *keyframes* candidatos conectados no grafo. Dessa forma, é possível ter vários candidatos a *loop* se existem muitos lugares com aparência similar ao *keyframe* analisado. Ao realizar o fechamento de *loop*, os nós envolvidos são substituídos por arestas. Os pontos do mapa coincidentes são fundidos após serem reconhecidos pelo mesmo algoritmo de BA. Por fim, a posição da câmera é otimizada através do *Essential Graph*, distribuindo o erro de fechamento de loop ao longo do grafo.

3.2 Sequências utilizadas

Os dados coletados continham imagens de plantações de milho em diferentes estágios de crescimento. As sequências escolhidas para os testes foram obtidas em intervalos entre uma e duas semanas, localizadas no mesmo campo e na mesma fileira de uma plantação de milho. Uma visualização das sequências está mostrada na Figura 8, ordenadas cronologicamente.

É possível identificar algumas características dinâmicas do ambiente agrícola ao comparar as sequências da Figura 8. A primeira delas é a diferença de iluminação, dependendo do clima no dia da coleta das imagens. Também nota-se a variação na presença de folhas que se estendem em frente ao robô (tapando parcialmente o campo de visão da câmera) e que se estendem no chão. Além disso, a orientação da câmera se mostra diferente ao longo de uma sequência e entre elas, decorrência do controle manual utilizado no robô para captura dos dados e do terreno irregular.

Figura 8 – Sequências obtidas em uma única fileira de milho em crescimento ao longo de 3 meses

Fonte: DASLab

Espera-se que, em estágios mais avançados de crescimento, os algoritmos de mapeamento e navegação simultâneos tenham pior desempenho. Isso se deve à maior probabilidade de existência de objetos (como folhas, por exemplo) se deslocando e, portanto, a violação a suposição de ambientes estáticos se daria em maior grau (como argumentado na Seção 1.1). Para tanto, foram extraídos resultados para as nove sequências apresentadas na Figura 8, com o objetivo de comparar e verificar essa hipótese.

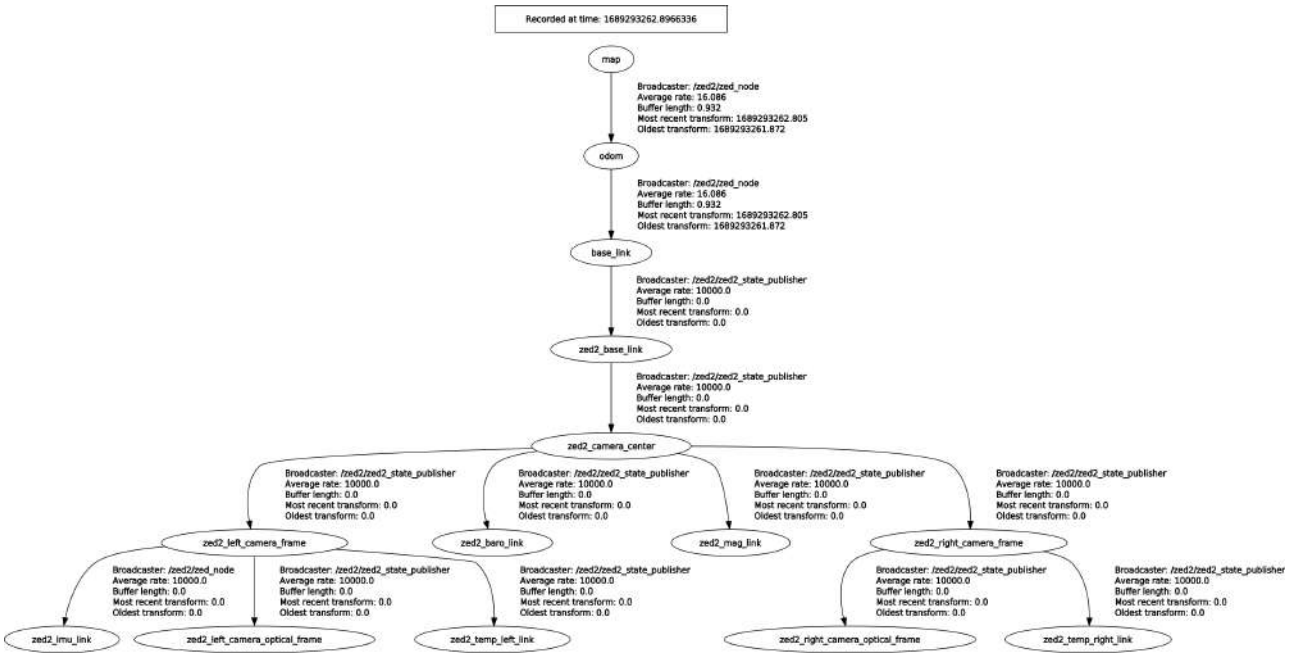
3.3 Aplicação do ORB-SLAM 2

Durante as pesquisas, foram encontradas duas diferentes implementações para o algoritmo ORB-SLAM 2. A primeira delas é uma versão *standalone* (RAULMUR, 2023), desenvolvida pelos autores do método. A segunda está integrada como um pacote ROS, chamado de orb_slam2_ros, e já inclui alguns exemplos de utilização com diferentes modelos de câmera (Mur-Artal et al., 2022b). Esta última foi escolhida, já que facilitou a integração entre o sistema de descompressão de dados (citado na Seção 2.1) e o algoritmo ORB-SLAM 2.

Para a utilização do pacote, algumas configurações são necessárias. Elas são definidas em um arquivo XML chamado *launch* (OPENROBOTICS, 2023b), uma estrutura genérica do ROS para inicializar diferentes nós com determinados parâmetros de forma facilitada. Nele, os tópicos de entrada das imagens do pacote foram remapeados para aqueles que são publicados durante a descompressão do arquivo SVO.

Além disso, habilitou-se o recebimento dos parâmetros intrínsecos da câmera pelo pacote com uma mensagem especializada para isso, a *sensor_msgs/CameraInfo* (OPENROBOTICS,

Figura 9 – Árvore de transformações do robô durante a coleta de dados



2023a). Ela especifica os comprimentos focais e centros principais da câmera, além do tamanho em pixels da imagem. Para isso, também foi remapeado o tópico relevante para aquele fornecido durante a descompressão do arquivo SVO.

Outra informação necessária é a localização da câmera em relação ao centro do robô - isso permite obter a posição do robô com dados obtidos a partir da câmera. Para isso, ROS conta com uma árvore de transformações, atualizada a partir de um tópico específico (*"/tf"*) (OPENROBOTICS, 2023d). Assim, é possível manter registro da relação entre todos os sistemas de coordenada do robô e externos a ele (como a origem da odometria ou de um mapa). Essa informação também foi gravada durante a coleta de dados (como mostrado na Figura 9) e foi reproduzida em tempo real durante os testes.

A Figura 9 mostra os principais sistemas de coordenada do sistema. O sistema *"base_link"* está fixo no robô e representa seu centro; o sistema *"zed2_camera_center"* representa o centro da câmera ZED2 e é passada como parâmetro para o pacote do ORB-SLAM 2. Além disso, o sistema *"map"* indica o centro do mapa gerado (a ser atualizado pelo algoritmo de localização) e o sistema *"odom"* indica a origem da odometria do robô.

3.4 Obtenção de métricas

Para medir o desempenho do algoritmo ORB-SLAM 2, utilizou-se a distância percorrida pelo robô até a falha do sistema de localização. Para identificá-la, utilizou-se de dois métodos principais. O primeiro método é a utilização da imagem de *debug* fornecida pelo pacote através do tópico */slam/debug_image*. Esta imagem contém as características identificadas a cada frame da câmera e também informa visualmente quando o sistema falha. Ambas as situações estão mostradas nas Figuras 10 e 11.

Figura 10 – Imagem de *debug* durante o funcionamento do algortimo

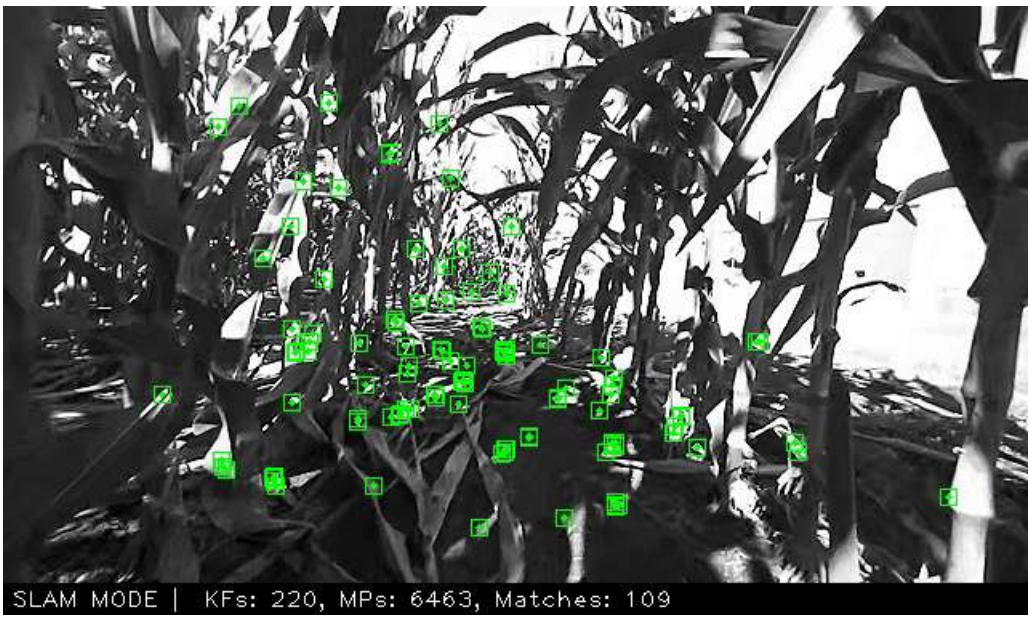


Figura 11 – Imagem de *debug* quando há falha do algoritmo. Nesse caso, há colisão do robô com a plantação

O segundo método é a verificação da atualização da posição do robô pelo algoritmo ORB-SLAM 2 através do tópico */slam/pose* (versão stereo) ou do tópico */orb_slam2_mono/pose* (versão monocular). Quando há falha do sistema de localização, a posição do robô não é mais atualizada.

Assim, o ponto de início para o registro da distância percorrida acontece assim que o robô inicia seu movimento e inicializa as primeiras *features*, como mostrado na Figura 10. O ponto de finalização da medição ocorre quando a posição do robô não é mais atualizada ao ocorrer falha no sistema de navegação.

Na avaliação das sequências, três distâncias foram utilizadas. As duas primeiras delas são as distâncias percorridas pelo robô de acordo com o sistema SLAM monocular e stereo. A terceira é obtida de acordo com o algoritmo de odometria visual nativo da câmera ZED2. Elas são calculadas a partir da soma da distância euclidiana entre as posições consecutivas do robô obtidas por ambos os algoritmos (SLAM e Odometria Visual). Para cada sequência, 6 medições foram realizadas, exceto no caso onde uma oclusão foi observada logo no início da fileira, caso em que apenas 3 medições foram realizadas. A média das medições para cada tipo de sistema e tecnologia também foi calculada.

Além disso, também utilizou-se a ferramenta *RViz* (OPENROBOTICS, 2023c) para realizar uma visualização qualitativa da trajetória do robô informada pelo sistema SLAM e pela Odometria Visual da câmera ZED2. Ela também está disponível como um pacote ROS e é amplamente utilizada para observação dos dados de um sistema robótico.

3.5 Resultados e discussões

Ao executar o sistema SLAM com os dados da sequência do dia 08 de setembro, os dados foram visualizados na ferramenta *RViz*, obtendo a Figura 12.

Em uma análise qualitativa inicial, pode-se perceber pela Figura 12 que a trajetória indicada pela odometria visual difere em direção do que aquela gerada pelo sistema SLAM. Apesar disso, ambas são retilíneas e aparentemente coerentes. Supõe-se que o sistema SLAM gera uma trajetória que não está alinhada em relação ao eixo de coordenadas fixo na ferramenta *RViz*. Entretanto, como as distâncias são calculadas usando a distância euclidiana entre posições consecutivas do robô gerada por cada algoritmo, a inclinação não deve afetar os dados quantitativos.

Ao realizar as medições de distância de acordo com os métodos descritos nas seções anteriores, obteve-se os dados mostrados na Tabela 1. A primeira observação que ela sugere é que algumas sequências tem distâncias muito curtas (como aquelas indicadas com data em vermelho). Através de inspeção das sequências, observou-se que, nesses casos, a visão da câmera do robô foi bloqueada logo no início da fileira, impedindo a extração de *features* e consequentemente, levando à falha do sistema SLAM. Além disso, as demais sequências eventualmente também falharam devido a oclusões, oscilações do robô ou por choque do robô com a cultura.

Figura 12 – Ferramenta RViz durante a execução do SLAM. O número 1 indica a posição do robô (seta vermelha) segundo a Odometria Visual e a *pointcloud* da câmera ZED2; o número 2 indica a posição do robô (seta rosa) e *features* (pontos brancos) segundo o algoritmo SLAM

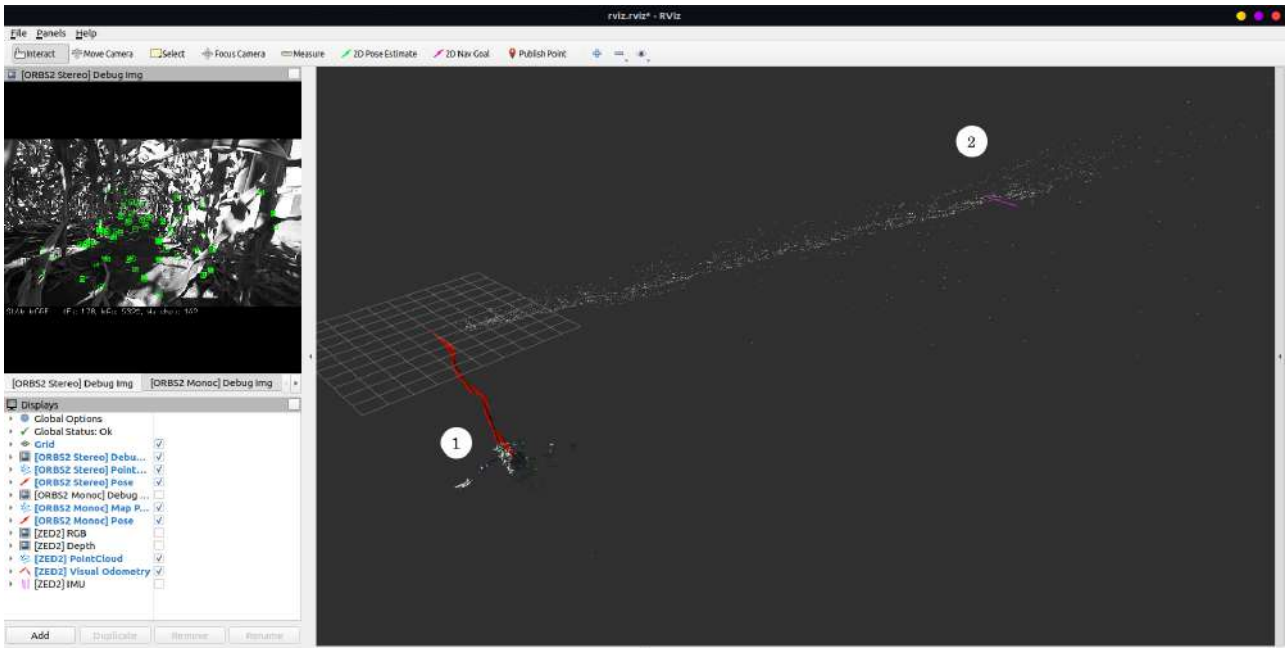


Tabela 1 – Distância média percorrida pelo robô até a falha do sistema SLAM. Datas indicadas em colorido simbolizam oclusões no início da sequência

	[ZED] Stereo (m)	**[SLAM] Stereo (m)**	**[ZED] Mono (m)**	**[SLAM] Mono (m)**
20/09	4,74	11,50	0,83	2,24
08/09	10,95	26,06	25,09	22,20
01/09	0,66	1,55	0,50	0,42
15/08	8,27	19,46	0,79	0,60
04/08	1,17	2,77	1,43	0,96
21/07	3,36	8,16	2,33	1,16
14/07	2,49	6,11	2,00	1,38
05/07	3,20	7,67	0,98	0,95
29/06	7,15	17,28	4,96	3,37

É também possível verificar que o sistema SLAM Stereo possui maior robustez que o sistema SLAM Monocular. Isso era esperado, já que o sistema stereo possui mais informações para obter *features*, em comparação com o sistema monocular. Em média, excluindo as sequências com oclusões logo no início, o sistema SLAM Stereo funcionou durante 7,71m (ZED)/18,37m (SLAM). Em contrapartida, o sistema SLAM Monocular funcionou durante 2,89m (ZED)/2,81m (SLAM).

Ademais, pode-se verificar que a distância calculada para o sistema de Odometria Visual da câmera ZED2 e aquela para o sistema SLAM diferem de forma significativa. Entende-se que o sistema SLAM não é preciso no ambiente em questão, já que as *features* identificadas em um *frame* mudam de posição no *frame* seguinte.

Segundo os dados quantitativos, a hipótese de que sequências com plantio em estágio menos avançados de crescimento teriam melhor desempenho não pôde ser exatamente confirmada. Devido ao dinamicismo do ambiente agrícola, desde que existam folhas na frente do robô, oclusões serão frequentes e o sistema de SLAM provavelmente falhará, independente do estágio. Assim, supõe-se que sistemas SLAM podem funcionar melhor em cenários agrícolas onde não há presença de longas folhas nas plantas.

4 DETECTOR DE OBJETOS

A partir dos resultados pouco promissores com a aplicação direta de SLAM, buscou-se desenvolver algoritmos que pudessem auxiliar na sua aplicação para o ambiente agrícola.

Como argumentado na Seção 3.2, sistemas dinâmicos violam o princípio de ambientes estáticos, que é a base de sistemas SLAM. Com o objetivo de mitigar esse problema, decidiu-se por construir um detector de objetos utilizando Deep Learning, aderindo a direção apontada pela literatura (XIAO et al., 2019).

Um detector de objetos consiste em um modelo capaz de predizer caixas ao redor dos objetos alvo. Para esta tarefa, definiu-se o objeto alvo como a parte superior das plantas, supondo que elas sejam a parte mais dinâmica capturada pela câmera. Assim, posteriormente, poderia-se remover os pontos da etapa descrita na Seção 3.1.1 que estivessem dentro das detecções do modelo.

4.1 Deep Learning

O primeiro passo para desenvolver esse detector foi buscar um entendimento básico sobre redes neurais. Para isso, utilizou-se de alguns cursos de Deep Learning oferecidos na plataforma online Coursera (COURSERA, 2021) pela organização DeepLearning.AI (DEEPLEARNING.AI, 2021a).

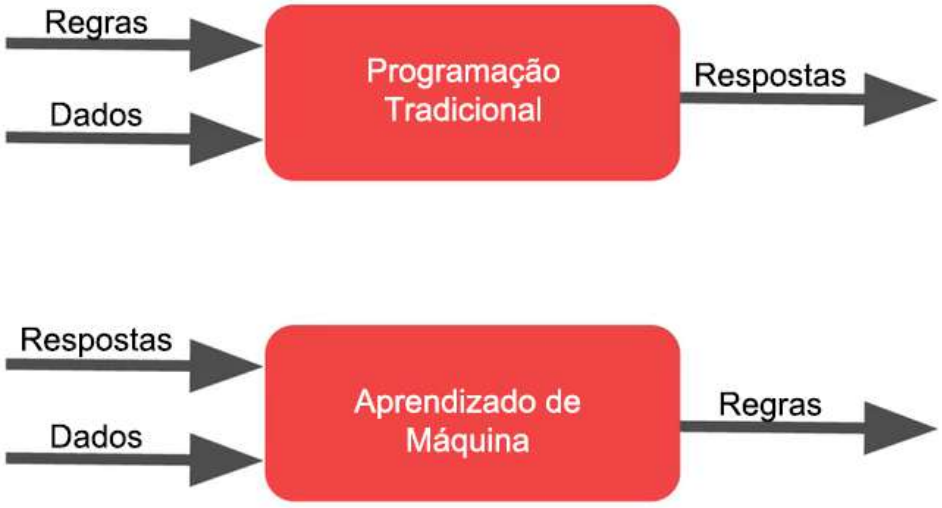
Realizou-se três cursos teóricos pertencentes ao *"Programa de Cursos Integrados sobre Aprendizagem Profunda"*, sendo eles: *"Neural Networks and Deep Learning"*, *"Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization"* e *"Convolutional Neural Networks"* (DEEPLEARNING.AI, 2021c). A seguir, parte dos conhecimentos obtidos são explicados.

4.1.1 Conhecimentos básicos

Na construção de algoritmos tradicionais, informações de entrada e o estabelecimento de regras são essenciais para obter os resultados esperados. No entanto, os programas que envolvem aprendizado de máquina se baseiam em um outro paradigma: as regras que regem o conjunto de dados em questão não é conhecido. Nessa abordagem, o programa recebe informações de entrada e aquelas esperadas como saída e é responsável por entender como esses dois conjuntos se relacionam (Figura 13). Além disso, busca generalizar as relações encontradas para possibilitar a análise futura de dados desconhecidos.

A unidade básica de processamento em uma rede neural é chamada de neurônio (também chamado de *"perceptron"*), cujo nome é proveniente da inspiração biológica para a técnica de aprendizado aqui descrita. Todo neurônio é ligado a um conjunto de dados de entrada. No treinamento da rede, cada exemplo desse grupo de informações já está associado a uma resposta

Figura 13 – Diferenças entre programação tradicional e aprendizado de máquina



Fonte: *Introduction to TensorFlow for Artificial Intelligence, Machine Learning, and Deep Learning* (DEEPLEARNING.AI, 2021b) (adaptado pelo autor)

esperada. Em uma aplicação de classificação de imagens, por exemplo, o conjunto de dados é composto por várias figuras diferentes, com suas respectivas classes já anotadas separadamente.

Seja $A^{[0]}$ a matriz de dados de entrada de dimensões $n_0 \times m$, onde n_0 é o número de informações por exemplo e m é o número de exemplos do conjunto de dados em questão. A ela, é aplicada uma operação linear através das matrizes de parâmetros treináveis $W^{[1]}$ (dimensões $n_1 \times n_0$) e $B^{[1]}$ (dimensões $n_1 \times 1$), onde n_1 é o número de neurônios da primeira camada. A Equação 4.1 mostra a expressão para $Z^{[1]}$, resultado da operação descrita. Vale ressaltar que, da forma apresentada, está implícita uma técnica chamada de *broadcasting*, na qual a soma da matriz $B^{[1]}$ é feita coluna a coluna. Apesar da notação não ser matematicamente precisa, é frequentemente utilizada na sintaxe de linguagens de programação e por isso, será utilizada durante esta seção.

$$Z^{[1]}_{n_1 \times m} = W^{[1]}_{n_1 \times n_0} A^{[0]}_{n_0 \times m} + B^{[1]}_{n_1 \times 1} \tag{4.1}$$

Em seguida, o resultado da Equação 4.1 é aplicado em uma função de ativação, não linear (representada por *g(z)*). Esse passo é muito importante, já que permite que a rede neural aprenda padrões complicados presentes no relacionamento entre os dados de entrada e saída. Existem vários tipos como Sigmoid, Tanh, ReLU e Leaky ReLU. Uma das mais utilizadas é a função ReLU, cuja expressão é dada pela Equação 4.2 (aplicada em cada elemento da matriz Z). Em um modelo de rede, pode-se adotar diferentes funções de ativação para diferentes camadas.

$$g(z) = max(0, z) \tag{4.2}$$

A aplicação da função de ativação em todos os elementos da matriz $Z^{[1]}$ produz a matriz $A^{[1]}$, que é efetivamente a saída da primeira camada da rede. Em uma Rede Completamente

Conectada (FNC, *"Fully Connected Layer"*), a saída de cada neurônio de uma camada anterior está ligada à subsequente. Dessa forma, pode-se generalizar as Equações 4.1 e 4.2 de forma a obter as Equações 4.3 e 4.4, para toda camada L.

$$Z^{[L]}_{n_L \times m} = W^{[L]}_{n_L \times n_{(L-1)}} A^{[L-1]}_{n_{(L-1)} \times m} + B^{[L]}_{n_L \times 1} \tag{4.3}$$

$$A^{[L]} = g(Z^{[L]}) = max_Z(0, z) \tag{4.4}$$

Os resultados de uma rede são comparados às informações presentes no *dataset* de treino para uma determinada tarefa. Para medir o quanto os dados estão condizentes com aqueles esperados, é utilizada uma função de erro específica. Para realizar a classificação de dados em apenas duas conjuntos (também chamado de *Logistic Regression*, "Regressão Logística" em tradução livre), pode-se utilizar a função de erro $E(\hat{y}, y)$ descrita na Equação 4.5. A variável $\hat{y}$ representa o resultado da rede, enquanto que a variável y o valor esperado. Quanto mais $\hat{y}$ se aproxima de y (que pode assumir os valores 0 ou 1), menor é o valor da função E. Vale ressaltar que a Equação 4.5 é específica para a tarefa em questão e pode variar para cada problema.

$$E(\hat{y}, y) = -(ylog(\hat{y}) + (1 - y)log(1 - \hat{y})) \tag{4.5}$$

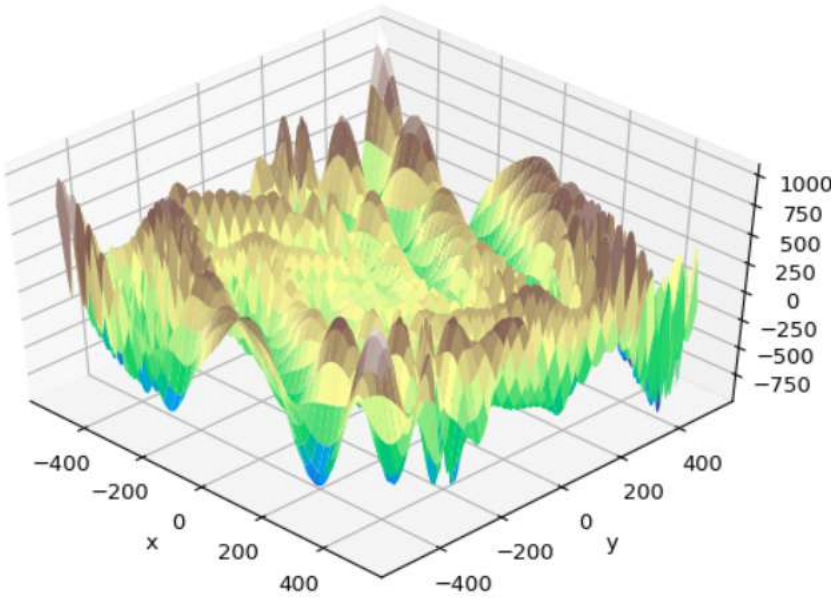
Através das Equações 4.3 e 4.4, pode-se deduzir analiticamente (através do cálculo de derivadas) qual é o impacto de cada parâmetro na função de custo para cada exemplo do *dataset*. Esse processo é chamado de *backpropagation* ("propagação reversa", em tradução livre) e é aliado a uma técnica de otimização global. Uma representação hipotética da função de erro (onde os eixos x e y seriam possíveis parâmetros) está retratada na Figura 14, onde é possível identificar visualmente vários mínimos locais. O objetivo é encontrar os valores de parâmetros em um espaço vetorial com muitas dimensões, que geram o menor erro médio global possível da rede.

Um dos métodos de otimização de parâmetros é chamado de *Stochastic gradient descent* ("Descida gradiente estocástica", em tradução livre). Seja dw_p e db_p as derivadas da função E em relação aos respectivos p-ésimos parâmetros. A atualização dos parâmetros em questão é realizada de acordo com as Equações 4.6, onde α é a taxa de aprendizado, um hiperparâmetro ajustado manualmente.

$$\begin{aligned} w_p &= w_p - \alpha dw_p \\ b_p &= b_p - \alpha db_p \end{aligned} \tag{4.6}$$

Um outro algoritmo de otimização é chamado de *Gradient descent with momentum* ("Descida gradiente com momento", em tradução livre). A intenção desse método é de diminuir as possíveis oscilações no treinamento dos parâmetros em busca do mínimo custo global e procurar aumentar o impulso nessa direção. Para isso, duas novas variáveis são computadas, dadas pelas Equações 4.7, onde β_1 é um hiperparâmetro definido manualmente.

Figura 14 – Função de erro hipotética



Fonte: *Everything You Need to Know about Gradient Descent Applied to Neural Networks* (DURáN, 2019)

$$\begin{aligned} V_{dw} &= \beta_1 V_{dw} + (1-\beta_1) d_w \\ V_{db} &= \beta_1 V_{db} + (1-\beta_1) d_b \end{aligned} \tag{4.7}$$

A atualização iterativa dos parâmetros é feito de forma diferente, segundo as Equações 4.8.

$$\begin{aligned} w_p &= w_p - \alpha V_{dw} \\ b_p &= b_p - \alpha V_{db} \end{aligned} \tag{4.8}$$

Um dos métodos de otimização mais utilizados em diversas aplicações diferentes de redes neurais é chamado de *Adam*. Nele, o conjunto de Equações 4.7 é utilizado juntamente com outras duas variáveis, definidas nas Equações 4.9, onde β_2 é outro hiperparâmetro escolhido.

$$\begin{aligned} S_{dw} &= \beta_2 S_{dw} + (1-\beta_2) d_w^2 \\ S_{db} &= \beta_2 S_{db} + (1-\beta_2) d_b^2 \end{aligned} \tag{4.9}$$

Os parâmetros são atualizados a cada iteração t de acordo com o conjunto de Equações 4.10.

$$w_p = w_p - \alpha \frac{V_{dw}}{\sqrt{S_{dw}}} \frac{\sqrt{(1-\beta_2^t)}}{(1-\beta_1^t)}$$
$$b_p = b_p - \alpha \frac{V_{dw}}{\sqrt{S_{dw}}} \frac{\sqrt{(1-\beta_2^t)}}{(1-\beta_1^t)} \tag{4.10}$$

4.1.2 Redes Neurais Convolucionais

Durante o desenvolvimento das arquiteturas de *Deep Learning*, percebeu-se que, no trabalho com imagens, a utilização exclusiva de FNC (*"Fully Connected Layers"* - "Camadas Completamente Conectadas" em tradução livre) se mostra menos eficiente para o processamento de figuras tanto analisando os resultados quanto o esforço computacional necessário.

A alternativa foi utilizar uma nova operação: a convolução, que é baseada na utilização de filtros (também chamados de *kernel*). Estes são matrizes com dimensões bem menores que as da imagem, desenvolvidos para ressaltar alguma característica. Dessa forma, exemplos clássicos de filtros são aqueles capazes de detectar linhas verticais ou linhas horizontais, utilizados em métodos de Visão Computacional. No contexto da aplicação de redes neurais, o conteúdo das matrizes não é desenvolvido manualmente, mas é aprendido automaticamente através do treinamento da rede. Ou seja, o algoritmo é quem "decide" quais são as características relevantes para que o seu resultado seja compatível com aquele esperado.

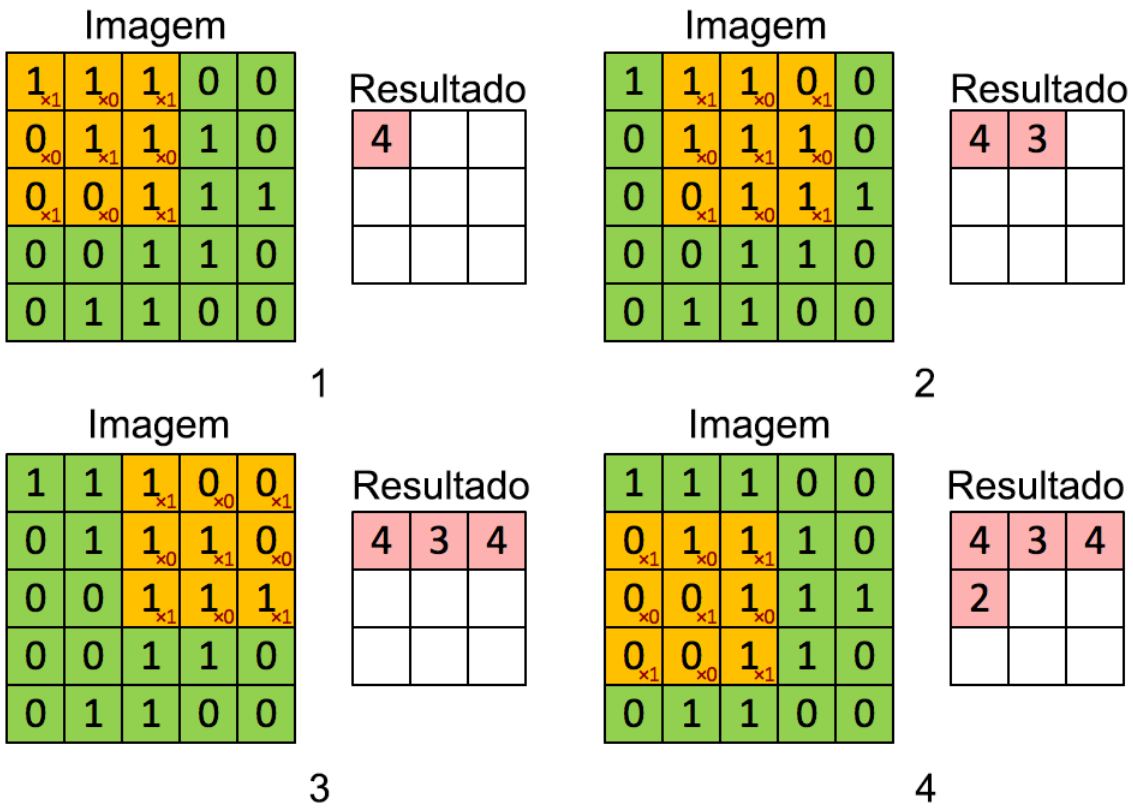
4.1.2.1 Operação de convolução

A convolução é um processo que se utiliza de várias multiplicações entre os valores da imagem e do *kernel*. Suponha que exista uma imagem de dimensões 5x5 e pretende-se aplicar um filtro de dimensões 3x3. Os valores das células do filtro são multiplicados um a um com seus correspondentes da imagem e somados, gerando uma nova matriz que, se interpretada como outra imagem, tem características importantes ressaltadas. Esse processo está retratado de forma simplificada na Figura 15.

Nela, pode-se perceber quatro das etapas da convolução. Na matriz "Imagem", os números maiores representam a intensidade dos pixels, enquanto que os menores (localizados no canto inferior direito), representam os valores do *kernel*. A soma de todas as multiplicações em cada etapa geram os números retratados na matriz "Resultado". Vale ressaltar que os números utilizados não representam uma imagem real e foram escolhidos apenas para demonstração.

Essa operação, apesar de extrair características relevantes da imagem, tem o efeito colateral de a diminuir, impedindo sua utilização consecutiva. Esse problema pode ser resolvido utilizando uma técnica adicional, chamada de *padding* ("preenchimento", em tradução livre). A ideia é preencher as bordas da imagem com valores nulos de forma que o processo de convolução não diminua os mapas de características em redes profundas.

Figura 15 – Demonstração de quatro passos da operação de convolução



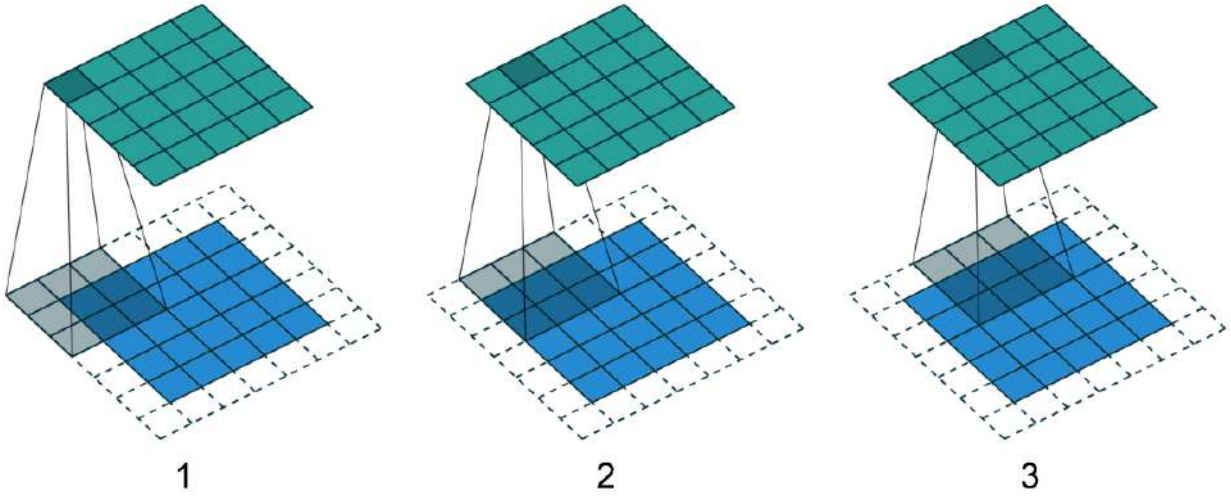
Fonte: *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* (SAHA, 2018) (adaptado pelo autor)

Duas formas são comumente utilizadas para se referir à aplicação de *padding*. A primeira é chamada de *"valid"*, que significa que nenhum preenchimento será realizado e algumas células do mapa de característica podem ser ignoradas. A segunda é chamada de *"same"*, que busca realizar o preenchimento de forma igual em todos os lados da imagem. Se alguma das dimensões é ímpar, mais uma dimensão de valores nulos é criada, para o aproveitamento de toda a informação disponível durante a passagem do filtro.

A aplicação de *same padding* está mostrada na Figura 16, na qual o preenchimento está representado pelos quadrados pontilhados, a imagem pelos quadrados em azul, o resultado da operação, em verde, e a sombra mostra onde o *kernel* está sendo utilizado a cada passo. No caso, é aplicado um filtro 3x3 em uma imagem 5x5 e o resultado também é um mapa de características com dimensões 5x5.

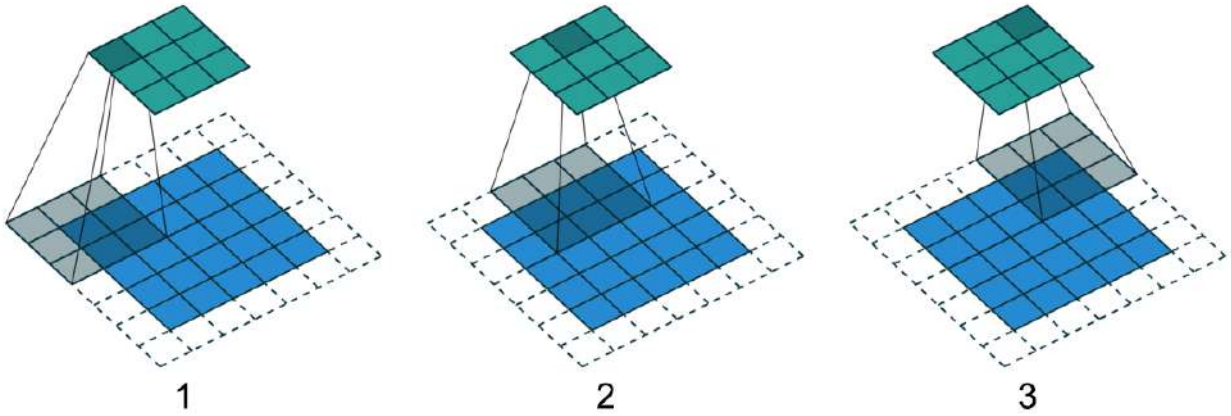
Por fim, há também a possibilidade de aplicar filtros utilizando padrões diferentes de espaçamento. A essa característica se dá o nome de *stride* (traduzido para "passo largo"). Nas Figuras 15 e 16, o valor do *stride* é 1, já que o filtro se desloca apenas uma célula em uma dimensão a cada passo. Na Figura 17, são apresentados três passos de uma convolução com *stride* igual a 2 e *padding* igual a 1. De fato, entre consecutivos passos, o *kernel* se movimenta duas células da imagem. Vale ressaltar que o resultado da operação é menor do que a imagem, já que a passagem do filtro é "menos detalhista".

Figura 16 – Operação de convolução com utilização de *padding*



Fonte: *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* (SAHA, 2018) (adaptado pelo autor)

Figura 17 – Operação de convolução com utilização de *stride* e *padding*



Fonte: *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* (SAHA, 2018) (adaptado pelo autor)

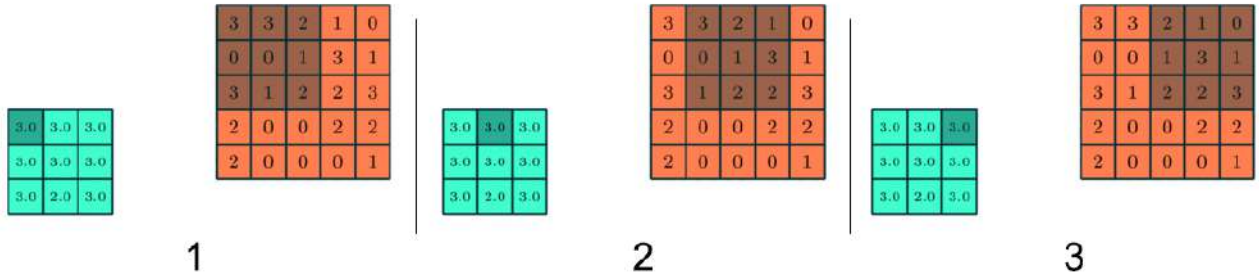
O processo de convolução pode ser realizado com (ou sem) a inclusão das técnicas de *padding* e *stride*, se tornando uma escolha de modelagem durante o projeto da rede. É possível prever, no entanto, qual será o tamanho da resposta de uma convolução conhecendo os parâmetros desejados.

Seja $n \times n$ as dimensões da imagem na qual será a aplicada a convolução e $f \times f$ as dimensões do filtro. Além disso, seja p o valor de *padding* e s o valor de *stride* aplicados, o mapa de características resultante da operação terá dimensões $r \times r$, onde r é dado pela Equação 4.11.

$$r = \frac{n + 2p - f}{s} + 1 \tag{4.11}$$

Por fim, a convolução é uma operação que pode ser aplicada em volumes. Isso é útil principalmente ao considerar imagens coloridas, que possuem canais registrando a intensidade dos *pixels* para cada cor. No caso de imagens RGB, existem três canais distintos para cores:

Figura 18 – Demonstração de três passos da operação *pooling*



Fonte: *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* (SAHA, 2018) (adaptado pelo autor)

vermelho, verde e azul e, portanto, sua "terceira dimensão" é igual a 3.

A aplicação de um filtro em uma camada de convolução deve satisfazer uma regra: a terceira dimensão do *kernel* aplicado deve ser ser igual ao da camada anterior, seja ela uma imagem ou um mapa de características resultante de outra convolução. A terceira dimensão da saída de uma camada deste tipo é o número de filtros aplicados. Ou seja, se forem aplicados 5 filtros, a camada resultante terá dimensões $r \times r \times 5$.

4.1.2.2 Operação de pooling

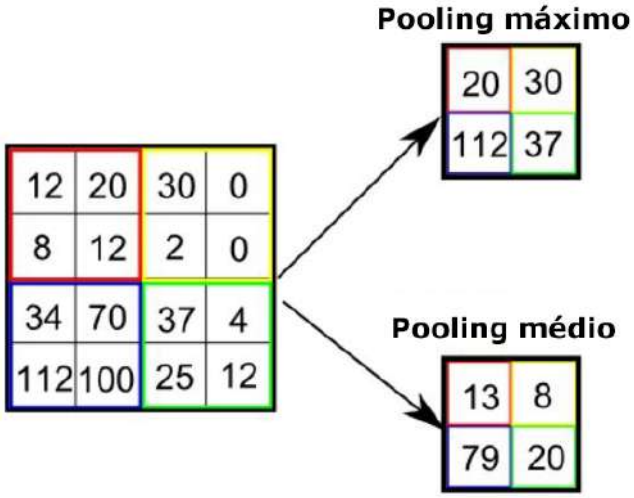
Uma operação importante no desenvolvimento de Redes Neurais Convolucionais é chamada de *pooling* ("junção", em tradução livre). Na produção de um modelo de *Deep Learning*, frequentemente é necessário diminuir as camadas para tornar a extração de características mais robusta e também para acelerar o processo computacional. Para implementar essa ideia são utilizados filtros, assim como para a operação de convolução e, portanto, a técnica de *stride* (explicada na Seção 4.1.2.1) é muito utilizada.

Existem dois tipos de *pooling* comumente utilizados: *max pooling* e *average pooling*. O primeiro (mostrado na Figura 18) consiste no cálculo do maior valor (máximo) das células abordadas pelo filtro. O segundo se utiliza da mesma ideia, mas o cálculo feito é de média dos valores das células incluídas pelo *kernel*.

Para comparação, seja um mapa de características de dimensões 4×4 e um filtro *pooling* de tamanho 2×2. Utilizando *stride* igual a 2, obtém-se diferentes resultados com os diferentes tipos de *pooling*, como mostrado na Figura 19. Há uma preferência geral dos pesquisadores em utilizar a versão na qual há o cálculo do máximo valor, ao invés da média.

Vale ressaltar que, no caso da operação de *pooling*, a aplicação dos filtros não requer treino de nenhum parâmetro. Isso se deve ao fato de que a aplicação dos mesmos se dá pela obtenção dos valores máximos ou médios dos dados da camada em questão, sem necessidade de ter algum aprendizado.

Figura 19 – Diferenças entre *max pooling* e *average pooling*



Fonte: *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* (SAHA, 2018) - tradução livre

4.1.3 Métricas para classificação e detecção

Em uma tarefa de classificação, espera-se que um algoritmo de *Deep Learning* seja capaz de avaliar, dentre um conjunto pré-determinado, a qual classe a imagem em questão predominantemente pertence. A primeira dificuldade é a de transformar os mapas de características (frequentemente dispostos de forma 2D ou até mesmo 3D) em números que simbolizem as classes de interesse. Para resolver esse problema, duas ferramentas são utilizadas: *"flatten layer"* ("camada de achatamento", em tradução livre) e *softmax layer* ("camada softmax", em tradução livre).

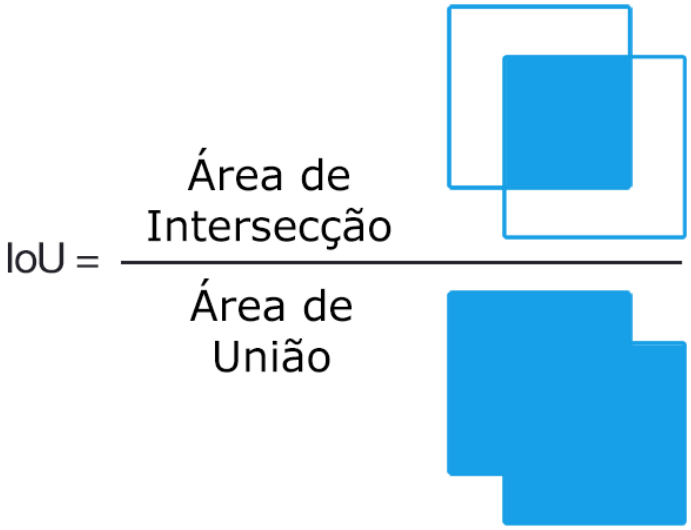
A primeira deles é, na prática, apenas o "enfileiramento" das células de um mapa de características. Ou seja, os valores armazenados em uma matriz 2D ou 3D são achatadas em um vetor de uma dimensão. Dessa forma, esses valores podem ou não ser alimentados em camadas completamente conectadas, gerando processamento adicional que é adotado em alguns modelos. O essencial é que essa mudança de formato permite que seja utilizada como entrada da camada *softmax*, cujo número de neurônios é igual ao número de classes pré-determinadas que se deseja realizar a classificação.

Softmax é o nome dado para uma função de ativação que é comumente utilizada para transformar os valores dos mapas de características em probabilidades. Seja n o número de classes pré-determinadas e z o valor resultante da propagação da rede para cada neurônio da camada *softmax*, então o valor de saída ϕ do j-ésimo neurônio é dado pela expressão da Equação 4.12.

$$\phi_j = \frac{\exp(z_j)}{\sum_{i=1}^{n} \exp(z_i)} \tag{4.12}$$

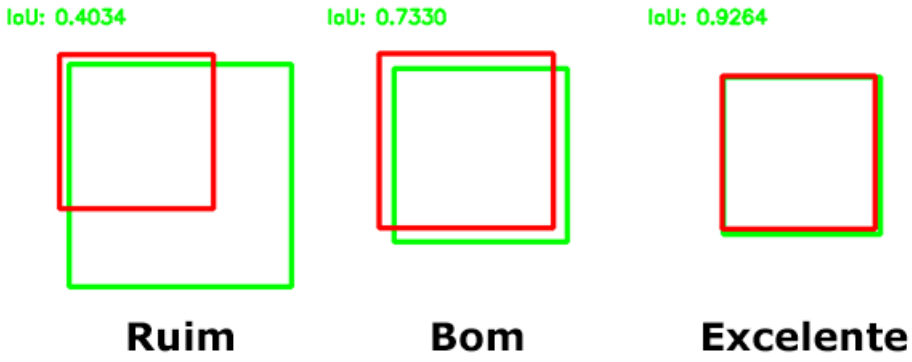
Através do processo descrito, o processamento convolucional de uma imagem pode ser traduzido em valores que simbolizam a probabilidade daquela imagem pertencer a certa classe. Sabendo que uma imagem de entrada faz parte da c-ésima classe, então espera-se que o c-ésimo neurônio da camada *softmax* se aproxime o máximo possível de 1, enquanto que os demais fiquem

Figura 20 – Representação visual do cálculo do índice *IoU*



Fonte: *Intersection over Union (IoU) for object detection* (ROSEBROCK, 2016) - tradução livre

Figura 21 – Análise da detecção com base no índice IoU



Fonte: *Intersection over Union (IoU) for object detection* (ROSEBROCK, 2016) - tradução livre

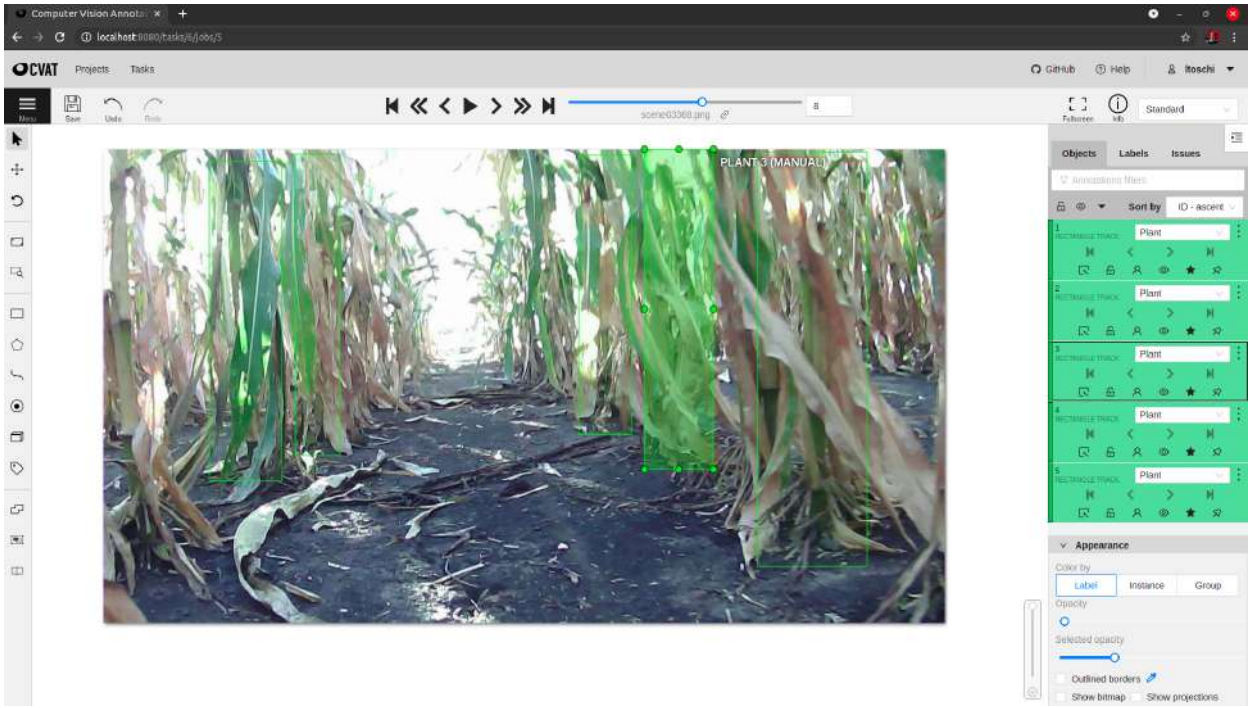
próximos a 0. A diferença entre o valor resultante e o esperado é contabilizado pela rede neural para realizar o aprendizado de parâmetros.

Em uma tarefa de detecção de objetos, é necessária uma abordagem diferente. A rede neural deve ser capaz de elaborar caixas em volta dos objetos da imagem e compará-las com aquelas disponíveis no *dataset* de treino. Para realizar essa comparação, é utilizado o índice IoU - *Intersection over Union* ("Intersecção sobre União", em tradução livre), também chamado de Índice de Jaccard. Como mostrado na Figura 20, ele é dado pela divisão entre a área de intersecção pela área de união de duas caixas.

Se a área de intersecção for igual a de união, o índice IoU é igual a 1, que indica um encaixe perfeito entre duas caixas. Se a área de intersecção é nula, significa que o índice também o é, de forma que as caixas estão completamente isoladas uma da outra. Vale ressaltar que a área de união nunca é nula, já que só faz sentido aplicar essa métrica com caixas de áreas não nulas.

Um exemplo de análise do índice de Jaccard está retratado na Figura 21. Seja a caixa verde aquela representada no *dataset* e a vermelha aquela gerada pela rede neural. Da esquerda

Figura 22 – Interface da ferramenta CVAT



para direita, existem exemplos cada vez melhores de encaixe e os respectivos valores de IoU. As caixas com melhores IoU são as escolhidas como resultados da tarefa de detecção de objetos.

4.2 Imagens e construção de dataset

Para o treinamento de um modelo detector de objetos, é necessária a construção de um dataset rotulado. Nele, deve-se ter imagens (do domínio de aplicação) com anotações indicando os objetos-alvo. Para esse processo, utilizou-se a ferramenta CVAT (*Computer Vision Annotation Tool*, "Ferramenta de Anotação para Visão Computacional", em tradução livre) (SEKACHEV et al., 2020). Uma imagem do uso da interface da ferramenta está mostrada na Figura 22.

O dataset final conta com mil imagens rotuladas manualmente a partir dos dados do robô TerraSentia. Dividiu-se o *dataset* em duas partes: a primeira possui 750 imagens e foi utilizada para o treinamento da rede; a segunda possui 250 imagens e foi utilizada para validar os resultados obtidos.

Vale ressaltar que as 1000 imagens fazem parte de um único vídeo completo gravado pelo robô TerraSentia. Uma possível melhoria deste projeto é a inclusão de outras situações em que o robô possa estar exposto como curvas, entradas e saídas de fileiras da cultura. Outra sugestão é a expansão do *dataset* para diferentes tipos de plantação, de forma a verificar se apenas um modelo é necessário para identificar diferentes espécies de plantas.

Na Figura 23, é possível observar uma amostra de algumas imagens da porção de validação do *dataset*. Nela, os resultados esperados do projeto estão explicitados com as caixas rosas e é possível entender o que o termo *"porção superior"* significa: se refere à região acima da raiz, próxima do caule principal. A última linha de imagens mostra alguns exemplos da dinamicidade

Figura 23 – Resultado esperado do detector de objetos

As predições esperadas estão marcadas em rosa

do ambiente agrícola, onde folhas tapam parcialmente a visão da câmera do robô.

4.3 Revisão Bibliográfica de técnicas de Deep Learning para SLAMIDE

Com o dataset desenvolvido, o próximo passo foi definir uma arquitetura para treinar o detector de objetos. Para isso, realizou-se uma revisão bibliográfica com métodos para SLAMIDE que utilizam Deep Learning. A seguir, alguns desses métodos serão descritos brevemente.

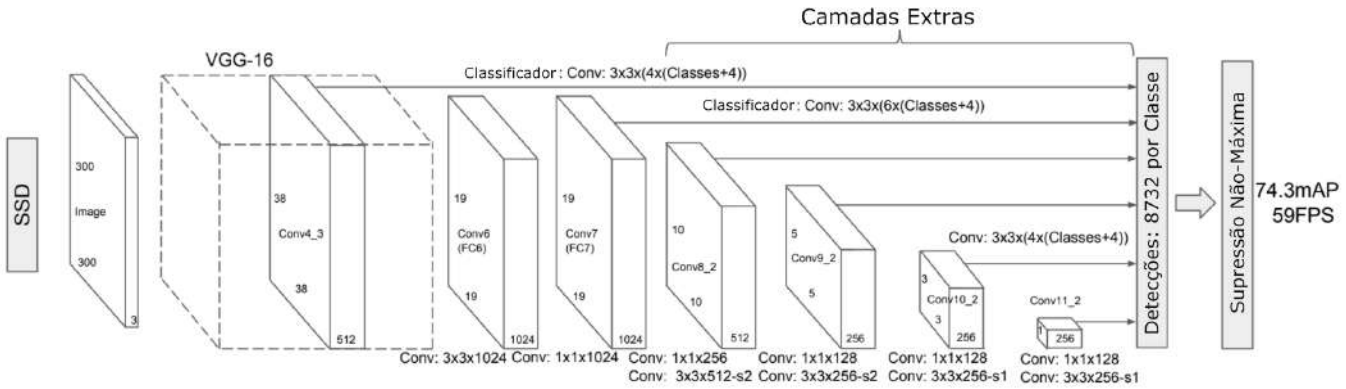
4.3.1 *Dynamic-SLAM*

O primeiro método apresentado é chamado de *Dynamic-SLAM* (XIAO et al., 2019). Para realização da detecção de objetos, é utilizada a rede neural *Single Shot Detector* (SSD) (LIU et al., 2016), cujo modelo está representado na Figura 24. Ele consiste na utilização de duas operações básicas: *convolução* e *pooling* (descritas nas seções 4.1.2.1 e 4.1.2.2, respectivamente).

As primeiras camadas da rede fazem parte uma rede neural pré-treinada, chamada de VGG-16 (SIMONYAN; ZISSERMAN, 2015). Acima dela, são colocadas mais camadas cujos resultados são agregados como predições finais da rede - essa estrutura permite que sejam detectadas características de diferentes tamanhos em relação à imagem. O formato dos objetos é estabelecido manualmente a partir da definição de proporções para k caixas padrão baseadas nos objetos que se quer fazer a identificação.

De forma convolucional, os objetos são referenciados pela rede a partir da posição de seu centro em relação à cada uma das células da imagem. Para cada célula de uma camada de classificação de tamanho $m \times n$, as k caixas padrões geram pontuações para as c classes

Figura 24 – Modelo da rede SSD



Fonte: *SSD: Single shot multibox detector* (LIU et al., 2016) - tradução livre

pré-determinadas mais quatro números que representam o *offset* da caixa em relação ao centro do quadrante. Logo, para cada célula, são geradas $(c + 4)kmn$ saídas.

Em uma rede com tarefas de localização e classificação, ambas são levadas em consideração para descrever o seu desempenho. Para a primeira, as caixas geradas são comparadas àquelas do *dataset* através do índice IoU (*Intersection over Union*, também chamado de índice de Jaccard). Para a segunda, o resultado da camada de classificação (*softmax*) para cada caixa padrão é a probabilidade do objeto fazer parte de certa classe. O tratamento da saída dessa rede neural está explicitado na Seção 4.1.3.

Para apenas uma imagem, o método descrito acima realiza a geração de muitas predições de classificação e localização. Para obter a melhor possível como saída da rede, é realizado um processo chamado de *Non-Maximum Suppresion*, no qual apenas as caixas com os maiores índices (citados anteriormente) são mantidas e as demais, removidas.

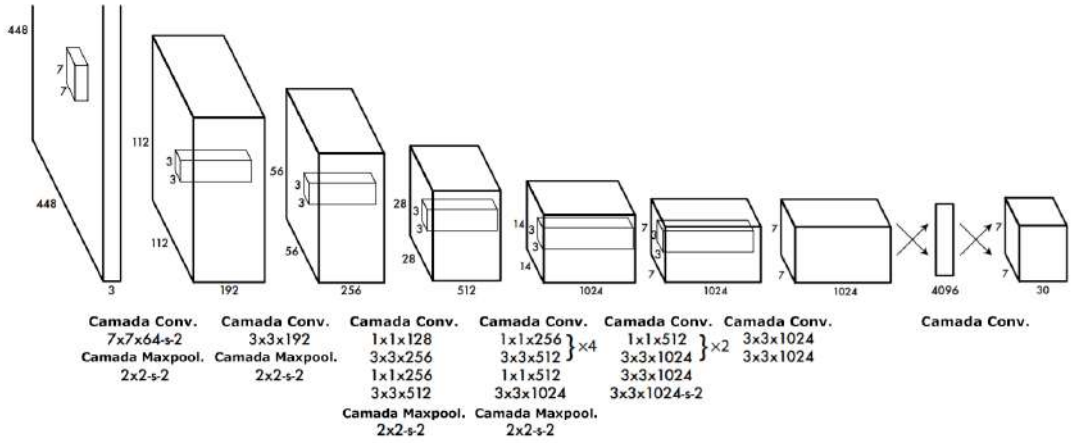
4.3.2 *Semantic SLAM*

O segundo método é chamado de *Semantic SLAM*. Com esse nome, foram encontrados dois artigos diferentes que, apesar de apresentarem estratégias diferentes para aplicação de SLAM, utilizam técnicas parecidas para realização da detecção de objetos. Assim, Zhang et al. (2018) utiliza a rede neural YOLO, enquanto que Han e Xi (2020a) utiliza uma evolução da mesma, YOLOv3.

YOLO, descrita por Redmon et al. (2016), é uma sigla para *You Only Look Once*, "Você olha apenas uma vez", em tradução livre. Nessa rede, a imagem é dividida (de forma convolucional) e, a partir da definição de caixas padrão, os objetos são identificados com apenas uma passada da imagem pela rede. Seu modelo está representado na Figura 25.

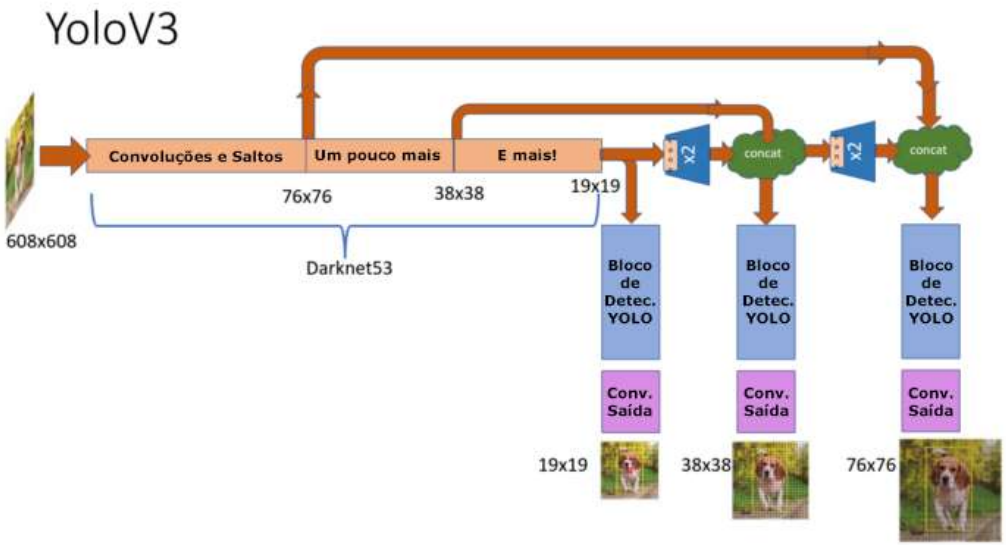
Sua evolução, YOLOv3 (REDMON; FARHADI, 2018), trabalha com detecção de objetos em múltiplas escalas (assim como a SSD), conectando algumas camadas anteriores da rede a sua saída. Como mostrado na Figura 26, ela é baseada na rede Darknet53, cuja principal função é de realizar convoluções e adiar conexões para extrair características da rede. Após a concatenação de camadas de diferentes tamanhos (19x19, 38x38 e 76x76), as saídas da rede consistem em

Figura 25 – Modelo da rede YOLO



Fonte: *You Only Look Once: Unified, Real-Time Object Detection* (Redmon et al., 2016) - tradução livre

Figura 26 – Modelo da rede YOLOv3



Fonte: *YOLOv3: An Incremental Improvement* (REDMON; FARHADI, 2018) - tradução livre

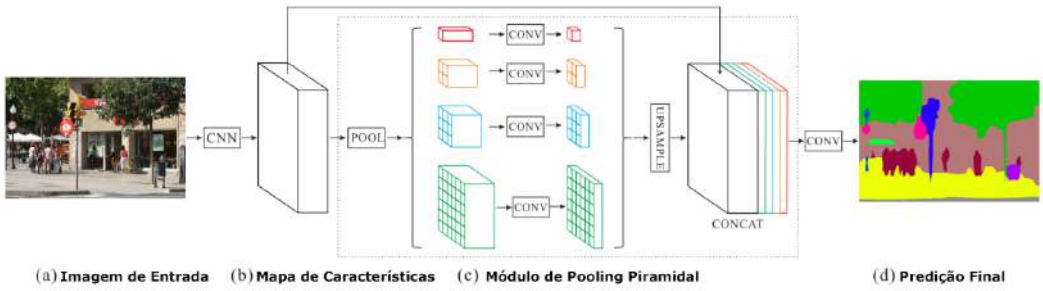
resultados de *offset* das caixas, tamanho das caixas, pontuação para se existe um objeto para aquela caixa e pontuação para a classificação do objeto em questão.

4.3.3 *PSPNet-SLAM*

O terceiro método é chamado de *PSPNet-SLAM* (Han; Xi, 2020b). Ele usa a rede *PSPNet* (ZHAO et al., 2017) para realizar uma tarefa um pouco diferente da detecção de objetos. A *PSPNet* se propõe a fazer a segmentação semântica das imagens, classificando cada um de seus *pixels* em diferentes classes - o modelo está representado na Figura 27. Segundo os autores da rede, esse método é mais eficiente do que a utilização de uma rede completamente conectada (FCN, *Fully Connected Network*).

Ele é baseado em um conjunto de camadas chamado de *Pyramid Pooling Module* (Módulo de *Pooling* Piramidal, em tradução livre). A imagem de entrada é, inicialmente, passada por

Figura 27 – Modelo da rede PSPNet



Fonte: *Pyramid Scene Parsing Network* (ZHAO et al., 2017) - tradução livre

uma rede neural convolucional para a extração das caraterísticas mais úteis para classificação (os autores usam a rede *ResNet* descrita por He et al. (2016)). Depois, a operação de *pooling* é realizada em diferentes padrões, gerando representações de diferentes porções da imagem, que também passam por operações de convolução para seleção de características. Esse formato permite a interpretação da imagem em contextos distintos em relação ao seu entorno de forma a obter melhores resultados.

Em seguida, cada uma das camadas passa por um processo de *upsampling* utilizando interpolação bilinear, de forma que a saída da rede tenha as mesmas dimensões que sua entrada. Os autores ressaltam que o módulo piramidal pode ser alterado (em formato e tamanho) para obtenção de resultados diferentes. No caso da implementação original, foram utilizadas camadas de 1x1, 2x2, 3x3 e 6x6.

4.4 Resultados da rede YOLOv3

A partir do trabalho de pesquisa, foi possível iniciar o treinamento de um dos modelos apresentados na Revisão Bibliográfica. Escolheu-se a arquitetura YOLOv3 por seus resultados satisfatórios em outros casos de uso e pela facilidade de encontrar implementações prontas.

As Figuras 28 e 29 descrevem o treinamento para o modelo YOLOv3 (REDMON; FARHADI, 2018). O gráfico da Figura 28 mostra o erro de treinamento, enquanto que o gráfico da Figura 29 mostra o erro de validação de treinamento. Como esperado, o erro da rede diminuiu ao longo do processo, convergindo depois de um total de 87 gerações levando à interrupção do treinamento (*"early stopping"*).

Passando as imagens do *dataset* de validação pelo modelo treinado da rede YOLOv3, obteve-se algumas amostras, compiladas na Figura 30. As caixas rosas representam os rótulos previamente feitos no *dataset* e as caixas amarelas representam predições da rede. Estas últimas apresentam também o resultado de identificação de objetos (pela palavra *"Plant"*) e um valor, que representa a confiança em cada predição.

Analisando a Figura 30, duas observações podem ser realizadas. Nos casos *(a)* e *(b)*, a rede foi capaz de encontrar bons resultados que não foram manualmente marcados nas caixas padrões do *dataset*. Por outro lado, nos casos *(c)* e *(d)*, a rede não conseguiu encontrar algumas

Figura 28 – Erro de treinamento para a rede YOLOv3.

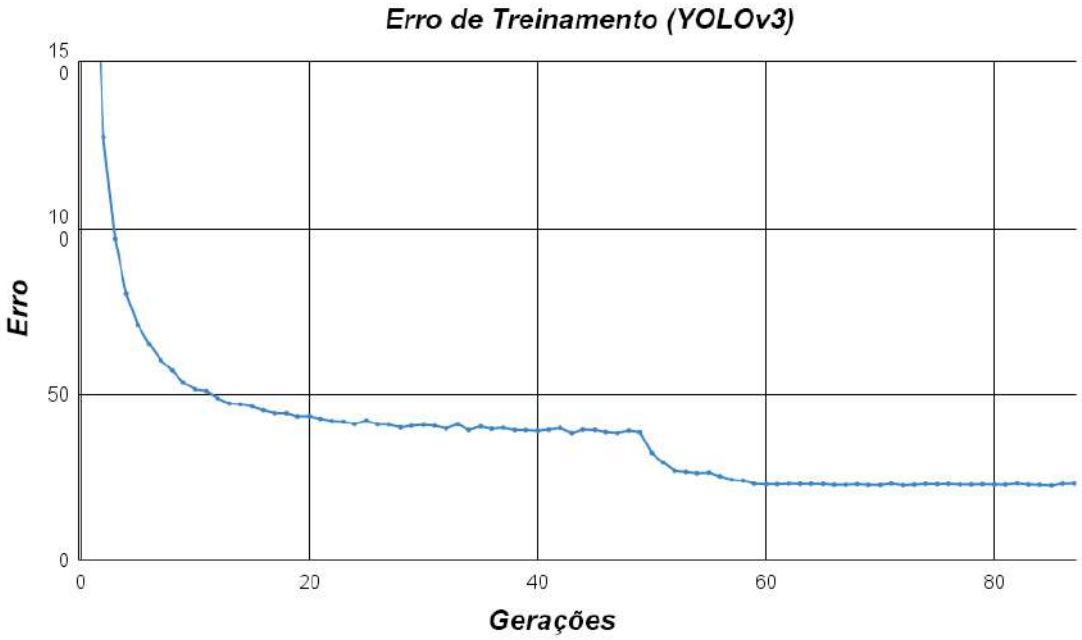


Figura 29 – Erro de validação durante o treinamento da rede YOLOv3.

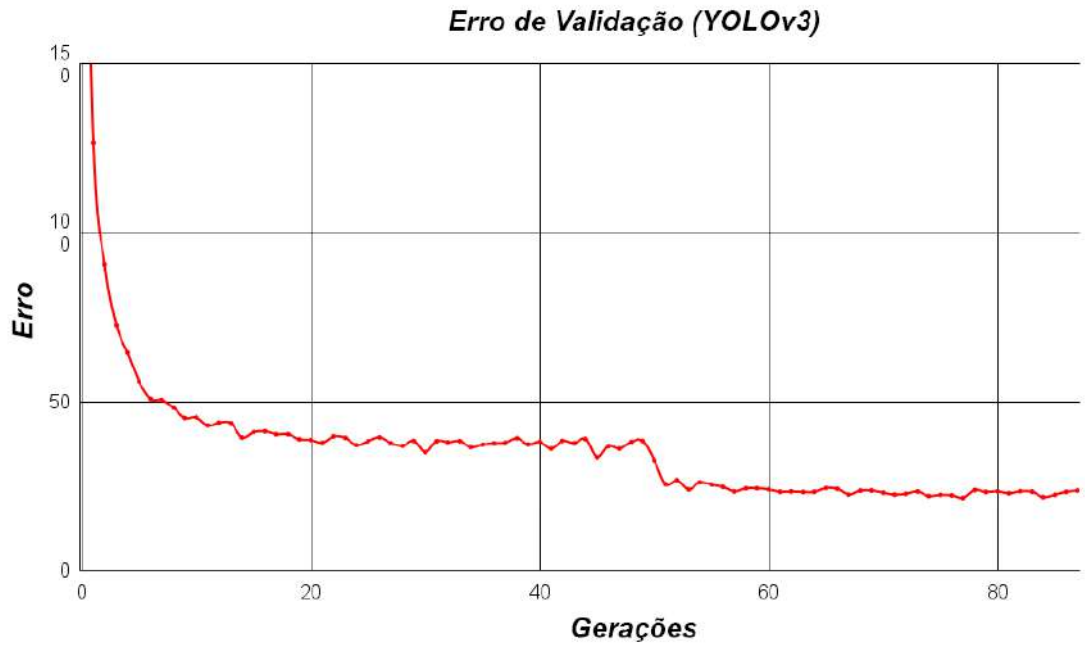
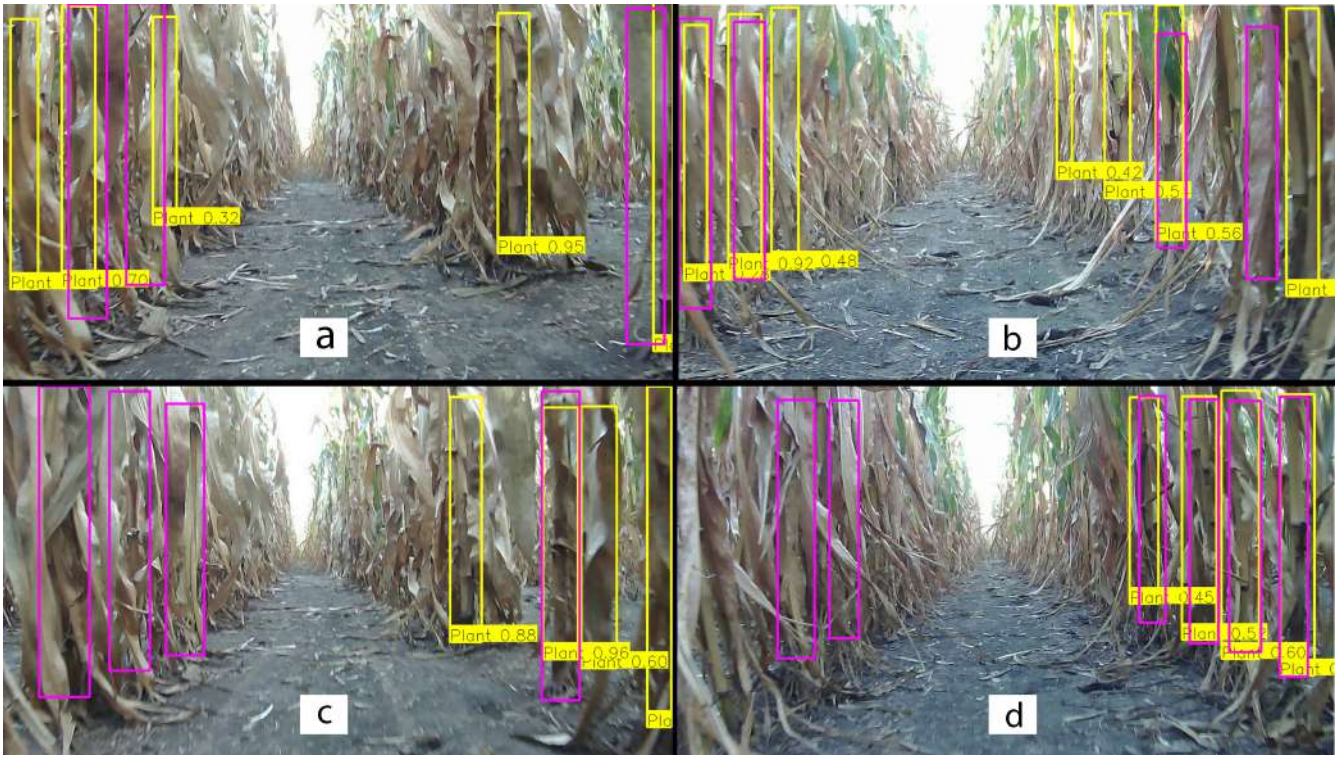


Figura 30 – Amostras de saída do modelo treinado



Legenda: *(a) e (b)* representam boas predições que não marcadas no *dataset*; *(c) e (d)* representam falhas da rede em identificar predições do *dataset*.

das plantas que foram manualmente marcadas no *dataset*.

A primeira observação pode ser explicada pelo método de construção do *dataset*: as imagens foram manualmente rotuladas por um humano. Ou seja, elas foram subjetivamente escolhidas - algumas vezes identificando aquelas que estavam mais explícitas e por outras marcando aquelas que estavam menos explícitas. Portanto, as omissões do *dataset* são justificadas pelo próprio método de rotulamento. Surpreendentemente, isso não impediu a rede de generalizar a informação dos *pixels* das imagens, ainda gerando bons resultados. Dessa forma, uma possível melhoria deste projeto é incorporar as predições da rede como rótulos do *dataset* e retreinar o modelo, buscando uma melhor otimização.

A segunda observação pode ser explicada pela própria natureza do funcionamento de uma rede neural. Apesar do treinamento ser capaz de realizar um processo de generalização dos pesos da rede, não é possível garantir perfeita precisão em imagens que foram apresentadas ao algoritmo posteriormente. Diferentemente da primeira observação, esse era um resultado esperado pelo autor e será discutido quantitativamente a seguir.

4.5 Métricas e resultados quantitativos

Usualmente, em uma tarefa de detecção de objetos, a métrica **mAP** (*Mean Average Precision*) é utilizada (YOHANANDAN, 2021). Ela mede o quanto o modelo treinado foi capaz de ajustar suas predições aos rótulos presentes no *dataset*. O valor calculado para essa métrica foi de $65,70\%$, um resultado que não descreve o desempenho real da rede, já que nem todas as

plantas foram explicitadas na rotulagem do *dataset*.

4.6 Parâmetros obtidos

Assim, para medir a precisão da rede, novas métricas foram desenvolvidas. Em cada uma das 250 imagens da porção de validação do *dataset*, o número de caixas padrão (n_{gt}) e o número de predições da rede (n_{pred}) foram medidas manualmente. Além disso, contou-se o número de "boas predições" (n_{gpred}) e a quantidade destas que coincidiram com as caixas padrões (n_{gtpred}). O termo "boas predições" tem como fundamento a questão: *"essa predição delimita a maioria dos pixels de uma planta?"*.

Com os valores descritos, também é possível definir o *"total virtual"* de plantas que são identificáveis em cada *frame* (n_{vtotal}), considerando o critério humano (rótulos do *dataset*) e a generalização alcançada pela rede. Este valor é dado pela Equação 4.13.

$$n_{vtotal} = n_{gt} + (n_{gpred} - n_{gtpred}) \tag{4.13}$$

4.7 Definição de métricas

A primeira métrica a ser calculada é a precisão da rede (P), definida como na Equação 4.14. A expressão mede a razão entre o número de boas predições da rede em relação ao total de predições.

$$P = \frac{n_{gpred}}{n_{pred}} \tag{4.14}$$

A segunda métrica é a coincidência da rede (C), definida como na Equação 4.15. A expressão mede a razão entre o número de predições que coincidem com as caixas padrão com o total de predições.

$$C = \frac{n_{gtpred}}{n_{pred}} \tag{4.15}$$

A terceira métrica é a incoincidência da rede (NC), definida como na Equação 4.16. A expressão mede a razão entre o número de predições da rede que não coincidem com as caixas padrão com o total de caixas padrão.

$$NC = \frac{n_{gt} - n_{gtpred}}{n_{gt}} \tag{4.16}$$

A quarta métrica mede a quantidade de predições extras da rede (EB), definida como na Equação 4.17. A expressão mede a razão entre o número de predições "extras" da rede (ou seja, que não tinham sido rotuladas no *dataset*) em relação ao total de predições.

$$EB = \frac{n_{gpred} - n_{gtpred}}{n_{pred}} \tag{4.17}$$

A quinta métrica mede a quantidade de predições extras da rede em relação ao *"total virtual"* de plantas (EBOT), definida como na Equação 4.18. A expressão mede a razão entre o número de predições "extras" da rede em relação ao *"total virtual"* mencionado.

$$EBOT = \frac{n_{gpred} - n_{gtpred}}{n_{vtotal}} \tag{4.18}$$

4.8 Cálculo das métricas

Calculando a média dos cinco parâmetros para cada imagem do dataset de validação, obteve-se o resultado das métricas como mostrado na Tabela 2. O prefixo *"m"* indica a média dos valores ao longo de todo o *dataset* de validação.

mAP	**mP**	**mC**	**mNC**	**mEB**	**mEBOT**
65,70%	93,88%	62,61%	25,71%	31,48%	26,18%

Tabela 2 – Valores médios das métricas da rede YOLOv3.

Observando a Tabela 2, é possível observar que a média da precisão do modelo (mP) é relativamente boa ($93,88\%$). Mesmo que $62,61\%$ (mC) das predições coincidam com as caixas padrão, $31,48\%$ (mEB) das predições foram boas e não coincidiram com os rótulos do *dataset*. Em relação ao *"total virtual"* de plantas em cada imagem, $26,18\%$ (mEBOT) das predições não possuem alguma referência no *dataset*. No entanto, $25,71\%$ (mNC) das caixas padrão não foram identificadas pelo modelo.

5 ABORDAGEM FINAL

Como explicado na Seção 4, os sistemas dinâmicos violam o princípio de ambientes estáticos. Ou seja, durante a extração de características, pontos que deveriam pertencer ao cenário estático se confundem com o movimento de objetos da cena. Após o desenvolvimento do detector de objetos, outras ideias foram levadas em consideração.

A mais promissora entre elas foi a de melhorar a estratégia de extração de pontos do ambiente. Ao invés de tentar filtrar a extração de pontos descritores, percebeu-se que poderia-se extrair a característica mais estável no cenário em questão: os pontos de emergência das plantas. O ponto de emergência é definido neste projeto como o ponto onde a planta se estende verticalmente a partir do solo. Na prática, o contato da planta com o solo se dá por uma superfície; o ponto aqui descrito simplifica essa superfície como um único ponto, por questões práticas.

Mesmo com o movimento das plantas pela ação do vento ou o crescimento ao longo da safra, o ponto de emergência de cada planta não deve se alterar ao longo do tempo. Vale ressaltar, porém, que oclusões da câmera devido às folhas (como observado na Seção 3.5) ainda são um problema persistente, mesmo com essa nova abordagem.

O sistema descrito neste trabalho é parecido àquele desenvolvido para o algoritmo ROW SLAM (YUAN et al., 2021) e foi inspirado por este. Um diagrama contendo uma visão geral do detector de pontos de emergência está mostrado na Figura 31. Ele está disponível como um pacote do framework Robot Operating System (ROS) em um repositório do Github (TOSCHI, 2023).

5.1 Modelo de segmentação

A estratégia básica deste sistema é detectar o solo e os caules das plantas, encontrar suas representações geométricas e o ponto onde essas representações se intersectam. Assim, a orientação dos caules é de extrema importância.

Nesse contexto, o detector de objetos apresentado na Seção 4 pode ser melhorado para uma tarefa diferente: a segmentação semântica (como no modelo PSPNet, apresentado apenas de forma representativa na Seção 4.3.3). Dessa forma, para encontrar o ponto de emergência, buscou-se treinar um modelo que classificasse cada um dos pixels da imagem.

5.1.1 Dataset do modelo de segmentação

Algumas sequências de imagens foram divididas em lotes para facilitar a rotulagem de forma modular. Cada conjunto contém duzentas imagens extraídas de pontos aleatórios e apresenta características distintas, como estágio de crescimento, situação (entrada, durante ou saída de uma fileira de cultivo), presença de plantas daninhas e condições de iluminação.

Para rotular os dados, foi utilizada a ferramenta CVAT (Computer Vision Annotation

Figura 31 – Resumo do sistema

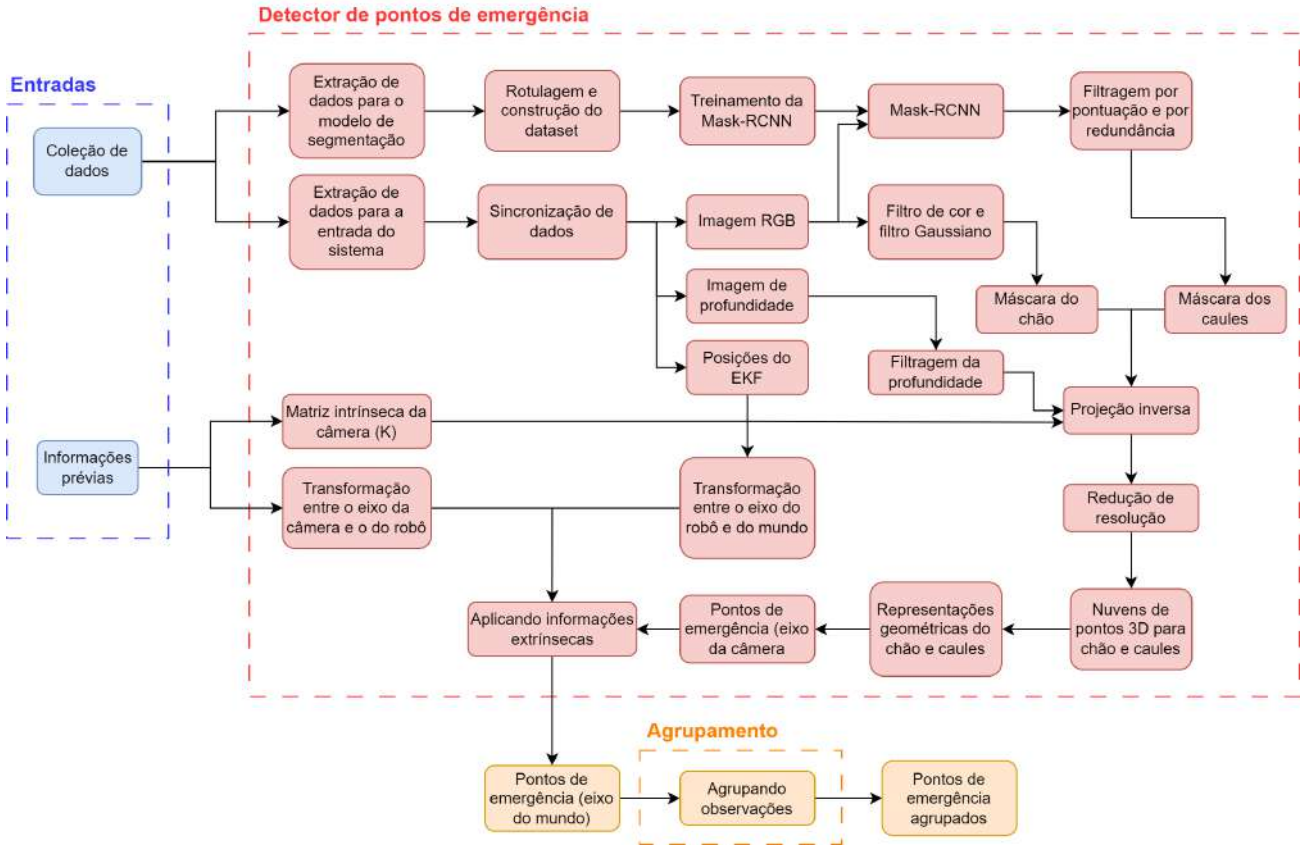
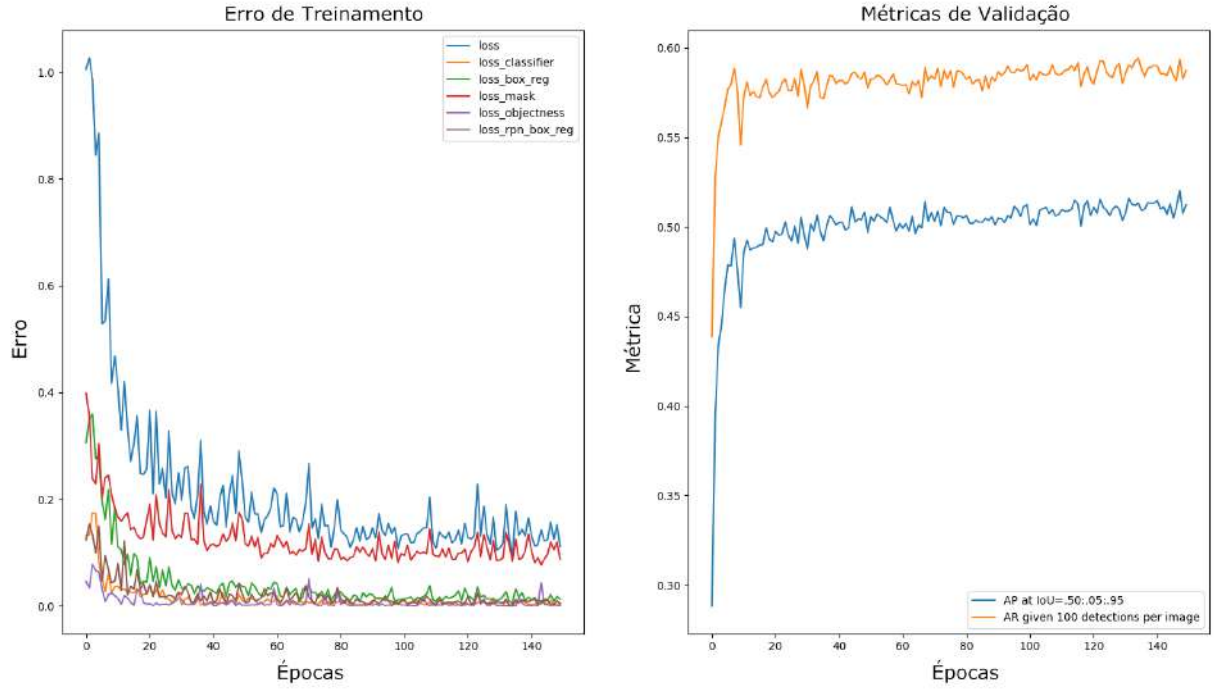


Figura 32 – Uma imagem RGB e sua respectiva máscara de caules. Imagem RGB (esquerda), máscara de caules (direita)

Tool) (SEKACHEV et al., 2020) também utilizada na Seção 4. O rotulamento foi feito por três pessoas diferentes: o autor deste trabalho e mais dois membros do DASLab. Ao final do processo, os rótulos foram revisados para garantir consistência.

Em cada imagem, as instâncias das plantas cultivadas foram anotadas usando polígonos. A Figura 32 mostra um exemplo de uma imagem RGB anotada e sua máscara correspondente, onde cada instância de planta cultivada possui uma cor diferente.

Figura 33 – Treinamento do modelo de segmentação e métricas de validação



5.1.2 Treinamento do modelo de segmentação

O modelo de detecção utiliza a Mask-RCNN (HE et al., 2018), uma arquitetura de segmentação de máscara de ponta. Ele foi escolhido devido à sua boa precisão e disponibilidade de implementação usando a biblioteca *torchvision* (PYTORCH, 2023). O modelo foi treinado usando pesos de backbone pré-treinados com o conjunto de dados COCO (LIN et al., 2015) e alcançou 46,7% no conjunto de dados de validação usando a biblioteca de avaliação COCO em Python (COCODATASET, 2023).

Devido ao processo de rotulagem ser muito demorado, o modelo de segmentação foi treinado iterativamente com diferentes quantidades de dados até que alcançasse resultados satisfatórios. O modelo final utilizou 533 imagens, divididas em conjunto de dados de treinamento (85%) e conjunto de dados de validação (15%).

A Figura 33 mostra a evolução das métricas de treinamento e validação ao longo das épocas. A biblioteca de avaliação COCO em Python utiliza principalmente duas métricas para avaliar o detector: a precisão média (AP) e o recall médio (AR). A primeira mede a precisão das previsões, enquanto a segunda mede o quanto o modelo consegue encontrar os alvos (SHAH, 2023).

5.2 Carregando dados para a pipeline

Para carregar os dados na pipeline, as poses do robô, as imagens RGB e as imagens de profundidade devem ser sincronizadas. A sincronização é feita iterando sobre os dados menos frequentes e obtendo os outros dois componentes que têm o carimbo de tempo mais próximo do primeiro. Os dados são carregados uma vez a cada ciclo da pipeline usando uma estrutura de

gerador em Python (STRATIS, 2023), evitando problemas de memória.

5.3 Calculando os pontos 3D no eixo de coordenadas da câmera

Existem algumas etapas para realizar a projeção inversa dos caules no espaço tridimensional do frame global. A primeira etapa é encontrar onde eles estão em cada imagem individual. Se houver caules na imagem (sem oclusão total), os parâmetros intrínsecos da câmera (fornecidos pela equipe DASLab) e a imagem de profundidade são usados em conjunto para encontrar os respectivos pontos 3D no eixo da câmera.

O modelo de segmentação recebe a imagem RGB devidamente carregada e produz máscaras para cada caule detectado. O modelo também fornece pontuações que representam a confiança em cada previsão. Depois disso, as máscaras são filtradas pelas suas pontuações, suprimindo as detecções com pontuações baixas.

Ainda há um problema: algumas detecções representam a mesma instância de caule, gerando redundância indesejada. Para resolver isso, uma curva média é obtida pela média das coordenadas X da máscara para cada coordenada Y. Uma linha é ajustada usando os dados da curva média, aplicando o algoritmo Random Sample Consensus (RANSAC) (INTERNATIONAL, 1981) implementado na biblioteca Sklearn (LEARN, 2023c). A linha é então extrapolada até a parte inferior da imagem, resultando em um ponto. A coordenada X desse ponto é comparada com a mesma coordenada das outras máscaras. Se elas estiverem suficientemente próximas, as detecções são mescladas.

O próximo passo é encontrar o chão na imagem usando limiarização de cores com a biblioteca OpenCV (OPENCV, 2023a). Primeiro, a imagem RGB é convertida para o espaço de cores HSV. Depois disso, limites inferiores e superiores são definidos manualmente para cada canal HSV para corresponder à cor do chão. A imagem é então binarizada, resultando em uma máscara. Um filtro gaussiano (OPENCV, 2023b) também é aplicado à máscara, removendo pontos desconectados indesejados.

Com as máscaras para cada caule e o chão, é possível projetar os pontos de volta para o espaço tridimensional usando informações de profundidade coletadas da câmera estéreo. Infelizmente, os dados de profundidade possuem ruído que afeta a estimativa do caule. Para resolver esse problema, é implementado um filtro de profundidade. Para cada máscara de caule, os dados de profundidade correspondentes são usados para avaliar um histograma com 500 seções. A distância mais frequente e um número determinado de vizinhos são usados como limites inferior e superior para recortar as informações de profundidade para essa máscara.

Finalmente, a Equação 5.1 foi usada para realizar a projeção inversa dos pontos, onde z é a informação de profundidade para um único ponto (em metros) e K é a matriz de câmera intrínseca. O vetor 2D é escrito em coordenadas homogêneas.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{3D} = z \cdot K^{-1} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}_{2D} \tag{5.1}$$

Aplicando a Equação 5.1 a cada ponto da máscara 2D, obtemos uma nuvem de pontos 3D para cada caule e para o chão. O próximo passo é construir representações geométricas para ambos. Para as plantas cultivadas, é necessário um ponto e um vetor para representar a posição e a orientação do caule, respectivamente. Para o chão, é necessário um ponto e dois vetores para descrever um plano.

O ponto escolhido para as plantas cultivadas e para o chão é o ponto médio, calculado pela média de todos os pontos das respectivas nuvens de pontos. Os vetores são obtidos usando a Análise de Componentes Principais (PCA) (JOLLIFFE; CADIMA, 2016) implementada na biblioteca Sklearn (LEARN, 2023b). Para as plantas cultivadas, o primeiro componente principal é o vetor de orientação. Para o plano do chão, os dois primeiros componentes principais são usados para calcular o vetor normal do plano por meio do produto cruzado entre eles.

Por fim, os pontos emergentes são calculados encontrando a interseção entre as linhas das plantas cultivadas e o plano do chão usando as Equações 5.2 e 5.3.

$$d = \frac{(\mathbf{p}_0 - \mathbf{l}_0) \cdot \mathbf{n}}{\mathbf{l} \cdot \mathbf{n}} \tag{5.2}$$

$$\mathbf{p}_{em} = \mathbf{p}_0 + d \cdot \mathbf{l} \tag{5.3}$$

Na Equação 5.2, d é o escalar de linha que representa o ponto de interseção, $\mathbf{p}_0$ é um ponto do plano, $\mathbf{l}_0$ é um ponto da linha, $\mathbf{n}$ é o vetor normal do plano e $\mathbf{l}$ é o vetor da linha. Na Equação 5.3, $\mathbf{p}_{em}$ é o vetor do ponto emergente desejado.

Para melhorar o desempenho, a biblioteca Open3D (OPEN3D, 2019) foi utilizada para realizar o downsampling com voxels. Isso funciona agrupando pontos próximos em voxels e calculando a média dos pontos dentro de cada voxel ocupado. Para as nuvens de pontos dos caules e do chão, usou-se uma grade de voxel com tamanhos 5 cm e 1 cm, respectivamente.

5.4 Adicionando informação extrínseca

Uma vez que os pontos emergentes são calculados no sistema de coordenadas da câmera, as representações geométricas são transformadas para o sistema de coordenadas global adicionando duas transformações. A primeira é a transformação entre o sistema de coordenadas da câmera e o sistema de coordenadas do robô - ela é sempre a mesma, dada pela tradução e rotação entre a câmera e o centro do robô. A segunda é a transformação entre o sistema de coordenadas global e o sistema de coordenadas do robô, dada pelos dados de pose do EKF.

A Equação 5.4 mostra a expressão usada para transformar um ponto 3D (coordenadas homogêneas) do sistema de coordenadas da câmera para o sistema de coordenadas global. O t_{3x1}

é a matriz de translação e R_{3x3} é a matriz de rotação (calculada com os valores de yaw (ψ), pitch (θ) e roll (φ) conforme mostrado na Equação 5.5).

$$w \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix}_W = \begin{bmatrix} R_{3x3} & t_{3x1} \\ 0_{1x3} & 1 \end{bmatrix}_{WB} \begin{bmatrix} R_{3x3} & t_{3x1} \\ 0_{1x3} & 1 \end{bmatrix}_{BC} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_C \tag{5.4}$$

$$R_{3x3} = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\varphi) & -sin(\varphi) \\ 0 & sin(\varphi) & cos(\varphi) \end{bmatrix} \tag{5.5}$$

5.5 Agrupando observações

O processo descrito produz observações 3D dos pontos emergentes, mas não a correlação entre eles. Esta seção descreve duas abordagens para agrupar as detecções em diferentes instâncias de caules. A primeira utiliza o algoritmo Simple Online and Realtime Tracking (SORT) (BEWLEY et al., 2016) e a segunda utiliza um método de agrupamento não supervisionado chamado Density-based Clustering Algorithm (DBSCAN) (DENG, 2020).

O primeiro método foi testado usando uma implementação em Python desenvolvido por Bewley (2022). O algoritmo SORT consiste no rastreamento de caixas delimitadoras em diferentes sistemas de coordenada. Ele utiliza um Filtro de Kalman para cada rastreador, assumindo um modelo de velocidade constante. No entanto, devido a oclusões, à baixa taxa de quadros da câmera e à violação do modelo de velocidade constante, este método não obteve bom desempenho e não foi utilizado na versão final.

O segundo método foi testado utilizando a implementação da biblioteca Sklearn (LEARN, 2023a). Este método agrupa as coordenadas dos pontos emergentes com base em sua proximidade espacial e requer dois parâmetros para funcionar de forma eficaz. O primeiro parâmetro determina o número mínimo de pontos necessários para que um grupo seja considerado um cluster válido, enquanto o segundo parâmetro, denotado por ϵ, especifica a distância máxima que um ponto pode estar de outros e ainda ser considerado parte do mesmo cluster. Após experimentação, verificou-se que o uso de um mínimo de 3 pontos e um valor de ϵ de 5 cm produziu melhores resultados. Ao contrário do primeiro método, a abordagem DBSCAN também é capaz de identificar e remover outliers.

5.6 Resultados

Em uma pequena sequência, os resultados do método descrito podem ser observados nas Figuras 34 e 35.

Figura 34 – Visualização 3D de uma sequência pequena

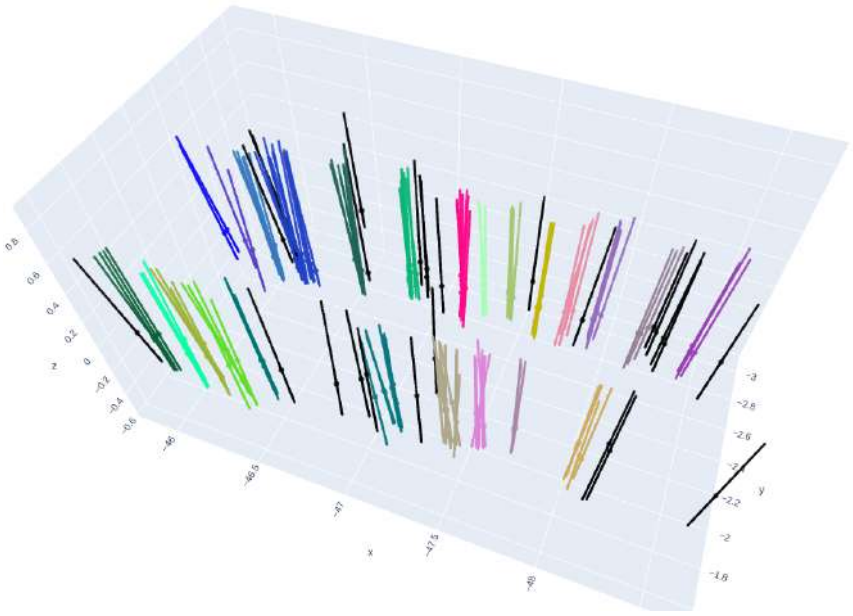
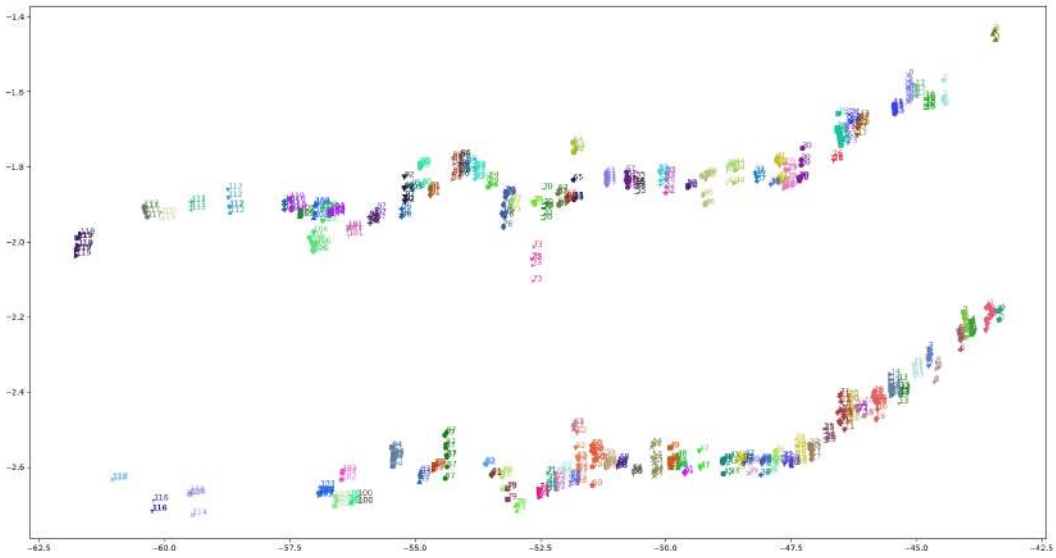


Figura 35 – Visualização 2D de uma sequência pequena

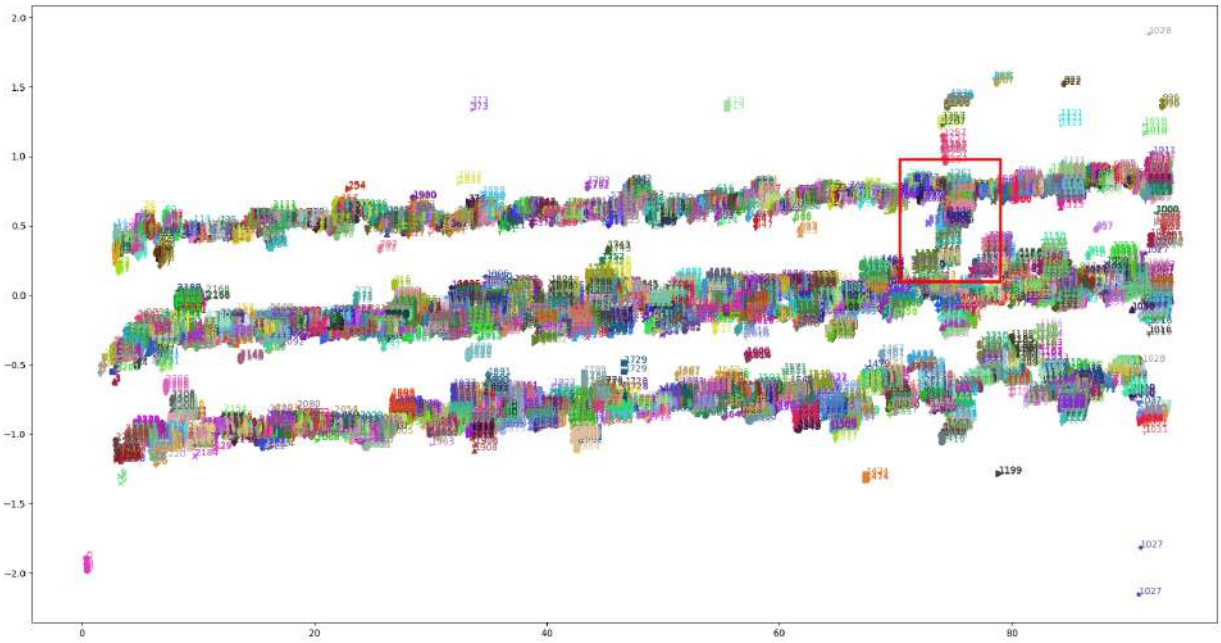


A Figura 34 mostra a visualização em 3D de uma seção das plantações, onde a cor preta simboliza outliers, e a Figura 35 mostra uma vista superior de toda a sequência. Em ambas as figuras, cores não pretas representam instâncias de caules individuais.

Para uma sequência grande, a Figura 36 mostra a vista superior. Nessa figura, é possível observar que algumas instâncias de caules aparecem dentro da linha de cultivo (retângulo vermelho), o que não faz sentido na estrutura da cultura. Ao analisar os dados, foi descoberto que o robô ficou preso por vários segundos na linha de baixo olhando na direção da primeira, tornando os dados visuais pouco confiáveis.

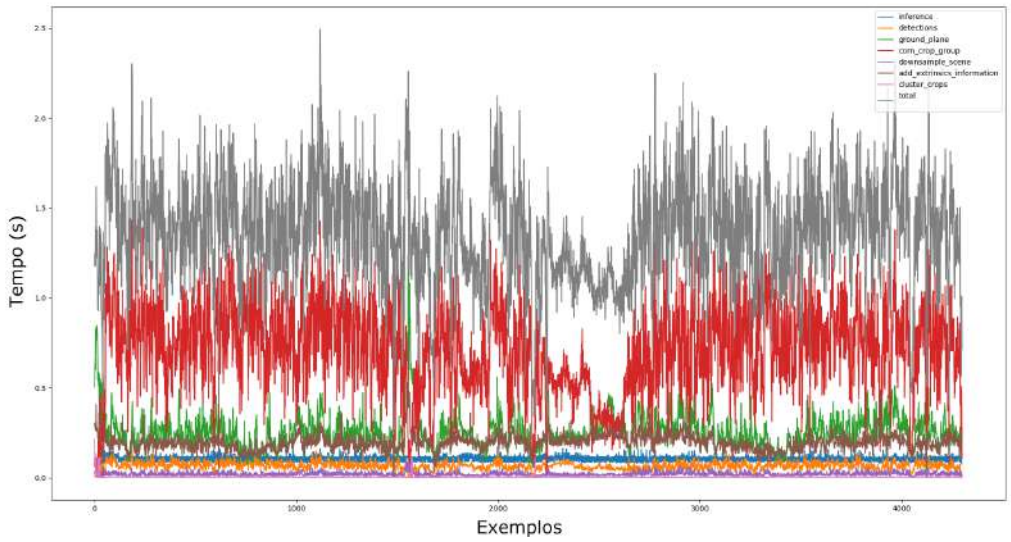
A Figura 37 mostra os tempos de execução para as partes mais custosas do pipeline ao obter os resultados mostrados na Figura 36. Os testes foram realizados em uma máquina equipada com um processador Intel Core i7 e uma GPU GeForce RTX 2070 Mobile.

Figura 36 – Visualização 2D de uma sequência grande



Legenda: Retângulo vermelho mostra resultados incompatíveis.

Figura 37 – Medidas de tempo para uma sequência grande.



Atualmente, a tarefa mais custosa é a retroprojeção dos caules de milho (rótulo *corn_crop_group*), seguida da retroprojeção do plano do solo (rótulo *ground_plane*). Em média, a abordagem leva 1,5 segundos para executar uma única iteração de todo o pipeline.

5.7 Possíveis melhorias

Essa abordagem possui algumas dependências críticas nos dados de entrada. Para que funcione adequadamente, a coleta de dados precisa seguir alguns requisitos. O primeiro é ter uma estimativa confiável de pose pelo EKF. Algumas sequências de dados fornecidas tiveram problemas com a convergência do EKF, reduzindo os dados disponíveis para trabalhar. Além

disso, o problema descrito na Figura 36 pode ser resolvido com um melhor controle do robô durante a extração de dados. A equipe DASLab planeja obter mais dados levando em consideração esses detalhes em uma nova oportunidade.

A estimativa dos pontos emergentes pode ser aprimorada melhorando a estimativa do plano do solo. A segmentação baseada em limiar de cor depende muito de valores arbitrários e não é geral o suficiente para diferentes casos. Rotulagem de mais imagens pode ser feita para melhorar o conjunto de dados do modelo de segmentação e encontrar máscaras para o solo. Isso não prejudicaria o desempenho, pois o modelo já precisa ser executado para a segmentação dos caules.

O desempenho de tempo também pode ser melhorado abordando a seção de retroprojeção de maneira diferente. Como descrito aqui, todos os pontos nas máscaras detectadas são calculados no espaço 3D para gerar uma nuvem de pontos. Utilizar um método de amostragem para selecionar pontos para a projeção pode ser mais eficaz e requer mais testes. Além disso, algumas partes do código podem ser refatoradas para usar abordagens vetorizadas, possivelmente alcançando um tempo de execução mais rápido.

6 CONCLUSÃO

Neste trabalho, foi proposto um estudo da aplicação de sistemas SLAM para o meio agrícola utilizando o robô TerraSentia. A aplicação dessa tecnologia em ambientes dinâmicos (como o agrícola) ainda é um problema aberto na literatura.

Em um primeiro momento, buscou-se realizar a aplicação direta de um algoritmo SLAM chamado ORB-SLAM2 em sequências de dados capturadas pelo TerraSentia. Esperava-se que, em estágios mais avançados de desenvolvimento das plantas, o volume de objetos móveis atrapalhasse o desempenho do sistema. Essa hipótese não pôde ser confirmada completamente, já que a oclusão da câmera por esses mesmos objetos (principalmente folhas) forçava a quebra do sistema SLAM. De toda forma, a autonomia do algoritmo para câmeras monoculares e estéreo foi apenas de alguns metros para as duas métricas utilizadas, indicando que a aplicação de algoritmos SLAM de forma direta não parece viável.

Dessa forma, o objetivo trabalho foi transicionado para o desenvolvimento de módulos auxiliares para SLAM. Partindo do princípio que algoritmos SLAM consideram que o ambiente está estático durante sua operação, a primeira ideia foi encontrar as regiões da imagem que provavelmente conteriam os pontos mais dinâmicos: a parte superior das plantas; posteriormente, esses pontos dinâmicos podem ser removidos para melhorar a consistência do sistema SLAM. Assim, foi desenvolvido um detector de objetos utilizando técnicas de Deep Learning. O detector obteve bons resultados nas métricas apresentadas, se tornando satisfatório para uma futura aplicação juntamente com um algoritmo SLAM.

Por fim, um novo sistema foi desenvolvido com o objetivo de testar mais uma abordagem. Ao invés de filtrar os pontos detectados pelo algoritmo SLAM, o sistema de percepção poderia identificar diretamente os pontos mais estáticos da cena: os pontos de emergência das plantas. Eles são invariáveis com o tempo, já que não mudam com o crescimento das plantas e nem devem ser suscetíveis à ação do vento. Esse novo sistema contém uma pipeline mais complexa, envolvendo a projeção de pontos 3D e um modelo de segmentação semântica para a identificação dos caules com mais precisão.

Entre as futuros trabalhos possíveis para complementar este trabalho, estão:

- Coleta de mais dados com o robô TerraSentia tomando mais cuidados em relação à trajetória
- Rotulagem do solo do ambiente no dataset do detector de pontos de emergência
- Otimização do detector de pontos de emergência para melhorar seu tempo de execução
- Aplicação dos métodos apresentados em conjunto com algoritmos de SLAM

Por fim, pode-se concluir que o objetivo primário deste trabalho foi alcançado através da finalização de cada um dos objetivos secundários descritos no início deste documento.

REFERÊNCIAS

BEWLEY, A. **SORT**. 2022. <https://github.com/abewley/sort>. Accessed on 16 April 2023.

BEWLEY, A. et al. Simple online and realtime tracking. In: **2016 IEEE International Conference on Image Processing (ICIP)**. [S.l.]: IEEE, 2016.

CHEN, Y.; CHEN, Y.; WANG, G. **Bundle Adjustment Revisited**. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1912.03858>.

COCODATASET. **COCO API**. 2023. <https://github.com/cocodataset/cocoapi>. Accessed on 17 April 2023.

COURSERA. **Seu percurso rumo ao sucesso**. 2021. Acessado em 8 mar. 2021. Disponível em: <https://www.coursera.org/>.

DEEPLEARNING.AI. **Build your AI career with DeepLearning.AI**. 2021. Acessado em 8 mar. 2021. Disponível em: <https://www.deeplearning.ai/>.

_____. **Certificado Profissional Desenvolvedor de DeepLearning.AI no TensorFlow**. 2021. Acessado em 1 mar. 2021. Disponível em: <https://www.coursera.org/professional-certificates/tensorflow-in-practice>.

_____. **Programa de cursos integrados, aprendizagem profunda**. 2021. Acessado em 15 fev. 2021. Disponível em: <https://pt.coursera.org/specializations/deep-learning>.

DENG, D. Dbscan clustering algorithm based on density. In: **2020 7th International Forum on Electrical Engineering and Automation (IFEEA)**. [S.l.: s.n.], 2020. p. 949–953.

DURáN, J. **Everything You Need to Know about Gradient Descent Applied to Neural Networks**. 2019. Acessado em 8 abr. 2021. Disponível em: <https://medium.com/yottabytes/everything-you-need-to-know-about-gradient-descent-applied-to-neural-networks-d70f85e0cc14>.

EARTHSENSE, I. **EarthSense Agricultural Intelligence**. 2021. Acessado em 13 jun. 2021. Disponível em: <https://www.earthsense.co/>.

GALVEZ-LóPEZ, D.; TARDOS, J. D. Bags of binary words for fast place recognition in image sequences. **IEEE Transactions on Robotics**, v. 28, n. 5, p. 1188–1197, 2012.

Gao, X. et al. Review of wheeled mobile robots' navigation problems and application prospects in agriculture. **IEEE Access**, v. 6, p. 49248–49268, 2018.

Han, S.; Xi, Z. Dynamic scene semantics slam based on semantic segmentation. **IEEE Access**, v. 8, p. 43563–43570, 2020.

_____. Dynamic scene semantics slam based on semantic segmentation. **IEEE Access**, v. 8, p. 43563–43570, 2020.

HE, K. et al. **Mask R-CNN**. 2018.

He, K. et al. Deep residual learning for image recognition. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2016. p. 770–778.

HIGUTI, V. et al. Under canopy light detection and ranging-based autonomous navigation. **Journal of Field Robotics**, v. 36, 12 2018.

INTERNATIONAL, S. **Random Sample Consensus (RANSAC) algorithm**. 1981. <https://www.sri.com/hoi/random-sample-consensus-ransac-algorithm/>. Accessed on 8 April 2023.

JOLLIFFE, I. T.; CADIMA, J. Principal component analysis: a review and recent developments. **Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences**, v. 374, n. 2065, p. 20150202, 2016.

KAYACAN, E.; ZHANG, Z.; CHOWDHARY, G. Embedded high precision control and corn stand counting algorithms for an ultra-compact 3d printed field robot. In: . [S.l.: s.n.], 2018.

LEARN, S. **DBSCAN**. 2023. <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html>. Accessed on 9 April 2023.

_____. **PCA**. 2023. <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>. Accessed on 8 April 2023.

_____. **RANSACRegressor**. 2023. <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RANSACRegressor.html>. Accessed on 8 April 2023.

LIN, T.-Y. et al. **Microsoft COCO: Common Objects in Context**. 2015.

LIU, W. et al. Ssd: Single shot multibox detector. **Lecture Notes in Computer Science**, Springer International Publishing, p. 21–37, 2016. ISSN 1611-3349. Disponível em: <http://dx.doi.org/10.1007/978-3-319-46448-0_2>.

MEEUSSEN, W. **robot_pose_ekf**. 2023. <http://wiki.ros.org/robot_pose_ekf>. Accessed on 13 April 2023.

MUR-ARTAL, R.; MONTIEL, J. M. M.; TARDÓS, J. D. ORB-SLAM: a versatile and accurate monocular SLAM system. **CoRR**, abs/1502.00956, 2015. Disponível em: <http://arxiv.org/abs/1502.00956>.

Mur-Artal, R.; Tardós, J. D. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. **IEEE Transactions on Robotics**, v. 33, n. 5, p. 1255–1262, 2017.

Mur-Artal, R. et al. **ORB-SLAM**. 2022. Acessado em 28 ago. 2022. Disponível em: <http://webdiis.unizar.es/~raulmur/orbslam/>.

_____. **orb_slam2_ros**. 2022. Acessado em 28 ago. 2022. Disponível em: <http://wiki.ros.org/orb_slam2_ros>.

NVIDIA. **CUDA Toolkit**. 2023. Acessado em 10 jul. 2023. Disponível em: <https://developer.nvidia.com/cuda-toolkit>.

_____. **Notebooks Gamer GEFORCE GTX 1650**. 2023. Acessado em 24 jul. 2023. Disponível em: <https://www.nvidia.com/pt-br/geforce/gaming-laptops/gtx-1650/>.

OPEN3D. **Point cloud**. 2019. <http://www.open3d.org/docs/0.8.0/tutorial/Basic/pointcloud.html>. Accessed on 8 April 2023.

OPENCV. **BRIEF (Binary Robust Independent Elementary Features)**. 2022. Acessado em 30 jul. 2022. Disponível em: <https://docs.opencv.org/3.4/dc/d7d/tutorial_py_brief.html>.

_____. **FAST Algorithm for Corner Detection**. 2022. Acessado em 30 jul. 2022. Disponível em: <https://docs.opencv.org/3.4/df/d0c/tutorial_py_fast.html>.

_____. **ORB (Oriented FAST and Rotated BRIEF)**. 2022. Acessado em 25 jul. 2022. Disponível em: <https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html>.

_____. **Understanding Features**. 2022. Acessado em 28 jul. 2022. Disponível em: <https://docs.opencv.org/3.4/df/d54/tutorial_py_features_meaning.html>.

_____. **Image Thresholding**. 2023. <https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html>. Accessed on 10 April 2023.

_____. **Smoothing Images**. 2023. <https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html>. Accessed on 10 April 2023.

OPENROBOTICS. **CameraInfo**. 2023. Acessado em 27 jun. 2023. Disponível em: <http://docs.ros.org/en/melodic/api/sensor_msgs/html/msg/CameraInfo.html>.

_____. **roslaunch**. 2023. Acessado em 23 jun. 2023. Disponível em: <http://wiki.ros.org/roslaunch>.

_____. **rviz**. 2023. Acessado em 5 jul. 2023. Disponível em: <http://wiki.ros.org/rviz>.

_____. **tf2**. 2023. Acessado em 19 mai. 2023. Disponível em: <http://wiki.ros.org/tf2>.

ORTIZ, L.; GONçALVES, L.; CABRERA, E. **A Generic Approach for Error Estimation of Depth Data from (Stereo and RGB-D) 3D Sensors**. 2017.

PYTORCH. **Mask R-CNN**. 2023. <https://pytorch.org/vision/main/models/mask_rcnn.html>. Accessed on 12 April 2023.

RAULMUR. **ORB-SLAM2**. 2023. Acessado em 7 mai. 2023. Disponível em: <https://github.com/raulmur/ORB_SLAM2>.

Redmon, J. et al. You only look once: Unified, real-time object detection. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2016. p. 779–788.

REDMON, J.; FARHADI, A. **YOLOv3: An Incremental Improvement**. 2018.

REVICH, J. et al. **Precision Farming - Cheating Malthus with Digital Agriculture**. [S.l.], 2016. 4–5/18 p.

ROBERT, J. **Hands-On Introduction to Robot Operating System (ROS)**. 2020. Acessado em 6 abr. 2021. Disponível em: <https://trojrobert.github.io/hands-on-introdution-to-robot-operating-system(ros)/>.

ROBOTICS, O. **ROS**. 2021. Acessado em 20 mar. 2021. Disponível em: <https://www.ros.org/>.

ROSEBROCK, A. **Intersection over Union (IoU) for object detection**. 2016. Acessado em 5 abr. 2021. Disponível em: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>.

SAHA, S. **A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way**. 2018. Acessado em 29 mar. 2021. Disponível em: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>.

SEKACHEV, B. et al. **opencv/cvat: v1.1.0**. Zenodo, 2020. Disponível em: <https://doi.org/10.5281/zenodo.4009388>.

SHAH, D. **Mean Average Precision (mAP) Explained: Everything You Need to Know**. 2023. <https://www.v7labs.com/blog/mean-average-precision>. Accessed on 16 April 2023.

SIMONYAN, K.; ZISSERMAN, A. **Very Deep Convolutional Networks for Large-Scale Image Recognition**. 2015.

SMITH, R. C.; CHEESEMAN, P. On the representation and estimation of spatial uncertainty. **The International Journal of Robotics Research**, v. 5, n. 4, p. 56–68, 1986. Disponível em: <https://doi.org/10.1177/027836498600500404>.

STEREOLABS. **Meet ZED 2**. 2023. <https://www.stereolabs.com/zed-2/>. Accessed on 13 April 2023.

_____. **StereoLabs**. 2023. Acessado em 24 jul. 2023. Disponível em: <https://www.stereolabs.com>.

_____. **Stereolabs ZED Camera - ROS Noetic Ninjemis Integration**. 2023. <https://github.com/stereolabs/zed-ros-wrapper>. Acessado em 13 abr. 2023.

_____. **Video Recording**. 2023. <https://www.stereolabs.com/docs/video/recording/>. Acessado em 13 abr. 2023.

STRATIS, K. **How to Use Generators and yield in Python**. 2023. <https://realpython.com/introduction-to-python-generators/>. Accessed on 5 April 2023.

TOSCHI, L. **Semantic feature detector for cornfields**. 2023. <https://github.com/toschilt/ts_semantic_feature_detector>. Accessed on 24 April 2023.

XIAO, L. et al. Dynamic-slam: Semantic monocular visual localization and mapping based on deep learning in dynamic environment. **Robotics and Autonomous Systems**, v. 117, p. 1 – 16, 2019. ISSN 0921-8890. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0921889018308029>.

YOHANANDAN, S. **mAP (mean Average Precision) might confuse you!** 2021. Acessado em 7 set. 2021. Disponível em: <https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2>.

YUAN, J. et al. **ROW-SLAM: Under-Canopy Cornfield Semantic SLAM**. [S.l.]: arXiv, 2021.

ZAFFAR, M. et al. Sensors, slam and long-term autonomy: A review. In: . [S.l.: s.n.], 2018. p. 285–290.

Zhang, L. et al. Semantic slam based on object detection and improved octomap. **IEEE Access**, v. 6, p. 75545–75559, 2018.

ZHAO, H. et al. **Pyramid Scene Parsing Network**. 2017.