**Universidade do Minho**
Escola de Engenharia

João Pedro Guimarães Araújo

# Ajuste De Posicionamento No Destino Final Para Sistemas De Entrega De Encomendas Realizadas Por Drones Autónomos

Dissertação de Mestrado
Mestrado Integrado em Engenharia
Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
**Professor Doutor Fernando Ribeiro**

junho de 2022

## DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

# Agradecimentos

Quero transmitir os meus mais sinceros agradecimentos a um conjunto de pessoas às quais foram incansáveis ao longo do meu percurso académico pelo seu apoio e ajuda.

Ao meu orientador Professor Fernando Ribeiro por todo apoio, empenho e disponibilidade demonstrada através desta dissertação.

Ao Tiago Ribeiro pelas trocas de opiniões e por me guiar em todas as etapas desta dissertação.

Aos meus pais que sempre me apoiaram e fizeram de tudo para que eu pudesse alcançar todos os meus objetivos, fazendo-me sempre acreditar nas minhas capacidades e para nunca desistir dos meus sonhos.

À minha namorada pelo apoio incondicional e motivação em todos os momentos, por me ter sempre ajudado perante todas as adversidades que fui encontrando ao longo deste percurso dando-me a força necessária para continuar este trabalho.

Aos meus amigos Diogo Costa, Daniel Fernandes, Fábio Pereira, Paulo Silva, João Rego e Hugo Ferreira pelos bons momentos partilhados.

## DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

# Abstract

Drones have been a platform that has aroused an increasing interest to several companies for the development of multiple autonomous applications including the package deliveries. This increasing interest in the use of autonomous drones to make deliveries it´s essentially due to a high potential for process optimization, improving business performance as well for being an efficient solution to overcome complex road conditions and congestion in urban areas. Thus, the improvement of this application to perform increasingly safer flights has been a challenging problem and of a great interest over the past few years.

The main goal of this project consisted in the development of a strategy which the drone when landing deviates backwards and forwards when detecting people, cars and cyclists in the landing area. In this approach, Supervised Learning was used to detect these objects. In addition, the drone's orientation and height control was carried out using PID control, as well as the trajectory planning, GPS implementation and obstacle detection and deviation using computer vision techniques in order to be able to perform fully autonomous drone flights.

**Keyword:** Supervised Learning, UAV, YOLOv3-Tiny, PID, Autonomous Drone Package Delivery, Robotics.

# Resumo

O *drone* tem sido uma plataforma que tem despertado um crescente interesse por parte de várias empresas para o desenvolvimento de diversas aplicações autónomas entre as quais a realização de entregas de encomendas. Este crescente interesse na utilização de *drones* autónomos para fazer entregas deve-se essencialmente ao seu elevado potencial para otimização de processos, melhoria de desempenho empresarial bem como por ser uma solução eficiente para ultrapassar as complexas condições rodoviárias e congestionamento das áreas urbanas. Assim, o aperfeiçoamento desta aplicação para a realização de voos cada vez mais seguros tem sido um problema desafiador e de grande interesse ao longo destes últimos anos.

O objetivo principal deste projeto consistiu no desenvolvimento de uma estratégia na qual o *drone* no momento da sua aterragem desvia-se para a frente ou para trás perante a deteção de pessoas, carros e ciclistas na área de aterragem. Nesta abordagem foi usado o *Supervised Learning* para a deteção destes objetos. Além disso, foi realizado o controlo de orientação e altura do *drone* com o uso do controlo PID, bem como o planeamento de trajetória, implementação GPS e deteção e desvio de obstáculos através do uso de técnicas de visão por computador de forma a ser possível realizar voos totalmente autónomos por *drones*.

**Palavras-chave:** *Supervised Learning*, UAV, *YOLOv3-Tiny*, PID, Entrega de encomendas por *drones* autónomos, Robótica.

# Índice

# Lista de Figuras

# Lista de Tabelas

# Lista de Siglas e Acrónimos

| | |
|---|---|
| AI | *Artificial Intelligence* |
| AP | *Average Precision* |
| CNN | *Convolutional Neural Network* |
| DNN | *Deep Neural Network* |
| FN | Falso Negativo |
| FP | Falso Positivo |
| FPS | *Frames Per Second* |
| GPS | *Global Positioning System* |
| GPU | *Graphical Processing Unit* |
| HSV | *Hue Saturation Value* |
| IoU | *Intersection Over Union* |
| mAP | *mean Average Union* |
| MPU | *Magnetic pickup* |
| ODE | *Open Dynamics Engine* |
| OpenCV | *Open Source Computer Vision Library* |
| PID | Proporcional-Integral-Derivativo |
| QZSS | *Quasi-Zenith Satellite System* |
| RGB | *Red Green Blue* |
| RPM | Rotação por minuto |
| TN | Verdadeiro Negativo |
| TP | Verdadeiro Positivo |
| TPU | *Tensor Processing Unit* |
| Wi-Fi | *Wireless Fidelity* |
| YOLO | *You Only Look Once* |

# Capítulo 1

# Introdução

Este capítulo tem como objetivo apresentar o enquadramento do tema que é abordado nesta dissertação de mestrado, bem como a motivação e os objetivos que devem ser alcançados. No final, é apresentada a estrutura da dissertação, que esclarece como será constituído cada capítulo e as abordagens que serão feitas em cada um destes.

## 1.1 Enquadramento e Motivação

Com o desenvolvimento tecnológico que tem vindo a surgir ao longo destes últimos anos na sociedade, existe uma crescente necessidade de automatizar determinadas tarefas de forma eficiente e precisa, por meio de plataformas robóticas.

O *Unmanned Aerial Vehicle* (UAV), ou *drone* - uma designação mais familiar para a generalidade da população - tem sido uma plataforma que tem despertado um grande interesse na criação de diversas aplicações benéficas. Isto deve-se ao facto da polivalência que estas aeronaves apresentam e na sua capacidade de incorporar ou acoplar diversas tecnologias capazes de satisfazer necessidades em vários sectores e de executar tarefas que seriam, por outros meios, morosas, desgastantes, caras, perigosas ou, até mesmo, impossíveis de realizar [1].

Ora, pelo seu elevado potencial e alto rendimento face a métodos alternativos em vários sectores, a conceção e a popularização de aplicações com *drones* constitui hoje em dia uma oportunidade para otimização de processos, melhoria de desempenho empresarial e transporte eficiente.

No que diz respeito a aplicações com *drones* autónomos no transporte de encomendas, são várias as empresas de renome no mercado atual. Veja-se o exemplo das transportadoras DHL [2], Amazon [2], [3], ou mesmo empresas a nível nacional, como a *Connect Robotics* [4], que têm apostado na utilização de *drones* para fazer entregas como uma solução para ultrapassar as complexas condições rodoviárias e o congestionamento nas áreas urbanas.

Com isto, surge assim a necessidade de adicionar funcionalidades a estes veículos autónomos de forma a realizarem trajetórias cada vez mais seguras. Dado que estes voos autónomos serão

realizados num ambiente maioritariamente citadino, espera-se que o agente autónomo consiga resolver os vários desafios que vão surgindo ao longo da sua trajetória, sejam eles para prevenção de colisões, ou de interação com os outros agentes presentes no ambiente no momento do voo e da aterragem.

## 1.2 Objetivos da Dissertação

Este projeto tem como principal objetivo o estudo e a implementação de uma rede neuronal que seja capaz de reconhecer certos alvos (automóvel, pessoa e ciclista) pela câmara inferior do *drone* ao efetuar uma aterragem. Com a deteção de algum destes objetos durante aterragem, pretende-se que o *drone* seja capaz de interagir com segurança, desviando-se desses obstáculos.

Para além disso, será também realizado o estudo da dinâmica do *drone*, a implementação do controlo de orientação, e o desvio de obstáculos pela câmara frontal do *drone*. Assim, os objetivos a alcançar são os seguintes:

- Estudo e implementação do controlo de orientação do *drone* em ambiente de simulação;
- Implementação de uma solução que permita o reconhecimento e o desvio de obstáculos pela câmara frontal do *drone*;
- Estudo de redes neuronais para o reconhecimento dos alvos definidos;
- Treino e teste da rede neuronal após a criação do *dataset*;
- Implementação, com a junção da rede neuronal, de uma estratégia capaz de desviar dos obstáculos para uma aterragem segura.

## 1.3 Organização e Estrutura da Tese

Este relatório está subdividido em 7 capítulos. Após este capítulo inicial, no capítulo 2, apresenta-se um estudo das tecnologias existentes dos *drones* comerciais, uma descrição do *quadcopter*, o seu princípio de funcionamento, bem como um estudo sobre *Machine Learning* e as arquiteturas existentes para a deteção de objetos. O terceiro capítulo, Plataforma e Ferramentas utilizadas, consiste em expor todos os *softwares*, ambientes e procedimentos utilizados. O capítulo seguinte, capítulo quatro, Modelos e Controlo do *Drone*, descreve as técnicas implementadas para o controlo da orientação e posição do *drone*. Para além disso, é também apresentada nesse capítulo a estratégia utilizada para o modo de navegação autónoma e a conversão das coordenadas GPS. O capítulo cinco, Deteção e Desvio de Obstáculos, indica para a deteção de obstáculos, o algoritmo de visão utilizado para a

deteção de cor e o cálculo para obter a distância do objeto em relação ao *drone*. Por último, são apresentados os cálculos das estratégias utilizadas para o desvio no sentido vertical e horizontal. O capítulo 6, Modo de Aterragem, descreve primeiramente os passos para treinar e testar a rede, bem como os procedimentos para criar o *dataset*. Posteriormente são relatados os resultados obtidos dos treinos e testes a várias redes com os diferentes *datasets* desenvolvidos, de forma a ser averiguado qual *dataset* criado é o mais vantajoso para o problema desta dissertação. Ainda neste capítulo é explicada a estratégia realizada para efetuar uma aterragem segura. Finalmente, no capítulo 7, são retiradas conclusões sobre o projeto e são dadas sugestões para trabalhos futuros.

# Capítulo 2

# Estado de Arte

O conteúdo deste capítulo encontra-se dividido em duas partes. A primeira parte aborda um estudo das tecnologias existentes dos *drones* comerciais, bem como os princípios de funcionamento de um *quadcopter* e os ângulos de Euler. Na segunda parte são relatados vários conceitos do *Machine Learning*, sendo feita uma análise sobre arquiteturas para a deteção de objetos.

## 2.1 *Drones* Comerciais

Perante o grande interesse nos serviços de entrega de encomendas por *drones* autónomos são apresentadas várias soluções inteligentes desenvolvidas por diversas empresas para a realização destas entregas de forma a serem cada vez mais seguras para a sua utilização.


Figura 2.1 - *Drone Prime Air* [5].

Em 2019, na conferência re:MARS da Amazon, foi apresentado o *drone Prime Air*, figura 2.1. Esta aeronave, controlada com seis graus de liberdade, contém diversos sensores e algoritmos avançados, como visão estéreo, para a deteção de objetos estáticos. Para a deteção de objetos em movimento, utiliza visão por computador e algoritmos de *Machine Learning*. Em relação à aterragem para a entrega da encomenda, só se realiza caso a área ao redor do local de entrega esteja livre de pessoas, animais ou obstáculos. Assim, para esta etapa o *drone* utiliza a visão por computador em paralelo com algoritmos de AI treinados para detetar pessoas e animais visualizados de cima. Para além disso, o uso de técnicas de visão por computador, permite a este *drone* reconhecer e evitar fios de telefone ou fios elétricos à medida que desce ou sobe da área de entrega [5].

Figura 2.2 - *Intelligent cabinet* da DHL [6].

Nesse mesmo ano, a DHL *Expess* em parceria com a Ehang inauguraram o primeiro serviço de entrega com *drones* autónomos, onde apresentaram uma solução capaz de enfrentar os vários desafios de entrega nas áreas urbanas da China. O *drone* utilizado é o EHang Falcon, que apresenta oito hélices distribuídos em quatro braços. No que diz respeito às funcionalidades desenvolvidas, este *drone* apresenta uma descolagem e aterragem na vertical, bem como um GPS de alta precisão e identificação visual, planeamento de trajetória e voo totalmente autónomo com ligação de rede em tempo real. Com uma carga de transporte máxima de 5 kg por voo, estes *drones* descolam e pousam em cima de *intelligent cabinets* desenvolvidas para carregar e descarregar as encomendas de forma totalmente autónoma, figura 2.2. Estes *cabinets* apresentam vários processos de automatização que incluem classificação, digitalização e armazenamento de correio bem como reconhecimento facial e ID *scanning* [6].

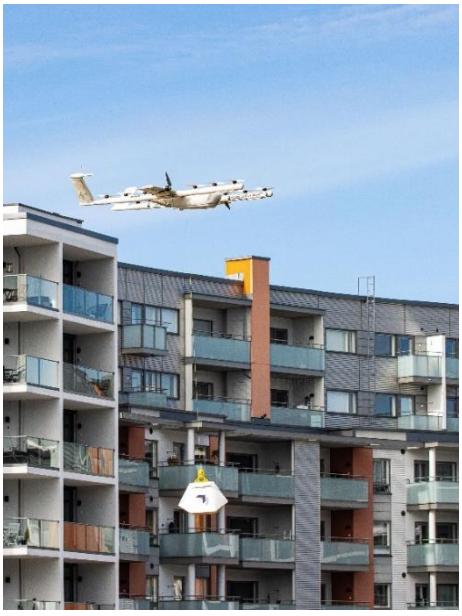


Figura 2.3 - *Drone da Wing* [7].

Além disto, existe a Alphabet´s Wing que consiste num serviço de entrega por *drones* projetado para entregar encomendas de porte pequeno que pesem aproximadamente 1,2 kg ou menos até ao local de entrega do cliente. Estas entregas são realizadas com os *drones* da *Wing*, figura 2.3, a voar a uma altura média de cerca de 45 metros acima do solo. Uma vez no destino do cliente, o *drone* desce a uma altura de 7 metros acima do solo e, em seguida, através de um cabo desce a encomenda e solta-a na área de entrega. Após ser realizada a entrega o *drone* volta a subir para a sua altura inicial e retorna para a base da *Wing*. Para a realização destes voos é utilizado o *software* deles designado por UTM (*Uncrewed Traffic Management*) que planeia uma rota para evitar obstáculos consoante os requisitos regulamentares. Ainda assim, os voos são supervisionados por pilotos treinados [7].

## 2.2 Descrição do *Quadcopter*

Como o nome indica os UAVs são *Unmanned Aerial Vehicles*, no qual o seu controlo de voo pode ser realizado remotamente por um piloto que se encontra no solo ou pode voar de forma autónoma. Esta possibilidade destes veículos serem controlados por um computador de bordo ou por um situado no solo, faz com que seja possível a transmissão de informações do seu estado, tais como, posição, velocidade, altitude ou qualquer outro parâmetro de telemetria.

Estes veículos, consoante o número de hélices, têm a sua própria designação que pode ser dividida em três principais tipos: *quadcopters*, *hexacopters* e *octacopters*. Para esta dissertação será utilizado um *quadcopter*, caracterizado pelos seus quatro conjuntos - motor e hélice -, posicionados à mesma distância. O *drone* apresenta assim um sistema de seis graus de liberdade, as coordenadas de XYZ (que descrevem a posição), e os ângulos de Euler $\theta, \phi$ e $\psi$ (que descrevem a sua orientação), com quatro variáveis controláveis (velocidade de cada um dos motores). Para o controlo de orientação e posição é efetuado através do aumento e diminuição da velocidade de cada um dos motores [8],[9].

### 2.2.1 Princípio de Funcionamento

Como referido anteriormente, o controlo do *quadcopter* é efetuado através da velocidade de rotação de cada motor. Esta velocidade de rotação gerada pela rotação das hélices produzem uma força de elevação e um binário, no qual, para que exista um equilíbrio do binário total do sistema, dois motores rodam no sentido dos ponteiros do relógio e os outros dois rodam no sentido anti-horário, figura 2.4 [8].

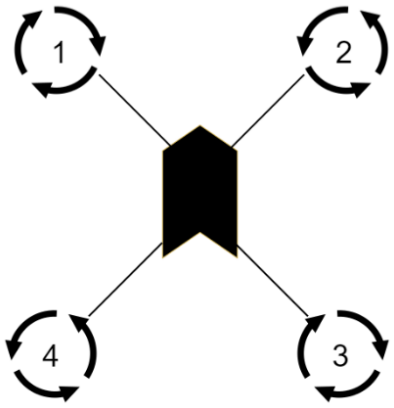
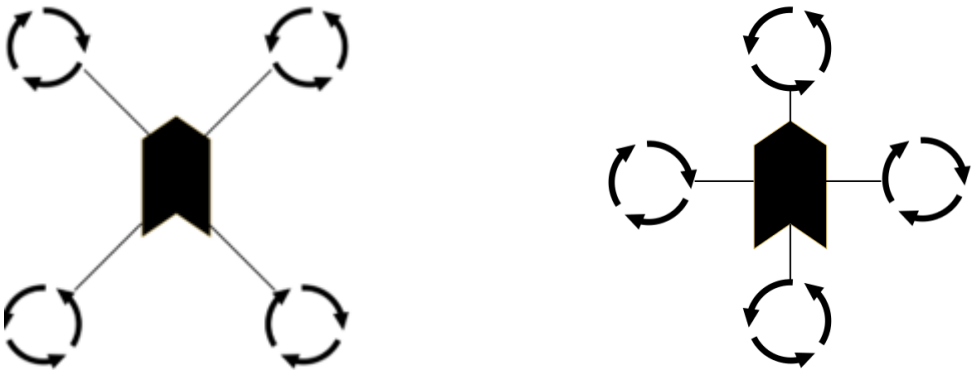



Figura 2.4 - Representação de disposição do sentido de rotação da velocidade dos motores. As setas representam a direção de rotação das hélices e as linhas e o centro representam a estrutura do *quadcopter*.

A descolagem de um *drone*, ocorre quando o somatório das forças geradas pela rotação das hélices é superior ao peso do *drone*, sendo ainda necessário, para que o mesmo consiga pairar no ar, que estas forças consigam igualar o peso do *drone*.

Na figura 2.5 são apresentadas as duas configurações mais comuns que uma estrutura de um *quadcopter* pode ter. Apesar da configuração em **x** ser a mais complexa no que diz respeito ao modelo matemático e controlo do *quadcopter* comparada com a configuração **+**, esta é ideal para a utilização de câmaras, pois nesta configuração as hélices do *quadcopter* não são captadas pelo plano da câmara, sendo por isso, a configuração escolhida para a realização desta dissertação [10].



(a) (b)

Figura 2.5 - Configurações em x (a) e em + (b).

A deslocação no espaço por um corpo rígido como o de um *quadcopter* é dada por seis graus de liberdade, sendo três translações (movimentação do centro de massa) e três rotações (movimentação em torno do centro de massa) ao longo de três eixos. Estes seis graus de liberdade correspondem aos movimentos na vertical, na lateral, para a frente e para trás, e às rotações em *roll*, *pitch* e *yaw* [9]. Deste modo, consoante os valores da velocidade de rotação de cada motor, é possível identificar quatro

movimentos básicos do *quadcopter*, que estão representados da figura 2.6 até à figura 2.9 (*quadcopter* na configuração em **x**).

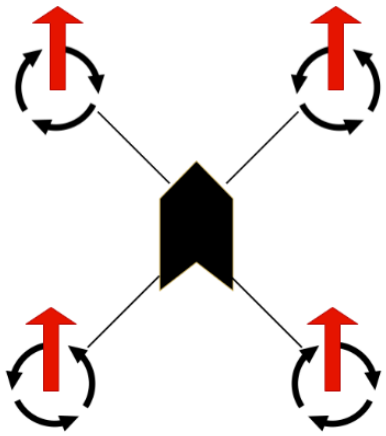


Figura 2.6 - Movimento de elevação.

Para que ocorra o movimento de elevação, figura 2.6, para além do somatório das forças geradas pela rotação das hélices terem que ser superiores ao peso do *quadcopter*, todos os motores têm que ter a mesma velocidade com a mesma aceleração. Assim, consoante a velocidade que é fornecida pelos motores, o *drone* irá subir, descer ou pairar no ar.

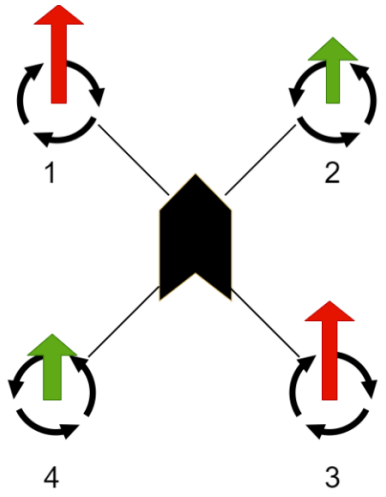



Figura 2.7 - Movimento *yaw*.

No movimento *yaw*, figura 2.7, os pares de motores (1 / 3) e (2 / 4) rodam a velocidades diferentes. Com esta diferença de velocidades, o binário total fica desequilibrado, fazendo com que o *drone* rode em torno de si próprio.

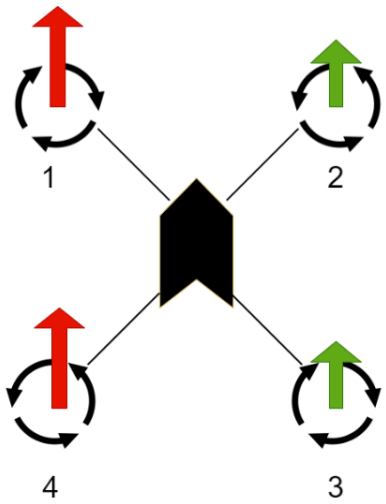



Figura 2.8 - Movimento *roll*.

Para que ocorra a deslocação do *drone* para a esquerda ou para a direita, ou seja, o movimento *roll*, figura 2.8, os pares de motores (1 / 4) e (2 / 3) têm que rodar a velocidades diferentes.

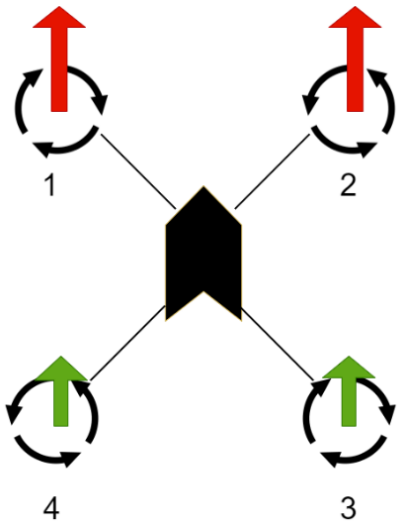



Figura 2.9 - Movimento *pitch*.

Por fim, no movimento *pitch*, figura 2.9, os pares de motores (1 / 2) e (4 / 3) rodam a velocidades diferentes. O resultado desta diferença faz com que o *drone* se desloque para a frente ou para trás.

### 2.2.2 Ângulos de Euler

Os ângulos de Euler permitem descrever a orientação de um corpo rígido num espaço tridimensional, isto é, permitem transformar as coordenadas de um ponto num referencial para as coordenadas do mesmo ponto noutro referencial. Assim, uma rotação em torno do eixo do x é denominada de *roll* ($\phi$), em torno de y é *pitch* ($\theta$) e em torno de z é *yaw* ($\psi$)[9].

Os ângulos de Euler representam uma sequência de três rotações elementares $(R_x, R_y\ e\ R_z)$[9]. Num espaço tridimensional, estas rotações começam a partir de uma orientação conhecida de um corpo rígido e são descritas através das seguintes matrizes de rotação:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \quad (1)$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad (2)$$

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

Assim, com a matriz de rotação $R_{zyx}$ (4) é possível passar as grandezas do referencial do *quadcopter* para o referencial fixo.

$$R_{zyx}(\phi, \theta, \psi) = R_z(\psi).R_y(\theta).R_x(\phi)$$
$$= \begin{bmatrix} \cos(\theta)\cos(\psi) & \sin(\phi)\sin(\theta)\cos(\psi) - \cos(\phi)\sin(\psi) & \cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\sin(\psi) \\ \cos(\theta)\sin(\psi) & \sin(\phi)\sin(\theta)\sin(\psi) + \cos(\phi)\cos(\psi) & \cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi) \\ -\sin(\theta) & \sin(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) \end{bmatrix} \quad (4)$$

## 2.3 *Machine Learning*

Arthur Samuel pioneiro da inteligência artificial define *Machine Learning* como sendo "o estudo que dá aos computadores a capacidade de aprender sem serem explicitamente programados" [11]. É a partir da experiência obtida da análise prévia de dados que este subcampo da *Artificial Intelligence* (AI) desenvolve algoritmos para pesquisa de padrões, que permitem ao sistema construir um modelo capaz de fazer previsões sobre dados futuros.

Um dos tópicos mais importantes da rede neuronal é o treino, no qual a rede aprende com os dados quais são as caraterísticas que correspondem a uma determinada classe, permitindo a previsão correta da classe dos dados de entrada. Com os padrões dos dados fornecidos, no treino são ajustados os pesos da rede neuronal com o objetivo de ser desenvolvido um modelo [12].

Contudo, para avaliar o desempenho do modelo e neste caso, a rede neuronal, os dados devem ser inicialmente divididos em dois conjuntos: o conjunto de treino, no qual estão englobados os dados

de treino para desenvolver o modelo, e o conjunto de teste. Deste modo, o conjunto de teste deve conter exemplos de dados diferentes do conjunto de treino dado que terá de ser avaliado se o algoritmo desenvolvido aprendeu a generalizar a partir do conjunto de treino ou se o memorizou. Assim, entende-se que um programa que generaliza bem será capaz de realizar uma previsão de forma mais eficaz com novos dados. A este problema de memorização do conjunto de treino por parte da rede chama-se *over-fitting* [11].

Os algoritmos de *Machine Learning* podem ser classificados conforme a quantidade e tipo de supervisão que recebem durante o treino. Para este efeito, o método mais utilizado é o sistema de *supervised learning*, no qual os dados de treino que são fornecidos ao algoritmo incluem as soluções desejadas, que são designadas por *label*. O objetivo do algoritmo de *supervised learning* é aprender uma função que, dada uma amostra de dados e saídas desejadas, melhor se aproxima da relação entre a entrada e a saída observável nos dados [13], [14].

Dentro do método de *supervised learning* existem dois algoritmos principais que definem a natureza da saída, a classificação e a regressão.

A classificação é utilizada quando o problema pode ser definido por um número finito de classes, ou seja, o programa deve prever a categoria, classe ou *label* mais provável para novas observações, como por exemplo, para classificar dígitos de 0 a 9 e, portanto, a saída é composta por essas 10 classes [11], [15].

Contrariamente, a regressão, cujo objetivo é utilizada quando a saída é um valor contínuo, mais precisamente, o programa deve prever o valor de uma variável de resposta contínua, como por exemplo, prever as vendas de um novo produto ou o salário de um trabalho com base da sua descrição [11], [15].

Por outro lado, em oposição à *supervised learning*, existe outro método denominado por *unsupervised learning* no qual aos dados de treino disponíveis não são fornecidos as *labels*, ou seja, que não são fornecidas as soluções desejadas. Desta forma, este método consiste em descobrir conjuntos de dados que podem ser relacionados, chamados *clusters*, dentro dos dados de treino [11].

Por fim, o método *semi-supervised learning* consiste em treinar o algoritmo com base em dados com e sem *label*, ou seja, faz uso dos dados dos métodos de aprendizagem descritos anteriormente. Desta forma, o objetivo deste método é classificar dados *unlabeled* com base em conjuntos de dados com *labels*. Neste segmento, um exemplo de *semi-supervised learning* é o algoritmo de *reinforcement learning*, no qual um programa maximiza o desempenho do algoritmo com base nas interações com o

ambiente. Assim, o sistema de *reinforcement learning* consiste em determinar o comportamento ideal de um sistema, tentando obter a maior recompensa imediata e cumulativa através dos *feedbacks* dados pelas interações com o ambiente, que podem ser positivos, chamados de *reward*, ou negativos, denominados por *penalty* [11], [13].

### 2.3.1 *Deep Learning* (*DNN*)

*Artificial Neural Networks* (*ANN*) são baseadas no sistema nervoso biológico, no intuito de resolver problemas de classificação e regressão [16]. Pela analogia com o cérebro humano, o elemento básico de processamento da rede neuronal é o neurónio, também chamado de nó (figura 2.10). Um agrupamento de neurónios forma aquilo a que se chama de camada e um conjunto de camadas forma a rede neuronal. A saída de cada neurónio consiste em após receber na entrada do nó um ou vários sinais que são multiplicados por um peso, m, que corresponde às conexões entre os diferentes neurónios das camadas adjacentes. De seguida, com o sinal total de entrada proveniente da soma do produto de cada operação é somado um valor *bias*. Por fim, esta soma dentro do neurónio é processada por uma função de ativação que dará origem à variável de saída do neurónio, que pode ser passada como entrada para outros neurónios, formando redes neuronais complexas [17].

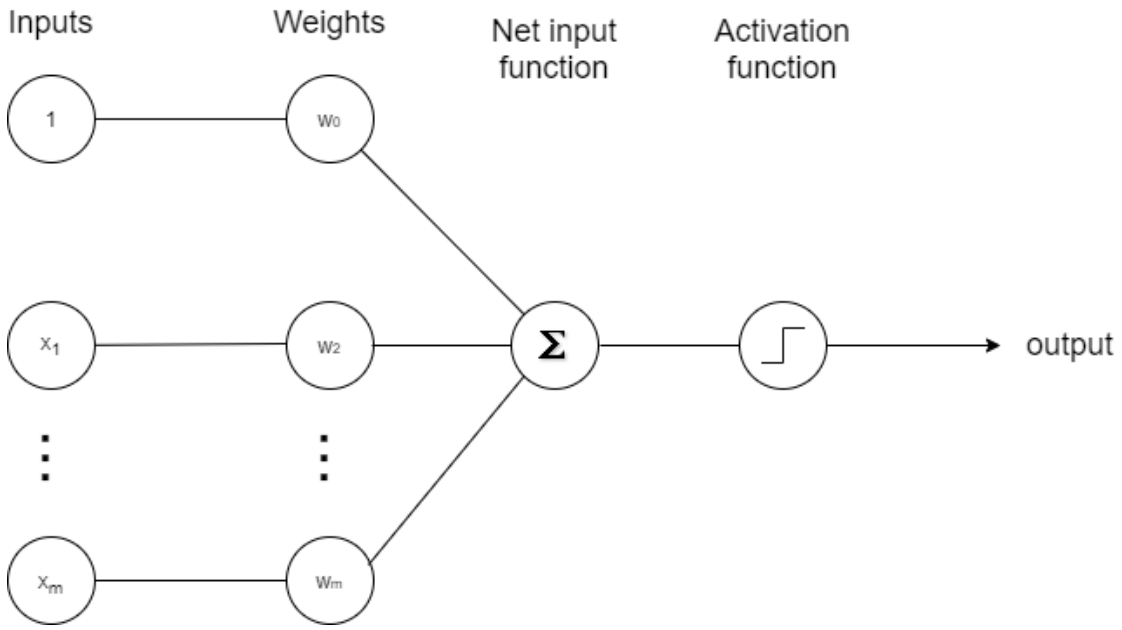



Figura 2.10 - *Artificial Neural Network Model*.

A função de ativação, que já foi descrita anteriormente, é uma função matemática para delimitar a faixa de saída de um neurónio, no qual pode ser linear ou não linear dependendo da função que representa, sendo que a escolha recai consoante aplicação da rede neuronal [18]. Algumas funções de ativação, bem como a sua expressão matemática, podem ser analisadas na tabela que se segue.

Tabela 2.1 - Tabela das funções de ativação.

| Função de ativação | Expressão matemática |
|---|---|
| *Binary Step function* | $f(x) = \begin{cases} 0 \; for \; x < 0 \\ 1 \; for \; x \geq 0 \end{cases}$ |
| *Hyperbolic Tan* (tanh) | $f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$ |
| *Rectified Linear Units* (ReLU) | $f(x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$ |
| *Sigmoid* | $S(x) = 1 + \frac{1}{1 + e^{-x}} + \frac{e^{x}}{e^{x} + 1}$ |
| *SoftMax* | $S(x) = 1 + \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$ |

A fim de melhorar o desempenho da rede neuronal, é necessário o uso de algoritmos de treino, através da implementação de uma função matemática, denominada *loss function*, cujo objetivo é configurar os parâmetros da rede durante o treino, como os pesos das conexões dos neurónios e os valores de polarização (*bias*), de forma a distribuir o conhecimento adquirido pelas diferentes conexões, aumentando por sua vez o comportamento da rede junto com as diferentes iterações. Assim, a escolha da *loss function* recai mais uma vez no domínio da aplicação.

Uma das *loss functions* conhecidas é o *gradient descend*, que através da inclinação da curva da função serve como guia apontando para o valor mínimo. Esta função tem como objetivo localizar o valor mínimo na curva inteira, representando as entradas em que a rede neuronal é mais precisa [16].

Dentro das redes neuronais, a *DNN* (*deep learning neural network*) é uma evolução das *Artificial Neural Networks – ANN* - que surgiu pela primeira vez como um algoritmo de aprendizagem em 1948 através de Donald Hebb (figura 2.11). A principal diferença entre a *ANN* e a *DNN* é o número de camadas escondidas em cada uma [12].

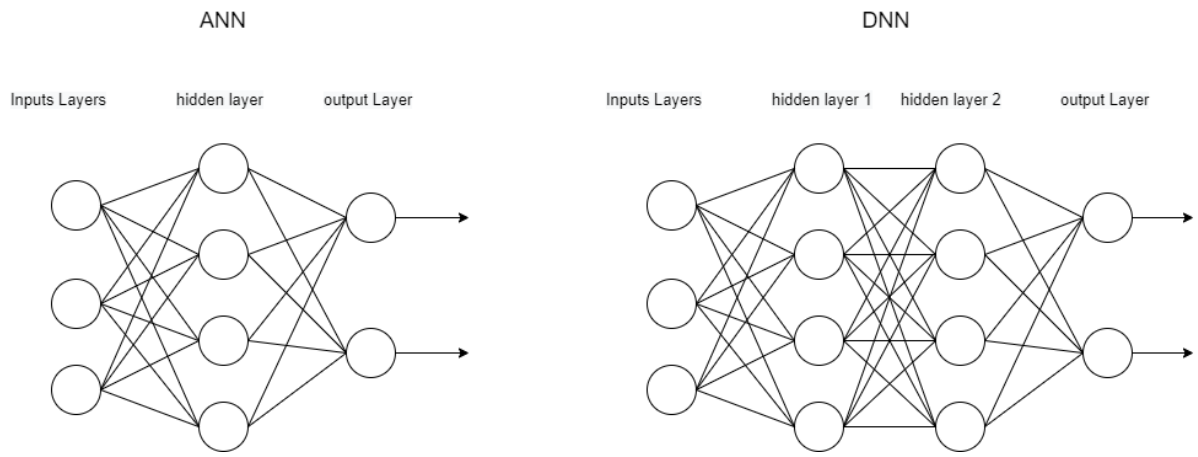



Figura 2.11 - Representações de uma *Simple Neuronal* e uma *Deep Learning Neural Network*.

A rede multicamada tradicional é uma rede neuronal constituída por vários neurónios ligados entre si, como o *DNN*, que são capazes de reconhecer objetos em imagens.

Uma imagem consiste numa matriz de números de várias dimensões e cada número representa um valor de pixel, no qual, se a intenção for treinar uma *DNN* para o reconhecimento de objetos, cada nó de entrada deve corresponder a um pixel da imagem. Deste modo, ao treinar uma rede com um tamanho de imagem 30x30, por exemplo, será necessário existir 900 nós de entrada na rede. Contudo, este número de nós de entrada pode aumentar drasticamente consoante a resolução da imagem ou se a imagem for colorida, como é o caso de uma imagem RGB (*Red Green Blue*) em que o número de nós de entrada aumenta 3x comparado a uma imagem sem cor da mesma resolução. Em adição a este elevado número de parâmetros necessários para implementar e treinar a rede, há, inevitavelmente, um aumento de recursos computacionais [12], [17].

### 2.3.2 *Convolution Neural Network* (*CNN*)

As *convolution neural network* (*CNN*), um tipo de modelo do *deep learning*, surgiu com a necessidade de desenvolver uma abordagem que permita a redução dos custos de processamento de grandes quantidades de dados. Assim, a *CNN* torna-se uma alternativa comum às implementações clássicas de *ANN* [18].

Tipicamente, a arquitetura da *CNN* é constituída por várias camadas, como *convolutional layer*, *pooling layer* e *fully-connected layer*, desempenhando em cada uma delas uma função predefinida nos seus dados de entrada. As primeiras duas camadas descritas são responsáveis pela extração de caraterísticas e a terceira camada além de ser responsável pela extração é também responsável pela classificação das mesmas.

A *convolutional layer* contém uma série de filtros também designados como *Kernel'* s. A convolução consiste em aplicar um *Kernel* que desliza por toda a camada de entrada da imagem, no qual, começa no canto superior esquerdo e termina no canto inferior direito [12]. Cada filtro é uma matriz de números inteiros, composto por um conjunto de pesos, que são multiplicados pelos valores dos pixéis da imagem de entrada. No final, será somado esse resultado, de forma a obter um único valor que representa uma célula, ou um pixel, da matriz da imagem de saída [19] (figura 2.12).

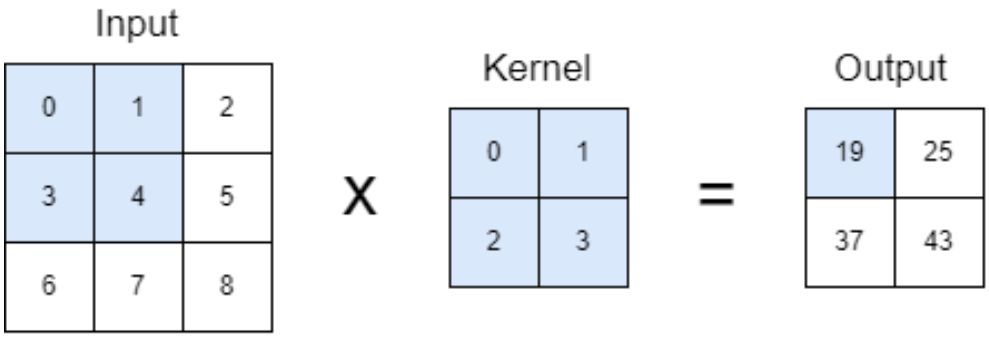



Figura 2.12 - *Convolutional layer*.

A *pooling layer* é responsável por reduzir as dimensões dos mapas de características, diminuindo desta forma os custos computacionais e a complexidade do modelo, permitindo assim controlar melhor o fenómeno de sobre ajuste, também designado como *overfitting* [20]. Esta camada usa também filtros, à semelhança da *convolutional layer*, mas ao invés de realizar a soma ponderada, executa funções aritméticas em todos os mapas de ativação anteriores [12]. Os métodos mais comuns da *pooling layer* são o *max pooling* (figura 2.13), que procura o valor máximo de todos os pixéis dentro do filtro, e o *average pooling* (figura 2.14), no qual realiza o cálculo do valor médio [18].

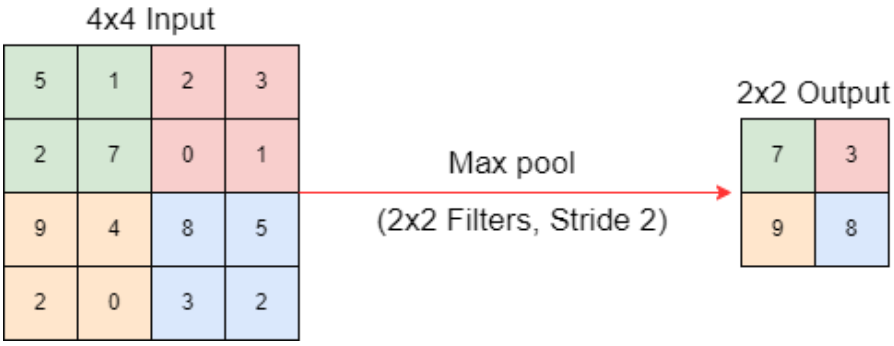



Figura 2.13 - *Max pooling*.

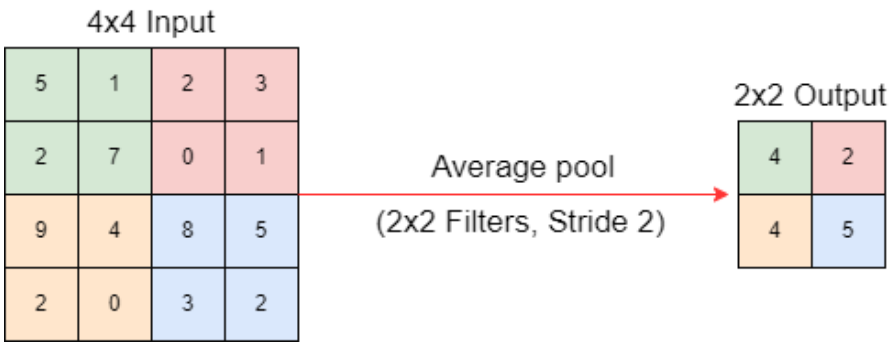



Figura 2.14 - *Average pooling*.

Por fim, após a imagem passar por um conjunto de *convolutional layer* e *pooling layer*, a *fully-connected layer* escolhe os recursos de alto nível e determina quais recursos mais se correlacionam com uma classe específica [12]. Esta camada é semelhante à forma como os nós estão organizados numa rede neuronal tradicional, onde cada nó em *fully-connected layer* está diretamente conectado a cada nó na camada anterior e na próxima [18].

### 2.3.3 *Intersection Over Union*

Para medir a precisão de um modelo na previsão da posição de um objeto é normalmente utilizada a IoU (*Intersection over Union*) como método de avaliação. Este método consiste numa *bounding box*, ou seja, numa caixa prevista na rede, em relação a uma caixa de verdade fundamental, fornecida pelo conjunto de dados. Deste modo, a IoU é uma relação que representa o quanto uma caixa prevista está contida pela verdade fundamental [21]. Este método pode ser determinado pela equação 5.

$$\mathrm{IoU} = \frac{\acute{a}rea\ de\ interse\c{c}\tilde{a}o}{\acute{a}rea\ de\ uni\tilde{a}o} \tag{5}$$

Como se pode observar pela equação anterior, para o cálculo do método o numerador corresponde à área de interseção e o denominador à área de união de ambas as caixas, ambas representadas na figura 2.15 [22].

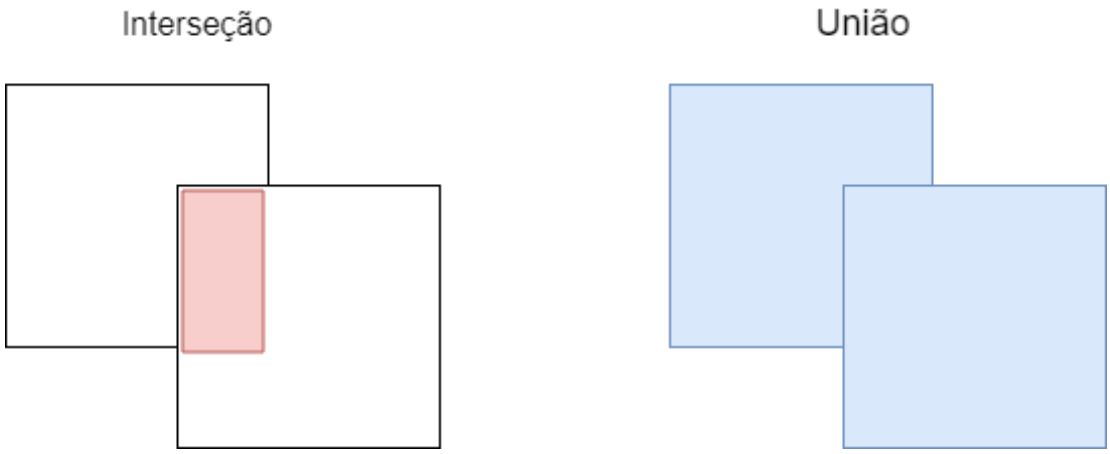



Figura 2.15 - *Intersection Over Union*.

### 2.3.4 *Average Precision* (*AP*)

Para avaliar o desempenho de um modelo é normalmente utilizado o valor de *average precision* (*AP*) como método de avaliação. Este método consiste em calcular os valores de *precision* e de *recall* para obter a média das precisões máximas em 11 níveis de *recall* [23]. A melhor maneira de avaliar o modelo é construir e analisar uma *Confusion Matrix*. Deste modo, é ilustrado na tabela 2.2 uma *Confusion Matrix*, que representa um problema de classificação binária com duas classificações possíveis: sim ou não.

Tabela 2.2 – *Confusion Matrix*.

| | **Previsão do modelo: Sim** | **Previsão do modelo: Não** |
|---|---|---|
| **Classificação atual: Sim** | TP | FN |
| **Classificação atual: Não** | FP | TN |

Como se pode observar, a *Confusion Matrix* apresenta a contagem de quatro termos básicos que entram no cálculo de qualquer métrica de desempenho do modelo. Assim, numa amostra positiva (sim) ao ser classificada corretamente, está-se perante um verdadeiro positivo (TP), caso contrário, se uma amostra positiva (sim) for classificada como negativa (não), é um falso positivo (FP). Para além disso, se uma amostra negativa (não) for classificada corretamente é um verdadeiro negativo (TN), ou se uma amostra negativa (não) for classificada como positiva (sim), é um falso negativo (FN).

Desta forma, a *precision* (equação 6) é a percentagem das previsões que estão corretas, ou seja, a proporção de casos positivos previstos que são de facto reais positivos [23].

$$Precision = \frac{TP}{\mathrm{TP} + \mathrm{FP}} \tag{6}$$

Por outro lado, é também normalmente utilizado o *recall*, que consiste na proporção de casos reais positivos que são corretamente previstos [23] - equação 7.

$$Recall = \frac{TP}{\mathrm{TP} + \mathrm{FN}} \tag{7}$$

O método, inicialmente, consiste em classificar as probabilidades das previsões fornecidas pela rede, de ordem decrescente de acordo com o nível de confiança previsto. Com o método IoU verifica-se se as previsões são positivas ou negativas ao comparar as previsões e a verdade fundamental [24]. De seguida, serão calculados os valores de *precision* e *recall*.

Para cada valor de *recall*, r, de 0 a 1 em incrementos de 0,1, a *precision* é interpolada substituindo-a pela maior precisão nos próximos *recall* $r' \geq$ r (equação 8) [24].

$$p_{inter}(r) = max_{r' \geq r}(r') \tag{8}$$

Por fim, a *average precision* é então obtida pela média das precisões máximas dos 11 valores de *recall* (0, 0.1, 0.2, ..., 0.9, 1), dados pela equação 9.

$$AP = \frac{1}{11} \sum_{r \in \{0.0, \ldots, 1\}} p_{inter}(r) \tag{9}$$

O mAP (*mean average precision*) consiste na média de todas as *average precisions* obtidas em cada classe.

### 2.3.5 *Convolutional Neural Network Architectures*

Uma das primeiras arquiteturas das *convolutional neural networks* que impulsionou o *deep learning* foi o *LeNet-5* desenvolvido por Yann Lecun et al, em 1998 [17]. Esta arquitetura foi fundamental para o campo do *deep learning* no que respeita ao reconhecimento de imagens. Esta arquitetura é constituída por três *convolutional layers,* duas *subsampling,* cujo resultado consiste em *pooling layer*, e duas *fully-connected layers* [25]. Estas camadas são a raiz de todas as arquiteturas recentes *convolutional neural network* [12].

Devido à incapacidade de computação para processar dados e à escassez de dados de imagem, só a partir de 2012 foram possíveis novos desenvolvimentos de arquiteturas. Nesse mesmo ano, foi ainda introduzida uma nova arquitetura nas *convolutional neural networks*, apresentada por A.

Krizhevsky et al, denominada por *AlexNet* [12]. Esta é uma rede semelhante à *LeNet-5*, mas com mais camadas - cinco *convolutional layers* e três *fully-connected layers* [26].

A introdução desta rede revolucionou a área do *deep learning*. Ao obter bons resultados no concurso de classificação de imagens, *ILSVRC* (*ImageNet Large Scale Video Recognition Competition*), em 2012, provou que a *CNN* pode ser utilizada para aprender recursos complexos em imagens, ao classificar mais de 1,2 milhões de imagens de alta resolução com um baixo erro comparado com os concorrentes dos últimos anos [27], [28]. Deste modo, esta rede impulsionou o seu desenvolvimento constante. No ano seguinte, em 2013, uma nova rede derivada do *AlexNet*, a rede *Overfeat* [29], proposta por P.Sermenete et al, demostrou que uma *CNN* não serve apenas para classificar, como também para detetar e localizar objetos em imagens, devido à previsão das *bounding boxes*, dando origem a novas arquiteturas relacionadas à deteção e localização de objetos em imagens, como as arquiteturas *R-CNN* (*Region Convolutional Neural Network*), *SSD* (*Single Shot Multibox Detector*) e *YOLO* (*You Only Look Once*).

Em 2014, a rede *GoogleNet* [30] e a rede *VGG* [31] foram as duas novas arquiteturas que obtiveram excelentes resultados no concurso *ILSVRC-2014*, com 6,6 % e 7,5 % [32] respetivamente, de classificação top-5 erro taxa. Ainda assim, em 2015 foi o ano em que foi desenvolvida a melhor arquitetura no *ILSVRC*, denominada por *ResNet* (*deep residual network*) [33], que venceu o concurso com 3,6 % [34] de taxa de erro.

A rede *ResNet* introduziu a ideia de *residual learning* para as *deep neural networks*. Esta arquitetura consiste em um *deep net* com 152 *layers* e no qual cada *layer* é chamada de *residual block* [12]. (figura 2.16).

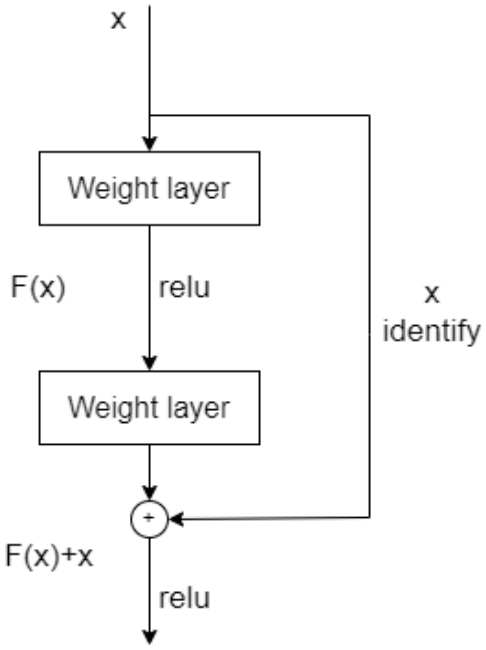



Figura 2.16 – *Residual block*.

Comparativamente às arquiteturas anteriores, o *residual block* ao invés de calcular toda a transformação da entrada, ou seja, x para F(x), permite calcular apenas a variação da entrada. Esta arquitetura torna a *backpropagation* um processo mais fácil [12].

### 2.3.6 Deteção de Objetos

A deteção de objetos tornou-se a base para resolver tarefas excessivamente complexas relacionadas com visão [35]. Conforme o que foi dito anteriormente, a introdução da previsão das *bounding boxes* na rede *Overfeat*, deu origem a novas arquiteturas relacionadas à deteção e localização de objetos em imagens, como as arquiteturas *R-CNN* (*Region Convolutional Neural Network*), *SSD* (*Single Shot Multibox Detector*) e *YOLO* (*You Only Look Once*).

Os métodos *SSD* e *YOLO* realizam as deteções como um problema de regressão, designado por *regression-based object detector* [12].

Uma das primeiras redes a aproveitar as qualidades das *CNN's* para a deteção de objetos nas imagens com precisão foi a *R-CNN*. Este método consiste em executar uma técnica chamada *Selective Search* que gera potenciais regiões (*region proposals*). Cada uma dessas propostas de região, que é dimensionada para um tamanho fixo, é submetida a uma *CNN* pré-treinada para ser classificada de forma a prever se pertence ou não ao objeto de interesse. Em caso afirmativo, depois da classificação serão otimizadas as potenciais regiões através do uso de um modelo de regressão linear. Esta otimização consiste em encontrar a região que mais se enquadra às dimensões do objeto.

### 2.3.7 *You Only Look Once* (*YOLO*)

O modelo *YOLO*, como foi descrito anteriormente, é uma arquitetura relacionada à deteção e localização de objetos em imagens. Este método consiste em detetar um objeto ao passar uma única vez em toda a rede, no qual prevê ao mesmo tempo as *bounding boxes* e as respetivas probabilidades de classe, ou seja, realiza todo o processo de deteção com apenas uma passagem [36]. Esta arquitetura introduzida no início de 2016, por J. Redmon et al, atingiu uma taxa de quadros de 21 fps, muito mais rápido do que a atual *R-CNN*, a rede *Faster R-CNN*.

A rede *YOLO* inspeciona toda a imagem de entrada ao dividir cada imagem em uma grelha de quadrados SxS, o que promove uma melhor distinção entre objetos e o fundo, potencializando, assim a precisão da rede. Cada quadrado da grelha é responsável por prever um conjunto de *bounding boxes* e de atribuir uma pontuação de confiança para cada caixa [36].

Esta confiança consiste numa pontuação de probabilidade em que reflete o quão precisas são as *bounding boxes* ($IOU_{Pred}^{Truth}$) e qual a probabilidade de um determinado objeto estar dentro dessas *bounding boxes* ($\Pr(object)$) [36]. Isto define-se pela equação 10.

$$\Pr(object) * IOU_{Pred}^{Truth} \tag{10}$$

Desta forma, se não existir nenhum objeto dentro da caixa, as pontuações de confiança devem ser zero. Caso contrário, a pontuação de confiança deve ser igual à interseção sobre a união (IOU) entre a caixa prevista e a verdade fundamental [36].

Por fim, após prever a probabilidade de uma classe estar contida dentro de cada quadrado da grelha ($\Pr\left(Class_i|Object\right)$), será obtida a probabilidade de uma determinada *bounding box* conter um tipo específico de um objeto $Pr(Class_i)$, ao combinar numa pontuação final a pontuação de confiança para cada *bounding box* e a previsão da classe, como é demonstrado na equação 11. No final são obtidas as representações das *bounding boxes* que apresentam um valor de confiança acima do valor limite [36].

$$\Pr(Class_i|Object) * \Pr(object) * IOU_{Pred}^{Truth} = Pr(Class_i) * IOU_{Pred}^{Truth} \tag{11}$$

Este modelo é composto por 24 *convolutional layer* seguidas por 2 *fully-connected layer*. Embora tenha sido inspirada na arquitetura *GoogleNet*, o modelo *YOLO* utiliza *convolutional layer* 1x1 seguida por uma *convolutional layer* 3x3. É de referir que na última camada é uma matriz 7x7x30, no qual o sistema divide a imagem em uma grelha 7x7. O x30 representa todos os valores que cada quadrado da

grelha calcula, ou seja, *bounding boxes* e todas as probabilidades de classe para cada quadrado [36]. Além disto, a última camada pode ser definida pela equação 12.

$$S * S * (B * 5 + C) \quad (12)$$

S representa o tamanho da grelha, B o número de *bounding boxes* para cada quadrado e C é o número de classes.

Uma das principais limitações do modelo *YOLO* consiste na deteção de objetos de pequena dimensão, dado que cada quadrado da grelha pode ter apenas uma classe e ter um número limitado de *bounding boxes* para prever [35], definido por B. Verifica-se também dificuldades em generalizar objetos com configurações ou formatos incomuns.

Tendo em atenção as desvantagens deste modelo, foram desenvolvidos novos modelos com uma melhor deteção ressalvando a velocidade de deteção, a saber, a rede *YOLOv2* [37] e a rede *YOLOv3* [38].

#### 2.3.7.1 *YOLOv2*

O modelo *YOLOv2* foi a primeira variante da rede *YOLO* original a ser desenvolvida. Esta nova variante levou a um aumento da performance do modelo *YOLO* original através da introdução de *Batch Normalization*, a um aumento da resolução das imagens para o classificador, bem como a um aumento da previsão de múltiplos objetos por cada célula pela qual a imagem se divide. Para além disso, é utilizado na rede *YOLOv2* um novo modelo de classificação, a arquitetura *Darknet-19*. Este novo modelo possui 19 *convolutional layers* e 5 *maxpooling layers* [37].

#### 2.3.7.2 *YOLOv3*

A rede *YOLOv3* foi publicado em 2018, sendo uma nova melhoria em relação às variantes anteriores do modelo *YOLO* original. Comparado com as versões anteriores, a rede *YOLOv3* apresenta uma melhor previsão das *bounding boxes* bem como uma melhor deteção de pequenos objetos. Quando uma nova imagem passa pela entrada da nova variante *multi-scale*, ela passa inicialmente pela *feature extractor* e com as características obtidas, o *Detector* obtém as *bounding boxes* e prevê a classe correspondente.

Deste modo, a rede *YOLOv3* usa um modelo de classificador melhorado, a arquitetura Darknet-53 (figura 2.17). Este classificador com 53 *convolutional layers*, utiliza ao longo da sua rede sucessivas *convolutional layers* 3x3 e 1x1 bem como várias *skip connections* [38].

| | Type | Filters | Size | Output |
|---|---|---|---|---|
| | Convolutional | 32 | 3 x 3 | 256 x 256 |
| | Convolutional | 64 | 3 x 3 / 2 | 128 x 128 |
| 1x | Convolutional | 32 | 1 x 1 | |
| | Convolutional | 64 | 3 x 3 | |
| | Residual | | | 128 x 128 |
| | Convolutional | 128 | 3 x 3 / 2 | 64 x 64 |
| 2x | Convolutional | 64 | 1 x 1 | |
| | Convolutional | 128 | 3 x 3 | |
| | Residual | | | 64 x 64 |
| | Convolutional | 256 | 3 x 3 / 2 | 32 x 32 |
| 8x | Convolutional | 128 | 1 x 1 | |
| | Convolutional | 256 | 3 x 3 | |
| | Residual | | | 32 x 32 |
| | Convolutional | 512 | 3 x 3 / 2 | 16 x 16 |
| 8x | Convolutional | 256 | 1 x 1 | |
| | Convolutional | 512 | 3 x 3 | |
| | Residual | | | 16 x 16 |
| | Convolutional | 1024 | 3 x 3 / 2 | 8 x 8 |
| 4x | Convolutional | 512 | 1 x 1 | |
| | Convolutional | 1024 | 3 x 3 | |
| | Residual | | | 8 x 8 |
| | Avgpool | | Global | |
| | Connected | | 1000 | |
| | Softmax | | | |

Figura 2.17 - *Darknet-53*.

Para prever as *bounding boxes*, o sistema usa as *anchor boxes* [38], no qual a largura e a altura de uma *bounding box* é prevista através de deslocamentos em relação às *anchor boxes* e as coordenadas do centro são previstas através de uma função *sigmóide*. Assim, quatro coordenadas são previstas pela rede para cada *bounding boxes*: $t_x, t_y, t_w, t_h$. Se for detetado um deslocamento no canto superior esquerdo da imagem $(c_x, c_y)$ e a *anchor box* para essa deteção tiver uma largura e altura $p_w, p_h$, a previsão do centro $(b_x, b_y)$, bem como a largura e altura $b_w, b_h$ da *bouding box* será obtida através das seguintes equações.

$$b_x = \sigma(t_x) + c_x \tag{13}$$

$$b_y = \sigma(t_y) + c_y \tag{14}$$

$$b_w = p_w e^{t_w} \tag{15}$$

$$b_h = p_h e^{t_h} \tag{16}$$

### 2.3.7.3 *YOLOv3-Tiny*

Para o aumento da performance em termos de velocidade para sistemas com pouca capacidade de processamento, foi desenvolvida uma nova versão melhorada do modelo *YOLOv3*, a rede *YOLOv3-tiny*.

A rede *YOLOv3-tiny* é uma versão reduzida da rede *YOLOv3* que permite a uma rede ser aproximadamente 442 % mais rápido que as restantes versões *YOLO* [39]. Esta nova arquitetura para além de ser composta por 7 *convolutional layers*, usa 6 *maxpool layers* para a *feature extraction* e 2 escalas para a *detection layer* [40] (figura 2.18).

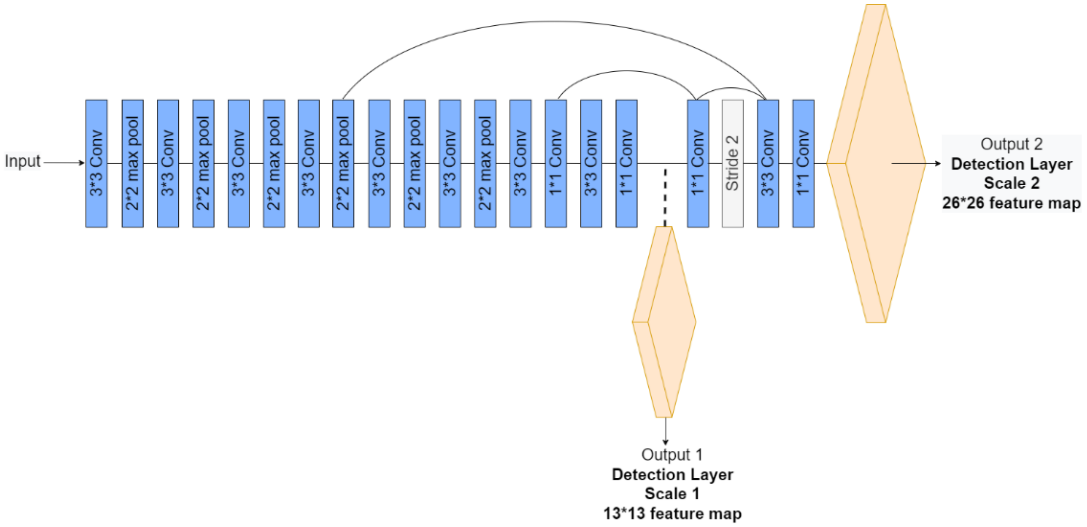



Figura 2.18 - Arquitetura da rede *YOLOv3-tiny*.

Apesar deste aumento de velocidade, esta rede apresenta uma pequena perda de exatidão. Contudo, comparativamente ao ganho obtido em velocidade que é fundamental para a deteção em tempo real, faz com que esta rede seja mais eficaz que o modelo *YOLOv3* na deteção e localização de objetos, bem como na respetiva identificação da classe em tempo real [39].

# Capítulo 3

# Plataforma e Ferramentas utilizadas

O *software* foi implementado no sistema operativo Ubuntu 20.04 num Asus ROG Strix GL553VD com um processador Intel Core i7-7700HQ CPU e uma placa gráfica Nvidia GeForce GTX 1050, sendo utilizado o *Python* como a principal linguagem de programação. O *Integrated Development Environment* (IDE) escolhido foi o *Pycharm Community* 2022.1.

## 3.1 *Parrot Bebop* 2

Neste trabalho, a implementação das características do *quadcopter* no ambiente de simulação foi baseado no *Parrot Bebop* 2. Este *drone* fabricado pela empresa *Parrot* é um *quadcopter* com quatro motores, que apresenta uma velocidade de rotação que vai desde os 7500 a 12000 rpm, tendo um peso de 500 gramas. As suas hélices, fabricadas em plástico, são capazes de alcançar uma velocidade horizontal de 16 m/s e na vertical de 6 m/s [41].

Em relação aos sensores, o *Bebop* 2 é equipado com um acelerómetro de 3 eixos MPU 6050 para medir aceleração do *drone* e um giroscópio de 3 eixos no mesmo chip MPU 6050 para a deteção da velocidade e rotação. Possui também um magnetómetro de 3 eixos AKM 6983 para medir a orientação absoluta do *drone* em relação ao campo magnético da terra. Para além disto, apresenta um sensor ultrassónico localizado por baixo do *drone* que permite determinar com precisão uma altitude até aos 8 metros acima do solo e um barómetro MS5607 que fornece informação mais precisa para altitudes acima dos 8 metros. A localização GPS é fornecida por um *GNSS* ou *Global Navigation Satellite System* NEO-M8 que faz uso dos sinais dos sistemas de navegação por satélite GPS, QZSS, GLONASS, BeiDou e Galileo [41][42].

Para além dos sensores descritos, o *drone* é equipado por uma câmara frontal grande-angular de 180 ° com uma resolução de 720 p, no qual a comunicação é realizada através de uma rede *Wi-Fi* com uma frequência de operação de 2,4 ou 5 GHz com um alcance de até 300 metros. O *Bebop* 2 apresenta uma autonomia aproximadamente de 25 minutos [41][42].

## 3.2 *CoppeliaSIM*

O programa escolhido para simular o ambiente a três dimensões é o *CoppeliaSIM*. Este simulador multiplataforma desenvolvido pela *Coppelia Robotics* é uma ferramenta bastante versátil que confere aos seus utilizadores vários modos de programação, nos quais o controlo dos modelos simulados para além de poderem ser feitos através de aplicações externas que se ligação ao simulador através de uma *Aplication Programming Interface* (API), podem ser também feitos através de *scripts*, *plugins*, nós *ROS* ou *BlueZero*, suportando várias linguagens de programação, tais como: C/C++, *Python*, *Java*, *Matlab* ou *Octave* [43].

Tendo em conta as várias funcionalidades que o *CoppeliaSIM* apresenta, este simulador conta com quatro motores de física (*Bullet*, *ODE*, *Vortex* e *Newton*), bem como uma biblioteca com vários robôs e sensores (visão, proximidade, acelerómetro, entre outros).

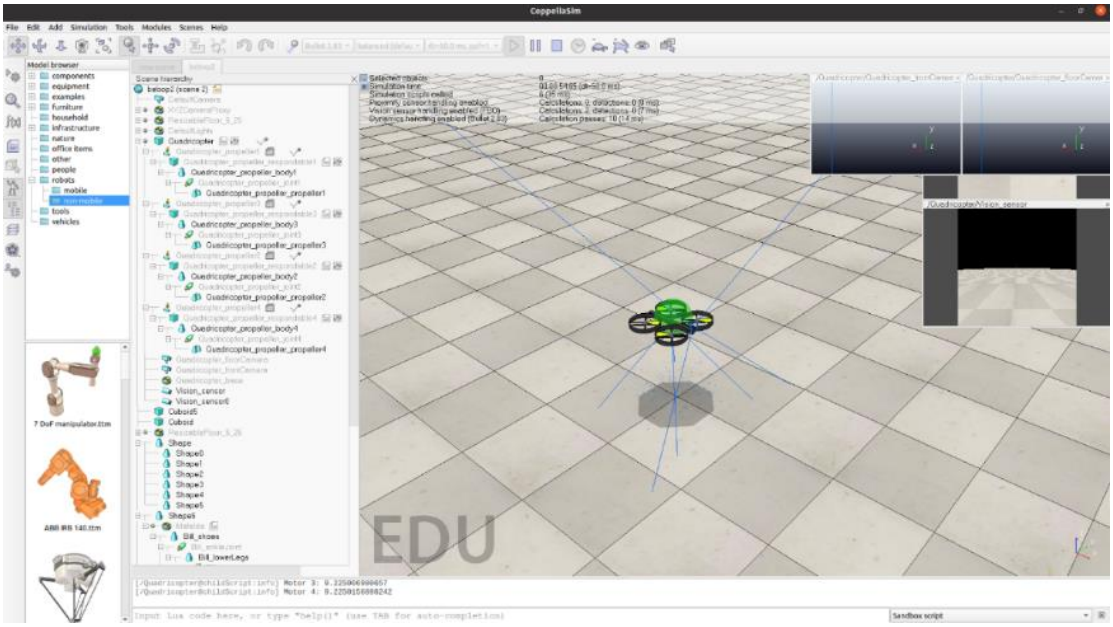



Figura 3.1 - *Quadcopter* no ambiente de simulação.

Desta forma para a realização da simulação do controlo do *drone*, como se pode observar pela figura 3.1, foi implementado no ambiente de simulação um *quadcopter* com dois sensores de visão.

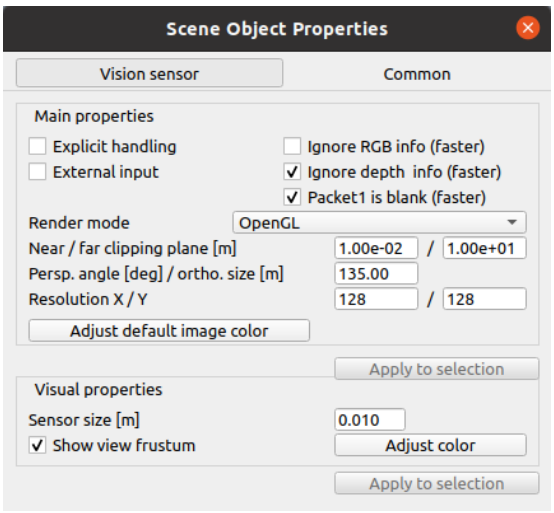



Figura 3.2 - Propriedades da câmara frontal.

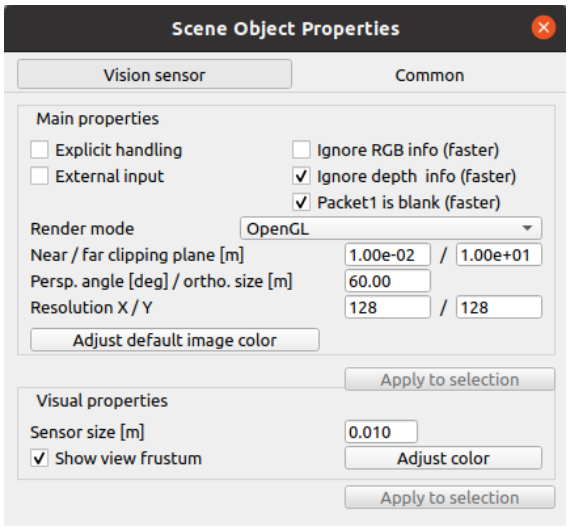



Figura 3.3 - Propriedades da câmara vertical.

O primeiro sensor de visão implementado corresponde à câmara frontal do *drone*, com um ângulo de abertura máxima de 135 ° e uma resolução de 128x128, figura 3.2. O segundo sensor de visão corresponde à câmara vertical e apresenta um ângulo de abertura de 60 ° com uma resolução de 128x128, figura 3.3. Para além disso, o *quadcopter* no ambiente de simulação foi definido um peso de 500 gramas, como se pode observar pela figura 3.4.


Figura 3.4 - Propriedades do peso do *quadcopter* no coppelia.

## 3.3 *OpenCV*

Desenvolvido pela *Intel* no ano 2000, o *OpenCV* (*Open Source Computer Vision Library*) é uma biblioteca que possui módulos de processamento de imagem, vídeo I/O, estruturas de dados, álgebra linear e interface gráfica com o utilizador, sendo deste modo uma biblioteca de desenvolvimento de aplicações mais específica para a área de visão por computadores.

Assim para este trabalho, foi utilizado o *OpenCV* para o reconhecimento de cor dos objetos.

## 3.4 *ROS – Robot Operating System*

Para a realização deste trabalho a *framework* usada foi o *ROS* (*Robot Operating System*), cujo objetivo consiste em simplificar a criação e o desenvolvimento de aplicações para *robots.* Este é um tipo de *middleware* que permite trabalhar com uma grande variedade de plataformas de robótica. O *ROS* possibilita a colaboração de diferentes entidades com a integração dos seus trabalhos.

Esta facilidade de integração deve-se em parte à arquitetura de comunicação que permite ao utilizador escolher que partes do *ROS* são úteis como também a integração de qualquer outro componente externo.

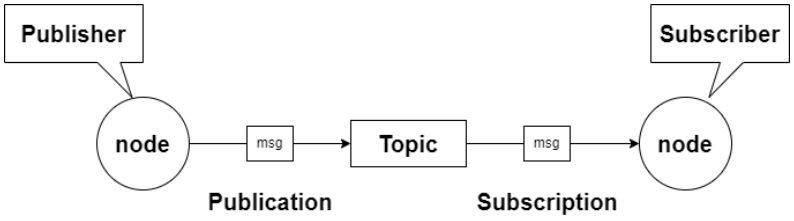



Figura 3.5 - Arquitetura de comunicação do *ROS*.

Como se pode observar pela figura 3.5, cada nó (*node*) que está presente na arquitetura de comunicação do *ROS* representa um conjunto de programas que tanto pode ser uma divisão lógica (na mesma máquina), como física (em máquinas diferentes). A comunicação é realizada com a publicação (*publication*) dos dados de um nó, definido como *publisher*, através de uma mensagem (msg) com o formato adequado para um tópico (*topic*). De seguida, para aceder a esses dados, um nó, definido como *subscriber,* só tem que subscrever (*subscription*) um tópico, e garantir que o protocolo é o mesmo para a receção da mensagem.

Em relação à arquitetura de comunicação *ROS* que este trabalho apresenta é constituída por três nós, cuja ilustração pode-se observar na figura 3.6. O *menu_drone* é um dos nós presentes na arquitetura e que corresponde a um *script* de *Pyhton* no qual, como o nome indica, é um menu onde o utilizador introduz as coordenadas para onde o *drone* tem de se deslocar. O *control_drone,* que

corresponde novamente a um *script* de *Pyhton,* é onde se realiza todo o controlo do *drone*. Por fim, o último nó presente nesta arquitetura é o *sim_ros_interface* que representa o ambiente de simulação no *CoppeliaSIM*.

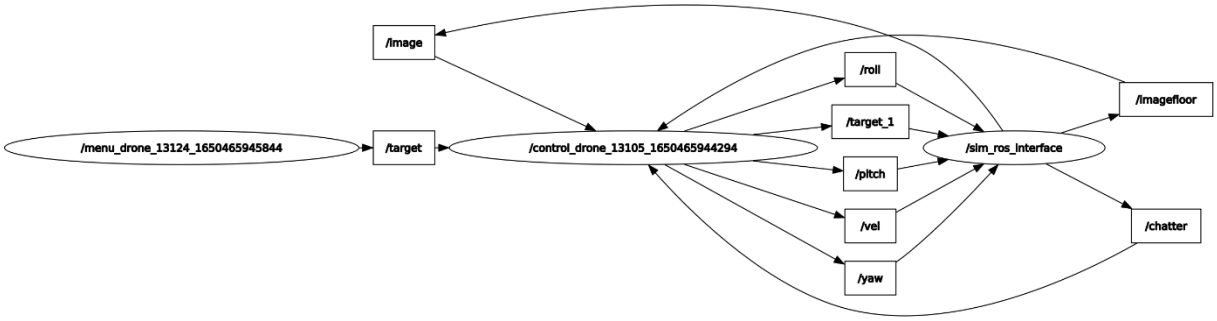



Figura 3.6 - Arquitetura da comunicação *ROS* desenvolvida neste projeto.

Através da figura 3.6 pode-se observar que são várias as comunicações *ROS* que os nós fazem entre si. Uma delas é a publicação da mensagem *target* proveniente da partilha de dados do nó *menu_drone* (*publisher*), que corresponde às coordenadas do destino desejado que o utilizador introduziu para o *drone*, no qual esses dados serão recolhidos pelo nó *control_drone* (*subscriber*). Para além disso, o nó *control_drone* e o nó *sim_ros_interface* irão interagir entre eles como *publisher* e *subscriber*. Quando o nó *control_drone* interage como *publisher*, ele partilha quatro variáveis para o cálculo das velocidades dos motores (*roll*, *pitch*, *yaw* e vel) do *quadcopter* e também das coordenadas intermédias (*target*_1), provenientes do modo de navegação. Em contrapartida, quando o nó *sim_ros_interface* interage como *publisher*, são partilhadas as coordenadas e a orientação do *quadcopter* (*chatter*) provenientes do ambiente de simulação, bem como o fluxo de dados das duas câmaras instaladas no *drone* (*image* e *imagefloor*).

## 3.5 *Google Colab*

O *Colab* é um serviço de blocos de notas alojados do *Jupyter* que promove o acesso livre para poderosas GPUs e TPUs provenientes dos servidores da *Google*. Este produto de pesquisa da *Google* permite que qualquer utilizador escreva e execute o código *Python* através do navegador, sendo especialmente adequado para *Machine Learning*, análise de dados e educação [44]. Por isto, e pela sua simplicidade de importar um conjunto de dados de imagens e treinar e avaliar um modelo de uma rede neuronal, é que o *Colaboratory* foi escolhido para este trabalho.

## 3.6 *Labelimg*

Nesta dissertação, para a criação das *labels* das imagens do *dataset* foi utilizado a ferramenta *LabelImg*. Esta ferramenta de anotação de imagem gráfica apresenta vários formatos de anotações diferentes, como é o caso do *CreateML*, *Pascal VOC* e *YOLO*, tendo este último sido o escolhido. Neste formato é criado um ficheiro de texto com o mesmo nome para cada ficheiro de imagem no mesmo diretório. Cada ficheiro de texto contém as anotações dos objetos desejados para a imagem correspondente, isto é, a classe do objeto, as coordenadas do objeto, a altura e a largura. A classe do objeto é um número de 0 a número de classes – 1. Em relação às coordenadas do objeto, a coordenada x é representada em percentagem de 0 a 1 do centro da *bounding box* em relação à largura da imagem, e a coordenada y é representada em percentagem de 0 a 1 do centro da *bounding box* em relação à altura da imagem. Por fim, do mesmo modo a altura e a largura são representadas em percentagem de 0 a 1 do centro da *bounding box* em relação à altura e largura da imagem, respetivamente.

Como se pode observar pela figura 3.7, inicialmente é selecionado através das *bounding boxes* a localização dos objetos desejados que estão presentes na imagem, bem como a classe a que cada objeto pertence. Neste caso, foram selecionados cinco *bounding boxes*, nos quais quatro representam a classe pessoa e uma a classe bicicleta.

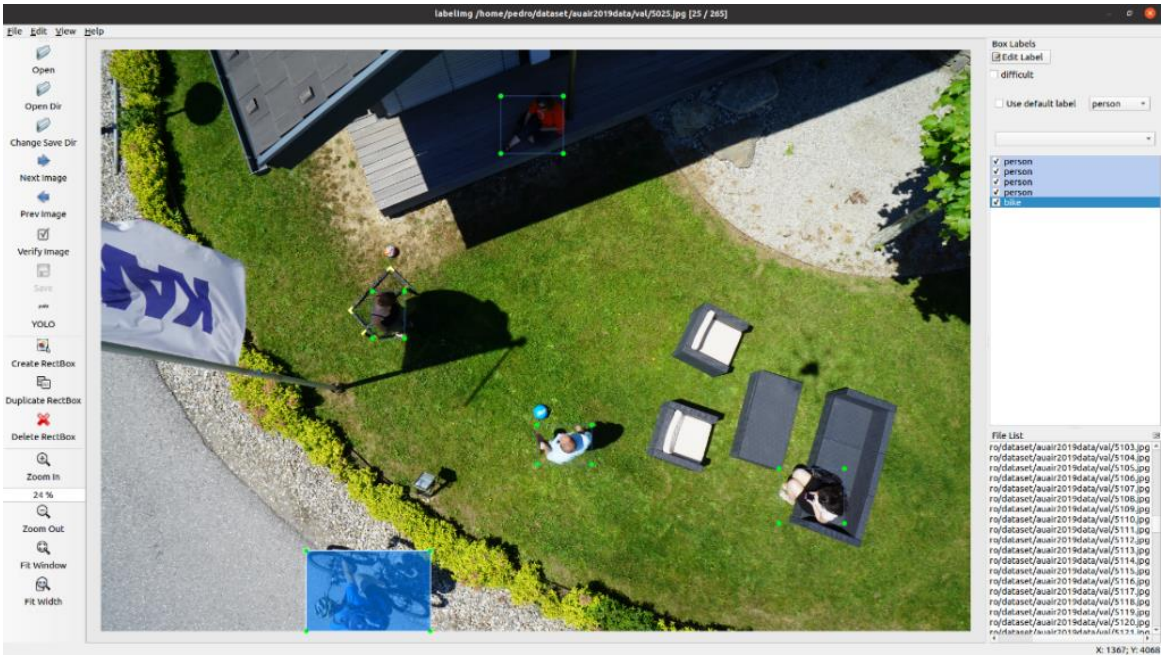



Figura 3.7 - Ambiente da ferramenta *Labelimg* (imagem proveniente do *dataset* [45]).

Por fim, é criado o ficheiro de texto, figura 3.8, com as anotações no formato *YOLO*, no qual cada linha representa um objeto da imagem correspondente.

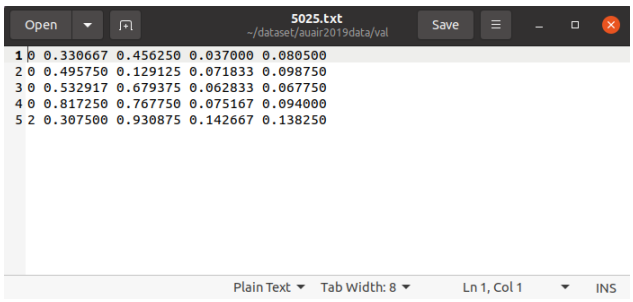



Figura 3.8 – Ficheiro de texto com as anotações no formato *YOLO* da imagem da figura 3.7.

# Capítulo 4

# Modelos e Controlo do *Drone*

Neste capítulo são descritas as técnicas implementadas para o controlo da orientação e posição do *drone* com estrutura em forma de x, bem como a estratégia utilizada para a implementação do GPS e planeamento de trajetória.

Para se poder descrever a dinâmica do *drone*, assumiu-se que este tem uma estrutura simétrica e rígida, sendo que o centro de gravidade do mesmo coincide com a origem do referencial do corpo.

## 4.1 Vetor das Entradas de Controlo

O vetor das entradas de controlo consiste em quatro variáveis (U) responsáveis por controlar todos os movimentos efetuados pelo *drone*, desde o movimento da elevação, aos movimentos *roll, pitch* e *yaw* [9]. Este vetor de entrada pode ser definido como:

$$U = [U_1 U_2 U_3 U_4] \tag{17}$$

A forma como é definido o vetor $U$ permite separar o sistema rotacional. Assim $U_1$ controla a altitude do *drone*, $U_2$ controla o ângulo *roll*, $U_3$ controla o ângulo *pitch* e $U_4$ controla o ângulo *yaw*.

Assim, o cálculo das velocidades de cada um dos motores, será a soma/subtração de cada variável de entrada:

$$Motor_{FrontRight} = U_1 + U_2 + U_3 + U_4 \tag{18}$$

$$Motor_{BackRight} = U_1 - U_2 - U_3 + U_4 \tag{19}$$

$$Motor_{BackLeft} = U_1 + U_2 - U_3 - U_4 \tag{20}$$

$$Motor_{FrontLeft} = U_1 - U_2 + U_3 - U_4 \tag{21}$$

Estes cálculos das velocidades de cada um dos motores são obtidos através da forma matricial do vetor de entrada de controlo calculada pela dissertação de Rui Martins [9].

## 4.2 Técnicas de Controlo

O sistema de controlo do *drone* é constituído por um controlador de posição, um controlador de orientação e um controlador de altura, como podemos observar pela figura 4.1.

Como o nome indica, o controlador de posição consiste no controlo da posição do *drone* no qual após ser definida a orientação do *drone* através dos comandos $\phi_{ref}$, $\theta_{ref}$ e $\psi_{ref}$, este tende a seguir para a posição desejada. Deste modo, ao contrário da orientação que é controlada através de $U_2$, $U_3$ e $U_4$ e do controlo de altura controlada por $U_1$, o controlo da posição do *quadcopter* é feito através dos ângulos $\phi$ e $\theta$.

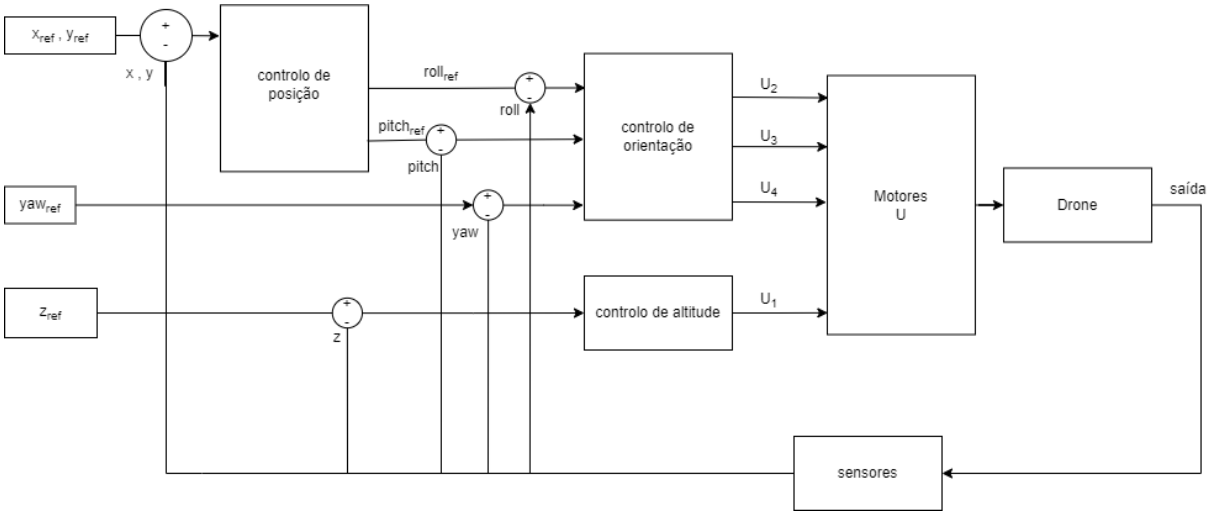



Figura 4.1 - Diagrama do sistema completo com os controladores [9].

### 4.2.1 Controlo PID

O controlo PID (Proporcional, Integral e Derivativo) é uma técnica de controlo em malha fechada que ajusta a saída da observação do erro entre a grandeza desejada e a grandeza medida. O controlador PID é caraterizado pelos coeficientes $K_P, K_i \; e \; K_d$, sendo a saída do controlador $u(t)$ em função do erro $e(t)$ dada por [9]:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt} \tag{22}$$

Para controlar a altitude do *quadcopter* é usada a seguinte expressão PID que gera a entrada $U_1$ onde o $z_{ref}$ corresponde à altitude de referência e o $z$ à altitude atual do *drone*.

$$U_1 = K_p\left(z_{ref} - z\right) + K_i \int \left(z_{ref} - z\right)dt + K_d\left(z_{ref} - z\right) \tag{23}$$

As expressões abaixo são responsáveis por controlar a orientação do *quadcopter*. Através dos ângulos de referência ($\phi_{ref}$, $\theta_{ref}$ e $\psi_{ref}$) e dos ângulos medidos ($\phi, \theta, \psi$), são produzidas as respetivas entradas de controlo ($U_2$, $U_3$ e $U_4$).

$$U_2 = K_p(\phi_{ref} - \phi) + K_i \int (\phi_{ref} - \phi)dt + K_d(\phi_{ref} - \phi) \tag{24}$$

$$U_3 = K_p(\theta_{ref} - \theta) + K_i \int (\theta_{ref} - \theta)dt + K_d(\theta_{ref} - \theta) \tag{25}$$

$$U_4 = K_p(\psi_{ref} - \psi) + K_i \int (\psi_{ref} - \psi)dt + K_d(\psi_{ref} - \psi) \tag{26}$$

Para o controlo de posição, ao contrário do controlo de orientação e de altura, o compensador produz uma aceleração de referência em vez de uma entrada de controlo. Deste modo, as coordenadas $x_{ref}$ e $y_{ref}$ correspondem à localização do destino onde se pretende que o drone se desloque e no que diz respeito as coordenadas $x$ e $y$ correspondem à localização atual do *drone*.

$$\ddot{x}_{ref} = K_p(x_{ref} - x) + K_i \int (x_{ref} - x)dt + K_d(x_{ref} - x) \tag{27}$$

$$\ddot{y}_{ref} = K_p(y_{ref} - y) + K_i \int (y_{ref} - y)dt + K_d(y_{ref} - y) \tag{28}$$

Com as acelerações de referência calculadas, através da expressão 29 que foi calculada na dissertação de Rui Martins [9], obtêm-se os ângulos de referência $\phi_{ref}$ e $\theta_{ref}$ de forma que o *quadcopter* consiga alcançar a posição comandada. A variável $m$ corresponde ao peso do *drone*.

$$\begin{bmatrix} \phi_{ref} \\ \theta_{ref} \end{bmatrix} = \frac{m}{U_1} \begin{bmatrix} -\ddot{x}_{ref} \sin(\psi) + \ddot{y}_{ref} \cos(\psi) \\ -\ddot{x}_{ref} \cos(\psi) - \ddot{y}_{ref} \sin(\psi) \end{bmatrix} \tag{29}$$

De acordo com Rui Martins [9], devido às aproximações feitas para que a expressão anterior seja válida, os ângulos $\phi_{ref}$ e $\theta_{ref}$ têm de estar limitados entre -20 ° e 20 °.

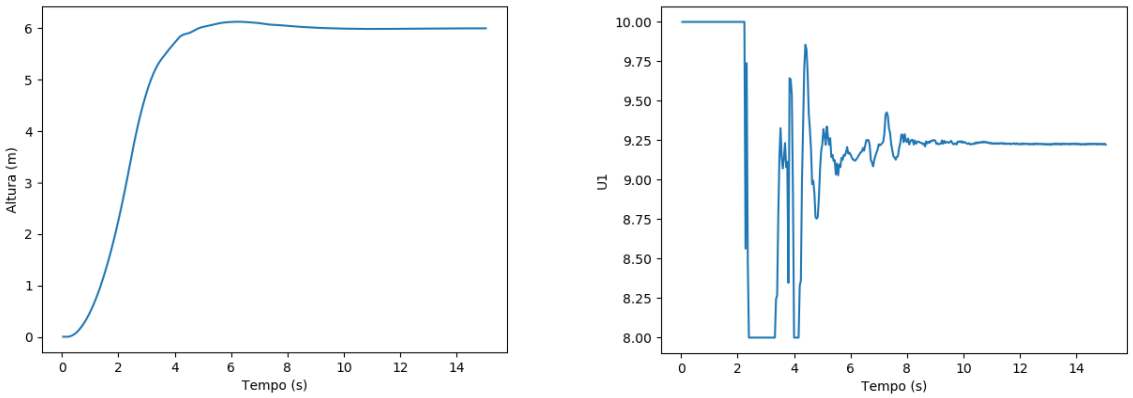
### 4.2.2 Simulação do Controlo do *Drone*

Através do ajuste manual e com o uso do método Ziegler-Nichols selecionaram-se os valores dos compensadores PID que tornaram a resposta mais próxima ao desejado. Os valores obtidos para os controladores PID utilizados na simulação estão representados na tabela que se segue.

Tabela 4.1 - Valores dos ganhos utilizados nos PID.

| | $\phi$ | $\theta$ | $\psi$ | $z$ | $x$ | $y$ |
|---|---|---|---|---|---|---|
| Proporcional ($K_p$) | 0,0003 | 0,0003 | 0,019 | 4 | 1,5 | 1,5 |
| Integral ($K_i$) | $1x10^{-7}$ | $1x10^{-7}$ | $9{,}429x10^{-6}$ | 2,5 | 0,001 | 0,001 |
| Derivativo ($K_d$) | 0,00025 | 0,00025 | 0,00805 | 4 | 15 | 15 |

Inicialmente simulou-se o controlo de altitude para uma altura de referência de 6 metros durante toda a simulação.

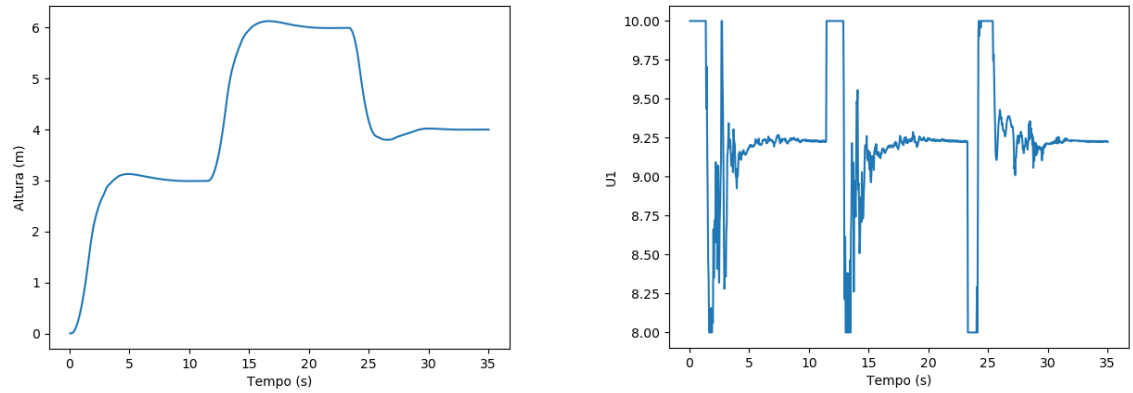



(a) (b)

Figura 4.2 - Controlo de altura com PID: a) controlo da posição de altura desejada de 6 metro; b) entrada de controlo U1.

Com a simulação do controlo de altura (figura 4.2) verificou-se que enquanto o *drone* não consegue descolar, a entrada de controlo $U_1$ atinge valores máximos, até que o *quadcopter* consiga levantar voo. Após a sua descolagem, o valor de controlo $U_1$ diminui, mas a aeronave continua a subir até atingir o valor de referência. Quando é alcançado o valor de referência para a altura, $U_1$ fica com um valor constante de aproximadamente 9,25, situação em que a força de impulsão das hélices é igual ao peso do *quadcopter*. Para além disso, perante esta subida o *drone* atingiu uma velocidade de aproximadamente de 1,5 m/s.

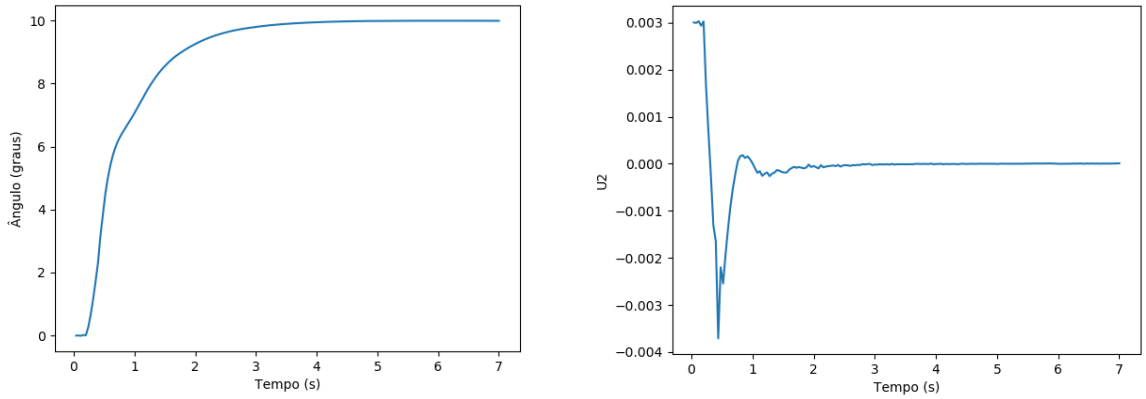
Para uma melhor demonstração simulou-se o controlo para 3 alturas diferentes (3 m, 6 m e 4 m), tendo-se obtido os seguintes gráficos da figura 4.3.




(a) (b)

Figura 4.3 - Controlo de altura com PID: a) controlo da posição de altura desejada de 3 metros, 6 metros e por fim 4 metros; b) entrada de controlo U1.

Posteriormente foi realizada a simulação do controlo do ângulo $\phi$, no qual colocou-se na referência uma entrada de 10 °.

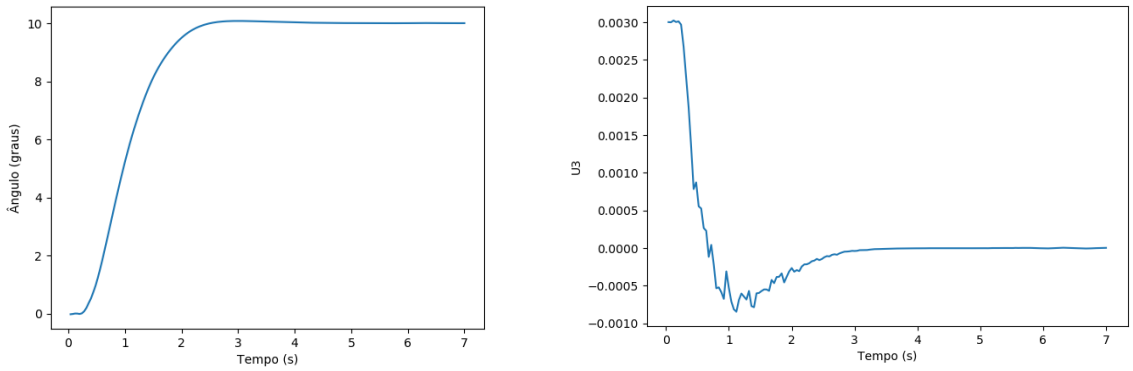



(a) (b)

Figura 4.4 - Controlo do ângulo ϕ com PID: a) controlo do ângulo *roll* a 10 °; b) entrada de controlo U2.

Como se pode observar pelo gráfico (a) da figura 4.4, verifica-se que o ângulo converge para o valor de referência de 10 graus. Após ser alcançado o valor de referência para o ângulo *roll*, $U_2$ fica com um valor constante de aproximadamente 0.

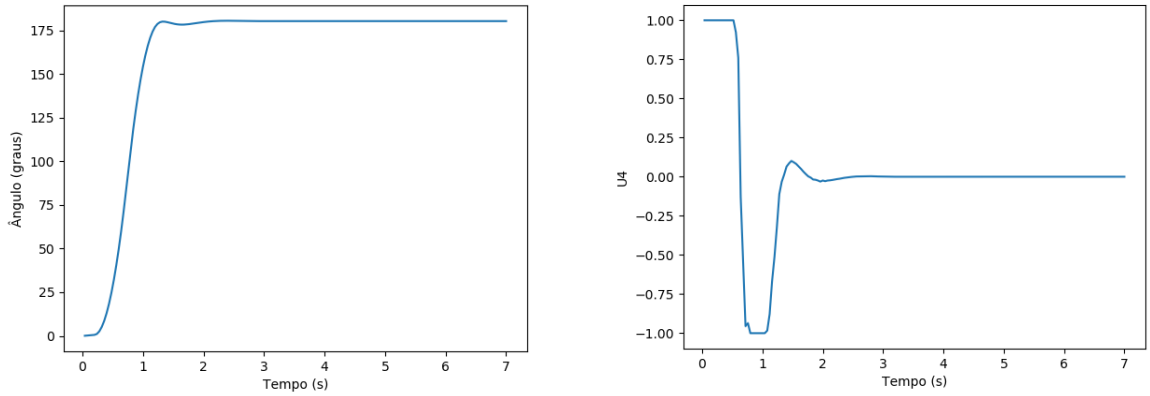
Da mesma forma, para a simulação do controlo do ângulo $\theta$ colocou-se na referência uma entrada de 10 °, tendo-se verificado no final da simulação o ângulo a convergir para o valor de referência e o $U_3$ a obter um valor constante de 0 (figura 4.5).




(a) (b)

Figura 4.5 - Controlo do ângulo θ com PID: a) controlo do ângulo *pitch* a 10 °; b) entrada de controlo U3.

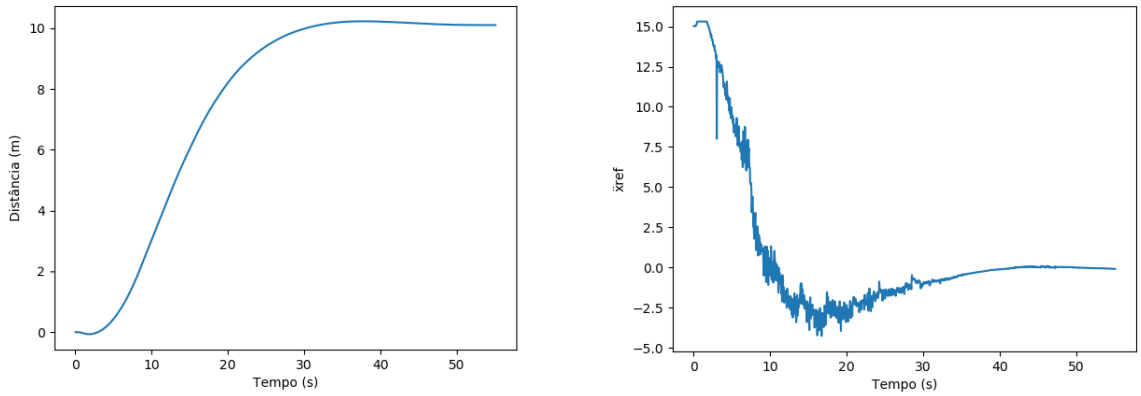
Para simular o controlo do ângulo $\psi$ colocou-se na referência uma entrada de 180 °, figura 4.6.




(a) (b)

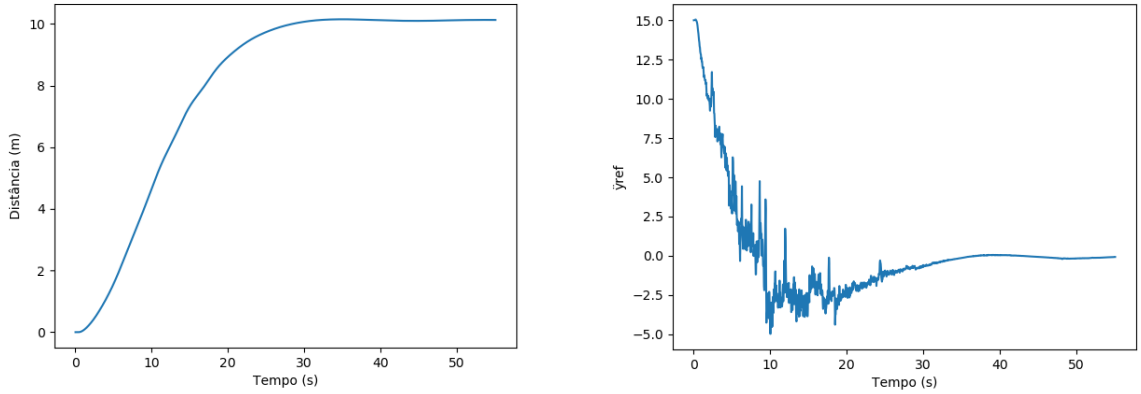
Figura 4.6 - Controlo do ângulo ψ com PID: a) controlo do ângulo *yaw* a 180 °; b) entrada de controlo U4.

Em relação ao controlo de posição, simulou-se o controlo da aceleração de referência $\ddot{x}_{ref}$ (figura 4.7) e $\ddot{y}_{ref}$ (figura 4.8) para uma deslocação de 10 metros no eixo do x e no eixo do y, respetivamente, durante toda a simulação.




(a) (b)

Figura 4.7 - Controlo da aceleração $\ddot{x}_{ref}$ com PID; a) controlo do $\ddot{x}_{ref}$ a $x_{ref}$ de 10 metros; b) entrada de controlo $\ddot{x}_{ref}$.




(a) (b)

Figura 4.8 - Controlo da aceleração $\ddot{y}_{ref}$ com PID; a) controlo do $\ddot{y}_{ref}$ a $y_{ref}$ de 10 metros; b) entrada de controlo $\ddot{y}_{ref}$.

Por fim, com as simulações anteriores verificou-se que as acelerações convergem para o valor de referência com uma velocidade aproximadamente de 0,40 m/s.

### 4.2.3 Movimento do ângulo *yaw*

Uma vez que para a prevenção de obstáculos é necessário a câmara frontal do *drone*, o movimento do ângulo *yaw* é fundamental para que o *quadcopter* esteja sempre bem direcionado de frente para o destino para o qual se tem de deslocar. Para isso, a estratégia utilizada para calcular o ângulo *yaw* para todos os quadrantes e eixos coordenados consiste na distância entre dois pontos, bem como no teorema de Pitágoras.

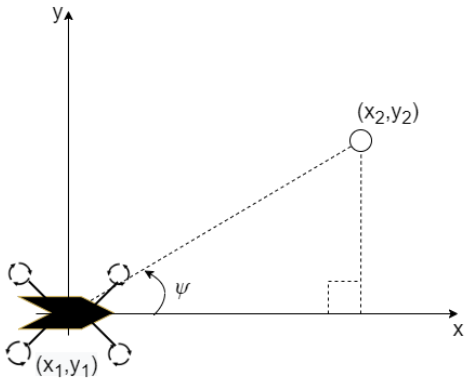


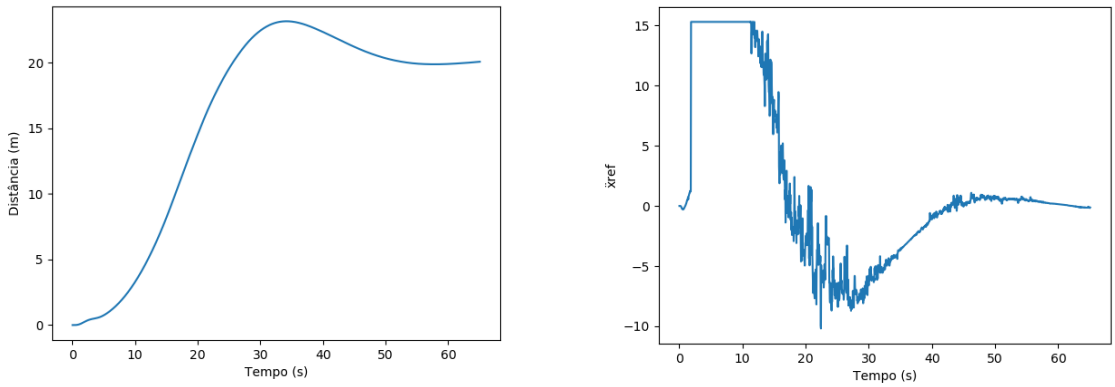

Figura 4.9 - Cálculo do ângulo *yaw* no 1 º quadrante.

Como se pode observar pela figura 4.9, com a localização do *drone* definida por $(x_1, y_1)$ e juntamente com as coordenadas do destino final $(x_2, y_2)$, através da seguinte equação é possível calcular o ângulo *yaw*:

$$\psi = \tan^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \tag{30}$$

### 4.2.4 Planeamento de trajetória

Como foi descrito anteriormente, o ajuste dos ganhos PID do controlo de posição foi feito através de várias simulações com deslocações de 10 metros com uma velocidade a rondar os 0,40 m/s. Contudo, estes ganhos obtidos tornam-se inapropriados para deslocações superiores a esta distância devido ao crescente aumento da velocidade que o *drone* atingia para estas distâncias mais longas, tal como se pode comprovar pela figura que se segue.




(a) (b)

Figura 4.10 - Controlo da aceleração $\ddot{x}_{ref}$ com PID; a) controlo do $\ddot{x}_{ref}$ a $x_{ref}$ de 20 metros; b) entrada de controlo $\ddot{x}_{ref}$.

A figura 4.10, representa então uma simulação de uma deslocação de 20 metros no eixo do x atingindo uma velocidade a rondar os 0,75 m/s. Através do gráfico (a), pode-se observar que o *drone* ultrapassa a meta dos 20 metros antes de se estabelecer nessa posição.

A estratégia utilizada para corrigir a deslocação que o *drone* percorre a mais bem como controlar a velocidade que ele atinge consiste em calcular pontos intermédios de 10 em 10 metros até que o *quadcopter* atinja o destino final. Esta estratégia, após ser estabelecida a distância total que o *drone* tem de percorrer, consiste em verificar se esta distância é superior a 10 metros. Caso seja, é calculado na trajetória o ponto intermédio a uma distância de 10 metros em relação ao *drone*. No final, são subtraídos esses 10 metros percorridos pelo *drone* à distância total.

Caso a distância seja inferior a 10 metros, o *drone* só vai percorrer a distância que lhe falta. Na situação do *drone* ter de percorrer 27 metros são calculados dois pontos intermédios de 10 em 10 metros e um no final de 7 metros. Como se pode consultar, na figura 4.11 são apresentados os cálculos realizados para esta estratégia, onde o $O_e$ representa a localização inicial do *drone*, $\psi$ o valor de *yaw* que o *drone* apresenta, $P_1$ e $P_2$ os pontos intermédios da deslocação e por fim o $P_3$ representa a localização do destino final.

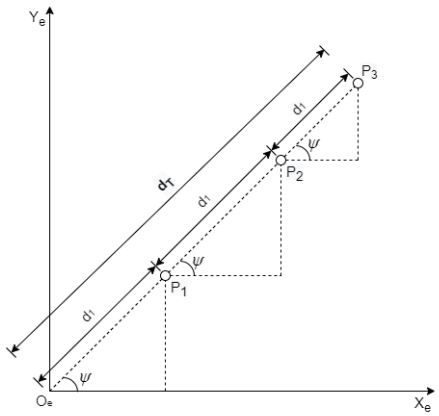



Figura 4.11 - Cálculos realizados para o planeamento de trajetória.

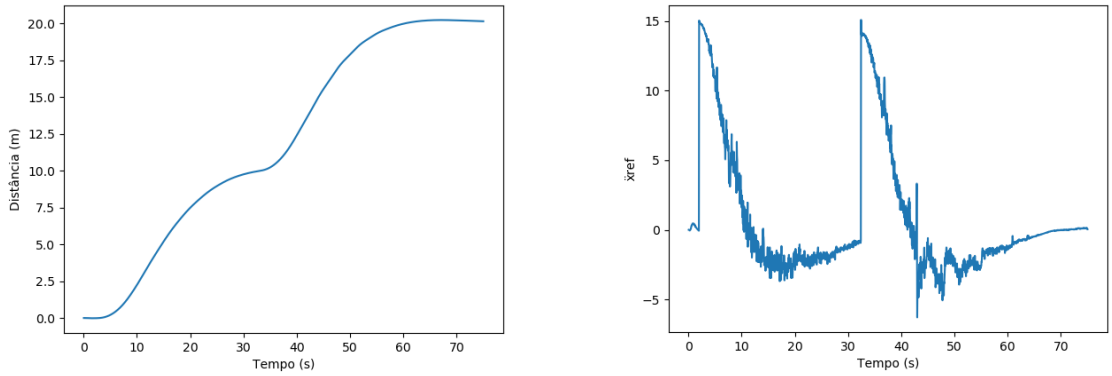
Deste modo, perante as coordenadas $O_e(x_0, y_0)$ e $P_3(x_3, y_3)$ é calculada a distância total $d_T$ que o *drone* tem de percorrer pela equação 31. Para obter as coordenadas dos pontos intermédios $P_1$ e $P_2$ é realizado o cálculo através da equação 32 com $d_1 = 10$, sendo que as coordenadas $(x_{atual}, y_{atual})$ representam a localização atual do *drone* e $(x_i, y_i)$ representam a localização para o qual o *drone* tem de se deslocar. No final, quando o *drone* se apresenta a menos de 10 metros de distância do destino final, através da mesma equação 32 é realizado o cálculo do destino final com o valor do $d_1$ igual à distância que falta percorrer.

$$\{d_T = \sqrt{(x_3 - x_0)^2 + (y_3 - y_0)^2} \tag{31}$$

$$\begin{cases} x_i = x_{atual} + (\cos\psi \times d_1) \\ y_i = y_{atual} + (\operatorname{sen}\psi \times d_1) \end{cases} \tag{32}$$

As equações explicadas anteriormente correspondem ao cálculo para esta estratégia para o primeiro quadrante e no qual foram realizados os cálculos para os restantes quadrantes e eixos coordenados. Para cada um dos novos cálculos foi implementada a equação 32 com alterações na soma, subtração e nos ângulos em alguns casos de forma a estratégia ser realizada corretamente em todas as direções.

Perante esta estratégia definida para o planeamento de trajetória, simulou-se novamente uma deslocação de 20 metros no eixo do x. Através do gráfico (a) da figura 4.12, verifica-se que o *drone* já não ultrapassa a meta dos 20 metros atingindo uma velocidade a rondar os 0,40 m/s a cada 10 metros percorridos, retificando desta forma o erro da distância que o *drone* percorria a mais, bem como o controlo da velocidade que o *drone* atinge.




(a) (b)

Figura 4.12 - Controlo da aceleração $\ddot{x}_{ref}$ com PID; a) controlo do $\ddot{x}_{ref}$ a $x_{ref}$ de 20 metros; b) entrada de controlo $\ddot{x}_{ref}$.

## 4.3 Coordenadas GPS

Após serem concluídas todas as etapas descritas anteriormente, foi realizada a conversão das coordenadas (latitude, longitude) em coordenadas no plano cartesiano (x, y). Esta conversão consiste em converter as coordenadas geográficas de uma área de referência centrada na localização inicial do *drone*. Assim, com as localizações do ponto superior esquerdo e do ponto inferior direito da área de referência, bem como com a localização do *drone*, através da fórmula obtida do método *equirectangular projection* [46] essas localizações são convertidas para posições x e y relativas ao mapa inteiro - eixo do x para a longitude; eixo do y a latitude. Este método assume que a Terra é esférica com um raio fixo (r). Deste modo, com as localizações das coordenadas no plano cartesiano dos dois pontos da área de referência relativamente à percentagem obtida do cálculo da localização relativo ao mapa inteiro da coordenada (x, y) do *drone* face à largura e altura do mapa, são obtidas as localizações x e y do *drone* relativas à área de referência.

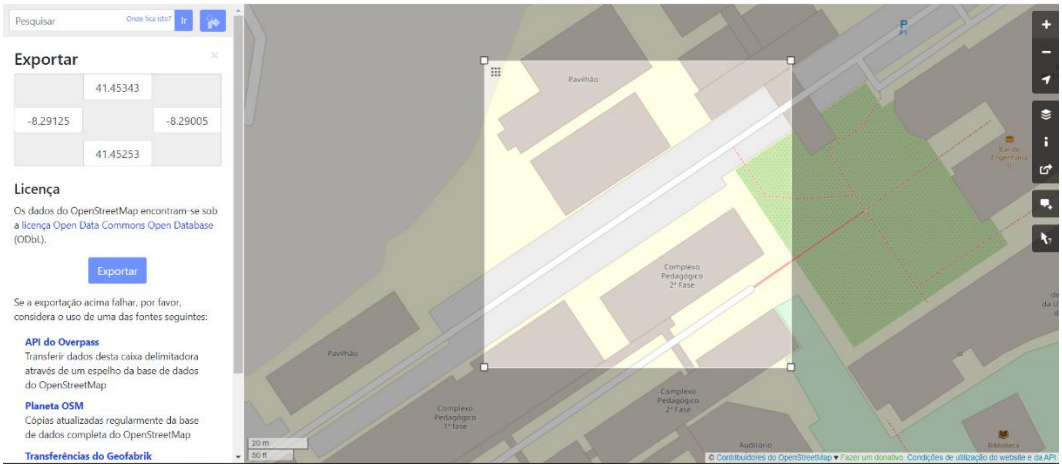



Figura 4.13 - Área selecionada no *Google maps*.

De forma a ser criada uma área de referência com o centro localizado na localização inicial que o *drone* apresenta, foi inicialmente selecionada, através do *Google Maps*, uma área com 100 metros de comprimento localizada no campus de Azurém da Universidade do Minho.

Como se pode observar pela figura 4.14, após serem obtidas as coordenadas geográficas do ponto superior esquerdo (41.45343, -8.29125) e do ponto inferior direito (41.45253, -8.29005) dessa área, figura 4.13, foi calculado o centro para esta área bem como a representação em coordenadas no plano cartesiano. Este centro que é representado pelos pontos $P_1$ e $P_4$ representa as coordenadas iniciais da localização do *drone* bem como a origem dos eixos coordenados.

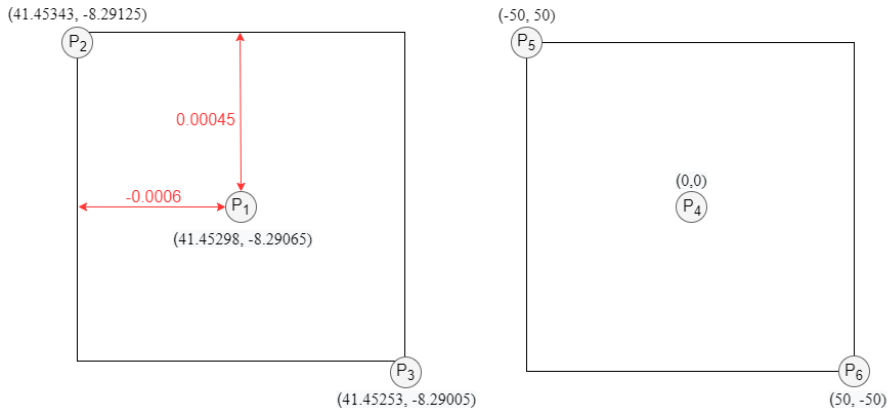



Figura 4.14 - Representações das áreas de referência das coordenadas geográficas (esquerda) e no plano (direita).

Assim, com as diferenças obtidas de latitude e longitude entre o centro da área e os dois pontos foram desenvolvidas as equações abaixo, onde, perante uma nova localização inicial do *drone* $P_1(lat_1, long_1)$ e através das equações 33 e 34 são calculados os pontos $P_2(lat_2, long_2)$ e $P_3(lat_3, long_3)$ da área de referência com 100 metros de comprimento centrado nas coordenadas iniciais do *drone*.

$$\begin{cases} lat_2 = lat_1 + 0{,}0045 \\ long_2 = long_1 + (-0{,}0006) \end{cases} \quad (33)$$

$$\begin{cases} lat_3 = lat_1 - 0{,}0045 \\ long_3 = long_1 - (-0{,}0006) \end{cases} \quad (34)$$

Desta forma, com os pontos $P_1(lat_1, long_1)$, $P_2(lat_2, long_2)$ e $P_3(lat_3, long_3)$ são calculadas as coordenadas (x, y) dos dois vértices relativas ao mapa inteiro, $P_{G2}(x_{G2}, y_{G2})$ e $P_{G3}(x_{G3}, y_{G3})$, da área de referência bem como a coordenada da localização inicial do *drone* relativa ao mapa inteiro, $P_{G1}(x_{G1}, y_{G1})$. Nas seguintes equações r representa o raio da terra, com um valor de 6371 quilómetros.

$$\begin{cases} x_{G2} = r \times long_2 \times \cos\left(\frac{lat_2 + lat_3}{2}\right) \\ y_{G2} = r \times lat_2 \end{cases} \quad (35)$$

$$\begin{cases} x_{G3} = r \times long_3 \times \cos\left(\frac{lat_2 + lat_3}{2}\right) \\ y_{G3} = r \times lat_3 \end{cases} \quad (36)$$

$$\begin{cases} x_{G1} = r \times long_1 \times \cos\left(\frac{lat_2 + lat_3}{2}\right) \\ y_{G1} = r \times lat_1 \end{cases} \quad (37)$$

Após serem realizados os cálculos anteriores é realizado o cálculo da percentagem das coordenadas x e y do *drone* relativo ao mapa inteiro em relação à largura e altura do mapa, respetivamente, através da seguinte equação.

$$\begin{cases} per_x = \left(\frac{x_{G1} - x_{G2}}{x_{G3} - x_{G2}}\right) \\ per_y = \left(\frac{y_{G1} - y_{G2}}{y_{G3} - y_{G2}}\right) \end{cases} \quad (38)$$

Por fim, com os pontos $P_5(x_5, y_5)$ e $P_6(x_6, y_6)$ é calculado pela equação 39 o ponto $P_4(x_4, y_4)$ da localização do *drone* relativo à área de referência.

$$\begin{cases} x_4 = x_5 + (x_6 - x_5) \times per_x \\ y_4 = y_5 + (y_6 - y_5) \times per_y \end{cases} \quad (39)$$

Como se pode observar pela figura 4.15, com a introdução da localização inicial GPS do *drone* com as coordenadas (41.45298, -8.29065) foram criados corretamente os dois pontos verticais da área de referência com um comprimento de 100 metros, verificando-se que o *drone* se localiza no centro desta área com as coordenadas (0,0).

```
-Coordenadas GPS da localização inicial do drone:
(latitude,longitude): 41.45298 -8.29065

-Área criada com as seguintes coordenadas GPS
Coordenadas GPS do ponto superior esquerdo:
(latitude,longitude) = ( 41.45343 , -8.29125 )
Coordenadas GPS do ponto inferior direito:
(latitude,longitude) = ( 41.45253 , -8.29005 )

-Área criada com as seguintes cooordenadas
Coordenadas do ponto superior esquerdo:
(x,y) = ( -50 , 50 )
Coordenadas do ponto inferior direito:
(x,y) = ( 50 , -50 )

Coordenadas da localização incial do drone:
x = 0.0
y = 0.0
```

Figura 4.15 - Resultados obtidos da área de referência com a introdução das coordenadas GPS do *drone*.

Após ser criada a área de referência para a conversão das coordenadas GPS do *target* do destino onde o *drone* tem que se deslocar é realizado o cálculo das coordenadas do *target* relativo ao mapa inteiro, através da equação 37, sendo obtido um novo $P_{G1}(x_{G1}, y_{G1})$. De seguida, através dos cálculos realizados anteriormente pelas equações 38 e 39 são obtidas as coordenadas x e y do *target* relativas à área de referência. Assim, perante esta implementação, realizou-se uma pequena deslocação de 12 metros no eixo do x no qual foram inseridas as seguintes coordenadas GPS do *target* (41.45298, -8.290506). Estas coordenadas GPS inseridas apresentam uma latitude referente ao eixo de origem e uma longitude referente a uma deslocação de 12 metros no eixo do x, que foi obtida através da seguinte equação onde o 50 corresponde à coordenada x do limite da área de referência, o 0,0006 que corresponde à diferença longitudinal entre os pontos $P_1$ e $P_3$ e por fim, o -8,29065 que corresponde à localização longitudinal inicial do *drone*.

$$long = -8{,}29065 + \frac{0{,}0006 \times 12}{50} \tag{40}$$

Como se pode observar pela figura 4.16, após serem inseridas as coordenadas GPS do *target* há uma conversão correta para as coordenadas x e y relativas à área de referência. Para esta conversão, as coordenadas GPS do *target* necessitam de estar dentro dos limites da área de referência.

```
Coordenadas GPS do destino final:
(latitude, longitude): 41.45298 -8.290506
Altura: 6

Conversão das coordenadas GPS do destino final:
x = 12.00000000007443
y = 0.0
z = 6.0
```

Figura 4.16 - Resultado obtido da conversão das coordenadas GPS do *target*.

# Capítulo 5

# Deteção e Desvio de Obstáculos

Neste capítulo será explicada toda a estratégia realizada para a deteção e desvio de obstáculos por parte do *quadcopter*. Assim, para a deteção de obstáculos é utilizada a visão da câmara frontal e vertical do *drone*. Estes obstáculos apresentam uma forma de um cubo com uma área de 1 metro quadrado nas suas faces e no qual, consoante a cor que estas caixas apresentam, verde ou vermelho, será realizado um desvio diferente. Deste modo, caso o *drone* esteja à frente de uma caixa de cor verde será realizado um desvio na vertical e caso a caixa seja de cor vermelha, o desvio será realizado na horizontal.

Em suma, será inicialmente explicada a forma como foi realizada a deteção de obstáculos e posteriormente serão descritas as estratégias para os desvios verticais e horizontais.

## 5.1 Deteção de obstáculos

A deteção de obstáculos consiste no reconhecimento e estimativa que um objeto alvo se encontra em relação ao *drone*. Desta forma é implementado um algoritmo de visão de deteção de cor que ao permitir calcular a área destes objetos alvos a diferentes distâncias permite posteriormente calcular uma função que estima a distância que um objeto alvo está em relação ao *drone*.

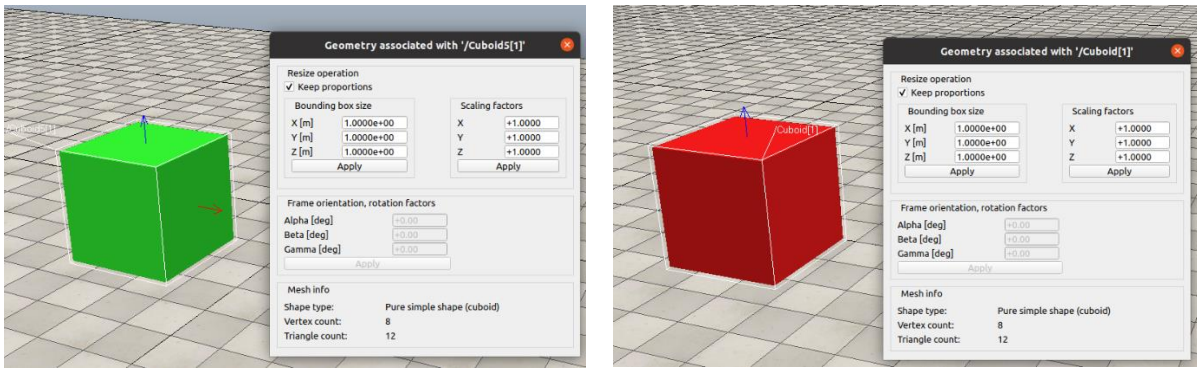



Figura 5.1 - Representações no ambiente de simulação da caixa verde (esquerda) e da caixa vermelha (direita) bem como as suas dimensões implementadas.

Como foi dito anteriormente, para esta dissertação estes objetos alvos podem apresentar uma cor vermelha ou verde. Como se pode observar pela figura 5.1, são representados no ambiente de simulação dois exemplares destes objetos alvo com 1 metro quadrado.

### 5.1.1 Algoritmo de Visão para Deteção de uma Cor

Na figura 5.2 é representado em diagrama de blocos o algoritmo de visão desenvolvido nesta dissertação.


Figura 5.2 - Diagrama de blocos do algoritmo de visão desenvolvido.

Como se pode observar pela figura 5.2, inicialmente quando uma nova imagem no espaço de cor RGB é adquirida no algoritmo, esta é filtrada por um filtro Gaussiano que permite suavizar os contornos dos objetos presentes na imagem. Para isto, é utilizada a função do *OpenCv GaussianBlur().*

Devido às características do espaço de cores RGB serem relativos à cor, não havendo desta forma um controlo da intensidade de cor, é realizada na imagem uma conversão para o espaço de cores HSV através da função *cvtColor()* do *OpenCV* que retorna uma imagem no espaço de cores pretendido. Assim, após a conversão, é realizada a segmentação da cor pretendida através da função *inRange()* do *OpenCv*, ao utilizar os valores máximos e mínimos de H, S e V.

O próximo passo deste algoritmo é a aplicação dos operadores morfológicos *open* e *close* que permitem remover algum ruído que normalmente aparece devido às condições externas. Deste modo, são utilizadas nesta etapa duas funções *morphologyEx()*, uma com *MORPH_OPEN* e outra com *MORPH_CLOSE*, associadas a um elemento estruturante através da função *getStructuringElement()*.

O último passo consiste na utilização da função *findContours()* que procura contornos na imagem permitindo encontrar os objetos alvo. É implementada de seguida a função *boundingrect()* para criar retângulos à volta dos contornos encontrados na imagem, retornando assim nessa função as coordenadas x e y, bem como a largura e altura dos retângulos.

Através das coordenadas, bem como da largura e altura que são fornecidas do retângulo criado perante o objeto presente na imagem é calculada a área do retângulo, que através de uma função que é posteriormente descrita é possível obter a distância que esse objeto está em relação ao *drone* por meio da sua área.

### 5.1.2 Cálculo da distância dos objetos

Para obter a distância a um objeto em relação ao *drone*, foi desenvolvida uma função que perante a área do objeto visualizada pela câmara do *drone* é obtido a distância entre os dois.

Para o cálculo desta função, visualizou-se a caixa verde a distâncias de 6 até 1,5 metros em relação ao *drone* (figura 5.3). Estas simulações consistiram no *drone* a voar a uma altitude de 0,5 metros de forma a estar centrado em relação ao objeto alvo.

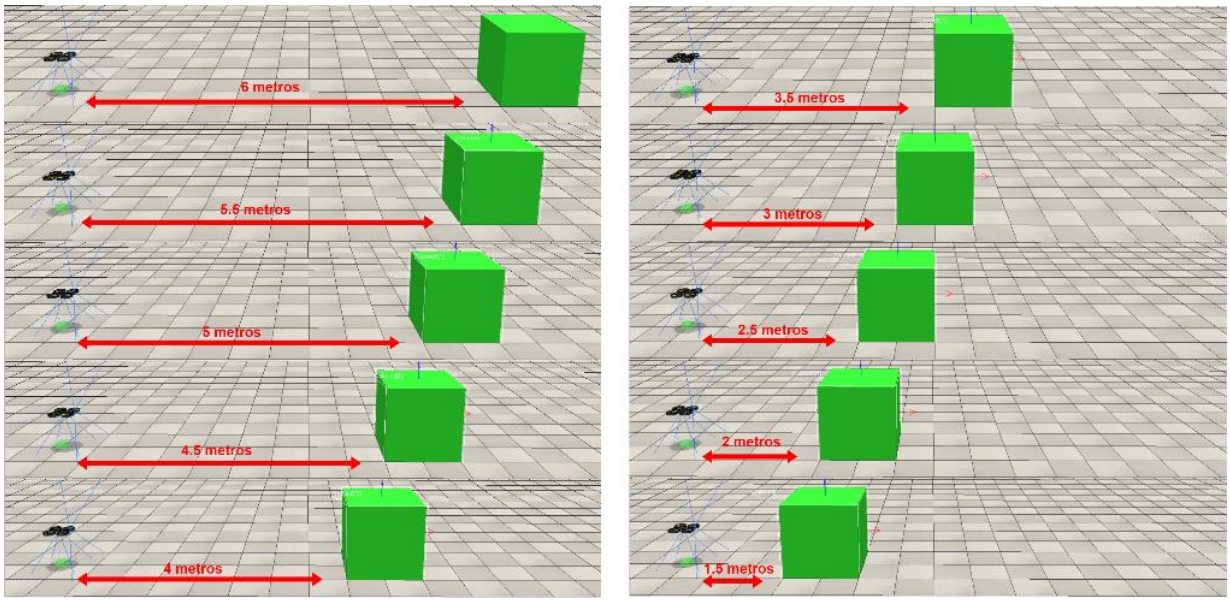



Figura 5.3 - Simulações realizadas de diferentes distâncias entre o *drone* e a caixa verde.

Como se pode observar pela figura 5.4, através do algoritmo de visão desenvolvido, é obtida a área do contorno da caixa verde a cada distância simulada. Estas distâncias vão de 1,5 metros até aos 6 metros, uma vez que a partir dos 6 metros a área do contorno do objeto alvo torna-se muito pequena.

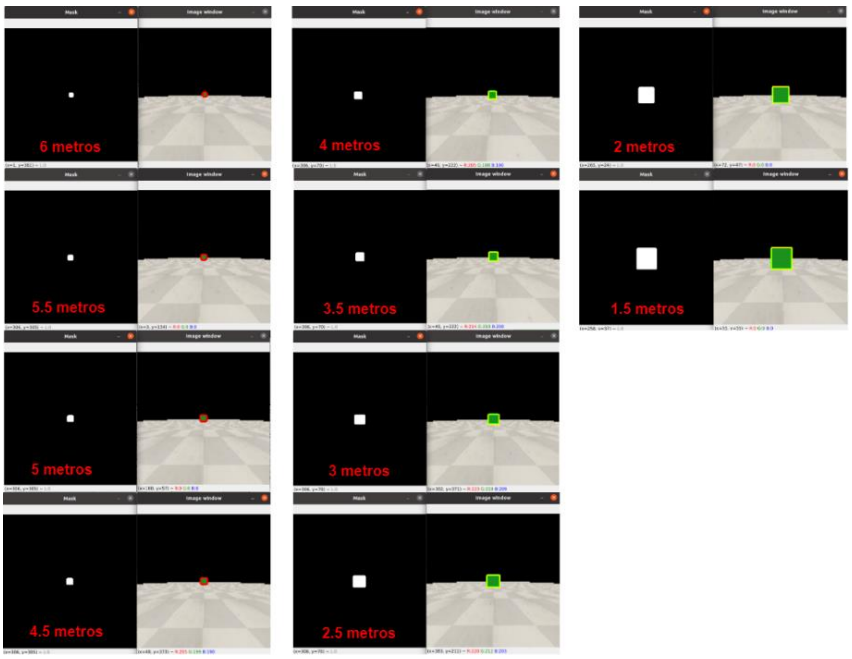



Figura 5.4 - Representações da área obtida do contorno da caixa verde para cada distância simulada.

Como resultado final desta simulação, os dados obtidos foram introduzidos para uma calculadora gráfica, e a partir disso, foi realizado uma regressão sendo obtido a equação 41 que representa a distância entre o *drone* e o objeto em relação à área do objeto que é visualizada pela câmara frontal do *drone*. Desta forma, a Distância corresponde à distância entre o *drone* e o objeto e a área que corresponde à área do objeto que é visualizada pela câmara frontal do *drone*.

$$Distância = \frac{\ln\left(\frac{\left(\left(\frac{39815{,}3938}{área}\right)-1\right)}{2{,}24144724}\right)}{0{,}9755884} \tag{41}$$

Na figura 5.5 é possível verificar a reta em tracejado utilizada para determinar a equação 41 e os pontos a azul que representam os dados obtidos da simulação e que foram utilizados para o cálculo desta regressão.

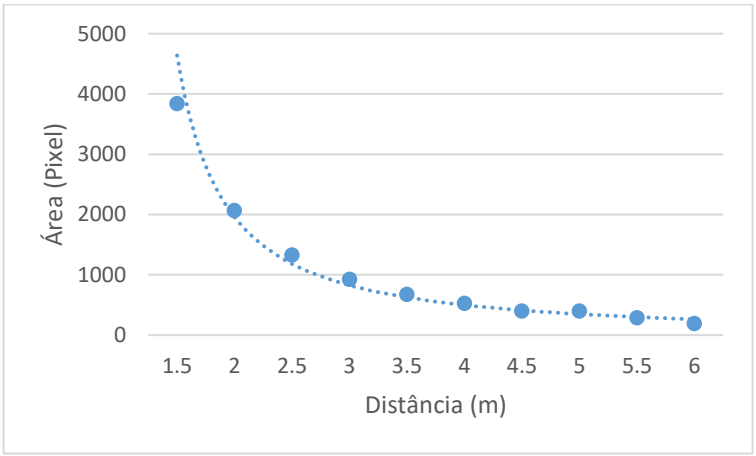



Figura 5.5 – Gráfico da área do objeto visualizada em relação à distância a que este está em relação ao *drone*.

## 5.2 Desvio de Obstáculos na Vertical

Após ser concluída a etapa da deteção de obstáculos, foi realizada a implementação do desvio dos objetos alvo.

Um dos dois desvios implementados nesta dissertação é o desvio na vertical. Em termos práticos, perante a deteção de uma caixa verde o *drone* faz um desvio vertical passando por cima desta caixa.

Desta forma, primeiramente será explicada a janela de segurança que contém a área mínima de passagem segura para o *drone* e em fase posterior será demonstrada a estratégia realizada para o desvio na vertical.

### 5.2.1 Janela de Segurança

Durante uma viagem autónoma, o *drone* vai visualizar ao longo do seu percurso vários objetos alvos que na verdade não serão uma ameaça para ele, dado estes estarem localizados em determinados pontos que o *drone* consegue passar a uma distância segura sem ter que ativar nenhuma estratégia de desvio. Desta forma, para aferir quais os objetos alvos são uma ameaça para o *drone* durante o seu percurso foi implementada uma janela de segurança baseada no artigo [47]. Esta janela de segurança nada mais é que a visualização da câmara frontal do *drone* com um retângulo implementado no meio da imagem com comprimento lateral de 2 metros que representa a área mínima de passagem segura.

Como será descrito posteriormente, o desvio vertical só é realizado quando o *drone* visualiza dentro da área mínima de passagem segura uma caixa verde a 2,5 metros de distância. Deste modo, para o cálculo da área mínima de passagem segura, é necessário calcular quantos pixéis correspondem 2 metros para essa distância.

Segundo o gráfico da figura 5.5, a área do quadrado a 2,5 metros de distância do *drone* corresponde a 1330 pixéis. Através da fórmula da área do quadrado obtêm-se que 36,5 pixéis corresponde a 1 metro. Assim, através de uma regra de três simples descobre-se que 73 pixéis correspondem a 2 metros. Desta forma, é possível localizar e representar na janela de segurança a área mínima de passagem segura com as seguintes localizações:

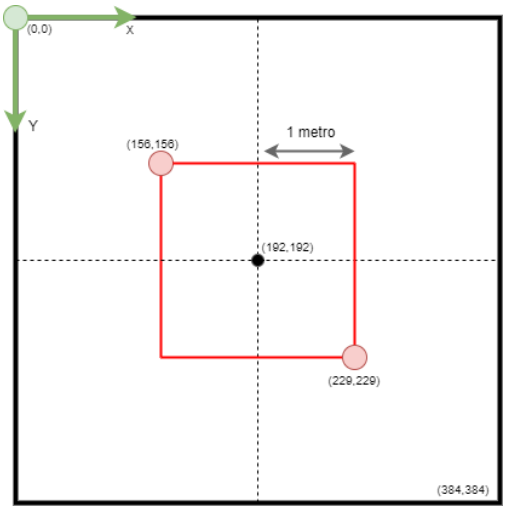



Figura 5.6 - Janela de segurança com a área mínima de passagem segura representada a vermelho.

Pela figura 5.6 pode-se observar a área mínima de passagem segura representada pelo quadrado vermelho, bem como as coordenadas dos seus limites para uma distância de 2,5 metros entre o *drone* e a caixa verde. Com a implementação desta estratégia é possível aferir para quais objetos o *drone* deve ou não deve realizar o desvio.

Como se pode observar pela figura 5.7 é demonstrada uma janela de segurança a visualizar duas caixas verdes a uma distância de 2,5 metros. Estas duas caixas estão afastadas uma da outra a uma distância de 2,5 metros.

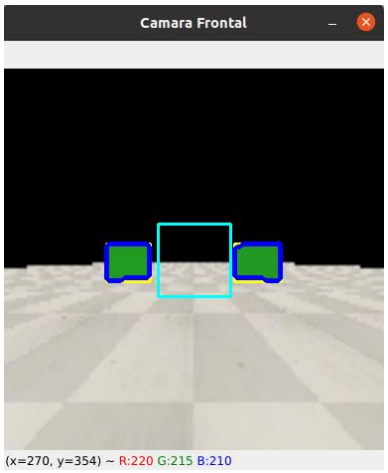



Figura 5.7 - Visualização da janela de segurança no ambiente de simulação perante duas caixas verdes.

Uma vez que nenhum dos objetos está contido dentro da área de segurança o *drone* não vai realizar nenhum desvio na vertical seguindo por sua vez a trajetória pelo meio das caixas verdes – imagem 5.8.

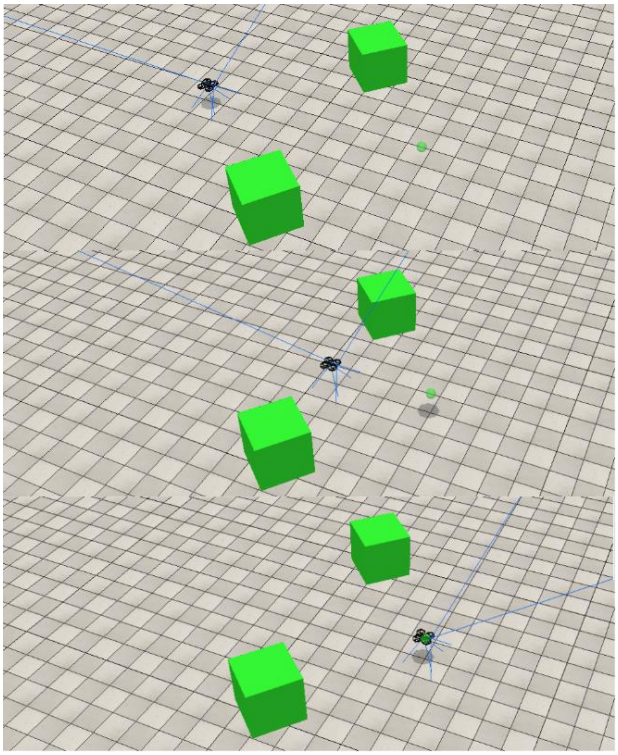


Figura 5.8 - Simulação realizada com o *drone* a passar entre duas caixas verdes.

Por outro lado, como já foi dito anteriormente, caso uma caixa verde esteja dentro da área de segurança o *drone* irá realizar o desvio na vertical, isto é, irá subir para contornar a caixa verde em segurança.

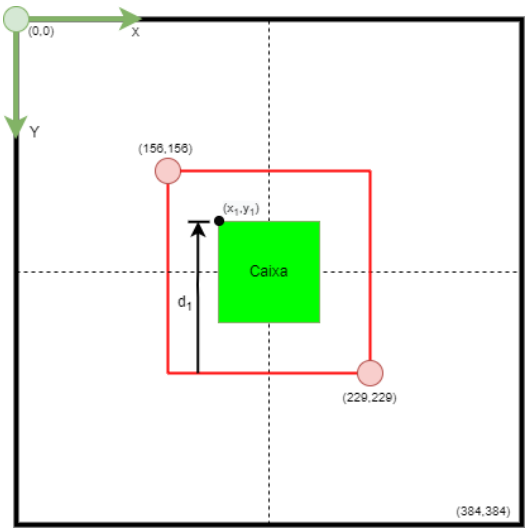



Figura 5.9 - Estratégia realizada para o cálculo da distância que o *drone* tem que subir para ultrapassar a caixa verde.

Através da coordenada $(y_1)$ da caixa verde - figura 5.9 - é possível calcular uma equação que fornece a distância $(d_1)$ que o *drone* deve subir para ultrapassar a caixa. Na equação 42, o valor 229 corresponde à coordenada da área de segurança e o valor 36,5 corresponde ao número de pixéis para um metro, como calculado anteriormente.

$$d_1 = \frac{229 - y_1}{36{,}5} \tag{42}$$

Com a implementação desta equação o *drone* ao avistar qualquer caixa verde a 2,5 metros de distância dentro da sua área de segurança vai ultrapassar este objeto de forma a que o objeto alvo não permaneça na área de segurança.

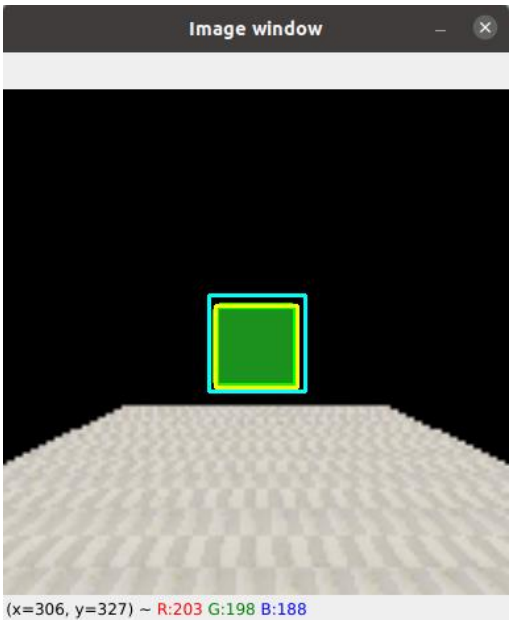



Figura 5.10 - Visualização da janela de segurança no ambiente de simulação perante uma caixa verde.

A figura 5.10 representa a visualização de uma janela de segurança de uma pequena simulação na qual o *drone*, bem como o centro da caixa verde. estavam a uma altitude de 6 metros e com uma distância entre eles de 2,5 metros. Uma vez que a caixa verde está contida na área de segurança é realizado o cálculo da distância que o *drone* tem de subir para ultrapassar o objeto.

```
--- Objeto verde detetado a 2.5 metros de distância ao drone. ---
Cálculo da distância que o drone deve subir = 1.5342465753424657
Coordenadas do ponto seguro:
x = 9.15584111213684
y = 0.0061285304836928844

Altura que o drone vai atingir para ultrapassar o objeto = 7.539916645990659
Coordenadas do drone a 2.5 metros do objeto verde:
x = 3.155841112136841
y = 0.0061285304836928844
```

Figura 5.11 - Resultados obtidos da simulação realizada.

Como se pode observar pela figura 5.11, o *drone* terá que subir 1,53 metros para ultrapassar a caixa verde, atingindo deste modo uma altura de 7,54 metros. Através da figura 5.12 pode-se observar que com esta nova altura calculada, a caixa verde não irá permanecer contida dentro da área de segurança, possibilitando ao *drone* a continuação da sua trajetória em segurança.

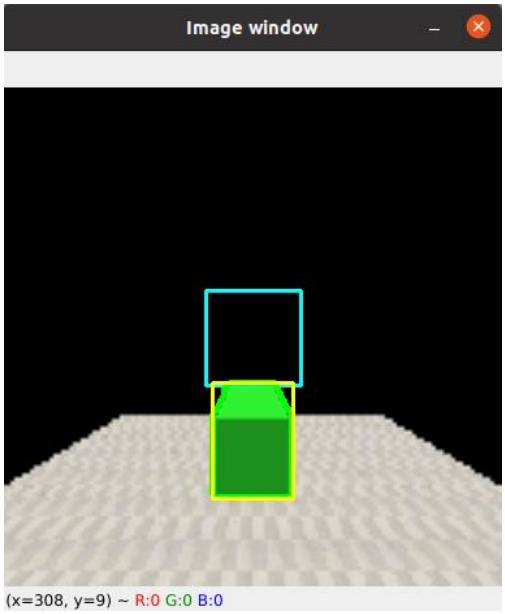



Figura 5.12 - Visualização da janela de segurança no ambiente de simulação perante o cálculo da distância a subir.

## 5.2.2 Planeamento do desvio na vertical

Como já referido anteriormente, o desvio na vertical consiste na prevenção de colisão de uma caixa verde que se apresente à frente da trajetória do *drone*. A figura 5.13 representa a estratégia desenvolvida para gerar este desvio e é baseada no artigo [47].

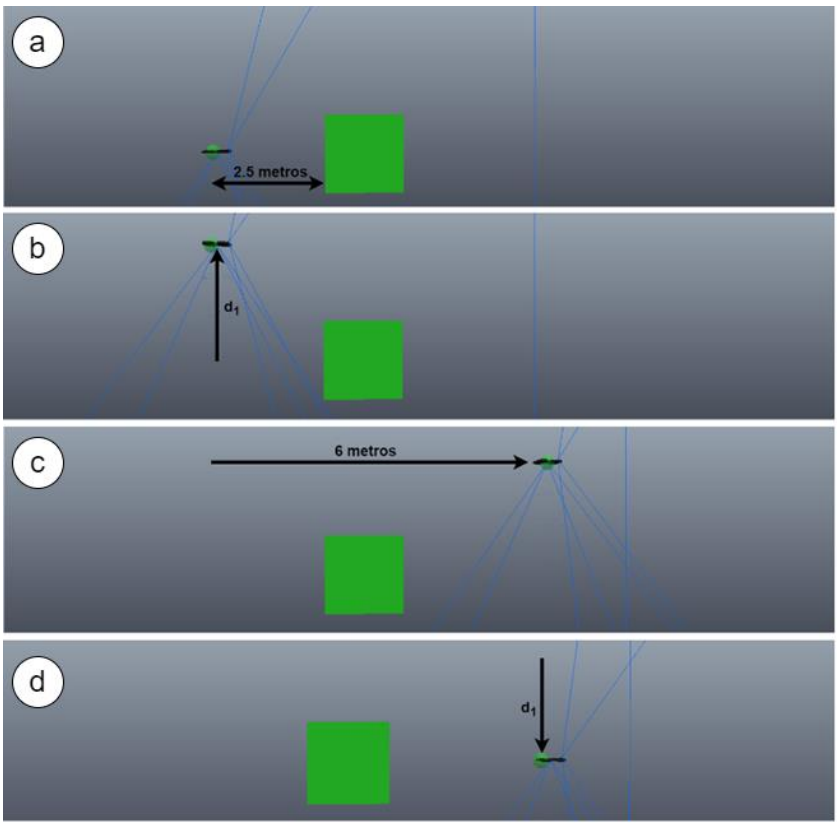



Figura 5.13 - Simulação do planeamento do desvio na vertical no ambiente de simulação.

Após ser detetada na área de segurança uma caixa verde a uma distância de 2,5 metros (figura 5.13), através da localização deste objeto alvo é calculada pela equação 42 a distância $d_1$ na qual o *drone* tem que subir para que o objeto não permaneça dentro da área de segurança. Para além disso, é determinada a localização de segurança a 6 metros de distância das coordenadas onde o *drone* detetou o objeto alvo, como se pode observar pela imagem c da figura 5.13. Por fim, o desvio vertical termina quando o *drone* volta para a sua altura inicial após chegar à localização de segurança. Na figura 5.14, que corresponde a uma visualização vista de cima, são apresentados os cálculos realizados para determinar as coordenadas da localização de segurança, onde o $O_e$ representa a localização inicial do *drone*, $\psi$ o valor de *yaw* que o *drone* apresenta e que é obtido atraves da equação 30, $P_0(x_0, y_0)$ o ponto onde o *drone* detetou a caixa verde a 2,5 metros de distância e $P_1(x_1, y_1)$ o ponto que representa a localização de segurança.

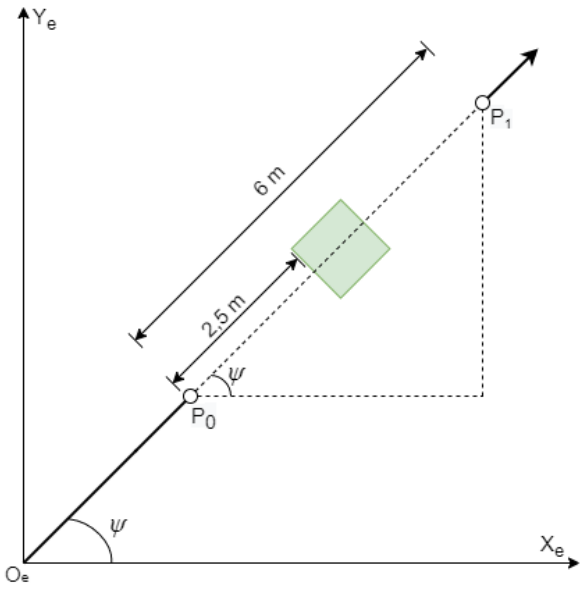



Figura 5.14 - Estratégia realizada para os cálculos do desvio na vertical.

Por meio da seguinte equação é realizado o cálculo das coordenadas do ponto $P_1(x_1, y_1)$, sendo que $d_2 = 6$ corresponde à distância entre os dois pontos, $P_0$ e $P_1$.

$$\begin{cases} x_1 = x_0 + (\cos\psi \times d_2) \\ y_1 = y_0 + (\operatorname{sen}\psi \times d_2) \end{cases} \quad (43)$$

A equação (43) corresponde ao cálculo para esta estratégia de desvio na vertical para o primeiro quadrante e no qual foram realizados os cálculos para os restantes quadrantes e eixos coordenados. Para cada um dos novos cálculos foi implementada a equação 43 com ligeiras alterações na soma, subtração e nos ângulos.

### 5.2.3 Implementação da Câmara Vertical

De forma a ser possível sobrevoar mais que um objeto ao mesmo tempo durante um desvio na vertical, foi acrescida à estratégia deste desvio a visão da câmara vertical do *drone*. Assim, para além do *drone* ter de chegar à localização de segurança, terá também, através da câmara vertical, de não visualizar nenhuma caixa verde para descer e regressar à sua altura inicial. Como tal, para o reconhecimento dos objetos alvo da câmara vertical foi implementado o algoritmo de visão desenvolvido anteriormente - figura 5.15.

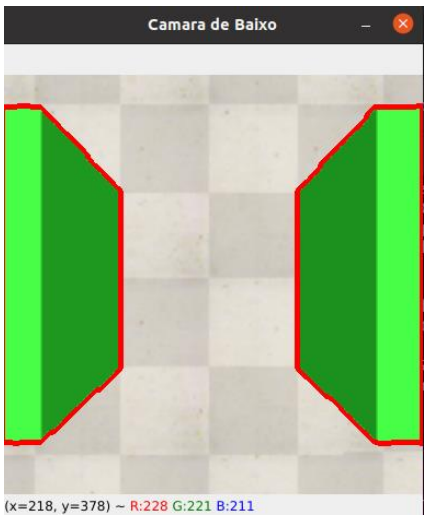



Figura 5.15 - Visualização de duas caixas pela câmara vertical do *drone* no ambiente de simulação.

## 5.3 Desvio de obstáculos na horizontal

O segundo dos dois desvios implementados nesta dissertação é o desvio na horizontal. Este desvio é realizado perante a deteção de uma caixa vermelha, para o qual consoante a localização a que esteja o objeto alvo na área de segurança, o *drone* irá contornar a caixa para a esquerda ou para a direita.

Desta forma, tal como foi feito anteriormente, será inicialmente explicada a janela de segurança que contém a área mínima de passagem segura utilizada para este desvio, e depois serão demonstradas as estratégias realizadas para o desvio na horizontal.

### 5.3.1 Janela de segurança

Pelo mesmo motivo de ser usada a janela de segurança para o desvio na vertical, foi implementado de igual forma uma janela de segurança para o desvio na horizontal. Contudo, para este desvio são utilizadas duas áreas de segurança distintas, uma vez que é necessário o reconhecimento da caixa vermelha a 3 metros e outra a 2 metros de distância em relação ao *drone*.

O desvio na horizontal inicia-se com o reconhecimento de uma caixa vermelha a 3 metros de distância dentro de uma área de segurança com 2 metros de comprimento, em que, após a deteção do objeto alvo a esta distância, o *drone* irá realizar uma travagem de 1 metro. Deste modo, o *drone*, ao ficar a 2 metros de distância em relação à caixa vermelha, através da área de segurança calculada para essa distância, irá apurar qual desvio deve realizar para contornar o objeto alvo. A escolha do desvio para a direita ou para a esquerda que o *drone* deve realizar, consiste no cálculo para o desvio mais curto, consoante a localização do objeto no interior da área de segurança.

Assim, através dos mesmos cálculos realizados para obter a área de segurança para o desvio vertical, foi calculada a área de segurança com 2 metros de comprimento para a visualização da caixa vermelha a 3 metros de distância. Perante este cálculo foi obtido que 61 pixéis corresponde a 2 metros para esta distância.

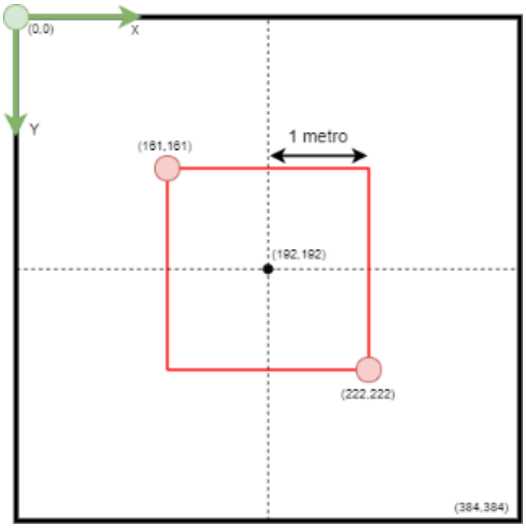



Figura 5.16 - Janela de segurança com a área mínima de passagem segura representada a vermelho para objetos a 3 metros de distância.

A figura 5.16 representa a área de segurança a vermelho para a deteção de objetos alvos a 3 metros de distância, bem como as suas coordenadas. Deste modo, o desvio horizontal irá se realizar quando o *drone* visualizar dentro desta área uma caixa vermelha a 3 metros de distância. Como se pode observar pela figura 5.17 é demonstrada uma janela de segurança a visualizar uma caixa vermelha a uma distância de 3 metros localizada nas seguintes coordenadas (7.5,0,0).

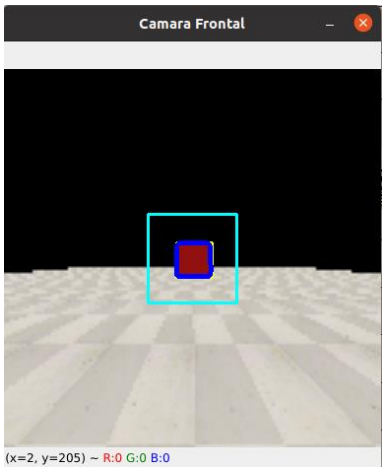



Figura 5.17 - Visualização da janela de segurança no ambiente de simulação perante uma caixa vermelha a 3 metros de distância.

Através desta simulação pode-se observar pela figura 5.18 que o *drone* visualizou a caixa vermelha com as seguintes coordenadas (4.65, 0.022, 0.5). Com este resultado verifica-se que a margem de erro deste método apresenta uma distância de 15 centímetros a mais.

```
Coordenadas do drone a 3 metros do objeto vermelho:
x =  4.6525163650512695
y =  0.02283509448170662
```

Figura 5.18 - Resultado obtido da localização que o *drone* apresentava perante a deteção da caixa vermelha.

Pela figura 5.19 é representada a área de segurança a vermelho com 2 metros de comprimento para a visualização de uma caixa vermelha a 2 metros de distância, bem como as suas coordenadas. Através dos mesmos cálculos realizados para as outras áreas de segurança, verificou-se que 91 pixéis corresponde a 2 metros para esta distância.

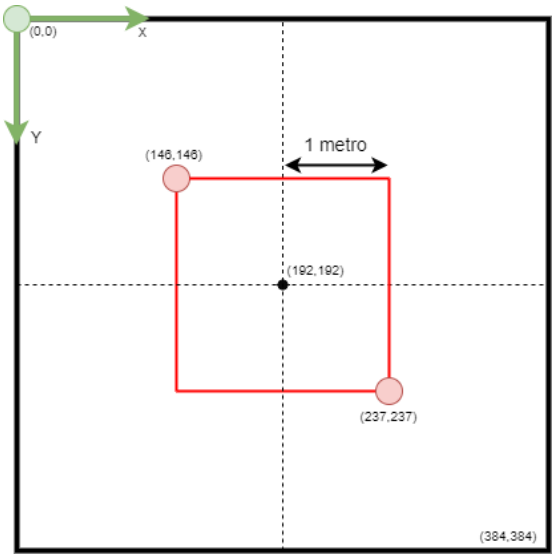



Figura 5.19 - Janela de segurança com a área mínima de passagem segura representada a vermelho para objetos a 2 metros de distância.

Como já referido, esta área de segurança tem como função aferir qual direção, para a direita ou para a esquerda, é a mais vantajosa para o *drone* efetuar o contorno do objeto alvo. Para além disso, após ser aferida a direção do desvio mais vantajoso é calculado a distância que o *drone* tem que se desviar para o objeto não permanecer dentro da área de segurança. Para isso, implementaram-se 2 limites verticais, localizados a ¼ e ¾ do quadrado de segurança, como se pode observar pela figura 5.20.

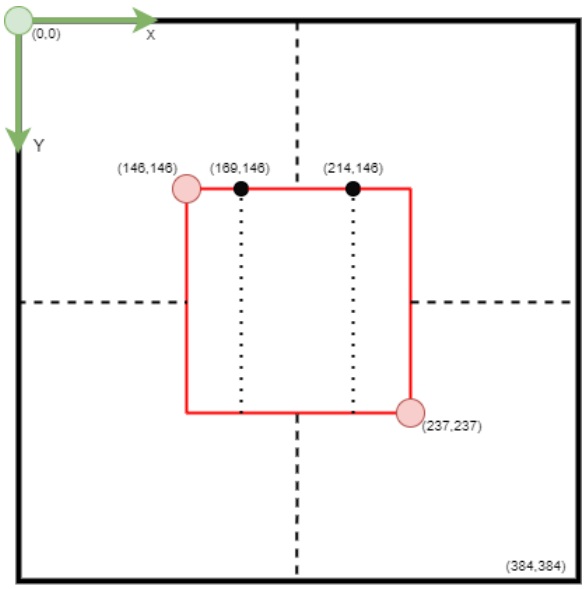



Figura 5.20 - Janela de segurança com a área mínima de passagem segura representada a vermelho com os dois limites implementados.

Consoante a localização dos vértices da caixa vermelha $(x_1, x_2)$, será aferida qual direção é a mais vantajosa para o *drone* realizar o desvio, figura 5.21. Assim, caso o vértice $x_1$ for superior ou igual a 169 o *drone* irá deslocar-se para a esquerda. Por outro lado, para que o *drone* se desloque para a direita o vértice $x_2$ tem que ser inferior a 214.

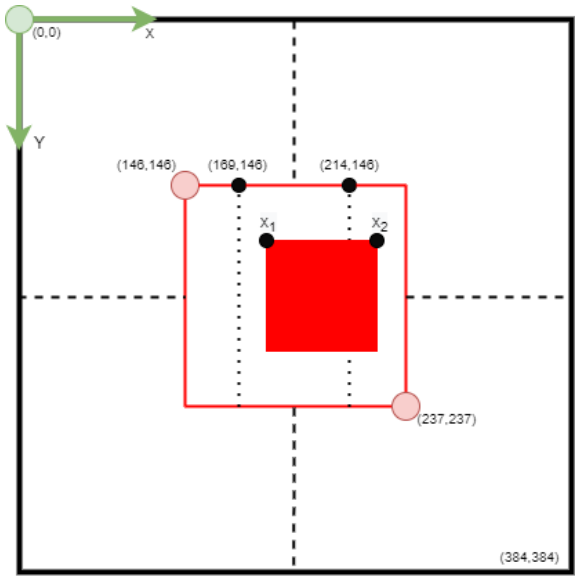



Figura 5.21 - Representação de uma caixa vermelha com os seus dois vértices $(x_1, x_2)$ dentro da área mínima de passagem.

Por fim, para o cálculo da distância que o *drone* tem que se deslocar horizontalmente foram implementadas as equações abaixo. A equação 44 é utilizada para o desvio à esquerda e a equação 45 é utilizada para o desvio à direita. Nas duas equações o valor 45,5 corresponde ao número de pixéis para um metro. Para alem disso, o valor 169 da equação 44 corresponde à coordenada x de ¼ do

quadrado de segurança e no que diz respeito ao valor 214 da equação 45 corresponde à coordenada x de ¾ do quadrado de segurança.

$$d_{\text{esquerda}} = \frac{x_1 - 169}{45{,}5} \tag{44}$$

$$d_{\text{direita}} = \frac{x_2 - 214}{45{,}5} \tag{45}$$

Após a implementação das equações, realizou-se uma pequena simulação (figura 5.22) que representa a visualização de uma janela de segurança, no qual uma caixa vermelha está deslocada 0,5 metros à esquerda em relação ao *drone*, bem como a uma distância entre eles de 2 metros.

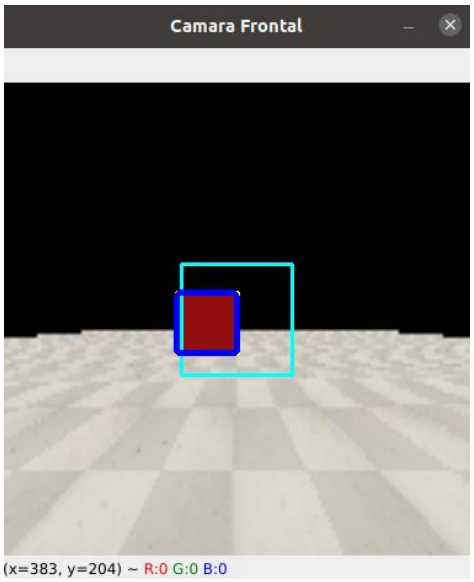



Figura 5.22 - Visualização na janela de segurança da caixa vermelha deslocada para a esquerda a 2 metros de distância do *drone*.

Como se pode observar pela figura 5.23, o *drone* irá deslocar-se 0,47 metros para a direita, tendo deste modo um erro de exatidão de 0,02 metros.

```
-- Deslocação do drone para a direita. -- 1
Distância do objeto perante o trajeto do drone =  -0.4777777777777778
```

Figura 5.23 - Resultado obtido perante a visualização da localização da caixa vermelha da figura 5.21.

### 5.3.2 Planeamento do desvio horizontal

O desvio na horizontal consiste na prevenção da colisão de uma caixa vermelha que se apresente à frente da trajetória do *drone*. Para este desvio foram implementadas duas estratégias, para a qual consoante a localização que o obstáculo apresente é realizado o contorno mais curto. Na figura 5.24, que corresponde a uma visualização vista de cima, são apresentados os cálculos realizados para a estratégia do desvio à esquerda e à direita, respetivamente tendo como referência o artigo [47].

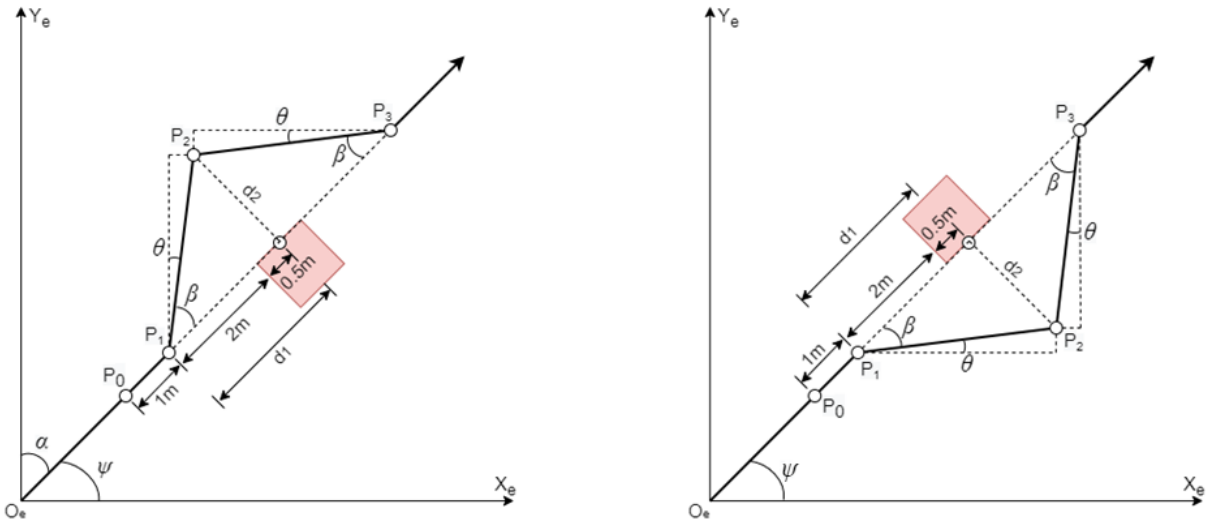



Figura 5.24 - Cálculos realizados para a estratégia do desvio à esquerda e à direita, respetivamente.

Através da figura 5.24 podem ser observadas que ambas as estratégias consistem na realização de 4 etapas representadas por 4 pontos ($P_0, P_1, P_2$ e $P_3$), onde o $\psi$ corresponde ao valor de *yaw* que o *drone* apresenta e que é obtido atraves da equação 30. Supondo que o *drone* está a realizar uma trajetória na direção $O_e\ P_0$ com uma velocidade máxima de 0,40 m/s, o ponto $P_0$ é atingido quando o *drone* visualiza pela sua janela de segurança uma caixa vermelha a 3 metros de distância. Ao atingir o ponto $P_0$ são calculadas as coordenadas do ponto $P_1(x_1, y_1, z_1)$ localizadas a 2 metros de distância do obstáculo sendo que a variável $d_1$ corresponde à distância que o *drone* apresenta em relação ao obstáculo no ponto $P_1$ mais 0,5 metros a essa distância.

Durante o trajeto entre estes dois pontos, para que o *drone* consiga realizar a travagem até ao ponto $P_1$, realizaram-se algumas alterações nos valores dos ganhos do PID do controlo de posição, para o qual, o valor do $k_p$ passou a ser 10 e 25 no $k_d$. Assim, é no ponto $P_1$ que será determinado qual desvio, para a direita ou para a esquerda, é o mais vantajoso de se realizar, tendo em conta a distância $d_2$, no qual o *drone* tem que atingir para contornar o objeto. O cálculo desta distância, consoante o desvio que se irá realizar, consiste na subtração ou na soma das distâncias $d_{esquerda}$ e $d_{direita}$ a 2,5 metros, respetivamente. Desta forma, a distância $d_2$ apresenta sempre um valor aproximado de 2 metros.

No caso de se realizar o desvio à esquerda, através das equações 46 são determinadas as coordenadas de $P_2(x_2, y_2, z_2)$ e para o cálculo das coordenadas de $P_3(x_3, y_3, z_3)$ são usadas as equações 47. Contudo, caso o desvio se realize à direita, as coordenadas de $P_2(x_2, y_2, z_2)$ são calculadas através das equações 48 e as coordenadas de $P_3(x_3, y_3, z_3)$ são obtidas pelas equações

49. Por fim, após ser atingido o ponto $P_3$, o *drone* retoma a sua trajetória concluindo deste modo o desvio horizontal.

$$\begin{cases} \beta = \tan^{-1}\left(\frac{d_2}{d_1}\right) \\ \alpha = 90 - \psi \\ \theta = \alpha - \beta \\ x_2 = x_1 + \sqrt{d_1^2 + d_2^2} \times \sin\theta \\ y_2 = y_1 + \sqrt{d_1^2 + d_2^2} \times \cos\theta \\ z_2 = z_1 \end{cases} \quad (46)$$

$$\begin{cases} \beta = \tan^{-1}\left(\frac{d_2}{d_1}\right) \\ \alpha = \psi \\ \theta = \alpha - \beta \\ x_3 = x_2 + \sqrt{d_1^2 + d_2^2} \times \cos\theta \\ y_3 = y_3 + \sqrt{d_1^2 + d_2^2} \times \sin\theta \\ z_3 = z_2 \end{cases} \quad (47)$$

$$\begin{cases} \beta = \tan^{-1}\left(\frac{d_2}{d_1}\right) \\ \alpha = \psi \\ \theta = \alpha - \beta \\ x_2 = x_1 + \sqrt{d_1^2 + d_2^2} \times \cos\theta \\ y_2 = y_1 + \sqrt{d_1^2 + d_2^2} \times \sin\theta \\ z_2 = z_1 \end{cases} \quad (48)$$

$$\begin{cases} \beta = \tan^{-1}\left(\frac{d_2}{d_1}\right) \\ \alpha = 90 - \psi \\ \theta = \alpha - \beta \\ x_3 = x_2 + \sqrt{d_1^2 + d_2^2} \times \sin\theta \\ y_3 = y_3 + \sqrt{d_1^2 + d_2^2} \times \cos\theta \\ z_3 = z_2 \end{cases} \quad (49)$$

As equações explicadas anteriormente correspondem ao cálculo para esta estratégia de desvio na horizontal para o primeiro quadrante e no qual foram realizados os cálculos para os restantes quadrantes e eixos coordenados. Para cada um dos novos cálculos foram implementadas as equações anteriores com ligeiras alterações na soma, subtração e nos ângulos.

### 5.3.3 Alteração do target

Perante a realização do planeamento de trajetória durante um voo surgem várias situações nas quais um *target* intermédio fica localizado muito próximo ou até mesmo dentro de um obstáculo, sendo este um problema para a realização de um desvio horizontal. Este é um problema uma vez que o *drone* após terminar este desvio desloca-se para o *target* intermédio. A estratégia utilizada para solucionar esta questão consistiu no cálculo de um novo *target* intermédio quando ocorre a visualização de uma caixa vermelha a 3 metros de distância dentro da área de segurança. Consoante a localização onde o *drone* se encontra no momento da deteção do objeto alvo, é calculado um novo *target* a 10 metros de distância, no qual, para o cálculo deste novo *target* é utilizada a mesma equação realizada no planeamento de trajetória.

A figura seguinte representa a realização de uma pequena simulação, onde o *drone* está a deslocar-se em direção ao *target* intermédio (representado pela esfera verde) localizado inicialmente com as coordenadas $(10, 0, 1)$. Nessa trajetória foi introduzida uma caixa vermelha com a localização de $(8, 0, 1)$. Esta localização corresponde ao centro da caixa.

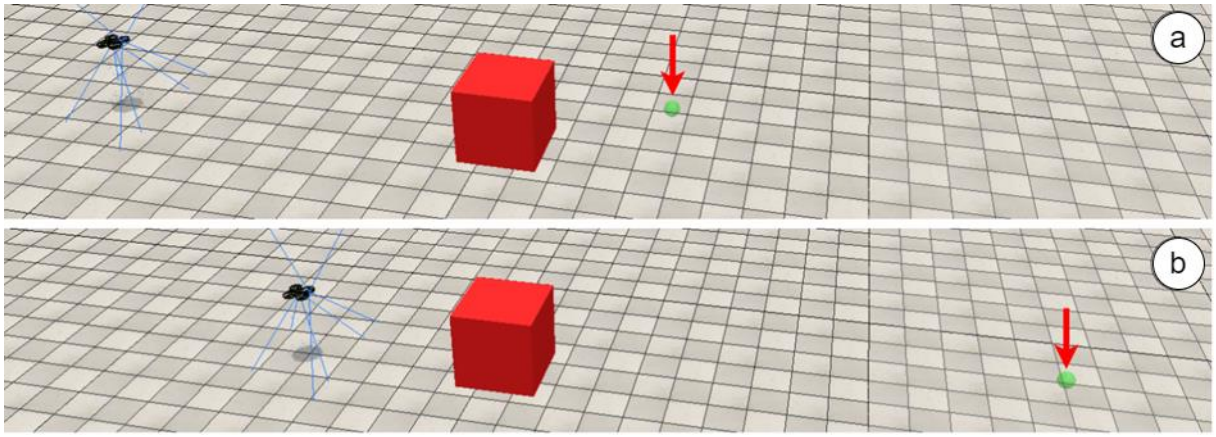



Figura 5.25 - Representação da alteração do *target* na simulação realizada.

Assim observa-se na figura 5.25, imagem b, que quando o *drone* a visualiza a caixa vermelha a 3 metros de distância altera o *target* intermédio para um novo. Através da figura 5.26, verifica-se que este novo *target* está a 10 metros de distância das coordenadas onde o *drone* visualizou a caixa vermelha a 3 metros de distância.

```
Coordenadas do drone a 3 metros do objeto vermelho:
x =  4.613525390625
y =  0.03793277218937874

Cálculo do novo target intermédio:
x =  14.613525390625
y =  0.03793277218937874
```

Figura 5.26 - Resultado obtido do novo *target* perante a simulação realizada.

# Capítulo 6

# Modo de Aterragem

Neste capítulo é explicado todo o desenvolvimento realizado para o modo de aterragem. Assim, após o *drone* chegar ao destino final, através da câmara vertical será inspecionada na área de aterragem a existência de algum objeto alvo. Caso exista algum destes objetos é realizada uma estratégia de desvio de forma que o *drone* se desloque para uma área que não permaneça com nenhum objeto alvo. Desta forma, a aterragem do *drone* será realizada quando não for detetado pela rede neuronal nenhum objeto na área de aterragem.

Em suma, neste capítulo será explicado o desenvolvimento da rede neuronal, bem como os resultados obtidos, sendo que no final será demonstrada a estratégia utilizada para efetuar a aterragem do *drone*.

## 6.1 Rede Neuronal

A primeira etapa a ser desenvolvida para a aterragem do *drone* é a implementação da rede neuronal para o reconhecimento de objetos alvo na área de aterragem. Esta implementação consistiu na criação de um *dataset*, que após criado serviu para treino e testes à rede.

### 6.1.1 *Dataset*

Através da visualização da câmara vertical, foi definido que as classes nos quais a rede neuronal tem que reconhecer são carros, pessoas e ciclistas. Para o efeito, foi criado um *dataset* que contém um conjunto de imagens com estas classes para treinar a rede.

Estas imagens foram obtidas através da seleção de 3 *datasets* já existentes, o AU-AIR [48], o ICG – DroneDataset [45] e A Benchmark and Simulator for UAV Tracking [49]. Todos os *datasets* utilizados consistem em imagens obtidas através de vídeos realizados por voos de UAVs com altitudes entre os 5 e os 35 metros. Assim, o *dataset* criado para esta dissertação apresenta 8694 imagens com as classes carro, pessoas e ciclistas, capturadas a diferentes alturas de voo, bem como a diferentes ângulos tornando a rede mais robusta.

Após ser realizada a seleção das imagens para o *dataset*, através do *software LabelImg* realizou-se a anotação de todas as imagens manualmente no formato *YOLO*. Foram realizadas estas anotações, uma vez que as entradas no algoritmo *YOLOv3-tiny* são imagens, e nas quais, para cada uma existe um ficheiro de texto que contém para cada objeto na imagem a classe_id, as coordenadas, a altura e a largura. Assim, para a anotação das imagens definiu-se que a classe_id para as pessoas é o número 0, para o carro o número 1 e para o ciclista o número 2.

Por fim, o *dataset* é dividido em dois *datasets*. Um para o treino e outro para a validação que consiste em fornecer uma avaliação imparcial do desempenho da rede.

Vale a pena notar que o *dataset* final é proveniente da realização de vários treinos, tendo sido aperfeiçoado com acréscimos de imagens de forma a obter-se resultados cada vez melhores para a deteção das classes definidas. Deste modo, no subcapítulo seguinte, será apresentado esse aperfeiçoamento realizado.

### 6.1.2 Treino

Com o *dataset* criado, a próxima etapa consiste no treino da rede. Para este treino é utilizada a rede *YOLOv3-Tiny*, devido à sua alta performance e por ser indicada para dispositivos que possuem um poder computacional limitado.

Através do *Google Colab Pro* realizaram-se todos os treinos, uma vez que esta ferramenta possibilita que o treino seja feito com poderosas GPUs tornando o treino mais rápido. Assim, para o treino da rede no *Colab* é realizada inicialmente a clonagem do repositório *Darknet*. *Darknet* é uma estrutura de rede neuronal de código aberto que apresenta, entre outras opções, o modelo *YOLOv3-tiny*, permitindo aos utilizadores usar duas dependências opcionais: *OpenCV* se os utilizadores quiserem uma maior variedade de tipos de formatos de imagens ou *CUDA* (*Compute Unified Device Architecture*) para habilitar a computação da GPU. Com o seguinte comando é clonado o *Darknet*:

```
!git clone https://github.com/AlexeyAB/darknet
```

De seguida, é configurado o Makefile para habilitar o *OpenCV*, *CUDA* e GPU. O *OpenCV* é habilitado porque após o treino será utilizado para processar imagens para prever se existe alguma classe presente. *CUDA* e GPU são habilitadas de forma a ser possível o treino através da GPU que é habilitada para *CUDA* da *Nvidia*. A configuração do *Makefile* é realizada através dos comandos abaixo, no qual inicialmente é alterado o diretório para darknet:

```
%cd darknet
!sed -i ´s/OPENCV=0/OPENCV=1/´ Makefile
!sed -i ´s/GPU=0/GPU=1/´ Makefile
!sed -i ´s/CUDNN=0/CUDNN=1/´ Makefile
```

Após a configuração do *Makefile*, o repositório é compilado com estas novas opções, através da execução do seguinte comando:

```
!make
```

De seguida, são estabelecidos todos os parâmetros bem como a estrutura da rede com todas as suas camadas ao ser fornecido o ficheiro de configuração *yolov3-tiny*.cfg. Assim, para o treino realizou-se a alteração dos valores de alguns parâmetros, entre os quais, o parâmetro *batch* que define o tamanho do lote utilizado para o treino. Este parâmetro, em vez de serem utilizadas todas as imagens do *dataset* de uma só vez para atualizar os pesos da rede, consiste em realizar uma pequena divisão de imagens do *dataset* em lotes, nos quais em cada iteração é utilizado um lote para atualizar os pesos.

Foi também alterado o valor do parâmetro *subdivisions*. Este parâmetro consiste em processar uma fração do tamanho do lote de uma só vez na GPU, em que uma iteração é concluída após todas as imagens do lote serem processadas. Para além disto, foi alterado o parâmetro classe, que representa o número de classes que o *dataset* apresenta, bem como, o parâmetro max_batches - calculado através da fórmula $(\text{número de classes} \times 2000)$ -, que especifica o número de iterações que um treino deve durar. Por fim, através da fórmula $(\text{número de classes} + 5) \times 3$ é alterado o valor do *filters*.

De seguida, são criados dois ficheiros. O primeiro ficheiro criado é o obj.names que contém o nome de cada classe do *dataset*. E o segundo ficheiro é o obj.data que contém as informações para o treino, com o seguinte formato:

```
Classes = <número de classes>
Train = <diretório do dataset de treino>
Valid = <diretório do dataset de validação>
Names = <diretório do arquivo obj.names>
Backup = <diretório no qual a rede criará um backup>
```

O *backup* consiste em armazenar os últimos pesos da rede. Desta forma, possibilita que um utilizador consiga voltar a treinar a rede com os últimos pesos guardados da rede caso o treino seja interrompido.

Após serem criados estes dois ficheiros é realizado o upload e alocação do *dataset* de treino e de validação para os diretórios definidos no ficheiro obj.data.

Para a realização do treino é usada a *transfer learning* que consiste no treino da rede a partir de um modelo pré-treinado, onde os pesos deste modelo pré-treinado são obtidos através do treino com o *dataset* COCO com cerca de 80 classes. Pode-se dizer, que a *transfer learning* consiste na melhoria de aprendizagem de uma nova tarefa através da transferência de conhecimento de uma tarefa relacionada que já foi aprendida. Assim, o comando utilizado para obter os pesos pré-treinados para o modelo *YOLOv3-tiny* foi:

```
!wget https://pjreddie.com/media/files/yolov3-tiny.weights
```

Para iniciar o treino da rede é utilizado o seguinte comando:

```
!./darknet detector train data/obj.data cfg/yolov3_training.cfg
    yolov3-tiny.weights -dont_show
```

Durante o treino, para além de serem guardados os pesos da rede no diretório *backup* no ficheiro *yolov3_training_last.weights* a cada 100 *batches*, é também calculada a mAP. Caso a mAP atinja um valor mais alto até aquele ponto, os pesos são guardados no ficheiro *yolov3_training_best.weights* no diretório *backup*. Ao ser finalizado o treino, os últimos pesos são guardados no ficheiro *yolov3_training_final.weights* no diretório *backup*.

### 6.1.3 Teste

Após o treino, foi realizado o teste à rede. Como já foi referido, o *dataset* criado consistiu no aperfeiçoamento com acréscimos de imagens de forma a obter resultados de classificação e localização das classes carro, pessoa e ciclista cada vez melhores por parte da rede. Para isso, foi desenvolvido um *script Python* para a realização de testes da rede em imagens e vídeos para ser possível comparar a evolução do *dataset* com os acréscimos de imagens. Por fim, com os pesos obtidos do treino do *dataset* final foi implementada a rede no *script* do controlo do *drone* e feito o teste no ambiente de simulação através da visualização da câmara vertical do *drone*.

Para a realização dos testes em ambos os *scripts*, a rede é implementada ao ser carregado o ficheiro do melhor peso bem como o ficheiro de configuração criado no treino. Após isso, é inserida na entrada da rede a *frame* proveniente da câmara, imagem ou vídeo. Depois da imagem passar pela rede são obtidos os resultados da classificação e localização das classes que foram detetadas, em que através do *OpenCV* são desenhadas as *bounding boxes,* bem como por cima delas um texto com a *label* e o valor de confiança de deteção consoante a informação obtida pela deteção da rede. Por fim, é

estabelecido um limite de confiança de deteção, de forma a limitar algumas deteções que podem não ser confiáveis devido ao seu baixo valor.

Para além disso, para ser verificado o desempenho da rede foi implementado um contador de FPS no *script Python*, no qual, o contador corresponde à quantidade de *frames* que são processados em 1 segundo.

## 6.2 Resultados

Neste subcapítulo são relatadas todas as etapas que foram concretizadas para se obter o *dataset*. Inicialmente são apresentados os resultados obtidos dos treinos de três *datasets* criados, e posteriormente através do *script Python*, serão demonstrados os testes realizados pelas redes obtidas. Por fim, é apresentado o *dataset* final com os resultados obtidos pelo treino, bem como os testes realizados através do *script Python* e do ambiente de simulação.

### 6.2.1 *Datasets* com 4 classes

Para o desenvolvimento do *dataset* final, foi inicialmente estipulado que o *dataset* ia conter as classes pessoa, carro, ciclista e carrinha. Contudo, devido à escassez de imagens de carrinhas obtidas por filmagens de *drones*, bem como pelo facto de não ser revelante a diferenciação entre a classe carro e carrinha para o problema desta dissertação, optou-se por retirar essa classe no *dataset* final. Assim, os três *datasets* criados apresentam 4 classes, no qual, a classe id para pessoa corresponde ao número 0, para carro o número 1, para ciclista o número 2 e para a carrinha o número 3.

#### 6.2.1.1 Parâmetros do Treino

Para a realização do treino dos *datasets* com 4 classes foram estipulados os valores dos parâmetros de treino representados pela seguinte tabela.

Tabela 6.1 - Parâmetros do treino.

| Parâmetros | Valor |
|---|---|
| *Batch* | 64 |
| *Subdivisions* | 16 |
| *Max_batches* | 8000 |
| Classes | 4 |
| *Filters* | 27 |

### 6.2.1.2 Resultados do Treino

Como o objetivo da dissertação é criar uma rede para detetar e localizar as classes definidas através da visualização da câmara vertical do *drone*, foram inicialmente selecionadas imagens com as classes pessoa, carro, ciclista e carrinha com um ângulo de visualização de 90 graus. No total, para este primeiro *dataset,* foram selecionadas 4601 imagens divididas em dois *datasets*: um para o treino com 4409 imagens e outro para a validação com 192 imagens.

```
class_id = 0, name = person, ap = 68.49%         (TP = 91, FP = 36)
class_id = 1, name = car, ap = 97.77%            (TP = 266, FP = 14)
class_id = 2, name = bike, ap = 28.72%           (TP = 5, FP = 9)
class_id = 3, name = van, ap = 100.00%           (TP = 6, FP = 5)
```

Após a realização de vários treinos com este *dataset*, são obtidos os resultados anteriores do melhor treino, no qual, são apresentados os valores da precisão média (ap) de cada classe, bem como os valores dos verdadeiros positivos (TP) e falsos positivos (FP). Verifica-se que para a classe pessoa (*person*) obteve-se uma *precision* de 71,7 %, para a classe carro (*car*) de 95 %, para a classe ciclista (*bike*) de 35 % e para a classe carrinha (*van*) de 54 %. Para além destes resultados foi também obtida uma precisão média com limite de 0,5 IoU de 73,74 %.

```
mean average precision (mAP@0.50) = 0.737441, or 73.74 %
```

Perante os resultados obtidos do treino da rede com o primeiro *dataset*, foram acrescentadas 814 imagens ao *dataset* de forma a melhorar os valores de *precision* das classes pessoa, ciclista e carrinha. Perante a escassez de imagens das classes ciclista e carrinha para um ângulo de visualização de 90 graus, algumas das novas imagens destas duas classes apresentam uma variação de ângulo de visualização. Deste modo, este segundo *dataset* criado contém 5223 imagens, no qual, para o *dataset* de treino apresenta 4958 imagens e para o de validação com 265 imagens.

```
class_id = 0, name = person, ap = 80.35%         (TP = 171, FP = 41)
class_id = 1, name = car, ap = 98.59%            (TP = 265, FP = 12)
class_id = 2, name = bike, ap = 93.29%           (TP = 43, FP = 7)
class_id = 3, name = van, ap = 100.00%           (TP = 6, FP = 4)
```

Com a realização de novos treinos à rede com o segundo *dataset*, verifica-se que para a classe pessoa (*person*) obteve-se uma *precision* de 80,7 %, para a classe carro (*car*) de 95,7 %, para a classe ciclista (*bike*) de 86 % e para a classe carrinha (*van*) de 60 %. Para além da rede ter obtido uma precisão média com limite de 0,5 IoU de 93,06 %.

```
mean average precision (mAP@0.50) = 0.930605, or 93.06 %
```

Comparativamente ao primeiro *dataset*, o segundo apresentou resultados efetivamente melhores para todas as classes, concluindo que o *dataset* ao conter imagens com vários ângulos de visualização tornou-se benéfico para a deteção e localização de objetos pela visualização da câmara vertical. Perante estes resultados são acrescentadas 3736 novas imagens com vários ângulos de visualização

ao segundo *dataset* de forma a melhorar os resultados no treino da rede para todas as classes. Este novo *dataset* contém 8694 imagens, sendo que para a realização do treino, para além de ser utilizado um *dataset* de treino com 8429 imagens, é utilizado o *dataset* de validação anterior com 265 imagens.

```
class_id = 0, name = person, ap = 79.75%          (TP = 169, FP = 43)
class_id = 1, name = car, ap = 98.87%             (TP = 266, FP = 12)
class_id = 2, name = bike, ap = 91.14%            (TP = 41, FP = 8)
class_id = 3, name = van, ap = 94.44%             (TP = 6, FP = 3)
```

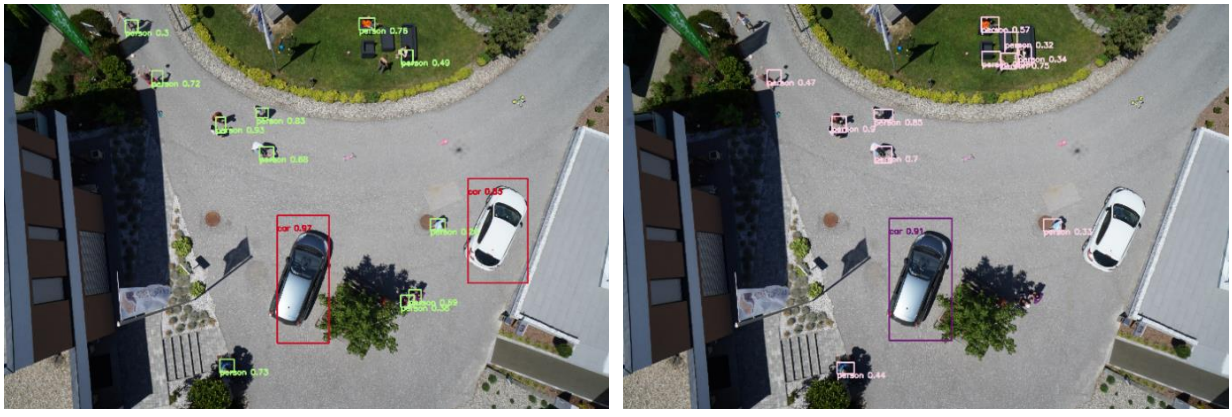
Com os resultados obtidos do melhor treino realizado à rede com o terceiro *dataset*, verifica-se que para a classe pessoa (*person*) obteve-se uma *precision* de 79,7 %, para a classe carro (*car*) de 95,7 %, para a classe ciclista (*bike*) de 83,6 % e para a classe carrinha (*van*) de 66,7 %. Obteve-se também uma precisão média com limite de 0,5 IoU de 91,05 %.

```
mean average precision (mAP@0.50) = 0.910522, or 91.05 %
```

Como se pode observar, apesar do terceiro *dataset* conter várias imagens de diversos ângulos de visualização, os resultados obtidos pelo treino da rede com este *dataset* apresentam valores muito idênticos comparativamente ao treino anterior. Deste modo, de forma a apurar qual dos dois *datasets* seria o mais vantajoso para esta dissertação foram realizados vários testes com os pesos obtidos dos treinos à rede.

### 6.2.1.3 Resultados dos Testes

Através do *script Python* desenvolvido foram realizados os testes às redes que foram treinadas com o segundo e terceiro *dataset*. Os primeiros testes realizados consistem na classificação e localização obtidas por estas duas redes perante a mesma imagem.



(a) (b)

Figura 6.1 - Resultados obtidos do teste às redes com o segundo (a) e terceiro (b) *datasets* (imagem proveniente do *dataset* [45]).

Deste modo, foi introduzido na entrada de ambos os modelos uma imagem que apresenta 2 carros e 13 pessoas. Como se pode observar pela figura 6.1, a rede obtida pelo treino com o segundo

*dataset* (a) apresentou uma melhor classificação e localização comparado ao teste da rede obtida pelo treino com o terceiro *dataset* (b). Apesar das sombras presentes na imagem, a rede do segundo *dataset* (a) reconheceu os dois carros e as 10 das 13 pessoas presentes na imagem. Em relação ao teste da rede do terceiro *dataset* (b), o resultado não foi o esperado, dado a rede apenas ter reconhecido 1 carro e 6 pessoas na imagem.

(a) (b)

Figura 6.2 - Resultados obtidos do teste às redes com o segundo (a) e terceiro (b) *datasets* (imagem proveniente do *dataset* [49]).

De seguida, foi introduzido na entrada dos modelos uma imagem com 3 pessoas com uma variação no ângulo de visão. Através da figura 6.2, pode-se observar que ambas as redes detetaram as 3 pessoas. Contudo, a rede obtida com o treino com o terceiro *dataset* (b) apresenta resultados significativamente melhores que a rede do segundo *dataset* (a). Este resultado é esperado uma vez que o terceiro *dataset* apresenta imagens com vários ângulos de visão da classe pessoa comparativamente ao segundo *dataset* que só apresenta imagens desta classe com um ângulo de visão de 90 graus.

De forma a testar a classificação e localização da classe ciclista, foi introduzida na entrada para ambas as redes a seguinte imagem (figura 6.3).

(a) (b)

Figura 6.3 - Resultados obtidos do teste às redes com o segundo (a) e terceiro (b) *datasets* (imagem proveniente do *dataset* [45]).

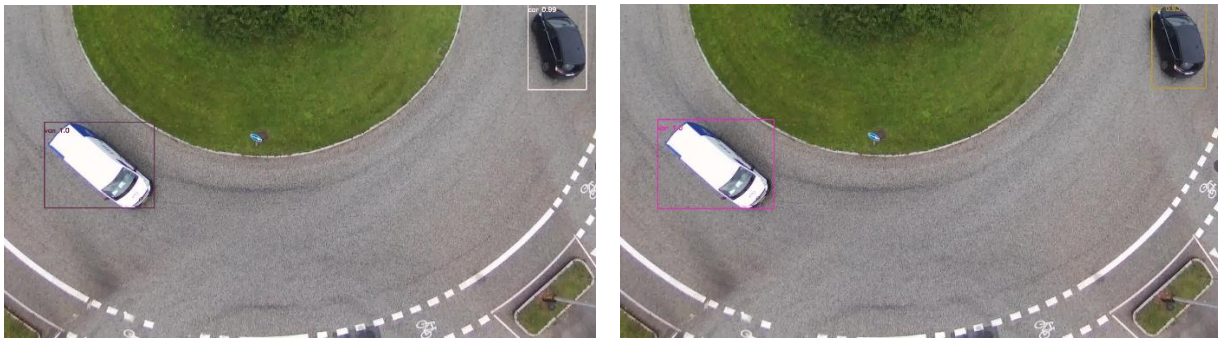
Como se observa pela figura anterior, apesar de existir na imagem bicicletas, ambas as redes detetaram corretamente a localização do ciclista. Contudo a rede do terceiro *dataset* (b) apresenta um valor superior de confiança de deteção comparativamente à rede do segundo *dataset* (a). De seguida foi introduzida na entrada das redes outra imagem com ciclistas para um outro ângulo de visão (figura 6.4).


(a) (b)

Figura 6.4 - Resultados obtidos do teste às redes com o segundo (a) e terceiro (b) *datasets* (imagem proveniente do *dataset* [49]).

Na figura 6.4 pode-se observar que ambas as redes dos dois *datasets* obtiveram valores acima dos 90 % de confiança de deteção. Este resultado deve-se por ambos os *datasets* apresentarem imagens com vários ângulos de visão da classe ciclista, sendo que a rede do terceiro *dataset* apresenta mais imagens desta classe.

Da mesma forma, foi realizado o teste às redes dos dois *datasets* para a classe carrinha.

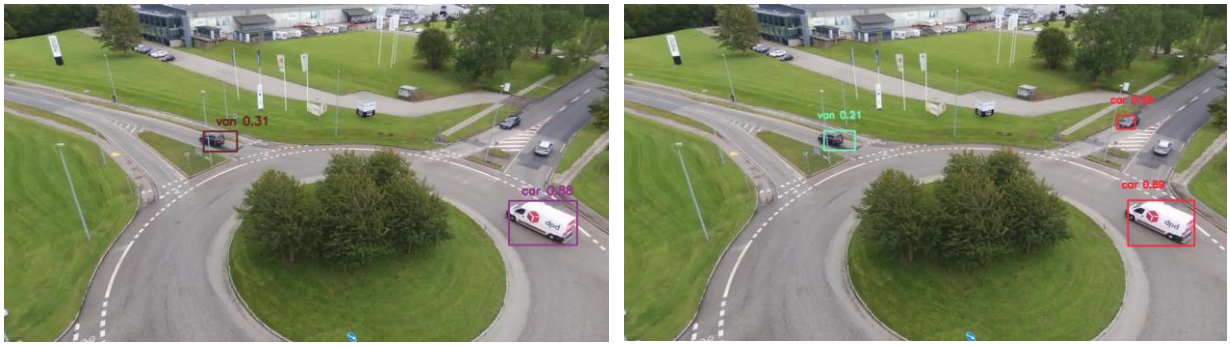


(a) (b)

Figura 6.5 - Resultados obtidos do teste às redes com o segundo (a) e terceiro (b) *datasets* (imagem proveniente do *dataset* [48]).

Com a introdução da imagem da figura 6.5 na entrada das duas redes, verifica-se que ambas as redes detetaram e localizaram bem a carrinha e o carro presentes na imagem. Contudo, devido aos valores obtidos da *precision* da classe carrinha dos treinos dos dois *datasets*, não era esperado um valor de confiança tão elevado para esta classe.

Posteriormente foi introduzido com outro ângulo de visão na entrada das duas redes outra imagem a conter uma carrinha e três carros (figura 6.6).



(a) (b)

Figura 6.6 - Resultados obtidos do teste às redes com o segundo (a) e terceiro (b) *datasets* (imagem proveniente do *dataset* [48]).

Através da figura 6.6, visualiza-se que a rede do segundo *dataset* (a) não conseguiu detetar e localizar corretamente nenhum dos veículos presentes na imagem. Em relação ao teste da rede do terceiro *dataset* (b), é reconhecido corretamente apenas um carro, sendo que ambas as redes não foram capazes de detetar corretamente a carrinha. No que diz respeito à classe carro, houve significativamente uma melhoria na classificação e localização desta classe no teste da rede do terceiro *dataset*. Esta melhoria deve-se ao acréscimo de imagens desta classe com vários ângulos de visualização.

(a) (b)

Figura 6.7 - Resultados obtidos do teste às redes com o segundo (a) e terceiro (b) *datasets* (imagem proveniente do *dataset* [50]).

Por fim, foi introduzida na entrada das redes uma imagem que apresenta 8 carros e 4 pessoas, figura 6.7. Perante os resultados obtidos dos dois testes às redes, pode-se observar que ambas não conseguiram reconhecer as pessoas na imagem. Em relação à classificação e localização da classe

carro, o terceiro *dataset* (b) conseguiu reconhecer mais carros comparando ao segundo *dataset* (a), sendo que a rede treinada com o segundo *dataset* (a) apresenta valores de confiança superiores na deteção das classes carro.

Após terem sido realizados os testes com imagens, foi introduzido na entrada de ambas as redes um pequeno vídeo a visualizar vários veículos estacionados, bem como um pedestre a passear no passeio. Para este vídeo foi utilizada a câmara de um Samsung Galaxy S10+, no qual a filmagem decorreu no segundo andar de um prédio.

Perante as classificações e localizações obtidas pelos testes de ambas as redes, é possível observar que a rede do terceiro *dataset* é mais eficaz e precisa em reconhecer as classes presentes ao longo do vídeo.


(a) (b)

Figura 6.8 -Resultados obtidos do teste com um *frame* do vídeo às redes com o segundo (a) e terceiro (b) *datasets*.

Na figura 6.8 são apresentados os resultados obtidos por ambas as redes num mesmo *frame* do vídeo. Como se pode observar, a rede do segundo *dataset* (a) comparado à rede do terceiro *dataset* (b) não conseguiu reconhecer a pessoa no passeio bem como há uma incorreta deteção de um objeto presente na varanda. Para além disso, ambas as redes reconheceram incorretamente a carrinha.

#### 6.2.1.4 Conclusão dos resultados

Apesar de ambas as redes que foram obtidas pelo treino dos dois *datasets* terem valores muito similares em termos de resultados do treino, é certo que ao longo dos testes realizados apresentaram várias diferenças entre si.

Conclui-se que a rede do terceiro *dataset* demonstrou ser mais precisa e eficaz para a classificação e localização das classes carro, pessoa e ciclista comparado com a rede do segundo *dataset*. Em relação à classe carrinha, devido à falta de imagens desta classe, ambos os *datasets* apresentaram uma grande dificuldade no reconhecimento ao longo dos testes realizados.

### 6.2.2 *Dataset* com 3 classes

Dados os maus resultados no treino e no teste da rede para classificação e localização da classe carrinha, optou-se por retirar esta classe, uma vez que se verificou não ser relevante a diferenciação entre esta e a classe carro para o problema desta dissertação.

Assim todas as carrinhas identificadas nas imagens do terceiro *dataset* foram anotadas para a classe carro resultando num novo *dataset* com 3 classes. Perante este novo *dataset* a classe id para pessoa corresponde ao número 0, para carro o número 1 e para ciclista o número 2.

#### 6.2.2.1 Parâmetros do treino

Para a realização do treino do *dataset* final foram estipulados os valores dos parâmetros de treino representados na tabela abaixo.

Tabela 6.2 - Parâmetros do treino à rede com o *dataset* final.

| Parâmetros | Valor |
|---|---|
| *Batch* | 64 |
| *Subdivisions* | 16 |
| *Max_batches* | 6000 |
| Classes | 3 |
| *Filters* | 24 |

#### 6.2.2.2 Resultado do treino

Uma vez que o *dataset* final é proveniente do terceiro *dataset* com 4 classes, o seu conjunto de dados também contém 8694 imagens, estando dividido em dois *datasets*. Um para o treino com 8429 imagens e outro para a validação com 265 imagens.

```
class_id = 0, name = person, ap = 79.68%         (TP = 171, FP = 42)
class_id = 1, name = car, ap = 98.98%            (TP = 272, FP = 10)
class_id = 2, name = bike, ap = 94.68%           (TP = 43, FP = 3)
```

Com a realização de vários treinos com este *dataset*, são obtidos os resultados anteriores do melhor treino. Verifica-se que para a classe pessoa (*person*) obteve-se uma *precision* de 80,4 %, para a classe carro (*car*) de 96,5 % e para a classe ciclista (*bike*) de 93,5 %. A rede obteve também uma precisão média com limite de 0,5 IoU de 91,11 %.

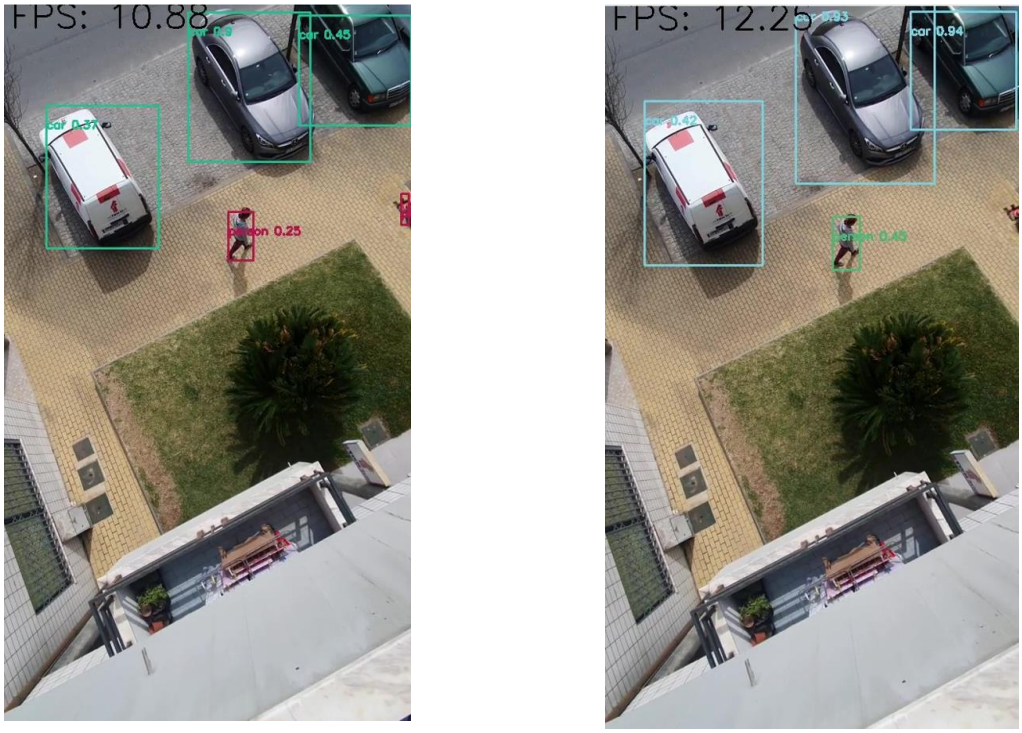
```
mean average precision (mAP@0.50) = 0.911150, or 91.11 %
```

Comparativamente ao resultado obtido com o treino da rede com o terceiro *dataset* com 4 classes, o *dataset* final obteve melhores resultados no treino, destacando que a *precision* da classe ciclista melhorou substancialmente, ao passar de 83,6 % para 93,5 %.

#### 6.2.2.3 Resultados do teste

Após o treino da rede com o *dataset* final, realizaram-se vários testes a esta nova rede. Estes testes consistiram inicialmente numa pequena comparação entre a rede do *dataset* final e a rede do terceiro *dataset*, através do *script Python*, no qual, foi introduzido na entrada de ambas as redes o mesmo vídeo já utilizado anteriormente.

Perante as classificações e localizações obtidas pelos testes de ambas as redes, é possível observar que a rede do *dataset* final apresenta um processamento de imagem mais elevado, bem como uma maior eficácia e precisão em reconhecer as classes presentes ao longo do vídeo. Ressalvando que, para esta rede, todas as carrinhas passaram a ser reconhecidas corretamente como carros.




(a) (b)

Figura 6.9 - Resultados obtidos do teste com um *frame* do vídeo às redes com o terceiro *dataset* (a) e com o *datasets* final (b).

Na figura 6.9 são apresentados os resultados obtidos por ambas as redes num mesmo *frame* do vídeo. Como se pode observar, a rede do terceiro *dataset* (a) comparado à rede do *dataset* final (b) apresenta valores de confiança mais baixos nas classes carro e pessoa bem como uma incorreta deteção presente no passeio e na carrinha. Para além disso, verifica-se que esta rede (b) apresenta menos fps.

De seguida, foi introduzida na entrada para ambas as redes a seguinte imagem, anteriormente utilizada para testar a classificação e localização da classe ciclista (figura 6.10).


(a) (b)

Figura 6.10 - Resultados obtidos do teste às redes com o terceiro *dataset* (a) e com o *dataset* final (b) (imagem proveniente do *dataset* [49]).

Na figura 6.10 pode-se observar que ambas as redes detetaram os 2 ciclistas. Contudo, a rede obtida com o treino com o terceiro *dataset* (a) apresenta resultados significativamente melhores que a rede do *dataset* final (b).

Posteriormente foram realizados testes com a rede obtida do treino com o *dataset* final no ambiente de simulação. Para estes testes realizaram-se a classificação e localização das classes através da introdução dos *frames* na entrada da rede obtidos pela câmara vertical do *drone*. Deste modo, com a introdução de uma pessoa, carro ou ciclista no ambiente de simulação foram realizados testes que consistiram no reconhecimento da classe visualizada pela câmara vertical a uma altura de 6 metros em relação ao chão (figura 6.11).

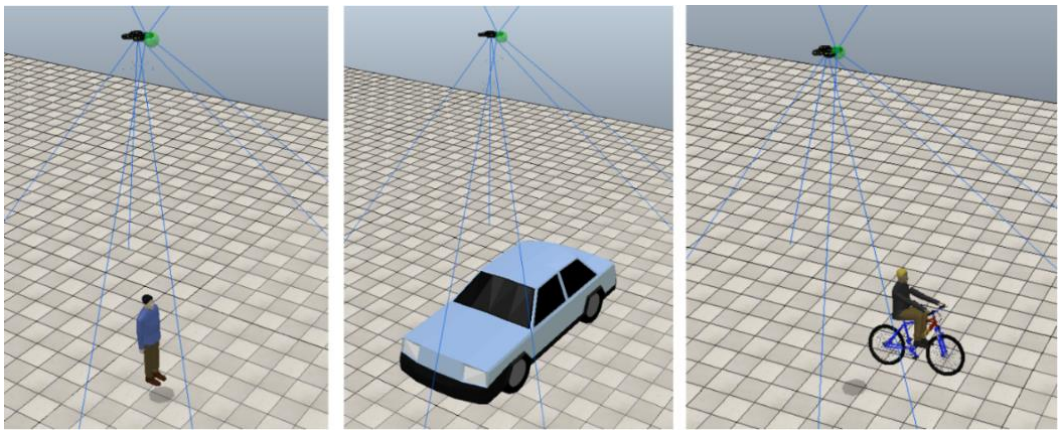


Figura 6.11 - Representações no ambiente de simulação das classes pessoa, carro e ciclista.

Os resultados obtidos pelos testes realizados representam a primeira deteção que a rede faz quando visualiza alguma classe através da câmara do *drone*.

Para o primeiro teste à rede realizou-se a deteção de uma pessoa no ambiente de simulação.

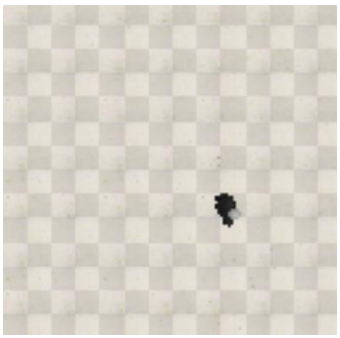


Figura 6.12 - Visualização de uma pessoa pela câmara vertical do *drone* no ambiente de simulação.

Perante a visualização da câmara vertical do *drone*, figura 6.12, a rede classificou e localizou corretamente a pessoa, como se pode observar pela figura 6.13. A localização da *bounding box* obtida representa as coordenadas (x, y) do canto superior esquerdo, largura e altura.

```
Label: person
Valor de confiança: 0.9337605237960815
Coordenadas da bounding box do objeto: [253, 239, 33, 62]
```

Figura 6.13 - Resultado obtido do teste à rede perante a visualização da pessoa.

Para além deste resultado a rede demonstrou ser eficaz para detetar esta classe em vários testes realizados.

Posteriormente, foram realizados vários testes à rede para a deteção de um carro. Ao longo desses testes foi possível observar que a rede perante a visualização do carro é bastante eficaz para a deteção da classe.

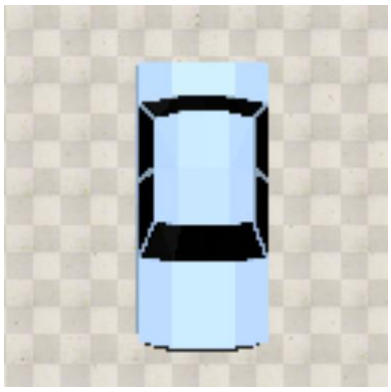


Figura 6.14 - Visualização de um carro pela câmara vertical do *drone* no ambiente de simulação.

Na figura 6.14, encontra-se representada a visualização pela câmara vertical do *drone* onde foram obtidos os resultados da rede (figura 6.15). É verificado que a rede conseguiu reconhecer com sucesso o carro com um valor de confiança de 0,80.

```
Label: car
Valor de confiança: 0.8040021657943726
Coordenadas da bounding box do objeto: [99, 88, 126, 142]
```

Figura 6.15 - Resultado obtido do teste à rede perante a visualização do carro.

Por fim, os últimos testes realizados à rede no ambiente de simulação consistiram na deteção da classe ciclista. Com os primeiros resultados obtidos destes testes, foi possível observar uma grande dificuldade na deteção desta classe, dado muitas vezes ser confundido com a classe pessoa. Deste modo, para os seguintes testes realizados o ciclista foi implementado com uma ligeira rotação no ambiente de simulação. Assim, com essa rotação no ciclista foram obtidos melhores resultados nos testes.

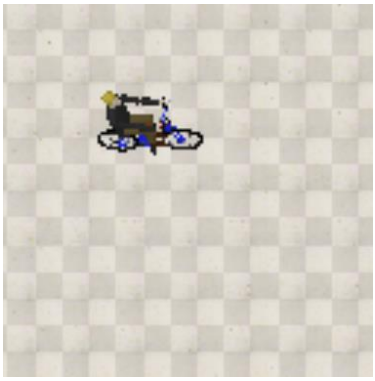


Figura 6.16 - Visualização de um ciclista pela câmara vertical do *drone* no ambiente de simulação.

Perante a visualização da câmara vertical do *drone*, figura 6.16, a rede classificou e localizou corretamente o ciclista com um valor de confiança de 0,84, como podemos observar pela figura 6.17.

```
Label: bike
Valor de confiança: 0.8433020114898682
Coordenadas da bounding box do objeto: [77, 104, 113, 51]
```

Figura 6.17 - Resultado obtido do teste à rede perante a visualização do ciclista.

## 6.3 Aterragem

Neste último subcapítulo serão relatadas todas as etapas que foram realizadas para a aterragem por parte do *drone*. Inicialmente será descrito o planeamento efetuado para a realização de uma aterragem segura e, de seguida, será explicada a janela de visualização utilizada para o cálculo do desvio após a deteção de uma classe por parte da rede. Por fim, é apresentada a estratégia utilizada para a realização do desvio.

### 6.3.1 Planeamento de aterragem

Após serem concluídas todas as etapas anteriormente descritas, foi realizado o planeamento da aterragem para o *drone* (figura 6.18). Nisto, foi estipulado que a condução autónoma realizada pelo *drone* consiste em voar a uma altura de 6 metros em relação ao chão. Assim, como já foi referido anteriormente, o modo de aterragem é iniciado com a chegada do *drone* ao destino final, sendo esta realizada em 2 etapas.

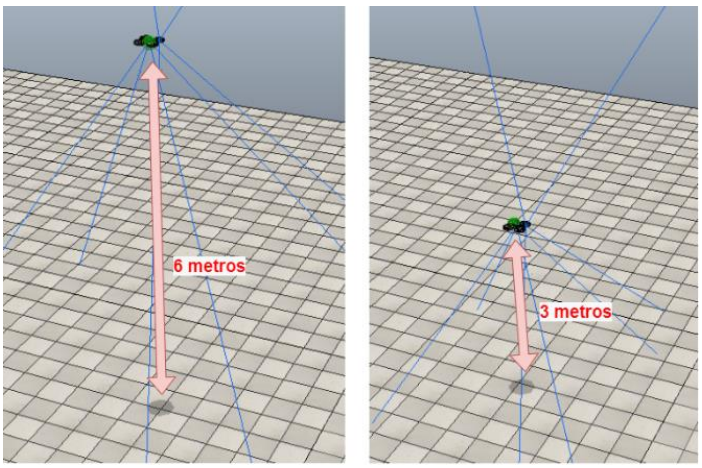



Figura 6.18 - Representações da altura que o *drone* apresenta na primeira (esquerda) e na segunda (direita) etapa.

A primeira etapa consiste em ativar a rede para a deteção das classes pessoa, carro e ciclista a 6 metros de altura através da visualização da câmara vertical do *drone*. Perante a deteção de alguma classe é desativada a rede e feito pelo *drone* um desvio de 2 metros em relação ao *target* final. Assim, consoante a localização obtida pela *bounding box* detetada pela rede, será estipulada através da janela de visualização qual o desvio para a frente ou para trás que será realizado, para que o *drone* se desloque para um novo *target* onde não permaneça nenhum objeto de uma das classes. Desta forma, após o *drone* chegar a este novo *target* é novamente ativada a rede para a deteção na nova área de aterragem.

Esta etapa dá-se por concluída quando não for detetada nenhuma classe durante 1 minuto, sendo a rede desativada para o *drone* realizar a descida até aos 3 metros de altura.

A segunda etapa é realizada quando o *drone* atinge os 3 metros de altura. Esta etapa consiste em realizar todos os passos efetuados na primeira etapa, excetuando no final que caso não seja detetada nenhuma classe por parte da rede é finalizada a aterragem.

Devido ao *drone* estar a uma altura baixa bem como a rede ter sido treinada com um *dataset* com imagens obtidas por voos de *drones* com alturas a variar entre os 5 e os 35 metros, a rede só demonstrou ser eficaz para a deteção da classe pessoa.

Figura 6.19 – Visualização de uma pessoa pela câmara vertical do *drone* no ambiente de simulação.

Como se pode observar pela figura 6.20, perante a visualização da câmara vertical do *drone* a 3 metros de altitude, figura 6.19, a rede classificou e localizou corretamente a pessoa com um valor de confiança de 0,84.

```
Label: person
Valor de confiança: 0.8492438197135925
Coordenadas da bounding box do objeto: [211, 86, 54, 103]
```

Figura 6.20 – Resultado obtido do teste à rede perante a visualização da pessoa.

### 6.3.2 Janela de visualização

Durante a realização de uma aterragem, são inúmeras as possibilidades de localização que um objeto de uma das classes pode apresentar na área de aterragem. Perante a sua deteção por parte da rede torna-se essencial aferir qual o desvio para a frente ou para trás o *drone* deve realizar de forma a chegar a uma área de aterragem sem nenhuma classe presente. Assim, é implementada uma janela de visualização. Esta janela nada mais é que a visualização da câmara vertical do *drone* e na qual apresenta dois limites verticais, localizados a ½ e ¾, como se pode observar pela figura 6.21.

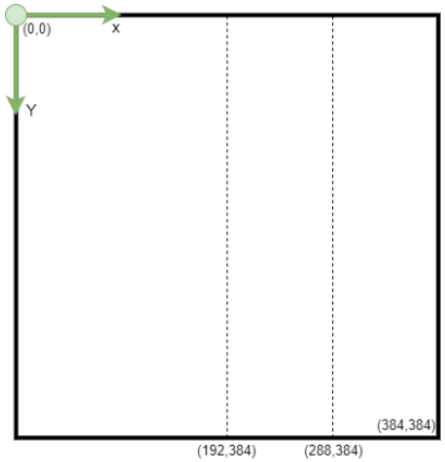



Figura 6.21 - Janela de visualização com os dois limites representados.

Assim, consoante a localização que a *bounding box* apresenta em relação aos limites verticais presentes na janela de visualização é possível aferir qual desvio o *drone* deve realizar. Na figura 6.22 é representado dentro da janela de visualização uma *bounding box* a vermelho bem como as coordenadas dos seus dois vértices superiores $(x_1, y_1)$ e $(x_2, y_2)$. Caso o $x_1$ for superior a 192 o *drone* irá deslocar-se para trás. Da mesma forma, o desvio irá realizar-se caso a *bounding box* apresentar um $x_1$ inferior ou igual a 192 e um $x_2$ superior a 288. Por outro lado, se esta caixa tiver um $x_1$ inferior ou igual a 192 e um $x_2$ inferior ou igual a 288 o desvio realiza-se para a frente.

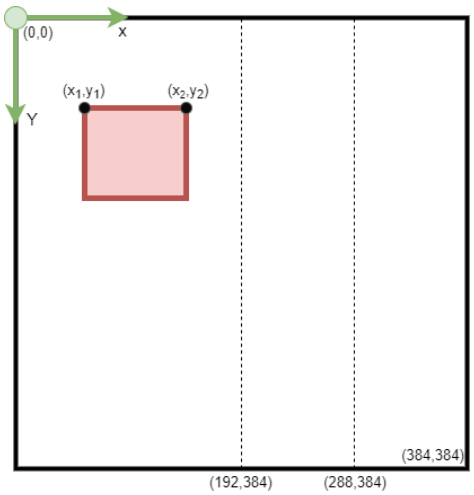



Figura 6.22 - Representação de uma *bounding box* a vermelho com os seus dois vértices $(x_1, y_1)$ e $(x_2, y_2)$ dentro da janela de visualização.

Contudo verifica-se que esta implementação da janela de visualização consiste apenas nos casos em que uma rede só deteta um objeto de uma das classes na área de aterragem. Para os casos em que uma rede obtém mais que uma deteção, é utilizada a *bounding box* da deteção com maior valor de confiança para aferir qual desvio o *drone* irá realizar. Para isso, realizou-se uma pequena simulação onde no local de aterragem do *drone* estão um carro e duas pessoas. Obteve-se a seguinte janela de visualização (figura 6.23).

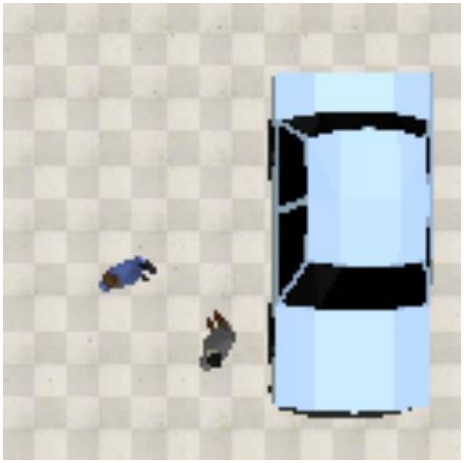


Figura 6.23 - Visualização da câmara vertical do *drone* da simulação.

Como se verifica pela figura 6.24 a rede detetou corretamente o carro e as duas pessoas presentes na área de aterragem, sendo que a pessoa de camisola cinzenta apresentou o maior valor de confiança entre as três deteções obtidas por parte da rede. Uma vez que a *bounding box* da deteção desta pessoa apresenta um $x_1$ com um valor de 165 e um $x_2$ com um valor de 199, o *drone* deslocar-se-á para a frente.

```
Label: car
Valor de confiança: 0.45834246277809143
Coordenadas da bounding box do objeto: [242, 92, 113, 136]
Numero de objetos: 3

Label: person
Valor de confiança: 0.2585877478122711
Coordenadas da bounding box do objeto: [89, 213, 42, 33]
Numero de objetos: 3

Label: person
Valor de confiança: 0.9204564690589905
Coordenadas da bounding box do objeto: [165, 264, 34, 45]
Numero de objetos: 3

Cálculo do desvio perante a deteção person com o valor de confiança de 0.9204564690589905
Deslocação do drone para a frente.
```

Figura 6.24 - Resultados obtidos do teste à rede da simulação.

### 6.3.3 Estratégia de desvio

Como já foi referido anteriormente, o desvio do *drone* consiste no cálculo de um novo *target* a 2 metros de distância em relação ao *target* final. Assim, na figura 6.25 é apresentado o cálculo realizado para a estratégia do desvio para a frente, onde o $O_e$ representa a localização inicial do *drone* antes de realizar o voo autónomo.

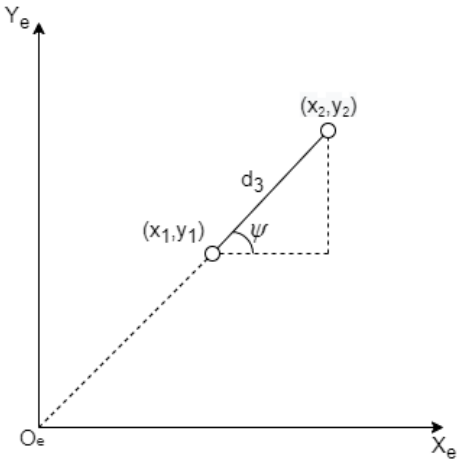



Figura 6.25 - Cálculo da estratégia de desvio.

Como se pode observar pela figura 6.25, o $(x_1, y_1)$ e o $(x_2, y_2)$ representam o *target* final e o novo *target*, respetivamente. Para além disso, o $d_3$ representa a distância do desvio que o *drone* tem que deslocar e o $\psi$ consiste no valor *yaw* que o *drone* apresenta. Assim, através da seguinte equação são obtidas as novas coordenadas do novo *target*.

$$\begin{cases} x_2 = x_1 + \left((\cos\psi) \times d_3\right) \\ y_2 = y_1 + \left((\operatorname{sen}\psi) \times d_3\right) \end{cases} \quad (50)$$

Em relação ao cálculo para se obter as coordenadas para o novo *target* quando é realizado o desvio para trás consiste em utilizar a mesma equação 50 com a alteração do valor $d_3$ para um valor negativo.

A equação explicada anteriormente corresponde ao cálculo para esta estratégia de desvio para o primeiro quadrante e no qual foram realizados os cálculos para os restantes quadrantes e eixos coordenados. Para cada um dos novos cálculos foi implementada a equação 50 com ligeiras alterações na soma, subtração e nos ângulos.

# Capítulo 7
# Conclusões

Neste documento de dissertação de mestrado é descrito e explicado o desenvolvimento da realização de uma condução autónoma por um *quadcopter*, sendo divida em três partes. Na primeira parte, são apresentadas as técnicas de controlo de posição e de orientação utilizadas e no qual são demonstrados todos os parâmetros e resultados simulados dos ajustes PIDs. Perante estes ajustes conseguiu-se controlar a orientação e a altura do *drone*. Contudo, verifica-se no decorrer de algumas simulações que o ambiente de simulação torna-se por vezes lento de tal forma que o *drone* ultrapassa ligeiramente os *targets* definidos antes de se estabelecer neles. De seguida, são apresentados o planeamento de trajetória e a conversão das coordenadas GPS para coordenadas cartesianas. Ambas as estratégias ao serem simuladas apresentaram erros de cálculos bastantes reduzidos sendo confiáveis para a sua utilização. Para além disso, com a implementação do planeamento de trajetória verificou-se um correto controlo na velocidade máxima que o *drone* atinge perante qualquer deslocação que percorra, sendo bastante importante este controlo para a realização das estratégias dos desvios de obstáculos.

Na segunda parte, é apresentada inicialmente a equação desenvolvida para obter a distância entre o *drone* e o objeto alvo. Foi possível observar que esta equação consegue calcular corretamente distâncias até os 4 metros.

De seguida, é apresentado e explicado o desvio na vertical, onde ao longo de vários testes a janela de visualização demonstrou ser eficaz para o cálculo da distância que o *drone* tem que subir para ultrapassar o obstáculo. Contudo, esta janela de visualização apresenta algumas limitações no que diz respeito à forma como a caixa é visualizada pelo *drone* uma vez que este método só funciona bem perante a visualização de uma face de uma caixa. Isto deve-se ao facto da equação implementada para o cálculo da distância entre o *drone* e a caixa ser referente à área visualizada da face do obstáculo alvo.

Da mesma forma, apesar da janela de visualização ser eficaz para o cálculo da distância para o desvio para a direita ou para a esquerda, são verificadas as mesmas limitações para o desvio horizontal. Para além disso, foram simulados com sucesso os desvios na vertical e na horizontal. Porém, perante as simulações realizadas foi possível observar que em relação ao desvio horizontal

algumas destas simulações o *drone* necessitou ligeiramente de mais de um metro para efetuar a travagem.

Por fim, a terceira parte tem como foco a realização da aterragem do *drone*. Assim, primeiramente é explicada como o *dataset* foi realizado e desenvolvido. Em seguida, é explicado em detalhe como foi realizado o treino da arquitetura *YOLOv3-Tiny*, incluindo as linhas de comando utilizadas. Para além disso, foram apresentados os três métodos utilizados para testar as redes obtidas. Deste modo, foram apresentados os resultados obtidos dos testes e treinos das redes com os *datasets* com 4 classes. Perante estes resultados foi possível observar que o *dataset* que contém na sua base de dados imagens com vários ângulos de visualização demonstrou melhores resultados para a classificação e localização para o reconhecimento das classes definidas. Contudo ambas as redes obtidas pelo treino com os *datasets* com 4 classes obtiveram maus resultados para a deteção da classe carrinha devido à falta de imagens desta classe nos *datasets*. Desta forma, foi criado um último *dataset* sem esta classe proveniente do *dataset* com 4 classes que obteve os melhores resultados nos testes anteriormente realizados.

Após o treino à rede com o *dataset* com 3 classes foram realizados testes a esta rede e no qual foram obtidos melhores resultados de deteção e localização nas classes pessoa e carro, bem como mais fps comparativamente aos restantes testes realizados às redes obtidas com os *datasets* com 4 classes. Contudo, a classe ciclista é a classe na qual a rede apresentou mais dificuldades em classificar e localizar. A seguir, com os testes realizados no ambiente de simulação verificou-se que a rede classificava e localizava corretamente todas as classes definidas a uma altura de 6 metros. No entanto, foi necessário implementar uma ligeira rotação ao ciclista para que a rede conseguisse detetar corretamente esta classe. Ao ser obtida a rede com o *dataset* com 3 classes foi apresentada e explicada a estratégia para efetuar uma aterragem segura. Esta estratégia perante a deteção do objeto alvo com maior valor de confiança o *drone* irá deslocar-se para a frente ou para trás. Deste modo, foram realizadas com sucesso várias aterragens em segurança no ambiente de simulação ao efetuar esta estratégia de desvio. Porém, verificou-se que o *drone* em algumas simulações deslocava-se repetidamente para a frente e para trás até conseguir deslocar-se para uma área sem nenhum objeto.

## 7.1 Trabalhos futuros

Perante os resultados apresentados, para trabalhos futuros para este projeto propõem-se o aumento de precisão na classificação e localização dos objetos presentes na área de aterragem. Para tal, esse aumento de precisão pode ser feito através da introdução de mais imagens a diferentes

ângulos das classes abordadas para o *dataset* e adicionar novas classes, como por exemplo árvores e arbustos, de forma a tornar a rede cada vez mais completa. Para além disso, com o *dataset* criado pode se realizar uma análise para outras redes, como por exemplo a rede *slimYOLOv3*.

Em relação ao modo de aterragem, propõem-se a implementação na estratégia de desvio a movimentação para a esquerda e para a direita de forma a tornar mais eficiente a movimentação do *drone* para uma área de aterragem desocupada.

Por fim, no que diz respeito à deteção e desvio de obstáculos propõem-se a implementação de uma rede neuronal para a classificação e localização dos obstáculos bem como a introdução de sensores de distância no *quadcopter* de forma a obter distâncias mais precisas em relação aos obstáculos.

# Referências


[1] Beirão, B. M. B. (2019). O panorama dos *drones* em Portugal. Dissertação de mestrado, Técnico de Lisboa, Lisboa, Portugal.

[2] "DHL *Express* inicia serviço de entrega com *drones*." [*Online*]. *Available*: https://pplware.sapo.pt/gadgets/dhl-express-inicia-servico-de-entrega-com-*drones*/. [*Accessed*: 08-Jan-2021]

[3] "Amazon lança *drone* inovador para entrega de encomendas." [*Online*]. *Available*: https://www.jornaldenegocios.pt/empresas/tecnologias/detalhe/amazon-lanca-drone-inovador-para-entrega-de-encomendas. [*Accessed*: 08-Jan-2021]

[4] "AUTOMAÇÃO E ENTREGAS POR *DRONES*." [*Online*]. *Available*: https://www.connect-robotics.com/pt. [*Accessed*: 05-Oct-2022]

[5] "A drone program taking flight." [*Online*]. Available: https://www.aboutamazon.com/news/transportation/a-drone-program-taking-flight. [*Accessed*: 05-Oct-2022]

[6] "*DHL EXPRESS LAUNCHES ITS FIRST REGULAR FULLY-AUTOMATED AND INTELLIGENT URBAN DRONE DELIVERY SERVICE*." [*Online*]. *Available*: https://www.dhl.com/global-en/home/press/press-archive/2019/dhl-express-launches-its-first-regular-fully-automated-and-intelligent-urban-drone-delivery-service.html. [*Accessed*: 05-Jun-2022]

[7] "*Learn about how Wing delivery works*." [*Online*]. *Available*: https://wing.com/about-delivery/. [*Accessed*: 05-Nov-2022]

[8] Mardan, M. (2016). *Attitude and position control of quadrotors: design, implementation and experimental evaluation.* Dissertação de mestrado, *University of Manitoba*, Winnipeg, Manitoba, Canada. doi: 10.1177/1729881417709242.

[9] Martins, R. M. F. (2019). Controlo de *quadcopter*. Dissertação de mestrado, Universidade da Madeira, Madeira, Portugal [*Online*]. *Available*: http://hdl.handle.net/10400.13/2509

[10] Zimmerman, N. M. (2016). *Flight control and hardware design of multi-rotor systems*. Dissertação de mestrado, Marquette University, Milwaukee, Estados Unidos.

[11] Hackeling, G. (2014). *Mastering Machine Learning with scikit-learn*. Birmingham, UK: Packt Publishing Ltd. ISBN: 978-1-78398-836-5.

[12] Pinto, T. A. B. (2018). *Object detection with artificial vision and neural networks for service robots*. Dissertação de mestrado, Universidade do Minho, Guimarães, Portugal.

[13] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019, ISBN: 978-1-492-03264-9.

[14] "*Supervised vs. Unsupervised Learning*." [*Online*]. *Available*: https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d. [*Accessed*: 28-Nov-2020]

[15] "*Classification Versus Regression — Intro To Machine Learning* #5." [*Online*]. *Available*: https://medium.com/simple-ai/classification-versus-regression-intro-to-machine-learning-5-5566efd4cb83. [*Accessed*: 30-Nov-2020]

[16] "Como as redes neurais aprendem?" [*Online*]. *Available*: https://docs.microsoft.com/pt-br/archive/msdn-magazine/2019/april/artificially-intelligent-how-do-neural-networks-learn. [*Accessed*: 12-Nov-2020]

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "*Gradient-based learning applied to document recognition*," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[18] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "*Understanding of a convolutional neural network*," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6, doi: 10.1109/icengtechnol.2017.8308186 .

[19] "*An introduction to Convolutional Neural Networks.*" [*Online*]. *Available*: https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7. [*Accessed*: 19-Dec-2020]

[20] S. Sakib, N. Ahmed, A. J. Kabir, and H. Ahmed, "*An Overview of Convolutional Neural Network: Its Architecture and Applications*," 2019, doi: 10.20944/preprints201811.0546.v1.

[21] "*Intersection over Union (IoU) for object detection.*" [*Online*]. *Available*: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. [*Accessed*: 01-Apr-2021]

[22] "*Object detection with neural networks — a simple tutorial using keras*." [Online]. *Available*: https://towardsdatascience.com/object-detection-with-neural-networks-a4e2c46b4491. [*Accessed*: 01-Apr-2021]

[23] "mAP (*mean Average Precision) for Object Detection.*" [*Online*]. *Available*: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173. [*Accessed*: 01-Apr-2021]

[24] "*Evaluation of ranked retrieval results.*" [*Online*]. *Available*: https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-ranked-retrieval-results-1.html. [*Accessed*: 01-Apr-2021]

[25] "LeNet-5 – *A Classic CNN Architecture*." [*Online*]. *Available*: https://engmrk.com/lenet-5-a-classic-cnn-architecture/. [*Accessed*: 01-Mar-2021]

[26] "AlexNet: *The Architecture that Challenged CNNs*." [*Online*]. *Available*: https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951.

[*Accessed*: 01-Mar-2021]

[27] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *and others*, "*Imagenet large scale visual recognition challenge*," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015, doi: 10.1007/s11263-015-0816-y.

[28] "Imagenet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012)." [*Online*]. *Available*: http://www.image-net.org/challenges/LSVRC/2012/results.html. [*Accessed*: 01-Feb-2021]

[29] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "*Overfeat: Integrated recognition, localization and detection using convolutional networks*," *arXiv preprint arXiv:1312.6229*, 2013.

[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "*Going deeper with convolutions*," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[31] K. Simonyan and A. Zisserman, "*Very deep convolutional networks for large-scale image recognition*," *arXiv preprint arXiv:1409.1556*, 2014.

[32] "*Imagenet Large Scale Visual Recognition Challenge 2014* (ILSVRC2014)." [*Online*]. Available: http://www.image-net.org/challenges/LSVRC/2014/results.php. [*Accessed*: 01-Feb-2021]

[33] K. He, X. Zhang, S. Ren, and J. Sun, "*Deep residual learning for image recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[34] "*Imagenet Large Scale Visual Recognition Challenge 2015* (ILSVRC2015)." [*Online*]. *Available*: http://www.image-net.org/challenges/LSVRC/2015/results.php. [*Accessed*: 01-Feb-2021]

[35] C. B. Murthy, M. F. Hashmi, N. D. Bokde, and Z. W. Geem, "*Investigations of Object Detection in Images/Videos Using Various Deep Learning Techniques and Embedded Platforms—A Comprehensive Review*," *Applied Sciences*, vol. 10, no. 9, p. 3280, 2020, doi: 10.3390/app10093280.

[36] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "*You only look once: Unified, real-time object detection," in Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[37] J. Redmon and A. Farhadi, "*YOLO9000: better, faster, stronger*," *in Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[38] J. Redmon and A. Farhadi, "*Yolov3: An incremental improvement*," *arXiv preprint arXiv:1804.02767*, 2018.

[39] P. Adarsh, P. Rathi, and M. Kumar, "*YOLO v3-Tiny: Object Detection and Recognition using one stage improved model," in 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2020, pp. 687–694, doi:

10.1109/ICACCS48705.2020.9074315.

[40] W. Fang, L. Wang, and P. Ren, "*Tinier-YOLO: A real-time object detection method for constrained environments," IEEE Access*, vol. 8, pp. 1935–1944, 2019, doi: 10.1109/ACCESS.2019.2961959.

[41] S. Diaz and B. Garate, "Planificación e instrumentación de vuelo de una flotilla de drones," 2018.

[42] "Parrot Bebop 2." [*Online*]. *Available*: https://comparardrones.com.br/drone/parrot-bebop-2/?fbclid=IwAR3Ljhzx9XdLhh3lCmd-7MuGv6rmHvFbiz0XWgXHaXIzG6pXp2hOV1aDXIs. [*Accessed*: 21-Apr-2022]

[43] Teixeira, F. M. (2020). Simulação de um Sistema Robótico de Co-transporte. Dissertação de mestrado, Instituto Superior de Engenharia do Porto, Porto, Portugal.

[44] "*Google Colab*." [*Online*]. Available: https://research.google.com/colaboratory/intl/pt-PT/faq.html. [Accessed: 05-Mar-2022]

[45] "Semantic Drone Dataset." [Online]. Available: http://dronedataset.icg.tugraz.at/. [Accessed: 08–03-2022]

[46] G. J. Fouché and R. Malekian, "*Drone as an autonomous aerial sensor system for motion planning*," *Measurement*, vol. 119, pp. 142–155, 2018.

[47] D. Wang, W. Li, X. Liu, N. Li, and C. Zhang, "UAV *environmental perception and autonomous obstacle avoidance: A deep learning and depth camera combined solution," Computers and Electronics in Agriculture*, vol. 175, p. 105523, 2020.

[48] I. Bozcan and E. Kayacan, "Au-air: *A multi-modal unmanned aerial vehicle dataset for low altitude traffic surveillance*," *in 2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 8504–8510, doi: 10.1109/ICRA40945.2020.9196845.

[49] "*A Benchmark and Simulator for* UAV *Tracking*." [*Online*]. *Available*: https://cemse.kaust.edu.sa/ivul/uav123. [*Accessed*: 03-Aug-2022]

[50] P. Zhu, L. Wen, D. Du, X. Bian, H. Fan, Q. Hu, and H. Ling, "*Detection and Tracking Meet Drones Challenge*," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021, doi: 10.1109/TPAMI.2021.3119563.