



UNIVERSIDADE DE
COIMBRA

José Pedro Peixe Monteiro

**DETEÇÃO E CLASSIFICAÇÃO DE OBSTÁCULOS À
NAVEGAÇÃO DE VEÍCULOS AUTÓNOMOS EM
AMBIENTES FLORESTAIS**

Dissertação no âmbito do Mestrado em Engenharia Mecânica na Especialidade de Energia e Ambiente orientada pelo Professor Doutor Carlos Xavier Pais Viegas e pelo Professor Doutor Miguel Rosa Oliveira Panão e apresentada ao Departamento de Engenharia Mecânica da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

fevereiro de 2023

1 2



9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE
COIMBRA

Deteção e classificação de obstáculos à navegação de veículos autónomos em ambientes florestais

Dissertação apresentada para a obtenção do grau de Mestre em Engenharia Mecânica
na Especialidade de Energia e Ambiente

Detection and classification of obstacles to navigation of autonomous vehicles in forest environments

Autor

José Pedro Peixe Monteiro

Orientadores

Carlos Xavier Pais Viegas

Miguel Rosa Oliveira Panão

Júri

Presidente	Professor Doutor Ricardo Nuno Madeira Soares Branco Professor Auxiliar, DEM-UC
Orientador	Professor Doutor Carlos Xavier Pais Viegas Professor Auxiliar Convidado, DEM-UC Professor Doutor Pedro Mariano Simões Neto Professor Associado com Agregação, DEM-UC
Vogais	Professor Doutor Nuno Miguel Fonseca Ferreira Professor Coordenador com Agregação, ISEC

Colaboração Institucional



Bold Robotics

Coimbra, fevereiro, 2023

Agradecimentos

Começo por deixar o meu agradecimento ao Professor Doutor Carlos Viegas e ao Professor Doutor Miguel Panão pela orientação deste trabalho e todo o apoio concedido. A todos os professores de quem fui aluno ao longo do meu percurso académico, e que permitiram chegar a este ponto. Deixo também um agradecimento à equipa do Field Tech Laboratory, pela boa disposição e vontade com que me prestaram toda a ajuda necessária, que foi uma enorme contribuição para este trabalho.

Não posso igualmente deixar de agradecer aos meus amigos e colegas com quem partilhei inúmeras experiências e bons momentos ao longo destes anos, sem os quais este percurso não teria o mesmo significado. A vocês ficarei para sempre grato.

Por fim, deixo o meu agradecimento à minha família, à minha irmã, aos meus pais, por todo o apoio incondicional, pela possibilidade de cumprir este objetivo, pelos conselhos, e por acreditarem sempre no meu sucesso.

Resumo

Os incêndios florestais são fenómenos recorrentes todos os anos em Portugal, com efeitos devastadores para a população e economia, e uma das medidas para a sua prevenção passa pela redução do material vegetal combustível em certas zonas da floresta, o que se torna uma tarefa difícil através de métodos convencionais. Este trabalho, no âmbito de projetos como o F4F e E-Forest em desenvolvimento pela ADAI e pelo Field Tech Laboratory, visa possibilitar a deteção e classificação de obstáculos à navegação de uma máquina autónoma em ambiente florestal, que desempenha a função primária de remoção do combustível florestal em excesso. Utilizando uma abordagem de visão computacional com recurso a uma arquitetura de Rede Neural Convolutiva (CNN), entre outras ferramentas, é possível realizar deteção e classificação de objetos na informação visual retirada do ambiente em redor por diferentes sensores instalados. Esta informação é diferenciada em classes principais de obstáculos, como troncos de árvores, seres vivos e combustível florestal. A correta identificação destas classes é essencial para fornecer dados fidedignos que possam ser utilizados pelos controladores de movimento e navegação da máquina. Foi selecionado e treinado o modelo CNN YOLOv5 em *datasets* destinados à deteção de troncos, combustível florestal e cursos de água. Foi também testada a deteção de pessoas com recurso a imagens a cores e térmicas. O *dataset* mais adequado para a deteção de troncos foi o “Trees_dataset”, que obteve uma mAP50 de 94%, ao executar o modelo nas imagens de teste. A deteção de combustível florestal e cursos de água obtiveram resultados razoáveis, embora melhoráveis, com uma mAP50 nas imagens de teste de 68 e 79%, respetivamente. A deteção de pessoas com imagens a cores e térmica obteve resultados positivos, onde o modelo atingiu confianças de 40 a 50%, mesmo quando a pessoa na imagem estava a uma distância considerável. A ferramenta Adaptive Clustering integrada no ROS foi testada para a deteção de obstáculos com o LiDAR, e teve os resultados esperados, detetando os obstáculos definidos.

Palavras-chave: máquina autónoma, visão computacional, deteção de objetos, Rede Neural Convolutiva (CNN), ambiente florestal.

Abstract

Forest fires are recurring phenomena every year in Portugal, with devastating effects for the population and economy, and one of the measures for their prevention goes through the reduction of combustible vegetal material in certain areas of the forest, which becomes a strenuous task using conventional methods. This work, in the scope of projects as F4F and E-Forest, currently in development by ADAI and Field Tech Laboratory, proposes to enable the detection and classification of obstacles in the navigation path of an autonomous machine in a forestry environment, whose primary job is to remove the excess combustible vegetation. Using a computer vision approach with resource to Convolutional Neural Networks (CNN), as well as other tools, it is possible to perform obstacle detection and classification on the visual information retrieved from the surrounding environment by the different sensors installed. This information is distinguished into major classes of obstacles, as tree trunks, living beings and combustible vegetation, for example. The correct labelling of these classes is essential to allow the supply of reliable data which can that way be used by the machine's movement and navigation controllers. YOLOv5 CNN model was selected and trained on datasets aimed at the detection of tree trunks, combustible vegetation, and water lines. People detection with resource to colour and thermal imaging was also tested. The most adequate dataset for trunk detection was "Trees_dataset", which obtained a mAP50 value of 94% when the model was executed on the test images. The detection of combustible vegetation and water lines attained reasonable results, albeit improvable, with a mAP50 value for the test images of 68 and 79%, respectively. The detection of people with colour and thermal imaging had positive results, where the model achieved confidence values of 40 to 50% while the person on the image was at a considerable distance. The ROS integrated tool, Adaptive Clustering, was tested for obstacle detection with the LiDAR, and had the expected results, detecting the defined obstacles.

Keywords: autonomous machine, computer vision, object detection, Convolutional Neural Network (CNN), forest environment.

Índice

Índice de Figuras	ix
Índice de Tabelas	xi
Siglas	xiii
1. INTRODUÇÃO.....	15
1.1. Motivação	15
1.2. Objetivos.....	15
1.3. Metodologia.....	16
2. Estado de Arte	17
2.1. Visão computacional.....	17
2.2. Redes Neurais Convolucionais	17
2.3. Segmentação semântica e deteção de obstáculos.....	22
2.4. Avaliação do desempenho e resultados	28
3. Métodos para a deteção de obstáculos.....	38
3.1. Hardware.....	38
3.2. Recolha de dados	38
3.3. Escolha do modelo CNN	40
3.4. Descrição dos <i>datasets</i>	41
3.4.1. Deteção de troncos.....	42
3.4.2. Deteção de seres vivos.....	45
3.4.3. Deteção de combustível florestal.....	45
3.4.4. Deteção de cursos e planos de água.....	46
3.5. Descrição do treino	47
3.5.1. Treino deteção de troncos.....	48
3.5.2. Treino deteção de combustível florestal.....	51
3.5.3. Treino deteção de cursos de água	51
4. Resultados.....	53
4.1. Escolha do modelo.....	53
4.2. Resultados do Treino	53
4.2.1. Resultados <i>datasets</i> de troncos	54
4.2.2. Resultados <i>dataset</i> <i>Vegetation_detection</i>	57
4.2.3. Resultados <i>dataset</i> <i>Water_detection</i>	58
4.3. Avaliação dos modelos treinados.....	59
4.3.1. Avaliação deteção de troncos	60
4.3.2. Avaliação deteção de seres vivos	64
4.3.3. Avaliação deteção de combustível florestal	65
4.3.4. Avaliação deteção de cursos de água	67
4.4. Deteção de obstáculos com o LiDAR.....	68
5. Conclusões e Trabalho Remanescente	69
REFERÊNCIAS BIBLIOGRÁFICAS	71
Anexo A.....	77

ÍNDICE DE FIGURAS

Figura 2.1: Resultados qualitativos para MobileNetv2 (esquerda), ResNet50 (centro) e Adapnet++ (direita) em comparação com o ground truth (em cima) [2].	23
Figura 2.2: Classificação numa imagem do <i>dataset</i> Oporto [3].	24
Figura 2.3: Exemplo de imagem anotada para o treino, a área denotada a vermelho representa a zona transponível pelo robô [18].	24
Figura 2.4: Exemplo de mapa de transponibilidade estimada [5].	26
Figura 2.5: Deteção estimada pelo modelo visual [32].	27
Figura 2.6: Ilustração da métrica IoU [54].	28
Figura 2.7: Resultados do SLAM semântico [3].	31
Figura 2.8: Evolução do F1-score para diferentes valores de confiança [33].	36
Figura 3.1: Configuração dos sensores para a recolha de dados	39
Figura 3.2: Posição dos sensores e cenário da recolha de dados.	40
Figura 3.3: Exemplo de imagem do “Trees_dataset” utilizada no treino.	43
Figura 3.4: Exemplo de imagem do “TreeDetection_dataset” utilizada no treino.	43
Figura 3.5: Exemplo de imagem do “Woods_dataset” utilizada no treino.	44
Figura 3.6: Exemplo de imagem do <i>dataset</i> “Trees_augmented” utilizada no treino.	44
Figura 3.7: Exemplo de imagem do <i>dataset</i> “Vegetation_detection” utilizada no treino.	46
Figura 3.8: Exemplo de imagem do <i>dataset</i> “Water_detection” utilizada no treino.	47
Figura 3.9: Exemplo de comando de treino dado ao YOLOv5 no <i>notebook</i> do Google Colab	47
Figura 3.10: Exemplo da disposição do texto dentro dos ficheiros das anotações utilizadas pelo YOLO.	49
Figura 4.1: Evolução da mAP50 ao longo dos treinos com o Trees_dataset.	54
Figura 4.2: Evolução da mAP50 ao longo dos treinos com o Woods_dataset.	55
Figura 4.3: Evolução da mAP50 ao longo dos treinos com o TreeDetection_dataset.	56
Figura 4.4: Evolução da mAP50 ao longo do treino com o Trees_augmented.	57
Figura 4.5: Evolução da mAP50 ao longo do treino com o Vegetation_detection.	58
Figura 4.6: Evolução da mAP50 ao longo dos treinos com o Water_detection.	59
Figura 4.7: Imagem retirada do vídeo a cores utilizado no teste qualitativo do Trees_dataset_Colab_Fine.	61
Figura 4.8: Imagem retirada do vídeo térmico utilizado no teste qualitativo do Trees_dataset_Colab_Fine.	61

Figura 4.9: Imagem retirada do vídeo utilizado no teste qualitativo do Trees_dataset_Colab_Fine.	62
Figura 4.10: Imagem retirada do vídeo utilizado no teste qualitativo do TreeDetection_dataset_Colab_Fine.	63
Figura 4.11: Imagem retirada do vídeo utilizado no teste qualitativo do Trees_augmented.	64
Figura 4.12: Imagem retirada do vídeo térmico utilizado no teste qualitativo do YOLOv5- small pré treinado.	65
Figura 4.13: Imagem retirada do vídeo utilizado no teste qualitativo do modelo treinado no Vegetation_detection original.	66
Figura 4.14: Imagem retirada do vídeo utilizado no teste qualitativo do modelo treinado no Vegetation_detection alterado.	66
Figura 4.15: Exemplo de imagem de teste do Water_detection em que o modelo foi executado.	67
Figura 4.16: Visualização #1 no RVIZ do output gerado pelo Adaptive Clustering nos dados da floresta.	68
Figura A.1: Visualização no RVIZ dos dados obtidos com o Velodyne (esquerda, cima) e com a Realsense (direita, cima), e imagens retiradas dos vídeos a cores (esquerda, baixo) e térmico (direita, baixo), a 2,5m dos troncos.	77
Figura A.2: Cenário da recolha de dados com o LiDAR no interior do edifício do LAI. ...	77
Figura A.3: Exemplo do código em Python utilizado para a alteração das classes nos ficheiros das anotações.	78
Figura A.4: Visualização no RVIZ do output gerado pelo Adaptive Clustering nos dados captados na floresta, com ênfase na ilustração dos troncos detetados.	78
Figura A.5: Visualização no RVIZ do output gerado pelo Adaptive Clustering nos dados captados no LAI.	79
Figura A.6: Imagem retirada do vídeo a cores utilizado no teste qualitativo do YOLOv5- small pré treinado.	79

ÍNDICE DE TABELAS

Tabela 2.1: Resultados das médias das métricas de avaliação [2].	30
Tabela 2.2: Complexidade e tempo de inferência para imagens 640x480 [2].	30
Tabela 2.3: Resultados do treino [18]......	32
Tabela 2.4: Tempo de processamento dos modelos DeepLabv3+ e YOLOv5 em imagens 1440 x 1080 [22].	32
Tabela 2.5: Resultados de AP e F1 da experiência #3 considerando todas as detecções (confiança = 0%) [31].	34
Tabela 2.6: Tempos de inferência médios dos detetores em diferente <i>hardware</i> [31]......	34
Tabela 3.1: Número de imagens de cada <i>dataset</i> e sua respectiva repartição para treino, validação e teste.....	42
Tabela 3.2: Parâmetros dos treinos com o Woods_ <i>dataset</i>	50
Tabela 3.3: Parâmetros dos treinos com o TreeDetection_ <i>dataset</i>	50
Tabela 3.4: Parâmetros dos treinos com o Vegetation_ <i>detection</i>	51
Tabela 3.5: Parâmetros dos treinos com o Water_ <i>detection</i>	52
Tabela 4.1. Comparação de métricas de avaliação e tempo de inferência para os modelos YOLOv5 e v7 no MS COCO128 (GTX 1660 SUPER).	53
Tabela 4.2: Valores das métricas de avaliação dos treinos com o Trees_ <i>dataset</i>	54
Tabela 4.3: Valores das métricas de avaliação dos treinos com Woods_ <i>dataset</i>	55
Tabela 4.4: Valores das métricas de avaliação dos treinos com o TreeDetection_ <i>dataset</i> . ..	56
Tabela 4.5: Valores das métricas de avaliação do treino com o Trees_ <i>augmented</i>	57
Tabela 4.6: Valores das métricas de avaliação dos treinos com o Vegetation_ <i>detection</i> . ..	58
Tabela 4.7: Valores das métricas de avaliação dos treinos com o Water_ <i>detection</i>	59
Tabela 4.8: Valores das métricas de avaliação e tempo de execução nas imagens de teste do Trees_ <i>dataset</i>	60
Tabela 4.9: Valores das métricas de avaliação e tempo de execução nas imagens de teste do TreeDetection_ <i>dataset</i>	62
Tabela 4.10: Valores das métricas de avaliação e tempo de execução nas imagens de teste do Trees_ <i>augmented</i>	63
Tabela 4.11: Valores das métricas de avaliação e tempo de execução nas imagens de teste do Vegetation_ <i>detection</i>	66
Tabela 4.12: Valores das métricas de avaliação e tempo de execução nas imagens de teste do Water_ <i>detection</i>	67

SIGLAS

AP – Average Precision

CNN – Convolutional Neural Network

FN – False Negative

FP – False Positive

GPU – Graphics Processing Unit

IMU – Inertial Measurement Unit

LAI – Laboratório de Aerodinâmica Industrial

LiDAR – Light Detection and Ranging

RAM – Random Access Memory

RGB-D – Red Green Blue - Depth

ROS – Robot Operating System

SLAM – Simultaneous Localization and Mapping

TP – True Positive

UC – Universidade de Coimbra

1. INTRODUÇÃO

1.1. Motivação

Todos os anos a Europa tem sido fustigada por números elevadíssimos de incêndios florestais de grandes dimensões que consomem centenas de milhares de hectares de território florestal, ameaçando populações que têm sofrido perdas inaceitáveis de vidas humanas e prejuízos imensos. Os países do sul europeu com clima mediterrânico são os mais afetados, principalmente Portugal, que lidera em anos consecutivos no número de ocorrências de incêndio [1]. Uma das medidas mais eficazes para a prevenção destes incêndios, passa pela criação de faixas de gestão de combustível em redor de aglomerados habitacionais, ao longo de estradas, caminhos ferroviários, linhas de alta tensão, entre outras infraestruturas inseridas na interface urbano florestal. Um dos principais problemas na implementação destas medidas é falta de mão de obra, tal como o custo que esta acarreta e o risco a que os trabalhadores podem estar sujeitos em certas situações, como o trabalho em encostas com grande inclinação. Por outro lado, há o desenvolvimento atual das tecnologias de inteligência artificial e *deep learning*, principalmente as redes neurais convolucionais (CNN), que têm sido imensamente estudadas e utilizadas em aplicações de visão computacional para o reconhecimento de objetos e segmentação semântica de imagem [21]. Estas circunstâncias têm despertado o interesse na investigação e desenvolvimento de máquinas autónomas com capacidade de operar em ambientes florestais complexos, incluindo para o desempenho da função de facilitar as operações de redução do combustível florestal em excesso, como prevenção dos incêndios [1].

1.2. Objetivos

Na implementação de uma máquina autónoma em ambiente florestal, um dos principais desafios para a sua operação em segurança e cumprimento eficaz da sua função, é a robusta deteção e classificação em tempo real de obstáculos à sua navegação [2]. O objetivo deste trabalho é a criação de mecanismos e implementação de algoritmos e ferramentas computacionais que permitam tirar partido dos diferentes sensores que a máquina tem disponíveis, de forma a permitir o reconhecimento do ambiente em redor, e a

capacidade de classificar as entidades detetadas em diferentes classes: combustível florestal, troncos de árvores, cursos de água e seres vivos. A correta identificação e classificação dos objetos proporciona a possibilidade de a máquina apresentar uma resposta adequada relativamente a cada um, garantindo uma operação em segurança.

1.3. Metodologia

Os sensores utilizados incluem um LiDAR 3D que fornece dados em forma de uma nuvem de pontos, descrevendo o ambiente 360° em seu redor, uma câmara de visão stereo que fornece informação em RGB-D, ou seja, imagens a cores convencionais e perceção de profundidade, e também uma câmara de infravermelhos.

Para a deteção do combustível a remover pretende-se utilizar a imagem a cores da câmara de visão stereo (RGB-D) como input para o algoritmo de uma arquitetura de rede neural convolucional (CNN), que processa a imagem e efetua segmentação semântica ou deteção de objetos, devolvendo-a com as classes identificadas. Para isto o modelo CNN é treinado num *dataset* adequado e posteriormente avaliado quanto à sua eficácia.

Os obstáculos à navegação podem ser detetados recorrendo aos dados obtidos pelos três sensores, imagem a cores e informação de profundidade proveniente da câmara de visão stereo, a nuvem de pontos gerada pelo LiDAR e a imagem térmica proveniente da câmara de infravermelhos. Estes dados são depois processados por ferramentas e pelo modelo CNN que permitem a deteção de obstáculos físicos que se possam opor à passagem da máquina, como árvores e cursos de água profundos. À semelhança da deteção do combustível florestal, o modelo CNN utilizado para a deteção destes objetos, vai ser treinado em *datasets* selecionados e o seu desempenho avaliado.

Os seres vivos são detetados de forma semelhante ao combustível, desta vez recorrendo também à imagem térmica da câmara de infravermelhos, para além da imagem a cores da câmara de visão stereo, servindo de input ao modelo CNN pré treinado para a deteção das classes que se pretendem, neste caso o uso da imagem térmica adiciona redundância à capacidade de deteção em situações de pobre visibilidade da câmara a cores, que podem ocorrer durante o trabalho da máquina, como a presença de poeira, fumo e outras partículas em suspensão no ar.

2. ESTADO DE ARTE

Existem estudos para a implementação de máquinas autônomas em ambientes florestais com uma variedade complexa de vegetação e ambiente destruturado, por exemplo Couceiro et al. [1], Andrada et al. [2], Russel et al. [3], Lourenço et al. [4], Nasir et al. [5], onde um dos principais desafios é a detecção e classificação de obstáculos, função essencial à sua navegação segura. Para além disso, no caso de uma máquina autônoma cujo objetivo é a remoção do material combustível em excesso, é também necessário a correta identificação deste.

2.1. Visão computacional

Nesta área da inteligência artificial, são abordados métodos para a percepção e compreensão de imagens por computadores, provenientes de câmaras e sensores, de forma a criar um conhecimento do ambiente capturado para que se possa identificar elementos de interesse e/ou à detecção de obstáculos como assistência à navegação autônoma, por exemplo.

As Redes Neurais Convolucionais (CNN) são a abordagem mais comum em aplicações de visão computacional para detecção e segmentação semântica de objetos, devido ao seu elevado desempenho na detecção e classificação, como é demonstrado na competição de reconhecimento de imagens Imagenet [16].

2.2. Redes Neurais Convolucionais

As Redes Neurais Convolucionais (CNN) são arquiteturas cuja estrutura é inspirada pelos neurónios dos cérebros humano e animal, nomeadamente a sequência de células que compõem o córtex visual. São muito utilizadas em *deep learning*, principalmente no processamento e análise de imagens. As CNN diferem de outras redes neurais no aspeto que apresentam séries de camadas convolucionais e permitem identificar e classificar elementos/objetos numa imagem inicial, com a aprendizagem do algoritmo sobre a importância atribuída a esses elementos/objetos através de treino. Têm a vantagem de reduzir significativamente o poder computacional necessário em relação a algoritmos definidos manualmente, uma característica essencial à segmentação semântica e detecção de objetos em tempo real para navegação autônoma [17].

Andrada et al. [2], analisaram diferentes arquiteturas CNN para segmentação semântica e registaram o seu desempenho, considerando uma aplicação em navegação de robótica florestal. Utilizaram Bonnetal [6], uma ferramenta *open source* que visa simplificar o treino e implementação de arquiteturas CNN para segmentação semântica de imagens e vídeo através do ROS (Robot Operating System) [52]. Efetuaram testes com as arquiteturas MobileNetV2 [7], ResNet50 [8], assim como uma arquitetura [9] com o codificador Adapnet++ e o decodificador eASPP, adaptados ao projeto.

Le et al. [18] implementaram as arquiteturas ERFNet [19] e Darknet (YOLOv3) [20], para além da ResNet [8] e da MobileNetV2 [7], no estudo de um robô autónomo capaz de seguir fileiras em horticultura de ambiente variado, como estufas, pérgulas e pomares.

Mendes et al. [22] fizeram um estudo para a classificação de vegetação florestal a ser limpa por uma máquina autónoma, comparando os modelos YOLOv5 [24] e DeepLabv3+ [23] através de um teste na classificação de um *dataset* de imagens, adquirido pelos autores, com o objetivo de determinar o modelo mais adequado a esta tarefa. Este *dataset*, após anotação manual de *bounding boxes*, facilitada pela ferramenta do Roboflow [55] foi utilizado para treinar o YOLOv5 durante 1000 *epochs* com recurso ao Google Colab [56]. As classes anotadas consistiam em “Live Vegetation”, “Grass”, “Cutted-Vegetation”, “Tree-trunk” e “Dead Vegetation”. O modelo DeepLabv3+ usado estava pré-treinado no *dataset* CityScapes [57] pelo que foi aplicado diretamente no *dataset* dos autores sem treino adicional. A principal diferença entre as duas arquiteturas testadas é a forma como classificam as imagens, já que o YOLOv5 fez deteção de objetos e o DeepLabv3+ efetua segmentação semântica da imagem completa.

da Silva et al. [31] testaram as arquiteturas SSD [37], SSDLite [7], MobileNetV2 [7], ResNet50 [8], Inception-v2 [38], MobileDet [39] e YOLOv4 [40] para a capacidade de reconhecer troncos de árvores, principalmente eucaliptos e pinheiros, a ser evitados por uma máquina com navegação autónoma. Os modelos foram treinados e testados com imagens a cores no espectro visível e imagens térmicas captadas em florestas de Portugal, organizadas num *dataset* anotado que foi aumentado através de técnicas como rotação da imagem, alterações na escala, saturação, contraste, entre outras, tendo sido disponibilizado publicamente pelos autores juntamente com o original. Todos os modelos testados beneficiaram de *transfer learning* ao serem utilizados ficheiros *weights* pré-treinados no *dataset* MS COCO.

da Silva et al. [33], neste trabalho posterior a [31], testaram 13 diferentes modelos de detecção de objetos e testaram a sua implementação em 4 diferentes plataformas de processamento móveis, na tentativa de obter uma detecção robusta de troncos de árvores em tempo real, assim como ao seu mapeamento recorrendo a uma câmara de profundidade. Os modelos selecionados foram o SSD, MobileNetV1 [58], V2 e V3-Small/Large [59], EfficientDet Lite0, 1, 2 [60], YOLOv4 Tiny, YOLOv5 Nano/Small, YOLOv7 Tiny [43, 44], YOLOR-CSP [61, 62] e DETR-ResNet50 [63]. O *dataset* utilizado pelos autores corresponde ao obtido no seu trabalho anterior [31], com a adição de mais 2430 imagens e anotações, sendo este também disponibilizado publicamente pelos autores. Para o treino dos modelos a testar, o *dataset* sofreu ainda um acréscimo através das técnicas de aumentação já aplicadas no estudo prévio. As plataformas de processamento móveis testadas consistiram num Raspberry Pi 4B, uma NVIDIA Jetson Nano, um Google Coral USB Accelerator e uma plataforma Intel OAK_D que inclui uma IMU, câmara a cores e visão stereo, e uma unidade de processamento capaz de executar a rede neural para detecção de objetos, tudo no mesmo dispositivo.

ResNet50

ResNet (Residual Network) [8] é uma arquitetura CNN muito utilizada em aplicações com visão computacional e foi desenvolvida de forma a suportar o treino de redes neurais profundas com centenas de camadas. ResNet50 é apenas uma variante desta arquitetura que apresenta 50 camadas. As arquiteturas residuais adicionam atalhos na conexão entre as camadas, que em vez de processar a informação sequencialmente, permitem saltar várias camadas. As vantagens destas conexões é que não aumentam a complexidade computacional nem adicionam parâmetros extra. Esta arquitetura não apresenta um aumento de erros de treino com o aumento da profundidade da rede neural, mostrando por outro lado, melhorias na precisão com o aumento da profundidade. [8,34]

MobileNet

As MobileNets [58] são arquiteturas CNN leves e eficientes, desenvolvidas com o intuito de obter bons valores de desempenho em aplicações com dispositivos móveis, cujo poder de processamento é mais baixo, efetuando um balanço entre tempo de processamento e precisão. São formadas por convoluções separáveis em profundidade (*depth-wise*

separable convolutions) que tornam o modelo mais compacto e com um desempenho computacional superior [7, 31, 32, 58, 59]

Adapnet++

Adapnet++ [9] é uma arquitetura CNN recente projetada para o uso em visão computacional de forma a efetuar segmentação semântica de imagens, função muito útil na navegação de robôs autônomos com visão computacional. É descrita como compacta e pode ser treinada utilizando uma única GPU de consumidor [9].

ERFNet

Tal como a MobileNetV2 [7], a ERFNet [19] aponta em reduzir drasticamente o poder computacional necessário sem comprometer a qualidade dos resultados, mantendo ainda assim uma boa precisão na segmentação semântica. A sua arquitetura baseia-se em ligações residuais e convoluções fatorizadas, o que permite atingir esse balanço de desempenho [19].

Darknet - YOLO

A arquitetura Darknet [50] serve de suporte para o algoritmo base YOLO [49], e tal como a ResNet50 [8] e a ERFNet [19], utiliza conexões residuais. É escrita em C e CUDA, suportando a execução em CPU e GPU. Este modelo muito popular, efetua identificação de objetos numa imagem, que identifica no output visual através de caixas coloridas envolvendo os mesmos, e com uma legenda da classe assim como a confiança da previsão em percentagem decimal [22].

DeepLab

O modelo DeepLab [23] foi desenvolvido para efetuar segmentação semântica de imagens, identificando no seu output visual os pixels da imagem classificados numa certa categoria. Nas iterações mais recentes é possível utilizar várias arquiteturas como algoritmo base, como a MobileNet [7] e ResNet [8], contendo também um módulo com estrutura ASPP [22].

SSD

O SSD [37] (Single Shot Detector) é um modelo de rede neural, desenhado para a detecção de objetos numa imagem, atribuindo caixas que envolvem a entidade de interesse, incluindo a sua classe e confiança de detecção, à semelhança do YOLO [49]. É um modelo rápido que utiliza a VGG16 [42] como arquitetura base.

SSD Lite

O SSD Lite [7] foi proposto pelos autores da MobileNetV2 e é projetado para a utilização em dispositivos móveis, beneficiando de um custo computacional reduzido e menor número de parâmetros em relação ao SSD [37], [31].

Inception-v2

Inception-v2 [38] é a segunda versão do modelo Inception e introduz a característica *batch-normalization* que ajuda a um treino mais rápido da rede neural. Esta característica adiciona apenas mais dois parâmetros por ativação e leva em consideração o fenómeno de *covariate shift* que complica o processo de treino. Isto levou a que, na data do seu lançamento, este modelo tenha ultrapassado outros existentes, precisando de menos etapas de treino [31].

MobileDet

O modelo MobileDet [39] é bastante recente e é também projetado para a aplicação em dispositivos móveis. Efetua detecção de objetos utilizando o mesmo método que o YOLO [49] e o SSD [37], por exemplo.

EfficientDets

Os modelos EfficientDet [60], tal como os modelos descritos anteriormente, são dirigidos a plataformas de processamento móveis, sendo projetados para uma maior eficiência e desempenho computacional do que outros detetores que precedem a sua criação. Estes modelos trazem melhorias como, por exemplo, um método que dimensiona a

profundidade, envergadura e resolução para todos os componentes da arquitetura em simultâneo [33].

2.3. Segmentação semântica e deteção de obstáculos

O objetivo da segmentação semântica é dividir, ou segmentar, os pixéis numa dada imagem nas diferentes categorias semânticas definidas pelo utilizador, para posterior processamento e análise. Esta tecnologia potenciada por algoritmos de *deep learning* é interessante nas mais variadas aplicações, como a navegação de veículos autónomos, vídeo vigilância, análise de imagens de satélite, visão computacional e até na análise de imagens médicas para a identificação de várias doenças. Existem desenvolvimentos para a fusão da informação proveniente de diferentes sensores de forma a aumentar a precisão da segmentação, como câmaras multiespectrais, infravermelhos, de profundidade, e sensores LiDAR que permitem obter uma geometria 3D do ambiente em redor [21].

Algumas arquiteturas CNN, como o YOLO [49], ao invés de segmentação semântica por pixel, efetuam deteção de objetos, fornecendo uma imagem com caixas sobrepostas a esta, delimitando as zonas da imagem em que foi identificada a classe que se pretende. Isto permite uma redução significativa no tempo de processamento por imagem em relação à segmentação semântica completa da mesma, tornando estes modelos mais adequados à visão computacional em tempo real, principalmente quando são executados em unidades de processamento móveis com baixo consumo de energia [22].

Em [2], Andrada et al. recorreram ao *dataset* SEMFIRE, com 500 imagens obtidas e anotadas pelos mesmos autores. As imagens foram recolhidas com recurso a uma câmara multiespectral em florestas de Portugal, que forneceu informação visual em formato NGR (canais Near Infrared (NIR), Red e Green). Estas imagens foram posteriormente anotadas manualmente, segmentando os pixéis em 6 diferentes classes: plano de fundo, material combustível, copas das árvores, troncos, humanos e animais. Os autores avaliaram depois o desempenho de diferentes modelos CNN neste *dataset*, efetuando uma segmentação semântica da imagem, como se pode observar na Figura 2.1.

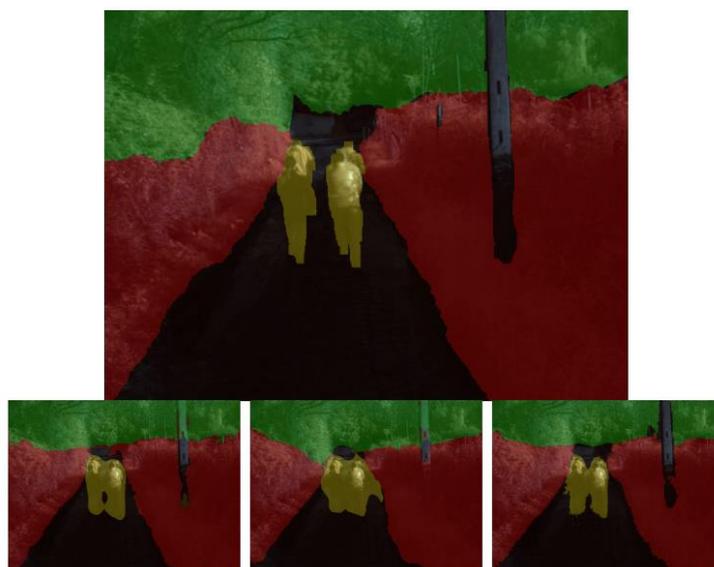


Figura 2.1: Resultados qualitativos para MobileNetv2 (esquerda), ResNet50 (centro) e Adapnet++ (direita) em comparação com o ground truth (em cima) [2].

Em [3], Russel et al. equiparam um *drone* com duas câmaras RGB alinhadas em configuração stereo, separadas por 0,25m, e um LiDAR 3D com o objetivo de mapear a floresta e de identificar o combustível florestal a ser removido por um veículo terrestre autónomo. As imagens recolhidas, pertencentes a um *dataset* que os autores designaram de Oporto, foram segmentadas com recurso a uma arquitetura CNN em 4 categorias: plano de fundo, material combustível, copas das árvores e troncos. Os autores realizaram depois a fusão destas imagens classificadas com os dados da nuvem de pontos obtida pelo LiDAR e com a odometria do *drone*, modificando uma metodologia de segmentação semântica RGB-D – SLAM [53] de forma a projetar as deteções das imagens para o domínio 3D da nuvem de pontos. Utilizando as propriedades extrínsecas do LiDAR em relação à câmara, os dados obtidos pelo primeiro sensor são transformados no sistema de coordenadas da câmara. De seguida, com as propriedades intrínsecas da câmara calibrada projetaram cada ponto obtido pelo LiDAR no plano da imagem. Os pontos que se encontram dentro do campo de visão da câmara têm uma classe atribuída correspondente à do seu pixel equivalente no mapa semântico. A nuvem de pontos segmentada é por último transformada no sistema inercial de referência com recurso à pose do *drone*, estimada pelo sistema de SLAM. Na Figura 2.2 encontra-se a representação da segmentação semântica obtida pelos autores na imagem RGB de uma das câmaras pela execução da arquitetura CNN, onde o combustível florestal se encontra a vermelho, os troncos das árvores a castanho, as copas das árvores a verde, e o fundo a preto.



Figura 2.2: Classificação numa imagem do *dataset* Oporto [3].

Em [18], Le et al., utilizaram uma câmara RGB-D instalada num robô autónomo, permitindo-lhes adquirir imagens, anotá-las, e posteriormente treinar, testar e comparar o desempenho de diferentes arquiteturas CNN para a identificação de um corredor de passagem entre fileiras de horticultura. As imagens foram anotadas de forma a demarcar os corredores e zonas onde o robô pode transitar com segurança, encontrando-se na Figura 2.3 um exemplo destas.



Figura 2.3: Exemplo de imagem anotada para o treino, a área denotada a vermelho representa a zona transponível pelo robô [18].

Brandenburg et al. [30] recorreram de forma semelhante a uma câmara RGB-D e ao algoritmo YOLOv3 [20], a ser executado numa plataforma de desenvolvimento Xilinx ZCU104, que foi treinado em 1454 imagens de várias fontes para a deteção de morangos,

num estudo para a implementação de um robô que faça a sua colheita de forma autónoma em ambientes variados e condições de luminosidade inconstantes.

Em [5], Couceiro et al. abordaram a localização e navegação de uma máquina autónoma pesada em ambiente florestal, que para obter informação dos seus arredores é equipada com dois LiDAR 3D, e uma câmara de profundidade RGB-D. De forma a realizar deteção de obstáculos, utilizaram várias ferramentas que comunicam entre si numa pipeline de processamento do ROS, permitindo o planeamento da rota otimizada a tomar pelo robô. Para a localização da máquina foi utilizada a ferramenta cartógrafo [11], que faz uso da informação fornecida pela ferramenta de odometria visual RTAB-Map [12] e dos sensores LiDAR e câmara RGB-D. O cartógrafo estima a pose da máquina, propriedade necessária à criação de um mapa de elevação para determinar a transponibilidade do terreno [13], cujo exemplo obtido durante um teste da máquina, está representado na Figura 2.4 . Um mapa de 40m x 40m foi criado com o pacote ROS para mapeamento de elevação, cujo nóduo fornece na verdade dois mapas de elevação, um mapa em bruto e um mapa refinado. Para estimar os espaços transponíveis é utilizado o mapa refinado [14], e um mapa de transponibilidade 2D é calculado a partir de três diferentes mapas 2D, sendo eles, declive, degrau e rugosidade/irregularidade. O mapa de transponibilidade é depois utilizado pelo algoritmo RRT [15] na identificação de fronteiras para exploração do robô. Estas fronteiras, juntamente com a informação sobre a pose da máquina, são depois recebidas pelo pacote de navegação do ROS [26] onde está configurado um planeador global de navegação no mapa de transponibilidade, A* [27], do qual está dependente um planeador local de navegação, TEB [28],[29], atualizado pelo algoritmo de *voxel* 3D STVL [25] para a deteção de obstáculos na trajetória da máquina.

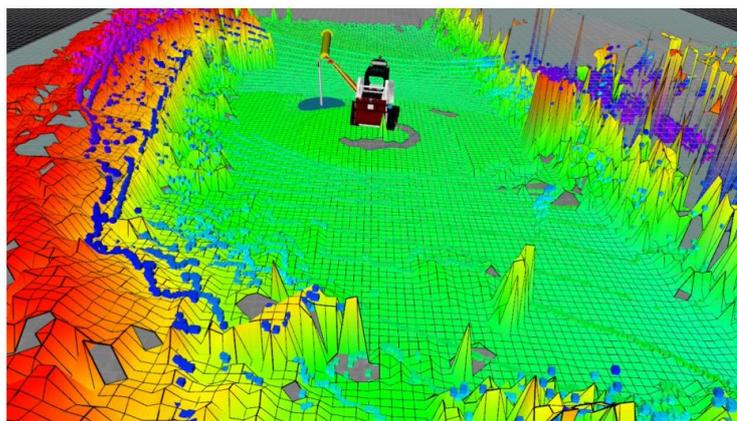


Figura 2.4: Exemplo de mapa de transponibilidade estimada [5].

Em [32], Grondin, et al. implementaram e modificaram algoritmos, criando modelos para a deteção de troncos de árvores em ambiente florestal, fornecendo também uma estimativa do diâmetro do tronco, da sua inclinação e do ponto na base do tronco onde efetuar o corte, com o objetivo de permitir a utilização numa máquina autónoma para o abate. Na Figura 2.5 encontra-se uma representação exemplo do resultado obtido pelos autores, com o ponto de vista a partir da cabine de uma máquina florestal de abate numa floresta do Canadá, onde cada árvore é identificada com uma cor diferente, caixa envolvente, máscara de segmentação e pontos chave (pontos vermelhos). São estes pontos chave anotados nas imagens de treino que permitem posteriormente estimar o diâmetro e inclinação das árvores, tal como a zona onde se deve executar o corte. Para o treino e teste dos algoritmos recorreram a três diferentes *datasets* de ambiente florestal, os quais designaram de SynthTree43k, CanaTree100 e Portugal. O primeiro, tal como o nome sugere, consiste em 43 000 imagens RGB-D sintéticas criadas com recurso a um popular motor gráfico de jogos de vídeo. O CanaTree100 é um *dataset* com imagens reais recolhidas em florestas do Canadá e anotadas, resultando em 100 imagens RGB-D com mais de 920 árvores representadas. O *dataset* Portugal corresponde ao criado por da Silva et al. [31], que foi implementado com a finalidade de testar os modelos desenvolvidos num domínio diferente do qual foram treinados.



Figura 2.5: Detecção estimada pelo modelo visual [32].

Andrada et al. [34] propuseram e testaram a implementação de uma câmara multiespectral e um sensor LiDAR 3D para a identificação e localização do combustível florestal em excesso a ser removido. Para a segmentação semântica recorreram à arquitetura Adapnet++, selecionada pelos mesmos autores em [2] devido ao seu desempenho, que processa a imagem multiespectral fornecida nos canais *near infrared*, *red* e *green*, e atribui uma de 6 classes aos pixéis. De forma a utilizar a percepção de profundidade, o LiDAR e a câmara foram calibrados para extrair os parâmetros intrínsecos e extrínsecos. Um padrão em xadrez forneceu a informação intrínseca à câmara e marcadores ArUco forneceram a informação 3D com as coordenadas da câmara no espaço. Com essas coordenadas e os marcadores ArUco, definiram a correspondência entre a câmara e o LiDAR, e obtiveram os parâmetros extrínsecos recorrendo ao algoritmo de Kabsch [35]. Com os sensores calibrados utilizaram matrizes de transformação estáticas para projetar a nuvem de pontos em coordenadas de pixéis, sendo que filtraram a nuvem de pontos para corresponder ao campo de visão da câmara, com o objetivo de reduzir o poder de processamento necessário. A projeção da nuvem de pontos no plano de imagem resulta num mapa de profundidade demasiado disperso, por isso utilizaram a ferramenta IP-Basic [36] para estimar os pontos em falta. Para a localização do combustível florestal, desenvolveram uma técnica onde uma pipeline de processamento recebe a imagem que sofreu segmentação semântica e calcula o centroide de cada mancha de combustível. Estes centroides têm depois as suas coordenadas

de pixel projetadas em coordenadas espaciais, que são posteriormente utilizadas pelos módulos de decisão e planeamento do movimento da máquina.

2.4. Avaliação do desempenho e resultados

Para a medição do desempenho das diferentes arquiteturas e seguinte comparação é comum utilizar a ferramenta Weights and Biases [10]. As diferentes métricas de medição do desempenho baseiam-se nos conceitos básicos de verdadeiros positivos, *True Positives* (TP), falsos positivos, *False Positives* (FP) e falsos negativos, *False Negatives* (FN). As métricas mais comuns para avaliar modelos de deteção de objetos são a Intersection over Union (IoU), Recall, Precision, F1 score e Average Precision (AP).

A Intersection over Union, ou índice de Jaccard, é obtida pela razão entre a área de sobreposição da caixa estimada com a caixa original (*ground truth*), e a área total delimitada pelas duas caixas, como ilustrado na Figura 2.6. IoU toma assim valores entre 0 e 1, com os valores mais próximos da unidade a manifestar uma previsão mais fidedigna. Este valor é depois comparado com um valor limite estabelecido x , o que define quando uma deteção é TP, FP ou FN. Por exemplo, $IoU \geq x$ corresponde a um TP, $IoU \leq x$ corresponde a um FP e $IoU = 0$ corresponde a um FN.

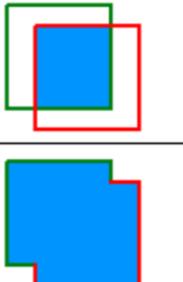
$$IoU = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{área de sobreposição}}{\text{área da união}}$$


Figura 2.6: Ilustração da métrica IoU [54].

A métrica Recall, equação (2.1), representa a capacidade do modelo para reconhecer os dados anotados (*ground-truth*). É dependente do limite estabelecido para a IoU e demonstra em percentagem a quantidade de previsões corretas em relação a todas as caixas originais (*ground-truth*) existentes. Se o número de falsos negativos, FN, for elevado, Recall toma um valor baixo.

$$Recall = \frac{TP}{TP + FN} \quad (2.1)$$

A métrica Precision, equação (2.2), representa a capacidade do modelo para representar apenas objetos relevantes. Demonstra em percentagem a quantidade de previsões corretas e tal como a Recall, depende do limite estabelecido para a IoU. Se o número de falsos positivos, FP, for elevado, Precision toma um valor baixo.

$$Precision = \frac{TP}{TP + FP} \quad (2.2)$$

A métrica F1 score, equação (2.3), combina as métricas Precision e Recall, calculando a sua média harmónica. Isto permite, por exemplo, comparar dois modelos, em que um deles apresente o valor de Recall mais elevado e o outro o valor de Precision mais elevado. Analisando o F1 score de cada um, é possível determinar o mais vantajoso. Com valores de Precision e Recall elevados, F1 toma também valores elevados.

$$F1 = \frac{2TP}{2TP + FN + FP} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (2.3)$$

A métrica Average Precision, equação (2.4) onde p é Precision e r é Recall, corresponde à área por baixo da linha do gráfico Precision-Recall. É calculada para cada classe de objeto e tal como as suas variáveis, o seu valor varia entre 0 e 1. Nos modelos treinados com o *dataset* MS COCO é comum utilizar variações de AP, calculadas para diferentes valores de IoU, como AP50, onde IoU corresponde a 50%, e AP@50:5:95 que é a média da AP para valores de IoU desde 50% até 95% com incrementos de 5%. A mean Average Precision (mAP), equação (2.5) onde N representa o número total de classes de deteção e AP_i o valor de AP para a respetiva i -ésima classe, corresponde apenas à média da AP de todas as classes de deteção [54].

$$AP = \int_0^1 p(r) dr \quad (2.4)$$

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (2.5)$$

Andrada et al. [2], no seu estudo, chegaram à conclusão de que a arquitetura com o melhor desempenho em geral é a Adapnet++ - eASPP pois é a única que apresentou reconhecimento de animais, apesar de a ResNet50 conseguir um desempenho ligeiramente superior no reconhecimento de humanos. A Tabela 2.1 mostra os valores das métricas de avaliação obtidas para os diferentes modelos.

Quanto ao tempo de execução de inferência, representado na Tabela 2.2, é mostrado como proporcional ao número de parâmetros de cada arquitetura, com o menor tempo para a MobileNetV2, seguida da Adapnet++ - eASPP e da ResNet50. Observaram também que o uso de *transfer learning* através de *weights* pré-treinados é benéfico para os resultados.

Tabela 2.1: Resultados das médias das métricas de avaliação [2].

Encoder	Decoder	mIoU	mRecall	mF1-Score
MobileNetv2	ASPP	0.5352	0.7672	0.6328
ResNet50	ASPP	0.5326	0.7283	0.6339
Adaptnet++	eASPP	0.6475	0.8967	0.7677

Tabela 2.2: Complexidade e tempo de inferência para imagens 640x480 [2].

Encoder	Decoder	# Parâmetros	Tempo de Inferência (ms)
MobileNetv2	ASPP	2 154 862	11.6
ResNet50	ASPP	49 652 118	53
Adapnet++	eASPP	32 345 870	40.9

Russel et al. [3], com a arquitetura CNN de segmentação semântica SegFormer [64], depois de treinada no *dataset* Sete Fontes, que está incluído no *dataset* SEMFIRE dos autores de [2], foi avaliada no *dataset* Oporto obtido pelos autores deste trabalho, não visto durante o treino, onde conseguiram um IoU de 78,2% para “combustível florestal” e 95,3% para

“não combustível”, traduzindo-se num IoU médio de 86,7%. Na Figura 2.7 representa-se a nuvem de pontos final conseguida após a fusão da segmentação da imagem RGB com a informação fornecida pelo LiDAR, onde o combustível florestal se encontra a vermelho, os troncos das árvores a verde, as copas das árvores a violeta, e o fundo a preto, com os pontos brancos de classe indefinida.

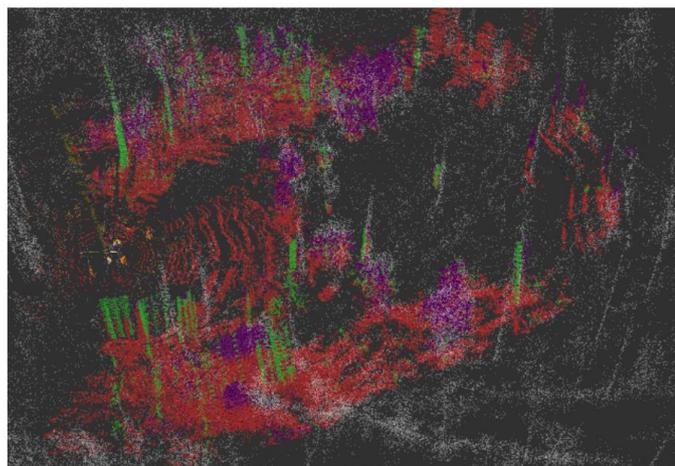


Figura 2.7: Resultados do SLAM semântico [3].

Le et al. [18], obtiveram o tempo de processamento de cada imagem, tempo de inferência, para cada arquitetura CNN, sendo a ERFNet a detentora do menor tempo, enquanto a Darknet53 apresentou ser a mais lenta. O tempo de inferência determina a frequência de imagens que é possível o sistema processar por segundo, em Hz ou FPS (*Frames Per Second*), sendo que numa aplicação em que é necessária uma segmentação semântica em tempo real, são preferíveis taxas de quadros mais elevadas de forma a acompanhar o movimento do veículo. Acerca da precisão e IoU, todas as arquiteturas CNN testadas pelos autores obtiveram valores muito semelhantes e altos. Estes dados estão representados na Tabela 2.3.

Tabela 2.3: Resultados do treino [18].

Architecture	mAcc	mIoU	mIoU da classe 1	Tempo de inferência (ms)
ResNet18	0.986	0.941	0.899	~ 54
ResNet50	0.987	0.946	0.906	~ 142
ResNet152	0.985	0.939	0.895	~ 190
Darknet 21	0.985	0.938	0.892	~ 118
Darknet 53	0.987	0.947	0.908	~ 206
ERFNet	0.986	0.941	0.898	~ 48
MobileNet V2	0.984	0.935	0.888	~ 55

Paulo A. S. Mendes et al. [22] concluíram que o modelo YOLOv5 é o mais vantajoso em comparação com o DeepLabv3+, pois conseguiu identificar a vegetação seca ou morta, aspeto onde o DeepLabv3+ falhou. Devido à diferença de classificação de imagens dos dois modelos e o facto de que o DeepLabv3+ estava pré treinado, não é possível recorrer às métricas convencionais para os comparar, mas o YOLOv5 apresentou uma confiança máxima na deteção de “live vegetation” da ordem dos 92-95%, 89-92% para “grass”, 88% para “tree-trunk” e 40% para “cut-vegetation”. Quanto às métricas de avaliação durante o treino, o valor máximo de mAP50 foi cerca de 65%, para a mAP50:95 correspondeu a cerca de 28%, a Recall e a Precision obtiveram valores de cerca de 60% e 70%, respetivamente. O YOLOv5 apresenta também uma grande vantagem na rapidez de treinamento do algoritmo e no tamanho do ficheiro dos *weights*, assim como nos tempos de processamento de imagem, característica vital na deteção e classificação em tempo real. Os tempos de processamento foram obtidos com um CPU Intel i7 6500U e representados na Tabela 2.4.

Tabela 2.4: Tempo de processamento dos modelos DeepLabv3+ e YOLOv5 em imagens 1440 x 1080 [22].

Modelo	Imagem 1	Imagem 2	Imagem 3	Imagem 4
DLV3+ (s)	4.860	4.720	4.730	4.710
YOLOv5 (s)	0.215	0.230	0.225	0.251

Brandenburg et al. [30], na tentativa de identificar morangos num ambiente de cultivo, testaram o modelo YOLOv3, previamente treinado pelos mesmos, no *dataset* de validação composto por 146 imagens. Definiram o limite inferior de confiança do modelo para 70% de forma a reduzir falsos positivos, o que permitiu um resultado de 78,3% de Recall neste teste. No teste seguinte recorreram a um vídeo do Youtube com um total de 7439 quadros, dos quais em 3371 o modelo identificou morangos, e destes, em 2346 a confiança de detecção era baixa, entre 30 e 60%. Após inspeção das imagens verificaram que apesar dos baixos valores de confiança, o modelo retornou poucos FP. Porém, não conseguiu capturar uma grande parte de todos os morangos existentes numa dada imagem, assim como teve dificuldade em distinguir morangos em proximidade, para além de outros problemas como morangos não estarem completamente envolvidos pela caixa de detecção. Os autores atribuíram estes problemas de desempenho ao tamanho e qualidade do *dataset*, que são características vitais no treino de algoritmos CNN. Na avaliação do desempenho computacional, o YOLOv3 executou a 13,05 FPS na plataforma Xilinx ZCU104, a 2,66 FPS num Raspberry Pi 3B e a 30 FPS num computador de trabalho equipado com uma NVIDIA Quadro P2200.

da Silva et al. [31], no estudo para a detecção de troncos de árvores, determinaram que o modelo com valores mais elevados nas métricas AP e F1 foi o YOLOv4 Tiny, com cerca de 90% em ambas, enquanto o MobileDet foi o que apresentou menor desempenho nesta área. Por outro lado, em relação à variação decrescente de AP e F1 com o aumento de confiança da detecção, foram os modelos Inception-v2 e MobileNetV2 que apresentaram os melhores resultados, ou seja, menor variação. Pelo contrário, a arquitetura ResNet50 foi a que apresentou a maior variação, decrescente, de AP e F1, com o aumento da confiança de detecção. Quanto aos tempos de inferência verificaram que foram os modelos MobileNetV2 e YOLOv4 Tiny os detentores do menor tempo, ao executar na CPU e GPU, respetivamente. A arquitetura ResNet50, em contrapartida, apresentou os maiores tempos de inferência em ambas as plataformas de processamento, em especial na CPU, onde a execução de cada imagem demorou cerca de 1,8 s, tornando inviável o uso em tempo real. Na Tabela 2.5 apresentam-se os valores de Average Precision (AP) e F1-score obtidos da experiência número 3 feita pelos autores onde os modelos foram testados no inteiro arquivo de avaliação contendo imagens térmicas e no espectro visível. Na Tabela 2.6 é apresentado o desempenho computacional dos diferentes modelos quando executados no CPU e GPU. Observando

ambas tabelas é possível constatar rapidamente que o YOLOv4 Tiny é o modelo mais vantajoso, tal como os autores concluíram no final, principalmente para o caso em que os valores de confiança possam ser desprezados.

Tabela 2.5: Resultados de AP e F1 da experiência #3 considerando todas as deteções (confiança = 0%) [31].

Modelo	AP (%)	F1 (%)
SSD MobileNetV2	72.68	80.74
SSD Inception-v2	75.29	83.98
SSD ResNet50	78.19	84.75
SSDLite MobileDet	68.08	73.53
YOLOv4 Tiny	89.84	89.37

Tabela 2.6: Tempos de inferência médios dos detetores em diferente *hardware* [31].

Modelo	CPU (ms)	GPU (ms)
SSD MobileNetV2	58	15
SSD Inception-v2	118	19
SSD ResNet50	1789	50
SSDLite MobileDet	85	17
YOLOv4 Tiny	95	9

Em [33], da Silva et. al., compararam os 13 modelos após treino no *dataset* original e no *dataset* aumentado recorrendo à métrica F1-score, onde concluíram que os modelos não beneficiaram do *dataset* aumentado e as diferenças de F1-score deste para o *dataset* original são desprezáveis. O YOLOv7 Tiny foi o detentor do melhor valor desta métrica, na deteção de troncos em imagens 640 x 640 pixéis, com 90,62 % no *dataset* original e 90,35% no *dataset* aumentado, valor obtido também pelo YOLOR-CSP. Na comparação da evolução do F1-score com o valor de confiança, representada na Figura 2.8, foram novamente os modelos YOLOv7 Tiny e YOLOR-CSP que lideraram com linhas bastante semelhantes, seguidos pelo YOLOv5 Small.

Grondin, et al. [32], no seu estudo para a deteção de árvores para abate, treinaram os modelos no *dataset* SynthTree43k e afinaram (*finetuning*) os mesmos no *dataset* real, CanaTree100. Testando depois neste último, os valores obtidos referentes à deteção com caixas envolventes (*bounding box*) variaram entre 86,3 e 90,4% para AP50, 59,6 a 64,1% para AP50:95, 90,7 a 94,9% para AR50 (Average Recall) e por fim, 66,9 a 70,4% para AR50:95. A respeito da máscara de segmentação aplicada, os valores oscilaram entre 82,8 e 87,2% para AP50, 54,2 a 60,0% para AP50:95, 87,5 a 91,5% para AR50 e terminando, 60,2 a 65,2% para AR50:95. Na estimativa do diâmetro, posição do corte de abate e inclinação da árvore, os erros variaram entre 2,8 e 3,8cm, 6,1 a 7,0cm, e entre 1,1 e 2,5° respetivamente. Os autores atribuem o erro descrito na estimativa da posição do corte à dificuldade de estimar a distância ao solo, pois este erro distribui-se maioritariamente na vertical. Nos testes de transferência de domínio, onde primeiro o modelo foi treinado no *dataset* de florestas de Portugal [31] e testado no CanaTree100 sem afinação, a Average Recall para as caixas envolventes tomaram os valores máximos de 73,6% para AR50 e 42,3% para AR50:95. Quanto às máscaras de segmentação, os valores máximos foram de 82% para AR50 e 49,9% para AR50:95. No teste simétrico em que o modelo treinado no *dataset* CanaTree100 foi avaliado no *dataset* Portugal, obtiveram 49,1% para a AR50 e 20,0% para a AR50:95, relativamente às caixas envolventes. Os valores correspondentes às máscaras de segmentação foram 69,2% para AR50 e 34% para AR50:95. Concluíram que a transferência de domínio reduz significativamente o desempenho dos modelos, neste caso em especial quando foi treinado com imagens de florestas do Canadá e avaliado em imagens de florestas portuguesas.

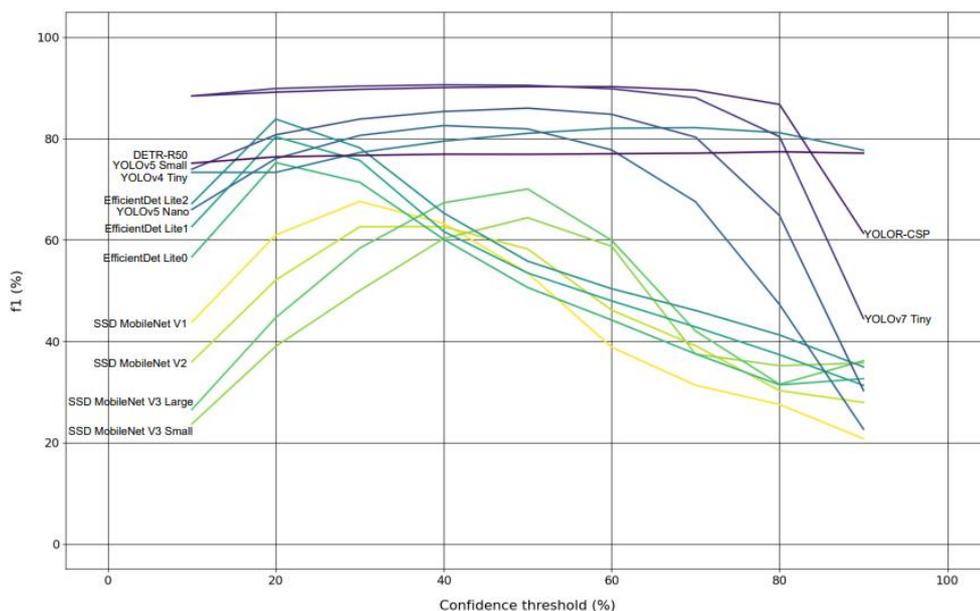


Figura 2.8: Evolução do F1-score para diferentes valores de confiança [33].

Na avaliação dos tempos de inferência em cada plataforma de processamento, o YOLOv5 Nano obteve o menor tempo na NVIDIA Jetson Nano com 20,21ms para uma imagem 320 x 320 pixels. Nesta plataforma o pior desempenho pertence ao YOLOR-CSP com 616,09ms para uma imagem 640 x 640. No Raspberry Pi 4 foi o SSD MobileNetV3 Small o detentor do menor tempo com 22,25ms para uma imagem 320 x 320. Novamente o resultado mais desfavorável pertence ao YOLOR-CSP com 8652,50ms para uma imagem 640 x 640, sendo este o pior valor de toda a experiência. No Google Coral Accelerator o menor tempo de inferência foi obtido pelo SSD MobileNetV1 com 7,30ms para uma imagem 300 x 300, e o pior modelo foi o SSD MobileNet V3 Large com 104,03ms para uma imagem 320 x 320. Foi testada também uma GPU de consumidor não destinada à implementação móvel, a NVIDIA RTX 3090, onde o melhor desempenho pertence ao YOLOv4 Tiny com apenas 1,93ms de tempo de inferência numa imagem 416 x 416, e o modelo mais lento foi EfficientDet Lite2 com 18,76ms numa imagem 448 x 448.

Andrada et al. [34], levaram a máquina florestal para uma zona onde puderam testar a capacidade de deteção e localização do combustível florestal a ser removido, através da gravação de informação proveniente de diferentes sensores ao longo de 2 horas de operação. Extraíram e anotaram 500 imagens dessa gravação onde depois avaliaram o método de deteção. Para a identificação de combustível florestal obtiveram um F1-score de 70,6% e um IoU de 54,6%, enquanto nas restantes classes as métricas tomaram os valores de 49,3% e

41,9% para F1-score e IoU médio, respetivamente. Os autores referem que estes valores não foram mais elevados pois o ambiente escolhido para o teste contém poucos elementos característicos de uma floresta, tal como reduzida diversidade dentro das classes detetadas. Além disso, existia pouca luminosidade à hora do dia a que foram recolhidas as imagens, característica à qual as câmaras são bastante sensíveis, com exceção das camaras térmicas. A localização do combustível foi avaliada de forma apenas qualitativa devido à falta de valores *ground truth* desse tipo. Pela inspeção visual dos resultados concluíram que os centroides das manchas de combustível florestal foram criados corretamente, as suas coordenadas reais corresponderam às coordenadas de pixel em x e y. A posição segundo z, dependente de um mapa de profundidade, mostrou erros que podem atingir os 3m de distância euclidiana.

3. MÉTODOS PARA A DETEÇÃO DE OBSTÁCULOS

3.1. Hardware

Os equipamentos disponíveis e destinados a serem equipados na máquina, os quais são utilizados para a aquisição de dados, consistem numa câmara de profundidade Intel RealSense D435i [46], uma câmara térmica FLIR ADK [47], e um LiDAR 3D Velodyne VLP-16 [48]. Durante a operação autónoma os programas serão potenciados por uma NVIDIA Jetson Xavier NX equipada na máquina juntamente com os sensores, permitindo um processamento com baixo consumo de energia, uma mais-valia quando todos estes componentes são alimentados por uma bateria.

Para o treino, implementação e teste de algoritmos CNN e outros programas fez-se uso de um computador local equipado com um processador de seis núcleos AMD Ryzen 3600, 16GB de RAM DDR4 e uma GPU NVIDIA GTX 1660 SUPER com 6GB de memória de vídeo. Para o treino do algoritmo CNN nos *datasets*, recorreu-se também ao *notebook* online do Google Colab, que disponibiliza gratuitamente, com tempo limitado, uma GPU profissional, onde em todas as sessões na realização deste trabalho correspondeu a uma NVIDIA T4 com 16GB de memória de vídeo disponível.

O computador local possui instalado o Ubuntu 20.04.5 LTS (Focal Fossa) como sistema operativo, Python 3.8.10, Pytorch 1.8.1, NVIDIA Driver 470.161.03, CUDA Toolkit 10.1 e o ROS 1 Noetic.

O *notebook* do Google Colab, é baseado no Ubuntu 20.04.5 LTS, apresentando as versões Python 3.8.10, Pytorch 1.13.1, NVIDIA Driver 510.47.03 e CUDA Toolkit 11.6 (Verificado a 05/02/2023).

Para a recolha de dados com os sensores foi ainda usado um computador portátil a utilizar o Ubuntu 18.04.6 LTS como sistema operativo e com o ROS 1 Noetic instalado.

3.2. Recolha de dados

De forma a possibilitar o teste dos métodos propostos, foram recolhidos alguns dados com recurso aos três sensores descritos acima no subcapítulo 3.1. O Velodyne, a câmara RealSense e a câmara FLIR foram montados num suporte, impresso em 3D na oficina do

Field Tech Laboratory., no edifício do LAI-UC. A configuração da montagem, representada na Figura 3.1, foi escolhida na tentativa de fazer coincidir a origem dos campos de visão dos sensores, para obter os dados na mesma posição, ao mesmo tempo permitindo que o suporte criado fosse acoplado a um tripé, de modo a dar aos dispositivos uma elevação ao solo aproximada à qual estariam se montados na máquina autónoma.



Figura 3.1: Configuração dos sensores para a recolha de dados

Os dados foram recolhidos numa zona florestal a cerca de 300m a sudeste do edifício do LAI e consistiram na gravação do output produzido pelos sensores durante 120 segundos, em dois momentos. No primeiro o tripé foi posicionado a 5,5m de um grupo de troncos de eucalipto, sendo a distância correspondente ao tronco mais próximo, com o objetivo de registar o perfil destes e também da zona em redor, através da nuvem de pontos gerada pelo Velodyne assim como a imagem de profundidade gerada pela câmara RealSense. A nuvem de pontos do Lidar foi gravada recorrendo ao ROS, através do comando “rosvbag record --duration=120 /velodyne_points”, onde /velodyne_points é o nome do tópico publicado no ROS enquanto o Velodyne está em funcionamento e a fornecer dados. A esta distância, 5,5m, a câmara RealSense não conseguiu gerar uma imagem de profundidade, pelo que foi necessário aproximar o tripé até 2,5m do tronco mais próximo. Voltou a ser gravado um ficheiro com a nuvem de pontos do Lidar e de seguida o output da RealSense, também através do ROS com um comando semelhante ao anterior, diferente apenas no nome do tópico, /camera/depth/color/points. Foram ainda gravados dois vídeos, um a cores, com a câmara visível da RealSense e um de imagem térmica com a câmara FLIR, cuja finalidade

é testar a capacidade do modelo CNN para a deteção de pessoas em ambiente florestal e parcialmente ocultas por folhagem. A Figura 3.2 demonstra a posição dos sensores e ilustra o local durante a recolha dos dados. No Anexo A a Figura A.1 representa uma combinação de quatro imagens, com a visualização no RVIZ da gravação obtida com o Velodyne, e com a câmara RealSense, para além de uma imagem a cores retirada do vídeo gravado com a RealSense, e uma imagem retirada do vídeo térmico capturado pela câmara FLIR. As quatro imagens apresentam uma direção do campo de visão semelhante. O RVIZ é uma ferramenta de visualização integrada no ROS.



Figura 3.2: Posição dos sensores e cenário da recolha de dados.

Para além desta recolha de dados, foi também gravado um ficheiro *rosbag* da nuvem de pontos do Velodyne, recolhido no interior do edifício do LAI, com o propósito de criar o perfil de obstáculos para posteriormente testar a sua deteção baseada no LiDAR. O local e cenário da gravação é representado na Figura A.2 do Anexo A.

3.3. Escolha do modelo CNN

Abordando a deteção de objetos em ambiente florestal e em tempo real recorrendo a imagens no espectro visual e infravermelho, pela literatura consultada pode tomar-se como melhor opção os algoritmos YOLO, devido à sua popularidade e consequente disponibilidade de documentação online, relativa facilidade de implementação e elevado desempenho computacional, adequado a dispositivos móveis. O uso de modelos de

segmentação semântica requer um poder computacional muito superior, podendo tornar inviável a sua implementação em plataformas como a NVIDIA Jetson.

Os modelos recentes mais utilizados são o YOLOv5 [24] e o YOLOv7 [43, 44], de referir que o YOLOv8 [45] foi divulgado durante a realização deste trabalho pelos autores do YOLOv5. Em [43, 44] os autores afirmam que o YOLOv7 apresenta maior precisão e desempenho computacional em comparação com o YOLOv5, entre outros modelos, quando testado no *dataset* MS COCO [51], potenciado por uma GPU NVIDIA V100. O *dataset* MS COCO é utilizado por ambos os autores para treinar e disponibilizar os modelos com ficheiros de pesos, *weights*, pré-treinados neste *dataset* que inclui 80 classes de deteção e cerca de 330 000 imagens.

No entanto, ambos os modelos têm diferentes versões de *weights* disponíveis, consoante a quantidade de parâmetros utilizados no algoritmo, que ditam o equilíbrio entre desempenho computacional e a precisão do modelo. Torna-se necessário efetuar um teste para determinar o mais adequado para operação em tempo real e possível futura implementação numa NVIDIA Jetson. Para isto, os *weights* selecionados foram o YOLOv5s (small), correspondendo à segunda versão com menor número de parâmetros, e o YOLOv7-tiny, apresentando o menor número de parâmetros deste último modelo. De seguida os modelos foram executados no MS COCO128, que consiste nas primeiras 128 imagens do *dataset* de treino correspondente ao MS COCO completo de 2017, e foram anotados os tempos de computação e as métricas de medição de desempenho correspondentes. Nesta comparação, cujos resultados se encontram no capítulo 4.1, foi o YOLOv5-small o modelo mais vantajoso.

3.4. Descrição dos *datasets*

As categorias abordadas para deteção consistem em pessoas, animais, troncos de árvores, planos/cursos de água e combustível florestal. Nesta fase pretende-se identificar estas classes recorrendo ao modelo CNN, para isso, foram escolhidos *datasets* existentes e adequados a serem utilizados para o treino. Na Tabela 3.1 encontra-se um resumo do número de imagens e sua repartição, para cada um dos *datasets* selecionados, apresentados à frente.

Tabela 3.1: Número de imagens de cada *dataset* e sua respetiva repartição para treino, validação e teste.

	# de imagens	train/val/test
Trees_dataset	2895	2026/579/240
TreeDetection_dataset	2478	2110/241/127
Woods_dataset	328	303/25/0
Trees_augmented	5944	5614/220/110
Vegetation_detection	199	174/17/8
Water_detection	480	338/95/47

3.4.1. Deteção de troncos

Com o objetivo de identificar troncos de árvores foram utilizados quatro *datasets* designados por “Trees_dataset”, “TreeDetection_dataset” [65], “Woods_dataset” [66] e “Trees_augmented” [67]. Os três últimos pertencem à lista de *datasets* do universo Roboflow [55] de livre utilização.

O “Trees_dataset” consiste no *dataset* original não aumentado criado por da Silva et al. [31], composto por um total de 2895 imagens repartidas por 2029 a cores e 866 térmicas, de três separadas florestas em Portugal que contêm maioritariamente eucaliptos. Este *dataset* está disponibilizado com as anotações em formato “XML”, não compatíveis com o YOLO, que utiliza ficheiros “.txt”. A conversão para este formato foi feita com recurso ao website Roboflow, que permite o upload do *dataset* e suas anotações originais, organizando-as automaticamente com as imagens e disponibilizando depois o download do *dataset* com as anotações no formato correto para o modelo de CNN que se pretende. O website possui também a opção de dividir o *dataset* em categorias de treino, validação e teste, onde para o “Trees_dataset” foi escolhida a divisão em 70%, 20% e 10%, respetivamente, correspondendo a 2026, 579 e 290 imagens. Na Figura 3.3 representa-se como exemplo uma das imagens anotadas utilizadas no treino.



Figura 3.3: Exemplo de imagem do “Trees_dataset” utilizada no treino.

O “TreeDetection_dataset” é composto por 2478 imagens e anotações de troncos, repartidas em 85% para treino, 10% para validação e 5% para teste, correspondendo a 2110, 241 e 127 imagens. Este *dataset* possui apenas imagens a cores do espectro visível e tem anotados troncos caídos no solo da floresta, como se observa na Figura 3.4 que é exemplo de uma das imagens anotadas utilizadas no treino.



Figura 3.4: Exemplo de imagem do “TreeDetection_dataset” utilizada no treino.

O “Woods_dataset” é composto por 328 imagens anotadas de troncos, repartidas apenas em dois arquivos para treino e validação com 92% e 8% respetivamente, correspondendo a 303 e 25 imagens. A ausência de repartição de teste criada pelo autor

assume-se devido ao reduzido tamanho do *dataset*, na tentativa de maximizar as imagens utilizadas para treino. Todas as imagens estão no espectro visível e na Figura 3.5 encontra-se um exemplo utilizado no treino.



Figura 3.5: Exemplo de imagem do “Woods_dataset” utilizada no treino.

O “Trees_augmented” é composto por 5944 imagens e anotações de troncos, repartidas em 94% para treino, 4% para validação e 2% para teste, correspondendo a 5614, 220 e 110 imagens. Este *dataset* também possui apenas imagens no espectro visível e à semelhança do “TreesDetection_dataset”, tem anotados troncos caídos no solo da floresta. Além disso as imagens sofreram uma técnica de aumento, por cisalhamento da imagem, como se observa na Figura 3.6 que é exemplo de uma das imagens anotadas utilizadas no treino.



Figura 3.6: Exemplo de imagem do *dataset* “Trees_augmented” utilizada no treino.

3.4.2. Deteção de seres vivos

Os autores dos modelos YOLO disponibilizam os ficheiros dos *weights* já treinados no *dataset* MS COCO, facilitando uma aplicação direta do modelo com resultados de deteção bastante robustos para pessoas e alguns animais. Isto deve-se a que dentro das 80 classes do MS COCO estão incluídas “person”, “cat”, “dog”, “horse”, “sheep”, “cow” e outros 5 animais não relevantes para este trabalho. Desta forma considera-se desnecessário o treino adicional em imagens do espectro visual para a deteção destas classes. Para a capacidade de detetar pessoas e animais com recurso à câmara térmica, os *weights* pré treinados também o permitem, sendo que da mesma forma, é desnecessário o treino adicional com *datasets* térmicos. Os *weights* pré treinados do YOLOv5-small foram aplicados nos vídeos a cores e térmico, obtidos na colheita de dados.

3.4.3. Deteção de combustível florestal

A deteção fiável do combustível florestal a ser removido pela máquina é uma tarefa complicada devido ao ambiente destruturado e complexo em que deve ser implementada. Além disso, os arbustos destinados à remoção podem esconder obstáculos como rochas, desníveis ou até troncos o que pode levar a danos na máquina. Uma abordagem mais robusta seria a utilização de um modelo CNN que realize segmentação semântica dos pixels da imagem obtida pelas câmaras, como nos estudos de Andrada et al. [2] e Russel et al. [3], por exemplo. Ainda assim foi selecionado um *dataset*, designado “Vegetation_detection” [69], para o treino do YOLOv5, que consiste naquele criado por Mendes et al. [22], com imagens recolhidas perto do Polo II da UC. À semelhança dos *datasets* descritos anteriormente, foi ele também retirado do Roboflow, com as apenas 199 imagens mantidas na repartição original imposta pelos autores, de 87% para treino, 9% para validação e 4% para teste, correspondendo a 174, 17 e 8 imagens. As imagens estão todas no espectro visível, anotadas com cinco classes diferentes de vegetação e na Figura 3.7 apresenta-se um exemplo de uma delas, utilizada no treino. Pela sua observação torna-se evidente que os autores implementaram várias técnicas de implementação, como introdução de ruído e alterações à exposição, recorrendo à ferramenta do Roboflow para este efeito.

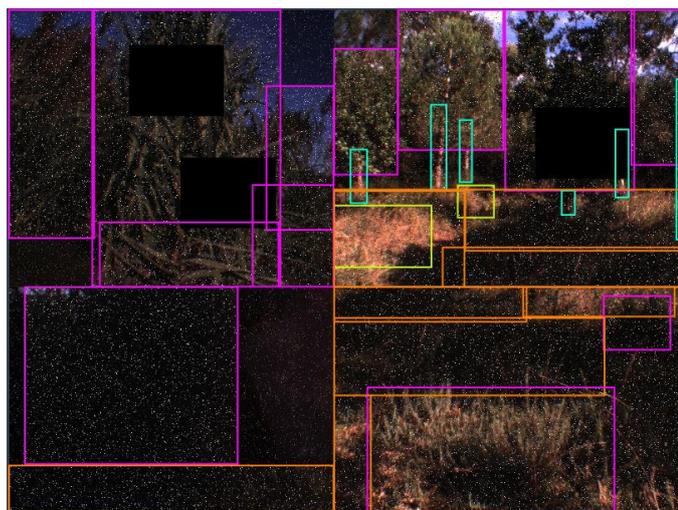


Figura 3.7: Exemplo de imagem do *dataset* “Vegetation_detection” utilizada no treino.

3.4.4. Deteção de cursos e planos de água

Na tentativa de treinar o modelo para a deteção de cursos e planos de água que possam impedir a passagem da máquina, foi escolhido um *dataset* do universo Roboflow, designado “H3K Dataset” [68], que consiste em 1406 imagens a cores de diversos cenários ambientais. Como este *dataset* possui várias classes de deteção e imagens sem planos de água, recorrendo à ferramenta de clonagem do Roboflow foram seleccionadas apenas as imagens que contêm a classe “Running_water”, juntamente com 12 imagens de floresta do mesmo *dataset*, mas sem esta classe, totalizando 480 imagens organizadas num novo *dataset* separado. As 12 imagens foram escolhidas manualmente para que possam servir de *background*, que são imagens sem anotações com a finalidade de controlar os FP do modelo de deteção. Este novo *dataset* foi designado “Water_detection” e repartido em 70% para treino, 20% para validação e 10% para teste, correspondendo a 338, 95 e 47 imagens. Na Figura 3.8 apresenta-se um exemplo de uma imagem deste *dataset* utilizada no treino.

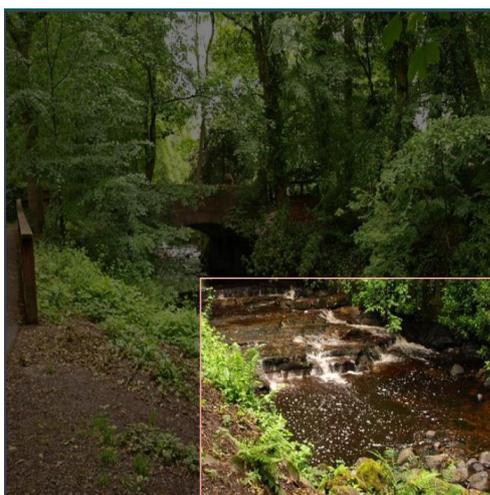


Figura 3.8: Exemplo de imagem do *dataset* “Water_detection” utilizada no treino.

3.5. Descrição do treino

O modelo YOLOv5 foi treinado recorrendo ao *notebook* do Google Colab e também a um computador local, o que permite uma pequena comparação entre os resultados obtidos para as duas plataformas, com o treino dos mesmos *datasets* em ambas, à exceção do “Trees_dataset” e “Trees_augmented”. Da mesma forma, foi também avaliada a influência da afinação do modelo (*finetuning*) em alguns dos *datasets*.

O modelo foi treinado utilizando os parâmetros com os valores indicados nas tabelas de cada *dataset*, representadas nos próximos subcapítulos, que foram introduzidos no comando de treino dado ao YOLOv5, como demonstrado no exemplo da Figura 3.9.

```
train.py --img 640 --batch 32 --epochs 250 --patience 50 --data
../Trees_dataset/data.yaml --weights yolov5s.pt --cache --name Trees_dataset_Colab
```

Figura 3.9: Exemplo de comando de treino dado ao YOLOv5 no *notebook* do Google Colab

No exemplo do comando acima, *batch* indica o número de imagens de treino que o modelo utiliza por iteração, este valor depende da GPU utilizada, *epoch* indica o número de vezes que o modelo percorre o *dataset* durante o treino, *patience* define o número de *epochs* ao fim dos quais o modelo interrompe o treino automaticamente caso não se registre melhorias, *data* indica o diretório do ficheiro “*.yaml” que contém por sua vez os diretórios das pastas com as imagens e anotações do *dataset*, *weights* define o ficheiro de pesos sobre o qual vai ser treinado, *cache* permite guardar as imagens de treino e validação na memória

RAM do computador para um treino ligeiramente mais rápido e, por fim, *name* é usado apenas para definir o nome do treino.

De forma a avaliar se a afinação do modelo nos *datasets* treinados melhoraria as métricas de avaliação, os ficheiros dos *weights* de alguns *datasets*, treinados com os hiper parâmetros padrão do YOLOv5 (ficheiro *hyp.scratch-low.yaml*) voltaram a ser treinados no mesmo *dataset*, mas desta vez com os hiper parâmetros adequados à afinação (ficheiro *hyp.VOC.yaml*) que proporcionam uma taxa de aprendizagem inicial muito mais reduzida, e são introduzidos no comando de treino ao escrever “--hyp hyp.VOC.yaml”. Outra abordagem em que foi testada a afinação do modelo, consiste em “congelar” o *backbone* do modelo durante o primeiro treino, que no caso do YOLOv5 small é composto por 10 linhas, representadas no ficheiro “yolov5s.yaml”. Este efeito é obtido pela introdução do parâmetro “--freeze 10” no comando de treino. Após o treino inicial o ficheiro *weights* resultante é treinado novamente utilizando o ficheiro de hiper parâmetros descrito anteriormente.

Os resultados dos treinos e destas comparações apresentam-se no capítulo 4.2. Em todos os treinos iniciais foi utilizado o ficheiro *weights* “yolov5s.pt” pré treinado no MS COCO, disponibilizado pelos autores do YOLOv5. Vários parâmetros durante os treinos foram registados automaticamente recorrendo ao Weights and Biases [10]. Nos subcapítulos seguintes apresenta-se a preparação dos *datasets* e a descrição do seu uso nos treinos.

3.5.1. Treino deteção de troncos

Na preparação dos *datasets* para o treino, de forma a permitir posteriormente a utilização de múltiplos ficheiros *weights* na deteção, estes necessitam de ser treinados com o mesmo número de classes. Para os *datasets* destinados à deteção de troncos a única classe utilizada em todos eles foi designada “trunk”. Dois destes *datasets*, “Trees_dataset” e “Trees_augmented”, têm originalmente 2 e 3 classes respetivamente, que estão presentes nos ficheiros “.txt” das anotações, quando adquiridos do Roboflow. Porém, todas estas classes representam troncos, levando a assumir que o seu propósito esteja ligado ao estudo particular dos seus autores, não relevante para este trabalho. Desta forma é necessário alterar o texto no interior dos milhares de ficheiros que compõem as anotações, onde cada um toma a disposição exemplificada na Figura 3.10.

```
0 0.7472222222222222 0.24166666666666667 0.24583333333333332 0.48333333333333334
2 0.95138888888888888 0.32777777777777778 0.09722222222222222 0.65555555555555556
1 0.29826388888888889 0.33425925925925926 0.07291666666666667 0.12962962962962962
0 0.02048611111111111 0.3560185185185185 0.04097222222222222 0.2657407407407407
2 0.28854166666666664 0.42361111111111111 0.02847222222222222 0.07314814814814814
```

Figura 3.10: Exemplo da disposição do texto dentro dos ficheiros das anotações utilizadas pelo YOLO.

No texto da Figura 3.10 as classes correspondem aos primeiros algarismos de cada linha, onde os seguintes indicam as coordenadas de cada caixa anotada que envolve um objeto na imagem. O objetivo é alterar os algarismos das classes para “0”, mantendo os que já possuem esse valor. Para isso foi utilizado um código em Python, sugerido numa discussão da Ultralytics Community [41], dos autores do YOLOv5. Este código foi modificado, pois o original alterava também algarismos das coordenadas das caixas, inviabilizando as anotações. Um exemplo da versão utilizada pode ser encontrado na Figura A.3 do Anexo A. Após isto, os ficheiros “data.yaml” de cada *dataset* puderam ser alterados para incluírem apenas a única classe “trunk”.

Nas tabelas seguintes, dentro das colunas de título Plataforma, “Colab” corresponde ao treino no Google Colab, e “PC” ao treino no computador local. Os nomes dos treinos com “Fine” incluído representam aqueles em que se testou a afinação do modelo com os hiperparâmetros “hyp.VOC.yaml”.

Trees_dataset

Este *dataset* foi utilizado para treinar o YOLOv5 em duas instâncias, primeiro um treino normal designado “Trees_dataset_Colab”, com os parâmetros semelhantes aos representados na Figura 3.9 e de seguida um treino de *finetuning* designado “Trees_dataset_Colab_Fine”, com o ficheiro *weights* obtido do anterior. Ambos foram realizados no Google Colab onde o *batch size* e a *patience* corresponderam a 32 e 50, respetivamente. Os *epochs* tomaram os valores de 250 para o treino normal e 200 para o *finetuning*.

Woods_dataset

Este *dataset* foi utilizado para quatro treinos, o primeiro e o segundo foram treinos normais realizados nas duas plataformas para comparação, o terceiro com *finetuning* usando

o ficheiro dos *weights* obtido no treino normal do Google Colab, e no último foi realizado um *finetuning* nos *weights* obtidos de um treino anterior em que foi impedida a aprendizagem do backbone com o parâmetro “freeze”. Esse treino intermédio não é apresentado pois a análise dos seus resultados é dispensável devido ao pobre desempenho nas métricas, gerado pelo congelamento do *backbone*. O teste com o congelamento foi feito apenas neste *dataset*. Na Tabela 3.2 representam-se os valores dos parâmetros usados.

Tabela 3.2: Parâmetros dos treinos com o Woods_dataset.

Nome do treino	Epochs	Batch size	Patience	Plataforma
Woods_dataset	250	16	50	PC
Woods_dataset_Colab	250	64	50	Colab
Woods_dataset_Colab_Fine	100	64	50	Colab
Woods_dataset_Colab_Freeze_Fine	250	64	50	Colab

TreeDetection_dataset

Este *dataset* à semelhança do anterior foi utilizado para a comparação das plataformas de treino e também para a avaliação do *finetuning*, com os parâmetros utilizados nos 3 treinos representados na Tabela 3.3.

Tabela 3.3: Parâmetros dos treinos com o TreeDetection_dataset.

Nome do treino	Epochs	Batch size	Patience	Plataforma
TreeDetection_dataset	250	16	50	Colab
TreeDetection_dataset_Colab	250	64	50	Colab
TreeDetection_dataset_Colab_Fine	200	64	50	PC

Trees_augmented

Devido ao tamanho deste *dataset*, perto de 6000 imagens, a execução do treino no *notebook* do Google Colab chegava ao limite de tempo disponível e assim foi apenas treinado

no computador local para 150 *epochs*, com um *batch size* de 16 e o valor de *patience* definido em 20.

3.5.2. Treino detecção de combustível florestal

Com este *dataset* foram realizados três treinos, sendo o primeiro uma tentativa de recriar os resultados obtidos por Mendes et al. [22], treinando o YOLOv5 de forma semelhante, apesar de os autores não referirem se utilizaram a versão *small* do modelo. Para os treinos seguintes foi feita uma alteração no *dataset*, semelhante à descrita no subcapítulo anterior, onde o código em Python representado na Figura A.3 foi utilizado para reduzir o número de classes anotadas para apenas uma, renomeando as classes de vegetação como apenas “vegetation” e eliminando as anotações dos troncos de árvores. Com este *dataset* alterado foram realizados mais dois treinos cujos nomes possuem “alt” para os diferenciar, com objetivo de comparar ao original e também entre plataformas. Na Tabela 3.4 apresentam-se os parâmetros para estes três treinos, onde *patience* não tem valor pois a função de paragem automática estava desligada, de modo a completar todos os *epochs*.

Tabela 3.4: Parâmetros dos treinos com o Vegetation_detection.

Nome do treino	Epochs	Batch size	Patience	Plataforma
Vegetation_detection_Colab	1000	64	-	Colab
Vegetation_detection_Colab_alt	1000	64	-	Colab
Vegetation_detection_alt	1000	16	-	PC

3.5.3. Treino detecção de cursos de água

Este *dataset* foi utilizado para dois treinos nas plataformas diferentes, com os respetivos parâmetros apresentados na Tabela 3.5.

Tabela 3.5: Parâmetros dos treinos com o Water_detection.

Nome do treino	Epochs	Batch size	Patience	Plataforma
Water_detection	300	16	100	PC
Water_detection_Colab	400	64	100	Colab

4. RESULTADOS

4.1. Escolha do modelo

Após o teste de ambos os algoritmos no MS COCO128, obtiveram-se os valores das métricas mAP50, mAP50:95, Precision e Recall, assim como o tempo médio de inferência por imagem e a correspondente taxa de quadros (FPS), para cada um dos modelos, utilizando os mesmos parâmetros, com o limite IoU=0,60 e um tamanho de imagem correspondente a 640 x 640 pixels. Os resultados apresentam-se na Tabela 4.1:

Tabela 4.1. Comparação de métricas de avaliação e tempo de inferência para os modelos YOLOv5 e v7 no MS COCO128 (GTX 1660 SUPER).

	mAP50	mAP50:95	Precision	Recall	Tempo médio (ms)	FPS
YOLOv5-small	0,713	0,475	0,709	0,634	6,8	147
YOLOv7-tiny	0,649	0,435	0,632	0,597	16,0	63

Pela observação dos resultados pode concluir-se que no *hardware* utilizado para o teste, o modelo YOLOv5 small foi superior ao YOLOv7 tiny em todos os aspetos, apresentando valores de mAP, Precision e Recall mais elevados enquanto o tempo de processamento de imagem foi significativamente inferior. Desta forma o YOLOv5-small foi selecionado para as restantes tarefas deste trabalho.

4.2. Resultados do Treino

Neste capítulo são apresentados os resultados dos treinos para todos os *datasets*, onde as métricas de avaliação correspondem às obtidas para a execução nas imagens de validação dos ficheiros *weights* correspondentes ao melhor *epoch*. Nas tabelas apresentadas neste subcapítulo o tempo médio de inferência por imagem é representado por “t”.

4.2.1. Resultados *datasets* de troncos

Trees_dataset

Na Tabela 4.2 apresentam-se os valores obtidos para as métricas de avaliação do Trees_dataset, que permitem concluir que o treino de *finetuning* realizado não obteve resultados consideráveis sobre o desempenho, sendo os valores próximos para os dois treinos. Por outro lado, estes resultados são bastante satisfatórios, com a mAP50 a ultrapassar os 90%. O *epoch* 230 foi o melhor para o treino normal, enquanto o mesmo aconteceu no sexto *epoch* para o treino de *finetuning*, que parou automaticamente nos 55 *epochs*. Na Figura 4.1 está representada a evolução da mAP50 durante os treinos.

Tabela 4.2: Valores das métricas de avaliação dos treinos com o Trees_dataset.

Nome do treino	mAP50	mAP50:95	Precision	Recall
Trees_dataset_Colab	0,923	0,611	0,901	0,893
Trees_dataset_Colab_Fine	0,927	0,609	0,892	0,898

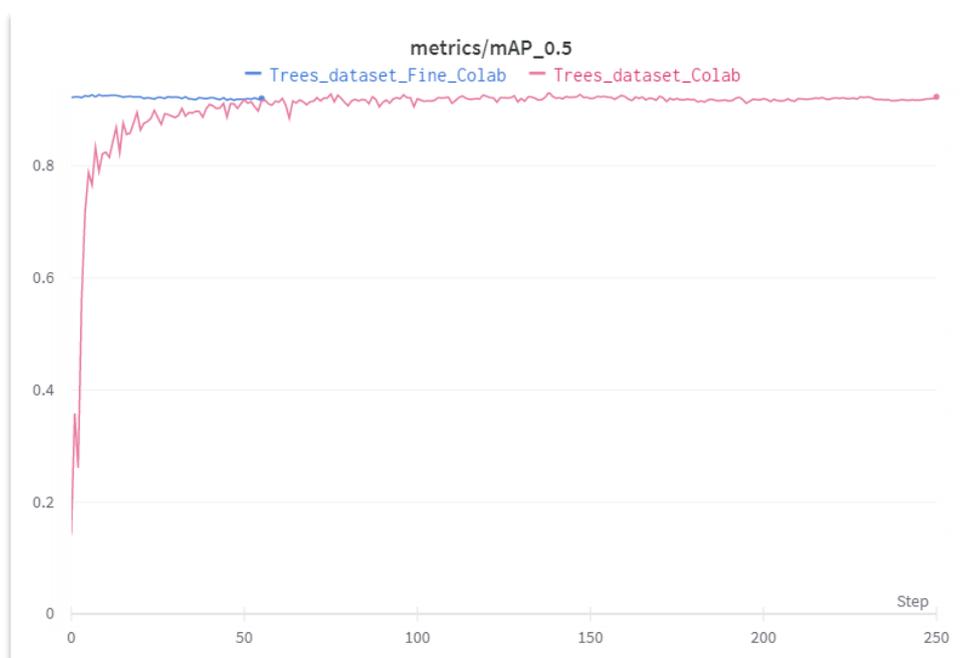


Figura 4.1: Evolução da mAP50 ao longo dos treinos com o Trees_dataset.

Woods_dataset

Os resultados obtidos para os valores das métricas dos quatro treinos com este *dataset* apresentam-se na Tabela 4.3, onde se pode reparar que o treino no computador local obteve um valor 5% superior na mAP50 e 8,3% superior na Precision, apesar de o valor de Recall ser 2,7% inferior. Neste *dataset* o uso de *finetuning* foi benéfico para os resultados, onde o congelamento do *backbone* melhorou a mAP50 e a Recall. Na Figura 4.2 está representada a evolução da mAP50 durante os quatro treinos, onde é possível observar a vantagem dos treinos de afinação.

Tabela 4.3: Valores das métricas de avaliação dos treinos com Woods_dataset.

Nome do treino	mAP50	mAP50:95	Precision	Recall
Woods_dataset	0,948	0,662	0,925	0,963
Woods_dataset_Colab	0,898	0,660	0,842	0,990
Woods_dataset_Colab_Fine	0,971	0,691	0,960	0,885
Woods_dataset_Colab_Freeze_Fine	0,990	0,685	0,931	0,994

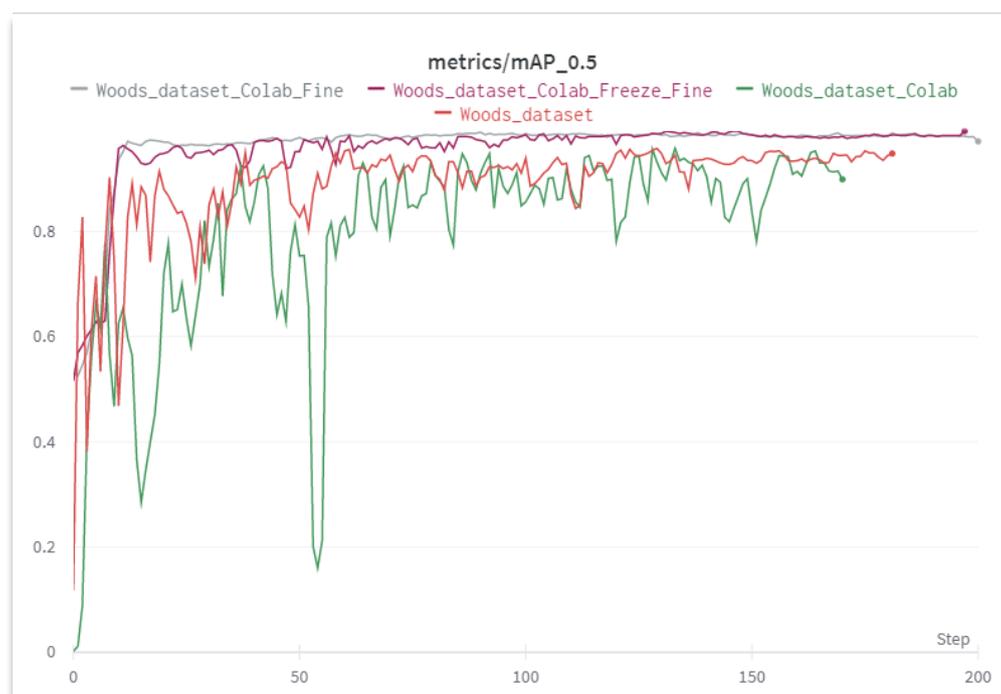


Figura 4.2: Evolução da mAP50 ao longo dos treinos com o Woods_dataset.

TreeDetection_dataset

Nos treinos com este dataset, cujos valores da métricas se encontram representados na Tabela 4.4: Valores das métricas de avaliação dos treinos com o TreeDetection_dataset., observa-se que os resultados são muito semelhantes entre os três, o que significa que a plataforma onde foram treinados e a realização de finetuning não tem influência. Ainda assim os valores obtidos são razoáveis, apesar de mais baixos que os *datasets* anteriores. A Figura 4.3 representa a evolução da mAP50 ao longo dos treinos.

Tabela 4.4: Valores das métricas de avaliação dos treinos com o TreeDetection_dataset.

Nome do treino	mAP50	mAP50:95	Precision	Recall
TreeDetection_dataset	0,868	0,552	0,801	0,783
TreeDetection_dataset_Colab	0,865	0,543	0,802	0,773
TreeDetection_dataset_Colab_Fine	0,866	0,559	0,802	0,789

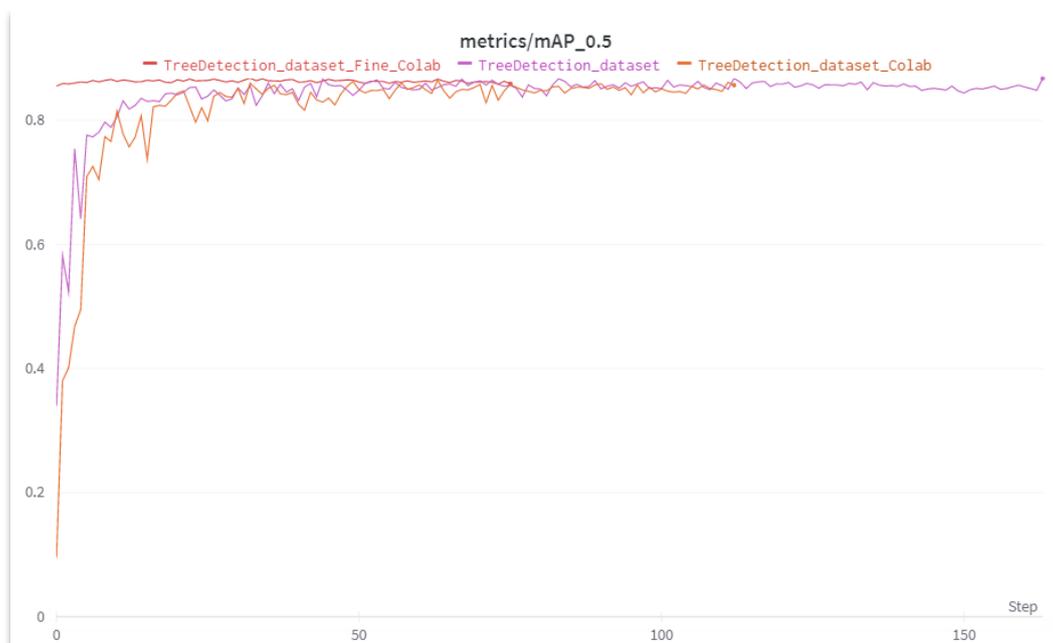


Figura 4.3: Evolução da mAP50 ao longo dos treinos com o TreeDetection_dataset.

Trees_augmented

O treino do YOLOv5 com este *dataset* resultou nos valores para as métricas de avaliação representados na Tabela 4.5: Valores das métricas de avaliação do treino com o *Trees_augmented*. A evolução da mAP50 ao longo deste treino está representada na Figura 4.4. O melhor ficheiro *weights* ocorreu aos 82 *epochs*, e a interrupção automática do treino após 103 *epochs* e mais de 37 horas de execução. Obtiveram-se bons valores com a mAP50 e a Precision perto dos 90%.

Tabela 4.5: Valores das métricas de avaliação do treino com o *Trees_augmented*.

Nome do treino	mAP50	mAP50:95	Precision	Recall
Trees_augmented	0,881	0,515	0,894	0,805

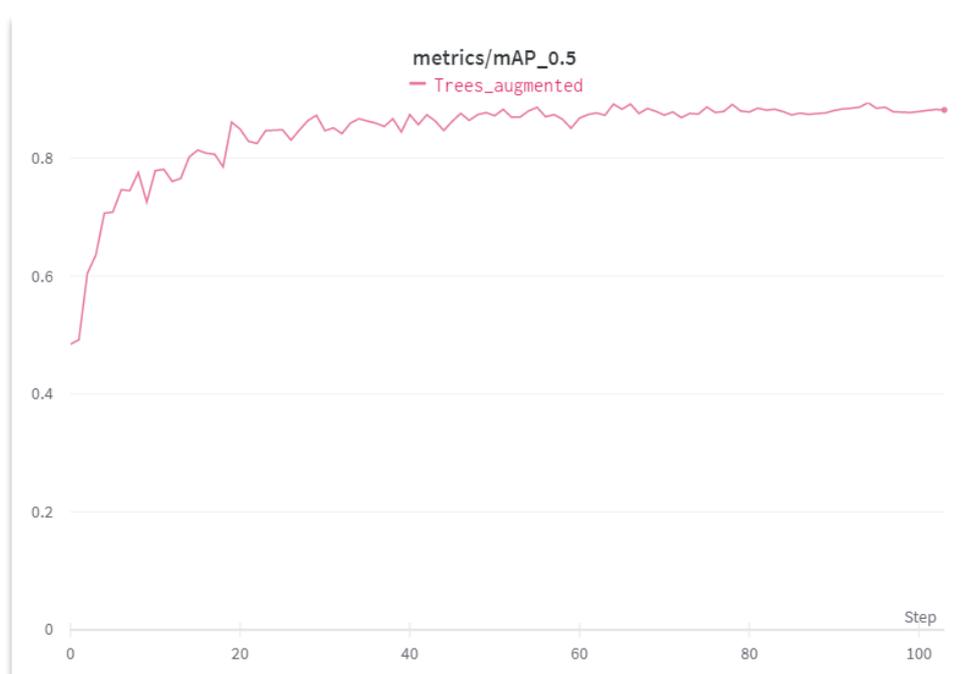


Figura 4.4: Evolução da mAP50 ao longo do treino com o *Trees_augmented*.

4.2.2. Resultados *dataset* Vegetation_detection

Representam-se na Tabela 4.6 os valores das métricas para os três treinos com este *dataset*, onde os valores para o treino com o *dataset* original ficaram bastante próximos dos obtidos por Mendes et al. [22]. Após a alteração do *dataset* para apenas uma classe de

vegetação é evidente que os resultados foram superiores, à exceção da Precision, principalmente para o treino realizado no Google Colab que aqui levou uma vantagem de 6,1% na mAP50, 4,7% na mAP50:95, 3,8% na Precision, apesar de ser o mesmo valor do *dataset* original, e por fim, 1,4% na Recall, em relação ao treino no computador local. Na Figura 4.5 pode-se observar a evolução da mAP50 ao longo dos treinos.

Tabela 4.6: Valores das métricas de avaliação dos treinos com o Vegetation_detection.

Nome do treino	mAP50	mAP50:95	Precision	Recall
Vegetation_detection_Colab	0,667	0,369	0,796	0,631
Vegetation_detection_Colab_alt	0,779	0,478	0,796	0,713
Vegetation_detection_alt	0,718	0,431	0,758	0,699

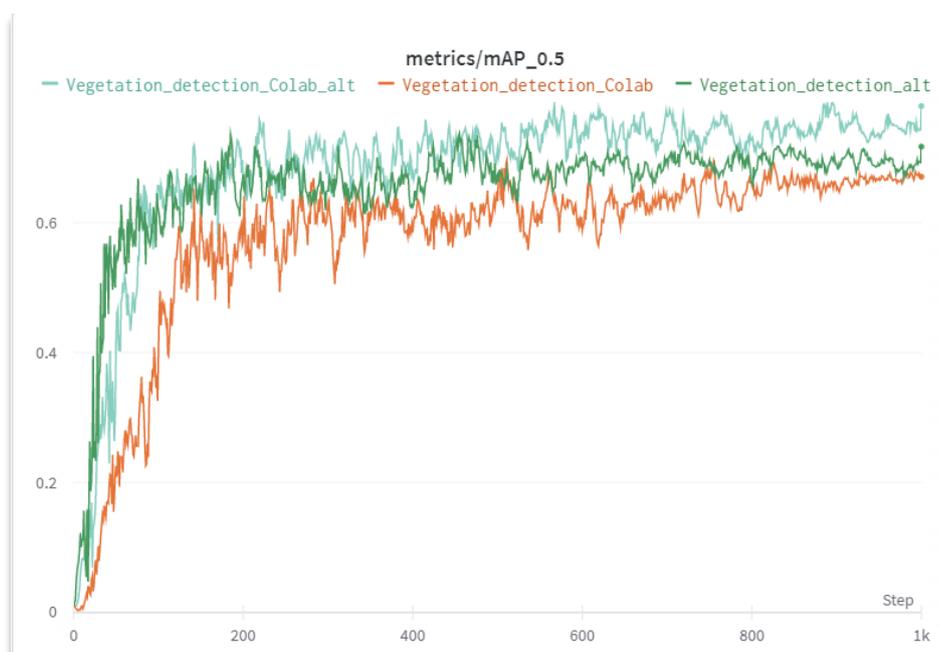


Figura 4.5: Evolução da mAP50 ao longo do treino com o Vegetation_detection.

4.2.3. Resultados *dataset* Water_detection

À semelhança do “Vegetation_dataset”, pode-se observar a partir da tabela que contém os valores obtidos para métricas, que o treino no Google Colab teve vantagem sobre aquele realizado no computador local. Porém, esta comparação não é justa pois o valor de *epochs*

definido para o Colab foi superior em 100, apesar de a *patience* ser a mesma. Esta diferença teve motivo na tentativa de perceber se o modelo continuaria a evoluir para um treino mais longo, que não foi replicado no computador local devido ao enorme tempo requerido, que para o treino anterior já somava perto de 6 horas. Na figura representa-se a evolução da mAP50 ao longo dos treinos, com a linha do gráfico a evidenciar que ainda possa crescer ligeiramente para um treino mais longo.

Tabela 4.7: Valores das métricas de avaliação dos treinos com o Water_detection.

Nome do treino	mAP50	mAP50:95	Precision	Recall
Water_detection	0,828	0,557	0,831	0,698
Water_detection_Colab	0,856	0,573	0,861	0,768

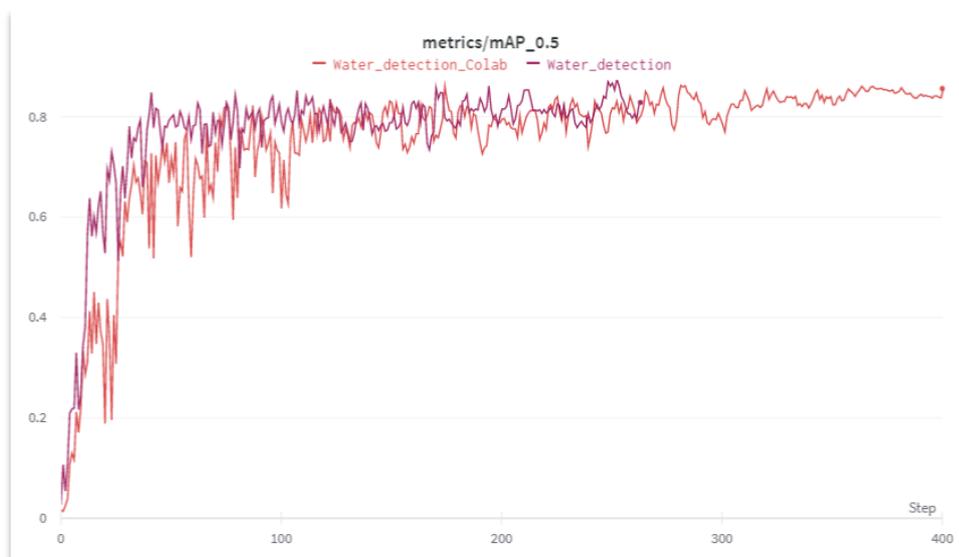


Figura 4.6: Evolução da mAP50 ao longo dos treinos com o Water_detection.

4.3. Avaliação dos modelos treinados

À semelhança do subcapítulo anterior, de seguida apresentam-se tabelas com os valores das métricas de avaliação obtidos para a execução dos ficheiros *weights* do melhor *epoch* obtido no treino, nas imagens de teste de cada *dataset*. De realçar que os modelos não vêem estas imagens durante o treino, apesar de normalmente serem semelhantes ao restante

dataset. Todos os testes de execução dos modelos tiveram um limite de confiança igual a 25% e um limite IoU de 45%. São também apresentados testes qualitativos, recorrendo às imagens recolhidas, entre outras.

4.3.1. Avaliação deteção de troncos

Trees_dataset

Os valores das métricas obtidos para a execução nas imagens de teste deste *dataset* são apresentados na Tabela 4.8, para os diferentes *weights* dos treinos. Observa-se que foram registados valores ligeiramente mais altos do que aqueles obtidos durante o treino para a mAP50 e mAP50:95, que subiram 1,5% e 1,8 a 2%, respetivamente. A Precision do modelo com afinação também subiu 1,6%, assim como a Recall do modelo normal, com um pequeno aumento de 1%. Tal como no treino, a diferença entre os dois modelos é pouco perceptível, ainda assim como a versão com *finetuning* leva uma ligeira vantagem na mAP50 e na Precision, vai ser a selecionada para o teste de deteção nos vídeos a cores e térmico captados no local da recolha de dados. Na Figura 4.7 encontra-se um exemplo de uma das imagens retiradas do vídeo a cores, onde deteta corretamente os troncos de maior visibilidade e em geral as deteções são bastante satisfatórias sem um excesso de *bounding boxes*. Na Figura 4.8 apresenta-se uma imagem retirada do vídeo térmico captado com a câmara FLIR. No vídeo térmico o modelo teve alguma dificuldade em detetar os troncos devido a que as imagens térmicas utilizadas no seu treino estão numa paleta de cores diferente desta.

Tabela 4.8: Valores das métricas de avaliação e tempo de execução nas imagens de teste do Trees_dataset.

Nome do treino	mAP50	mAP50:95	Precision	Recall	t (ms)
Trees_dataset_Colab	0,938	0,629	0,899	0,913	7,4
Trees_dataset_Colab_Fine	0,942	0,629	0,908	0,898	7,4



Figura 4.7: Imagem retirada do vídeo a cores utilizado no teste qualitativo do `Trees_dataset_Colab_Fine`.



Figura 4.8: Imagem retirada do vídeo térmico utilizado no teste qualitativo do `Trees_dataset_Colab_Fine`.

Woods_dataset

Este *dataset* como não possui imagens de teste, vai ser apresentado apenas um resultado qualitativo, utilizando o ficheiro *weights* da versão que obteve as melhores métricas durante o treino, o “`Woods_dataset_Colab_Freeze_Fine`”. Na Figura 4.9 encontra-se um exemplo de uma imagem retirada desse teste onde se ilustra um claro FP. O modelo obteve resultados muito pobres, falhando em detetar qualquer tronco de forma fiável. Isto deve-se principalmente a que o *dataset* de treino contém maioritariamente troncos de grande diâmetro e muito ampliados, ocupando grande parte da imagem.

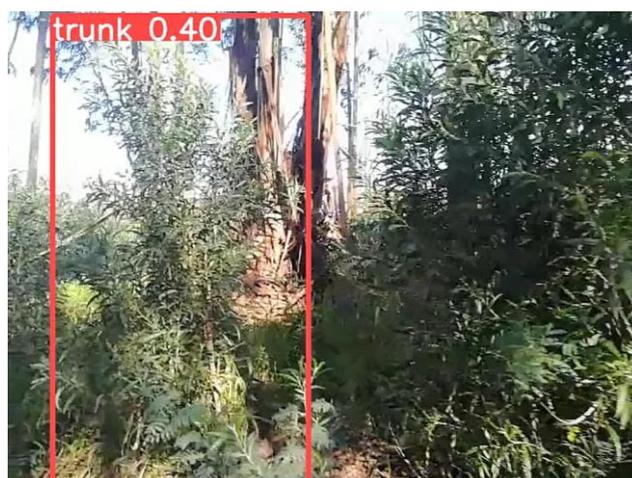


Figura 4.9: Imagem retirada do vídeo utilizado no teste qualitativo do Trees_dataset_Colab_Fine.

TreeDetection_dataset

Na tabela apresentam-se os valores das métricas após execução nas imagens de teste deste *dataset*, onde é possível reparar que os seus valores foram um pouco inferiores aos obtidos durante o treino, com a mAP50 a diminuir 1,5 a 3,4%, a mAP50:95 a diminuir 1,1 a 2,1%, e a Recall a diminuir 0,8 a 1,8% exceto para o “TreeDetection_dataset_Colab_Fine”. A Precision por outro lado aumentou ligeiramente, entre 0,6 e 1,3%. Apesar de os treinos nas diferentes plataformas e com *finetuning* mostrarem resultados muito semelhantes, para o teste qualitativo é selecionado o “TreeDetection_dataset_Colab_Fine”, que obteve resultados interessantes, detetando grande parte dos troncos presentes na imagem ao longo do vídeo. No entanto, fornece alguns FP, como está ilustrado na Figura 4.10 com a incorreta deteção do tronco no solo.

Tabela 4.9: Valores das métricas de avaliação e tempo de execução nas imagens de teste do TreeDetection_dataset.

Nome do treino	mAP50	mAP50:95	Precision	Recall	t (ms)
TreeDetection_dataset	0,834	0,531	0,814	0,765	7,6
TreeDetection_dataset_Colab	0,842	0,526	0,812	0,765	7,6
TreeDetection_dataset_Colab_Fine	0,851	0,548	0,808	0,793	7,5

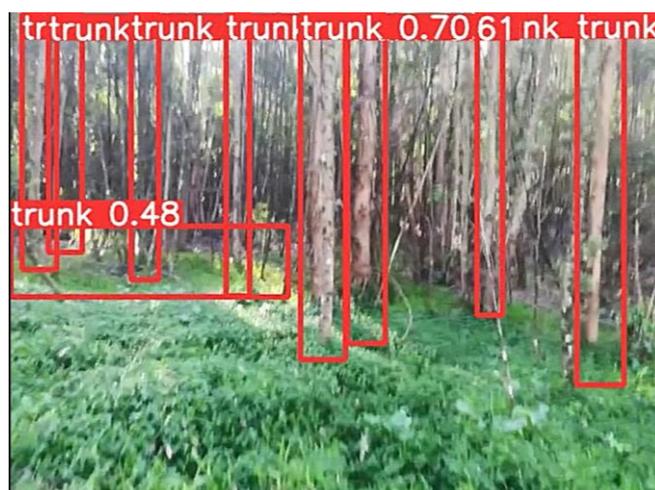


Figura 4.10: Imagem retirada do vídeo utilizado no teste qualitativo do TreeDetection_dataset_Colab_Fine.

Trees_augmented

As métricas de avaliação, representadas na tabela, obtidas nas imagens de teste, apresentam uma considerável redução nos seus valores em relação aos obtidos durante o treino com este *dataset*. A mAP50 diminuiu 11,2%, a mAP50:95 diminuiu 18%, a Precision diminuiu 9,7% e a Recall diminuiu 6,7%. No teste de execução no vídeo a cores, do qual uma das imagens está representada na Figura 4.11, o modelo treinado neste *dataset* teve um desempenho inferior ao “Trees_dataset” e ao “TreeDetection_dataset”, detetando por vezes muitos troncos por imagem sem aplicar as *bounding boxes* de forma adequada, incluindo vários troncos em cada uma e prolongando-as até ao fundo da imagem, como se pode observar na figura referida antes.

Tabela 4.10: Valores das métricas de avaliação e tempo de execução nas imagens de teste do Trees_augmented.

Nome do treino	mAP50	mAP50:95	Precision	Recall	t (ms)
Trees_augmented	0,769	0,335	0,797	0,738	7,4



Figura 4.11: Imagem retirada do vídeo utilizado no teste qualitativo do `Trees_augmented`.

4.3.2. Avaliação deteção de seres vivos

De forma a averiguar se os *weights* pré treinados no MS COCO do YOLOv5-small são adequados à deteção de pessoas em imagens térmicas, foi feita a sua execução no vídeo térmico captado com a câmara FLIR na recolha de dados, num teste qualitativo, onde se apresenta na Figura 4.12 o exemplo de uma imagem retirada do vídeo processado pelo modelo. A pessoa presente na imagem é corretamente detetada com bons valores de confiança mesmo quando parcialmente oculta por vegetação. Da mesma forma na Figura A.6 o modelo deteta a pessoa na imagem a cores de forma ainda mais robusta com 40 a 50% de confiança mesmo com esta a cerca de 40m de distância num vídeo de baixa resolução (640x640).



Figura 4.12: Imagem retirada do vídeo térmico utilizado no teste qualitativo do YOLOv5-small pré treinado.

4.3.3. Avaliação detecção de combustível florestal

De um modo geral, as métricas de avaliação obtidas para a execução nas imagens de teste do “Vegetation_detection” sofreram uma redução significativa, com a mAP50 a diminuir 11,5 a 22,5%, a mAP50:95 a diminuir 5 a 21,5%, a Precision a diminuir 2,9 a 17,3% e a Recall a diminuir 3,7 a 9,5%. Com as exceções da mAP50 e Recall do “Vegetation_detection_Colab”, que aumentaram ligeiramente. Ao contrário dos resultados obtidos para o treino, aqui foi a versão “Vegetation_detection_Colab” treinada no *dataset* original que apresentou os valores mais altos, seguida pela versão do *dataset* alterado treinada no computador local. Dito isto, estas são as versões selecionadas para o teste qualitativo, exemplificado na Figura 4.13 para o “Vegetation_detection_Colab” e na Figura 4.14 para o “Vegetation_detection_alt”, em que os modelos foram executados no vídeo a cores da recolha de dados. Ambos fazem uma detecção razoável da vegetação, com um número elevado de *bounding boxes* presentes numa dada imagem, sendo que o modelo treinado no *dataset* original identifica também corretamente alguns troncos de árvores.

Tabela 4.11: Valores das métricas de avaliação e tempo de execução nas imagens de teste do Vegetation_detection.

Nome do treino	mAP50	mAP50:95	Precision	Recall	T (ms)
Vegetation_detection_Colab	0,681	0,319	0,706	0,641	5,4
Vegetation_detection_Colab_alt	0,554	0,263	0,623	0,618	5,3
Vegetation_detection_alt	0,603	0,301	0,729	0,662	5,3

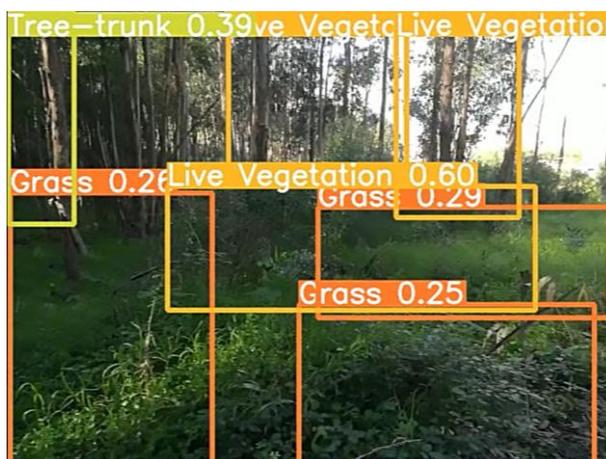


Figura 4.13: Imagem retirada do vídeo utilizado no teste qualitativo do modelo treinado no Vegetation_detection original.



Figura 4.14: Imagem retirada do vídeo utilizado no teste qualitativo do modelo treinado no Vegetation_detection alterado.

4.3.4. Avaliação detecção de cursos de água

Tal como no “Vegetation_dataset”, de um modo geral as métricas obtidas para a execução nas imagens de teste do “Water_detection” sofreram uma diminuição. A mAP50 diminuiu 2,9% a 8,6%, a mAP50:95 diminuiu 4,3 a 6,3%, a Precision diminuiu 7,7 a 10,1% e a Recall diminuiu 3,7% para a versão treinada no Colab, mas aumentou 7,1% para aquela treinada no computador local. De entre as duas versões conclui-se que o melhor desempenho pertence à “Water_detection” apesar das diferenças não serem significativas. Como não foram recolhidas imagens com cursos de água, nem encontrados outros *datasets* relevantes para testar, apresenta-se apenas na Figura 4.15 uma das imagens de teste, onde o modelo detetou o pequeno curso de água apesar da baixa confiança.

Tabela 4.12: Valores das métricas de avaliação e tempo de execução nas imagens de teste do Water_detection.

Nome do treino	mAP50	mAP50:95	Precision	Recall	t (ms)
Water_detection	0,799	0,514	0,730	0,769	7,1
Water_detection_Colab	0,770	0,510	0,784	0,731	7,4



Figura 4.15: Exemplo de imagem de teste do Water_detection em que o modelo foi executado.

4.4. Deteção de obstáculos com o LiDAR

Após os testes com o YOLOv5, foi testado um *package* disponível ao público, da autoria de Yan et al. [70], designado Adaptive Clustering e destinado à deteção de obstáculos a partir da nuvem de pontos obtida pelo LiDAR Velodyne. O seu código é instalado e executado através do ROS, estando desenvolvido para o funcionamento em tempo real, pois subscreve ao tópico publicado pelo Velodyne, “/velodyne_points”, e processa a nuvem de pontos, desenhando caixas visuais que envolvem os conjuntos de pontos (*clusters*) que identifica. Um dos tópicos que publica depois no ROS é o “/adaptive_clustering/markers” que inclui os marcadores visuais sobrepostos à nuvem de pontos. Foi possível testar o seu funcionamento nos dados recolhidos com o Velodyne, reproduzindo-os com o comando “*roslaunch play*”. Para um dos ficheiros gravados na floresta perto do laboratório, o resultado obtido do seu processamento pelo Adaptive Clustering, visualizado no RVIZ, está representado na Figura 4.16 e na Figura A.4. Pela sua observação conclui-se que esta ferramenta possibilita a deteção dos troncos das árvores, identificando também alguns dos arbustos circundantes. O processamento do ficheiro gravado no interior do LAI representa-se na Figura A.5, a partir da qual se observa em comparação com a Figura A.2 do mesmo local, que o Adaptive Clustering identifica os obstáculos definidos, nomeadamente a caixa de cartão, o pilar, a pessoa que está a recolher os dados, e até alguma folhagem da planta ocultada parcialmente pelo pilar.

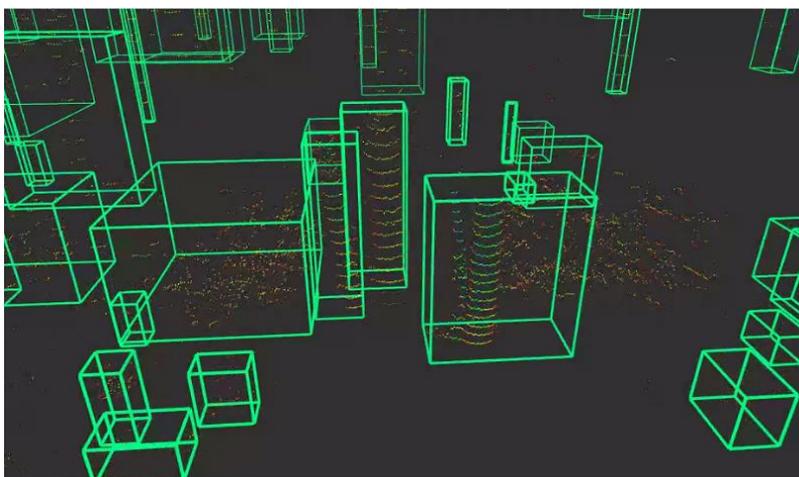


Figura 4.16: Visualização #1 no RVIZ do output gerado pelo Adaptive Clustering nos dados da floresta

5. CONCLUSÕES E TRABALHO REMANESCENTE

Ao longo deste trabalho foi estudada a implementação do modelo CNN YOLOv5-small destinado à detecção de objetos que possam significar obstáculos à navegação de uma máquina autónoma em ambiente florestal. Este foi depois treinado em seis diferentes *datasets* anotados, selecionados da base de dados do Roboflow e de literaturas consultadas, sendo quatro deles destinados à detecção de troncos de árvores, um para a detecção de combustível florestal e o último para a detecção de cursos de água. Foi também averiguada a capacidade de detecção de pessoas em imagens a cores e térmicas com o YOLOv5 pré treinado. O YOLOv5 foi treinado nestes *datasets* de diferentes formas, para avaliar a influência do treino local ou online, e do uso de *finetuning*. O *dataset* que apresentou os melhores resultados no treino foi o “Woods_dataset”, com a sua versão afinada e treinada online, com valores de mAP50, mAP50:95, Precision e Recall correspondentes a 99%, 68,5%, 93,1%, 99,4%, respetivamente. Porém estes valores foram obtidos para um *dataset* relativamente pequeno. O *dataset* que apresentou os melhores resultados na avaliação com as imagens de teste foi o “Trees_dataset” na sua versão afinada e treinada online, registando valores de mAP50, mAP50:95, Precision e Recall superiores aos do treino, correspondentes a 94,2%, 62,9%, 90,8%, 89,8%, respetivamente. É o *dataset* mais adequado à detecção de troncos, no ambiente utilizado nos testes. Ao testar a detecção de troncos com os quatro *datasets* em simultâneo Na detecção de combustível florestal o YOLOv5, nas imagens de teste, obteve valores de mAP50, mAP50:95, Precision e Recall de 68,1%, 31,9%, 70,6% e 64,1% respetivamente, quando treinado online no *dataset* “Vegetation_detection” original. Conclui-se para a detecção de combustível florestal que o uso de um detetor de objetos faz um trabalho razoável, mas um modelo de segmentação semântica, como utilizado em literaturas revistas, seria mais adequado. Quanto à detecção de cursos de água os valores de mAP50, mAP50:95, Precision e Recall nas imagens de teste foram de 79,9%, 51,4%, 73,0% e 76,9%, respetivamente. Porém, o *dataset* utilizado não é adequado a um ambiente florestal, podendo este ser alvo de melhoria pela captura de um *dataset* ou substituição por um mais apropriado já existente. Na comparação entre os métodos de treino verificou-se que são dependentes do tipo de *dataset*, mas em geral o treino online ou local não apresenta diferenças significativas, e da mesma forma, as melhorias introduzidas pelo *finetuning* são pouco perceptíveis nas métricas de avaliação. Por fim, foi testada uma ferramenta que tira

partido da nuvem de pontos gerada pelo Lidar para a deteção de obstáculos que obteve resultados bastante interessantes e mostra o potencial da utilização deste sensor.

Sobre o trabalho remanescente, são imensas as tarefas a abordar para a deteção de obstáculos e navegação fiável num ambiente tão complexo como o florestal. Como exemplos, a realização de mapas de elevação do terreno e detetar o tipo de superfície, para determinar a capacidade de a máquina transpor o mesmo, o uso do LiDAR e da câmara de profundidade para estimar a distância de objetos e o seu tamanho, estimar o diâmetro dos troncos das árvores, até detetar ignições que possam ocorrer durante o trabalho da máquina com a câmara térmica. Como melhoria do estudo feito poderiam ser implementados modelos de segmentação semântica, adquiridos e anotados *datasets* específicos à aplicação, com um treino mais longo dos modelos, a fusão da segmentação semântica com os dados do LiDAR e da câmara de profundidade, e também o teste da capacidade de deteção em condições não ideais que podem surgir durante o trabalho da máquina, como poeira, chuva, nevoeiro, entre outras.

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] M. S. Couceiro, D. Portugal, J. F. Ferreira, and R. P. Rocha, “SEM-FIRE: Towards a new generation of forestry maintenance multi-robot systems,” in 2019 IEEE/SICE International Symposium on System Integration (SII), 2019, pp. 270–276.
- [2] Maria Eduarda Andrada, Joao Filipe Ferreira, David Portugal and Micael Couceiro, "Testing Different CNN Architectures for Semantic Segmentation for Landscaping with Forestry Robotics". In: IROS 2020 Workshop on Perception, Planning and Mobility in Forestry Robotics (WPPMFR 2020), Las Vegas, NV, USA (virtual workshop), 29 October 2020.
- [3] David Jacob Russell, Tito Arevelo, Chinmay Garg, Winnie Kuang, Francisco Yandun, David Wettergreen, George Kantor. “UAV Mapping with Semantic and Traversability Metrics for Forest Fire Mitigation”. ICRA 2022 Workshop in Innovation in Forestry Robotics: Research and Industry Adoption, 2022.
- [4] Dora Lourenço, João Filipe Ferreira and David Portugal, "3D Local Planning for a Forestry UGV based on Terrain Gradient and Mechanical Effort". In: IROS 2020 Workshop on Perception, Planning and Mobility in Forestry Robotics (WPPMFR 2020), Las Vegas, NV, USA (virtual workshop), 29 October 2020.
- [5] Ahmad Kamal Nasir, Andre G. Araujo and Micael S. Couceiro, "Localization and Navigation Assessment of a Heavy-Duty Field Robot", Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2020), Workshop on Perception, Planning and Mobility in Forestry Robotics (WPPMFR 2020).
- [6] A. Milioto and C. Stachniss, “Bonnet: An open-source training and deployment framework for semantic segmentation in robotics using cnns,” CoRR, vol. abs/1802.08960, 2018.
- [7] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks”, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4510-4520
- [8] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” CoRR, vol. abs/1512.03385, 2015.
- [9] A. Valada, R. Mohan, and W. Burgard, “Self-supervised model adaptation for multimodal semantic segmentation,” International Journal of Computer Vision (IJCV), jul 2019. Special Issue: Deep Learning for Robotic Vision.
- [10] L. Biewald, “Experiment tracking with weights and biases,” 2020. Software available from wandb.com.

- [11] W. Hess, D. Kohler, H. Rapp, and D. Andor, “Real-time loop closure in 2d lidar slam,” in 2016 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2016, pp. 1271–1278.
- [12] M. Labb´e and F. Michaud, “Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation,” *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.
- [13] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, “Robot-centric elevation mapping with uncertainty estimates,” in *Mobile Service Robotics*. World Scientific, 2014, pp. 433–440.
- [14] P. Fankhauser, M. Bloesch, and M. Hutter, “Probabilistic terrain mapping for mobile robots with uncertain localization,” *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3019–3026, 2018.
- [15] H. Umari and S. Mukhopadhyay, “Autonomous robotic exploration based on multiple rapidly-exploring randomized trees,” in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017, pp. 1396–1402.
- [16] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge”, 2015
- [17] Alzubaidi, L., Zhang, J., Humaidi, A.J. et al., “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions.”, *J Big Data* 8, 53 (2021).
- [18] Tuan Le, Vignesh Ponnambalam, Jon Glenn Gjevestad and Pål From, "A Supervised Learning Solution for Autonomous Row Following Tasks in Horticulture" In: *IROS 2020 Workshop on Perception, Planning and Mobility in Forestry Robotics (WPPMFR 2020)*, Las Vegas, NV, USA (virtual workshop), 29 October 2020
- [19] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, “Erfnet: Efficient residual factorized convnet for real-time semantic segmentation,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2017.
- [20] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *arXiv preprint arXiv:1804.02767*, 2018.
- [21] Yujian Mo, Yan Wu, Xinneng Yang, Feilin Liu, Yujun Liao, “Review the state-of-the-art technologies of semantic segmentation based on deep learning”, *Neurocomputing*, Volume 493, Pages 626-646, 2022. <https://doi.org/10.1016/j.neucom.2022.01.005>
- [22] Paulo A. S. Mendes, A. Paulo Coimbra and Aníbal T. de Almeida, “Vegetation classification using DeepLabv3+ and YOLOv5”, *ICRA 2022 Workshop in Innovation in Forestry Robotics: Research and Industry Adoption*, 2022.

-
- [23] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," arXiv preprint arXiv:1412.7062, 2014.
- [24] GitHub, Ultralytics YOLOv5, acessado a 26 de janeiro de 2023, em <https://github.com/ultralytics/yolov5>
- [25] S. Macenski, D. Tsai, and M. Feinberg, "Spatio-temporal voxel layer: A view on robot perception for the dynamic world," *Inter-national Journal of Advanced Robotic Systems*, vol. 17, no. 2, p.1729881420910530, 2020.
- [26] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *2010 IEEE international conference on robotics and automation*. IEEE, 2010, pp. 300–307.
- [27] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
- [28] C. Rosmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," in *ROBOTIK 2012; 7th German Conference on Robotics*. VDE, 2012, pp. 1–6.
- [29] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann and T. Bertram, "Efficient trajectory optimization using a sparse model," *2013 European Conference on Mobile Robots*, Barcelona, Spain, 2013, pp. 138-143, doi: 10.1109/ECMR.2013.6698833.
- [30] Samuel Brandenburg, Pedro Machado, Pedro Machado, Nikesh Lama and T.M. McGinnity, "Strawberries Detection Using a Heterogeneous Multi-Processor Platform", 2020, arXiv:2011.03651
- [31] da Silva, D.Q.; dos Santos, F.N.; Sousa, A.J.; Filipe, V. "Visible and Thermal Image-Based Trunk Detection with Deep Learning for Forestry Mobile Robotics". *J. Imaging* 2021, 7, 176. <https://doi.org/10.3390/jimaging7090176>
- [32] Grondin, Vincent, et al. "Tree detection and diameter estimation based on deep learning." arXiv preprint arXiv:2210.17424 (2022).
- [33] da Silva, D.Q.; dos Santos, F.N.; Filipe, V.; Sousa, A.J.; Oliveira, P.M. Edge AI-Based Tree Trunk Detection for Forestry Monitoring Robotics. *Robotics* 2022, 11, 136. <https://doi.org/10.3390/robotics11060136>
- [34] Andrada, M. Eduarda, et al. "Integration of an Artificial Perception System for Identification of Live Flammable Material in Forestry Robotics." *2022 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2022.

- [35] W. Kabsch, “A solution for the best rotation to relate two sets of vectors,” *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, vol. 32, no. 5, 1976.
- [36] J. Ku, A. Harakeh, and S. L. Waslander, “In defense of classical image processing: Fast depth completion on the cpu,” in *15th Conference on Computer and Robot Vision (CRV)*, pp. 16–22, IEEE, 2018.
- [37] Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. SSD: Single Shot MultiBox Detector. In *European Conference on Computer Vision*; Springer: Cham, Switzerland, 2016; pp. 21–37.
- [38] Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv 2015*, arXiv:1502.03167.
- [39] Xiong, Y.; Liu, H.; Gupta, S.; Akin, B.; Bender, G.; Wang, Y.; Kindermans, P.J.; Tan, M.; Singh, V.; Chen, B. MobileDets: Searching for Object Detection Architectures for Mobile Accelerators. *arXiv 2021*, arXiv:2004.14525.
- [40] Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv 2020*, arXiv:2004.10934.
- [41] Ultralytics Community, URL: <https://community.ultralytics.com/t/how-to-combine-weights-to-detect-from-multiple-datasets/38/26>
- [42] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 2015*, arXiv:1409.1556.
- [43] Chien-Yao Wang; Alexey Bochkovskiy; Hong-Yuan Mark Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors”, *arXiv:2207.02696*, 2022.
- [44] GitHub, WongKinYiu YOLOv7, acessido a 26 de janeiro de 2023, em <https://github.com/WongKinYiu/yolov7>
- [45] GitHub, Ultralytics YOLOv8, acessido a 26 de janeiro de 2023, em <https://github.com/ultralytics/ultralytics>
- [46] Intel, Depth Camera D435i, <https://www.intelrealsense.com/depth-camera-d435i/>
- [47] Teledyne FLIR, FLIR ADK, <https://www.flir.eu/products/adk/>
- [48] Velodyne Lidar, Puck, <https://velodynelidar.com/products/puck/>
- [49] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, “You Only Look Once: Unified, Real-Time Object Detection”, *arXiv:1506.02640*, 2015.

-
- [50] Joseph Redmon, “Darknet: Open Source Neural Networks in C”, <http://pjreddie.com/darknet/>, 2013--2016.
- [51] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár, “Microsoft COCO: Common Objects in Context”, arXiv:1405.0312, 2014.
- [52] Quigley, Morgan & Conley, Ken & Gerkey, Brian & Faust, Josh & Foote, Tully & Leibs, Jeremy & Wheeler, Rob & Ng, Andrew. (2009). ROS: an open-source Robot Operating System. ICRA Workshop on Open Source Software. 3.
- [53] Xuan, Zhang and David, Filliat, “Real-time voxel based 3D semantic mapping with a hand held RGB-D camera”, GitHub repository, 2018.
URL: https://github.com/floatlazer/semantic_slam
- [54] R. Padilla, S. L. Netto and E. A. B. da Silva, "A Survey on Performance Metrics for Object-Detection Algorithms," 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), Niteroi, Brazil, 2020, pp. 237-242, doi: 10.1109/IWSSIP48289.2020.9145130.
- [55] Dwyer, B., Nelson, J. (2022), Solawetz, J., et. al. Roboflow (Version 1.0) [Software]. Available from <https://roboflow.com>. computer vision.
- [56] Bisong, E. (2019). Google Colaboratory. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-4470-8_7
- [57] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.
- [58] Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv 2017, arXiv:1704.04861.
- [59] Howard, A.; Sandler, M.; Chu, G.; Chen, L.C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. Searching for MobileNetV3. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Republic of Korea, 27 October–2 November 2019.
- [60] Tan, M.; Pang, R.; Le, Q.V. EfficientDet: Scalable and Efficient Object Detection. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020; pp. 10778–10787. [CrossRef] 38. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv 2020, arxiv:2004.10934.
-

- [61] Wang, C.Y.; Mark Liao, H.Y.; Wu, Y.H.; Chen, P.Y.; Hsieh, J.W.; Yeh, I.H. CSPNet: A New Backbone that can Enhance Learning Capability of CNN. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Seattle, WA, USA, 14–19 June 2020; pp. 1571–1580.
- [62] Wang, C.Y.; Yeh, I.H.; Liao, H.Y.M. You Only Learn One Representation: Unified Network for Multiple Tasks. arXiv 2021, arXiv:2105.04206.
- [63] Carion, N.; Massa, F.; Synnaeve, G.; Usunier, N.; Kirillov, A.; Zagoruyko, S. End-to-End Object Detection with Transformers. In Proceedings of the Computer Vision—ECCV 2020; Vedaldi, A., Bischof, H., Brox, T., Frahm, J.M., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 213–229.
- [64] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, “SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers,” pp. 1–18, 2021. [Online]. Available: <http://arxiv.org/abs/2105.15203>
- [65] Roboflow, Tree Detection Dataset, Open Source Dataset, Derrick, Roboflow Universe, nov 2022. URL: <https://universe.roboflow.com/derrick-unczb/tree-detection-tnfyc>, visited on 06-02-2023.
- [66] Roboflow, woods Dataset, Open Source Dataset, zumoku, Roboflow Universe, jun 2022. URL: <https://universe.roboflow.com/zumoku/woods-oaoq4>, visited on 06-02-2023.
- [67] Roboflow, Trees augmented 2 Dataset, Open Source Dataset, Andrey Markov, Roboflow Universe, oct 2021. URL: <https://universe.roboflow.com/andrey-markov/trees-augmented-2>, visited on 06-02-2023.
- [68] Roboflow, H3K Dataset, Open Source Dataset, New Workspace, Roboflow Universe, aug 2021. URL: <https://universe.roboflow.com/new-workspace-rhchg/h3k-5vcm2>, visited on 06-02-2023.
- [69] Roboflow, Vegetation_detection Dataset, Open Source Dataset, paulo_mendes_33@hotmail.com, Roboflow Universe, feb 2023. URL: https://universe.roboflow.com/paulo_mendes_33-hotmail-com/vegetation_detection, visited on 06-02-2023.
- [70] Zhi Yan and Tom Duckett and Nicola Bellotto, “Online learning for 3D LiDAR-based human detection: Experimental analysis of point cloud clustering and classification methods”, Autonomous Robots, 2019. URL: https://github.com/yzrobot/adaptive_clustering

ANEXO A

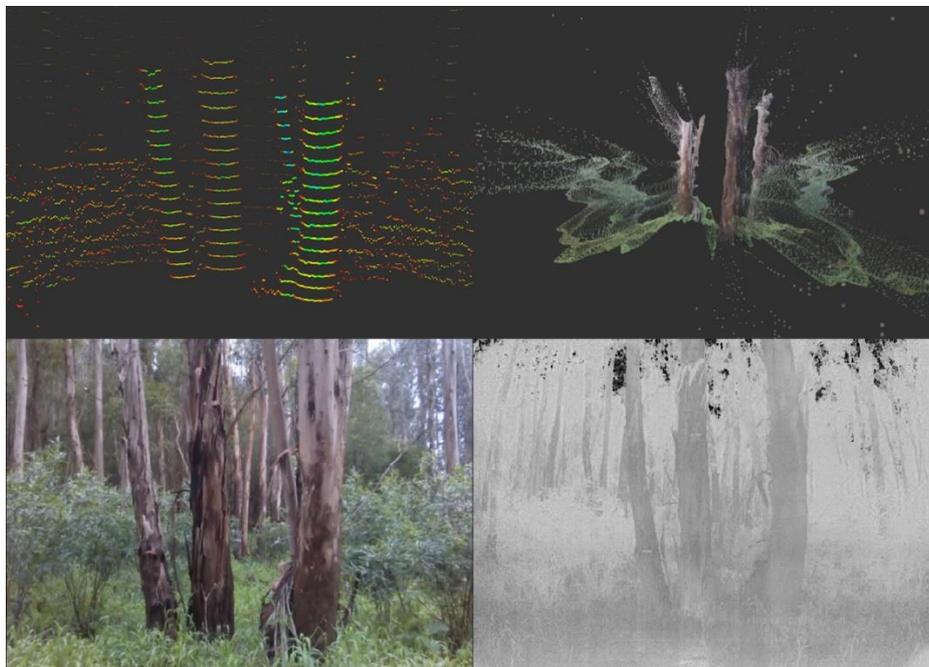


Figura A.1: Visualização no RVIZ dos dados obtidos com o Velodyne (esquerda, cima) e com a Realsense (direita, cima), e imagens retiradas dos vídeos a cores (esquerda, baixo) e térmico (direita, baixo), a 2,5m dos troncos.



Figura A.2: Cenário da recolha de dados com o LiDAR no interior do edifício do LAI.

```
import os, fnmatch

def findReplace(directory, filePattern):
    for path, dirs, files in os.walk(os.path.abspath(directory)):
        for filename in fnmatch.filter(files, filePattern):
            filepath = os.path.join(path, filename)

            file=open(filepath,"r")
            replaced_content = ""
            for line in file:
                line=line.strip()
                if line[0]=="0":
                    new_line=line.replace('0','0',1)
                    replaced_content = replaced_content + new_line + "\n"
                if line[0]=="1":
                    new_line=line.replace('1','0',1)
                    replaced_content = replaced_content + new_line + "\n"
                if line[0]=="2":
                    new_line=line.replace('2','0',1)
                    replaced_content = replaced_content + new_line + "\n"

            file.close()

            write_file=open(filepath, "w")
            write_file.write(replaced_content)
            write_file.close()

findReplace('Downloads/Datasets/TreeDetection_dataset/train/labels/', "*.txt")
```

Figura A.3: Exemplo do código em Python utilizado para a alteração das classes nos ficheiros das anotações.

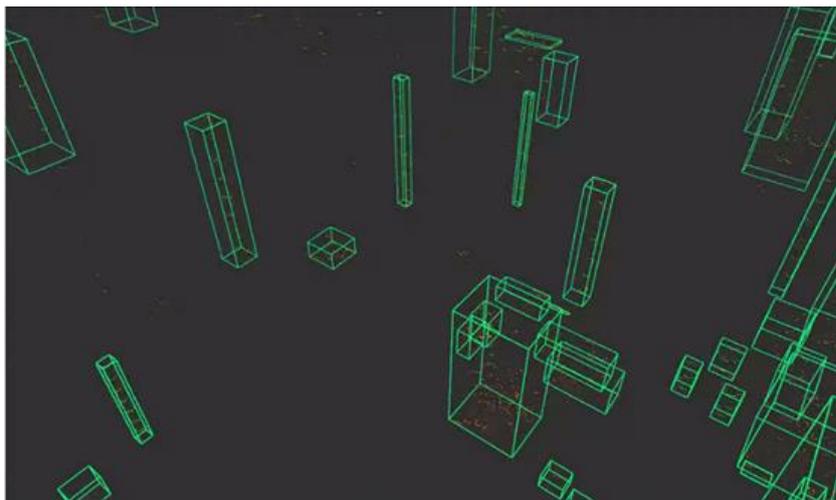


Figura A.4: Visualização no RVIZ do output gerado pelo Adaptive Clustering nos dados captados na floresta, com ênfase na ilustração dos troncos detetados.

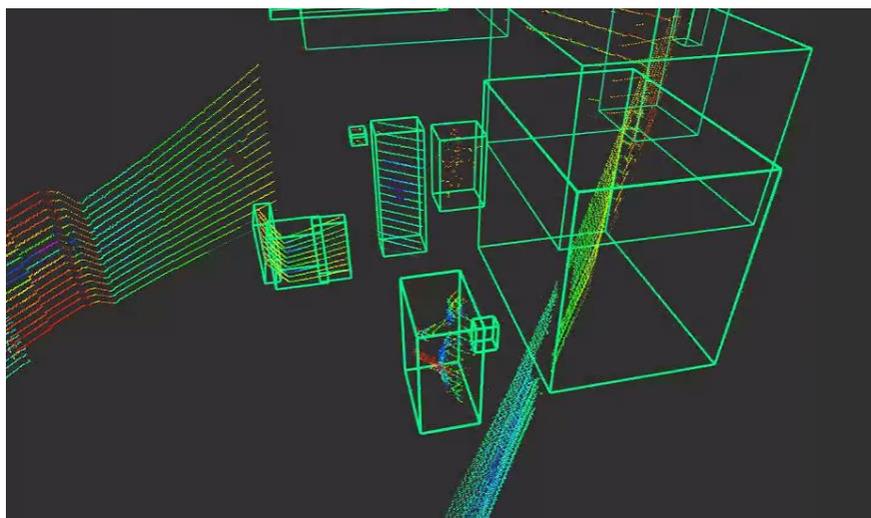


Figura A.5: Visualização no RVIZ do output gerado pelo Adaptive Clustering nos dados captados no LAI.

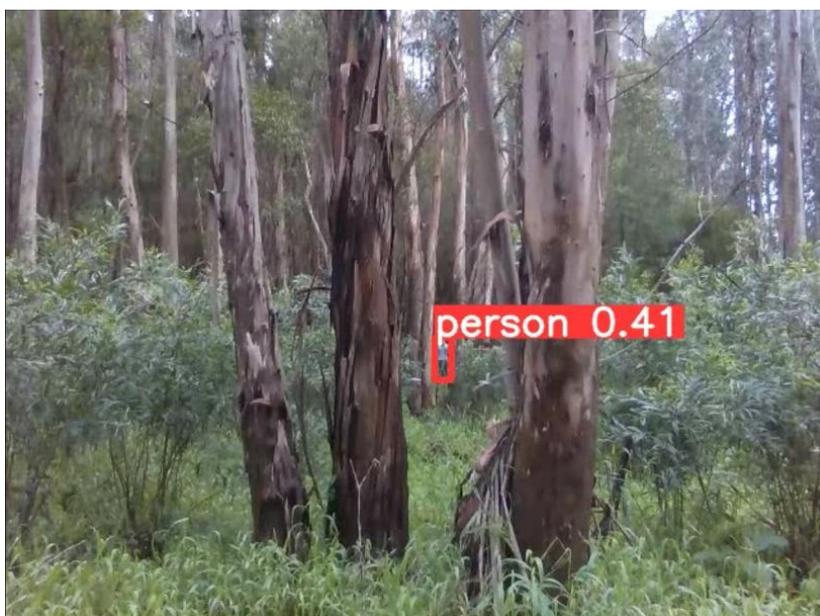


Figura A.6: Imagem retirada do vídeo a cores utilizado no teste qualitativo do YOLOv5-small pré treinado.