UNIVERSIDADE
BEIRA INTERIOR

Sistema de Navegação Autónoma de Rover Robótico Multitarefa Destinado a Atividades Agrícolas baseado em Visão Computacional por Deteção de Troncos de Árvores
Aplicação a Pomares de Pessegueiros

José Pedro Gouveia Pires Simões

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletromecânica
(2º ciclo de estudos)

Orientador: Prof. Doutor Pedro Dinis Gaspar

outubro de 2021

Agradecimentos

Aos meus pais e irmã pelo apoio emocional, pelos incentivos de não desanimar nos momentos mais difíceis, nunca desistirem de mim, e acreditarem e investirem na minha pessoa de modo a ter um futuro e carreira que eles acreditam que eu mereça.

À minha namorada Micaela pelo carinho, amor e palavras de apoio. A sua presença incondicional e preocupação foram imprescindíveis para a minha estabilidade moral perante todas as dificuldades que compareceram.

À minha tia Carlota pela preocupação constante de que cumpria todos os requisitos, assegurando que mantinha uma posição responsável.

Ao Professor Doutor Pedro Dinis Gaspar pelas suas palavras de sabedoria que me guiaram do início ao fim deste projeto, por todos os recursos e meios que garantiu que houvesse para a boa execução desta dissertação e pela orientação pontual, bem guiada e sempre com grande ambição de um bom trabalho realizado.

Ao Engenheiro Eduardo Assunção por toda a sua ajuda, novos conhecimentos e práticas transmitidas, pela sua companhia e pelas suas opiniões certeiras de soluções a bloqueios de progresso.

Ao técnico Pedro Guerra pela sua disponibilidade e orientação de todos os recursos necessários.

Aos meus amigos que me fizeram companhia durante o desenvolvimento desta dissertação e durante todo o percurso académico, e por palpites construtivos que desencadearam ideias para soluções.

Por fim, ao meu cão, Balu, por, apesar de não saber ler nem nos entender, o entusiasmo com que ele se despedia ou me recebia todos os dias, independentemente do meu humor, elevava sempre o espírito.

A investigação desenvolvida nesta dissertação faz parte das atividades do projeto PrunusBot – Sistema robótico aéreo autónomo de pulverização controlada e previsão de produção frutícola, Operação nºPDR2020-101-03158 (líder), Consórcio nº340, Iniciativa nº 140, promovido pelo PDR2020 e cofinanciado pelo FEADER e União Europeia no âmbito do Programa Portugal 2020.

A todos dedico esta dissertação, desejo que inspire e que sirva de prova que não falharei a nenhum dos demais e que colabore com uma pequena ajuda para um futuro melhor deste planeta e humanidade.

Resumo

Introduzir a robótica na agricultura pode permitir uma subida na produtividade e reduzir os custos e desperdícios associados. As suas capacidades podem ser ampliadas comparativamente à função humana, permitindo que um robô consiga efetuar o trabalho que um humano realizaria, mas com maior precisão, repetibilidade e sem fadiga. Nesta dissertação é desenvolvido um algoritmo de reconhecimento de troncos de pessegueiros dispostos em filas comuns de um pomar, como sistema de navegação autónoma e auxiliar anti-choque, de um rover robótico destinado a aplicações agrícolas.

A técnica utilizada para esta solução tecnológica foi a visão computacional, ou seja, a criação de um modelo de deteção de objetos com base em Redes Neuronais Convolucionais. A plataforma de construção do algoritmo foi *Tensorflow*, com destino a ser implementado num *Raspberry Pi 4*. O modelo foi alicerçado num sistema de deteção SSD *MobileNet* 640x640 com aprendizagem de transferência da base COCO17. Apuraram-se 89 imagens para o treino do modelo, recorrendo-se a 90% para treino e as restantes para teste. Aplicou-se ao modelo uma quantização integral completa, convertendo o modelo de *32float* para *uint8* e compilando-o para suportar *Edge TPU*, adequado para aplicações móveis.

A estratégia de orientação consiste em duas condições: uma deteção dupla cria uma linha imaginária com estrutura de uma função linear que atualiza a cada deteção da mesma configuração e, a partir do declive desta reta ou do desvio horizontal das caixas delimitadoras de deteções únicas, são dadas ordens de viragem ou de manter a marcha.

Uma avaliação aritmética do modelo mostrou que este tem uma precisão e uma revocação de 94.4%. Após a quantização, estes valores passaram a 92.3% e 66.7%, respetivamente. Estas métricas e os resultados das simulações mostram que, estatisticamente, o modelo mostrou-se adequado a cumprir os objetivos propostos.

Palavras-chave

Agricultura de precisão, Deteção de objetos, Pomar, Navegação, Robô terrestre, Visão robótica, Redes Neuronais Convolucionais, Tensorflow, Raspberry Pi 4, SSD MobileNet, Quantização

Abstract

Introducing robotics in agriculture can allow a raise in productivity and a reduction on costs and waste. Its capabilities can be enhanced to or above the human level, enabling a robot to function like a human does, but with higher precision, repeatability and with little to no effort. This dissertation develops a detection algorithm of peach trunks in orchard rows, as an autonomous navigation and anti-bump auxiliary system of a terrestrial robotic rover for agricultural applications.

The approach involved computational vision, more specifically, the creation of an object detection model based on Convolutional Neural Networks. The framework of this algorithm is *Tensorflow*, for implementation in a *Raspberry Pi 4*. The model's core is the detection system SSD *MobileNet* 640x640 with transfer learning from the COCO 2017 database. 89 pictures were capture for the database of the model, which 90% were used for training and the other 10% for testing. The model was converted for mobile applications with a full integer quantization, from *32float* to *uint8*, and it was compiled for *Edge TPU* support.

The orientation strategy consists in two conditions: a double detection forms a linear function, represented by an imaginary line, which updates every two simultaneous trunks detected. Through the slope of this function and the horizontal deviation of a single detected bounding box from created line, the algorithm orders the robot to adjust the orientation or keep moving forward.

The arithmetic evaluation of the model shows a precision and recall of 94.4%. After the quantization, the new values of these metrics and 92.3% and 66.7%, respectively. These simulation results prove that, statistically, the model is able to perform the navigation task.

Keywords

Precision agriculture, Object detection, Orchard, Navigation, Terrestrial rover, Robotic vision, Convolutional Neural Networks, Tensorflow, Raspberry Pi 4, SSD MobileNet, Quantization

Índice

Lista de Figuras

Lista de Tabelas

Nomenclatura

Geral:

b	Ordenada na origem da função linear;
m	Declive da função linear;
q	Valor quantiado;
r	Valor real pré-quantiado;
S	Escala de quantização;
z	Ponto zero da quantização.

Acrónimos:

2D	Duas dimensões;
3D	Três dimensões;
ANN	Artificial Neural Networks;
API	Application Programming Interface;
BPNN	Back Propagation Neural Network;
CEP	Circle of Error Probable;
CNN	Convolutional Neural Networks;
COCO	Common Objects in Context;
CPU	Central Process Unit;
CUDA	Compute Unified Device Architecture;
EIF	Extended Information Filter;
EKF	Extended Kalman Filter;
FLOPS	FLoating-point Operations Per Second;
FPN	Feature Pyramid Networks;
GLONASS	Globalnaya Navigatsionnaya Sputnikovaya Sistema;
GNSS	Global Navigation Satellite System;
GPS	Global Positioning System;
GPU	Graphics Processing Unit;
HSV	Hue, Saturation, Value;
IDE	Integrated Development Environment;
IMU	Inertial Measurement Unit;
IoU	Intersection over Unit;

LiDAR	Light Detection And Ranging;
ONU	Organização das Nações Unidas;
PDA	Personal Digital Assistant;
RANSAC	RANdom SAmple Consensus;
ReLU	Rectified Linear Unit;
RGB	Red Green Blue;
RMSE	Root Mean Square Error;
RTK	Real Time Kinematic;
SLAM	Simultaneous Localisation And Mapping;
SVM	Support Vector Machines;
TIC	Tecnologias de Informação e Comunicação;
UAV	Unmanned Aerial Vehicle;
UGV	Unmanned Ground Vehicle;
UKF	Unscented Kalman Filter.

1. Introdução

1.1. Enquadramento

A agricultura é a base da produção de alimentos. É dos setores de atividade humana mais antigos - desde há milhares de anos que o Homem trabalha a terra e cultiva alimentos para sustento próprio ou de uma comunidade. Outrora, as comunidades eram relativamente pequenas e um maior número de pessoas não tinha acesso ou possibilidade para trabalhar noutro setor, logo os agricultores existentes eram suficientes para prover a população alvo.

Atualmente, a crescente população mundial, densidades populacionais nacionais distintas, as grandes diferenças de poder económico, as alterações climáticas e o afastamento da população jovem do setor primário trazem dificuldades ao domínio agrícola em cumprir a alta procura de produção, sem haver mão-de-obra suficiente e desperdício de recursos que podem levar a maiores gastos ou afetar o ambiente (Calicioglu et al., 2019).

Introduzir a robótica na agricultura pode permitir uma subida na produtividade e reduzir os custos e desperdícios associados. A robótica tende a imitar os comportamentos e gestos humanos nas áreas em que é aplicada. As suas capacidades podem ser ampliadas comparativamente à função humana, permitindo que um robô consiga efetuar o trabalho que um humano realizaria, mas com maior precisão, repetibilidade e sem fadiga. Algumas atividades agrícolas que podem ser robotizadas e automatizadas são a colheita, verificação do estado de crescimento da cultura, aplicação de pesticidas e herbicidas, navegação autónoma, mapeamento de pomares e campos, entre outros (Duckett et al., 2018).

Apesar de todas as vantagens que a introdução da robótica em qualquer ramo aparenta trazer, um robô não tem a capacidade de raciocínio que um humano tem. Com isto, um robô, num ambiente com eventos aleatórios ou sujeitos à variação dos fatores ambientais, só funcionaria corretamente se estivesse programado para eventos específicos com algum padrão ou caraterística distinta. Este aspeto está presente na agricultura, pois terá que se sujeitar a terrenos irregulares, culturas variáveis ou distintas, crescimento das plantas, condições ambientais variáveis, terá que transmitir segurança tanto a operários como a si mesmo, e o manuseamento de certas atividades, plantas ou frutos é diferente e requer técnicas e cuidados diferentes (Marinoudi et al., 2019).

A Agricultura 4.0 foi oficializada 2 anos após a Indústria 4.0, em 2015. Esta já abordava os programas computacionais e o desenvolvimento robótico, sendo a novidade a implementação da Internet das Coisas, *Big Data*, inteligência artificial, métodos de apoio à decisão e sensorização remota. Todos estes conceitos introduzidos numa agricultura mais avançada vincam as quatro grandes necessidades: produtividade, utilização racional de recursos, adaptação às alterações climáticas e redução do desperdício alimentar (Zhai et al., 2020).

1.2. O problema em estudo e a sua relevância

A falta de mão-de-obra no setor agrícola tem vindo a crescer nos últimos anos, a um nível global. A população jovem de países desenvolvidos que cresce em zonas urbanas tende a manter-se nelas e trabalhar num setor industrial ou de serviços, e grande parte daquela que nasce e cresce em zonas rurais pretende mover-se para os centros urbanos à procura de um emprego que não se relaciona com o campo. O setor primário nos países desenvolvidos encontra-se envelhecido ou necessita de recorrer à imigração de pessoas que se oferecem para trabalhar nesse setor. A falta de mão-de-obra é um problema pois não é suficiente para cumprir as necessidades para aprovisionar uma população crescente (Duckett et al., 2018).

Em Portugal, em 2020, apenas cerca de 260 milhares de pessoas se encontram empregadas no setor primário, ou seja 5,4% da população empregada – valor muito reduzido comparado ao de 2000, sendo este de cerca de 645 milhares de pessoas (PORDATA - População Empregada: Total e Por Grandes Sectores de Actividade Económica, n.d.). Nos 17 objetivos de desenvolvimento sustentável definidos pela ONU, o segundo mais importante, logo a seguir à exterminação da pobreza, é a erradicação da fome, alcance da segurança alimentar, nutrição equilibrada e a promoção de uma agricultura sustentável (ONU, n.d.).

A preocupação de um futuro que pode levar à escassez de alimentos e nutrição, simplesmente porque o setor agrícola não tem capacidades, mão-de-obra e recursos para acompanhar a crescente procura, é real e não pode ser ignorada. Várias soluções são avançadas, sendo uma dela a robótica automatizada aplicada no setor agrícola (Zhai et al., 2020).

Um robô específico para uma tarefa agrícola pode cumpri-la com precisão, eficácia, sem repercussões físicas ou psicológicas, estando apenas limitado à sua autonomia. O interesse desta área por investigadores e investidores permite a sua expansão, o que oferece maiores funcionalidades disponíveis e avanços tecnológicos que implicam um aumento na produtividade, redução de desperdício de recursos e de alimento, maior preocupação e cuidado com o fator ambiental. Todos estes fatores combinados apresentam uma solução para o problema anteriormente referido.

1.3. Objetivos e contribuição da dissertação

A plataforma robótica desenvolvida por Veiros (2020) destina-se a múltiplas tarefas agrícolas, entre as quais, apanha de frutos caídos no chão, aplicação particularizada de herbicida, contagem de frutos nas árvores, entre outros projetos que poderão ser desenvolvidos. Esta plataforma robótica pretende-se que seja autónoma na realização das tarefas. Assim sendo, é necessário o desenvolvimento de um sistema de navegação autónoma. Nesta dissertação é então desenvolvido um algoritmo de reconhecimento de troncos de pessegueiros dispostos em filas comuns de um pomar, como sistema de navegação e auxiliar antichoque, para fins de locomoção autónoma do rover robótico destinado a aplicações agrícolas.

Pretende-se que o rover robótico faça uso de dois sistemas de controlo, regulação e comando da locomoção. Um será baseado em sistema de posicionamento global (GPS – *Global Positioning System*), permitindo que o trajeto autónomo a ser realizado pelo rover robótico seja pré-programado. Este sistema de locomoção não faz parte desta dissertação. O outro sistema de controlo, regulação e comando da locomoção autónoma fará uso de uma câmara RGB (Red-Green-Blue) disposta sobre o rover e de um algoritmo de deteção de troncos. Este sistema será prioritário para evitar que a plataforma robótica choque com as árvores e para garantir que esta se desloque numa linha reta sem grandes desvios com uma margem de erro reduzida. É sobre este sistema de locomoção que a presente dissertação incide.

O algoritmo terá como input vídeo a tempo real captado por uma câmara montada na plataforma e, através de técnicas de Inteligência Artificial por Aprendizagem Profunda (*Deep Learning*), mais especificamente por Redes Neuronais Convolucionais (*Convolutional Neural Networks* – CNN), fará a segmentação da imagem e detetará os troncos das árvores. A estratégia de orientação será a criação de uma função linear, utilizando dois pontos dos limites de deteção de dois troncos, formando uma linha que acompanha aproximadamente a orientação da fila de árvores do pomar. Ordens serão enviadas ao robô para virar para a esquerda, direita ou prosseguir a marcha caso a caixa que delimita a deteção do tronco se encontra muito desviada da linha ou, no caso de detetar dois troncos, o declive da função linear atingir valores que indiquem que o robô segue uma trajetória irregular e potencial de choque.

Uma deslocação autónoma permite uma melhor gestão de tempo, pois não há necessidade de haver um operário a controlar a navegação do robô, logo o robô efetua uma tarefa demorada e mundana sem interrupções a qualquer hora do dia ou até mesmo da noite (Barawid et al., 2007). Estas novas soluções tecnológicas aplicadas à agricultura conduzem ao aumento da eficiência dos processos e consequentemente ao aumento da produtividade, redução do desperdício, e maior resiliência aos fatores dinâmicos inerentes à atividade agrícola, como sejam as condições ambientais adversas, pragas e pestes.

1.4. Visão geral e organização da dissertação

Esta dissertação está estruturada em cinco capítulos gerais: Introdução, Estado da Arte, Materiais e Métodos, Análise e Discussão de Resultados e Conclus.

A Introdução apresenta ao leitor uma visão abrangente e objetiva do tema geral da tese, incluindo as potencialidades da robótica perante as dificuldades no setor agrícola.

O segundo capítulo, Estado da Arte, funda os conhecimentos, as investigações e as várias técnicas abordadas na navegação autónoma de robôs aplicados na agricultura. Dessas técnicas, as redes neuronais convolucionais foram as optadas para o desenvolvimento desta tese, daí o aprofundamento teórico e a explicação do funcionamento estrutural e íntegro dos sistemas.

Os Materiais e Métodos apresentam a escolha e a explicação da abordagem a softwares e hardwares aplicados na fase prática do projeto. Inclui, também, os passos que o autor realizou, tal como as estratégias definidas para atingir um algoritmo funcional para a navegação autónoma da plataforma.

A secção da Análise e Discussão de Resultados une todos os desfechos do projeto, tal como dados matemáticos que comprovam a boa execução das tarefas e discussão dos ensaios e simulações realizadas.

Por fim, a Conclus integra todos os conhecimentos e resultados captados no andamento do projeto e transmite uma visão que finda a refletir o envolvimento que o bom sucesso das tarefas efetuadas terão no futuro desenvolvimento de projetos nesta plataforma robótica, e sugestões para o aperfeiçoamento do sistema criado nesta dissertação.

2. Estado da Arte

A robótica aplicada no setor agrícola proporciona um aumento no desempenho nas diversas tarefas em que se enquadra. Depois de se verificar que os robôs operam superiormente controlados ou supervisionados, o passo seguinte á a sua autonomização, incluindo a navegação.

A navegação de um robô é o processo que um robô efetua para se mover do ponto de partida até um local de chegada, seguindo um caminho projetado com recurso à informação local e ambiental que capta com os seus sensores e aparelhos auxiliares. Um robô para realizar uma travessia com sucesso necessita de ter boa perceção, registar a sua localização, ser cognitivo e possuir um bom controlo de movimento (Gao et al., 2018).

Existem inúmeras abordagens para o planeamento de navegação autónomo de um robô de aplicações agrícolas, seja com recurso a GPS ou sensores que permitem o mapeamento, localização, deteção de plantas, reconhecimento e medição (odometria).

No universo da robótica, os sensores internos dos mais comuns são os encoders rotativos, acelerómetros e giroscópios. Estes sensores detetam a velocidade linear e angular como a posição, detetando o comportamento dos componentes internos dos robôs. Estes métodos são ótimos em ambientes fechados, pois os agentes externos não interagem tanto com o robô. No entanto, num ambiente agrário onde o terreno é irregular e muitos obstáculos podem ser encontrados, os dados fornecidos por estes sensores são pouco fiáveis pois os agentes externos no meio ambiente, como a temperatura, a gravidade e a derrapagem das rodas podem rapidamente tornar estes sensores pouco recomendados na aplicação em pomares, vinhas ou quintas (Bechar & Vigneault, 2016). Daí a necessidade de outros aparelhos auxiliares ou soluções diferentes de navegação, tais como sistemas de localização por satélite ou sensores de luz ou proximidade. Estes serão descritos nos seguintes subcapítulos.

2.1. Agricultura de precisão

A agricultura de precisão é um sistema de gestão de culturas baseado na variabilidade temporal e espacial das caraterísticas do terreno. Este sistema representa um esquema baseado em informação e produção agrícola com o objetivo de aumentar o rendimento, eficiência, produtividade e lucros. A agricultura de precisão acentua a análise de dados espácio-temporais combinados, ao invés de informações separadas e individuais, recorrendo a gestão de informação, processamento

computacional, posicionamento no terreno, monitorização dos produtos, e sensorização de proximidade e remota (Pallottino et al., 2019).

Na agricultura de precisão, mais especificamente na área de navegação autónoma, muitos avanços e estudos foram já realizados. A seguir, encontra-se uma breve revisão de soluções já existentes (Bonadies & Gadsden, 2019).

Atualmente, os sistemas de informação geográfica e sistemas de posicionamento global (GPS – *Global Positioning System*) são os mais utilizados. De modo que estes sistemas tenham alta precisão em aplicações agrícolas, recetores GPS de precisão são utilizados, tais como sistemas de Posicionamento Cinemático em Tempo-Real (RTK – *Real Time Kinematics*) ou GPS Diferenciais, que são estações de referência localizadas no terreno de aplicação. Estes sistemas de simplificação, reduzem o erro de um sinal convencional de GPS de alguns metros para a ordem do centímetro.

Uma alternativa ao GPS é o recurso a sensores ou câmaras para avaliar o ambiente envolvente próximo da plataforma robótica. Sensores estes podem ser radares ou ultrassónicos que detetam obstáculos de grandes dimensões ou marcar o terreno com pontos de interesse. Sistemas de deteção de luz e distância, ou LiDAR (*Light Detection And Ranging*), podem ser utilizados para analisar o ambiente redor e conceber uma projeção bidimensional (2D) ou tridimensional (3D).

A visão por câmara pode ser usada para produzir imagens 2D ou 3D. De seguida, estas imagens serão processadas num algoritmo que permitirá a identificação dos objetos com limites e cores diferentes e categoriza em obstáculos a evitar ou não. Esta solução depende muito da luminosidade, pois esta é que dá a tonalidade dos obstáculos. Em contraste, a visão estéreo por câmaras cria imagens 3D do ambiente em redor, imitando a visão humana – combinando dados de duas imagens separadas da mesma cena e interpreta os planos dos vários objetos.

Entre todos os robôs já desenvolvidos, as plataformas robóticas movidas a rodas são das mais utilizadas. Estas estruturas, no setor agrícola, possuem, geralmente, quatro rodas, podendo estas serem movidas com recurso a lagartas, mas com menos frequência. A locomoção e viragem de quase todas estas estruturas é feita com tração das duas rodas de trás e as duas rodas da frente encarregam-se da viragem (Gao et al., 2018).

2.2. Sistemas de navegação autónoma

Neste subcapítulo descrevem-se várias abordagens a soluções estudadas e desenvolvidas relativamente à navegação autónoma de uma plataforma robótica em pomares. Vários tipos de abordagens com vários algoritmos, planos de navegação, sensores, aparelhos ou sistemas auxiliares podem ser utilizados. Estes variam em complexidade, custo e hardware e software.

Um exemplo de uma solução inovadora foi proposta por Edlerman & Linker (2019), relativa a um sistema mestre-escravo que permite orientar mais que um robô simultaneamente num pomar ou vinha, recorrendo a um drone que observa o terreno panoramicamente. Este drone beneficia de um sinal GPS juntamente com a sua câmara que proporciona uma boa visão do campo. Esta combinação de sistemas adquire informação local e posicional dos robôs terrestres e envia-a, recorrendo a uma frequência rádio, aos próprios, como representa a Figura 1. O objetivo deste projeto foi minimizar o equipamento e o pesado processamento computacional num único robô terrestre, para desta forma se utilizarem vários robôs terrestres, simultaneamente, mais baratos com um único drone que pratica o processamento da localização a tempo real. A desvantagem desta solução encontra-se na ainda reduzida autonomia energética do drone, que consequentemente limita a operação dos robôs terrestres.

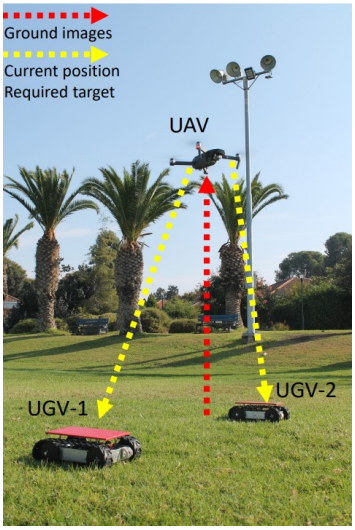


Figura 1. Configuração do sistema de (Edlerman & Linker, 2019).

Segue-se a seguir a descrição de vários projetos com variedades e aperfeiçoamentos de abordagens mais genéricas utilizadas na navegação autónoma robótica, especificamente no setor agrícola.

2.2.1. Navegação por sistema global de satélites

O GPS tornou-se essencial na vida humana na função de orientação, navegação e posicionamento, tanto no quotidiano como na agricultura. Um sistema assim oferece posicionamento absoluto, contudo o sinal em pomares pode ser distorcido ou bloqueado pelas copas densas das árvores (Bechar & Vigneault, 2016).

Em 2016, foram iniciados os serviços iniciais de um sistema global de navegação por satélite (GNSS) europeu, Galileo. Este sistema difere do GPS e até do sistema russo, o GLONASS, fornecendo extrema precisão e cronometria e contará com 24 satélites operacionais à volta do globo. O Galileo é autónomo e interoperável com os outros sistemas de navegação existentes, combinando a informação de vários satélites para uma precisão ideal. Este projeto europeu está sobre controlo civil, ao invés dos outros sistemas que são operados pelos militares (European Global Navigation Satellite Systems Agency, n.d.).

Para se utilizar um serviço deste género, é necessário determinar o nível de precisão do sistema em causa, segundo estimativas estatísticas como a raiz quadrada do erro médio (RMSE – *Root Mean Square Error*) e erro circular provável (CEP – *Circular Error Probable*), para o posicionamento bidimensional (Santra et al., 2019). Recetores de precisão podem ser também utilizados, implementando hardware RTK ou GPS diferencial no terreno, reduzindo o erro de metros provocado por um GPS comum para centímetros.

Khan et al. (2018) projetaram um robô terrestre, intitulado de AgBot, para aplicação de herbicida, orientando-se no terreno autonomamente. O sistema de navegação do AgBot é feito via GPS, controlado por um Navio 2.0 acoplado a um Raspberry Pi 3. Este equipamento funciona com software Mission Planner da ArduPilot Dev Team, que permite traçar caminhos segundo pontos GPS, como representa a Figura 2.

Figura 2. Trajeto criado no software Mission Planner da ArduPilot Dev Team (Khan et al., 2018).

Para auxiliar o sistema GPS, no caso de falha de sinal ou de falta de precisão do sinal, o robô priorizará um sistema de posicionamento cinemático em tempo real, para corrigir qualquer erro de localização efetuado pelo GPS. O robô recebe o feedback deste sistema secundário via sinal rádio e as correções são efetuadas por uma conexão Wi-Fi.

Zaidner & Shapiro (2016) pretendiam uma plataforma robótica terrestre que se movesse autonomamente, com uma boa precisão e com sistemas de baixo custo. Decidiram por optar num sistema híbrido: uma combinação de um sistema GPS com sensores internos. O controlo é realizado por quatro sistemas: sinal GPS, um sistema de navegação inercial auxiliar com acelerómetros e giroscópios, sistema visual de odometria para computar os obstáculos captados pela câmara e odometria de roda com encoders. A Figura 3 mostra o modelo cinemático da plataforma robótica e a plataforma em si.

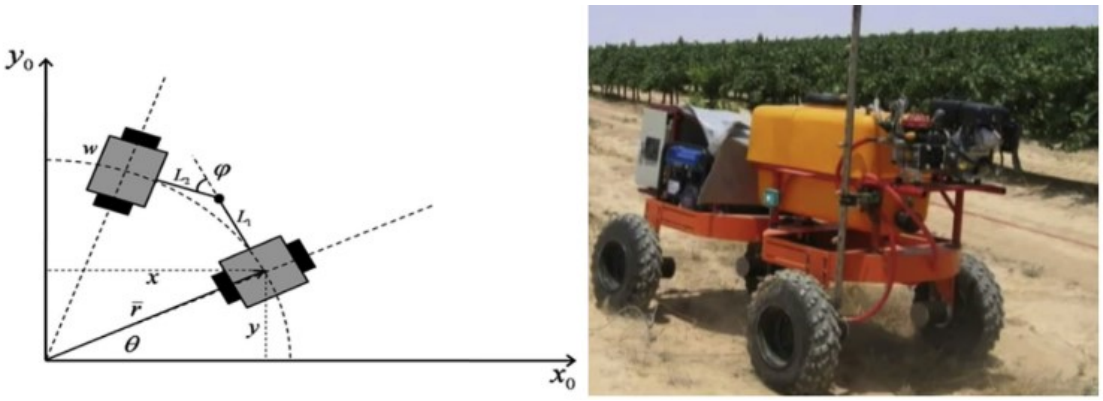


Figura 3. Modelo cinemático da plataforma robótica (esquerda) e a plataforma (direita) (Zaidner & Shapiro, 2016).

Zhang et al. (2016) desenvolveram um projeto onde dois tratores conseguiam operar simultaneamente no mesmo terreno, sem colidir ou interferir entre eles. Uma combinação de componentes foi necessária para o bom funcionamento destes robôs: um RTK GPS e uma unidade de medição inercial eram os componentes principais de navegação. Um PDA recebe o sinal de correção para o RTK GPS por um serviço GPS privado. Os robôs comunicam entre si por sinal Bluetooth, e um computador utilizado como controlador – a unidade de controlo do motor comunica com o computador através de uma rede CAN-BUS.

2.2.2. Navegação e mapeamento recorrendo a LiDAR

A tecnologia LiDAR determina distâncias aos objetos recorrendo a laser, calculando o tempo que a luz refletida do laser demora a chegar ao recetor. Já muita aplicada na robótica, esta tecnologia é muito vantajosa, pois não é afetada pelas variações de luminosidade – problema comum em ambientes exteriores, como pomares, vinhas e quintas. Esta tecnologia deteta todos os objetos ao seu alcance, permitindo a identificação individual de estruturas, se for isso o pretendido (Hiremath et al., 2014).

Esta tecnologia é muito utilizada na robótica em aplicações agrícolas. Permite a navegação autónoma, segundo a informação que o laser oferece. Estimativas como curso, desvio lateral, distância dos limites das filas dos pomares e no fim destas são um exemplo da informação que se consegue obter. Podem ser programados algoritmos de reconhecimento de certos objetos, como ervas, árvores, paus e folhas que auxiliam na navegação ou para outros objetivos (Hiremath et al., 2014).

Bayar et al. (2015) rejeitam a ideia da utilização de um sistema de navegação por GPS, devido aos altos custos, limitando-se a componentes mais acessíveis, como um sensor laser de duas dimensões e encoders nas rodas e volante. Sabendo que confiar somente em sensores internos de um robô aplicado na agricultura não é fiável, esta equipa concebeu um modelo para evitar erros de navegação e localização: adicionaram um termo harmónico ao modelo para prevenir as derrapagens das rodas, e demonstraram a estabilidade do sistema segundo o método Lyapunov. Programaram as viragens de modo a aumentar a visão do veículo para a fila destino, torná-las mais naturais, e limitar a razão de viragem. Todas estas funções são auxiliadas e apoiadas simplesmente pelos componentes já referidos, tornando este robô economicamente mais acessível.

Marden & Whitty (2014) projetaram um robô que se move autonomamente numa vinha, segundo técnicas de visão computacional. A localização e o mapeamento funcionam com recurso a linhas extraídas, com recurso ao método RANSAC (*RANdom SAmple Consensus*), de um LiDAR planar montado na frente do veículo. Depois destas linhas serem extraídas, são parametrizadas em coordenadas polares de raio e ângulo, assumidas infinitamente longas. Na Figura 4 é apresentada uma visão panorâmica dos obstáculos detetados pelo laser e esclarecimento da irregularidade da distribuição dos objetos desta solução tecnológica.

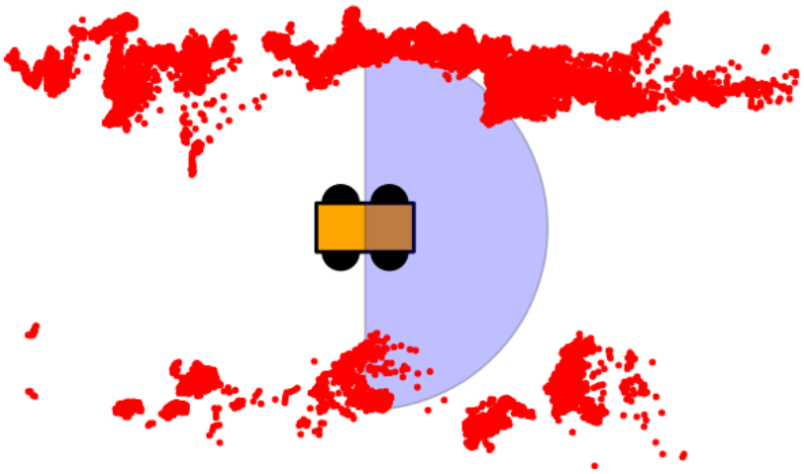


Figura 4. Visão panorâmica dos obstáculos detetados pelo laser e esclarecimento da irregularidade da distribuição dos objetos (Marden & Whitty, 2014).

Durante a navegação, o robô regista os dados e cria um mapa com as suas estruturas em tempo real, numa rede com filtro de Kalman, estendido para localização e mapeamento simultâneo EKF SLAM (*Extended Kalman Filter – Simultaneous Localization And Mapping*). Apesar do sensor ser planar, a localização e orientação do robô tem que ser considerada em três dimensões, devido ao terreno na vinha variar na altura e inclinação. Os parâmetros das filas foram registados, assumindo que as filas na vinha que integravam o algoritmo SLAM eram exatamente retas.

Zack (2015) formulou uma solução de navegação de uma plataforma robótica de quatro rodas, incorporando um sistema LiDAR 2D e encoders nas quatro rodas de modo a obter-se uma nuvem de pontos sendo esta a base do algoritmo.

Mooney & Johnson (2014) desenvolveram uma representação 3D de um pomar, recorrendo a um sensor LiDAR planar para reconhecer as árvores. As árvores, no algoritmo desenvolvido, são segmentadas e, de seguida, caraterizadas segundo a sua aparência. Desta maneira cria-se um banco de dados que categoriza as árvores formando um mapa com descrição de blocos, filas e colunas. Este método traz vantagens, como a organização e eficácia da deslocação e identificação de certas árvores ou blocos específicos, e a localização do robô nunca será uma incógnita. Na Figura 5 é apresentada uma vista panorâmica dos pontos guardados pelo LiDAR com escala de cor relativa à elevação do terreno, providenciada, por esta solução tecnológica.

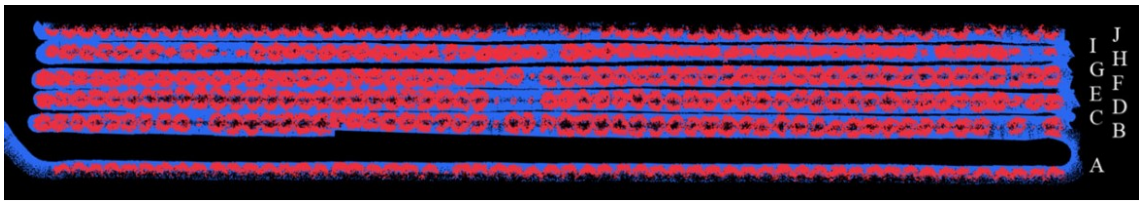


Figura 5. Vista panorâmica dos ponto guardados pelo LiDAR com escala de cor relativa à elevação do terreno (Underwood et al., 2015).

Rejas et al. (2015) implementaram um sensor 2D montado numa plataforma robótica rotativa que emitia o feixe laser e recebia a informação refletida deste, determinando as distâncias e objetos de interesse. O sistema era um Hokuyo 30LX LiDAR rodando dentro dum ângulo de 270º e um alcance de deteção de 0,1 a 30 metros. A velocidade de rotação era ajustada consoante a densidade de obstáculos ou a velocidade de deslocação do robô.

Yandún Narváez et al. (2016) principiaram uma solução para obter uma reconstrução tridimensional de um pomar para análise do seu comportamento térmico. Esta é feita recorrendo à con-

vergência de dados recolhidos por um LiDAR e imagens térmicas capturadas, sem recurso do tradicional sinal de navegação por satélite. Na Figura 6 encontra-se uma amostra da reconstrução tridimensional gráfica de uma linha de árvores.

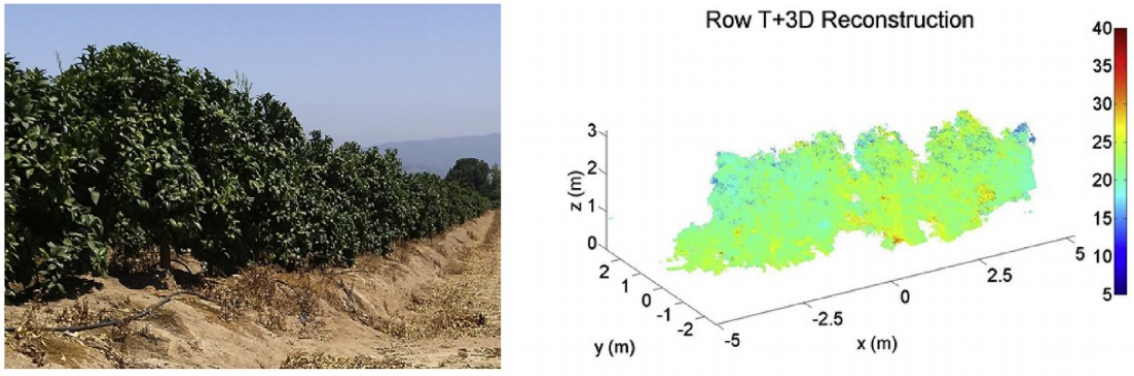


Figura 6. Amostra de reconstrução de uma linha de árvores (Yandún Narváez et al., 2016).

Underwood et al. (2016) apresentaram uma plataforma robótica terrestre que mapeia um pomar de amendoeiras. O sistema possui uma câmara e um sensor LiDAR. A câmara tem o propósito de adquirir informação fotográfica da condição das árvores e dos frutos para uma previsão de produção. O sensor LiDAR capta a informação geométrica do redor do robô e cria um mapa virtual identificando cada árvore individualmente.

2.2.3. Visão computacional

A visão computacional é um sistema altamente versátil, com muito potencial na navegação robótica em ambientes agrícolas. A boa programação de um algoritmo destes permite a medição, deteção e caraterização do pretendido. Estes sistemas são relativamente baratos, no entanto não são suficientes e é comummente necessário combiná-los com outros conjuntos de navegação. Na Figura 7 são apresentadas cinco fotografias tirados ao mesmo plano em horas diferentes do dia, o que mostra que os níveis de luminosidade interferem com a saturação e tonalidade do meio envolvente, o que se configura como uma desvantagem da visão robótica em aplicações no exterior (Bechar & Vigneault, 2016).

Figura 7. Fotografias tiradas ao mesmo plano em horas diferentes do dia (Bechar & Vigneault, 2016).

Chen et al. (2018) pensaram num método de deteção dos troncos das árvores misto, baseado em cor, textura e contorno, com a estrutura apresentada na Figura 8. Como sistema de apoio, para reduzir a possibilidade de erros, recorreram a sensores ultrassónicos que medem a distância entre a plataforma robótica e o tronco da árvore. Esta medição de distância apenas é processada quando o tronco é reconhecido.

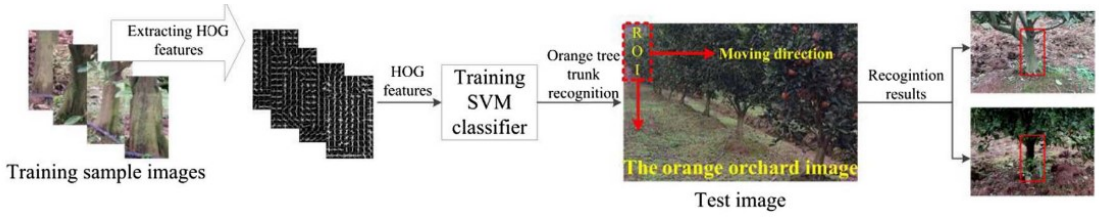


Figura 8. Sistema de reconhecimento dos troncos das árvores (Chen et al., 2018).

Shalal et al. (2013, 2015a) contribuíram com um algoritmo de deteção dos troncos das árvores recorrendo à fusão de dados recolhidos por câmaras monetariamente acessíveis e por sensores a laser. Esta combinação serve para detetar possíveis troncos das árvores e distinguir objetos com caraterísticas semelhantes à das árvores, de modo a fazer uma boa análise local. O sensor laser fornece distâncias, ângulos e largura dos troncos das árvores, ou de objetos que assimila a árvores,

e o vídeo da câmara vai distinguir os objetos não pretendidos das árvores segundo certas características, como a cor e limites. Na Figura 9 é apresentada uma imagem captada em configuração RGB, transformada em escalas de cinzento e configurado o limite da árvore dentro da região retangular de interesse.

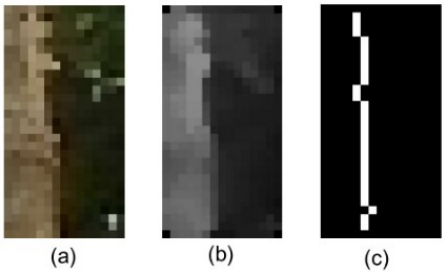


Figura 9. Em (a) representa-se a imagem captada em configuração RGB, em (b) é transformada em escalas de cinzento e em (c) é configurado o limite da árvore dentro da região retangular de interesse (Shalal et al., 2013).

Shalal et al. (2015a) armazenaram os dados recolhidos de maneira a obter-se um mapa, segundo o algoritmo EKF. Este mapa, exposto na Figura 10, mostrou-se consistente com a organização do pomar testado e, ao invés de registar filas únicas a separar os corredores, os sensores identificaram as árvores individualmente. Esta configuração permite que o robô possa ter outras funções que necessitem da identificação de árvores, como colheita, poda ou inspeção.

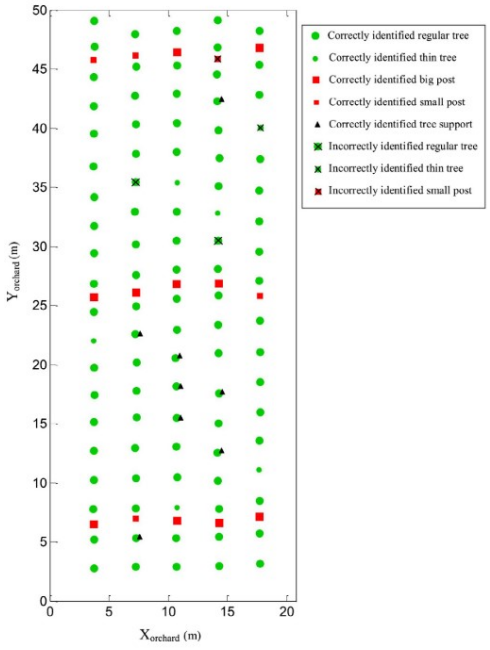


Figura 10. Mapa projetado com os objetos de interesse categorizados (Shalal et al., 2015a).

Ramos & Gaspar (2019) elaboraram um algoritmo de reconhecimento, em MATLAB, da navegação em filas de pomares baseado no processamento de imagem. O algoritmo foi dividido em sete partes: (1) conversão da imagem em escalas de cinzento; (2) imagem original é também transformada num formato HSV, de modo a avaliar os níveis de saturação; (3) o algoritmo aplica o método Otsu, com o objetivo de realçar as formas desejadas; (4) operação de morfologia, para eliminar elementos indesejados da imagem; (5) o algoritmo utiliza um operador Sobel para detetar os limites; (6) fim do processo de reconhecimento e aplica-se a transformada de Hough para identificar as linhas na imagem binária; (7) obtém-se o caminho de navegação traçado por entre as linhas do campo agrícola, conforme exposta na Figura 11.

Figura 11. As linhas azuis são as linhas limite detetadas através da transformada de Hough e a linha preta o trajeto obtido (Ramos & Gaspar, 2019).

Zhuang et al. (2019) conceberam um algoritmo capaz de compensar os níveis de iluminação de imagens captadas em pomares de lichias com diferentes condições de luz. Após essa correção na imagem, outro algoritmo distingue o fruto e o pedúnculo baseado nos níveis de cor. Dado que a lichia é um fruto extraído juntamente com o pedúnculo, outro algoritmo é culminado para identificar o ponto de corte para uma colheita automática. As imagens foram capturadas com câmaras RGB, destas imagens foram construídos histogramas a mostrar a distribuição da intensidade nos espaços de cor HSV.

2.2.4. Mapeamento

Os robôs autónomos aplicados na agricultura são empregues em locais desconhecidos por eles, daí o vasto número de investigações realizadas relativamente à navegação autónoma em terrenos irregulares e com possíveis obstáculos.

Muitos sistemas de navegação para estes robôs específicos baseiam-se no mapeamento do terreno. Dentro desta abordagem, o processo SLAM (Localização e Mapeamento Simultâneos) é o mais utilizado, que consiste na localização própria do robô num ambiente incógnito e, simultaneamente, com recurso a sensores externos, a informação do seu redor é captada e guardada na memória do robô, construindo um mapa digitalmente (Yousif et al., 2015).

Este processo de mapeamento apresenta bons resultados em ambientes de trabalho planos sem grandes alterações do layout base, como por exemplo, num armazém. No entanto, num pomar ou vinha, o terreno é extremamente irregular e aleatório – pedras, ervas, folhas entre outros agentes naturais entram na base de dados do robô e podem fornecer informação falaciosa que podem pôr em risco o bom funcionamento deste numa outra passagem nos pontos já identificados. Daí a necessidade de métodos e técnicas para aperfeiçoar ou criar um processo SLAM modificado e adaptado para aplicações agrícolas. O mapeamento Hector é um processo SLAM otimizado para navegação em corredores estreitos e ambientes complexos. Este algoritmo não recorre à odometria. Como sistema de navegação e localização, varre a área com os seus sensores e combina informações, juntamente com um filtro Gauss-Newton, para definir a posição do robô com base na distribuição de partículas. Este sistema cria várias camadas do mesmo mapa e guarda-as num servidor, definindo assim os mapas e as trajetórias (Lepej & Rakun, 2016).

Atualmente, o algoritmo SLAM pode ser aperfeiçoado aos ambientes agrícolas recorrendo a diferentes filtros. O Filtro de Kalman Estendido (EKF) é o mais comum nesta área: quando este é implementado, o vetor do sistema é composto por ambas a orientação do veículo e os parâmetros que definem as características do ambiente – este algoritmo pode, por exemplo, utilizar exclusivamente linhas, logo vai adquirir apenas informação geométrica do ambiente reconhecendo as linhas que as árvores fazem e a linha do corredor que deve percorrer. Outros filtros como o Filtro de Kalman sem odometria (UKF – *Unscented Kalman Filter*) que opera melhor comparativamente ao referido anteriormente em condições não lineares; o Filtro de Informação Estendida (IEF – *Extended Information Filter*) que aperfeiçoa o tempo de processamento na fase de correção do algoritmo SLAM, fazendo deste uma melhor opção em aplicações a tempo real; e o filtro de partículas que não está restringido aos processos Gaussianos e gere melhor as não linearidades no processo estimativo, no entanto a sua implementação a tempo real continua ainda restritiva (Auat Cheein et al., 2011).

Cernicchiaro et al. (2019) desenvolveram um código para uma plataforma robótica com o propósito de apanhar pêssegos caídos no chão, movida autonomamente segundo um mapa já atribuído. Quando o robô recebe informação que a sua bateria está com um nível baixo, este tem que definir o trajeto mais curto para voltar à estação de troca de baterias. A técnica utilizada foi o algoritmo D* desenvolvido em MATLAB, o qual, dependendo da posição do robô no mapa, avalia-o por completo e analisa a distância dos pontos ao destino. Após a análise, o robô percorre o trajeto definido. Na Figura 12 é apresentado um cenário em que o robô volta à estação de carregamento após indicação de baixo nível de bateria.

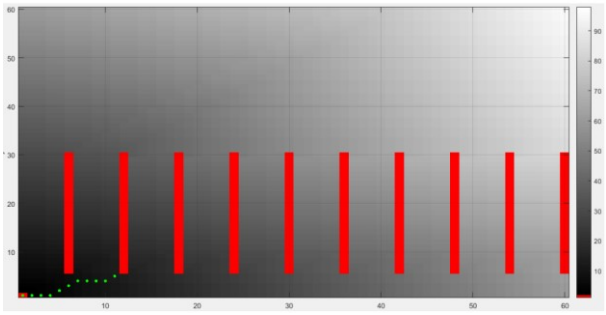


Figura 12. Cenário em que o robô volta à estação de carregamento após indicação de nível baixo de bateria (Cernicchiaro et al., 2019).

Blok et al. (2019) avaliaram a precisão de navegação e robustez de dois algoritmos de localização probabilística de um robô autónomo em pomares. Os algoritmos utilizados foram o filtro de Kalman e o filtro de partículas. O filtro de Kalman utiliza distribuições Gaussianas para se localizar, enquanto que o filtro de partículas recorre a amostras aleatórias com várias partículas para estimar a sua posição. Para recolher a informação necessária, o dispositivo utilizado foi um sensor LiDAR bidimensional. Concluíram que, relativamente à precisão e robustez de navegação, o filtro de partículas apresentava melhores resultados que o filtro de Kalman. A sua desvantagem perante o outro algoritmo era que o primeiro necessita de mais informação sobre o pomar a-priori e era mais intensivo computacionalmente.

2.2.5. Machine Learning

Atualmente, o *smart farming* está fortemente presente no ramo agrícola, abordando as várias soluções para aperfeiçoamento da produtividade, impacte ambiental, segurança dos alimentos e sustentabilidade. As novas tecnologias de informação e comunicação (TIC) continuamente desenvolvidas (a sensorização remota, a Internet das Coisas, computação em nuvem e análise *Big Data*) facilitam a medição, monitorização, e análise de vários fenómenos complexos, variáveis e imprevisíveis por contexto, situação e localização. Grande parte dos dados adquiridos pelas tecnologias de sensorização remota e da Internet das Coisas envolvem imagens. A análise destas imagens capturadas é importante no progresso de técnicas de identificação, classificação e deteção (Kamilaris & Prenafeta-Boldú, 2018).

Aprendizagem Máquina (*Machine Learning*), num modo simples, constatado por Samuel (1988), é a área de estudo que oferece, ao computador, a habilidade de aprender sem ser especificamente programado, ou seja, de programar um computador para que este possa aprender somente com os dados que lhe são fornecidos, ao contrário de outrora que, para se construir um algoritmo de deteção ou reconhecimento de algo, era necessário especificar uma lista de regras, podendo esta

ser curta ou muito extensa. Este método mais trivial ocupava muito tempo na definição das regras e de uma constante atualização de novas regras ou exceções que, eventualmente, surgem.

As técnicas mais comuns utilizadas, de *Machine Learning*, para análise de imagem incluem k-means, máquina de vetores de suporte (SVM – *Support Vector Machines*) e redes neuronais artificiais (ANN – *Artificial Neural Networks*), filtros *Wavelet*, índices de vegetação e análise de regressão. A aprendizagem profunda, ou *Deep Learning*, é uma abordagem moderna pertencente à área de *Machine Learning*, semelhante às ANN. A aprendizagem profunda constitui uma rede neuronal mais profunda que oferece uma representação hierárquica dos dados segundo várias convoluções – isto permite melhores capacidades de aprendizagem relativamente à captura da complexidade total da tarefa incumbida (Kamilaris & Prenafeta-Boldú, 2018).

Há vários tipos de algoritmos de *Machine Learning* que podem ser classificados segundo vários fatores, tais como: se são ou não treinados com supervisão humana, se têm capacidade de aprender incrementalmente, ou se funcionam comparando novos dados com dados reconhecidos ou se deteta padrões nos dados de treino e cria um modelo preditivo (Géron, 2019).

As tarefas do *Machine Learning* descrevem os termos de como um algoritmo processa uma coleção de caraterísticas medidas quantitativamente de algum objeto ou evento, sendo mais facilmente representada por um vetor onde cada elemento é uma caraterística, por exemplo os valores dos pixéis de uma imagem (Goodfellow et al., 2016).

A lista seguinte define e carateriza alguns tipos de *Machine Learning*:

- **Aprendizagem supervisionada:** Os algoritmos de aprendizagem supervisionada aprendem a associar um input com um output, dada uma série de exemplos deles. Estes exemplos são fornecidos, ou supervisionados, por ação humana (Goodfellow et al., 2016).

  Um modelo de aprendizagem supervisionada contém dados de treino com exemplos dos vetores de input juntamente com os vetores alvos correspondentes. Estes modelos podem ser de classificação, em que o objetivo é atribuir cada vetor de input a uma das categorias discretas de número finito; podem também ser de regressão, se o output pretendido consiste em uma ou mais variáveis contínuas (Bishop, 2006).

  Um exemplo da classificação é um filtro de e-mails de spam. Este é treinado com vários exemplos de e-mails com a sua classe atribuída (rótulo), podendo treinar assim, o modelo de classificação. Uma representação gráfica pode-se encontrar na Figura 13.

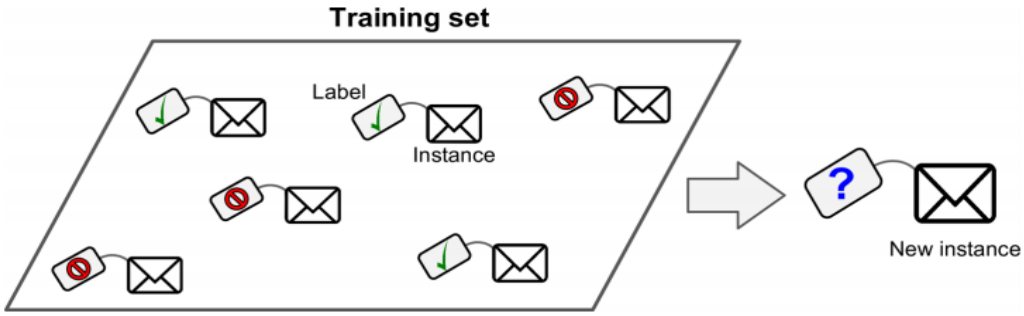


Figura 13. Base de dados rotulada para uma aprendizagem supervisionada (Géron, 2019).

Um exemplo da regressão é um modelo de previsão do preço de um carro (alvo), dada uma série de caraterísticas como quilometragem, marca e idade (preditores). Para treinar um modelo deste género é necessário fornecer muitos exemplos de alvos, incluindo os seus preditores e rótulos.

Alguns dos algoritmos mais importantes são (Géron, 2019):

- K-vizinhos mais próximos;
- Regressão linear e logística;
- Máquinas de Vetor Suporte;
- Árvores de decisão;
- Redes Neuronais Artificiais.

Para além dos modelos de classificação e regressão, existem algumas variantes mais exóticas utilizadas com frequência (Chollet, 2017):

- Geração de sequência: Dada uma imagem, prevê-se a legenda que a descreve.
- Previsão por árvore de síntese: Dada uma frase, prevê-se a sua decomposição numa árvore de síntese.
- Deteção de objetos: Dada uma imagem, cria-se uma caixa que delimita os objetos a identificar.
- Segmentação de imagem: Dada uma imagem, cria-se uma máscara que delimita um objeto específico.

- **Aprendizagem não supervisionada:** Um modelo de aprendizagem não supervisionada, ao contrário da supervisionada, vem sem rótulos ou respostas. A máquina ao analisar os dados pode encontrar padrões e alguma estrutura. As duas tarefas mais populares de aprendizagem não supervisionada são a análise de agrupamento de dados (*clustering*), e redução de dimensionalidade (Shukla, 2018). Este ramo de *Machine Learning* consiste na procura de transformações dos dados de input sem a assistência de quaisquer alvos, com o objetivo de visualização, compressão, redução de ruído ou compreensão da correlação da base de dados (Chollet, 2017).

  Um algoritmo de aprendizagem não supervisionada aprende propriedades úteis da estrutura segundo uma base de dados que contém diversas caraterísticas. Este algoritmo é útil na aprendizagem da distribuição probabilística da base de dados, seja explicitamente como uma estimativa de densidade ou implicitamente para tarefas como síntese ou redução de ruído (Goodfellow et al., 2016).

  O processo de *clustering* divide os dados em grupos individuais com caraterísticas semelhantes – classifica os dados sem conhecer os rótulos correspondentes. Os algoritmos mais famosos de *clustering* são os baseados em K-means, agrupamento hierárquico e maximização de expectativa (Géron, 2019; Shukla, 2018). Na Figura 14 encontra-se uma representação visual do que o algoritmo de *clustering* efetua.

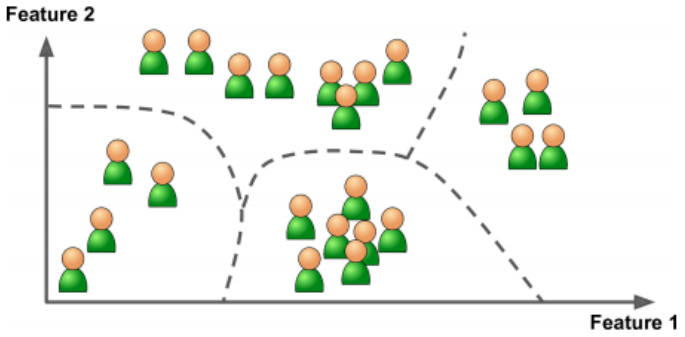


Figura 14. Clustering (Géron, 2019).

A redução de dimensionalidade manipula os dados para os visualizar numa perspetiva mais simples, responsabilizando-se em descartar caraterísticas mais redundantes e priorizando outras. Os algoritmos mais populares são a análise de componentes principais Kernel, incorporação localmente linear e incorporação de vizinhos estocásticos com distribuição t, sendo atualmente mais recorrente a implementação de autoencoders para esta tarefa (Géron, 2019; Shukla, 2018).

- **Aprendizagem por reforço:** Ao contrário das aprendizagens supervisionadas e não supervisionadas, na aprendizagem por reforço não há presença de um mentor, sendo este o seu ambiente ou redor. O sistema de aprendizagem recebe retropropagação (*feedback*) das suas ações, sem garantia se são as corretas ou não (Shukla, 2018).

  Na aprendizagem por reforço, o modelo recolhe a informação das reações do ambiente às ações tomadas pelo agente a ser treinado. Após recolhidos os dados suficientes, o modelo cria a solução mais adequada à situação. A Figura 15 demonstra o comportamento de um robô com um algoritmo de aprendizagem por reforço incorporado.

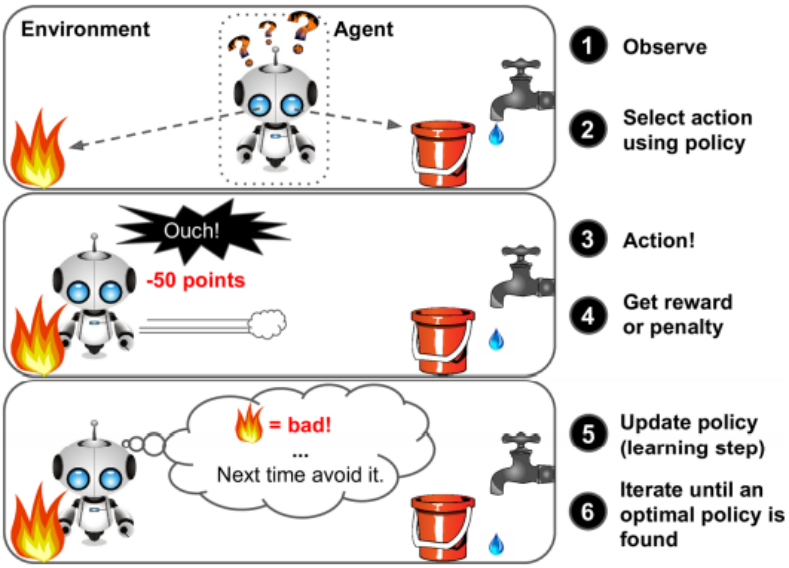


Figura 15. Aprendizagem por reforço (Géron, 2019).

Este ramo de aprendizagem é pouco frequente utilizado em hardware robótico, pois a veracidade física das ações incorretas pode danificar fisicamente o agente. Deste modo, algoritmos deste tipo são aplicados mais frequente a nível de software, tal como videojogos, ou em hardwares cujo ambiente de ação não seja nocivo à integridade física do robô (Chollet, 2017).

2.2.5.1. Redes Neuronais Artificiais

Uma rede neuronal artificial é um sistema de processamento computacional inspirado nos sistemas neuronais biológicos, como o cérebro humano. Tendo o cérebro neurónios, as redes neuro-

nais artificiais possuem, também, uma série de percetrões computacionais interconectados distribuídos que comunicam e aprendem uns com os outros de maneira a otimizar o resultado do output (Véstias, 2021).

A estrutura de uma rede neuronal artificial é constituída, geralmente, por um vetor de input multidimensional aplicado numa camada, para depois ser distribuído pelas várias camadas ocultas. As camadas ocultas tomam decisões baseadas na informação recebida das anteriores e avaliam as mudanças estocásticas caso estas fossem aperfeiçoar ou deteriorar o resultado do output. Este processo de avaliação, deteção, ou previsão do erro é chamado de aprendizagem (O'Shea & Nash, 2015). A Figura 16 representa, de maneira simplificada, a estrutura de uma rede neuronal artificial.

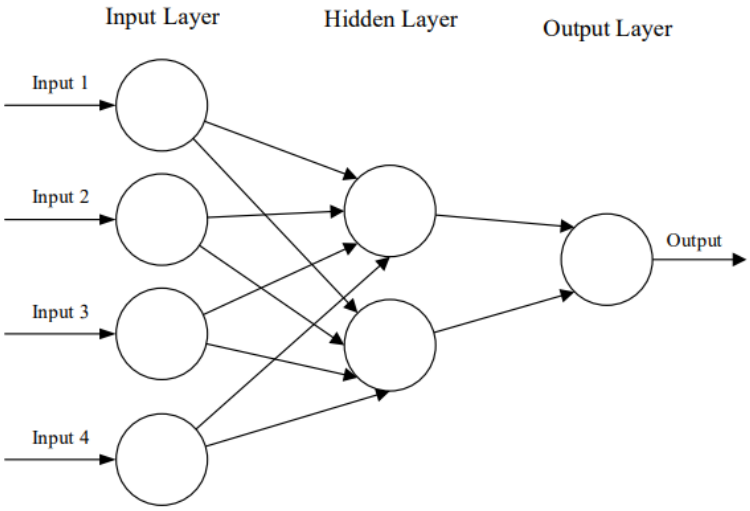


Figura 16. Rede neuronal artificial, de três camadas. Possui (da esquerda para a direita) uma camada de input, oculta e de output. Cada ligação, ou seta, contém um valor, chamado de peso (O'Shea & Nash, 2015).

Existem elementos comuns e importantes presentes nas redes neuronais artificiais, como os pesos e viés, e as funções de ativação, que foram criadas especificamente para redes neuronais. Segue-se uma explicação para esses elementos, feita por Scarpino (2018).

A cada sinal entre percetrões é atribuído um peso (*weight*) – um valor real que indica a sua influência. Os valores dos pesos são multiplicados aos inputs, sendo o output a soma de todos os produtos dos inputs com os pesos atribuídos. Os valores dos pesos são atualizados durante a execução do treino, para aperfeiçoar e adaptar o modelo.

Numa rede neuronal, as somas dos produtos dos inputs com os pesos não podem exceder um valor limite definido antes do treino, caso isso aconteça, o resultado desta função retorna o valor

de zero. De maneira a evitar estas situações, os limites são substituídos introduzindo, no modelo, viés (*bias*). Estes elementos não passam de inputs comuns, com um peso atribuído, geralmente com o valor igual a +1, que entrarão na função da soma dos produtos dos inputs com os pesos.

As funções de ativação aceitam um tensor com valores, dando como resultado outro tensor contendo os valores de output. As funções de ativação mais comuns aplicadas neste ramo são as retificadoras e classificadoras. Uma função retificadora (*Rectified Linear Unit function* – ReLU) tem um funcionamento bastante intuitivo: devolve o input se este for positivo ou devolve 0 se for negativo. Uma função classificadora tem a mesma representação que uma função logística (sigmóide), que possui propriedades que permitem classificar pontos em categorias.

2.2.5.2. Redes Neuronais Convolucionais

Uma rede neuronal convolucional (*Convolutional Neural Networks* - CNN) é um ramo das redes neuronais artificiais (O'Shea & Nash, 2015). São compostas por percetrões interligados que praticam o processo de otimização e de aprendizagem, tal como qualquer outra rede neuronal artificial. A rede na íntegra é função dos valores de input, que serão avaliados pelas várias camadas, e dos valores de peso que ditarão a dependência de cada percetrão transato que se juntará à equação pertencente do da próxima camada. A grande diferença das CNN para as outras redes neuronais é que esta primeira é utilizada maioritariamente no campo de reconhecimento de padrões em imagens.

A arquitetura das CNN é muito semelhante com uma rede neuronal artificial, apenas com algumas diferenças. Uma delas é que os percetrões possuem três dimensões: a dimensão espacial (a largura e comprimento da imagem) e a profundidade. A profundidade não se refere ao número total de camadas na rede, mas sim à terceira dimensão de um volume de ativação - ao contrário das outras redes, os percetrões de qualquer camada só conectarão a uma região pequena da camada que se procede (O'Shea & Nash, 2015).

Existem disponíveis grandes conjuntos de dados diferentes para abordar o problema. Estas consistem em várias camadas convolucionais, conjugadas e/ou completamente conectadas. As camadas convolucionais atuam como extratores de caraterísticas das imagens de entrada, cuja dimensionalidade é reduzida pelas camadas conjugadas, enquanto as camadas completamente conectadas funcionam como classificadoras. Tipicamente, na última camada, as camadas completamente conectadas exploram as caraterísticas já agudamente aprendidas, de modo a classificar a imagem de entrada em classes predefinidas (Canziani et al., 2016). A camada de *pooling* executa uma redução da resolução na dimensionalidade do input, reduzindo o número de parâmetros na ativação (Ciresan et al., 2011). Na Figura 17 é apresentada a arquitetura típica das CNN.

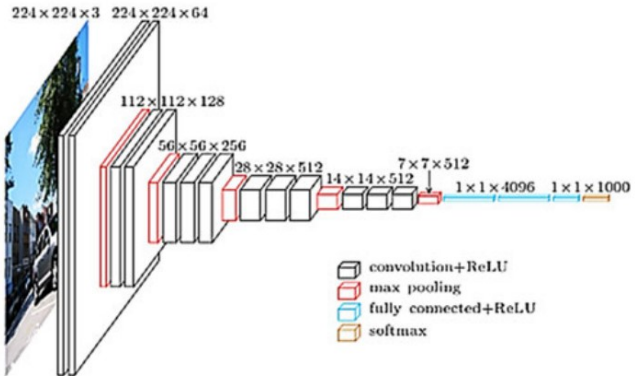


Figura 17. Representação da arquitetura das CNN (Kamilaris & Prenafeta-Boldú, 2018).

Fricker et al. (2019) recorreram à utilização de um classificador com CNNs para mapeamento autónomo de sete espécies de árvores, segundo imagens de alta resolução aéreas hiper-espectrais. As imagens foram adquiridas com uma plataforma de observação aérea da NEON, na região sul da Califórnia. Neste projeto compararam a identificação das árvores do método CNN com um método de imagem mais convencional, o RGB, chegando à conclusão de que o método CNN apresentava uma classificação bem-sucedida de 87%, superior à do método RGB que foi de 64%. A Figura 18 apresenta as espécies de árvores detetadas (cores) após o voo do drone, recorrendo ao classificador com CNNs.

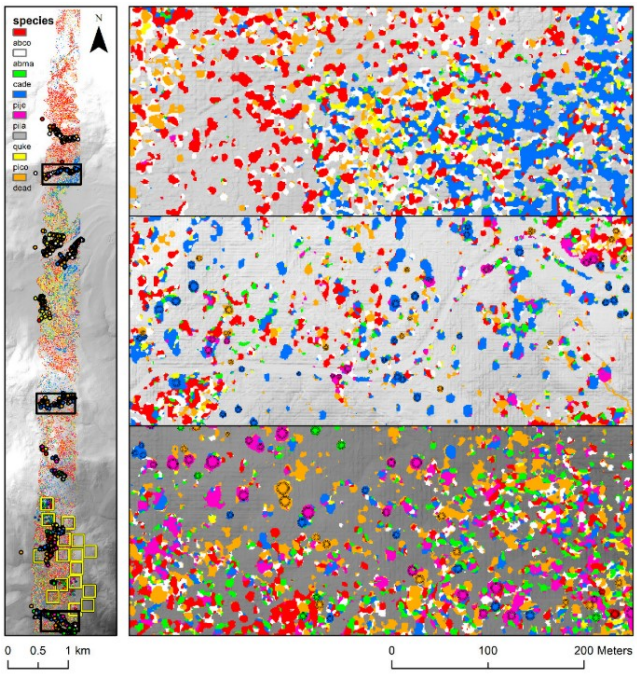


Figura 18. Espécies de árvores detetadas (cores) após o voo do drone, recorrendo ao classificador com CNNs (Fricker et al., 2019).

Csillik et al. (2018) estudaram um caso simples de deteção de árvores de citrinos segunda imagens captadas por um drone, combinando CNNs com análise de objetos pós-processados. Na classificação final, foram detetadas 3015 árvores individuais, comparativamente às 2912 árvores de referência.

O objetivo do estudo desenvolvido por Majeed et al. (2018) foi segmentar o tronco e os ramos de macieiras jovens, para treino autónomo num pomar. Um sensor Kinect V2 coletou pontos 3D e imagens coloridas (os objetos de fundo foram filtrados). Estes dados coletados foram processados, em MATLAB, numa CNN, para serem segmentados em troncos e ramos. Foram apurados bons resultados, que confirmam que os ramos e troncos, apesar de terem cores semelhantes, conseguem ser segmentados utilizando técnicas de aprendizagem profunda – neste caso SegNet.

Liu et al. (2016) desenvolveram um projeto, baseado em redes neuronais, que deteta maçãs nas árvores em operações noturnas. Durante a noite, a baixa luminosidade é a maior barreira para identificação de qualquer componente – a tonalidade dos objetos é mais reduzida, tornando estes muito semelhantes e indistinguíveis. O primeiro passo foi adquirir imagens das maçãs com uma luz artificial incidente. O segundo passo foi aplicar um algoritmo para reconhecimento das bordas das maçãs, com base nas diferentes regiões de cor dos outros objetos. A Figura 19 mostra a região de pixéis a delinear os limites do fruto.

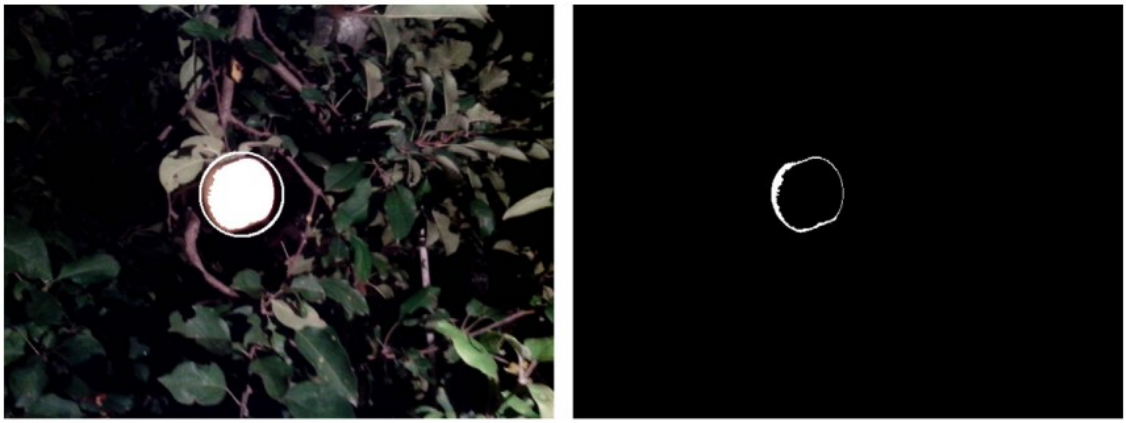


Figura 19. Classificação da região de pixéis em torno de uma maçã (X. Liu et al., 2016).

De modo a classificar a informação de cor das imagens, foi aplicada uma rede neuronal de retro propagação (BPNN – *BackPropagation Neural Networks*). Este processo reconhece as diferentes cores e descreve os limites numa gama de pixéis.

Foi proposto um método, por Shi et al. (2019), de segmentação de uma planta inteira, dividindo esta em folhas, caule, fruto, entre outros. Este método é baseado em aprendizagem profunda combinado com imagens capturadas por um sistema de câmaras múltiplo. As imagens bidimensionais de ângulos diferentes combinam a sua informação para criar uma nuvem de pontos tridimensional que representa a planta. O processo foi uma rede de segmentação semântica baseada numa rede completamente convolucional, combinada com uma rede de segmentação instantânea baseada numa rede neural convolucional de região mascarada. Na Figura 20 encontra-se uma representação da arquitetura da CNN aplicada.

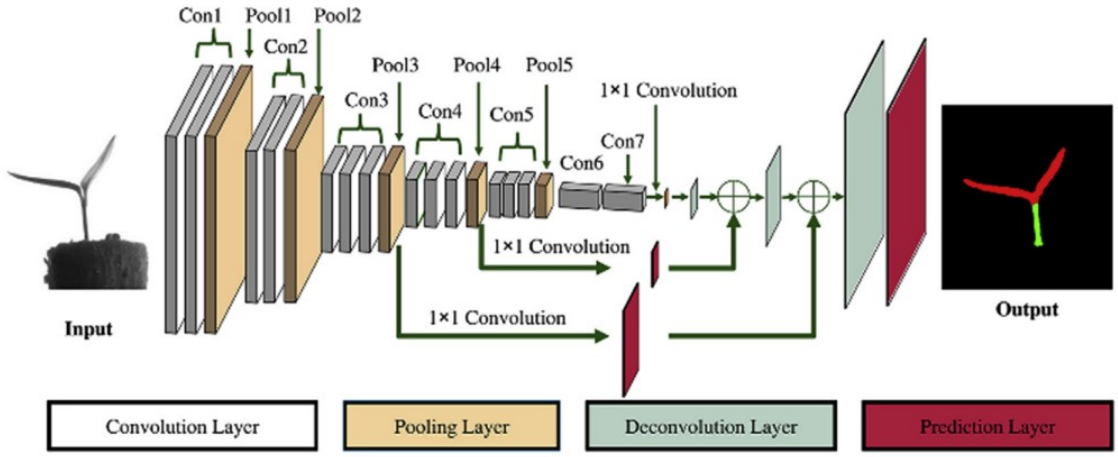


Figura 20. Arquitetura da rede completamente convolucional aplicada em (Shi et al., 2019).

2.2.6. Nota conclusiva dos sistemas de navegação autónoma

Existem vários métodos abordados para automatizar a navegação de uma plataforma robótica. Em ambientes simples, interiores e invariáveis pode-se aplicar um método único. No entanto, em ambientes agrários existem muitas variáveis que podem afetar a informação recolhida do robô, tal como terrenos irregulares, grande variedade de obstáculos e disposição aleatória destes, diferentes níveis de iluminação e condições climatéricas imprevisíveis. Uma culminação de métodos pode reduzir os erros e desenvolver uma solução. Como se pôde verificar, a maioria dos sistemas de navegação autónoma dos robôs aplicados na agricultura resulta de dois ou mais sistemas combinados entre si para fornecer ao robô todos os dados do seu envolvente.

Sensores internos como unidades de medição inerciais, giroscópios e encoders são sistemas que podem recolher informação falaciosa, pois num ambiente agrícola as irregularidades do terreno são elevadas e, por vezes, o robô pode-se deslocar ou derrapar sem os sensores conseguirem adquirir essa informação. Um sistema de sensores internos, neste setor, necessita de outra aborda-

gem principal, sendo esta apenas auxiliar. Robôs que utilizam o sinal GPS como sistema de navegação principal recorrem, geralmente, a um sistema auxiliar de laser, visão, ou interno de avaliação da proximidade, de modo a evitar colisões. Muitos sistemas que recorrem a LiDARs planares utilizam este como sistema único de navegação. Este sistema consegue captar informação do meio envolvente e criar uma nuvem de pontos que mapeia o pomar que percorre, guardando informação dos objetos e, possivelmente, categorizar estes se for necessário. A visão robótica é uma tecnologia bastante utilizada, onde as imagens de cor captadas podem ser corridas em algoritmos que classificam obstáculos pretendidos baseados na cor, textura e limites destes. Esta tecnologia pode ser combinada com redes neuronais convolucionais para melhor categorização de objetos específicos. A Tabela 1 representa, resumidamente, esta revisão bibliográfica. Classifica as investigações por sistemas utilizados de apoio à navegação; descrição dos métodos, hardware, software e algoritmos necessários para o funcionamento correto; e as tarefas alvo dos robôs.

Tabela 1. Tabela categorizada da revisao bibliográfica.

Referência	Sistemas	Métodos	Tarefa
(Edlerman & Linker, 2019)	UAV guia. UGVs com IMU	GPS e câmara no UAV. IMU no UAV e UGVs. Comunicação por rádio.	Multitarefas agrícolas
(Khan et al., 2018)	GPS e RTK-GPS	Navio 2.0 (Raspberry Pi 3). Mission Planner (Ardupilot). 2 recetores RTK no terreno conectados via Wi-Fi.	Herbicida e fertilizante
(Zaidner & Shapiro, 2016)	GPS e IMU	Combinação de sinal GPS com sistemas internos da plataforma robótica.	Pesticida
(Zhang et al., 2016)	RTK-GPS e IMU	RTK-GPS (GEONET). Tratores interligados via Bluetooth.	2 tratores agrícolas multitarefas
(Bayar et al., 2015)	Laser 2D e encoders	Dados laser processados por algoritmo de filtro de partículas. Algoritmo de prevenção de erros de derrapagem.	Navegação em pomares
(Marden & Whitty, 2014)	LiDAR 2D	Extração de linhas via RANSAC. Localização e mapeamento recorrendo ao EKF-SLAM.	Fotogrametria
(Zack, 2015)	LiDAR 2d e encoders	Nuvem de pontos captados pelos sensores criada para extração de linhas.	Multitarefas Agrícolas
(Underwood et al., 2015)	LiDAR 2D	Nuvem de pontos 3D obtida pelo LiDAR.	Reconhecimento e segmentação
(Rejas et al., 2015)	LiDAR 2D	Nuvem de pontos 3D obtida pelos LiDAR.	Mapeamento

Referência	Sistemas	Métodos	Tarefa
(Chen et al., 2018)	Visão e ultrassons	Classificação das árvores por histogramas de gradiente orientado e por máquina de vetores de suporte. Sensores ultrassónicos recolhem dados de localização.	Multitarefas agrícolas
(Shalal et al., 2013, 2015b, 2015a)	Visão e laser 2D. Algoritmo EKF-SLAM	Fusão da imagem profetada a 2D por uma câmara com a informação 3D recolhida pelo laser para deteção de árvores. Mapeamento e estimativa de posição recorrendo ao algoritmo EKF-SLAM.	Mapeamento
(Ramos & Gaspar, 2019)	Visão	Reconhecimento de um caminho baseado em imagens captadas e processadas por algoritmos, recorrendo ao software MATLAB.	Multitarefas agrícolas
(Fricker et al., 2019)	Visão e CNNs	Classificados Deep CNN aplicado em imagens panorâmicas hiper-espectrais de alta qualidade. Resultados comparads a um classificador baseado em RGB.	Mapeamento
(Csillik et al., 2018)	Visão e CNNs	Classificador CNN aplicado a imagens panorâmicas recorrendo ao software Trimble's eCognition Developer 9.3.	Deteção
(Majeed et al., 2018)	Sensor Kinect e CNNs	O sensor Kinect coleta pontos 3D e imagens RGB. Dados processados, em MATLAB, numa CNN (SegNet).	Segmentação de troncos e ramos
(Cernicchiaro et al., 2019)	Algoritmo D*	Plataforma retorna à estação de carregamento seguindo o caminho processado mais perto.	Apanha de pêssego caído no chão
(Blok et al., 2019)	LiDAR 2D	Avaliação de dois algoritmos de localização probabilística: filtro de partículas e filtro Kalman.	Multitarefas agrícolas

3. Materiais e Métodos

3.1. Software

O primeiro passo deste projeto consiste no desenvolvimento do algoritmo. Antes de qualquer montagem e ligação é necessário fundamentar o sistema com um algoritmo estável e preciso para a deteção correta dos troncos das árvores do pomar. Os próximos subcapítulos irão descrever o software, programas, aplicativos e linhas de comando necessárias para a realização da parte digital e algorítmica desta dissertação.

3.1.1. Sistema operativo e linguagem

A programação do algoritmo de deteção foi fundada em ambos os sistemas operativos *Windows 10* e *Linux* na linguagem *Python* 3.9.5.

Relativamente aos sistemas operativos, o *Windows 10* e *Linux* foram preferidos devido à facilidade de acesso a estes recursos e a compatibilidade de funcionalidades. O *Windows 10* e o *Linux* estão instalados num computador portátil e em servidor, respetivamente.

Relativamente à linguagem, *Python* é uma linguagem bastante intuitiva, ideal para diversas áreas, incluindo aprendizagem profunda, e a linguagem central de programação de *Tensorflow* (Scarpino, 2018), com plataformas open-source atualizadas especializadas em *Machine Learning* (*Tensorflow*) e visão computacional (*OpenCV*) – recursos úteis que serão explorados durante o processo desta dissertação.

O IDE (*Integrated Development Environment*) de preferência, onde foram corridos, alterados e processados os comandos, foi o *PyCharm* da *JetBrains*.

Relativamente aos recursos computacionais utilizados, as especificações dos aparelhos encontram-se na Tabela 2.

Tabela 2. Especificações dos recursos computacionais utilizados.

Especificações	**Computador Portátil**	**Servidor**
Sistema operativo	Windows 10	Linux
Unidade de processamento	AMD Ryzen 7 3700U 2.30 GHz	Intel Core i7-4790 3.60 GHz
Placa gráfica	AMD Radeon Vega 10 Graphics	NVIDIA GeForce RTX 2080 super
Memória RAM	12 GB	16 GB

A escolha da utilização de dois computadores diferentes teve os seus fundamentos: o computador portátil, não sendo o ideal para algumas atividades que se procederão, como o treino de deteção de imagens, beneficia da disponibilidade a tempo inteiro. Logo grande parte do código foi desenvolvido neste computador; o servidor contém um hardware mais potente e é compatível com um certo pacote, CUDA (*Compute Unified Device Architecture*) – uma plataforma de computação paralela desenvolvida pela *NVIDIA* que permite dividir as tarefas de processamento entre o CPU (*Central Process Unit*) e GPU (*Graphics Process Unit*), estando o primeiro a correr a secção sequencial enquanto a porção intensiva corre em milhares de núcleos distribuídos na unidade gráfica. A unidade gráfica tem uma capacidade de processamento forte, logo um treino de deteção de objetos correrá muito mais rápido que o mesmo código a processar na unidade de processamento do computador portátil.

Uma das dificuldades encontradas foi a diferença dos núcleos dos sistemas operativos, sendo o do *Windows 10* o *Windows* e do *Ubuntu* o *Linux*. Esta diferença faz com que se necessitasse de alterar diversos comandos para outra anotação.

3.1.2. Bibliotecas, API's e pacotes de programação

A biblioteca recorrida, suportada pela linguagem *Python*, adequada à implementação de CNN's foi ***Tensorflow***.

Tensorflow é o framework completo da *Google* para desenvolvimento de aplicações que executam tarefas de *Machine Learning*. Estas aplicações descobrem padrões num vasto número de dados injetados no algoritmo lidando com incertezas e probabilidades. Segue-se, nos próximos parágrafos, uma explicação do funcionamento, potencialidade e alguns elementos importantes presentes no framework (Scarpino, 2018).

Tensorflow utiliza centenas de servidores para treino rápido, e corre modelos treinados para inferência na produção de várias plataformas, que podem ir de clusters vastamente distribuídos a

dispositivos locais móveis. É também extremamente flexível para suportar experiências e investigação em novos modelos de *Machine Learning*. Utiliza um gráfico *dataflow* unificado para representar a computação num algoritmo e o estado no qual o algoritmo opera. Este framework permite que os vértices dos gráficos representem computações que tenham ou atualizem estados mutáveis, ao contrário dos sistemas *dataflow* tradicionais (Abadi et al., 2016).

As pontes, que contêm os valores dos pesos, movimentam os tensores, que são vetores multidimensionais, entre os nós, e *Tensorflow* insere a comunicação apropriada entre as subcomputações distribuídas.

A maioria das aplicações declaram variáveis, ao contrário de *Tensorflow* que declara *tensors* – vetores com zero ou mais dimensões. Quando um *tensor* é criado, transformado ou processado, a sua operação é guardada num gráfico. Este só é executado numa sessão que o fará na ordem operacional.

Tensorflow permite construir modelos de reconhecimento de imagens através de redes neuronais convolucionais. Através do processo de convolução, cada pixel na imagem é substituído com um produto escalar de duas matrizes bidimensionais. Uma das matrizes é um retângulo com largura e comprimentos de pixéis específicos que envolve parte da imagem, e a outra é um filtro com elementos que determinam o efeito que este tem na imagem, também chamado de *kernel*.

No *Tensorflow*, uma imagem equivale a um *tensor* que contem uma matriz para cada canal de cor da imagem (pode ter três: verde, vermelho e azul, ou um: escada de cinza).

O ***Tensorflow Object Detection API*** (disponível no *GitHub* oficial de *Tensorflow*), é um framework desenvolvido com base em *Tensorflow* que auxilia a construção e o treino de um modelo de deteção de objetos em imagens.

Estando a linguagem, biblioteca e interface de programação definidos para a criação de modelo de deteção de objetos, foi feita uma pesquisa para a seleção do modelo de deteção adequado às condições deste projeto.

Segundo Huang et al. (2017), que testaram diferentes arquiteturas de deteção (*Faster R-CNN*, R-FCN e SSD) avaliando fatores como velocidade, memória e precisão, o SSD com *MobileNet* e SSD com *Inception v2* foram os mais precisos dos modelos mais rápidos.

Complementando a escolha, os modelos SSD têm o melhor comportamento detetando objetos de maior dimensão, e o mesmo com *MobileNet*, numa análise dos FLOPS (*FLoating-point Operations Per Second*) no GPU e da memória consumida, é o mais leve computacionalmente.

O sistema de deteção de objetos selecionado neste projeto foi, então, o **SSD (*Single Shot Detector*)**. Este sistema, criado por Liu et al. (2016), é mais rápido que outros detetores do estado-da-arte, tais como o *Faster R-CNN*, sem sacrificar o desempenho. Esta vantagem faz com que o

SSD seja o detetor mais indicado para situações onde se pretende implementar um modelo de deteção de objetos num dispositivo móvel incapaz de acarretar mais potência. Na Figura 21 encontra-se uma representação gráfica da arquitetura do modelo de deteção SSD.

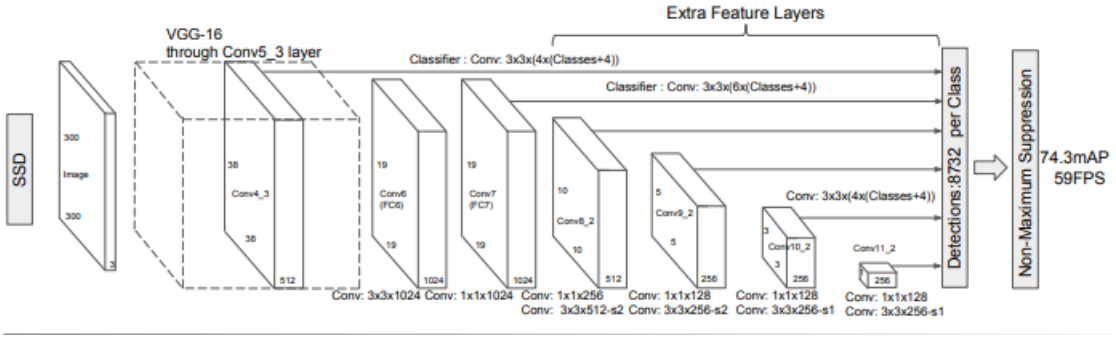


Figura 21. Modelo de deteção SSD. As camadas extra praticam a função de previsão dos desvios das caixas predefinidas de diferentes escalas e proporções e as pontuações associadas.

Liu et al. (2016) indica que o funcionamento base deste sistema consiste numa rede convolucional direta que gera uma coleção restrita de caixas delimitadoras e pontuações para a presença de instâncias das classes de objetos dentro dessas caixas, seguido por um passo de supressão não-máxima para formar as deteções finais. As primeiras camadas são fundamentadas numa arquitetura comum utilizada para classificação de imagens de alta qualidade.

Foram adicionadas algumas caraterísticas para diferenciar este detetor dos demais: o aumento da velocidade deve-se à eliminação das propostas de caixas delimitadoras e a subsequente fase de reamostragem de pixéis ou caraterísticas. Outros aperfeiçoamentos incluem a utilização de um filtro convolucional de dimensões reduzidas para prever categorias de objetos e desvios em localizações de caixas delimitadoras; utilização de preditores separados para deteções de proporções diferentes; e aplicações destes filtros referidos em diversos mapas de caraterísticas das últimas camadas de uma rede de maneira a executar deteção em várias escalas.

Howard et al. (2017) conceberam uma arquitetura de uma rede neuronal que permite que o modelo desenvolvido possua recursos de reduzida dimensão e peso computacional para aplicações remotas e móveis, titulada de ***MobileNet***. Este sistema foi testado num treino de deteção de objetos com dados COCO recorrendo a frameworks, como *Faster-RCNN* e SSD, obtendo resultados, comparados a treinos realizados com outros frameworks, com uma fração da complexidade computacional e tamanho do modelo para ambos os sistemas.

Este framework, segundo Howard et al. (2017), é construído a partir de convoluções separáveis em profundidade e previamente utilizadas em modelos *Inception* para abater os requisitos computacionais nas primeiras camadas.

As convoluções separáveis em profundidade são uma forma de convoluções comuns fatorizadas numa convolução em profundidade e numa convolução 1x1, denominada convolução ponto a ponto. Nas arquiteturas *MobileNet*, a convolução em profundidade aplica um filtro a cada canal de input, enquanto a ponto a ponto impõe uma convolução 1x1 para combinar os outputs.

Nesta arquitetura, todas as camadas seguem uma norma em lotes e uma função ReLU não linear com a exceção da camada final completamente conectada não linear, que regista uma camada softmax para classificação. Outro parâmetro que reduz o peso computacional deste sistema é um multiplicador de resolução ρ, que é aplicado na imagem de input e a representação interna de todas as camadas é, imediatamente, reduzida para o mesmo multiplicador.

3.2. Hardware

Os elementos físicos do projeto são descritos neste capítulo. Estes só foram abordados mais detalhadamente após a conclusão da fase digital do projeto.

A **plataforma robótica** onde é implementado o algoritmo de deteção de objetos para navegação autónoma foi concebida por Veiros (2020).

Trata-se de uma plataforma robótica terrestre multitarefas, com fins de navegação em pomares, idealmente pessegueiros, mas não exclusive. O robô projetado é fruto de uma das atividades do projeto PrunusBot, que promove o desenvolvimento de sistemas robóticos concentrados na inovação tecnológica na fruticultura. As tarefas principais, de momento, da plataforma robótica são a aplicação automática de herbicida, de forma económica e ecológica para controlo de flora infestante, e suporte de um braço robótico automatizado com finalidade de apanha de frutos caídos no chão que, potencialmente, trariam doenças e pragas para a árvore e, para mais, aproveitamento do fruto caído para alimentação de pequenos ruminantes.

Na construção desta plataforma robótica, os perfis do robô foram pensados de modo que o acoplamento de outros componentes para realizar outras tarefas fosse de fácil execução. Assim, a integração de uma câmara frontal para o intuito deste projeto será uma operação relativamente simples.

A Figura 22 contém uma imagem da plataforma robótica legendada com os componentes importantes:

1. Reservatório do herbicida;
2. Sistema de visão computacional destinado à deteção das ervas daninhas;
3. Manipulador robótico cartesiano que controlo o bico pulverizador;
4. Sistema de controlo com Raspberry Pi 4 e ESP32.

Figura 22. Plataforma robótica em questão legendada (Gaspar et al., 2020).

Dado que o objetivo deste projeto é o estabelecimento de um sistema que permite que a plataforma robótica se desloque autonomamente, para conciliar o algoritmo com os controladores das rodas, o algoritmo de deteção dos troncos das árvores, e a câmara que capta imagem a tempo real, é essencial um controlador com extrema mobilidade e com capacidade computacional para acarretar tantas competências.

O **Raspberry Pi 4** foi considerado a melhor solução para este problema. Um microcomputador de reduzidas dimensões, ideal para aplicações móveis e remotas, com uma capacidade de processamento computacional e de imagem (Lowe, 2017).

Não é inédito o recurso ao Raspberry Pi em situações de emprego de redes neuronais artificiais ou algoritmos de deteção de algum objeto ou indicação. Gupta et al. (2016) e Wazwaz et al. (2018) utilizaram este controlador para implementar um sistema de deteção de caras. Sumardi et al. (2018) empregaram o seu algoritmo de deteção das linhas da estrada num Raspberry Pi 4 para fins de locomoção autónoma; Wisnudhanti & Candra (2020) recorreram ao mesmo aparelho para

concretizar uma deteção de fantoches tradicionais indonésios através de CNNs, e, mais especificamente na agricultura de precisão, que é a área de aplicação deste projeto, Gonzalez-Huitron et al. (2021) criaram um algoritmo de deteção de doenças em folhas de tomateiros, via CNN, sendo, o Raspberry Pi 4, o aparelho onde este programa corre.

A **câmara** utilizada para a captação de imagem RGB do robô é a câmara módulo do Raspberry Pi versão 2 (Raspberry Pi, n.d.). Tem um sensor *Sony IMX219* de 8 MP. Suporta resoluções como 1080 com 30 FPS e 720p com 60 FPS.

3.3. Métodos

3.3.1. Configuração do ambiente de trabalho

O primeiro passo foi a configuração do ambiente digital de trabalho. Os computadores pessoais necessitam de ter o software, as bibliotecas e os pacotes necessários para o bom funcionamento de um modelo de deteção de objetos.

Para este projeto, recorreu-se à ferramenta *Anaconda* para se responsabilizar da gestão dos pacotes e dos ambientes virtuais, adicionando-a aos destinos de variáveis do ambiente do sistema, para que fosse o distribuidor de *Python* do sistema pré-definido.

Após a criação de um ambiente virtual e da sua ativação numa janela de terminal do computador, segue-se para a instalação e verificação desta do pacote *Tensorflow*.

No servidor, foi instalado o pacote CUDA, visto que a sua unidade gráfica é compatível com este. Este pacote não é obrigatório, mas é recomendado quando dada a possibilidade de o utilizar, dado que acelerará o processo de treino.

O próximo passo residiu em descarregar os modelos de *Tensorflow* do seu *GitHub*. Neste sistema há inúmeras implementações do estado-da-arte disponíveis para utilizadores usufruírem das capacidades de *Tensorflow,* das quais o *Object Detection API*, que foi usado neste projeto para a deteção de objetos.

Após todas as instalações e verificações, a fase de configuração dos sistemas computacionais foi concluída e pronta para a criação do modelo de treino de deteção de objetos. Esta instalação é completa e assegura a presença de pacotes extra essenciais, tais como o OpenCV, pyCOCOTools, NumPy, entre outros.

3.3.2. Construção da base de dados

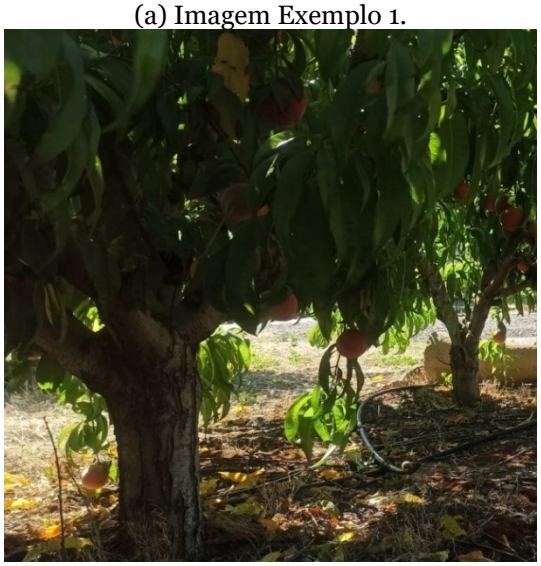
A base de dados é composta por fotografias e vídeos obtidos num pomar de pessegueiros, localizado em Orjais, Covilhã. Estes vídeos e fotografias foram tirados na perspetiva visual que a plataforma robótica irá ter quando praticar a sua locomoção correta, ou seja, encostada o mais à esquerda possível, e em locais aleatórios do pomar. Dessas fotografias, foram aproveitadas as que tinham uma imagem mais nítida dos troncos das árvores. No total, a base de dados estava com 89 imagens e suas anotações. A Figura 24 mostra quatro amostras das fotografias obtidas no pomar que têm conceção evidente dos troncos das árvores. Todas as imagens foram recortadas e redimensionadas para uma dimensão de 640x640 para estarem compatíveis com a configuração de input do modelo de deteção SSD que foi utilizado.

Recorrendo ao *labelImg,* de Tzutalin – uma ferramenta de anotação de imagens open-source – anotaram-se os troncos das imagens, criando uma área retangular com quatro coordenadas à volta dos troncos, e atribuíam-se os rótulos da classe (tronco) às caixas delimitadoras, ficando, estes dados, registados em ficheiros **.xml*. Uma amostra do processo da anotação está representada na Figura 23.

Figura 23. Processo de anotação da classe dos troncos, nas imagens.

(a) Imagem Exemplo 1.

(b) Imagem Exemplo 2.



(c) Imagem Exemplo 3.

(d) Imagem Exemplo 4.

Figura 24. Exemplo de quatro fotografias consideradas adequadas para o treino do modelo de deteção, com a perpetiva pretendida, cortadas e redimensionadas para o treino.

3.3.3. Criação do modelo

Estando os computadores com os ambientes de trabalho e virtuais definidos e com os requisitos, softwares e pacotes essenciais instalados e a base de dados com as imagens e anotações feitas, o processo da criação do modelo para ser treinado fica possível de concretizar corretamente.

A pasta que contém todos os dados dividiu-se em várias subpastas para uma melhor organização:

- **Anotações:** Esta pasta guarda os registos *Tensorflow* (**.record*), que são ficheiros gerados da conversão das anotações referidas anteriormente, que unem e convertem os dados anotados das classes e posições nas imagens utilizadas para o treino e teste.

- **Modelos-exportados:** Esta pasta guarda a(s) versão(ões) exportada(s) do(s) modelo(s) exportado(s) final(is).

- **Imagens:** Esta pasta possui uma cópia de todas as imagens da base de dados, tal como os respetivos ficheiros, que contêm as informações das anotações já feitas previamente (**.xml*), produzidos para cada uma.

    - **Treino:** Nesta pasta encontram-se as imagens e os ficheiros das anotações que serão utilizadas para o treino do modelo.

    - **Teste:** Nesta pasta encontram-se as imagens e os ficheiros das anotações que serão utilizados para o teste e avaliação da precisão e outros parâmetros e métricas do modelo.

- **Modelos-pré-treinados:** Esta pasta guarda os modelos pré-treinados adquiridos a que se podem recorrer no treino do modelo.

- **Modelos:** Esta pasta contém uma subpasta para cada treino que se realiza. Cada subpasta tem o ficheiro de configuração do modelo (*pipeline.config*), que possui as informações como a dimensão dos lotes, o número máximo de passos, as localizações dos ficheiros importantes, entre outras informações essenciais.

Foi crucial realizar uma seleção das imagens e respetivos ficheiros das anotações e distribuir uma porção para treino e outra para teste. Das 89 imagens no total, 90% foram escolhidas para treino e os restantes 10% para teste e avaliação da condição do modelo após o treino.

Com as anotações feitas e as imagens distribuídas para treino e teste, é fundamental um ficheiro específico com as diversas classes registadas, neste caso havendo só uma (tronco), denominado de mapa de rótulos. É um ficheiro de texto com a extensão **.pbtxt* (*label_map.pbtxt*) que é guardado na pasta Anotações. Segue-se o conteúdo textual desse ficheiro na Figura 25.

```
item {
      id: 1
      name: 'tronco'
}
```

Figura 25. Conteúdo do mapa de rótulos *label_map.pbtxt*.

Para se criar os registos *Tensorflow*, que abrangem a informação das anotações delimitadas nas imagens de treino e teste num só ficheiro **.record*, foi necessário um script (com nome *generate_tfrecord.py*). Uma amostra deste encontra-se no *GitHub* de Nicknochnack (2021). Na sua

execução, somente foi necessário informar o programa do destino da pasta de treino ou de teste (-x), do mapa de rótulos (-l) e o destino onde seria guardado o ficheiro gerado (-o). O comando, que se pode correr numa janela de terminal dentro do diretório onde está o script, situa-se na Figura 26.

```
python generate_tfrecord.py -x [DESTINO_DO_MODELO]/imagens/treino -l
[DESTINO_DO_MODELO]/anotações/label_map.pbtxt                      -o
[DESTINO_DO_MODELO]/anotações/treino.record

python generate_tfrecord.py -x [DESTINO_DO_MODELO]/imagens/teste -l
[DESTINO_DO_MODELO]/anotações/label_map.pbtxt                     -o
[DESTINO_DO_MODELO]/anotações/teste.record
```

Figura 26. Comando de criação de registos *Tensorflow* (em cima do treino; em baixo de teste).

Relativamente ao modelo pré-treinado, como já foi referido anteriormente no capítulo dos recursos de software, foi aplicado o sistema *SSD MobileNet*, devido à sua rapidez e baixo peso computacional – requisito necessário numa aplicação móvel como a de este projeto.

O modelo em concreto é o *SSD MobileNet* versão 2, com pesos de uma outra rede já treinada na base de dados COCO 2017 com resolução de input 640x640. Esta base de dados COCO (*Common Objects in COntext*) é constituída por modelos pré-treinados do estado-da-arte para evitar que desenvolvedores de algoritmos de deteção de objetos criem uma base de dados de raiz, visto que necessita de inúmeros exemplos de imagens em várias situações e perspetivas (Solawetz, 2020). Deste modo, o recurso a um modelo pré-treinado com esta base de dados torna o desenvolvimento de deteção de objetos muito mais prático e rápido.

O modelo em questão possui uma particularidade denominada de FPN (*Feature Pyramid Network*), cujo substitui o extrator de caraterísticas comum em outras abordagens de deteção de objetos, para um conceito em pirâmide, algo que é habitual em detetores de objetos expondo a imagem em diferentes escalas nessa forma, e gerando vários mapas de caraterísticas com melhor qualidade de informação, criando um caminho de sentido inverso do *dataflow* para construir camadas com resolução mais elevada. Ao mesmo tempo adiciona conexões laterais entre as camadas reconstruídas e os mapas de caraterísticas para ajudar o detetor a prever as localizações (Hui, 2018). Uma representação gráfica encontra-se na Figura 27.

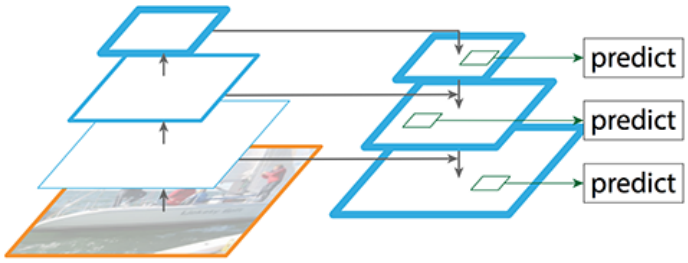


Figura 27. Esquema da organização das camadas de uma abordagem com FPN (Hui, 2018).

O modelo pré-treinado possui um ficheiro de configuração com todas as definições a aplicar no treino do modelo (ver Anexos desta dissertação – Figura 44 – a negrito nas linhas de comando estão os parâmetros alterados para as especificações deste modelo). Um esquema simplificado da estrutura do modelo no contexto deste projeto encontra-se na Figura 28. As alterações realizadas foram:

- **Número de classes:** Número de classes que estão definidas no modelo de deteção. Neste projeto existe apenas uma classe: o tronco da árvore.

- **Dimensão dos lotes:** A dimensão dos lotes é o número de amostras que entra na rede por unidade de tempo. Quanto maior a dimensão dos lotes, mais pesado e demorado será o treino. A dimensão dos lotes convém que seja um valor exponencial de base 2, devido à relação com o sistema binário. Para esta simulação definiu-se este valor para 8.

- **Número de passos:** O número de passos é o número de vezes que os lotes são passados pela rede. Após a execução de treinos de modelos protótipo anteriores, concluiu-se que 8000 passos é um número coerente para este projeto.

- **Destinos dos ficheiros:** Foi necessário estabelecer os destinos como a localização do checkpoint do modelo pré-treinado, do mapa de rótulos, dos registos *Tensorflow* de treino e dos registos *Tensorflow* de teste.

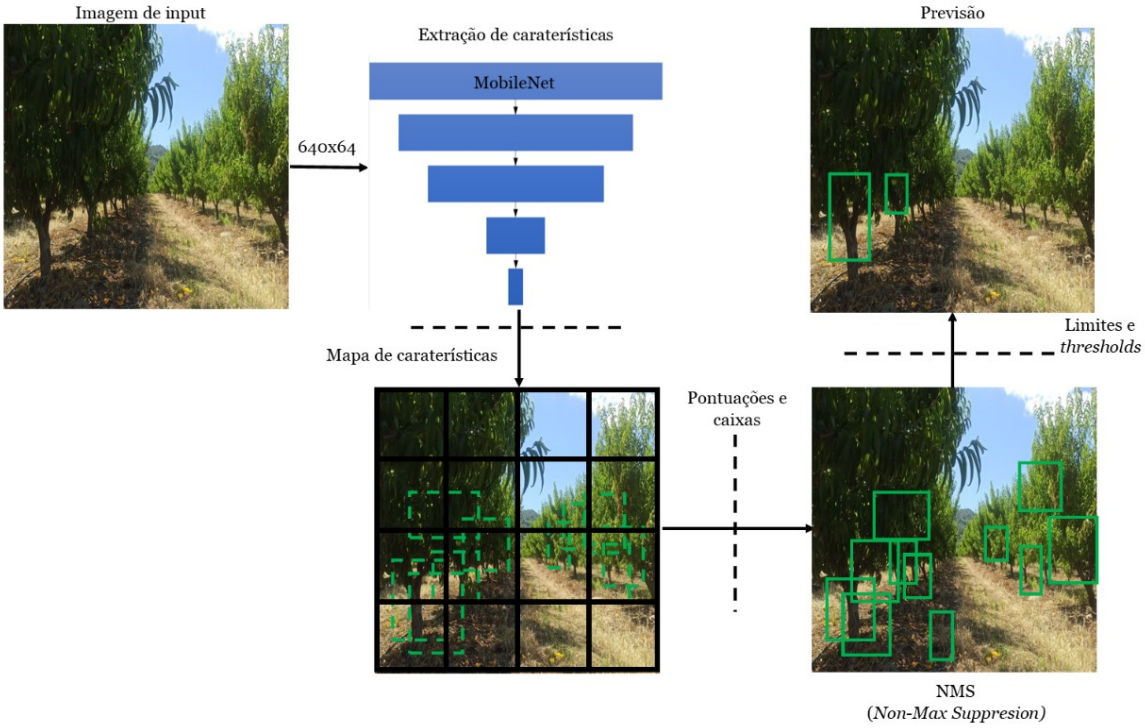


Figura 28. Esquema simplificado do CNN da essência deste projeto.

Por fim foi necessário um script que valida todos os dados, disponibilizado nos modelos adquiridos no *GitHub* de *Tensorflow*, o *model_main_tf2.py*. Executou-se o comando fornecendo o destino onde o modelo treinado será guardado e onde se encontra o ficheiro de configuração do treino.

3.3.4. Estratégia de orientação

Com o modelo já treinado e a detetar os troncos corretamente, é necessário formar uma estratégia para indicar à plataforma robótica quando e em que condições terá que ajustar a sua trajetória. Como já é conhecido, a câmara é montada na frente do robô e a visão que dispõe tem um panorama mais próximo da fila de árvores esquerda que da direita, como mostra a Figura 29.

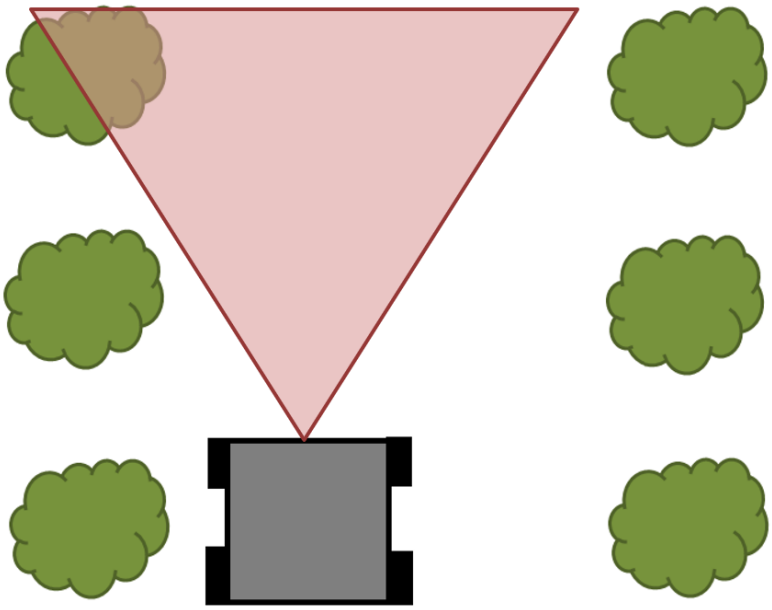


Figura 29. Visão panorâmica da plataforma robótica (perspetiva topográfica).

Observando a perspetiva da plataforma robótica, nota-se uma particularidade: as filas das árvores, tanto da direita como da esquerda, formam uma linha que converge num ponto comum, sendo este o final do corredor do pomar, como está representado na Figura 30. Optou-se, então, que, a tendência desta linha formada no lado esquerdo ter uma trajetória correta sem obstruções para o ponto de chegada da plataforma robótica seria o melhor método de orientação.

Figura 30. Linhas imaginárias para a estratégia de orientação.

Uma imagem, ou quadro de vídeo, possui coordenadas cartesianas x e y, o que permite conceber qualquer interação entre pontos espalhados pelo plano. A orientação dos eixos está representada na Figura 31.

Figura 31. Configuração e orientação dos eixos cartesianos no plano deste projeto.

Para abordar a técnica de orientação, recorreu-se à obtenção de coordenadas de dois pontos de duas caixas delimitadoras de troncos detetados, e, a partir destes, forma-se uma função linear que atualiza a cada quadro de vídeo que contém uma deteção da mesma configuração.

Uma função linear é, tipicamente, representada pela Equação (1), onde m é o declive da reta e b a ordenada na origem.

$$y = mx + b \tag{1}$$

A equação da função linear pode ser descoberta conhecidos dois pontos e as suas coordenadas. Para fácil compreensão, imaginam-se dois pontos num plano com os eixos x e y, o ponto A (Ax; Ay) e B (Bx; By), como representa a Figura 32. O declive de uma função linear representa a taxa de alteração das ordenadas da função com a alteração das abcissas, e pode ser calculado pela Equação (2), se se souber dois pontos pertencentes à reta. Para completar a função linear, ainda é necessário conhecer a ordenada que interseta a origem (b), ou seja, quando $x = 0$. Para isso, depois de o declive ter sido determinado, a partir da equação comum da função linear, a equação que define b encontra-se na Equação (3).

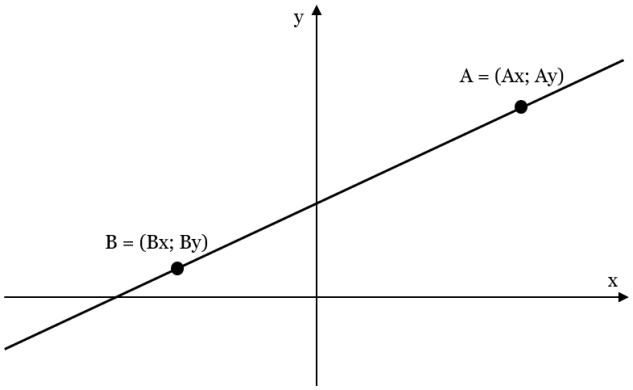


Figura 32. Representação gráfica genérica de uma função linear.

$$m = \frac{Ay - By}{Ax - Bx} \tag{2}$$

$$b = Ay - m \cdot Ax \;\; ou \;\; b = By - m \cdot Bx \tag{3}$$

Cada caixa delimitadora possui quatro valores: uma abcissa mínima (*xmin*) e máxima (*xmax*), e uma ordenada mínima (*ymin*) e máxima (*ymax*). Qualquer combinação das abcissas com as ordenadas resulta num vértice da caixa e, conhecendo a orientação dos eixos numa imagem, optou-se por se escolher o vértice com a abcissa máxima e ordenada máxima (*xmax*; *ymax*). Neste projeto, assumiu-se os dois pontos, *A* e *B*, como sendo vértices de duas caixas delimitadoras de dois troncos detetados com as abcissas e ordenadas máximas consideradas, estando um exemplo desta tática na Figura 33.

Figura 33. Função linear gerada pela deteção de dois troncos de pessegueiros.

Após a primeira criação da função linear com dois troncos detetados, esta é guardada e atualizada posteriormente caso haja novamente deteção dupla. Esta função linear será o ponto base de orientação e terá duas condições que fazem com que a plataforma robótica se ajuste á trajetória.

A primeira condição é o desvio de um só tronco detetado. Haverá, inevitavelmente, devido a obstruções visuais, troncos de formas ambíguas e à própria imprecisão do modelo, algumas frações de tempo em que só é detetado um tronco. Com a função linear criada, o algoritmo recolhe a ordenada máxima ($ymax$) do único tronco detetado e calcula uma abcissa pertencente à função ($xref$). De seguida, esse valor *xref* é comparado com a abcissa máxima ($xmax$) da caixa delimitadora e, dada uma tolerância segundo o eixo x de 150 pixéis, o algoritmo avalia se é necessário efetuar uma das três decisões:

- Se $xmax > xref + 150$, a plataforma vira à **direita** para deslocar a linha da função linear para ficar mais próxima do vértice da caixa em avaliação.

- Se $xmax < xref - 150$, a plataforma vira à **esquerda** para deslocar a linha da função linear para ficar mais próxima do vértice da caixa em avaliação.

- Se $xref - 150 < xmax < xref + 150$, o algoritmo assume que a plataforma está com uma boa orientação e **prossegue a marcha**.

Esta condição é essencial, mas não suficiente, porque o robô pode desviar-se demasiado da trajetória sem deteções e assim a estratégia de deteção do tronco relativamente à função linear pode dar indicações incorretas. Desse modo, para que a plataforma não tenha uma tendência, ou de colisão com a fila de árvores, ou de afastamento desta, é feita também uma avaliação do declive da função linear criada. Uma reta com uma inclinação muito acentuada pode significar que as árvores estão afastadas do panorama comum, ou, no caso contrário, se a reta estiver muito abatida, com um aspeto mais horizontal, significa que o robô tem uma tendência de deslocamento perpendicular relativamente à fila de árvores.

Para evitar esta situação, quando o declive é calculado, verifica-se se este está dentro de um intervalo aceitável de valores. As indicações são as seguintes:

- Se $m < -1$, significa que a reta criada pela função está a ficar mais acentuada, assim sendo a afastar-se, logo o robô ajusta a trajetória para a **esquerda** (imagem da direita da **Erro! A origem da referência não foi encontrada.**).

- Se $m > -0{,}2$, significa que a reta criada pela função está a ficar mais abatida, assim sendo em colisão com a fila, logo o robô ajusta a trajetória para a **direita** (imagem da esquerda da **Erro! A origem da referência não foi encontrada.**).

- Se $-1 < m < -0{,}2$, o algoritmo assume que a trajetória da plataforma é aceitável e **prossegue a marcha**.

É importante dizer que, no início da marcha do robô, enquanto não houver uma deteção dupla de troncos, o robô continua o seu movimento a direito até adquirir informação para que possa fazer a avaliação do redor.

Esta estratégia é possível graças a um algoritmo criado que faz a deteção com o modelo treinado que, consequentemente, dará a ordem de movimentos segundo comandos de output e estruturas de seleção. Uma amostra do algoritmo encontra-se em Anexo - Figura 45.

3.3.5. Quantização do modelo para configuração móvel

O modelo treinado e o algoritmo que faz a deteção e o método de decisão de orientação foi implementado num *Raspberry Pi 4*. Este modelo, apesar de ser mais leve que outros sistemas de deteção devido ao SSD *MobileNet*, continua a ter um peso computacional excessivo para uma configuração móvel, logo é necessário aliviar ainda mais a potência computacional requerida.

Tensorflow Lite é um framework, tal como *Tensorflow*, para uso em dispositivos móveis, mas que somente faz inferência – o processo de deteção de objetos. Um modelo comum *Tensorflow*, para que seja executado no formato *Lite*, necessita de ser convertido (Khandelwal, 2020).

Para auxiliar o *Raspberry Pi 4* a executar a inferência do modelo, decidiu-se complementar o dispositivo com um *Edge TPU*, um coprocessador conectado por USB que divide as tarefas de processamento, tornando programas destinados a inferência mais rápidos. Este dispositivo paralelo suporta somente inferências de estrutura com palavras inteiras de 8-bit. Foi, então, necessária uma **quantização** do modelo de deteção. Este processo é explicado de seguida.

O procedimento de adaptação do modelo criado para a configuração móvel passou por uma conversão do modelo *Tensorflow* comum para uma configuração pós-treino mais leve, ou seja, *TFLite*, sofrendo, ao mesmo tempo, uma quantização de 32-bit flutuante (*32float*), para 8-bit inteiros positivos (*8uint*). Procedeu-se à compilação do modelo já quantiado para uma estrutura suportada pelo operador *Edge TPU*. Somente após a instalação do detetor *TFLite* e da biblioteca *Edge TPU* no *Raspberry Pi 4*, é que é possível correr o modelo de deteção adaptado para um bom desempenho num dispositivo móvel.

A criação de um modelo *Tensorflow Lite*, ou *TFLite*, possui várias técnicas e configurações recomendadas, dependendo do hardware que seja utilizado. Como neste projeto o modelo se encontra num microcomputador para situações de mobilidade, a técnica de quantização integral completa é a mais indicada, tornando o modelo aproximadamente 4x menor, triplicando a velocidade de execução e apto para ser executado com a biblioteca *Edge TPU*. Este método converte as funções,

pesos e viés que estão em valores flutuantes de 32-bit em inteiros de 8-bit positivos (*Tensorflow*, n.d.-c). A amostra do script de conversão e quantização do modelo *Tensorflow Lite* encontra-se em Anexo - Figura 46.

No algoritmo do modelo *TFLite*, para se proceder à inferência, é necessário simplesmente carregar-se o interpretador (*Tensorflow Lite* e *Edge TPU*), repartir o tensor e obter o de input e output, processar e transformar a imagem a ler num tensor, executar a inferência e obter a imagem de output com a deteção.

O algoritmo de inferência de *Tensorflow* para *Tensorflow Lite* é bastante diferente, tendo sido necessária uma restruturação drástica. Relativamente à implementação da estratégia de orientação no novo algoritmo reconfigurado foi simples. Apenas foi necessário identificar as variáveis que definem as caixas delimitadoras e aplicar a conversão da escala.

A particularidade da escala de um modelo de inferência quantiado é, tanto os detalhes do input como de output, possuem um fator de escala de quantização e um ponto de identificação do zero. Aplicando a Equação (4) (Sahni, 2018) aos dados de input e output do modelo quantiado (coordenadas das caixas delimitadoras e valores de certezas), onde r é o valor real original *32float*, S o fator de escala de quantização, q a representação quantiada, e z o ponto zero, o valor retornado é o valor original *32float* do modelo prévio. Com esta conversão, é possível aplicar a estratégia de orientação originalmente concebida.

$$r = S \cdot (q - z) \tag{4}$$

Em Anexo encontra-se a amostra do algoritmo reconfigurado para *TFLite* para inferência em vídeo (Figura 47) e em câmara (Figura 48).

4. Análise e Discussão de Resultados

Este capítulo relata os dados e os resultados dos testes que comprovam o bom funcionamento do algoritmo e da resposta da plataforma robótica. As métricas são registos feitos durante o treino ou comparações com a base de dados do teste para avaliar o algoritmo matematicamente e graficamente.

Foi desenvolvida uma simulação, tanto em imagens como em vídeos capturados no pomar para analisar o comportamento da plataforma robótica num percurso correto e em rotas falaciosas para verificar se o algoritmo provoca um ajuste na trajetória.

4.1. Métricas

Durante o processo de treino, registou-se, em ficheiros de checkpoint, dados que relatam o desempenho tanto do próprio treino como uma avaliação da precisão da deteção nas imagens de teste comparada com a anotação feita previamente.

A função de perda (*loss*) é uma função escalar que quantifica a diferença entre o valor previsto de um dado de entrada e o valor verdadeiro (Abadi et al., 2016). Os valores que tornam possível a construção da função de perda são recolhidos durante o treino do modelo. A Figura 34 representa o gráfico da função de perda do modelo, com 8000 passos. Como se pode verificar, o valor de perda no início era aproximadamente 0,6, decrescendo até alcançar valores constantes a rondar 0,2 próximo dos 3000 passos, atingindo ao fim de 8000 passos um valor baixo de 0,16. Este resultado indica que os passos atribuídos no treino foram suficientes, dando uma margem para redundância reduzida e atingir um valor mínimo estável.

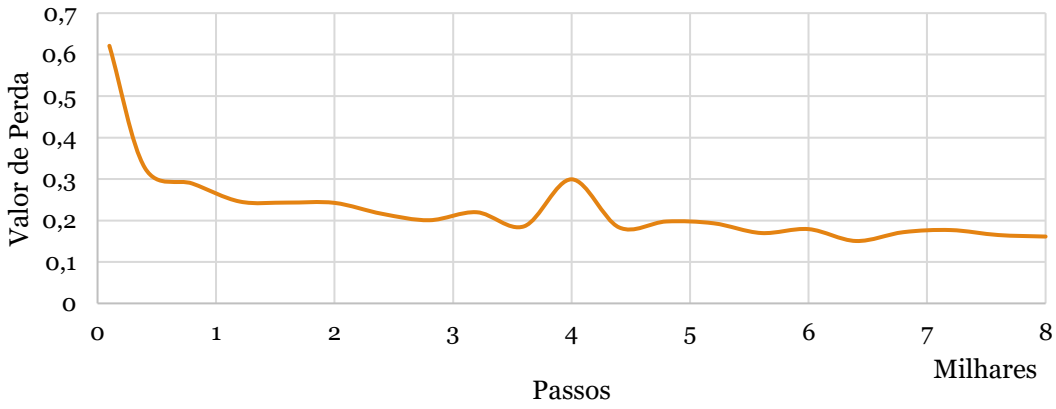


Figura 34. Evolução do valor da função de perda do treino do modelo.

Para avaliar o desempenho do modelo após o treino, compara-se uma deteção feita nas imagens de teste pelo modelo treinado com a anotação realizada. Os resultados são divididos em precisão e revocação (*recall*).

Estas duas métricas envolvem verdadeiros e falsos positivos e negativos. Um positivo verdadeiro corresponde a uma deteção correta de um objeto, enquanto um positivo falso é uma deteção feita, mas incorretamente, ou seja, a deteção está na imagem, mas não no objeto. Um negativo verdadeiro acontece quando não é detetado o objeto, estando este ausente da imagem, e um negativo falso é a ocorrência de uma não deteção, estando o objeto presente. Na Figura 35 encontram-se quatro exemplos das situações possíveis que acontecem durante a execução do modelo, e úteis para avaliação do seu desempenho.

A métrica de precisão retrata a proporção das deteções corretas. A métrica de revocação retrata a proporção dos objetos reais que se detetou. O valor de precisão é determinado pela fração das deteções positivas verdadeiras com todas as deteções efetivamente presentes. A revocação representa a percentagem das deteções positivas verdadeiras do número total de objetos presentes na imagem, detetados ou não.

Outra métrica muito popular recorrida em aplicações de deteção de objetos, que compara a semelhança entre dois objetos arbitrários, é o índice de Jaccard (*Intersection over Unit – IoU*) (Assunção et al., 2020; Rezatofighi et al., 2019). A fórmula matemática que retrata esta métrica está na equação (4), sendo A a área delimitada da deteção da inferência e B a área real incumbida no teste. Esta avaliação permite verificar o desvio das deteções nos objetos pretendidos.

$$IoU = \frac{|A \cap B|}{|A \cup B|} \tag{5}$$

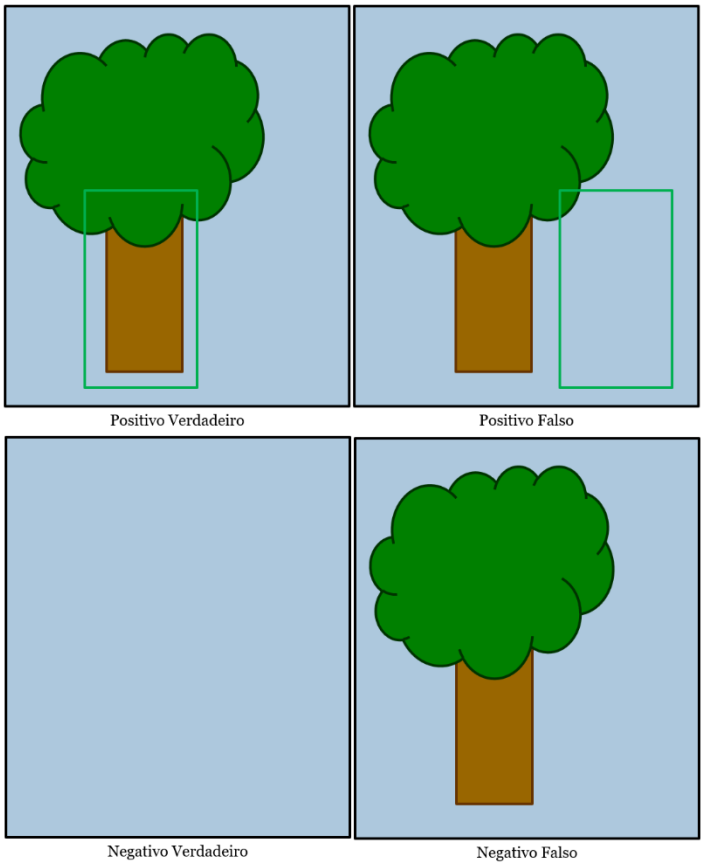


Figura 35. Situações possíves na deteção.

O valor da precisão média do modelo, com um *IoU* de 0.5, foi de 94.4%, o que significa que quase todos os positivos das deteções nas imagens de teste, com pelo menos metade da área ocupada pela anotação feita nas imagens de teste, são verdadeiros, tornando a resposta do modelo com elevada precisão. Para aferir o valor de revocação, verificou-se que, nas imagens de teste, em 18 troncos presentes, 17 foram detetados, resultando numa percentagem de 94.4%. Estas métricas mostram que o modelo original mostra um ótimo desempenho a detetar os troncos com alta precisão sem falhar quase nenhum.

A quantização de um modelo, inevitavelmente, afeta o desempenho deste. A conversão dos elementos internos da rede treinada de *32float* para *8uint* afetará todas as interações, viés e pesos entre os percetrões, alterando os valores de output.

Praticou-se a inferência, com uma certeza mínima de 0%, do modelo *TFLite* com *Edge TPU* nas 9 fotografias que se utilizaram para a avaliação do treino e verificou-se que em 13 deteções que ocorreram, 12 eram efetivamente troncos de árvores, resultando numa precisão de 92,3%. Verificou-se, também, que nessas imagens foram anotados 18 troncos, resultando numa revocação de 66,7%. Esta estimativa mostra que a perda da precisão é quase nula, sendo só afetada a revocação.

Deste modo, confia-se que as deteções que este modelo infere são de facto troncos, podendo apenas falhar alguns troncos no curso da navegação.

O valor das métricas não sustenta uma exatidão da precisão e revocação real do modelo. Comparando a deteção das imagens de teste com a anotação, é permitido verificar que, por vezes, a avaliação do modelo consegue ser mais exata que a própria anotação, como mostra a Figura 36 que ocorre uma deteção de um tronco que não foi anotado previamente ao treino.

(a) Imagem de teste. (b) Anotação feita antes do treino.

Figura 36. Comparação da inferência realizada na imagem de teste (a) com a anotação feita antes do treino (b).

4.2. Simulação

Antes de implementar o algoritmo na plataforma robótica, pretendeu-se realizar uma simulação no computador, primeiro numa imagem, e depois num vídeo, para se verificar se o programa funciona corretamente. Para testar o comportamento da plataforma robótica numa simulação, ao invés de se usar os outputs da viragem das rodas como ordem, recorre-se a mensagens de texto visíveis no terminal do IDE.

Para uma correta deteção, foi incrementado um limite mínimo de certeza de 12% para o modelo original, e 16% para o modelo quantiado. Estes valores equivalem ao dobro da falta de precisão aproximada de cada modelo. Um número máximo de três caixas também foi implementado presentes ao algoritmo para que este só registe troncos quase certos sem interferência de dados falsos. Um exemplo de uma deteção dupla numa imagem com a linha da função linear já foi visualizado neste projeto anteriormente, no subcapítulo Estratégia de orientação, na Figura 33. Neste caso, a perspetiva mostra uma trajetória aceitável, logo o feedback do programa é uma ordem para prosseguir a marcha, como mostra a Figura 37.

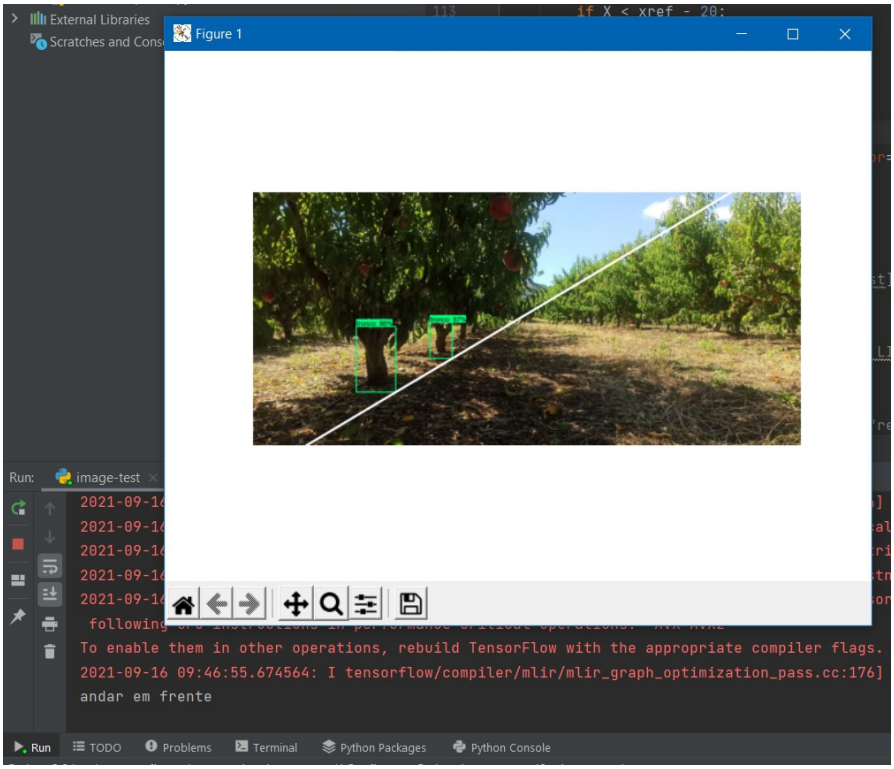
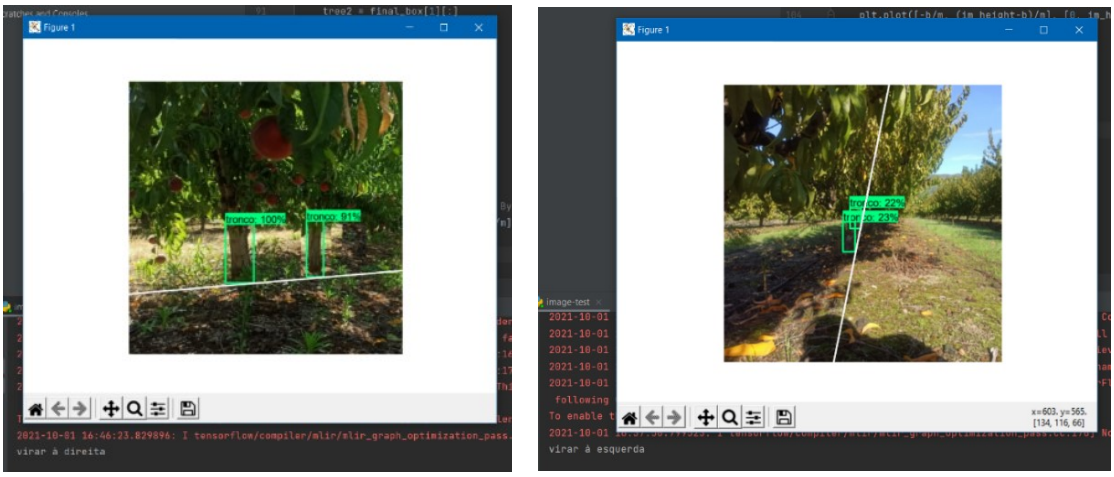


Figura 37. Simulação na imagem numa situação de orientação aceitável.

Na Figura 38 reconhece-se uma linha vermelha horizontal com o vértice (*xmax*; *ymax*) como ponto médio. Esta condição traduz a tolerância dada ao desvio da caixa delimitadora, com valor de 150 pixéis para cada lado, se só um tronco for detetado. Qualquer desvio para além deste limite aciona a ordem de ajuste de trajetória.

Figura 38. Linha representativa do desvio da deteção singular permitido segundo o eixo das abcissas.

Foram recolhidas fotografias, no pomar, com perspetivas de trajetória erradas propositadamente para testar o comportamento do robô na simulação prévia de imagem e vídeo. A Figura 39(a) representa uma trajetória em rota de colisão com a fila de árvores, daí a reta da função linear estar tão abatida. Como se pode verificar na janela de execução do IDE, a ordem é de virar à direita, devido ao declive estar com valores $m > -0{,}2$. Noutra situação, como na Figura 39(b), o robô aparenta estar a ir na direção da fila de árvores da direita, como a reta da função linear está mais acentuada, ou seja, com valores $m < -1$, a indicação dada ao robô é de virar à esquerda.



(a) declive abatido: virar à direita (b) declive acentuado: virar à esquerda

Figura 39. Situações possíveis onde o valor do declive da reta auxilia na orientação.

Fazem parte do material de simulação, vídeos para testar o comportamento e as indicações dadas à plataforma robótica. Esta simulação será mais concreta, pois a plataforma tem que ter capacidades de ajustar a sua trajetória e avaliar o seu redor a tempo real. Presentes estão três figuras que são capturas de uma simulação em vídeo:

Figura 40 está o início do vídeo: como se pode verificar, a plataforma tem ordem para seguir em frente pois ainda não recebeu qualquer informação do seu redor. Na Figura 41 surgem os primeiros momentos em que a função linear é criada, tendo esta um declive dentro do intervalo aceitável, daí a ordem de prosseguir a marcha. Já na Figura 42, um tronco é detetado desviado para além da tolerância da linha da função, a ordem dada à plataforma robótica é de ajustar a sua trajetória para o lado direito, para que a linha fique mais próxima do tronco e mantenha um rumo correto.

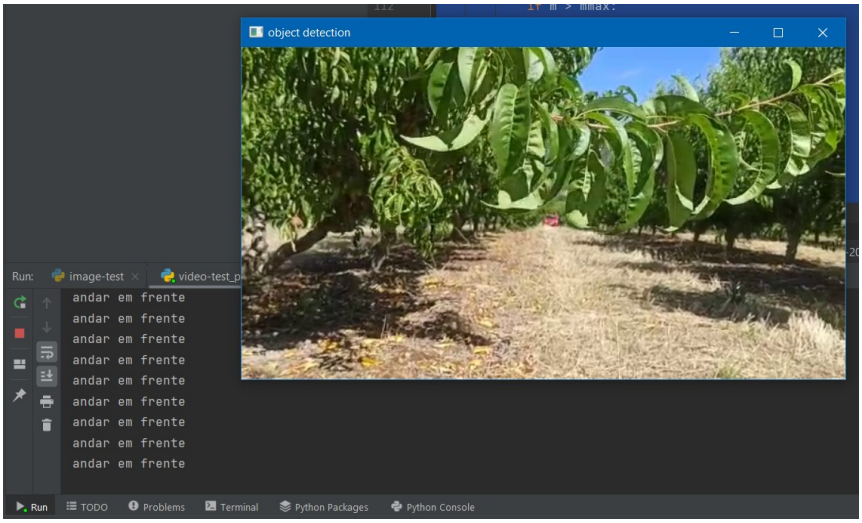


Figura 40. Vídeo de simulação - ínicio sem deteções.

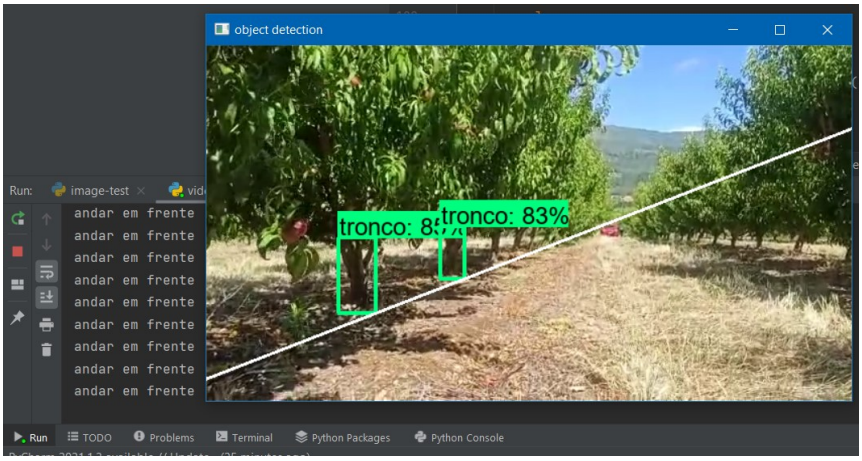


Figura 41. Vídeo de simulação - primeira função linear criada.

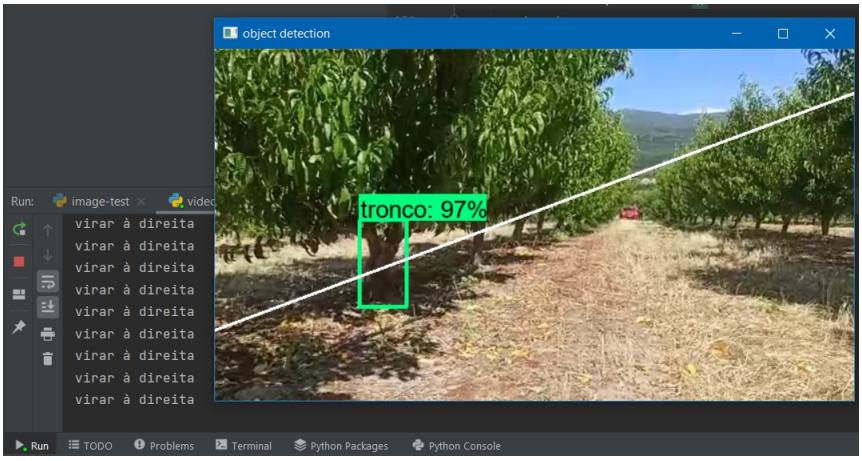
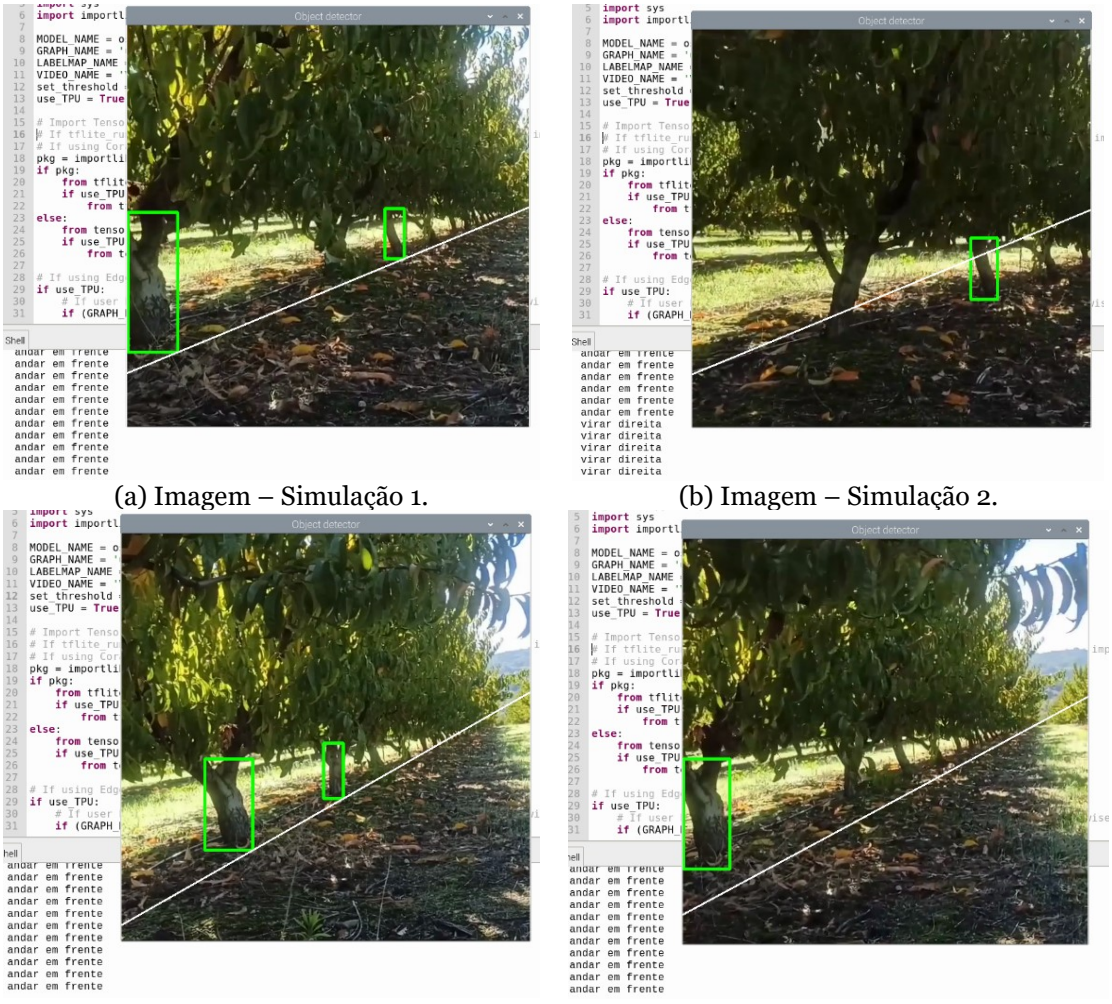


Figura 42. Vídeo de simulação - primeiro tronco detetado desviado.

4.3. Simulação do modelo convertido no microcomputador

Após ter sido verificado que a estratégia de orientação estava a funcionar corretamente, prosseguiu-se ao teste do modelo convertido para *uint8* e compilado para suporte com *Edge TPU*. O intuito desta simulação é verificar se o desempenho foi afetado e o modelo está apto para orientar a plataforma robótica a tempo real.

No subcapítulo Métricas está representada a avaliação aritmética realizada ao modelo convertido, mostrando que se mantém semelhante ao original, sem perdas relevantes de desempenho. Realizou-se uma avaliação do tempo entre inferências para determinar se este é adequado para praticar a deteção simultaneamente dando ordens de orientação sem atraso de informação. Verificou-se que o início de execução levou aproximadamente 0,6 seg. e, daí em diante, as inferências tiveram um intervalo de tempo entre si de cerca de 0,37 seg. Na Figura 43 são apresentadas imagens da simulação do modelo no *Raspberry Pi 4* com a inferência e com a ordem de orientação.



(a) Imagem – Simulação 1.

(b) Imagem – Simulação 2.

(c) Imagem – Simulação 3.

(d) Imagem – Simulação 4.

Figura 43. Imagens da simulção feita no *Raspberry Pi 4* num vídeo do modelo convertido para *TFLite* e compilado para *Edge TPU*.

4.4. Nota conclusiva

Reunindo os resultados das simulações e os valores das métricas dos vários modelos, obtiveram-se as seguintes conclusões:

Relativamente aos valores das métricas, matematicamente e estatisticamente, o modelo original tem um desempenho excelente com ambas uma precisão e revocação de 94,4%. No entanto, foi crucial a conversão e compilação de um modelo *32float* para *8uint* com suporte de *Edge TPU*. Só desta maneira se conseguiu garantir uma maior velocidade de execução da inferência e um peso computacional muito mais reduzido para execução num *Raspberry Pi 4*, tornando, assim, o modelo ideal para tarefas móveis.

Feita a avaliação à inferência nas mesmas imagens de teste com o modelo quantiado e compilado, o valor da precisão mantém-se praticamente inalterado, provando que quase não existem deteções residuais. Porém, o modelo ficou com uma revocação mais baixa, podendo não identificar troncos por que passe. Todavia, após realizadas as simulações em vídeo, assume-se que esta baixa de valores é aceitável e o modelo está apto para ser testado no terreno.

A execução da inferência no *Raspberry Pi 4*, com a conversão e compilação do modelo original, é relativamente rápida com intervalos de tempo aproximadamente de 0,37 seg. entre inferências dos quadros de vídeos, dando bastante tempo à plataforma robótica para se ajustar com uma informação do ambiente redor praticamente exata.

O comportamento do algoritmo mostrou-se adequado e a cumprir os objetivos propostos. As simulações em imagens e vídeos, tanto do modelo original como do convertido, mostraram resultados positivos, com poucas falhas de deteção. O algoritmo inferia a deteção dupla para criação da função linear de referência relativamente rápida e mesmo com deteções individuais, as ordens dadas eram as corretas.

5. Conclusões

5.1. Conclusões gerais

A aplicação da robótica na agricultura proporciona um aumento de produtividade e uma redução de custos e desperdícios associados. Porém, a sua implementação enfrenta as dificuldades que a robótica comum possui, como a ambientes com eventos aleatórios ou sujeitos a variação das condições climatéricas. Face a esta potencialidade e dificuldades, investidores e investigadores olham para a oportunidade de trazer inovação ao mercado e superar dificuldades futuras, como o crescimento da população e a procura de empregos fora do setor primário.

Este projeto enquadra-se nesta área e finda trazer inovação e contribuição para projetos futuros relacionados. Trata-se da criação de um modelo de deteção de troncos de pessegueiros em filas comuns de um pomar, para fins de navegação autónoma e sistema de anti-choque auxiliar, de uma plataforma robótica terrestre. Pretendeu-se formar a estratégia de orientação à volta da interação de dois troncos detetados simultaneamente. Esta deteção dupla permite a criação de uma função linear e sua linha imaginária. A partir do valor do declive desta e do desvio horizontal das caixas delimitadoras de troncos detetados singularmente, o algoritmo encaminha ordens aos pins do sistema robótico que controlam a locomoção e viragem do robô.

Após uma investigação aprofundada e revisto o estado da arte que envolve este problema, entre sistema de GPS, sensores LiDAR, odometria, entre outros sistemas abordados, optou-se pela deteção de objetos a partir de Redes Neuronais Convolucionais, devido à crescente popularidade desta área e ao ser um projeto inovador e com ambição de fazer parte de inspirações ou investigações futuras.

De modo a não se construir um modelo de deteção de raiz, optou-se por uma abordagem de aprendizagem por transferência. Após uma investigação para reconhecer o melhor sistema de deteção para o caso, um sistema SSD *MobileNet* com resolução de input 640x640 treinado com base de dados COCO 2017 foi o considerado a melhor alternativa. Este sistema foi adaptado e é considerado a melhor opção para aplicações móveis devido ao seu reduzido peso computacional e rapidez de execução.

O modelo original já treinado tem uma estrutura de 32 bits com palavras flutuantes. Tal estrutura não é ideal para aplicações em dispositivos móveis, ficando muito lento e sobrecarregando o aparelho, aquecendo-o, podendo danificar irreparavelmente. Para evitar tal situação, procedeu-se à quantização do modelo. Das demais quantizações que existem, a quantização integral completa foi a utilizada. Esta reconfiguração do modelo é a mais vincada. Necessita de uma base de dados representativa, e limita as operações, parâmetros e ativações de *32float* para *uint8*. Isto permite que o modelo corra muito mais suavemente no *Raspberry Pi 4*. Para reforçar a boa execução do modelo, foi compilado para compatibilidade com *Edge TPU*, uma unidade de processamento paralela ao microcomputador que auxilia o CPU nas operações.

As métricas avaliam o modelo pós-treino original e quantiado. Os valores de precisão do modelo original reduziram-se de 94,4% para 92,3%, e os valores de revocação de 94,4% para 66,7%. Os valores de precisão mostram-se praticamente inalterados e consideraram-se mais que aceitáveis para a aplicação. A queda da revocação é um pouco mais significativa e deve-se, certamente, à quantização integral completa.

As simulações realizadas, tanto em vídeo como em imagens, comprovaram que o modelo corresponde aos objetivos propostos. As indicações dadas foram corretas, tanto na condição do declive como do desvio de uma deteção única. No geral, o modelo, durante os vídeos detetou grande parte do tempo dois troncos, atualizando sempre a função linear de referência e, quando não detetava dois troncos, a deteção singular era suficiente para indicar à plataforma robótica para se manter na trajetória correta.

O tempo de inferência no *Raspberry Pi 4* foi de, aproximadamente, 0,37 segundos. Esta fração de tempo é suficientemente curta e dá uma margem aceitável ao robô para se ajustar à trajetória e cumprir as ordens dadas, sem grande atraso entre a indicação e a ação.

5.2. Sugestões de trabalhos futuros

Devido à complexidade e ao tempo reservado para o desenvolvimento do algoritmo, treino de modelos e criação de comandos necessários a conversão e inferência, existem aspetos que podem ser abordados futuramente.

A questão de testar o algoritmo no terreno é crucial, uma vez que só assim os problemas relacionados com os agentes exteriores imprevisíveis se poderão aferir e corrigir. Testes do sistema de navegação autónoma em diferentes alturas do ano e diversas condições climatéricas são fundamentais para avaliar a precisão do modelo face às variações induzidas na iluminação como nas cores de troncos, árvores, solo, entre outros. Verificando todas estas condições, o algoritmo e o modelo devem ser ajustados às várias situações, tornando o robô não só autónomo como extremamente versátil.

Adaptar o modelo a outras culturas para além de pessegueiros expandia a prática da navegação desta plataforma e a execução das suas tarefas, contribuindo, de uma maneira mais vasta, para o desenvolvimento na área da robótica aplicada na agricultura.

Havendo sempre margem de aperfeiçoamento, outras abordagens de outros sistemas de deteção com configurações diferentes do presente neste projeto – o SSD *MobileNet* – poderiam ser testadas, podendo formar um modelo de deteção de troncos mais preciso, revocado, rápido ou mais leve computacionalmente.

Como foi referido durante o capítulo Estado da Arte, muitos dos projetos de navegação autónoma aplicados na agricultura são união de dois sistemas diferentes, de modo a garantir uma navegação com o mínimo de erro permitido. Este sistema poderia ser conciliado com outro, como por exemplo de Posicionamento Global por Satélite (GPS), tornando a navegação do robô muito mais precisa.

Referências Bibliográficas

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., … Zheng, X. (2016). Tensorflow: A System for Large-Scale Machine Learning. *Proceeding of the 12th USENIX Symposium on Operating Systems Design and Implementation3*, 265–283.

Assunção, E., Gaspar, P. D., Mesquita, R., Veiros, A., & Proença, H. (2020). Resultados preliminares de deteção de imagens de pêssegos aplicando o método Faster R-CNN. *Revista Da Associação Portuguesa de Horticultura*, 1–7.

Auat Cheein, F., Steiner, G., Perez Paina, G., & Carelli, R. (2011). Optimized EIF-SLAM algorithm for precision agriculture mapping based on stems detection. *Computers and Electronics in Agriculture*, *78*(2), 195–207. https://doi.org/10.1016/j.compag.2011.07.007

Barawid, O. C., Mizushima, A., Ishii, K., & Noguchi, N. (2007). Development of an Autonomous Navigation System using a Two-dimensional Laser Scanner in an Orchard Application. *Biosystems Engineering*, *96*(2), 139–149. https://doi.org/10.1016/j.biosystemseng.2006.10.012

Bayar, G., Bergerman, M., Koku, A. B., & Konukseven, E. I. (2015). Localization and control of an autonomous orchard vehicle. *Computers and Electronics in Agriculture*, *115*, 118–128. https://doi.org/10.1016/j.compag.2015.05.015

Bechar, A., & Vigneault, C. (2016). Agricultural robots for field operations: Concepts and components. *Biosystems Engineering*, *149*, 94–111. https://doi.org/10.1016/j.biosystemseng.2016.06.014

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer. https://doi.org/10.13109/9783666604409.185

Blok, P. M., van Boheemen, K., van Evert, F. K., IJsselmuiden, J., & Kim, G. H. (2019). Robot navigation in orchards with localization based on Particle filter and Kalman filter. *Computers and Electronics in Agriculture*, *157*(January), 261–269. https://doi.org/10.1016/j.compag.2018.12.046

Bonadies, S., & Gadsden, S. A. (2019). An overview of autonomous crop row navigation strategies for unmanned ground vehicles. *Engineering in Agriculture, Environment and Food*, *12*(1), 24–31. https://doi.org/10.1016/j.eaef.2018.09.001

Calicioglu, O., Flammini, A., Bracco, S., Bellù, L., & Sims, R. (2019). The future challenges of food and agriculture: An integrated analysis of trends and solutions. *Sustainability (Switzerland)*, *11*(1). https://doi.org/10.3390/su11010222

Canziani, A., Paszke, A., & Culurciello, E. (2016). *An Analysis of Deep Neural Network Models for Practical Applications*. 1–7. http://arxiv.org/abs/1605.07678

Cernicchiaro, C., Gaspar, P. D., & Aguiar, M. L. (2019). *Fast Return Path Planning for Agricultural Autonomous Terrestrial Robot in a Known Field*. *13*(2), 79–87.

Chen, X., Wang, S., Zhang, B., & Luo, L. (2018). Multi-feature fusion tree trunk detection and orchard mobile robot localization using camera/ultrasonic sensors. *Computers and Electronics in Agriculture*, *147*(December 2017), 91–108. https://doi.org/10.1016/j.compag.2018.02.009

Chollet, F. (2017). *Deep Learning with Python* (Vol. 53, Issue 9). Manning Publications.

Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., & Schmidhuber, J. (2011). Flexible, High Performance Convolutional Neural Networks for Image Classification. *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, *22*, 1237–1242.

Csillik, O., Cherbini, J., Johnson, R., Lyons, A., & Kelly, M. (2018). Identification of citrus trees from unmanned aerial vehicle imagery using convolutional neural networks. *Drones*, *2*(4), 1–16. https://doi.org/10.3390/drones2040039

Duckett, T., Pearson, S., Blackmore, S., Grieve, B., Chen, W.-H., Cielniak, G., Cleaversmith, J., Dai, J., Davis, S., Fox, C., From, P., Georgilas, I., Gill, R., Gould, I., Hanheide, M., Hunter, A., Iida, F., Mihalyova, L., Nefti-Meziani, S., … Yang, G.-Z. (2018). *Agricultural Robotics: The Future of Robotic Agriculture*. http://arxiv.org/abs/1806.06762

Edlerman, E., & Linker, R. (2019). Autonomous multi-robot system for use in vineyards and orchards. *27th Mediterranean Conference on Control and Automation, MED 2019 - Proceedings*, 274–279. https://doi.org/10.1109/MED.2019.8798538

Fricker, G. A., Ventura, J. D., Wolf, J. A., North, M. P., Davis, F. W., & Franklin, J. (2019). A convolutional neural network classifier identifies tree species in mixed-conifer forest from hyperspectral imagery. *Remote Sensing*, *11*(19). https://doi.org/10.3390/rs11192326

Galileo — Sistema de Navegação por Satélite Europeu | European Global Navigation Satellite Systems Agency. (n.d.). Retrieved March 24, 2021, from https://www.gsa.europa.eu/galileo-—-sistema-de-navegação-por-satélite-europeu

Gao, X., Li, J., Fan, L., Zhou, Q., Yin, K., Wang, J., Song, C., Huang, L., & Wang, Z. (2018). Review of wheeled mobile robots' navigation problems and application prospects in agriculture. *IEEE Access*, *6*, 49248–49268. https://doi.org/10.1109/ACCESS.2018.2868848

Gaspar, P. D., Simões, M. P., Ramos, A., Assunção, E., Mesquita, R., & Veiras, A. (2020). *PrunusBOT - Sistema robótico aéreo autónomo de pulverização controlada e previsão de produção frutícola*. https://goprunus.wixsite.com/prunoideas

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media.

Gonzalez-Huitron, V., León-Borges, J. A., Rodriguez-Mata, A. E., Amabilis-Sosa, L. E., Ramírez-Pereda, B., & Rodriguez, H. (2021). Disease detection in tomato leaves via CNN with lightweight architectures implemented in Raspberry Pi 4. *Computers and Electronics in Agriculture*, *181*(January), 1–9. https://doi.org/10.1016/j.compag.2020.105951

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Gupta, I., Patil, V., Kadam, C., & Dumbre, S. (2016). Face detection and recognition using Raspberry Pi. *WIECON-ECE 2016 - 2016 IEEE International WIE Conference on Electrical and Computer Engineering*, 83–86. https://doi.org/10.1109/WIECON-ECE.2016.8009092

Hiremath, S. A., van der Heijden, G. W. A. M., van Evert, F. K., Stein, A., & Ter Braak, C. J. F. (2014). Laser range finder model for autonomous navigation of a robot in a maize field using a particle filter. *Computers and Electronics in Agriculture*, *100*, 41–50. https://doi.org/10.1016/j.compag.2013.10.005

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. http://arxiv.org/abs/1704.04861

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., & Murphy, K. (2017). Speed/accuracy trade-offs for modern convolutional object detectors. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017-Janua*, 3296–3305. https://doi.org/10.1109/CVPR.2017.351

Hui, J. (2018). *Understanding Feature Pyramid Networks for object detection (FPN)*. Medium. https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c

JetBrains. (n.d.). *PyCharm*. Retrieved July 26, 2021, from https://www.jetbrains.com/pycharm/

Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). A review of the use of convolutional neural networks in agriculture. *Journal of Agricultural Science*, *156*(3), 312–322. https://doi.org/10.1017/S0021859618000436

Khan, N., Medlock, G., Graves, S., & Anwar, S. (2018). GPS Guided Autonomous Navigation of a Small Agricultural Robot with Automated Fertilizing System. *SAE Technical Papers*, *2018-April*, 1–7. https://doi.org/10.4271/2018-01-0031

Khandelwal, R. (2020, June 15). A Basic Introduction to TensorFlow Lite. *Towards Data Science*. https://towardsdatascience.com/a-basic-introduction-to-tensorflow-lite-59e480c57292

Lepej, P., & Rakun, J. (2016). Simultaneous localisation and mapping in a complex field environment. *Biosystems Engineering*, *150*, 160–169. https://doi.org/10.1016/j.biosystemseng.2016.08.004

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9905 LNCS*, 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

Liu, X., Zhao, D., Jia, W., Ruan, C., Tang, S., & Shen, T. (2016). A method of segmenting apples at night based on color and position information. *Computers and Electronics in Agriculture*, *122*, 118–123. https://doi.org/10.1016/j.compag.2016.01.023

Lowe, D. (2017). *Electronics All-in-One For Dummies* (2nd ed.). John Wiley & Sons, Inc.

Majeed, Y., Zhang, J., Zhang, X., Fu, L., Karkee, M., Zhang, Q., & Whiting, M. D. (2018). Apple Tree Trunk and Branch Segmentation for Automatic Trellis Training Using Convolutional Neural Network Based Semantic Segmentation. *IFAC-PapersOnLine*, *51*(17), 75–80. https://doi.org/10.1016/j.ifacol.2018.08.064

Marden, S., & Whitty, M. (2014). GPS-free Localisation and Navigation of an Unmanned Ground Vehicle for Yield Forecasting in a Vineyard. *Proceedings of the 13th International Conference IAS-13*.

Marinoudi, V., Sørensen, C. G., Pearson, S., & Bochtis, D. (2019). Robotics and labour in agriculture. A context consideration. *Biosystems Engineering*, *184*, 111–121. https://doi.org/10.1016/j.biosystemseng.2019.06.013

Nations, U. (n.d.). *Goal 2 | End hunger, achieve food security and improved nutrition and promote sustainable agriculture*. Retrieved March 16, 2021, from https://sdgs.un.org/goals/goal2

Nicknochnack. (2021). *GitHub - nicknochnack/GenerateTFRecord*. https://github.com/nicknochnack/GenerateTFRecord

O'Shea, K., & Nash, R. (2015). *An Introduction to Convolutional Neural Networks*. 1–11. http://arxiv.org/abs/1511.08458

Pallottino, F., Antonucci, F., Costa, C., Bisaglia, C., Figorilli, S., & Menesatti, P. (2019). Optoelectronic proximal sensing vehicle-mounted technologies in precision agriculture: A review. *Computers and Electronics in Agriculture*, *162*(March), 859–873. https://doi.org/10.1016/j.compag.2019.05.034

Pi, R. (n.d.). *Camera Module 2*. Retrieved September 17, 2021, from https://www.raspberrypi.org/products/camera-module-v2/

PORDATA - População empregada: total e por grandes sectores de actividade económica. (n.d.). Retrieved March 16, 2021, from https://www.pordata.pt/Portugal/População+empregada+total+e+por+grandes+sectores+de+actividade+económica-32-2743

Ramos, A. R., & Gaspar, P. D. (2019). Algorithm for path recognition in-between tree rows for agricultural wheeled-mobile robots. *International Journal of Mechanical and Mechatronics Engineering*, *13*(1), 34–37.

Rejas, J. I., Sanchez, A., Glez-De-Rivera, G., Prieto, M., & Garrido, J. (2015). Environment mapping using a 3D laser scanner for unmanned ground vehicles. *Microprocessors and Microsystems*, *39*(8), 939–949. https://doi.org/10.1016/j.micpro.2015.10.003

Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., & Savarese, S. (2019). Generalized intersection over union: A metric and a loss for bounding box regression. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2019-June*, 658–666. https://doi.org/10.1109/CVPR.2019.00075

Sahni, M. (2018). 8-Bit Quantization and TensorFlow Lite: Speeding up mobile inference with low precision. *Heartbeat*. https://heartbeat.comet.ml/8-bit-quantization-and-tensorflow-lite-speeding-up-mobile-inference-with-low-precision-a882dfcafbbd

Samuel, A. L. (1988). Some Studies in Machine Learning Using the Game of Checkers. II—Recent Progress. *Computer Games I*, *44*(1), 366–400. https://doi.org/10.1007/978-1-4613-8716-9_15

Santra, A., Mahato, S., Dan, S., & Bose, A. (2019). Precision of satellite based navigation position solution: A review using NavIC data. *Journal of Information and Optimization Sciences*, *40*(8), 1683–1691. https://doi.org/10.1080/02522667.2019.1703264

Scarpino, M. (2018). *TensorFlow For Dummies*. John Wiley & Sons, Inc.

Shalal, N., Low, T., McCarthy, C., & Hancock, N. (2013). *A preliminary evaluation of vision and laser sensing for tree trunk detection and orchard mapping BT - 2013 Australasian Conference on Robotics and Automation, ACRA 2013, December 2, 2013 - December 4, 2013*. 2–4.

Shalal, N., Low, T., McCarthy, C., & Hancock, N. (2015a). Orchard mapping and mobile robot localisation using on-board camera and laser scanner data fusion - Part A: Tree detection. *Computers and Electronics in Agriculture*, *119*, 254–266. https://doi.org/10.1016/j.compag.2015.09.025

Shalal, N., Low, T., McCarthy, C., & Hancock, N. (2015b). Orchard mapping and mobile robot localisation using on-board camera and laser scanner data fusion - Part B: Mapping and localisation. *Computers and Electronics in Agriculture*, *119*, 267–278. https://doi.org/10.1016/j.compag.2015.09.026

Shi, W., van de Zedde, R., Jiang, H., & Kootstra, G. (2019). Plant-part segmentation using deep learning and multi-view vision. *Biosystems Engineering*, *187*, 81–95. https://doi.org/10.1016/j.biosystemseng.2019.08.014

Shukla, N. (2018). *Machine Learning with TensorFlow*. Manning Publications.

Solawetz, J. (2020). *An Introduction to the COCO Dataset*. Roboflow. https://blog.roboflow.com/coco-dataset/

Sumardi, S., Taufiqurrahman, M., & Riyadi, M. A. (2018). Street Mark Detection Using Raspberry PI for Self-Driving System. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, *16*(2), 629–634. https://doi.org/10.12928/telkomnika.v16i1.4509

Tensorflow. (n.d.-a). *GitHub - tensorflow/models/research/object_detection*. Retrieved July 20, 2021, from https://github.com/tensorflow/models/tree/master/research/object_detection

Tensorflow. (n.d.-b). *GitHub - tensorflow/models*. Retrieved September 13, 2021, from https://github.com/tensorflow/models

Tensorflow. (n.d.-c). *Post-training quantization | TensorFlow Lite*. Tensorflow. Retrieved September 21, 2021, from https://www.tensorflow.org/lite/performance/post_training_quantization

Tzutalin. (2018). *GitHub - tzutalin/labelImg*. https://github.com/tzutalin/labelImg

Underwood, J. P., Hung, C., Whelan, B., & Sukkarieh, S. (2016). Mapping almond orchard canopy volume, flowers, fruit and yield using lidar and vision sensors. *Computers and Electronics in Agriculture*, *130*, 83–96. https://doi.org/10.1016/j.compag.2016.09.014

Underwood, J. P., Jagbrant, G., Nieto, J. I., & Sukkarieh, S. (2015). Lidar-Based Tree Recognition and Platform Localization in Orchards. *Journal of Field Robotics*, *32*(8), 1056–1074. https://doi.org/10.1002/rob.21607

Veiros, A. (2020). *Sistema robótico terrestre para apoio a atividades de manutenção de solo em pomares de prunóideas*. Universidade da Beira Interioer.

Véstias, M. P. (2021). Convolutional Neural Networks. In M. Khosrow-Pour D.B.A. (Ed.), *Encyclopedia of Information Science and Technology* (5th ed., p. 12). IGI Global. https://doi.org/10.4018/978-1-7998-3479-3

Wazwaz, A. A., Herbawi, A. O., Teeti, M. J., & Hmeed, S. Y. (2018). Raspberry Pi and computers-based face detection and recognition system. *2018 4th International Conference on Computer and Technology Applications, ICCTA 2018*, 171–174. https://doi.org/10.1109/CATA.2018.8398677

Wisnudhanti, K., & Candra, F. (2020). Image Classification of Pandawa Figures Using Convolutional Neural Network on Raspberry Pi 4. *Journal of Physics: Conference Series*, *1655*(1), 1–8. https://doi.org/10.1088/1742-6596/1655/1/012103

Yandún Narváez, F. J., Salvo del Pedregal, J., Prieto, P. A., Torres-Torriti, M., & Auat Cheein, F. A. (2016). LiDAR and thermal images fusion for ground-based 3D characterisation of fruit trees. *Biosystems Engineering*, *151*, 479–494. https://doi.org/10.1016/j.biosystemseng.2016.10.012

Yousif, K., Bab-Hadiashar, A., & Hoseinnezhad, R. (2015). An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics. *Intelligent Industrial Systems*, *1*(4), 289–311. https://doi.org/10.1007/s40903-015-0032-7

Zack, R. I. (2015). *Row Following and Turning Through Lidar-based Autonomous Navigation in an Orchard Environment*. 153. https://search.proquest.com/docview/1775393687?accountid=8113

Zaidner, G., & Shapiro, A. (2016). A novel data fusion algorithm for low-cost localisation and navigation of autonomous vineyard sprayer robots. *Biosystems Engineering*, *146*, 133–148. https://doi.org/10.1016/j.biosystemseng.2016.05.002

Zhai, Z., Martínez, J. F., Beltran, V., & Martínez, N. L. (2020). Decision support systems for agriculture 4.0: Survey and challenges. *Computers and Electronics in Agriculture*, *170*(January), 105256. https://doi.org/10.1016/j.compag.2020.105256

Zhang, C., Noguchi, N., & Yang, L. (2016). Leader-follower system using two robot tractors to improve work efficiency. *Computers and Electronics in Agriculture*, *121*, 269–281. https://doi.org/10.1016/j.compag.2015.12.015

Zhuang, J., Hou, C., Tang, Y., He, Y., Guo, Q., Zhong, Z., & Luo, S. (2019). Computer vision-based localisation of picking points for automatic litchi harvesting applications towards natural scenarios. *Biosystems Engineering*, *187*, 1–20. https://doi.org/10.1016/j.biosystemseng.2019.08.016

6. Anexos

```
model {
  ssd {
    num_classes: 1
    image_resizer {
      fixed_shape_resizer {
        height: 640
        width: 640
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 4e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.01
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.997
          scale: true
          epsilon: 0.001
        }
      }
      use_depthwise: true
      override_base_feature_extractor_hyperparams: true
      fpn {
        min_level: 3
        max_level: 7
        additional_layer_depth: 128
      }
    }
    box_coder {
      faster_rcnn_box_coder {
        y_scale: 10.0
        x_scale: 10.0
        height_scale: 5.0
        width_scale: 5.0
      }
    }
    matcher {
      argmax_matcher {
        matched_threshold: 0.5
        unmatched_threshold: 0.5
        ignore_thresholds: false
        negatives_lower_than_unmatched: true
        force_match_for_each_row: true
```

```
      use_matmul_gather: true
    }
  }
  similarity_calculator {
    iou_similarity {
    }
  }
  box_predictor {
    weight_shared_convolutional_box_predictor {
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 4e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.01
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.997
          scale: true
          epsilon: 0.001
        }
      }
      depth: 128
      num_layers_before_predictor: 4
      kernel_size: 3
      class_prediction_bias_init: -4.6
      share_prediction_tower: true
      use_depthwise: true
    }
  }
  anchor_generator {
    multiscale_anchor_generator {
      min_level: 3
      max_level: 7
      anchor_scale: 4.0
      aspect_ratios: 1.0
      aspect_ratios: 2.0
      aspect_ratios: 0.5
      scales_per_octave: 2
    }
  }
  post_processing {
    batch_non_max_suppression {
      score_threshold: 1e-08
      iou_threshold: 0.6
      max_detections_per_class: 100
      max_total_detections: 100
      use_static_shapes: false
    }
    score_converter: SIGMOID
  }
  normalize_loss_by_num_matches: true
  loss {
    localization_loss {
```

```
        weighted_smooth_l1 {
        }
      }
      classification_loss {
        weighted_sigmoid_focal {
          gamma: 2.0
          alpha: 0.25
        }
      }
      classification_weight: 1.0
      localization_weight: 1.0
    }
    encode_background_as_zeros: true
    normalize_loc_loss_by_codesize: true
    inplace_batchnorm_update: true
    freeze_batchnorm: false
  }
}
train_config {
  batch_size: 8
  data_augmentation_options {
    random_horizontal_flip {
    }
  }
  data_augmentation_options {
    random_crop_image {
      min_object_covered: 0.0
      min_aspect_ratio: 0.75
      max_aspect_ratio: 3.0
      min_area: 0.75
      max_area: 1.0
      overlap_thresh: 0.0
    }
  }
  sync_replicas: true
  optimizer {
    momentum_optimizer {
      learning_rate {
        cosine_decay_learning_rate {
          learning_rate_base: 0.08
          total_steps: 2000
          warmup_learning_rate: 0.026666
          warmup_steps: 1000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
  fine_tune_checkpoint:                                    "pre-trained-
models/ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8/checkpoint/ckpt-
0"
  num_steps: 8000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "detection"
  fine_tune_checkpoint_version: V2
}
```

```
train_input_reader {
  label_map_path: "annotations/label_map.pbtxt"
  tf_record_input_reader {
    input_path: "annotations/train.record"
  }
}
eval_config {
  metrics_set: "coco_detection_metrics"
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/test.record"
  }
}
```

Figura 44: Código de configuração do treino do modelo SSD *MobileNet* com aprendizagem por transferência COCO17 adaptado ao modelo em questão.

```
import os
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util
import cv2
import numpy as np

# Paths & Files
paths = {
    'video_path': os.path.join('Videos'),
    'checkpoint_path': os.path.join('checkpoint')
}
files = {
    'label_map': os.path.join('label_map.pbtxt'),
    'pipeline_config': os.path.join('pipeline.config')
}

## Load Train Model From checkpoint
# Load pipeline config and build a detection model
configs                                                              =
config_util.get_configs_from_pipeline_file(files['pipeline_config'])
detection_model  =  model_builder.build(model_config=configs['model'],
is_training=False)
# Restore Checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['checkpoint_path'],             'ckpt-
0')).expect_partial()


@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
```

```
category_index                                                    =
label_map_util.create_category_index_from_labelmap(files['label_map'])
video_path = os.path.join(paths['video_path'], '2.mp4')

cap = cv2.VideoCapture(video_path)
imW = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
imH = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

while cap.isOpened():
    ret, frame = cap.read()
    if imW != imH:
        frame = frame[0:imH, 0:imH]
    frame = cv2.resize(frame, (640, 640))
    image_np = np.array(frame)

    input_tensor  =  tf.convert_to_tensor(np.expand_dims(image_np,  0),
dtype=tf.float32)
    detections = detect_fn(input_tensor)

    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    detections['detection_classes']                                =
detections['detection_classes'].astype(np.int64)

    label_id_offset = 1
    image_np_width_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
        image_np_width_detections,
        detections['detection_boxes'],
        detections['detection_classes'] + label_id_offset,
        detections['detection_scores'],
        category_index,
        use_normalized_coordinates=True,
        max_boxes_to_draw=3,
        min_score_thresh=.12,
        agnostic_mode=False)

    boxes = np.squeeze(detections['detection_boxes'])
    scores = np.squeeze(detections['detection_scores'])
    min_score_thresh = 0.12
    bboxes = boxes[scores > min_score_thresh]
    final_box = []
    for box in bboxes:
        ymin, xmin, ymax, xmax = box
        final_box.append([xmin * 640, xmax * 640, ymin * 640, ymax *
640])
    final_box = np.round(final_box)
    # COORDENADAS: Xmin, Xmax, Ymin, Ymax
    # Ponto a considerar: (Xmax, Ymax)

    if final_box.size >= 8:
        tree1 = final_box[0][:]
        tree2 = final_box[1][:]
        Ax = int(tree1[1])
        Ay = int(tree1[3])
```

```
        Bx = int(tree2[1])
        By = int(tree2[3])

        # y = mx + b
        m = (By - Ay) / (Bx - Ax)
        b = Ay - m * Ax

        cv2.line(image_np_width_detections, (int(-b / m), 0), (int((640-
b)/m), 640), (255,255,255), thickness=2)

    if 'm' in locals():
        if final_box.size >= 8:
        ## ORIENTAÇAO PELO DECLIVE
            mmin = -1
            mmax = -.2

            if m < mmin:
                print('virar à esquerda')
            elif m > mmax:
                print('virar à direita')
            else:
                print('andar em frente')
        elif final_box.size == 4:
        ## ORIENTACAO PELO DESVIO DE X
            X = int(final_box[0][1])
            Y = int(final_box[0][3])
            xref = (Y - b)/m
            if X < xref-150:
                print('virar à esquerda')
            elif X > xref+150:
                print('virar à direita')
            else:
                print('andar em frente')
        else:
            print('andar em frente')

        cv2.line(image_np_width_detections, (int(-b / m), 0), (int((640-
b)/m), 640), (255,255,255), thickness=2)
    else:
        print('andar em frente')

    cv2.imshow('object detection', image_np_width_detections)

    if cv2.waitKey(10) & 0xFF == ord('q'):
        cap.release()
        cv2.destroyAllWindows()
        break
```

Figura 45. Amostra do código de deteção de objetos para um modelo não quantiado, tenstando um vídeo, com resposta em texto das ordens de viragem (*Tensorflow*).

```
import tensorflow as tf
import os
import numpy as np
import pathlib
import glob
import cv2
import glob

# Carregamento e conversão das imagens de treino em vetores de 32bits
```

```
cv_img = []
for img in glob.glob("train/*.jpeg"):
    n = cv2.imread(img)
    n = cv2.resize(cv2.cvtColor(n, cv2.COLOR_BGR2RGB), (640, 640))
    n = n.astype(np.float32)/255
    cv_img.append(n)
cv_img = np.asarray(cv_img)
print(cv_img)
print(type(cv_img))

saved_model_dir = os.getcwd()

# Definição da função dos dados representativos recorrendo às imagens
de treino
def representative_data_gen():
  for                          input_value                          in
tf.data.Dataset.from_tensor_slices(cv_img).batch(1).take(89):
    yield [input_value]

# Comando de conversao em uint8
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_dir)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
converter.representative_dataset = representative_data_gen
converter.target_spec.supported_ops                                    =
[tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.uint8
converter.inference_output_type = tf.uint8
tflite_model = converter.convert()

#Confirmação do tipo de formato dos input e output (se foram convertidos
para 8bit)
interpreter = tf.lite.Interpreter(model_content=tflite_model)
input_type = interpreter.get_input_details()[0]['dtype']
print('input: ', input_type)
output_type = interpreter.get_output_details()[0]['dtype']
print('output: ', output_type)

# Guardar o modelo convertido
open("detect.tflite", "wb").write(tflite_model)
```

Figura 46: Amostra do código de conversão do modelo *Tensorflow* para *Tensorflow Lite* com quantiação integral completa uint8.

```
import os
import argparse
import cv2
import numpy as np
import sys
import importlib.util

MODEL_NAME = os.getcwd()
GRAPH_NAME = 'uint8detect_edgetpu.tflite'
LABELMAP_NAME = 'labels.txt'
VIDEO_NAME = 'Videos/6.mp4'
set_threshold = 0.16
use_TPU = True

# Import TensorFlow libraries
#   If   tflite_runtime   is   installed,   import   interpreter   from
tflite_runtime, else import from regular tensorflow
```

```
# If using Coral Edge TPU, import the load_delegate library
pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
    from tflite_runtime.interpreter import Interpreter
    if use_TPU:
        from tflite_runtime.interpreter import load_delegate
else:
    from tensorflow.lite.python.interpreter import Interpreter
    if use_TPU:
        from tensorflow.lite.python.interpreter import load_delegate

# Get path to current working directory
CWD_PATH = os.getcwd()

# Path to video file
VIDEO_PATH = os.path.join(CWD_PATH,VIDEO_NAME)

# Path to .tflite file, which contains the model that is used for object
detection
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

# Load the label map
with open(PATH_TO_LABELS, 'r') as f:
    labels = [line.strip() for line in f.readlines()]

# Have to do a weird fix for label map if using the COCO "starter model"
from
# https://www.tensorflow.org/lite/models/object_detection/overview
# First label is '???', which has to be removed.
if labels[0] == '???':
    del(labels[0])

# Load the Tensorflow Lite model.
# If using Edge TPU, use special load_delegate argument
if use_TPU:
    interpreter = Interpreter(model_path=PATH_TO_CKPT,

experimental_delegates=[load_delegate('libedgetpu.so.1.0')])
    print(PATH_TO_CKPT)
else:
    interpreter = Interpreter(model_path=PATH_TO_CKPT)

interpreter.allocate_tensors()

# Get model details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
# print('output_details:', output_details)
# print('input_details:', input_details)
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]

# Open video file
video = cv2.VideoCapture(VIDEO_PATH)
imW = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
imH = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
```

```
while(video.isOpened()):

    # Acquire frame and resize to expected shape [1xHxWx3]
    ret, frame = video.read()
    if not ret:
      print('Reached the end of the video!')
      break
    if imW != imH:
        frame = frame[0:imH, 0:imH]
    frame_resized = cv2.resize(frame, (width, height))
    frame_resized = cv2.cvtColor(frame_resized, cv2.COLOR_BGR2RGB)
    output_scale, output_zero_point = output_details[0]["quantization"]

    input_data             =             np.expand_dims(frame_resized,
axis=0).astype(input_details[0]['dtype'])

    # Perform the actual detection by running the model with the image
as input
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()

    # Retrieve detection results
    boxes  =  interpreter.get_tensor(output_details[0]['index'])[0]  #
Bounding box coordinates of detected objects
    classes  =  interpreter.get_tensor(output_details[1]['index'])[0]  #
Class index of detected objects
    scores  =  interpreter.get_tensor(output_details[2]['index'])[0]  #
Confidence of detected objects
    threshold = int(set_threshold / output_scale + output_zero_point)
    final_box = []

    # Loop over all detections and draw detection box if confidence is
above minimum threshold
    for i in range(len(scores)):
        if (scores[i] > threshold):

            # Get bounding box coordinates and draw box
            ymin = boxes[i][0]
            xmin = boxes[i][1]
            ymax = boxes[i][2]
            xmax = boxes[i][3]
            # Interpreter can return coordinates that are outside of
image dimensions, need to force them to be within image using max() and
min()
            ymin = int(max(1, (output_scale * (ymin - output_zero_point)
* height)))
            xmin = int(max(1,(output_scale * (xmin - output_zero_point)
* width)))
            ymax   =   int(min(height,   (output_scale   *   (ymax   -
output_zero_point) * height)))
            xmax   =   int(min(width,   (output_scale   *   (xmax   -
output_zero_point) * width)))

            cv2.rectangle(frame_resized, (xmin,ymin), (xmax,ymax), (10,
255, 0), 4)
            final_box.append([xmin, xmax, ymin, ymax])

    final_box = np.round(final_box)

    if final_box.size >= 8:
```

```
        tree1 = final_box[0][:]
        tree2 = final_box[1][:]
        Ax = int(tree1[1])
        Ay = int(tree1[3])
        Bx = int(tree2[1])
        By = int(tree2[3])

        # y = mx + b
        m = (By - Ay) / (Bx - Ax)
        b = Ay - m * Ax

        cv2.line(frame_resized, (int(-b / m), 0), (int((height-b)/m),
int(height)), (255,255,255), thickness=2)

    if 'm' in locals():
        if final_box.size >= 8:
        ## ORIENTACAO PELO DECLIVE
            mmin = -1
            mmax = -.2

            if m < mmin:
                print('virar esquerda')
            elif m > mmax:
                print('virar direita')
            else:
                print('andar em frente')
        elif final_box.size == 4:
        ## ORIENTACAO PELO DESVIO DE X
            X = int(final_box[0][1])
            Y = int(final_box[0][3])
            xref = (Y - b)/m
            if X < xref-150:
                print('virar esquerda')
            elif X > xref+150:
                print('virar direita')
            else:
                print('andar em frente')
        else:
            print('andar em frente')

        cv2.line(image_resized, (int(-b / m), 0), (int((height-b)/m),
int(height)), (255,255,255), thickness=2)
    else:
        print('andar em frente')

# All the results have been drawn on the frame, so it's time to display
it.
    frame_resized = cv2.cvtColor(frame_resized, cv2.COLOR_RGB2BGR)
    cv2.imshow('Object detector', frame_resized)

    # Press 'q' to quit
    if cv2.waitKey(1) == ord('q'):
        break

# Clean up
video.release()
```

```
cv2.destroyAllWindows()
```

Figura 47. Algoritmo de inferência do modelo *TFLite* em *EdgeTPU* num ficheiro de vídeo.

```
import os
import argparse
import cv2
import numpy as np
import sys
import time
from threading import Thread
import importlib.util

# Define VideoStream class to handle streaming of video from webcam in
separate processing thread
#       Source       -       Adrian       Rosebrock,       PyImageSearch:
https://www.pyimagesearch.com/2015/12/28/increasing-raspberry-pi-fps-
with-python-and-opencv/
class VideoStream:
    """Camera object that controls video streaming from the Picamera"""
    def __init__(self,resolution=(640,480),framerate=30):
        # Initialize the PiCamera and the camera image stream
        self.stream = cv2.VideoCapture(0)
        ret            =            self.stream.set(cv2.CAP_PROP_FOURCC,
cv2.VideoWriter_fourcc(*'MJPG'))
        ret = self.stream.set(3,resolution[0])
        ret = self.stream.set(4,resolution[1])

        # Read first frame from the stream
        (self.grabbed, self.frame) = self.stream.read()

      # Variable to control when the camera is stopped
        self.stopped = False

    def start(self):
      # Start the thread that reads frames from the video stream
        Thread(target=self.update,args=()).start()
        return self

    def update(self):
        # Keep looping indefinitely until the thread is stopped
        while True:
            # If the camera is stopped, stop the thread
            if self.stopped:
                # Close camera resources
                self.stream.release()
                return

            # Otherwise, grab the next frame from the stream
            (self.grabbed, self.frame) = self.stream.read()

    def read(self):
      # Return the most recent frame
        return self.frame

    def stop(self):
      # Indicate that the camera and thread should be stopped
        self.stopped = True

MODEL_NAME = os.getcwd()
```

```
GRAPH_NAME = 'uint8detect_edgetpu.tflite'
LABELMAP_NAME = 'labels.txt'
set_threshold = 0.16
resW, resH = args.resolution.split('x')
imW, imH = 640, 640
use_TPU = True

IM_DIR = 'Imagens'

# Import TensorFlow libraries
#   If   tflite_runtime   is   installed,   import   interpreter   from
tflite_runtime, else import from regular tensorflow
# If using Coral Edge TPU, import the load_delegate library
pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
    from tflite_runtime.interpreter import Interpreter
    if use_TPU:
        from tflite_runtime.interpreter import load_delegate
else:
    from tensorflow.lite.python.interpreter import Interpreter
    if use_TPU:
        from tensorflow.lite.python.interpreter import load_delegate

# If using Edge TPU, assign filename for Edge TPU model
if use_TPU:
    # If user has specified the name of the .tflite file, use that name,
otherwise use default 'edgetpu.tflite'
    if (GRAPH_NAME == 'detect.tflite'):
        GRAPH_NAME = 'detect_edgetpu.tflite'

# Get path to current working directory
CWD_PATH = os.getcwd()

# Path to .tflite file, which contains the model that is used for object
detection
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)

# Load the label map
with open(PATH_TO_LABELS, 'r') as f:
    labels = [line.strip() for line in f.readlines()]

# Have to do a weird fix for label map if using the COCO "starter model"
from
# https://www.tensorflow.org/lite/models/object_detection/overview
# First label is '???', which has to be removed.
if labels[0] == '???':
    del(labels[0])

# Load the Tensorflow Lite model.
# If using Edge TPU, use special load_delegate argument
if use_TPU:
    interpreter = Interpreter(model_path=PATH_TO_CKPT,

experimental_delegates=[load_delegate('libedgetpu.so.1.0')])
    print(PATH_TO_CKPT)
else:
    interpreter = Interpreter(model_path=PATH_TO_CKPT)
```

```
interpreter.allocate_tensors()

# Get model details
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
# print('output_details:', output_details)
# print('input_details:', input_details)
height = input_details[0]['shape'][1]
width = input_details[0]['shape'][2]

# Initialize frame rate calculation
frame_rate_calc = 1
freq = cv2.getTickFrequency()

# Initialize video stream
videostream = VideoStream(resolution=(imW,imH),framerate=30).start()
time.sleep(1)

while True:

    # Start timer (for calculating frame rate)
    t1 = cv2.getTickCount()

    # Grab frame from video stream
    frame1 = videostream.read()

    # Acquire frame and resize to expected shape [1xHxWx3]
    frame = frame1.copy()
    frame_resized = cv2.resize(frame, (width, height))
    frame_resized = cv2.cvtColor(frame_resized, cv2.COLOR_BGR2RGB)
    input_data              =              np.expand_dims(frame_resized,
axis=0).astype(input_details[0]['dtype'])
    output_scale, output_zero_point = output_details[0]["quantization"]

    # Perform the actual detection by running the model with the image
as input
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()

    # Retrieve detection results
    boxes   =   interpreter.get_tensor(output_details[0]['index'])[0]   #
Bounding box coordinates of detected objects
    classes  =  interpreter.get_tensor(output_details[1]['index'])[0]  #
Class index of detected objects
    scores  =  interpreter.get_tensor(output_details[2]['index'])[0]  #
Confidence of detected objects
    threshold = int(set_threshold / output_scale + output_zero_point)
    final_box = []

    # Loop over all detections and draw detection box if confidence is
above minimum threshold
    for i in range(len(scores)):
        if (scores[i] > threshold):

            # Get bounding box coordinates and draw box
            ymin = boxes[i][0]
            xmin = boxes[i][1]
            ymax = boxes[i][2]
            xmax = boxes[i][3]
```

```
            # Interpreter can return coordinates that are outside of
image dimensions, need to force them to be within image using max() and
min()
            ymin = int(max(1, (output_scale * (ymin - output_zero_point)
* height)))
            xmin = int(max(1,(output_scale * (xmin - output_zero_point)
* width)))
            ymax   =   int(min(height,   (output_scale   *   (ymax   -
output_zero_point) * height)))
            xmax   =   int(min(width,   (output_scale   *   (xmax   -
output_zero_point) * width)))

            cv2.rectangle(frame_resized, (xmin,ymin), (xmax,ymax), (10,
255, 0), 4)

            final_box.append([xmin, xmax, ymin, ymax])

    final_box = np.round(final_box)

    if final_box.size >= 8:
        tree1 = final_box[0][:]
        tree2 = final_box[1][:]
        Ax = int(tree1[1])
        Ay = int(tree1[3])
        Bx = int(tree2[1])
        By = int(tree2[3])

        # y = mx + b
        m = (By - Ay) / (Bx - Ax)
        b = Ay - m * Ax

        cv2.line(frame_resized, (int(-b / m), 0), (int((height-b)/m),
int(height)), (255,255,255), thickness=2)

    if 'm' in locals():
        if final_box.size >= 8:
        ## ORIENTACAO PELO DECLIVE
            mmin = -1
            mmax = -.2

            if m < mmin:
                print('virar esquerda')
            elif m > mmax:
                print('virar direita')
            else:
                print('andar em frente')
        elif final_box.size == 4:
        ## ORIENTACAO PELO DESVIO DE X
            X = int(final_box[0][1])
            Y = int(final_box[0][3])
            xref = (Y - b)/m
            if X < xref-150:
                print('virar esquerda')
            elif X > xref+150:
                print('virar direita')
            else:
                print('andar em frente')
        else:
            print('andar em frente')
```

```
        cv2.line(frame_resized, (int(-b / m), 0), (int((height-b)/m),
int(height)), (255,255,255), thickness=2)
    else:
        print('andar em frente')

    frame_resized = cv2.cvtColor(frame_resized, cv2.COLOR_RGB2BGR)
# Draw framerate in corner of frame
    cv2.putText(frame_resized,'FPS:
{0:.2f}'.format(frame_rate_calc),(30,50),cv2.FONT_HERSHEY_SIMPLEX,1,(2
55,255,0),2,cv2.LINE_AA)
# All the results have been drawn on the frame, so it's time to display
it.
    cv2.imshow('Object detector', frame_resized)
# Calculate framerate
    t2 = cv2.getTickCount()
    time1 = (t2-t1)/freq
    frame_rate_calc= 1/time1

    # Press 'q' to quit
    if cv2.waitKey(1) == ord('q'):
        break

# Clean up
cv2.destroyAllWindows()
videostream.stop()
```

Figura 48. Algoritmo de inferência do modelo *TFLite* em *EdgeTPU* num *Raspberry Pi* com câmara.