ESCOLA NAVAL

talant de bien faire

Rodolfo Miguel Pandaio dos Santos Carapau

Sistema de Controlo Externo para um Piloto Automático

Implementação de um driver ROS para um Piloto Automático, no âmbito da Interoperabilidade de Sistemas

Dissertação para obtenção do grau de Mestre em Ciências Militares Navais, na especialidade de Engenharia Naval, Ramo de Armas e Eletrónica

Alfeite

2017

ESCOLA NAVAL

talant de bien faire

Rodolfo Miguel Pandaio dos Santos Carapau

Sistema de Controlo Externo para um Piloto Automático

Implementação de um driver ROS para um Piloto Automático, no âmbito da Interoperabilidade de Sistemas

Dissertação para obtenção do grau de Mestre em Ciências Militares Navais, na especialidade de Engenharia Naval, Ramo de Armas e Eletrónica

Orientação de: CTEN EN-AEL Mário Rui Monteiro Marques

O Aluno Mestrando

_____

ASPOF EN-AEL Santos Carapau

O Orientador

_____

CTEN EN-AEL Monteiro Marques

Alfeite
2017

Epígrafe

"Uma viagem de mil milhas começa com um único passo"

(**Lao Zi**)

Dedicatória

Este trabalho é dedicado à minha família, namorada, amigos e camaradas pelo seu consistente contributo, que se demonstrou fundamental para a realização desta dissertação.

Agradecimentos

Agradeço ao meu orientador CTEN EN-AEL Mário Rui Monteiro Marques pela disponibilidade, ajuda e acompanhamento prestado, e pelas oportunidades que criou durante a realização da dissertação.

Ao camarada e amigo Alexandre Valério Rodrigues, pela cooperação e camaradagem prestada na realização da dissertação.

Ao Professor Doutor Victor Lobo que, como diretor do CINAV, proporcionou a divulgação e enriquecimento desta dissertação.

Á empresa UAVision, pela ajuda disponibilizada a nível de material e conhecimento.

Ao 1TEN EN-AEL Nuno Alexandre Antunes Martins Pessanha Santos, pela partilha de conhecimentos e pela sua disponibilidade.

À minha família e amigos, pela sua compreensão e apoio prestado.

Por fim, à Inês pela sua compreensão, apoio e constante transmissão de confiança.

Resumo

A evolução de veículos não tripulados disponibilizou uma oportunidade valiosa para o crescimento da capacidade operacional militar em ambientes marítimos, através do desempenho de missões como: reconhecimento marítimo, vigilância, patrulhamento, e busca e salvamento. De modo a desenvolver este potencial, a implementação de uma estrutura interoperável que contêm veículos não tripulados que cooperam e interagem entre si com um objetivo comum, é essencial.

Esta dissertação foca-se na apresentação dos benefícios provenientes do uso de veículos não tripulados, das vantagens associadas ao desenvolvimento de interoperabilidade entre veículos não tripulados, e de uma implementação prática que consiste num controlador externo para um veículo aéreo não tripulado que permite a adição de novas funcionalidades para o veículo aéreo não tripulado, como a aterragem visual autónoma em navios de guerra. O desenvolvimento do controlador explora o potencial do *Robotic Operating System*, que providencia uma variedade de bibliotecas, ferramentas e funcionalidades, assim como a capacidade de abstração de *hardware*, promovendo e auxiliando o desenvolvimento de novas tecnologias. A validação do controlador foi baseada na construção de cenários de teste que permitiram a análise dos dois principais módulos do controlador: módulo de comando e controlo, e módulo de aterragem visual autónoma. Os resultados obtidos foram positivos, tendo demonstrado um desempenho promissor, no que toca à passagem de comandos de voo através do controlador, verificando uma baixa percentagem de erros na deteção da marca visual de aterragem, o que permite a aterragem visual autónoma.

O trabalho realizado apresenta-se como um passo em frente no que toca ao desenvolvimento de veículos não tripulados associados à Marinha Portuguesa, tornando-se numa referência para desenvolvimento futuro.

Palavras-chave: Interoperabilidade, Veículos Aéreos Não Tripulados, Aterragem Visual Autónoma

Abstract

The current evolution of unmanned vehicles has unlocked a valuable opportunity to extend military operational capability in maritime environments, through the performance of missions such as: maritime reconnaissance, surveillance, patrolling, and search and rescue. To further develop this potential, it's essential to implement an interoperable structure that integrates several unmanned vehicles, which cooperate and interact together to achieve a common goal.

This thesis aims for presenting the benefits of the use of unmanned vehicles, the benefits of the developing interoperability between unmanned vehicles, and at presenting a practical implementation of an on-board unmanned aerial vehicle controller that allows the addition of several new functionalities to the unmanned aerial vehicle, like performing visual autonomous landing on warships. The development of the controller explores the potential of the Robotic Operating System, which provides a variety of libraries, tools and functionalities, as well as the capability of hardware abstraction, while promoting and supporting the development of new technologies. The validation of the controller was based on test scenarios designed for the analysis of the two main modules of the controller: command & control module and the autonomous visual landing module. The results were positive, showing a promising performance when passing flight commands through the controller, and a low error percentage during the visual marker detection, which allows autonomous visual landing.

This thesis will provide a step further on the development of unmanned vehicles associated to the Portuguese Navy, becoming a standard for future development.

Keywords: *Interoperability, Unmanned Aerial Vehicles, Autonomous Visual Landing*

Índice

Índice de Equações

Índice de Figuras

Índice de Tabelas

Lista de Abreviaturas, Siglas e Acrónimos

2D – Bi-dimensional

3D – Tri-dimensional

BCS - *Body Coordinate System*

DARIUS - *Deployable Search and Rescue Integrated Chain with Unmanned Systems*

EPnP - *Efficient Perspective-n-Point*

GPS – *Global Positioning System*

GCS – *Ground Control Station*

GSM - *Global System for Mobile Communications*

ICARUS - *Integrated Components for Assisted Rescue and Unmanned Search Operations*

IEEE – *Institute of Electrical and Electronics Engineers*

JAUS - *Joint Architecture for Unmanned Systems*

LIDAR – *Light Detection and Ranging*

MAVLink – *Micro Aerial Vehicle Communication Protocol*

MVANT - Micro-veículo Aéreo Não Tripulado

NATO – *North Antlantic Treaty Organization*

PnP - *Perspective-n-Point*

RADAR – *Radio Detection and Ranging*

RGB – *Red, Green, Blue*

ROS – *Robotic Operating System*

TCP - *Transmission Control Protocol*

SITL – *Software-In-The-Loop*

STANAG - *Standardization Agreement*

UDP - *User Datagram Protocol*

VANT – Veículo Aéreo Não Tripulado

VNT – Veículo Não Tripulado

VSNT - Veículos Subsuperfície Não Tripulados

VSUNT - Veículo Superfície Não Tripulado

VTNT - Veículos Terrestres Não Tripulados

XML - *eXtensible Markup Language*

Introdução

"…E ao imenso e possível oceano

Ensinam estas Quinas, que aqui vês,

Que o mar com fim será grego ou romano:

O mar sem fim é português…"

(Pessoa, 1934)

A importância do mar e de ambientes marítimos é demonstrada através do valor que este representa para a economia mundial. Valor que reside em rotas comerciais de tráfego marítimo, em transporte de pessoas e mercadorias, pescas e plataformas petrolíferas *off-shore*, que se traduzem em grandes fontes de rendimentos. No que toca a Portugal, a exploração do potencial apresentado pelo mar é de extrema relevância, uma vez que possuí a terceira maior Zona Económica Exclusiva da União Europeia, e décima primeira maior área mundial de águas jurisdicionais do mundo (incluindo mar territorial e Zona Económica Exclusiva) (Marinha Portuguesa, 2010). Considerando a proposta de extensão da plataforma continental Portuguesa, o território marítimo Português tomará valores na ordem dos quatro milhões de quilómetros quadrados (concretamente, 3,877,408 km^2), o que permitirá constatar que "97% de Portugal é Mar" (EMEPC, 2009) (EMEPC, 2017). Como tal, existe uma necessidade permanente de preservar e proteger o valor associado a este território, bem como todo o tipo de atividades inerentes à atividade marítima, o que motiva o investimento tecnológico aplicado a ambientes marítimos.

O desenvolvimento de Veículos Não Tripulados (VNTs), em particular, de **Veículos Aéreos Não Tripulados (VANTs)**, tem vindo a promover o seu potencial no que toca ao desempenho de missões/tarefas de reconhecimento, vigilância e patrulhamento marítimo, busca e salvamento, e segurança marítima em geral. O empenho de VANTs para a execução destas missões/tarefas, resulta na extensão da capacidade operacional de um navio, apresentando vantagens no que toca a custos financeiros e humanos

reduzidos, quando comparado com veículos aéreos tripulados. O valor associado à utilização de VANTs pode ser promovido através do empenho simultâneo de múltiplos VANTs, ou outro tipo de VNTs, o que promove o estudo de **interoperabilidade** – um conceito chave que define a qualidade da operação conjunta entre sistemas distintos.

No contexto da *North Atlantic Treaty Organization* (NATO), interoperabilidade consiste na capacidade que múltiplas nações possuem para desempenhar treinos, exercícios, tarefas e missões em conjunto, de forma eficaz (NATO, 2006). O *Institute of Electrical and Electronics Engineers* (IEEE) (IEEE, 2016), define interoperabilidade como a capacidade de um sistema trabalhar com outros sistemas sem grandes limitações ou esforço adicional por parte do utilizador, tendo em conta que apenas se torna possível através da utilização **de normas de referência (*standards*[1])**.

A utilização de normas de referência permite a criação de um modelo para a implementação de uma tecnologia, numa dada área, ao estabelecer princípios comuns entre duas entidades distintas, garantido um ponto de homogeneidade entre entidades heterogéneas (Pagano, Candela, e Castelli, 2013). Desta forma, o desenvolvimento e validação de interoperabilidade entre VNTs implica a implementação de uma norma de referência.

No contexto desta dissertação, a promoção de interoperabilidade de sistemas, encontra-se diretamente associada à implementação de um controlador externo para um VANT, desenvolvido através de uma norma de referência (*Robotic Operating System*), que garanta o comando e controlo do veículo e que permita a adição de novas funcionalidades que possam promover interoperabilidade, expandido a capacidade do piloto automático pré-existente. A utilização deste tipo de modelos de referência permite uma normalização para a implementação de sistemas de controlo externos. Ao

[1] *Standard* - Documento que define as características de um produto, processo ou serviço. Características tais como dimensões, aspetos de segurança e requisitos de desempenho. IEEE. (2016). IEEE - Institute of Electrical and Electronics Engineers Organization. Consultado em 12 de Junho de 2016 no IEEE Institute of Electrical and Electronics Engineers
Web Site: https://www.ieee.org/education_careers/education/standards/standards_glossary.html

associar a normalização de uma estrutura de referência à modelação de um sistema sem recorrer ao *software*[2] de origem, obtém-se um sistema apto para ser programado por qualquer utilizador familiarizado com o *Robotic Operating System* (ROS) (e.g. Romero, 2014) (e.g. Quigley, 2015), sem comprometer o código da estrutura original. Para validar o controlador externo, para além da implementação da capacidade de passagem de comandos de voo para o piloto automático do VANT, foi adicionada uma funcionalidade de aterragem visual autónoma direcionada para o convés de voo de um navio de guerra.

O produto final consiste na criação de um sistema de controlo externo para um piloto automático de um VANT, validado através da adição de novas funcionalidades, promovendo o desenvolvimento futuro de VANTs, e de outros VNTs, que integrem um sistema de controlo de referência comum que contribuirá para a interoperabilidade de sistemas, no âmbito da Marinha Portuguesa.

Motivação

A escolha do tema da dissertação foi essencialmente motivada pela evolução corrente de VANTs, pelos benefícios associados à utilização de VANTs e à interoperabilidade de sistemas. Em particular, pelo baixo custo e empenho do interveniente humano, pela capacidade de desempenharem diversas tarefas do âmbito civil e militar e pela simplicidade de empenhamento destes meios.

Estes benefícios têm vindo a ser demonstrados, no âmbito da Marinha, através de projetos como o SEAGULL e o SUNNY, que se baseiam no desenvolvimento de VANTs que contribuem, respetivamente, para: perceção situacional de tarefas como deteção de derrames de petróleo e substâncias perigosas, sistemas de seguimento marítimo, reconhecimento de padrões de comportamento e monitorização de parâmetros ambientais; desenvolvimento e implementação de soluções para o patrulhamento

[2] *Software* – Designação de um conjunto virtual de instruções que definem programas, aplicações e que possibilitam a interação com o computador, por parte do utilizador. Tech Terms (2017). *Software*. Consultado em 02 de Janeiro de 2017 no Tech Terms web site.
Web Site: https://techterms.com/definition/software

eficaz de fronteiras (terrestres e marítimas) e para a deteção de tráfego ilegal de pessoas entre fronteiras, através de múltiplos VANTs (Marques, 2016a) (SUNNY, 2014).

No âmbito da interoperabilidade entre VNTs, os projetos de referência apresentados nesta dissertação demonstram os benefícios do desenvolvimento de interoperabilidade, motivando o estudo da temática. Consistem nos projetos *Integrated Components for Assisted Rescue and Unmanned Search Operations* (ICARUS) e *Deployable Search And Rescue Integrated Chain with Unmanned Systems* (DARIUS) (subcapítulo 1.1.3.), que apresentam uma estrutura interoperável composta por diversos VNTs, com o principal objetivo de realizar operações de busca e salvamento (Marques, 2016b) (Chrobocinski, 2012).

Quanto ao sistema de controlo externo desenvolvido, a funcionalidade de aterragem visual autónoma em navios foi escolhida com base no conceito do desenvolvimento de um sistema que garantisse uma redundância para aterragem de um VANT, num ambiente sujeito a empastelamento de sinal GPS (*Global Positioning System*) ou de inibição de outros tipos de sistemas de aterragem, garantindo assim um sistema de suporte de emergência para a aterragem de um VANT.

Com o desenvolvimento do sistema de controlo externo é dado um passo inicial para a integração de novos meios não tripulados na Marinha Portuguesa, desenvolvidos de acordo com a norma de referência validada pelo sistema de controlo externo, com o objetivo de simplificar e promover a adição de novas funcionalidades que promovam interoperabilidade.

Objetivos

O tema em estudo tem como principal objetivo o **desenvolvimento de interoperabilidade entre VNTs através da criação de um controlador externo que amplifica e expande as capacidades de um VANT, com recurso a uma norma de referência (ROS)**.

Para além do objetivo principal, esta dissertação visa o cumprimento dos seguintes objetivos:

- Apresentar a importância de VNTs para aplicações militares;
- Promover o estudo de interoperabilidade de sistemas;
- Promover o estudo de sistemas de aterragem visual autónoma de VANTs;
- Relacionar o conceito de interoperabilidade com VNTs, destacando os seus benefícios;
- Promover a utilização do ROS como uma norma de referência para o desenvolvimento de novas funcionalidades e de interoperabilidade;
- Adição de funcionalidades ao controlador externo de modo a validar a sua implementação.

Metodologia

A realização da dissertação foi baseada em pesquisa e estudo de artigos científicos, teses de mestrado e livros de referência, contando ainda com a cooperação da empresa UAVision através da disponibilização de meios e conhecimento relativo à área de VANTs.

De modo a promover a elaboração da dissertação, foram realizados os seguintes trabalhos:

- Publicação do artigo "Unmanned Aerial Systems in Military Environments: The Benefits of Interoperability", no âmbito da conferência "Sea-Conf 2017" (Apêndice G);
- Publicação do artigo "Interoperability of Unmanned Systems in Military Maritime Operations: Developing a controller for unmanned aerial systems operating in maritime environments", no âmbito da conferência "MTS/IEEE OCEANS'17" (Apêndice F);

- Colaboração para a elaboração do artigo “STRONGMAR Summer School 2016 – Joining theory with a practical application in Underwater Archeology”, no âmbito da conferência “MTS/IEEE OCEANS’17”;
- Colaboração para a elaboração do artigo “CBRN remote sensing using unmanned aerial vehicles: Challenges addressed in the scope of the GammaEx project regarding hazardous materials and environments”, no âmbito da conferência “6th International Conference on Risk Analysis and Crisis Response 2017”.

O método de desenvolvimento da dissertação foi definido de acordo com o processo representado na Figura 1, que contempla a procura de informação, a criação e implementação do controlador, e a otimização do controlador através da análise de resultados obtidos.

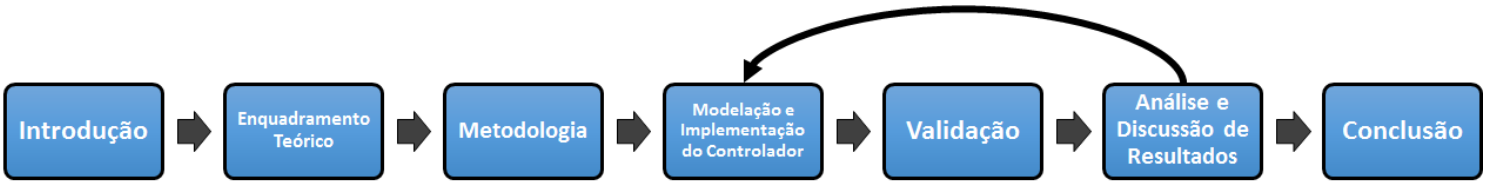

Figura 1 - Representação da metodologia utilizada na dissertação.

Com base no método de desenvolvimento e objetivos propostos para a elaboração da dissertação foi adotada a metodologia de investigação *Design Science Research* (Peffers, 2007) (Capítulo 2).

Estrutura

A estrutura da dissertação é composta por um capítulo de introdução, cinco capítulos de desenvolvimento e um capítulo de conclusão:

Introdução – É descrito o âmbito e contexto da dissertação, bem como a sua pertinência, apresentando o foco da investigação e a metodologia de investigação aplicada ao desenvolvimento da dissertação.

Capítulo 1: Enquadramento Teórico – Introduz e identifica conceitos essenciais para o estudo de interoperabilidade e para a criação de um controlador externo para um piloto automático de um VANT, através da descrição de normas de referência (*standards*) utilizadas, projetos com VNTs e da composição de um sistema e aplicações de um VANT. O capítulo é concluído com a descrição de conceitos relacionados com a implementação da funcionalidade de aterragem visual autónoma, nomeadamente, o estudo de sistemas de aterragem com marcas visuais, do modelo de um quadrirrotor e de processamento de imagem.

Capítulo 2: Metodologia – Apresenta a metodologia *Design Science Research* (Peffers, 2007) e enquadra a aplicação da metodologia com a realização da dissertação.

Capítulo 3: Modelação e Implementação do Controlador - É descrito o processo de criação e desenvolvimento do controlador, através da representação da arquitetura do sistema onde se insere, da arquitetura controlador e da descrição do funcionamento de cada módulo. O capítulo é concluído com a representação da ferramenta de operação do controlador.

Capítulo 4: Validação – São apresentados os cenários propostos para a execução de testes ao controlador criado, em particular, testes ao módulo de comando e controlo, e testes ao módulo de aterragem visual autónoma.

Capítulo 5: Análise e Discussão de Resultados – São testados os cenários propostos e apresentados os resultados obtidos, sendo os resultados sujeitos a análise e interpretação.

Conclusão - É apresentada uma síntese do trabalho realizado, as principais conclusões obtidas e propostas de trabalho futuro.

Capítulo 1. Enquadramento Teórico

1.1. Interoperabilidade de Sistemas

1.1.1. Introdução

A aplicação do conceito de interoperabilidade no funcionamento de um sistema permite o desenvolvimento e expansão das suas capacidades através da possibilidade de interagir com múltiplos sistemas diferentes, que desempenhem funções distintas, de modo a alcançar de forma eficaz um objetivo comum. Este é um conceito amplo que se estende a diferentes áreas de estudo, sendo diversas as suas definições: na área da saúde, a interoperabilidade consiste na capacidade de sistemas de informação de saúde funcionarem em conjunto por forma a desenvolverem a eficácia da prestação de serviços de saúde (HIMSS, 2016); no que toca a telecomunicações, interoperabilidade é definida como a capacidade de disponibilizar e receber serviços provenientes de sistemas distintos, permitindo que estes serviços cooperem de forma eficaz (Bloomfield, 1999); no âmbito da *North Atlantic Treaty Organization* (NATO), interoperabilidade consiste na capacidade que múltiplas nações possuem para desempenharem treinos, exercícios, tarefas e missões em conjunto, de forma eficaz (NATO, 2006); o IEEE (IEEE, 1990) e o *National Institute of Standards and Technology* (Huang, 2008) apresentam uma visão comum do conceito de interoperabilidade, que pode ser categorizada em níveis, tipos e graus, e que se baseia na capacidade de operação comum por parte de diferentes sistemas, ou componentes, cuja eficácia e sucesso se refletem na diminuição do esforço por parte do utilizador. Ambas entidades (IEEE e *National Institute of Standards and Technology*) referem que a interoperabilidade de sistemas deve ser alcançada através do uso de normas de referência (*standards*) (Capítulo 1.1.2.) que estabelecem características, parâmetros e regras de comunicação e interação; Quanto aos VNTs (Capítulo 1.1.3.), interoperabilidade baseia-se na capacidade de sistemas, unidades ou forças disponibilizarem e receberem serviços de outros sistemas, unidades e forças, de modo a desenvolver a eficácia de operação conjunta (Defense, 2013). O *Department of Defense* (2013) acrescenta, como exemplo, a implementação deste

conceito ao nível de equipamentos eletrónicos de comunicação, nos quais circulam informação e serviços, de forma satisfatória, entre utilizadores.

O desenvolvimento do conceito de interoperabilidade e de normas de referência, assim como as suas vantagens, têm vindo a ser demonstrados através de projetos que recorrem à cooperação e interação entre vários VNTs e o operador, para desempenhar funções de busca e salvamento, reconhecimento e patrulhamento, transporte de material, entre outras aplicações (Capítulo 1.1.3.).

1.1.2. Normas de Referência (*standards*)

A criação de normas de referência (*standards*) permite o desenvolvimento de estruturas, protocolos e arquiteturas que facilitam funcionamento comum, que previnem incompatibilidade, e que geram um ambiente de trabalho que apela a eficácia do desenvolvimento de um sistema. Apesar da adaptação a normas de referência (*standards*) apresentar um desafio, a sua utilização permite a redução de tempo e de custos a investir em desenvolvimento tecnológico, redução de riscos de investimento baseado em tecnologia desconhecida, aumenta a produtividade e eficácia de processos, promove interoperabilidade de sistemas, e motiva o desenvolvimento das próprias normas, promovendo e garantindo a sua qualidade.

Relativamente ao desenvolvimento de VNTs, existem várias normas em vigor que demonstram potencial para se destacarem como referências, devido à capacidade das suas funcionalidades. Como por exemplo, *Standardization Agreement 4586* (STANAG 4586) (Marques, 2015), a Joint Architecture for Unmanned Systems (JAUS) (Wade, 2006) o *Robotic Operating System* (ROS) (Capítulo 1.1.2.1) e o *Micro Air Vehicle Communication Protocol* (MAVlink) (Capítulo 1.1.2.2.).

1.1.2.1. *Robotic Operating System*

Desde o desenvolvimento inicial da área da robótica, a criação e desenvolvimento tecnológico tem vindo a crescer ao longo dos anos através da intervenção de centros militares de desenvolvimento tecnológico, empresas

especializadas e publicação de estudos, dissertações de mestrado e doutoramentos. No entanto, a evolução de tecnologia pré-existente encontrava-se condicionada devido à dificuldade de interpretação de alguns dos seus conceitos base e à falta de ferramentas de desenvolvimento de *software* normalizadas, o que motivava a reconstrução do projeto original, dando origem a um processo de "reinvenção da roda" (Figura 2).

Figura 2 - Processo de "reinvenção da roda" no desenvolvimento de robótica (Leitner, 2015).

A criação do ROS veio ao encontro deste problema, proporcionando uma solução ao disponibilizar bibliotecas, ferramentas e funcionalidades normalizadas com capacidade de abstração de *hardware*[3]. Simplificaram-se tarefas como gestão de memória e processos, processamento paralelo, e tornou-se possível a criação e

[3]*Hardware* – Componentes e dispositivos físicos de um computador. Tech Terms (2017). *Hardware*. Consultado em 02 de Janeiro de 2017 no Tech Terms web site.
Web Site: https://techterms.com/definition/hardware

interligação de vários pacotes que desempenham funções distintas desde *drivers*[4] de controlo de sensores e periféricos, interfaces, processamento de imagem e som, entre outras diversas aplicações distintas. Possibilitou ainda, a criação de uma comunidade em que os seus integrantes contribuem diariamente com os seus próprios projetos e pacotes de funções. Desta forma estes contributos podem ser reutilizados por outros utilizadores para a construção de novos projetos, sem a necessidade adicional de desenvolver funcionalidades já existentes.

O ROS é um *framework*[5] de edição livre que permite a criação de *software* de robótica, com o objetivo de simplificar o desenvolvimento de sistemas robóticos complexos e detalhados. Foi criado em 2007 através de projetos que desenvolveram o estudo de conceitos relacionados com inteligência artificial aplicada à robótica, na Universidade de Stanford (e.g. Quigley, 2015). O estudo desta temática foi motivado pelos recursos provenientes da Willow Garage Inc., o que permitiu a realização de testes a implementações do ROS. Posteriormente, o desenvolvimento do ROS contou com os contributos de múltiplos indivíduos e entidades, algo que se tornou possível devido ao carácter de edição livre do ROS, e que continuou até aos dias de hoje.

No que toca ao seu propósito, o ROS foi criado por forma a atingir os seguintes objetivos: ponto-a-ponto, baseado em ferramentas, poliglota, leve, e gratuito e de edição livre (e.g. Quigley, 2015) (Figura 3).

[4]*Driver* – Programa que permite a integração e operação de um determinado componente ou dispositivo físico, num sistema operativo. Tech Terms (2017). Drivers. Consultado em 02 de Janeiro de 2017 no Tech Terms web site.
Web Site: https://techterms.com/definition/driver

[5] *Framework* – Estrutura, real ou conceptual, por camadas que define que tipo de programas podem ser desenvolvidos e como se relacionam. IT(2016).Information Technology standards and organizations glossary. Consultado em 12 de Junho de 2016 no Whatls Web Site: http://whatis.techtarget.com/definition/framework

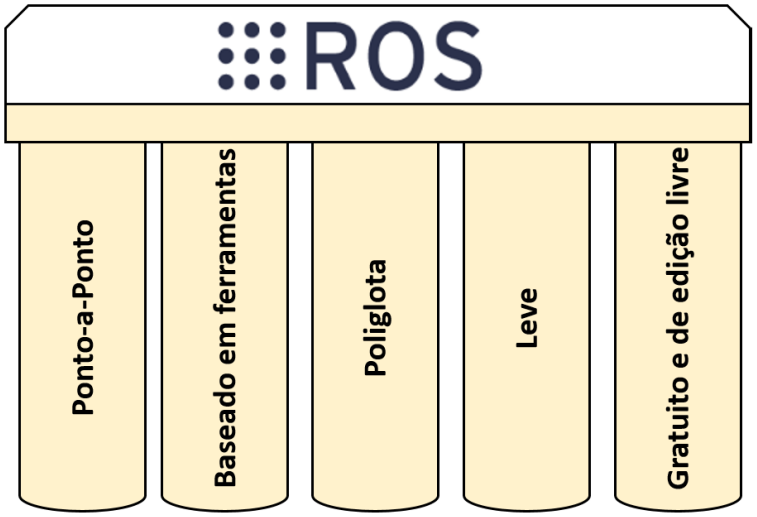

Figura 3 - Conceitos que suportam o ROS.

Conceptualmente, o ROS é definido por três níveis: nível de sistema de ficheiros, nível de grafo computacional, e o nível da comunidade (e.g. Romero, 2014) (e.g. Quigley, 2015).

Os recursos do sistema de ficheiros encontram-se organizados em pacotes e em pilhas. Pacotes são pastas que contêm "nós", bibliotecas externas, informação útil, ficheiros de configuração e manifestos que descrevem o conteúdo dos pacotes. Pilhas são pastas que contêm vários pacotes, e ficheiros de configuração que descrevem o conteúdo das pilhas, bem como as dependências relativas a outras pilhas.

O grafo computacional do ROS integra conceitos associados a *nós*, *tópicos*, *mensagens*, *serviços* e *sacos*:

- **Nó (*node*)**: Um *nó* é um programa executável que desempenha uma determinada função, sendo que quando é iniciado, necessita de se registar num *nó mestre* que guarda informação relativa ao registo. Desta forma, vários *nós* podem localizar outros *nós* através da informação dos registos contida no *nó mestre*, permitindo a comunicação direta entre *nós*. O *nó mestre* inclui ainda um *servidor de parâmetros*, onde os *nós* podem guardar

e partilhar dados. *Nós* podem comunicar, ou seja, enviar e receber dados (*mensagens*) ao publicar e subscrever *tópicos*.

- **Tópicos (*topics*)**: disponibilizam um meio para desempenhar trocas de informação assíncronas entre *nós*, ao transportarem *mensagens* de um certo tipo entre *nós*. Vários *nós* distintos podem subscrever e publicar dados no mesmo *tópico*.

- **Mensagens (*messages*):** representam estruturas que contêm dados de vários tipos (inteiros, booleanos, palavras, caracteres, etc.), ou até, outras *mensagens*. As *mensagens* são enviadas de *nó* para *nó*, encapsuladas em *tópicos*.

- **Serviços (*services*):** disponibilizam trocas de informação síncronas entre *nós*, sendo que são programas chamados através de *requisições*, publicando os seus resultados em *respostas*;

- **Sacos (*bags*):** são formatos utilizados para guardar e reencaminhar dados guardados em *mensagens*.

A comunicação entre *nós* é feita através de *tópicos*, de acordo com um modelo de publicação/subscrição. Os *nós* que pretendem transmitir informação, definem inicialmente o nome do *tópico* e o tipo de *mensagem* a transmitir. De seguida, iniciam o processo de publicação de *mensagens* através do *tópico* definido. A informação relativa ao nome e tipo de conteúdo dos *tópicos* que se encontram publicados é guardada no *nó* "roscore", assim como o nome do *nó* que publica o respetivo *tópico*. Desta forma, *nós* que pretendam obter um certo tipo de *mensagem* (transmitida por um *tópico* designado), subscrevem ao *tópico* que a contêm. Para tal, o *nó* subscritor obtêm a identificação do respetivo *tópico* ao aceder ao *nó* "roscore". Após a subscrição, o *nó* subscritor recebe *mensagens* contidas no *tópico* subscrito, garantindo um sistema de comunicação com implementação simplificada (e.g. Quigley, 2015).

De modo a demonstrar estes conceitos, o seguinte exemplo descreve o funcionamento de um sistema (hipotético) de anti-colisão associado a um sistema *Radio Detection and Ranging* (RADAR) e a um piloto automático de um determinado veículo. A Figura 4 representa os *nós* do sistema que desempenham funções distintas.

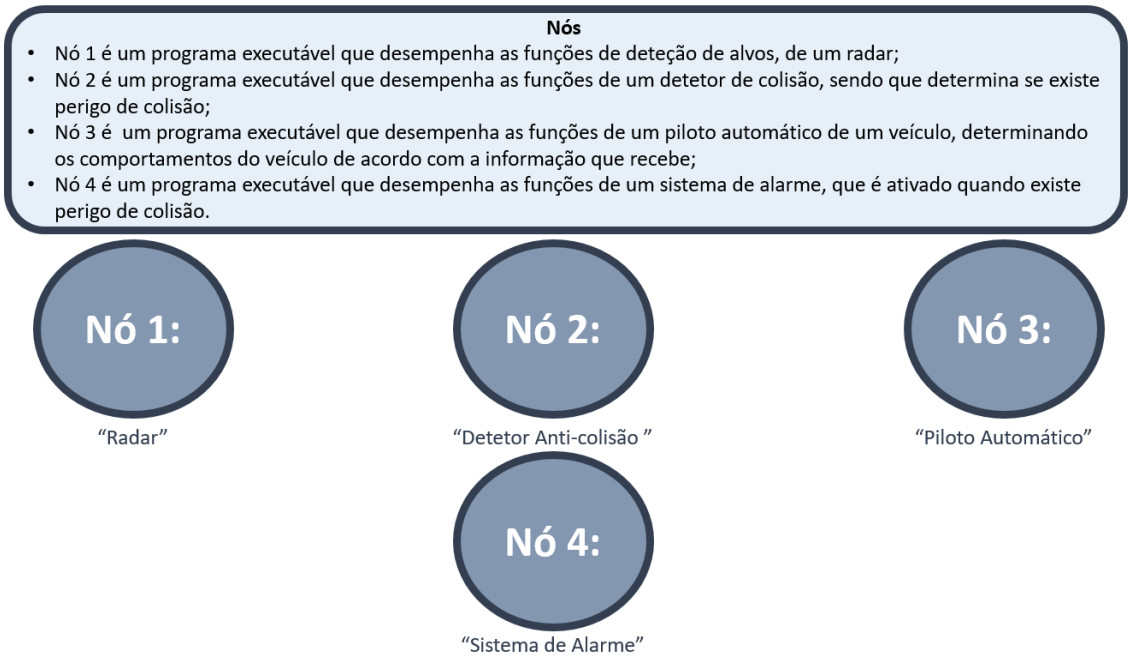

Figura 4 - Representação dos nós de um sistema desenvolvido em ROS.

O "Radar" (*nó* 1) publica informação sobre a posição e distância de alvos, relativa à localização do navio, no *tópico* "/posição_distancia_alvo". O "Detetor Anti-Colisão" subscreve o *tópico* "/posição_distancia_alvo" para receber a informação contida neste *tópico*. De acordo com a informação recebida, o "Detetor Anti-Colisão" determina o perigo de colisão, e posteriormente publica a informação relativa ao perigo de colisão

no *tópico* "/info_colisão", que é subscrito pelo "Piloto Automático". A seguinte Figura 5 esquematiza as interações relativas à publicação e subscrição de tópicos.

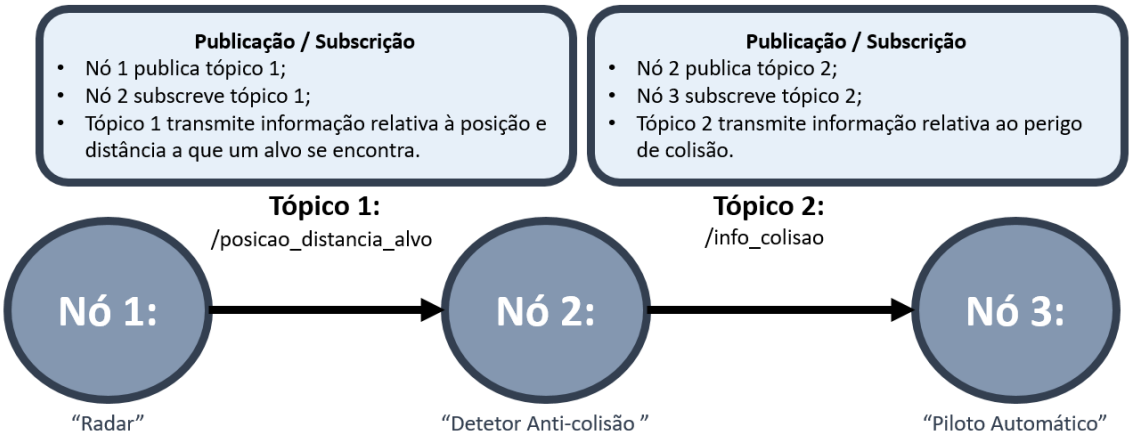

Figura 5 - Representação da comunicação através de tópicos de um sistema desenvolvido em ROS.

A Figura 6 representa as *mensagens* que são transmitidas e recebidas entre *nós*, bem um exemplo do conteúdo das respetivas *mensagens*.

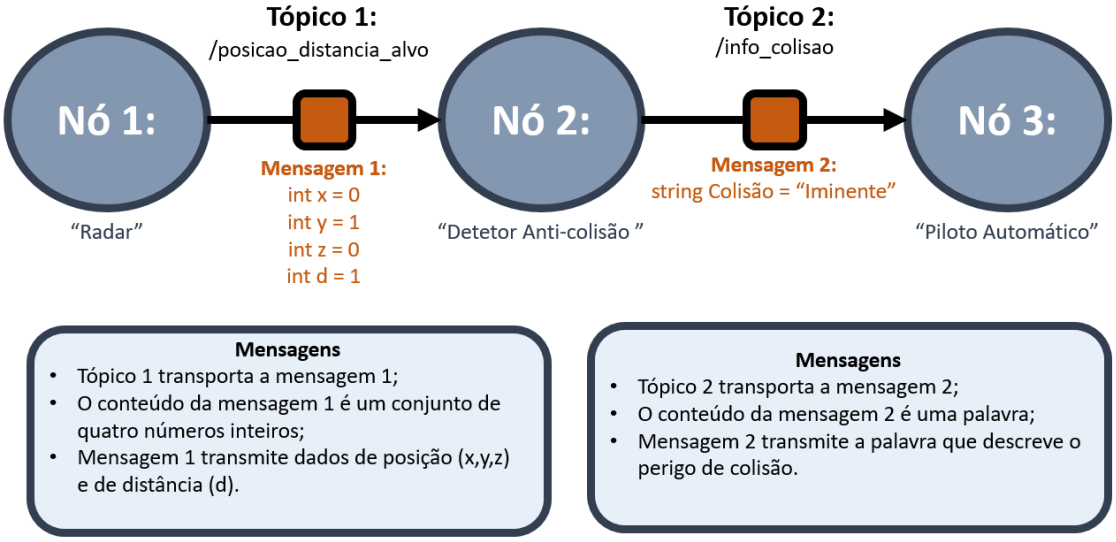

Figura 6 - Representação da troca de *mensagens*" entre *nós* de um sistema desenvolvido em ROS.

Na eventualidade de o perigo de colisão ser iminente, o “Piloto Automático” procede a alterar o seu rumo e velocidade, e chama o *serviço* prestado pelo “Sistema de Alarme”, para que o alarme de colisão iminente seja ativado (Figura 7).

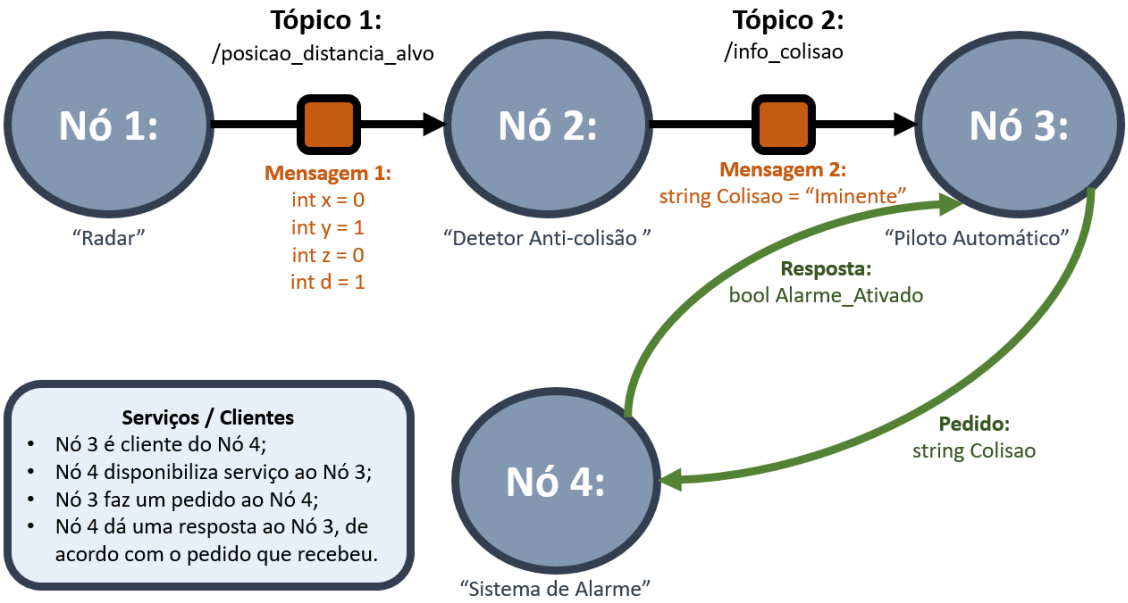

Figura 7 - Representação final de um sistema desenvolvido em ROS.

O ROS disponibiliza um sistema modular que permite o desenvolvimento de estruturas funcionais, flexíveis e versáteis, com a capacidade de suportar diferentes linguagens de programação e de carácter de edição-livre, o que promove o seu desenvolvimento através dos contributos da comunidade ROS.

1.1.2.2. *Micro Air Vehicle Communication Protocol*

O MAVLink é um protocolo de comunicação originalmente concebido para micro-veículos aéreos não tripulados (MVANTs), que atualmente é utilizado para operar diversos tipos de VNTs, tornando-se uma norma de referência no que toca a protocolos de comunicação de VNTs. O protocolo MAVLink é baseado numa leve biblioteca de mensagens utilizada para estabelecer comunicações entre o VNT e a estação de controlo em terra (*ground control station (GCS)*). As mensagens MAVLink encontram-se definidas e descritas no formato *eXtensible Markup Language* (XML), sendo que para cada tipo de mensagem se encontra atribuído um conteúdo, composto por parâmetros de diversos tipos. Estas mensagens são convertidas para ficheiros codificados em *C* por forma a serem transmitidas em pacotes, sendo transmitida uma sequência ordenada de *bytes* via um transdutor (*Universal Serial Bus*, *Wi-Fi*, etc.) (Coombes, 2012) (Reker, 2015).

Um pacote MAVLink encontra-se definido de acordo com a seguinte estrutura (Figura 8) (Reker, 2015):

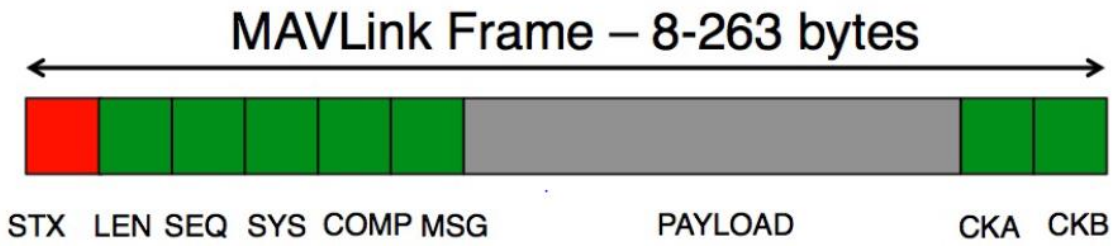

Figura 8 - Estrutura de um pacote MAVLink (Reker, 2015).

Em que cada parâmetro do pacote é definido de acordo com a seguinte tabela 1 (Reker, 2015):

Tabela 1 - Descrição de um pacote MAVLink (Reker, 2015).

Parte da Mensagem	Índice do *Byte*	Conteúdo	Descrição	Valor
Cabeçalho	0	STX	Define o início de um novo pacote.	V1.0: 0xFE /v0.9: 0x55
	1	LEN	Indica o comprimento da carga útil (*payload*).	0 – 255
	2	SEQ	Número de sequência. Permite a deteção de perda de pacotes.	0 – 255
	3	SYS	Número de identificação do sistema emissor. Permite a diferenciar diferentes VNTs que operem na mesma rede.	1 – 255
	4	COMP	Número de identificação do componente emissor. Permite diferenciar diferentes componentes do mesmo sistema.	0 – 255
	5	MSG	Número de identificação da mensagem. Define o conteúdo da carga útil (*payload*) e o método de descodificação.	0 - 255
Carga Útil (*payload*)	6 - (n+6)	PAYLOAD	Carga útil (*payload*) de tamanho variável definido em *bytes.*	(0 - 255) bytes
Controlo de erros	(n+7) até (n+8)	CKA/CKB	*Checksum A* e *Checksum B* desempenham a função de controlo de erros através de um *Cyclic Redundancy Check* duplo (Kowalk, 2006). Mensagens que sejam recebidas com *chesksum* errado, são descartadas.	

O tamanho dos pacotes MAVLink varia entre 8 e 263 *bytes,* sendo compostos por: cabeçalho, mensagem e correção de erros. O cabeçalho define a identificação do pacote, o comprimento da mensagem, o número de sequência do pacote, o número de identificação do sistema emissor, o número de identificação do componente emissor e o número de identificação da mensagem. A mensagem é transmitida como a carga útil (*payload*) do pacote, cujo conteúdo e tamanho varia consoante o tipo de mensagem a

enviar (e.g. *heartbeat*, aterragem, seguimento de *waypoints*[6], entre outros). A componente de controlo de erros do pacote permite avaliar se uma mensagem foi transmitida com sucesso, ou com erros, o que implica uma retransmissão da mensagem, garantindo robustez e fiabilidade ao processo de comunicação.

O protocolo MAVLink encontra-se implementado numa variedade de *softwares* de pilotos-automáticos e de ECTs, os quais incluem *Ardupilot*, *Parrot AR*, *Pixhawk*, *QGroundControl*, *ArduPilot Mega Planner*, entre outros (Coombes, 2012).

Apesar do protocolo MAVLink não possuir o mesmo grau de complexidade e de confidencialidade que outros protocolos militares de comunicação, possui um carácter gratuito que promove a sua acessibilidade e desenvolvimento ao permitir a criação e integração de novos formatos de mensagens. O facto de integrar linguagens leves, e de possuir pacotes com cabeçalho limitado a seis *bytes* permite um desempenho rápido e eficiente (Fuller, 2014) (Butcher, 2014).

[6] *Waypoint* – conjunto de coordenadas que identificam um ponto físico no espaço. Tipicamente utilizados em navegação com o GPS. STOREY, M. et al. (2006), Shared waypoints and social tagging to support collaboration in software development, In Proceedings of the 2006 20th anniversaryconference on Computer supported cooperative work

1.1.3. Veículos Não Tripulados

VNTs são definidos como veículos sem capacidade de integrar a bordo um operador humano com controlo sobre o veículo (Eisenbeiss, 2004). O que implica que o controlo seja exercido através dos modos: automático, semi-automático, teleoperado e controlo remoto (Huang, 2008).

A classificação de VNTs é feita de acordo com o meio onde operam, sendo que existem veículos aéreos não tripulados (VANTs), veículos terrestres não tripulados (VTNTs), veículos superfície não tripulados (VSUNTs) e veículos subsuperfície não tripulados (VSNTs) (Fleming, 2015) (Figura 9).

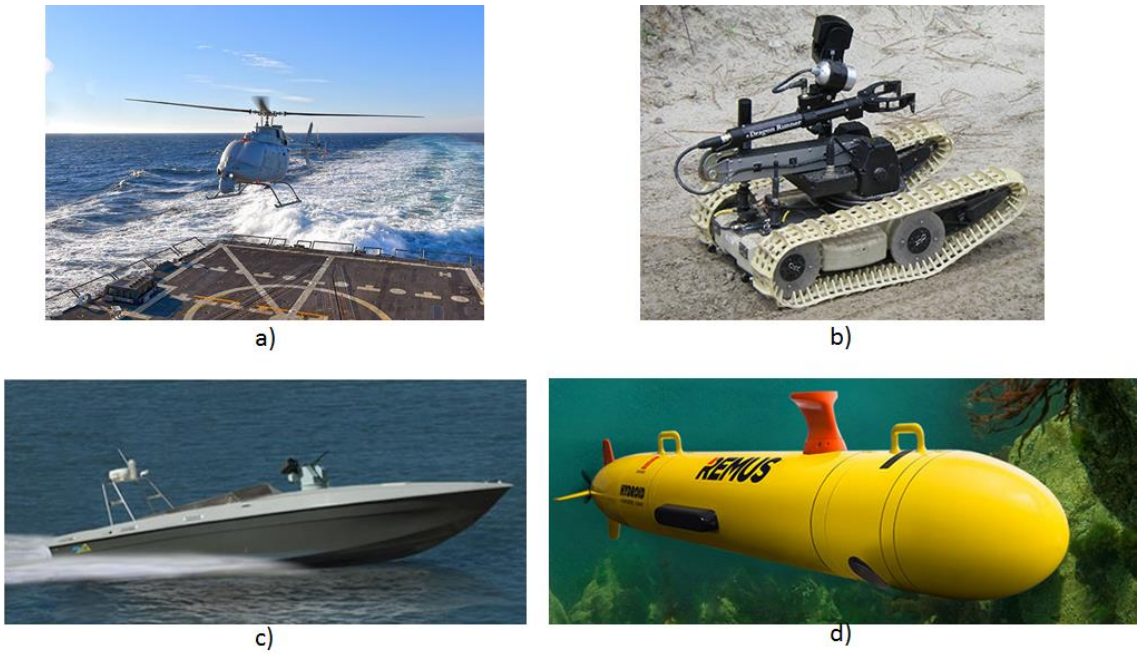

Figura 9 - Classificação de VNTs de acordo com o meio de operação: a) VANT (Northrop Grumman, 2017); b) VTNT (Qinetic, 2016); c) VSUNT (Defense Update, 2006); d) VSNT (Subsea World News, 2016).

A utilização de VNTs demonstra-se vantajosa, sendo que está associada ao facto de diminuir a presença do interveniente humano no local da ação, no desempenho de uma missão ou tarefa, o que se apresenta como uma das grandes vantagens da sua utilização. Esta vantagem é enaltecida pela utilização de VNTs para a execução de missões que envolvam a presença de material explosivo ou radioativo. O uso de VNTs

de pequenas dimensões permite o empenhamento de VNTs em missões que envolvem o acesso a espaços reduzidos e confinados, inacessíveis por um interveniente humano ou por um veículo tripulado. O empenhamento de VNTs demonstra-se, também, vantajoso em missões/tarefas repetitivas e entediantes, como vigilância e patrulhamento, sendo que mantêm o seu nível de eficácia constante durante todo o período de empenhamento. Outra vantagem proveniente do uso de VNTs reside no seu custo de empenhamento, quando comparado com o custo associado ao empenhamento de um veículo tripulado (Fleming, 2015).

De modo a aproveitar e a explorar as vantagens da utilização de VNTs, torna-se necessário desenvolver estruturas que promovam a interoperabilidade entre VNTs. O desenvolvimento de projetos como o *Integrated Components for Assisted Rescue and Unmanned Search Operations* (ICARUS) e o *Deployable SAR Integrated Chain with Unmanned Systems* (DARIUS) visa criar estruturas interoperáveis de VNTs por forma a executar missões de busca e salvamento em ambientes marítimos e terrestres, bem como durante uma catástrofe (Serrano, 2015).

O projeto ICARUS tem como objetivo o empenhamento de VNTs para missões de busca e salvamento, em particular, pretende desenvolver tecnologia para a deteção, localização e salvamento de vidas humanas. O projeto ICARUS estabelece uma rede de comunicações que integra GCSs e VNTs, por forma a promover o fluxo de informação, navegação autónoma, e deteção e identificação de vítimas, o que permite à estação base processar os dados recebidos em informação geográfica. Esta informação geográfica é utilizada para a criação de um mapa situacional que descreve o panorama e que auxilia o processo de tomada de decisão. O projeto ICARUS envolve a utilização de VANTs, VTNTs e VSUNTs que incorporam controladores desenvolvidos em ROS e JAUS, que são ligados ao módulo de comando e controlo através de adaptadores para JAUS, promovendo interoperabilidade e o uso de JAUS como uma norma de referência.

Desta forma, o projeto ICARUS consiste no desenvolvimento da cooperação entre VNTs para o desempenho de missões de busca e salvamento, desenvolvendo: sensores para a deteção de vítimas, uma rede de comunicações sem-fios, integrando ferramentas de busca e salvamento com VNTs em sistemas de comando e controlo, e garantindo treino e um sistema de suporte para equipas de busca e salvamento. O projeto ICARUS foi reconhecido internacionalmente ao receber o "*Multi-Robot Coordination Award*" no "*Multi-Domain Robotics Competition – EuRathlon 2015*" (Marques, 2016b) (Figura 10).

Figura 10 - Projeto ICARUS (Marques, 2016b).

O projeto DARIUS tem o objetivo de desenvolver uma estrutura funcional que permita a utilização de VNTs para desempenharem missões de busca e salvamento em situações de catástrofe, promovendo a partilha de informação e contribuindo para a interoperabilidade de sistemas. A implementação deste projeto permite ainda determinar os requisitos para futuros VNTs dedicados a missões de busca e salvamento. A organização do projeto DARIUS compreende três níveis operacionais (coordenação, tático e execução) que cooperam para dar resposta a cenários de catástrofes (terramotos, incêndios florestais, inundações, entre outros) através da intervenção de múltiplas agências internacionais, de acordo com a estrutura definida na Figura 11.

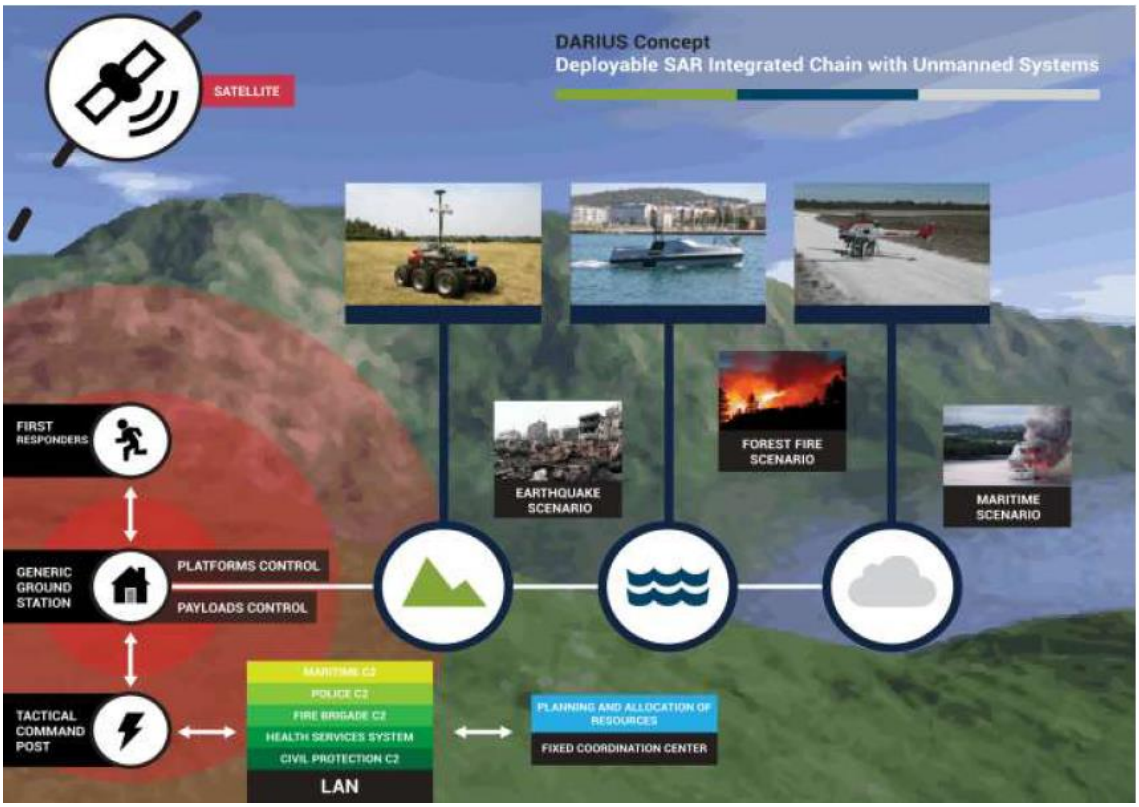

Figura 11 - Conceptualização do projeto DARIUS (Chrobocinski, 2012) .

O projeto DARIUS consiste no desenvolvimento de uma resposta rápida a cenários de catástrofe, que integra VNTs em estruturas de comando e controlo pré-existentes, estabelecendo uma rede de comunicações que engloba todo o sistema. Permite ainda a utilização de VNTs para retransmissão de comunicações, desenvolver capacidades de navegação de VNTs em ambientes hostis, e para a criação de interações entre VNTs e as primeiras ações de socorros (Chrobocinski, 2012).

A criação destes projetos é motivada por benefícios que resultam do desenvolvimento de interoperabilidade de sistemas, como:

- Redução de custos operacionais e de complexidade;
- Redução de problemas de incompatibilidade;
- Cooperação e interação entre sistemas distintos;

- Promoção da criação e crescimento de estruturas heterogéneas;
- Promoção de colaboração conjunta (entre diferentes entidades) e da criação de tecnologia conjunta;
- Crescimento da capacidade operacional de um sistema (e.g. NATO).

Contudo existem alguns desafios relacionados com o desenvolvimento de interoperabilidade de sistemas, associados a normas de referência (*standards*) de acesso restrito e a requisitos de segurança para o acesso a informação.

1.2. Veículo Aéreo Não Tripulado

1.2.1. Introdução

No âmbito da presente dissertação, torna-se essencial a descrição de VANTs, em particular, a sua definição e classificação (subcapítulo 1.2.2.), a constituição de um sistema de um VANT (subcapítulo 1.2.3.) e as suas principais aplicações no contexto civil e militar (subcapítulo 1.2.4.).

1.2.2. Definição

VANTs são plataformas sem capacidade de integrar a bordo um operador humano, que podem ser operados automaticamente ou manualmente através de uma GCS, para desempenhar tarefas e missões no meio aéreo (Bendea, 2008) (Cosic, 2013) (Austin, 2010) . Atualmente existem VANTs de asa fixa, asa rotativa, asa basculante (*flapping wings*) e dirigíveis, que podem ser classificados de acordo com o seu peso, ou de acordo com a altitude de operação (Figura 12) (Austin, 2010) (Gupta, 2013).

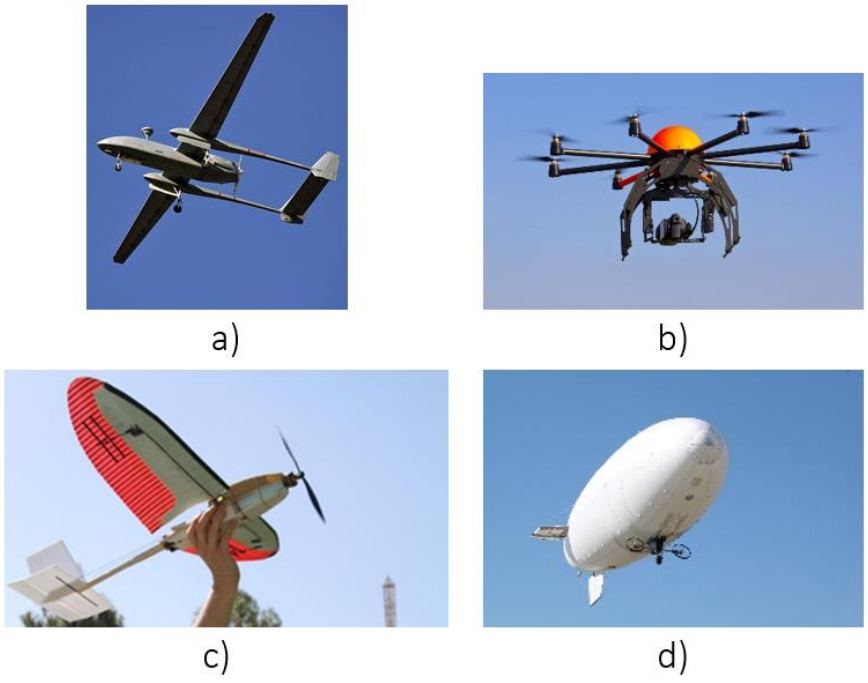

Figura 12 - Classificação de VANTs: a) asa fixa (Royal Australian Air Force, 2017); b) asa rotativa (Height Tech, 2016); c) asa basculante (*flapping wings*) (Mckeegan, 2011); d) dirigível (Aero Drum, 2016).

1.2.3. Sistema de um VANT

Austin (2010) define a composição de um sistema de um VANT nos seguintes subsistemas: veículo (ou plataforma), o sistema de navegação, carga útil (*payload*), sistema de comunicações, GCS, sistema de lançamento e recolha, interfaces, equipamento de apoio e de transporte. Fleming (2015) divide a composição de um sistema de um VANT em plataforma, carga útil (*payload*), GCS, infraestrutura de comunicações, operador humano e manutenção. Com base nas composições previamente definidas, a composição de um sistema de um VANT apresentada nesta dissertação encontra-se dividida em três partes principais: veículo (*vehicle*), ligação de dados (*datalink*) e comando e controlo (*command and control*). A Figura 13 representa a arquitetura funcional do modelo de um sistema de um VANT, descrito nesta dissertação.

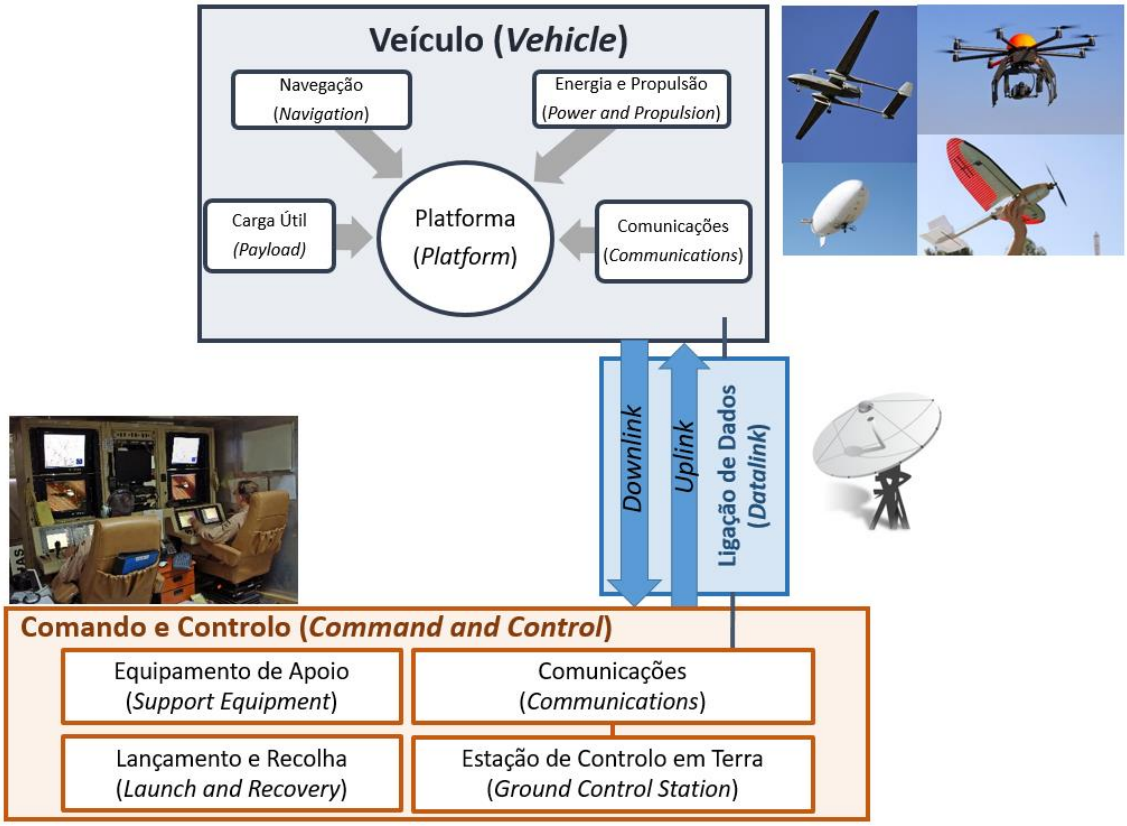

Figura 13 - Composição de um sistema de um VANT.

1.2.3.1. Veículo (*Vehicle*)

O veículo (*vehicle*) é composto por uma plataforma (*platform*) que integra a carga útil (*payload*), o subsistema de navegação (*navigation*), energia e propulsão (*power and propulsion*), e subsistema de comunicações (*communications*).

A plataforma (*platform*) de um VANT é um dos possíveis parâmetros de classificação, sendo que um VANT pode ser de asa fixa, asa rotativa, asa basculante (*flapping wings*) e dirigível. VANTs de asa rotativa podem ser classificados de acordo com o número de hélices (e.g. trirotor, quadrirrotor e hexarotor) (Gupta, 2013).

A carga útil (*payload*) é a denominação atribuída a qualquer carga, equipamento ou sensor que não faz parte da composição original do veículo, e que pode ser acoplado ao veículo por forma a promover o sucesso do VANT no desempenho de tarefas e missões específicas. O tipo de carga útil (*payload*) difere de acordo com o tipo de missão a desempenhar pelo VANT, sendo que pode variar desde sensores eletro-ópticos, cameras de infra-vermelho, sistemas RADAR, sistemas *Light Imaging Detection and Ranging* (LIDAR), sensores de condições atmosféricas, entre outros (Fleming, 2015; Austin, 2010).

Por forma a controlar o voo, o veículo possui um subsistema de navegação (*navigation*) que recolhe dados através de sensores embutidos no veículo, e que processa os dados recolhidos por forma a determinar a posição e a atitude (Capítulo 1.3.2.2) do veículo. O cálculo da posição e o controlo do movimento do veículo podem ser efetuados através de subsistemas de navegação: inercial, GPS, rádio, *waypoints*, *Doppler*, acústico, LIDAR, RADAR, visual, entre outros. Módulos de pilotos-automáticos como *Ardupilot* ou *Pixhawk*, desempenham a função de controlo local do veículo e de processamento de dados de navegação, atuando como o subsistema de navegação de VANTs em que se encontram integrados (Austin, 2010) (Elkaim, 2012) (Li, 2014).

A energia e propulsão (*power and propulsion*) garantem a operação e autonomia de um VANT. A fonte de energia de um VANT pode ser gasolina, gasóleo, baterias de lítio, hidrogénio líquido e energia solar (Griffis, 2009). Quanto à propulsão de um VANT,

Griffis (2009) define que os sistemas de propulsão podem ser motores recíprocos, motores rotativos de *Wankel*, turbinas a gás, foguetão, motores elétricos, baterias, *proton exchange membrane fuel cell*, células fotovoltaicas. A maioria destes sistemas de propulsão recorre a hélices como atuadores de propulsão.

1.2.3.2. Ligação de Dados (*datalink*)

A ligação de dados (*datalink*) estabelece a ligação entre os subsistemas de comunicações (*communications*) da GCS e do veículo (*vehicle*). Comunicações (*communications*) estabelecidas no sentido ascendente, da GCS para o veículo, que transmitem o plano de voo, comandos de voos para o subsistema de navegação, comando de controlo para a carga útil e informação sobre a posição do veículo relativa à GCS, são definidas por *uplink*. Comunicações no sentido descendente, do veículo para GCS, que transmitem dados sobre a posição do veículo, dados provenientes da carga útil (*payload*) e dos subsistemas, dados de temperatura e de níveis de combustível, são definidas por *downlink*.

A ligação de dados (*datalink*) pode ser estabelecida através de sistemas rádio, laser, fibra ótica, *Wi-Fi*, *Global System for Mobile Communications* (GSM), entre outros (Austin, 2010) (Wzorek, 2006) (Xie, 2016).

O protocolo MAVLink enquadra-se na componente da ligação de dados (*datalink*) de um sistema de um VANT, em particular, define o protocolo de comunicações utilizado para a transmissão e receção de comandos de voo, por parte da GCS.

1.2.3.3. Comando e Controlo (*Command and Control*)

Comando e controlo (*command and control*) contém GCS, equipamento de lançamento e recolha (*launch and recovery*), equipamento de apoio (*support equipment*) e subsistema de comunicações (*communications*).

A GCS é um centro de comando e controlo que pode ser estabelecido a bordo de uma plataforma terrestre, marítima ou aérea, a vários alcances, e que permite o controlo do VANT. Permite a transmissão de instruções de navegação, definição de

objetivos, monitorização de movimentos e de informação (imagens, vídeo, áudio, etc.) transmitida pelo VANT (Jovanovic, 2008). No caso em estudo nesta dissertação, o controlador ROS complementa e amplifica a capacidade de comando e controlo do sistema, garantindo um interface de controlo para o operador, e garantindo comunicações com o módulo do piloto automático.

O equipamento de lançamento e recolha (*launch and recovery*) é um fator condicionante da capacidade operacional de um VANT, cuja conceptualização lida com desafios como: garantir a segurança e a operabilidade do veículo durante o período de lançamento e de recolha, adaptação ao VANT, portabilidade, autonomia e manutenção reduzida. As plataformas de lançamento e recolha de VANTs podem ser de natureza elétrica, mecânica ou hidráulica, de acordo com o peso e dimensões do VANT, ou no caso de VANTs de pequenas dimensões, o lançamento e recolha pode ser manual. No caso de VANTs de asa rotativa e dirigíveis, não existe necessidade de plataformas de lançamento e recolha, dada a natureza da propulsão dos VANTs. No caso dos VANTs de asa fixa e de asa basculante, os sistemas de lançamento podem ser: lançamento assistido por foguete, lançamento por elástico, lançadores hidráulicos, elétricos e pneumáticos. Quanto aos sistemas de recolha, podem ser: rede, linha de recuperação, *skyhook*, *windsock* ou de paraquedas (Clapper, 2007; Eriksson, 2013).

Equipamento de apoio (*support equipment*) incluí manuais de manutenção de equipamentos, consumíveis, sobressalentes, ferramentas, equipamento subsidiário e transporte. O meio de transporte difere de acordo com as dimensões do VANT, sendo que pode ser via veículos terrestres, atrelados, aeronaves, embarcações e navios, ou até mochilas no caso de VANTs de dimensões reduzidas (Austin, 2010).

1.2.4. Aplicações

No âmbito civil, os VANTs podem desempenhar as seguintes tarefas/missões (Austin, 2010) (Nehme, 2006) (Rosa, 2016) (Skrzypietz, 2012):

- Transportar carga ou passageiros;
- Monitorização;
- Busca e Salvamento;
- Pesquisa;
- Exploração científica;
- Agricultura e silvicultura;
- Transmissões desportivas e realização de filmes;
- Engenharia e Construção;
- Fotografia;
- Combate a incêndios;
- Retransmissão de telecomunicações;
- Resposta a catástrofes e avaliação de danos;
- Previsão meteorológica através da análise de amostras.

No âmbito militar, os VANTs podem desempenhar as seguintes tarefas/missões (Austin, 2010) (Fletcher, 2000) (Nehme, 2006) (Rosa, 2016):

- Operações com material Nuclear, Biológico e Químico (NBQ);
- Reconhecimento e Recolha de informação;
- Entrega de carga útil (*payload*);
- Segurança Marítima;

- Guerra antiaérea;
- Vigilância;
- Operações de Interdição Marítima (MIO);
- Patrulhamento de Fronteiras;
- Inspeção/identificação;
- Apoio a forças de operações especiais;
- Comunicação (voz e dados).

1.3. Aterragem Autónoma de Veículos Aéreos Não Tripulados

1.3.1. Introdução

A aterragem de um VANT é um procedimento que define a sua capacidade operacional, sendo um dos momentos chave que contribui para o sucesso de um voo. Durante esta fase, o VANT assume a posição mais adequada para efetuar a descida, realizando constantes correções à sua posição relativa ao local de aterragem. O processo de correção da posição apresenta-se como um desafio durante a descida, particularmente em condições ambientais adversas que prejudiquem a capacidade do VANT identificar corretamente o local de aterragem. A capacidade de um VANT realizar aterragem autónoma visual, não só de forma intencional, mas também como um mecanismo de segurança na ocorrência de perdas de comunicações, perda de sinal GPS, baixo nível de autonomia, ou erros de *software*/*hardware*, valoriza a capacidade operacional de um VANT.

Uma vez que a aterragem do VANT é direcionada para o contexto naval, torna-se pertinente a adoção de uma abordagem que permita o uso de marcas de aterragem presentes no convés de voo de diversos tipos de navio, ou que possam ser integradas numa estrutura portátil e de montagem simples. Através de marcas fiduciais torna-se possível a aplicação de métodos de aterragem visual, que implicam baixo consumo de energia, carga útil (*payload*) e custos reduzidos (quando comparado com métodos de aterragem não-visual), recorrendo apenas a câmaras de vídeo (subcapítulo 1.3.2.). Permitem ainda a independência de sinal GPS para realizar a aterragem (Cesetti, 2010). Marcas visuais são definidas como um tipo de marcas fiduciais que representam os vários tipos de marcas de aterragem de referência, o que simplifica o processo de aquisição visual da marca.

De modo a implementar a capacidade de aterragem autónoma, é essencial conhecer o comportamento físico do VANT, em particular as expressões que definem a sua posição e orientação no espaço (subcapítulo 1.3.3.). A pose representa a posição e orientação de um corpo rígido no espaço em relação a um referencial em terra, ou no

caso desta dissertação, em relação à marca de aterragem, sendo um elemento essencial para a implementação da capacidade de aterragem autónoma visual.

A utilização de marcas visuais como marcas de aterragem implicam o estudo de métodos de processamento de imagem que permitam a identificação da marca de aterragem e a extração da pose do VANT (subcapítulo 1.3.4), com base na imagem identificada. O cálculo da pose de um VANT a partir de imagens obtidas de uma marca de aterragem de referência, por um câmara a bordo, permite efetuar o controlo da posição e orientação do VANT em relação à marca de aterragem, que por sua vez possibilita a aterragem.

Nesta dissertação, a implementação da funcionalidade de aterragem visual autónoma é direcionada para um VANT de asa rotativa, com quatro hélices – Quadrirrotor, equipado com uma câmara de vídeo situada por baixo do centro de massa do veículo, na mesma vertical.

1.3.2. Sistemas de Aterragem de Veículos Aéreos Não Tripulados – Marcas Visuais

Nesta dissertação, o estudo de sistemas de aterragem de VANTs incide sobre o uso de marcas fiduciais, em particular, marcas fiduciais visuais. Marcas fiduciais são definidas como pontos, naturais ou artificiais, que quando presentes no campo de visão de um sistema de visualização e processamento de imagem, se estabelecem como referências permitindo a identificação de elementos presentes na imagem. São utilizadas em diversas aplicações como a criação de placas de circuito impresso, realidade virtual, medicina, sistemas de aterragem visual, entre outros. A utilização de marcas fiduciais simplifica o processo de identificação do local de aterragem, bem como o processo de estima de altitude e de alinhamento entre dois ou mais VANTs. A classificação de marcas fiduciais pode ser feita de acordo com a seguinte divisão: marcas naturais (Cesetti, 2010), marcas de realidade aumentada (Chaves, 2015), marcas de infravermelhos (Lugo, 2011), marcas cooperativas (Feng, 2013) e marcas visuais (Sanchez-Lopez, 2013) (Figura 14).

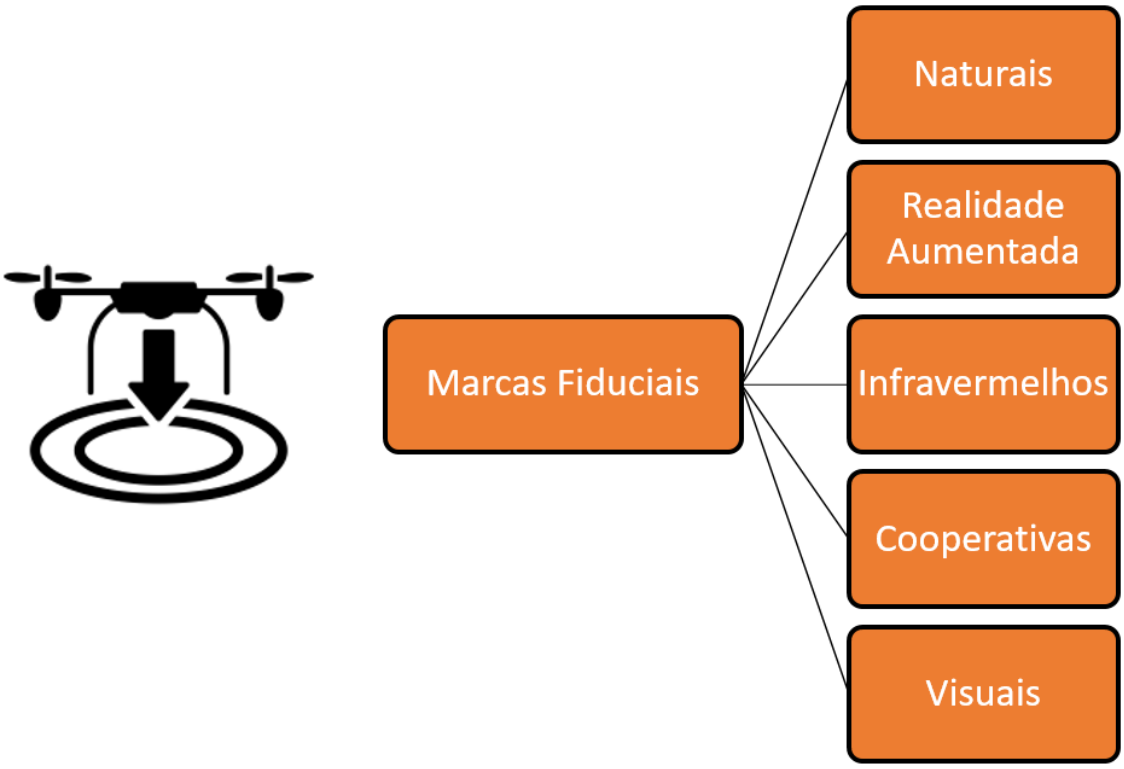

Figura 14 - Classificação de marcas fiduciais.

O método de aterragem associado ao uso de marcas visuais recorre ao uso de imagens e símbolos de referência para desempenhar a aterragem controlada, num local de aterragem designado. O processo de aterragem é iniciado com a aproximação e alinhamento do veículo com a marca visual de aterragem, que se torna possível através do processamento da marca presente nas imagens detetadas pelos sensores visuais do VANT. Este processamento implica a identificação da marca visual seguido pelo cálculo da pose do VANT, baseado na posição e orientação do VANT relativa à marca visual de aterragem. De seguida é calculada a distância da marca ao VANT, que é gradualmente reduzida de modo a aproximar o VANT do local de aterragem.

A implementação de métodos de aterragem visual autónoma, de VANTs de asa rotativa, têm sido alvo de investigação por parte de estudos que utilizaram o ROS

(Borshchova, 2016), e que associam aterragem de VANTs com outro tipo de VNTs, contribuindo para a interoperabilidade de sistemas (Weaver, 2013) (King, 2016). A Figura 15 representa um protótipo de um sistema, criado em ROS, que permite a aterragem de um VANT sobre um VSUNT (Weaver, 2013).

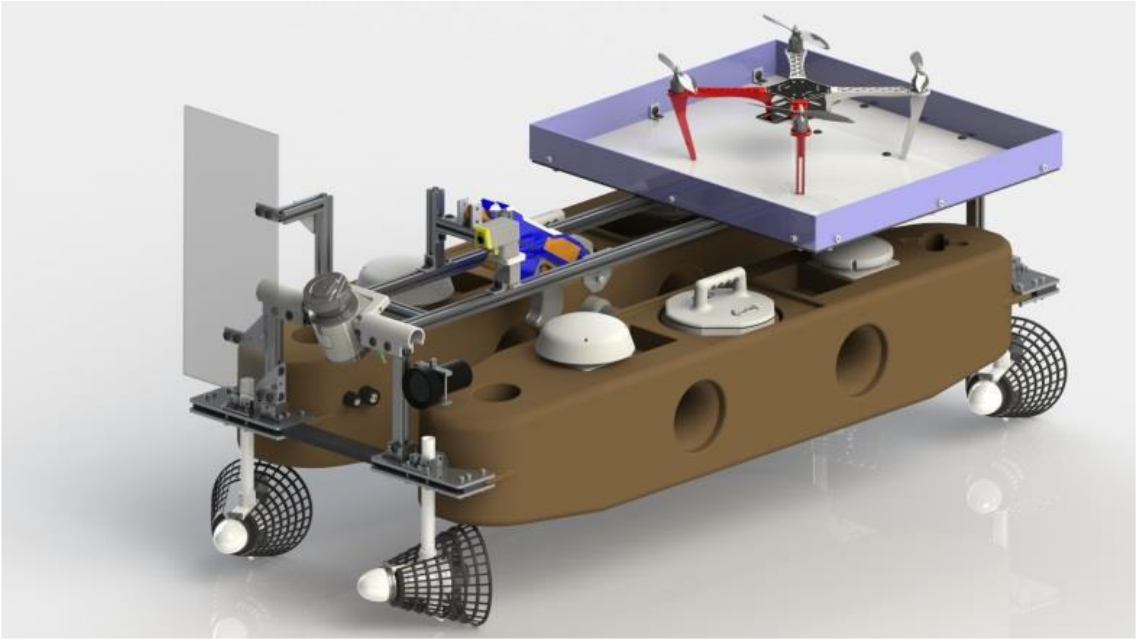

Figura 15 - Representação do sistema de aterragem composto por um VANT e por um VSUNT (Weaver, 2013).

Outros estudos encontram-se focados na aterragem de VANTs em plataformas navais, recorrendo a plataformas com seis graus de liberdade que simulam o movimento de navios (Figura 16) (Sanchez-Lopez, 2013) (Lee, 2012) (Garratt, 2009) (Santos, 2014).

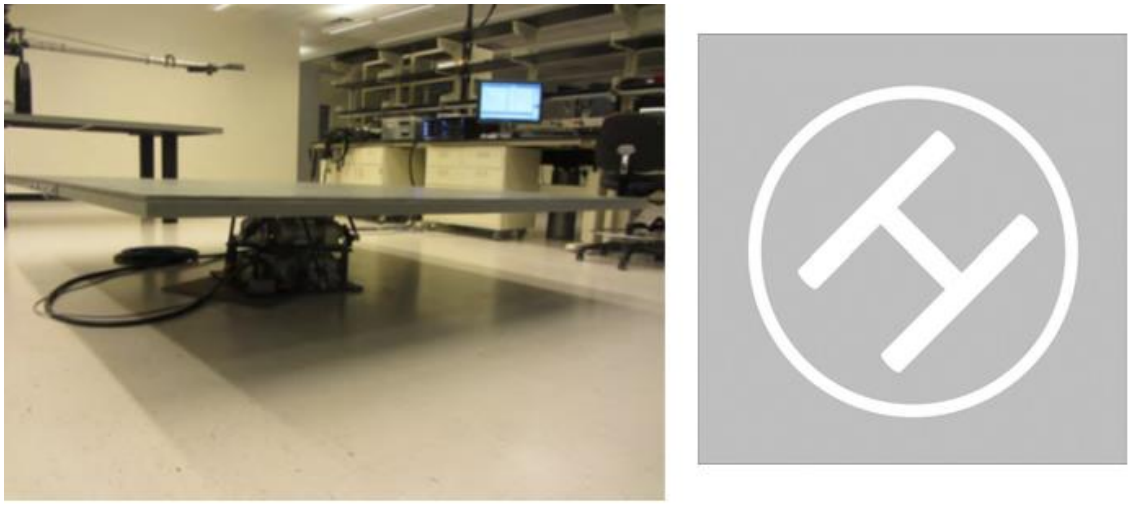

Figura 16 - Plataformas com seis graus de liberdade que simula o movimento de navios e respetiva marca visual de aterragem (Sanchez-Lopez, 2013).

1.3.3. VANT – Quadrirrotor

A abordagem inicial para a implementação da capacidade de aterragem visual autónoma carece da introdução do veículo a utilizar, em particular na definição do seu comportamento no espaço. No caso desta dissertação o estudo foi direcionado para um VANT de asa rotativa do tipo quadrirrotor, devido à sua popularidade e usabilidade, com principal destaque para a descrição da sua posição e orientação no espaço. Como tal é essencial a compreensão dos referenciais que permitem representar a posição e orientação (pose) de um VANT, uma vez que apenas conhecendo estes referenciais se torna possível descrever e definir o comportamento de um VANT durante o processo de aterragem.

1.3.3.1. Modelo

Um VANT do tipo quadrirrotor possuí estrutura fixa de configuração cruzada (em "X") com quatro hélices (frente, traseira, direita e esquerda) acopladas a um rotor, cada um. A movimentação de um quadrirrotor é originada através do sentido de rotação dos diferentes hélices, e através de variações na velocidade de rotação dos hélices. Dois hélices (frente e traseira) rodam no sentido dos ponteiros do relógio, enquanto os

restantes hélices (direita e esquerda) rodam no sentido oposto aos ponteiros do relógio, originando um fluxo de ar descendente que causa um impulso ascendente do veículo (Figura 17).

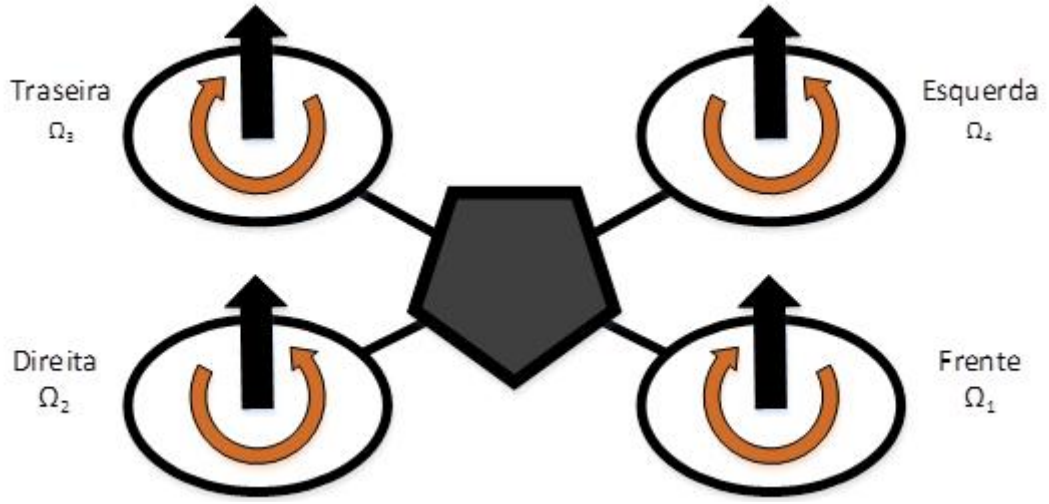

Figura 17 - Configuração de um modelo quadrirrotor.

1.3.3.2. Referenciais e Orientação de um Corpo Rígido

De modo a descrever um corpo rígido com seis graus de liberdade são definidos dois referenciais, o referencial inercial centrado num ponto em terra (referencial G), e um referencial BCS (*Body Coordinate System*) do veículo, centrado no respetivo centro de massa (referencial B) (Raza e Gueaieb, 2010) (Cai, 2011).

O referencial G (X_G, Y_G, Z_G) é um referencial fixo na superfície da terra (Figura 18).

O referencial B (X_B, Y_B, Z_B) é um referencial móvel com X_G a apontar para a frente do veículo, Y_G a apontar para a direita do veículo e Z_G a apontar para baixo (Figura 18).

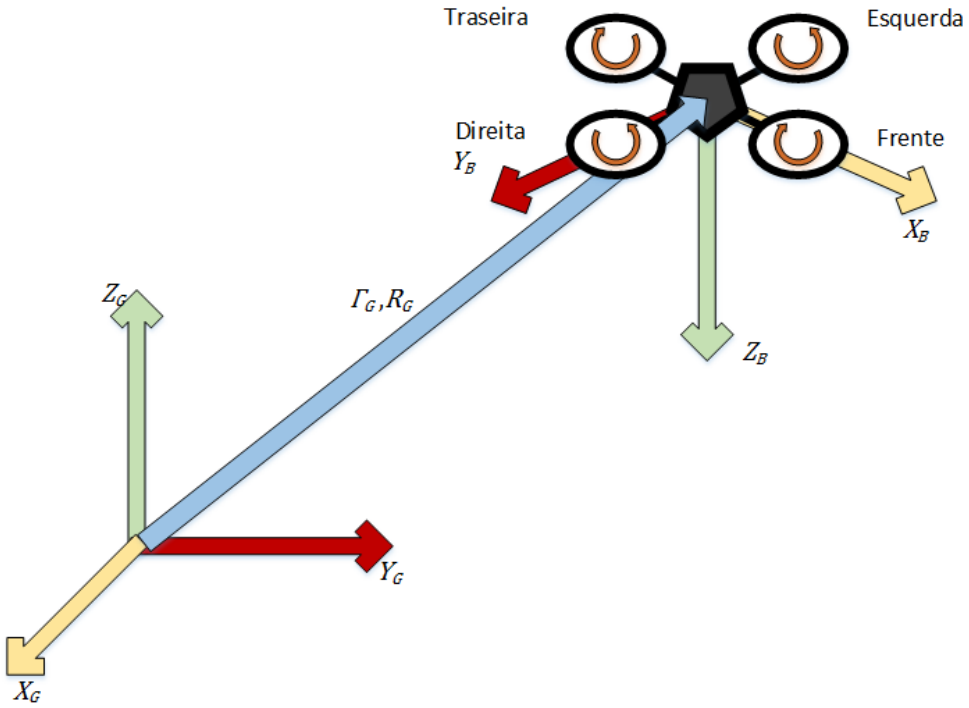

Figura 18 - Referencias (G e B) definidos para a descrição do modelo.

A representação do modelo quadrirrotor e a transformação entre referenciais estão associadas à definição da pose do veículo. A pose do quadrirrotor é um vetor com seis graus de liberdade composto por uma posição linear e uma atitude, de acordo com a Equação (1.1).

$$P^G = (\Gamma^G, \Theta^G) = (X, Y, Z, \phi, \theta, \psi) \quad (1.1)$$

A posição linear $\Gamma^G = (X, Y, Z)[m]$ encontra-se definida em coordenadas cartesianas, enquanto a atitude $\Theta^G = (\phi, \theta, \psi)[rad]$ é composta por ângulos de Euler, que representam as rotações em torno de cada eixo, X_G, Y_G, e Z_G, definidas como balanço (ϕ), cabeceio (θ) e guinada (ψ), respetivamente (e.g. Pervin, 1982) (e.g. Stevens, 2003) (Figura 19).

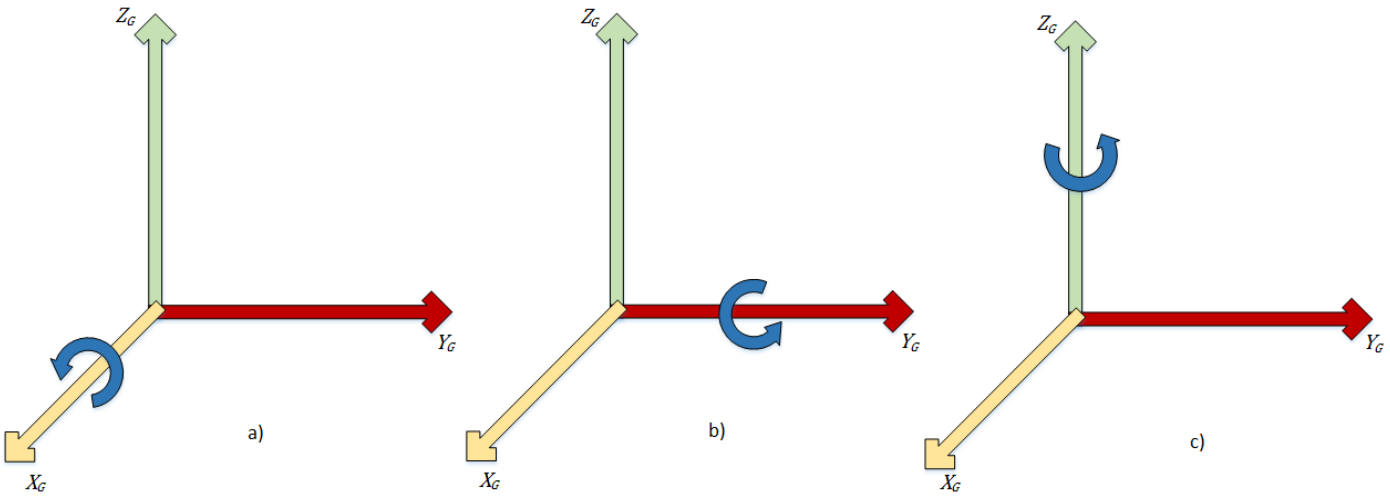

Figura 19 - Representação dos ângulos de Euler: a) balanço (ϕ); b) cabeceio (θ); c) guinada (ψ).

A velocidade do quadrirrotor é definida no referencial B por $V^B = (v^B, \omega^B)$. É composta por uma componente linear $v^B = (v_x, v_y, v_z)$ e uma componente angular $\omega^B = (\omega_x, \omega_y, \omega_z)$. A Equação (1.2) representa a velocidade do veículo relativa ao referencial B (Mahony, 2012) (Mellinger, 2010).

$$V^B = (v^B, \omega^B) = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z) \tag{1.2}$$

A relação entre os referenciais G e B é definida através da matriz de rotação R e do vetor de translação Γ, que permitem a transformação de um corpo rígido. A translação (mudança de posição entre dois referenciais) do referencial B para o referencial G é representada pelo vetor Γ^G. A rotação do referencial B para o referencial G é dada pelas Equações (1.3), (1.4) e (1.5) (Mahony, 2012) (Mellinger, 2010).

$$R_\phi^G = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\phi & sin\,\phi \\ 0 & -sin\,\phi & cos\,\phi \end{bmatrix}, R_\theta^G = \begin{bmatrix} cos\,\theta & 0 & -sin\,\theta \\ 0 & 1 & 0 \\ sin\,\theta & 0 & cos\,\theta \end{bmatrix}, R_\psi^G = \begin{bmatrix} cos\,\psi & sin\,\psi & 0 \\ -sin\,\psi & cos\,\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.3}$$

Em que,

$$R^G = R_\phi^G\, R_\theta^G\, R_\psi^G \tag{1.4}$$

$$R^G = \begin{bmatrix} cos\,\theta\, cos\,\psi & cos\,\theta\, sin\,\psi & -sin\,\theta \\ -cos\,\phi\, sin\,\psi + sin\,\phi\, sin\,\theta\, cos\,\psi & cos\,\phi\, cos\,\psi + sin\,\phi\, sin\,\theta\, sin\,\psi & sin\,\phi\, cos\,\theta \\ sin\,\phi\, sin\,\psi + cos\,\phi\, sin\,\theta\, cos\,\psi & -sin\,\phi\, cos\,\psi + cos\,\phi\, sin\,\theta\, sin\psi & cos\,\phi\, cos\,\theta \end{bmatrix} \tag{1.5}$$

Sendo R^G ortogonal ($R^{G^{-1}} = R^{G^T}$), com determinante igual a 1 ($\det R^G = 1$), de acordo com a sequência $R_\phi^G\ R_\theta^G\ R_\psi^G$.

A transformação de posições arbitrárias entre os referenciais G (posição p^G) e B (posição p^B) é possível através de uma transformação de corpo rígido, aplicada através da Equação (1.6) (Hartley, 2004).

$$\begin{bmatrix} p^G \\ 1 \end{bmatrix} = \begin{bmatrix} R^G & \Gamma^G \\ 0_{1x3} & 1 \end{bmatrix} \cdot \begin{bmatrix} p^B \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} p^B \\ 1 \end{bmatrix} = \begin{bmatrix} R^G & \Gamma^G \\ 0_{1x3} & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} p^G \\ 1 \end{bmatrix} \quad (1.6)$$

A representação de uma matriz de rotação R pode ser apresentada na forma de um Quaternião ou através da rotação de Rodrigues (Kuipers, 1999) (Gallego, 2015). Estes métodos de representação são relevantes para a dissertação, uma vez que facilitam a manipulação de dados presentes em matrizes de rotação e são utilizados pelo ROS e por funções do *OpenCV*[7].

Quaterniões permitem uma representação 3D de números complexos, bem como a descrição da rotação de um corpo rígido. A utilização de Quaterniões para a representação da rotação de um corpo rígido, permite evitar um problema de ambiguidade relacionado com a representação da rotação através de uma matriz R (ângulo de Euler), conhecido por *gimbal lock. Gimbal lock* ocorre quando dois de três anéis de um giroscópio giram no mesmo plano, o que resulta na perda de um grau de liberdade do espaço tridimensional (Koch, 2016). A utilização Quaterniões permite ainda uma maior eficiência computacional e maior resistência a erros de arredondamento.

Um Quaternião (e.g. Kuipers, 1999) unitário, $q = q_0 + iq_1 + jq_2 + kq_3 = [q_0 \quad q_1 \quad q_2 \quad q_3]^T$ e $\|q\| = 1$, é expresso como um vetor de quatro dimensões, associado a uma componente real e três componentes imaginárias (i, j, k), pertencente a um sistema numérico que expande os números complexos. A Equação (1.7) permite a conversão de ângulos de Euler para Quaterniões.

[7] Biblioteca de edição-livre para processamento de imagem, calibração, entre outros (Disponível em http://opencv.org/)).

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \pm\left(cos\ \frac{\phi}{2} cos\ \frac{\theta}{2} cos\ \frac{\psi}{2} + sin\ \frac{\phi}{2} sin\ \frac{\theta}{2} sin\ \frac{\psi}{2}\right) \\ \pm\left(sin\ \frac{\phi}{2} cos\ \frac{\theta}{2} cos\ \frac{\psi}{2} - cos\ \frac{\phi}{2} sin\ \frac{\theta}{2} sin\ \frac{\psi}{2}\right) \\ \pm\left(\ cos\ \frac{\phi}{2} sin\ \frac{\theta}{2} cos\ \frac{\psi}{2} + sin\ \frac{\phi}{2} cos\ \frac{\theta}{2} sin\ \frac{\psi}{2}\right) \\ \pm\left(cos\ \frac{\phi}{2} cos\ \frac{\theta}{2} sin\ \frac{\psi}{2} - sin\ \frac{\phi}{2} sin\ \frac{\theta}{2} cos\ \frac{\psi}{2}\right) \end{bmatrix} \quad (1.7)$$

A matriz de rotação R pode ser escrita em função de q, de acordo com a Equação (1.8) (e.g. Kuipers, 1999).

$$R = \begin{bmatrix} 1 - 2(q_2^2 - q_3^2) & 2(q_1 q_2 + q_3 q_0) & 2(q_1 q_3 - q_2 q_0) \\ 2(q_1 q_2 + q_3 q_0) & 1 - 2(q_1^2 + q_3^2) & 2(q_2 q_3 + q_1 q_0) \\ 2(q_1 q_3 + q_2 q_0) & 2(q_2 q_3 + q_1 q_0) & 1 - 2(q_1^2 + q_2^2) \end{bmatrix} \quad (1.8)$$

A rotação de Rodrigues consiste na definição de um eixo de rotação arbitrário (u) e de um ângulo (α) que estabelece a magnitude da rotação, ao longo do eixo (u) definido. Permite uma representação intuitiva da rotação de um corpo rígido, e maior flexibilidade na computação de dados. A fórmula de rotação de Rodrigues é descrita pela Equação (1.9), em que u define o eixo de rotação, normalizado à sua magnitude ($\|u\| = 2\ sin\ \alpha$), e que α define o ângulo de rotação (e.g. Gallego, 2015).

$$R = I + [u]_\times sin\ \alpha + [u]_\times^2 (1 - cos\ \alpha) \quad (1.9)$$

O vetor u e o ângulo α podem ser obtidos a partir de uma matriz de rotação de acordo com as Equações (1.10) e (1.11), em que $[u]_\times$, expresso na forma de uma matriz antissimétrica, representa o produto externo de modo a que facilite a computação ($[u]_\times v = u \times v$, sendo v um vetor arbitrário) (e.g. Gallego, 2015).

$$u = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix} \Rightarrow [u]_\times = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix} \quad (1.10)$$

$$\alpha = arcos\left(\frac{1}{2}\ [R_{11} + R_{22} + R_{33} - 1]\right) \quad (1.11)$$

1.3.4. Deteção e Processamento da Marca Visual de Aterragem

A utilização de uma marca visual de aterragem implica a definição de um método de análise de imagem, que permita a identificação da marca (subcapítulo 1.3.4.1.). Consequentemente, a identificação da marca permite a criação de um modelo bi-dimensional (2D) da marca, descrito por um conjunto de pontos definido no plano da imagem. O modelo 2D da marca é essencial para a implementação da capacidade de aterragem de um VANT, quando associado a um modelo que permita efetuar a correspondência de pontos da marca visual no espaço tri-dimensional (3D) com pontos 2D, no plano da imagem obtido por uma câmara.

O modelo geométrico *Pinhole Camera* (subcapítulo 1.3.4.2.) permite descrever a relação entre o espaço 2D e 3D, permitindo a descrição geométrica 3D da marca visual detetada, e estabelecendo uma relação entre o referencial da câmara e o referencial da marca de aterragem. Esta relação é definida através de vetores rotação e translação, que permitem efetuar o cálculo da pose da câmara (subcapítulo 1.3.4.3.).

Nesta dissertação, análise da pose e dos referenciais de um VANT encontra-se diretamente relacionada com o comportamento da câmara acoplada ao VANT, na medida em que a pose calculada, através da marca visual de aterragem, corresponde à pose da câmara. Como tal, foi efetuada uma aproximação do referencial da câmara ao referencial do VANT, assumindo a câmara como a origem do referencial B.

1.3.4.1. Deteção de Imagem

A deteção da marca visual de aterragem implica que a imagem visualizada seja sujeita a um processamento que permita ao algoritmo identificar a marca visual de aterragem. No âmbito desta dissertação, são processadas imagens RGB de 8 bits (*Red*, *Green*, *Blue*), que permitem representar uma imagem através de pixéis com qualquer cor do espetro visível. Cada pixel é definido de acordo com a sua posição na resolução da imagem, e de acordo com a sua cor, obtida através da combinação de diferentes proporções de vermelho, verde e azul (Carreira, 2013). A Figura 20 apresenta a representação digital de uma imagem RGB, composta por diversos pixéis.

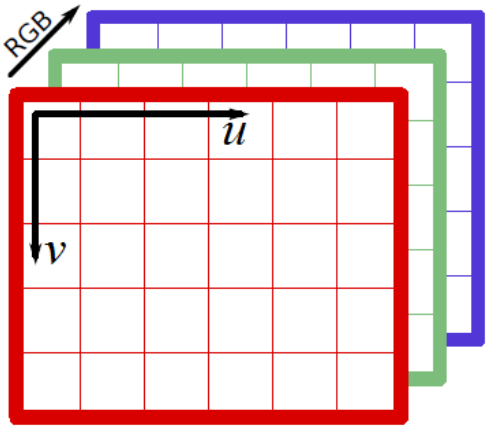

Figura 20 - Representação digital de uma imagem RGB (Carreira, 2013).

O processamento de imagens RGB, para a deteção da marca, baseia-se em destacar e localizar os principais contornos presentes na imagem com o objetivo de comparar os contornos existentes com os contornos de referência da marca visual, de modo a identificar a presença da marca visual na imagem. Para otimizar os resultados obtidos, este processo consiste:

- Numa conversão inicial da imagem visualizada, a cores, para uma versão a preto e branco, enaltecendo a perceção da intensidade da imagem;
- Aplicação de distorção *Gaussiana* para a eliminação de ruído de alta frequência;
- Aplicação de *Canny Edge Detection* para destacar os contornos da imagem;
- Deteção e quantização dos contornos da imagem.

A utilização de distorção *Gaussiana* (*Gaussian blur/smoothing*) em processamento de imagem consiste na convolução da imagem com uma função *Gaussiana*, que representa o ruído estatístico atribuído a uma imagem. A convolução resulta numa imagem (de aparência translúcida), sem ruído de alta frequência, o que se traduz na aplicação de um filtro passa-baixo (e.g. Eswar, 2015).

O destaque dos contornos de uma imagem é feito com base no algoritmo *Canny Edge Detection*, desenvolvido por John Canny (1986). O algoritmo é definido por quatro fases distintas: redução de ruído, mapeamento do gradiente de intensidade da imagem, deteção de potenciais candidatos a contornos existentes na direção do gradiente de intensidade de imagem e seleção de melhores candidatos a contornos com base em valores de referência máximo e mínimo. A aplicação do algoritmo *Canny Edge Detection* encontra-se exemplificado na Figura 21.

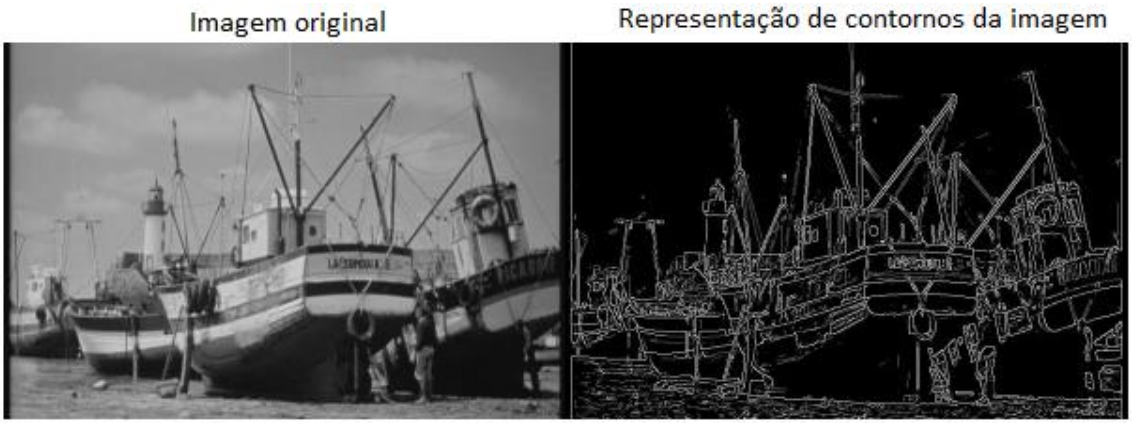

Figura 21 - Deteção de contornos através do algoritmo *Canny Edge Detection*.

Por fim, os contornos são detetados, separados e guardados individualmente em vetores distintos através do algoritmo proposto por Suzuki (1985), que se baseia no processamento de uma imagem binária através da técnica de seguimento de fronteiras, que permite identificar e distinguir pixéis que contêm contornos (com o valor binário 1 atribuído) dos restantes pixéis (com valor binário 0 atribuído).

1.3.4.2. Modelo *Pinhole Camera*

O modelo geométrico *Pinhole Camera* baseia-se numa caixa fechada, com um pequeno orifício (*pinhole*), que permite a passagem de raios de luz refletidos ou emitidos por um objeto, criando uma imagem (2D) invertida do objeto no plano da imagem (parte traseira da caixa) (e.g. Forsyth, 2002) (Figura 22).

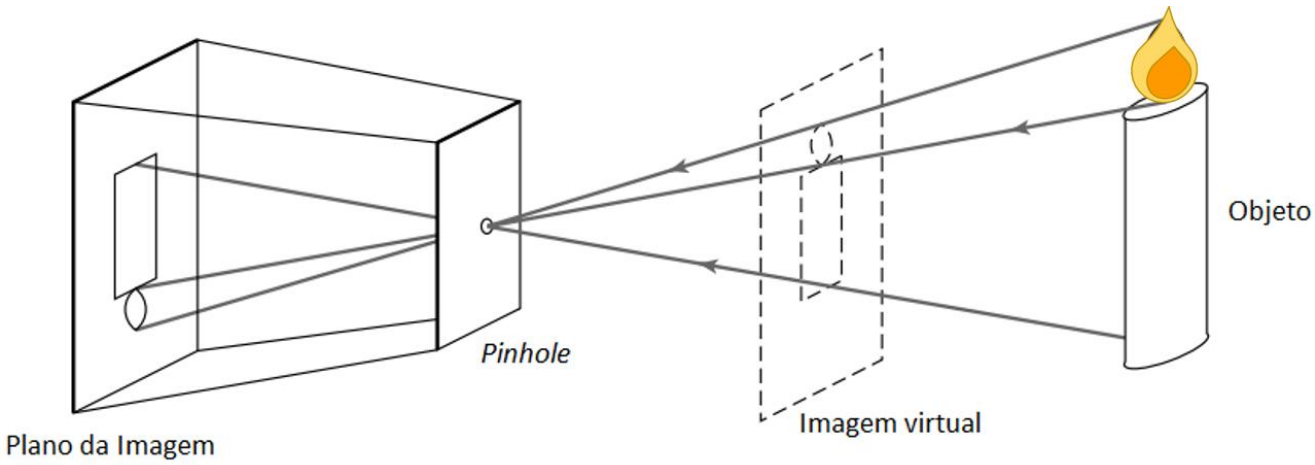

Figura 22 - Modelo *Pinhole Camera* (e.g. Forsyth, 2002).

Para efeitos de simplificação da implementação e análise do modelo, assume-se que a imagem é projetada no plano de imagem virtual (à frente do *pinhole*). Neste modelo adaptado (Figura 23), o plano da imagem (coincidente com o plano da imagem virtual) é criado ao longo do eixo z, designado por "eixo ótico", em que o ponto de interseção entre o eixo ótico e o plano da imagem é definido por "ponto principal". A distância focal (f) é definida como a distância entre o ponto principal, no plano da imagem, e o centro ótico (ou centro da câmara). De acordo com a Figura 23, um ponto no mundo real, representado por $W=[x_G, y_G, z_G]^T$, é representado no plano da imagem na posição $i=[u, v]^T$, definida através de um raio luminoso que intersecta simultaneamente o objeto, o plano da imagem e o centro ótico (e.g. Forsyth, 2002) (e.g. Prince, 2012) (e.g. Santos, 2014).

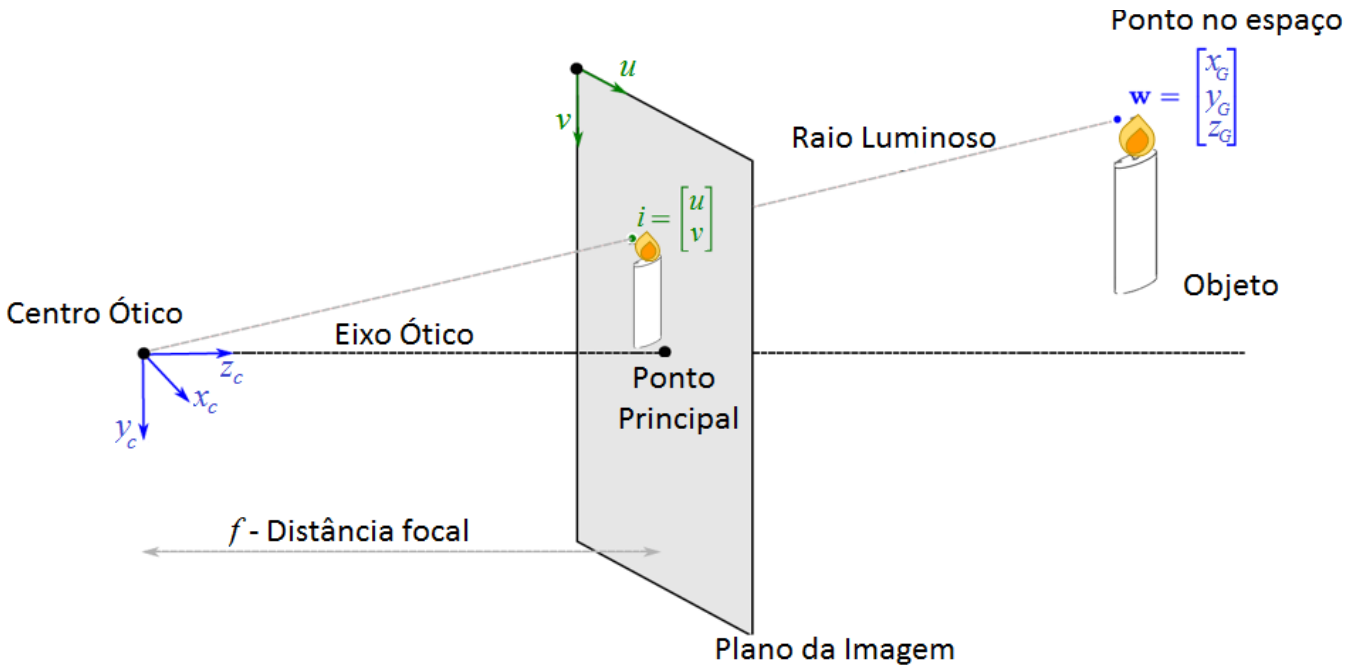

Figura 23 - Modelo *Pinhole Camera* adaptado (e.g. Prince, 2012).

À semelhança da relação estabelecida entre os referenciais do VANT (referencial B) e de um ponto à superfície da Terra (referencial G) (subcapítulo 1.3.3.2.), é possível estabelecer uma relação idêntica entre o referencial da câmara (será definido como referencial B, com base na aproximação efetuada entre a câmara e o VANT) e o referencial centrado num objeto no mundo 3D (referencial G). Com base na Equação (1.6), a Equação (1.12) descreve esta relação que permite a representação de um ponto definido no referencial G (Mundo) para um ponto definido no referencial B (câmara), através da matriz de rotação (R^B) e vetor de translação (Γ^B) que descrevem a orientação de um corpo, e a mudança entre a posição do centro de coordenadas do Mundo e o centro de coordenadas da câmara, respetivamente (e.g. Forsyth, 2002) (e.g. Prince, 2012) (e.g. Santos, 2014).

$$\begin{bmatrix} p^B \\ 1 \end{bmatrix} = {}^B M_G \begin{bmatrix} p^G \\ 1 \end{bmatrix} = \begin{bmatrix} R^B & \Gamma^B \\ 0_{1x3} & 1 \end{bmatrix} . \begin{bmatrix} p^G \\ 1 \end{bmatrix} \quad (1.12)$$

A relação entre um ponto $p_B = [x_B, y_B, z_B]^T$, definido no referencial da câmara (referencial B), e a sua projeção no plano da imagem (tipicamente representada em

pixéis) é dada pela Equação (1.13) (e.g. Forsyth, 2002) (e.g. Prince, 2012) (e.g. Santos, 2014).

$$i = \begin{bmatrix} u \\ v \end{bmatrix} = \frac{f}{z_B} \begin{bmatrix} x_B \\ y_B \end{bmatrix} \tag{1.13}$$

Relacionando a Equação (1.12) com a Equação (1.13), é possível representar o modelo geométrico de uma câmara ideal, expresso em coordenadas homogéneas (Equações (1.14) e (1.15)) (e.g. Forsyth, 2002) (e.g. Prince, 2012) (e.g. Santos, 2014).

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = Kp^B = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} \tag{1.14}$$

$$\Updownarrow$$

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} {}^B M_G \begin{bmatrix} p^G \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R^B & \Gamma^B \\ 0_{1x3} & 1 \end{bmatrix} \begin{bmatrix} x_G \\ y_G \\ z_G \\ 1 \end{bmatrix} \tag{1.15}$$

Este modelo que relaciona pontos 3D com 2D, é representado através do parâmetro $\lambda = z_B$, que representa um fator de escala homogéneo, e das matrizes K e ${}^B M_G$ que correspondem, respetivamente, aos parâmetros intrínsecos e extrínsecos da câmara (Figura 24).

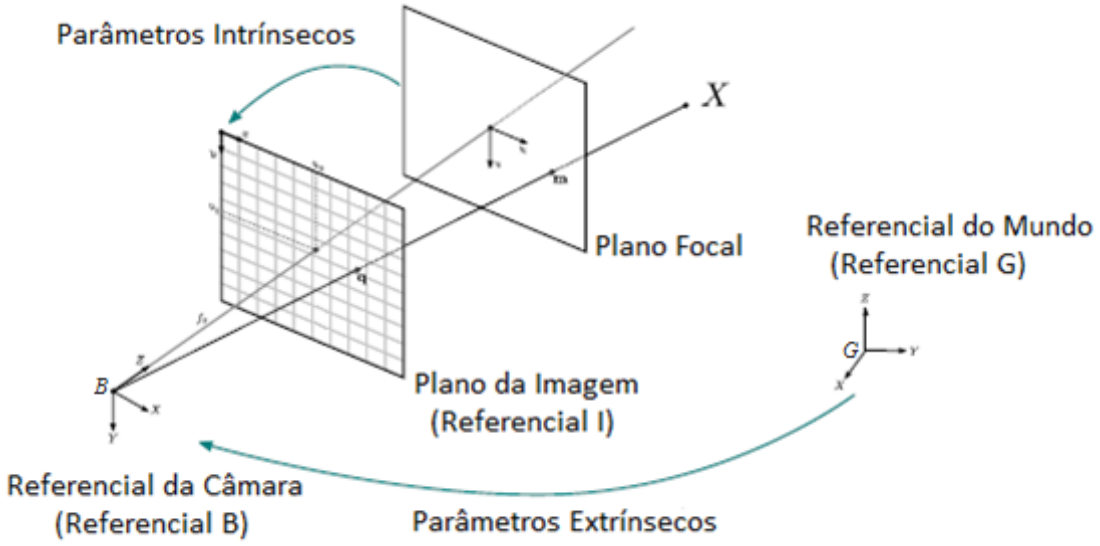

Figura 24 - Representação do modelo Pinhole Camera, e dos respetivos parâmetros intrínsecos e extrínsecos da câmara (Hartley, 2004).

Os parâmetros intrínsecos da câmara representam a componente ótica da câmara que, como representado pelo modelo *Pinhole Camera*, descreve a relação do

plano do *pinhole* com o plano da imagem (Hartley, 2004). O modelo representado pela Equação (1.15) é considerado ideal, como tal assume que a origem do referencial da imagem (referencial I: (u, v) em pixéis) corresponde ao ponto principal do plano da imagem. Uma vez que a origem do referencial da imagem se encontra convencionado como o canto superior esquerdo da imagem, é necessário representar uma correção dada por (c_x, c_y), que representa as coordenadas do ponto principal, e que se reflete na matriz de parâmetros intrínsecos da câmara (Equação (1.16)) (e.g. Forsyth, 2002) (e.g. Prince, 2012) (e.g. Santos, 2014).

$$K = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.16}$$

Outra característica do modelo ideal (Equação (1.15)) advêm da aproximação do formato de cada pixel, de uma imagem, ao formato de um quadrado de dimensões unitárias, algo que não corresponde à realidade. Esta aproximação pode ser corrigida através da adição dos parâmetros (η_x, η_y) que representam o número de pixéis por unidade de distância, no caso em que as coordenadas da imagem são medidas em pixéis (Equação (1.17)) (e.g. Forsyth, 2002) (e.g. Prince, 2012) (e.g. Santos, 2014).

$$K = \begin{bmatrix} f\eta_x & 0 & c_x & 0 \\ 0 & f\eta_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.17}$$

Outro fator que contribui para a representação da matriz de parâmetros intrínsecos da câmara é o ângulo de *skew* (S_θ), que descreve a inclinação da imagem. É um valor praticamente nulo, sendo desprezado na maioria dos casos.

Considerando todos os parâmetros intrínsecos da câmara $(f, c_x, c_y, \eta_x, \eta_y, S_\theta)$ é possível obter uma matriz K (Equação (1.18)) que corresponde a um modelo aproximado à realidade (Equação (1.19)) (e.g. Forsyth, 2002) (e.g. Prince, 2012) (e.g. Santos, 2014).

$$K = \begin{bmatrix} f\eta_x & S_\theta & c_x & 0 \\ 0 & f\eta_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1.18}$$

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f\eta_x & S_\theta & c_x & 0 \\ 0 & f\eta_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R^B & \Gamma^B \\ 0_{1x3} & 1 \end{bmatrix} \begin{bmatrix} x_G \\ y_G \\ z_G \\ 1 \end{bmatrix} \quad (1.19)$$

A matriz de parâmetros extrínsecos ${}^B M_G$ permite a transformação (orientação e posição) entre pontos definidos no referencial do Mundo (referencial G) e pontos definidos no referencial da câmara (referencial B) (Equação 1.12). Em que Γ^B representa a posição da origem do referencial do Mundo, expressa em coordenadas relativas ao referencial da câmara, e R^B representa a rotação que a câmara induz um pontos representados no referencial do mundo (Equação (1.20)) (e.g. Forsyth, 2002) (e.g. Prince, 2012) (e.g. Santos, 2014).

$$\begin{bmatrix} p^B \\ 1 \end{bmatrix} = \begin{bmatrix} R^B & \Gamma^B \\ 0_{1x3} & 1 \end{bmatrix} . \begin{bmatrix} p^G \\ 1 \end{bmatrix} \Leftrightarrow p^B = R^B p^G + \Gamma^B \quad (1.20)$$

A pose da câmara (R^G, Γ^G) (Equações (1.21) e (1.22)), representada em relação ao referencial do mundo, pode ser obtida através dos parâmetros extrínsecos da câmara, de acordo com a natureza de uma matriz de rotação (1.3.3.2.) expressa na Equação (1.21) (e.g. Forsyth, 2002) (e.g. Prince, 2012) (e.g. Santos, 2014).

$$R^{B^T} = R^{B^{-1}} = R^G \quad (1.21)$$

$$-R^{B^{-1}} \Gamma^B = \Gamma^G \quad (1.22)$$

Os parâmetros extrínsecos da câmara permitem ainda o cálculo de pontos definidos no referencial do Mundo através da relação representada na Equação (1.23) (e.g. Forsyth, 2002) (e.g. Prince, 2012) (e.g. Santos, 2014).

$$p^G = R^{B^{-1}} p^B - R^{B^{-1}} \Gamma^B = R^G p^B + \Gamma^G \quad (1.23)$$

O cálculo dos parâmetros extrínsecos, e consequentemente, da pose da câmara encontra-se associado à resolução do problema *Perspective-n-Point*. Este problema, e o respetivo método de resolução, encontram-se descritos no subcapítulo (1.3.4.3.) seguinte.

1.3.4.3. Cálculo da pose

O cálculo da pose da câmara (por inerência, do VANT) está associado à resolução da expressão definida na Equação (1.15) (subcapítulo 1.3.4.2.), que estabelece uma relação entre pontos 3D e a sua correspondência no plano 2D da imagem. A seleção de vários (*n*) pontos 3D para a resolução do problema (expresso na Equação (1.15)) representa o problema *Perspective-n-Point* (PnP). Através da resolução do problema PnP, é possível determinar a posição e orientação de um dado conjunto de *n* correspondências entre pontos 3D e as suas respetivas projeções 2D (Lepetit, 2009) (Prince, 2012).

Dos diversos métodos existentes para a resolução do problema PnP, o método implementado nesta dissertação baseia-se no *Efficient Perspective-n-Point* (EPnP), desenvolvido por Lepetit (2009) (Figura 25).

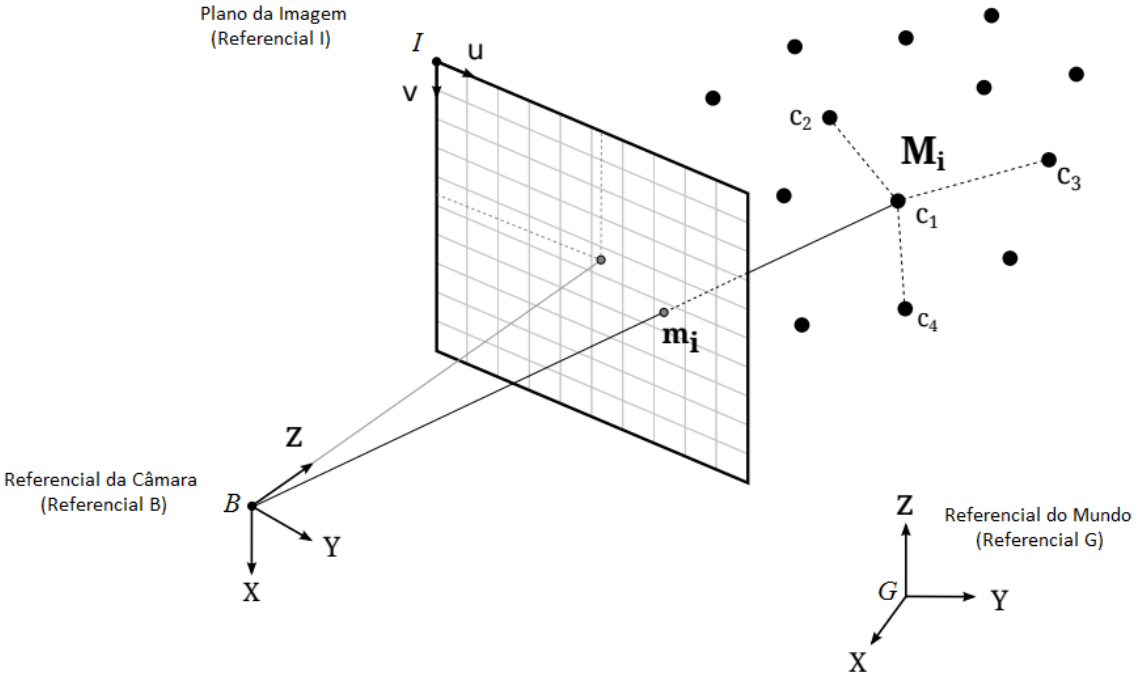

Figura 25 - Formulação do problema EPnP: Dado um conjunto de pontos 3D (M_i) representados no referencial do Mundo, e as suas respetivas projeções 2D (m_i) no plano da imagem, os pontos c_{1-4} formam a base que representa o conjunto através de uma combinação linear, de modo a obter a pose da câmara (Pi, 2015).

O algoritmo EPnP permite o cálculo eficiente da pose de uma câmara, através do conhecimento prévio dos parâmetros intrínsecos da câmara, dos n pontos definidos no espaço 3D e expressos no referencial do Mundo (referencial G), e das respetivas n correspondências no plano 2D da imagem (referencial I). Em que os pontos 3D, definidos no referencial do Mundo, são representados através da soma ponderada de quatro pontos não-complanares definidos como "pontos de controlo" (Equação (1.24)) (Lepetit, 2009).

$$p_i^G = \sum_{j=1}^{4} \alpha_{ij} c_j^G \tag{1.24}$$

Em que $p_i^G = [x_G, y_G, z_G]^T$ corresponde a um ponto 3D definido no referencial do Mundo, α_{ij} corresponde a um conjunto de coordenadas baricêntricas homogéneas e $c_j^G = [X_G, Y_G, Z_G]^T$ corresponde a um ponto de controlo 3D definido no referencial do Mundo. Aplicando esta igualdade (Equação (1.24)) ao referencial da câmara, obtêm-se quatro pontos de controlo c_j^B que representam a incógnita do problema, perfazendo um total de doze incógnitas (Lepetit, 2009).

Relacionando a Equação (1.24) com a Equação (1.14), obtêm-se a seguinte relação (Equação (1.25))[8] :

$$\lambda_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = K p_i^B = K \sum_{j=1}^{4} \alpha_{ij} c_j^B = \begin{bmatrix} f_x & 0 & u_c \\ 0 & f_y & v_c \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^{4} \alpha_{ij} \begin{bmatrix} x_j^B \\ y_j^B \\ z_j^B \end{bmatrix} \tag{1.25}$$

Em que λ_i representa um conjunto de fatores de escala, (u_i, v_i) representam coordenadas de $n = i$ pontos 2D no plano da imagem, e K representa a matriz de parâmetros intrínsecos da câmara (assumindo o modelo ideal). Através da igualdade $\lambda_i = \sum_{j=1}^{4} \alpha_{ij} z_i^B$, definida na última linha da Equação (1.25), torna-se possível obter duas

[8] Por simplificação, os parâmetros intrínsecos da câmara (c_x, c_y) serão representados por (u_c, v_c).

equações ((1.26) e (1.27)) linearmente independentes, para cada n ponto(s) de referência (Lepetit, 2009).

$$\sum_{j=1}^{4} \alpha_{ij} f_x x_j^B + \alpha_{ij}(u_c - u_i) z_j^B = 0 \tag{1.26}$$

$$\sum_{j=1}^{4} \alpha_{ij} f_y y_j^B + \alpha_{ij}(v_c - v_i) z_j^B = 0 \tag{1.27}$$

Ao concatenar estas equações, para todos os n pontos de referência, é possível obter um sistema linear homogéneo do tipo $\mathrm{M}\mathrm{x} = 0$, em que M representa uma matriz $2n \times 12$ composta pelo coeficientes conhecidos das Equações (1.26) e (1.27), $\mathrm{x} = [c_1^B, c_2^B, c_3^B, c_4^B]^T$ representa um vetor com doze incógnitas (cada c_j^B possui três coordenadas). A solução para este sistema é representada pela Equação (1.28) (Lepetit, 2009):

$$\mathrm{x} = [c_1^B, c_2^B, c_3^B, c_4^B]^T = \sum_{i=1}^{N} \beta_i \mathrm{v}_i \tag{1.28}$$

Em que v_i são os vetores próprios da matriz M, correspondentes aos N valores próprios nulos de M. Sendo obtidos através do cálculo dos vetores próprios nulos da matriz $\mathrm{M}^T\mathrm{M}$ de dimensões 12×12. O cálculo dos valores β_i é feito de acordo com aproximação representada na Equação (1.29), que define que a distância entre dois pontos de controlo, representados no referencial do Mundo, é igual à distância entre dois pontos de controlo representados no referencial da câmara (Lepetit, 2009) .

$$\left\| c_i^B - c_j^B \right\|^2 = \left\| c_i^G - c_j^G \right\|^2 \tag{1.29}$$

Esta aproximação permite definir a seguinte igualdade, cuja representação é referente ao caso de $N = 1$ (Equação (1.30)) (Lepetit, 2009):

$$\left\| \beta \mathrm{v}^{[i]} - \beta v^{[j]} \right\|^2 = \left\| c_i^G - c_j^G \right\|^2 \tag{1.30}$$

O que permite o cálculo do parâmetro β (para $N = 1$), de acordo com a Equação (1.31) (Lepetit, 2009):

$$\beta = \frac{\sum_{\{i,j\} \in [1;4]} \left\| \mathrm{v}^{[i]} - v^{[j]} \right\| . \left\| c_i^G - c_j^G \right\|}{\sum_{\{i,j\} \in [1;4]} \left\| \mathrm{v}^{[i]} - v^{[j]} \right\|^2} \tag{1.31}$$

Sabendo o valor de β, para cada ponto de referência, é possível obter os pontos de controlo $[c_1^B, c_2^B, c_3^B, c_4^B]^T$, e consequentemente é possível obter a matriz de parâmetros extrínsecos através da relação entre as Equações (1.15) e (1.25). A partir dos parâmetros extrínsecos da câmara, é possível obter a pose da câmara (posição e orientação em relação ao referencial do Mundo), como demonstrado nas Equações (1.21) e (1.22) (Lepetit, 2009).

Capítulo 2. Metodologia

A escolha da metodologia de investigação a aplicar na elaboração da dissertação define o carácter e emprego do trabalho a efetuar, representando um conjunto de etapas essenciais para o alcance dos objetivos. Esta escolha foi essencialmente baseada no principal objetivo deste trabalho: o desenvolvimento de interoperabilidade entre VNTs através da criação de um controlador externo que amplifica e expande as capacidades de um VANT, com recurso a uma norma de referência (ROS).

Deste modo, a metodologia de investigação aplicada à realização da dissertação consiste na metodologia *Design Science Research*. Peffers (2007), define que a aplicação da metodologia *Design Science Research* é focada num processo de investigação e de pesquisa, utilizado para a criação e teste de artefactos. Sendo adaptado à resolução de problemas de carácter prático.

A metodologia *Design Science Research* é composta seis etapas distintas (Peffers, 2007):

1. Identificação do problema e Motivação;
2. Objetivo da solução;
3. Design e Desenvolvimento;
4. Demonstração;
5. Avaliação;
6. Comunicação;

A seis etapas encontram-se complementadas por um processo iterativo de otimização e reajuste, baseado nos resultados obtidos.

A Figura 26 representa a metodologia de investigação aplicada à realização da dissertação, com base na metodologia *Design Science Research*.

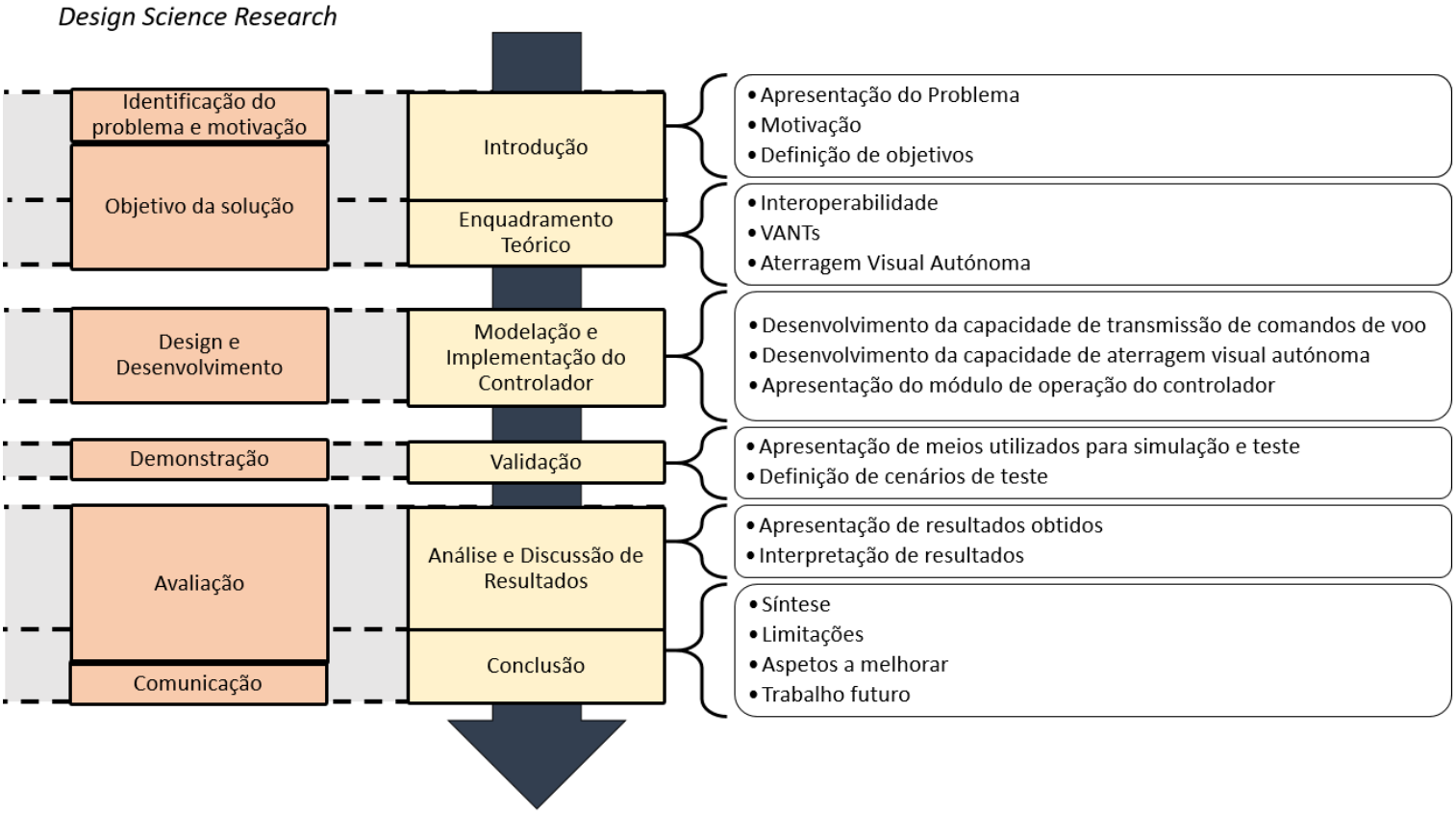

Figura 26 - Representação da metodologia Design Science Research aplicada à dissertação.

Cada etapa da metodologia *Design Science Research* pode ser relacionada com a presente dissertação através das seguintes correspondências (Peffers, 2007):

Identificação do problema e motivação – Inicialmente, é necessário enquadrar e definir o problema, enaltecendo a sua importância e pertinência. Este processo implica a introdução de estudos que abordem a temática da tese, e que apresentem implementações e soluções para o problema em estudo. Como tal, esta etapa enquadra-se no capítulo inicial, "Introdução", que introduz o conceito de interoperabilidade de sistemas aplicado a VANTs e o sistema desenvolvido que promove interoperabilidade entre VNTs, e que define o âmbito e pertinência do tema.

Objetivo da solução – Após a definição do problema, e da respetiva motivação, torna-se necessário estabelecer objetivos para o desenvolvimento do trabalho, o que implica a definição de propostas de solução do problema em estudo, baseadas em

conceitos teóricos. Através dos capítulos, "Introdução" e "Enquadramento Teórico", são propostos objetivos, o método de investigação a adotar e são introduzidos conceitos relacionados com interoperabilidade, normas de referência (*standards*), VNTs e sistemas de aterragem visual autónoma, que se apresentam como fundamento teórico essencial para a modelação e implementação do controlador.

Design e Desenvolvimento – Esta fase baseia-se no desenvolvimento do artefacto utilizado para a resolução do problema em estudo, apresentando características, funcionalidades e arquiteturas. No contexto da dissertação, enquadra-se no capítulo "Modelação e Implementação do Controlador", que apresenta a arquitetura do sistema desenvolvido, e detalha a composição de cada módulo (Comando e Controlo, Comunicação, e Aterragem Visual Autónoma) de funcionamento do controlador.

Demonstração – Após a criação do artefacto, a etapa de "Demonstração" apresenta métodos experimentais (ou simulados) para a solução do problema, através da utilização do artefacto. Consiste, essencialmente, no capítulo "Validação", que apresenta o material utilizado e os cenários de validação dos diferentes módulos do controlador.

Avaliação – Nesta fase, o investigador deve observar e medir o comportamento do artefacto no âmbito da solução do problema em estudo, comparando os resultados expectáveis, e desejáveis, com os resultados obtidos através de experimentação. De acordo com os resultados obtidos, e caso estes não correspondam com os resultados desejáveis, deve ser retomada a metodologia de investigação a partir da etapa de "Design e Desenvolvimento", de modo a corrigir o artefacto, e dando início a um processo iterativo de otimização do produto final. Através do capítulo "Análise e Discussão de Resultados" é possível aplicar esta etapa da metodologia *Design Science Research*, através da representação e interpretação dos resultados obtidos dos testes aos módulos de comando e controlo, e de aterragem visual autónoma.

Comunicação – Por fim, a etapa de “Comunicação”, que se baseia na divulgação do problema estudado, em particular, a sua importância, método de condução da pesquisa, e a eficácia da solução obtida para o problema. A realização desta etapa encontra-se diretamente relacionada com a publicação do trabalho desenvolvido em artigos científicos. No caso desta dissertação, o capítulo “Conclusão” resume o trabalho efetuado, incidindo na relevância da solução obtida, nas limitações existentes, nos aspetos a melhorar, e numa proposta de trabalho futuro. A realização da dissertação contou com a publicação de artigos científicos, que apesar de terem sido publicados numa intermédia do desenvolvimento da dissertação, contribuíram para a divulgação do trabalho realizado.

Capítulo 3. Modelação e Implementação do Controlador

3.1. Introdução

Este capítulo descreve o processo de conceptualização e criação do controlador externo, começando por introduzir o controlador na arquitetura de um sistema aéreo não tripulado (subcapítulos 3.2. e 3.3.1.). A descrição do desenvolvimento do controlador procede de acordo com a descrição dos módulos funcionais do controlador: módulo de comando e controlo (subcapítulos 3.3.2.), módulo de comunicação (subcapítulo 3.3.3.) e módulo de aterragem visual autónoma (subcapítulo 3.3.4.). O capítulo é concluído com a descrição da operação do controlador através de um interface gráfico, e das suas respetivas funcionalidades (subcapítulo 3.3.5.). O código criado para desenvolver o controlador externo encontra-se representado nos apêndices A, B, C, D e E.

3.2. Arquitetura do Sistema

A arquitetura do sistema, proposta nesta dissertação, é definida à imagem de um típico sistema de um VANT (subcapítulo 1.2.3.), sendo composta por uma GCS, uma ligação de dados (*datalink*) e pelo veículo. A inovação reside na adição de um controlador a bordo do VANT, que estabelece comunicações com a GCS, permitindo o comando e controlo do VANT através do controlador. A arquitetura do sistema encontra-se representada na Figura 27.

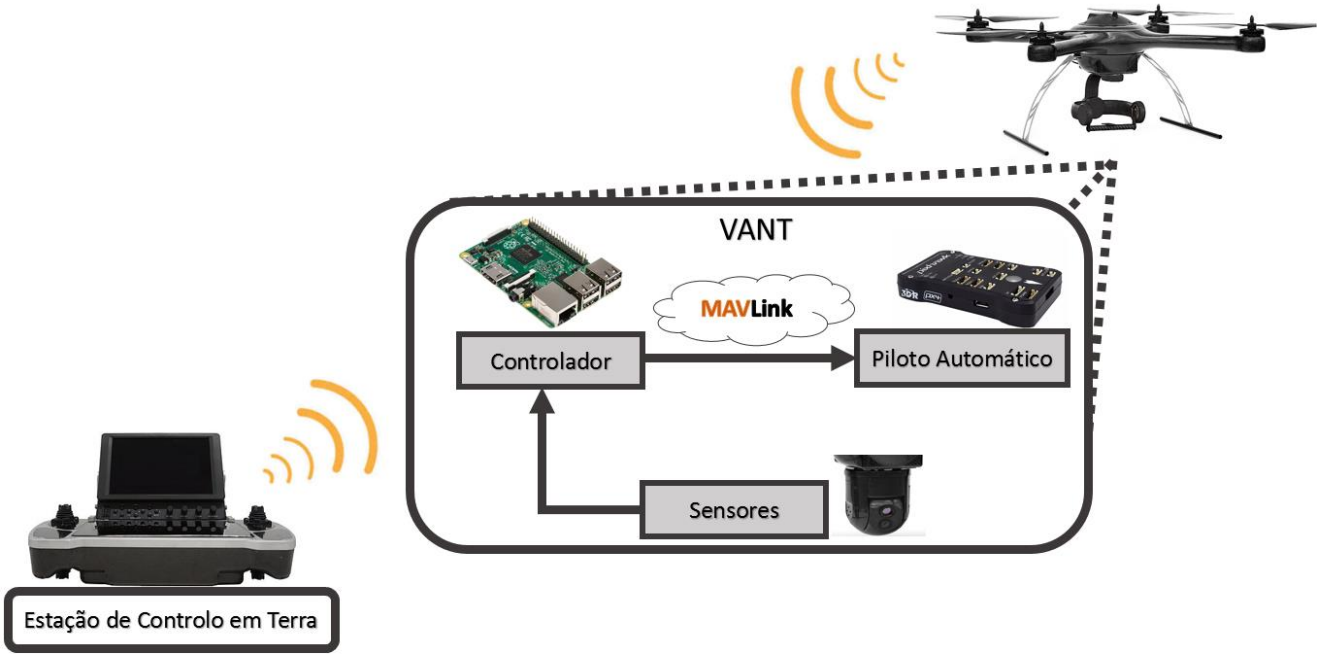

Figura 27 - Arquitetura do sistema.

O comando e controlo exercido pelo controlador traduz-se no processamento e encaminhamento de comandos de voo para o piloto automático, e na execução de funcionalidades adicionais, como a aterragem visual autónoma. A transmissão de comandos de voo, por parte do controlador, para o piloto automático é feita de acordo com o protocolo de comunicações MAVLink.

Para a execução da aterragem visual autónoma, o controlador recebe dados provenientes do sensor visual do veículo (e.g. câmara de vídeo) que são posteriormente processados para efetuar a localização da posição do ponto de aterragem, e para enviar para o piloto automático comandos de voo que permitam a aterragem do veículo, no ponto designado.

3.3. Controlador

3.3.1. Arquitetura

O controlador foi desenvolvido no sistema operativo *Ubuntu 16.04 LTS*, utilizando o ROS *Kinetic*, o que permitiu o acesso a pacotes com funções previamente desenvolvidas, promovendo a reutilização de funções desenvolvidas pela comunidade ROS. Os nós do controlador foram criados com base na linguagem de programação *Python*.

O desenvolvimento do controlador foi dividido em duas fases: fase inicial de implementação da capacidade de transmissão de comandos de voo através do controlador, fase secundária de implementação da capacidade de aterragem visual autónoma. A Figura 28 representa a visão conceptual do controlador, destacando o fluxo de informação principal associado à fase inicial de implementação, e o fluxo de informação secundário associado à fase de implementação a capacidade de aterragem visual autónoma.

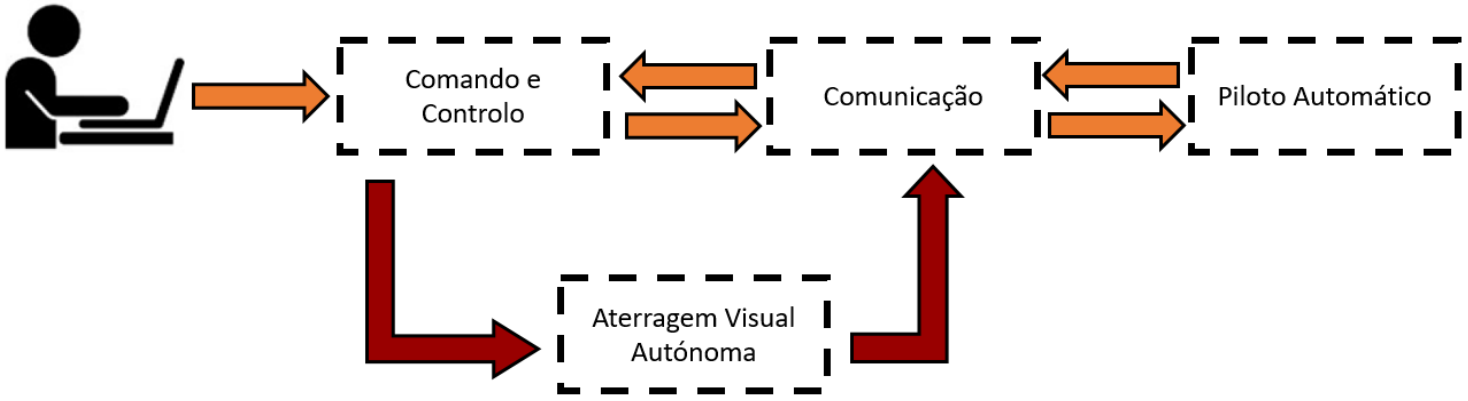

Figura 28 - Conceptualização do controlador.

A conceptualização do controlador é concretizada através da criação de nós ROS que desempenham funções distintas e que trocam mensagens, através de tópicos, de modo a executar tarefas definidas pelo operador. Estas interações encontram-se representadas na Figura 29, que define a arquitetura do controlador, integrado no sistema do VANT.

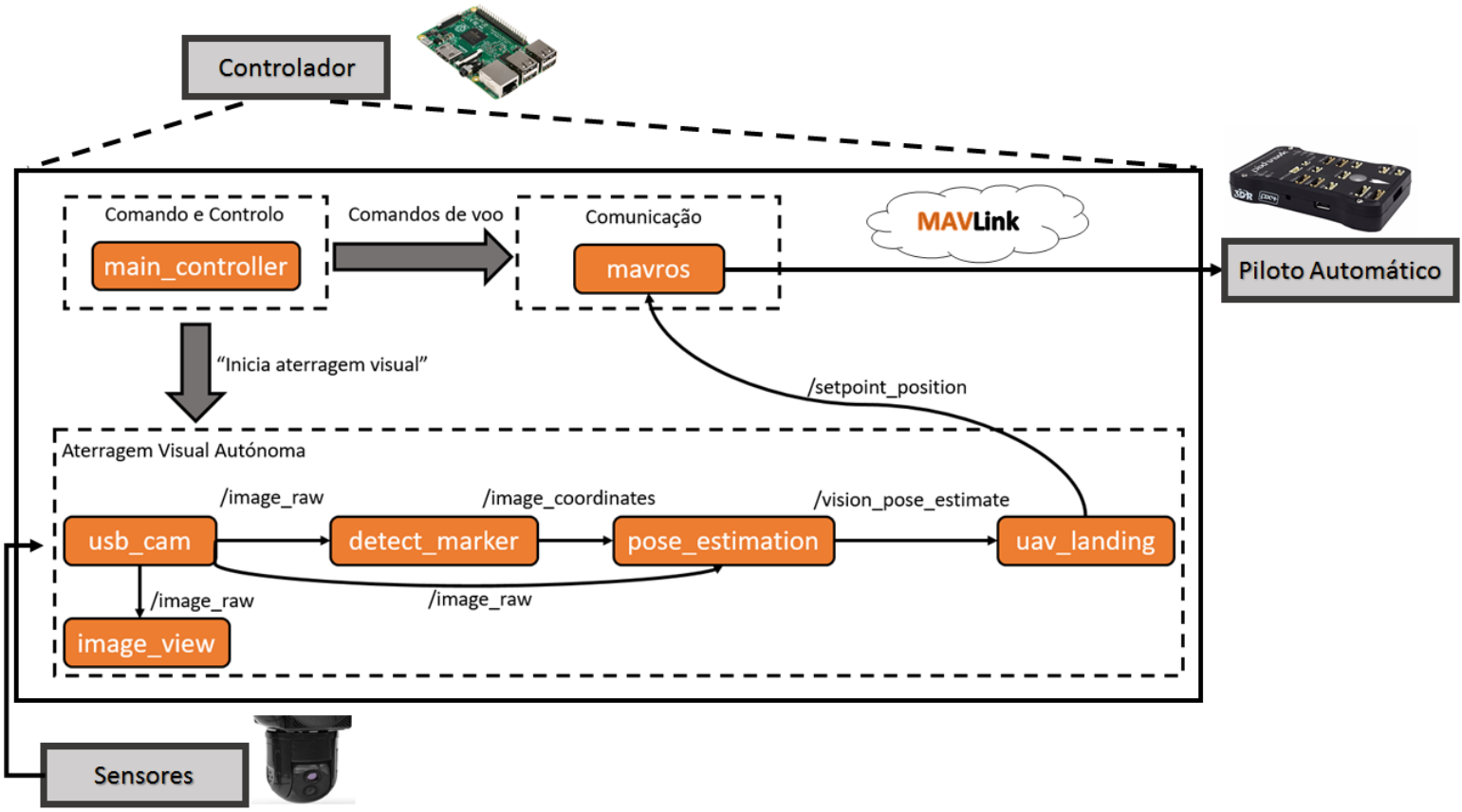

Figura 29 - Arquitetura do controlador, integrado no sistema do VANT.

Os módulos de Comando e Controlo, Comunicação e Aterragem Visual Autónoma, são compostos pelos seguintes nós ROS:

- ***usb_cam***: disponibiliza imagens proveniente de uma câmara de vídeo;
- ***image_view***: permite a visualização de imagens provenientes de uma câmara de vídeo;

- ***detect_marker***: deteta e analisa contornos de objetos presentes nas imagens provenientes do nó *usb_cam*, de modo a detetar a marca visual de aterragem, e a calcular as suas respetivas coordenadas no plano da imagem, expressas em pixéis;

- ***pose_estimation***: estima a pose do VANT através das coordenadas calculadas pelo nó *detect_marker*, bem como a posição do VANT em relação ao referencial com origem no ponto de aterragem;

- ***uav_landing***: executa o alinhamento do VANT com a marca visual de aterragem, e subsequente aterragem no centro da marca visual;

- ***mavros***: executa a função de intermediário de comunicação, ao gerar e transmitir mensagens MAVLink¸ com informação proveniente de outros nós, para o piloto automático, e vice-versa;

- ***main_controller***: permite ao operador a transmissão de comandos de voo para o piloto automático, e a inicialização da aterragem visual autónoma;

3.3.2. Módulo de Comando e Controlo

O comando e controlo, do controlador, é exercido através do nó *main_controller*. Permite a escolha do modo de voo (GUIDED, AUTO, MANUAL, STABILIZE, TEST), armar e desarmar o VANT, aterrar e descolar o VANT, e ler e atribuir ao VANT listas com *waypoints* (Figura 30).

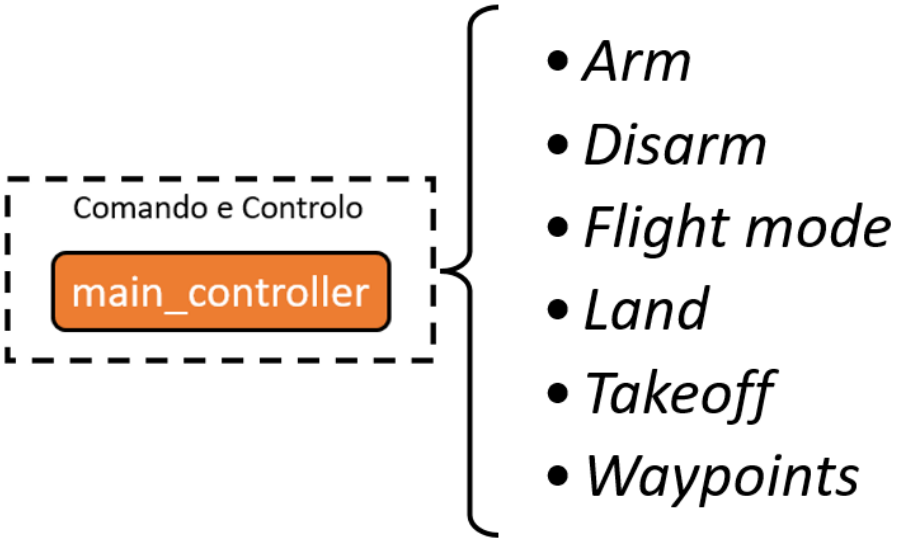

Figura 30 - Funcionalidades do *main_controller*.

A atribuição de comandos de voo torna-se possível através da colaboração entre o nó *main_controller* e o pacote *mavros* (subcapítulo 3.3.3). Os comandos para armar e desarmar o VANT (*arm* e *disarm*), para escolher o modo de voo (*Flight mode*), e para descolar e aterrar (*takeoff* e *land*) o VANT, são executados com recurso a serviços (*/mavros/cmd/arming, /mavros/set_mode*, */mavros/cmd/takeoff*, */mavros/cmd/land*) do pacote *mavros*. Este processa o pedido, convertendo-o numa mensagem MAVLink (COMMAND_LONG), com um parâmetro de identificação associado, que é posteriormente encaminhada para o piloto automático. A introdução de *waypoints* no piloto automático é possível através de uma utilidade do pacote *mavros*, *mavwp load*, que através de uma mensagem MAVLink (MISSION_ITEM) atribui ao piloto automático um conjunto de *waypoints*, definidos pelo operador, presentes num ficheiro de texto. A visualização e eliminação dos *waypoints* inseridos no piloto automático é possível

através da utilidade *mavwp load*, e do serviço */mavros/mission/clear*, respetivamente, que se encontram associados a mensagens MAVLink (MISSION_REQUEST e MISSION_CLEAR_ALL).

A Tabela 2 sintetiza a informação relativa à atribuição de comandos de voo, definindo o serviço, utilidade, mensagem MAVLink, e parâmetro associado a cada função.

Tabela 2 – Comandos de voo executados pelo nó *main_controller*.

main_controller					
Comandos de voo	**Serviços**		**Utilidades**	**Mensagem *MAVLink***	**Parâmetro**
Arm	*/mavros/cmd/arming*		N/A	COMMAND_LONG	MAV_MODE_FLAG_SAFETY_ARMED
Disarm	*/mavros/cmd/arming*		N/A	COMMAND_LONG	MAV_CMD_COMPONENT_ARM_DISARM
Flight mode	*/mavros/set_mode*		N/A	COMMAND_LONG	MAV_MODE_FLAG_[GUIDED/AUTO/MANUAL /STABILIZE/TEST]_ENABLED
Takeoff	*/mavros/cmd/takeoff*		N/A	COMMAND_LONG	MAV_CMD_NAV_TAKEOFF
Land	*/mavros/cmd/land*		N/A	COMMAND_LONG	MAV_CMD_NAV_LAND
Waypoints	*show*	N/A	*mwp show*	MISSION_REQUEST	N/A
	clear	*/mavros/mission/ clear*	N/A	MISSION_CLEAR_ALL	N/A
	load	N/A	*mavwp load*	MISSION_ITEM	N/A

3.3.3. Módulo de Comunicação

O módulo de comunicação do controlador recebe e processa os comandos de voo, transmitidos pelo módulo de Comando e Controlo, e encaminha os comandos recebidos no formato de mensagens MAVLink, para o piloto automático, que consequentemente procede à execução.

Por forma a realizar as funções do módulo de comunicação, foi implementado o pacote *mavros* (Ermakov, 2014), que permite estabelecer uma ligação entre a informação processada pelo ROS e o protocolo de comunicação MAVLink. O pacote *mavros* permite a interação de qualquer tipo de *software*, desenvolvido em ROS, com qualquer tipo de VNT utilize MAVLink como protocolo de comunicação, via ligação série.

Permite também estabelecer ligações com GCSs, uma vez que o nó principal do pacote *mavros* disponibiliza a troca de informação com utilizadores, que se ligam ao nó principal através de ligações *Transmission Control Protocol*[9] (TCP) ou *User Datagram Protocol*[10] (UDP), que pretendam receber e enviar dados através do *mavros*.

O pacote *mavros* consiste em três componentes distintas: *mavconn*, *nodelib*, *Plugin lib*. A componente *mavconn* é definida pela camada que envia e recebe mensagens MAVLink, estabelecendo trocas de informação internas e externas ao *mavros*. O núcleo do *mavros* é representado pela camada *nodelib*, que executa a funções principais do *mavros*. A camada *plugin lib* apresenta-se como uma extensão às capacidades do *mavros*, disponibilizando uma biblioteca de funções adicionais.

[9] *Transmission Control Protocol* – Protocolo de comunicação, fiável e orientado à ligação, utilizado para o envio pacotes de dados através da Internet. POSTEL, J. (1981). Transmission Control Protocol - DARPA Internet Program Protocol Specification. Information Sciences Institute University of Southern California.Marina del Rey.

[10] *User Datagram Protocol*- Protocolo de comunicação, pouco fiável e não-orientado à ligação, que permite enviar mensagens curtas (datagramas) através da Internet. POSTEL, J. (1980). User Datagram Protocol. Information Sciences Institute University of Southern California.

A Figura 31 representa a arquitetura do pacote *mavros*, no âmbito do controlador desenvolvido nesta dissertação.

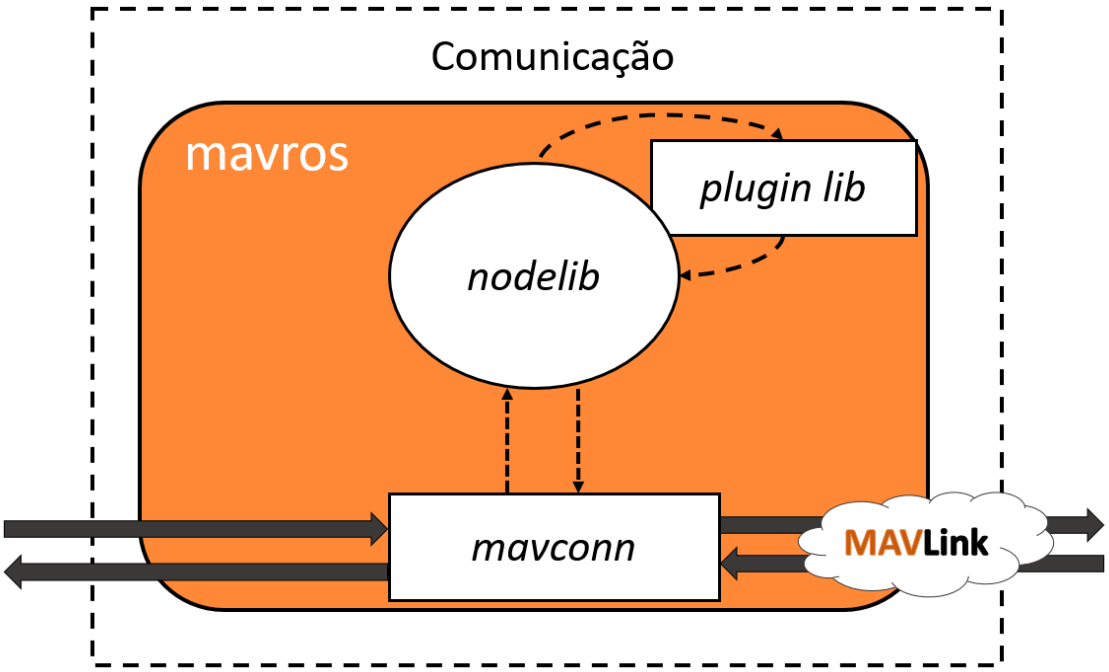

Figura 31 - Arquitetura do pacote *mavros*.

Nodelib contém todas as classes e funções principais do pacote *mavros*, sendo que é definido como o centro de processamento. Permite:

- Criação de classes que definem o tipo de ligação (TCP ou UDP), e que posteriormente estabelecem ligações entre o nó central *mavros* e *plugins*, ou com nós como o *main_controller*, ou nós que representem uma GCS;
- Tradução entre mensagens provenientes do piloto automático para mensagens ROS, customizadas, do tipo MAVLink (*Mavlink.msg*), permitindo o acesso a este tipo de mensagens por parte de outros nós externos de controlo, como por exemplo GCSs;
- A adição de *plugins* através da biblioteca *pluginlib* do ROS, permitindo a adição de novas capacidades, sem a necessidade de alterar o código fonte.

Mavconn é responsável pela codificação e descodificação de mensagens MAVLink, sendo uma componente essencial para o pacote *mavros*. A ligação é definida por URL, sendo que pode ser estabelecida via ligação série. O processamento efetuado pela componente *Mavconn* encontra-se associado a:

- Interligação de uma ligação série com um fluxo de dados que contém mensagens MAVLink enviadas;
- Leitura de mensagens através de funções da biblioteca C++ *Boost.Asio*;
- Análise de mensagens MAVLink recebidas. Após efetuado o controlo de erros de uma mensagem, a informação relativa ao cabeçalho e à carga útil (*payload*) da mensagem é guardada numa estrutura do tipo *mavlink_message*_t, que depende do tipo de mensagem recebida, sendo identificada através de um ficheiro de configuração XML que define os diversos tipos de mensagens.

Plugin lib representa a biblioteca de *plugins* do pacote *mavros*. Os *plugins* são adicionados ao sistema pela classe *MavRos*, através da *pluginlib* do ROS. Cada *plugin* encontra-se associado a uma classe, um tipo e a uma classe base, sendo que se encontram descritos num ficheiro *mavros_plugins.xml*.

No controlador desenvolvido nesta dissertação, foram utilizados os seguintes *plugins* (do *mavros*), que permitem a transmissão de comandos de voo por parte do módulo de comunicação:

- *sys_status*: disponibiliza informação atualizada sobre o estado da ligação, o tipo de piloto automático do VANT, o tipo de VANT, entre outras informações adicionais. No contexto do controlador, é utilizado para escolher o modo de voo (*flight mode*);
- *command*: permite a atribuição de comandos de voo ao piloto automático, em particular, aterragem, descolagem, armar e desarmar;

- *waypoint*: disponibiliza o acesso a dados da missão atribuída ao piloto automático, o que implica a atualização, visualização e eliminação da lista de *waypoints* definida como missão;

Foram ainda utilizados, para a componente de aterragem autónoma visual, os seguintes *plugins*:

- *setpoint_position*: permite enviar para o piloto automático posições, expressas em coordenadas cartesianas (x,y,z), que definem a posição de destino do VANT, expressa no referencial G (Capítulo 1.3.1);

- *vision_pose_estimate*: envia para o piloto automático a pose (posição e atitude) do VANT, estimada através da análise de marcas visuais de aterragem;

3.3.4. Módulo de Aterragem Visual Autónoma

O módulo de aterragem visual autónoma encontra-se adaptado a um tipo de quadrirrotor que possua uma câmara de vídeo acoplada sob o corpo do veículo e disposta por baixo do centro de massa do veículo, na mesma linha vertical. Para efeitos de cálculo a câmara de vídeo do quadrirrotor foi definida como a origem do referencial B, ou seja, o referencial do veículo. Esta aproximação induz um erro da ordem da distância existente entre a posição do centro de massa do veículo e a posição da câmara.

Para o desenvolvimento do módulo, os referenciais utilizados para a criação do módulo de aterragem visual autónoma:

1. Referencial B, com a origem situada na câmara de vídeo do quadrirrotor;

2. Referencial I, aplicado ao plano 2D das imagens detetadas pela câmara de vídeo do quadrirrotor;

3. Referencial G, com a origem situada no centro da marca visual de aterragem.

A arquitetura do módulo de aterragem visual autónoma é representada pela Figura 32.

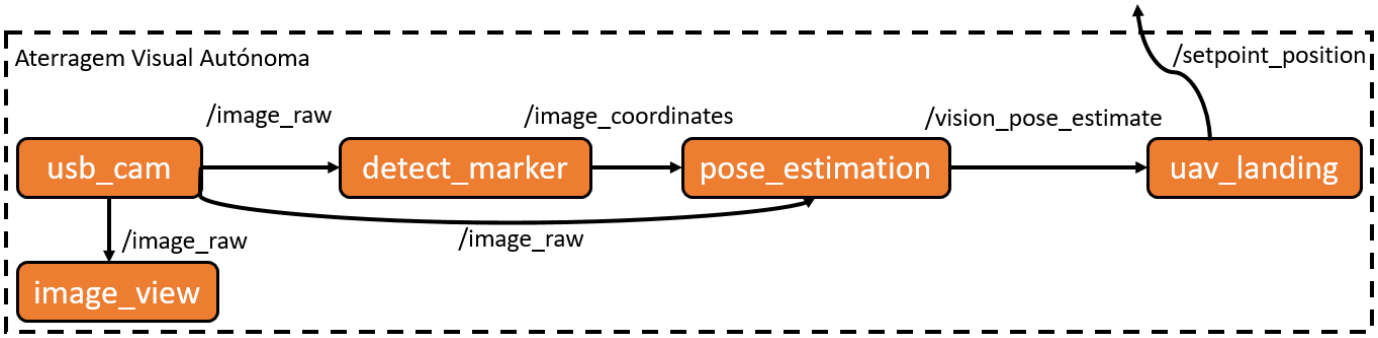

Figura 32 - Arquitetura do módulo de aterragem visual autónoma.

Inicialmente é detetada uma imagem pelo nó *usb_cam*, que é enviada no formato de uma mensagem do tipo *Image.msg*, através do tópico */image_raw*, para o nó de visualização de imagens (*image_view*), para o nó de deteção da marca visual de aterragem (*detect_marker*), e para o nó de cálculo da pose do VANT.

O nó *detect_marker*, ao receber imagens, processa cada imagem recebida com o objetivo de detetar objetos presentes na imagem, de modo a identificar a marca visual de aterragem. Após identificar a marca, envia as coordenadas, expressas em pixéis, no formato de uma mensagem do tipo *Coordinates.msg* através do tópico */image_coordinates*, para o nó *pose_estimation*.

O nó *pose_estimation* é responsável por determinar a posição e atitude (pose) do VANT através da identificação visual da marca de aterragem. Após calcular a pose do VANT, envia a informação no formato de uma mensagem *PoseStamped.msg* através do tópico /vision_pose_estimate, para o nó *uav_landing.*

O nó *uav_landing* recebe a pose estimada, obtendo a posição do VANT em relação ao referencial G, que define o centro da marca visual de aterragem como a origem. De acordo com a informação recebida, o nó (*uav_landing*) efetua o alinhamento gradual do VANT em relação à marca de aterragem, sendo que quando o VANT se

encontra alinhado, o nó inicia o movimento descendente do VANT, até concluir a aterragem.

A arquitetura do módulo de aterragem visual autónoma utiliza nós provenientes de pacotes ROS previamente criados por membros da comunidade (*usb_cam* e *image_view*), e nós criados e desenvolvidos no âmbito da dissertação (*detect_marker*, *pose_estimation*, *uav_landing*) que se encontram descritos nos seguintes subcapítulos.

3.3.4.1. Deteção da Marca Visual

O processo de deteção da marca visual de aterragem é executado através do nó *detect_marker*, que utiliza ferramentas ROS de receção e transmissão de mensagens, e funções de processamento de imagem da biblioteca *OpenCV*. O nó *detect_marker* foi criado com base na conceptualização definida no fluxograma representado na Figura 33.

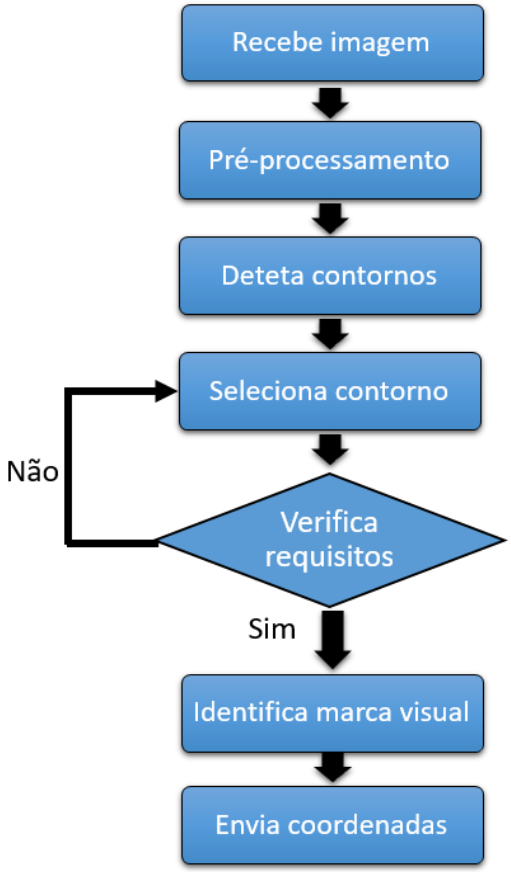

Figura 33 - Fluxograma do nó *detect_marker*.

Inicialmente, o nó *detect_marker* recebe imagens associadas à captura de vídeo processada pelo nó *usb_cam*, através do tópico */usb_cam/image_raw*, e processa sequencialmente e individualmente cada imagem recebida.

Para cada imagem recebida, é feito um pré-processamento com recurso a funções do *OpenCV*, que consiste conversão para imagens de 8-bits, na remoção de cor da imagem (*cv2.cvtColor*), na remoção de ruído de alta frequência (*cv2.GaussianBlur*) e

na deteção de vértices da imagem (*cv2.Canny*), de modo a destacar contornos de objetos presentes na imagem.

A deteção de contornos é feita através da função *cv2.findContours*, que se baseia num algoritmo de extração de contornos de imagens binárias (Suzuki, 1985), e que a partir da imagem pré-processada, devolve uma lista com vetores que contém informação relativa a cada contorno detetado na imagem.

A partir da lista de contornos obtida, é feita uma análise individual a cada contorno, de modo a que sejam detetados contornos correspondentes à marca visual de aterragem. A marca visual de aterragem selecionada corresponde ao desenho típico de um local de aterragem de um helicóptero, que é representado por um "H" inserido numa circunferência (Figura 34).

Figura 34 - Marca visual de aterragem.

Numa abordagem inicial, e para efeitos de simplificação, os requisitos para a deteção da marca visual de aterragem foram adaptados para a deteção da circunferência exterior ao "H".

Durante o processo de verificação de requisitos de cada contorno, são efetuados testes para determinar se o contorno corresponde a uma circunferência de raio superior a um raio mínimo pré-definido, se a região que limita a circunferência é

aproximadamente quadrada, e o contorno detetado apresenta erros de continuidade. Na eventualidade de o contorno analisado não corresponder aos requisitos, é processado o contorno seguinte presenta na imagem.

Ao ser detetada a marca visual são calculados os pontos correspondentes ao centro da circunferência, e aos vértices de um quadrado com o centro comum à circunferência, e de largura e altura iguais ao raio da circunferência. As coordenadas que

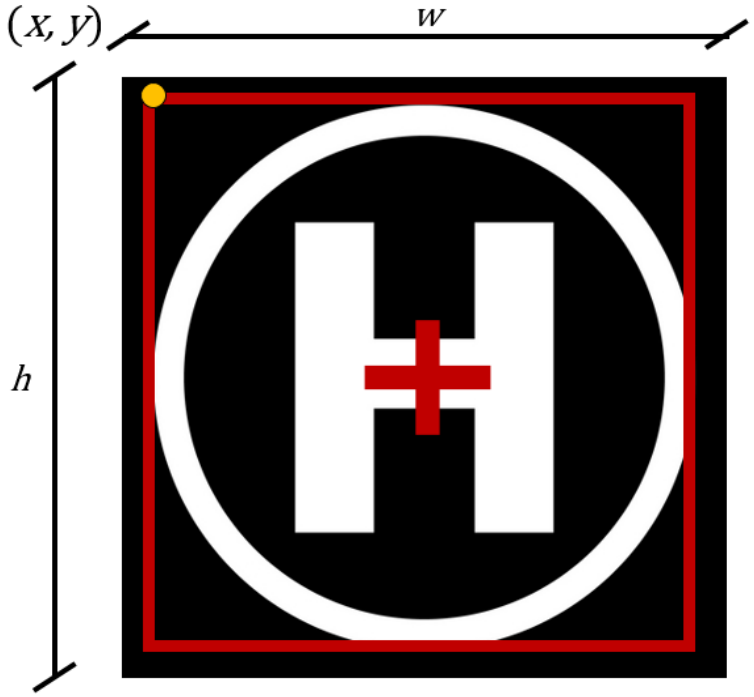

Figura 35 - Representação da deteção da marca visual de aterragem.

representam o vértice superior esquerdo do quadrado (x, y), e a altura e comprimento (h, w) do quadrado exterior à circunferência, são enviados no formato de mensagem do tipo *Coordinates.msg* para o nó *pose*_estimation, sendo posteriormente desenhados sobre a marca visual, e apresentados ao operador de acordo com a Figura 35.

3.3.4.2. Cálculo da Pose

O nó *pose_estimation* efetua a estima da pose do quadrirrotor com recurso a funções ROS e *OpenCV*. A estima da pose do quadrirrotor é obtida através da resolução do problema PnP, processada com base no método EPnP (subcapítulo 1.3.4.3.). O que implica a definição de parâmetros de entrada, nomeadamente: modelos 2D e 3D da

marca visual de aterragem, de uma matriz de parâmetros intrínsecos da câmara, e de um vetor que contêm os coeficientes de distorção associados à câmara de vídeo. A resolução do problema PnP resulta numa matriz de parâmetros extrínsecos da câmara que permitem o cálculo da pose do veículo.

O funcionamento do nó *pose_estimation* encontra-se descrito no fluxograma representado na Figura 36.

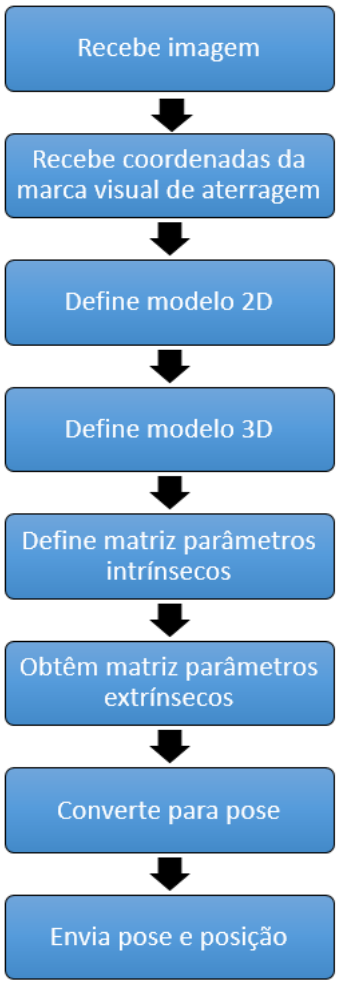

Figura 36 - Fluxograma do nó *pose_estimation*.

Após a receção das mensagens Coordinates.msg e Image.msg, que contêm os valores x, y, w, h (Capítulo) e a imagem visualizada, respetivamente, é definido o modelo 2D da marca visual de aterragem. O modelo é definido por um vetor que contêm

coordenadas 2D (altura e largura) dos quatro cantos e do centro da marca visual de aterragem, representados em pixéis.

O modelo 3D, à semelhança do modelo 2D, é definido por um vetor que contêm coordenadas 3D (*x, y, z*) dos quatro cantos e do centro da marca visual de aterragem, representados em metros e de acordo com o referencial G, sendo o centro da marca visual de aterragem correspondente à origem do referencial G.

De seguida, é definida a matriz de parâmetros intrínsecos da câmara, composta pela distância focal e as coordenadas correspondentes ao centro da marca visual, e o vetor de distorção focal. Para efeitos de implementação inicial e simulação, a distorção focal é assumida como nula, sendo que para reduzir o erro da estima da pose do quadrirrotor, torna-se necessária uma calibração completa da câmara para ser obtido o valor real de distorção focal (Sturm, 1999) (Bradski, 2013).

Após o cálculo dos parâmetros de entrada, é efetuada a resolução do problema PnP através da função *cv2.solvePnP*, que calcula vetores de rotação e de translação, que quando dispostos numa matriz quadrada 4x4, representam a matriz de parâmetros extrínsecos da câmara (Equação (3.1)).

$$\begin{bmatrix} R^B & \Gamma^B \\ 0_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} {R^B}_{11} & {R^B}_{12} & {R^B}_{13} & {\Gamma^B}_1 \\ {R^B}_{21} & {R^B}_{22} & {R^B}_{23} & {\Gamma^B}_2 \\ {R^B}_{31} & {R^B}_{32} & {R^B}_{33} & {\Gamma^B}_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

A matriz de parâmetros extrínsecos descreve a forma em que o mundo (referencial G) é transformado relativamente à câmara (referencial B), ou, por aproximação, relativamente ao veículo (referencial B). O que permite a transformação de pontos representados no referencial G para pontos representados no referencial B.

Sendo que a pose de um VANT é expressa relativamente ao referencial G, a partir da matriz de parâmetros extrínsecos é possível obter a pose do quadrirrotor, aplicando a inversão expressa na Equação (1.6.), e verificando as igualdades expressas nas seguintes Equações (3.2) e (3.3).

$$R^{B^T} = R^{B^{-1}} = R^G \tag{3.2}$$

$$-R^{B^{-1}}\Gamma^B = \Gamma^G \tag{3.3}$$

O cálculo da matriz R^B é efetuado através da aplicação da rotação de Rodrigues ao vetor de rotação obtido na solução do problema PnP.

A rotação do quadrirrotor (R^G) é convertida para um quaternião, otimizando o processamento e evitando o *gimbal lock*. Sendo posteriormente enviada para o nó *uav_landing.*

De acordo com a Equação (3.4), as coordenadas do quadrirrotor (p^B) em relação à marca visual de aterragem (referencial G) correspondem à posição expressa pelo vetor de translação, Γ^G.

$$\begin{bmatrix} p^G \\ 1 \end{bmatrix} = \begin{bmatrix} R^G & \Gamma^G \\ 0_{1x3} & 1 \end{bmatrix} \cdot \begin{bmatrix} p^B \\ 1 \end{bmatrix} = \begin{bmatrix} {R^G}_{11} & {R^G}_{12} & {R^G}_{13} & {\Gamma^G}_1 \\ {R^G}_{21} & {R^G}_{22} & {R^G}_{23} & {\Gamma^G}_2 \\ {R^G}_{31} & {R^G}_{32} & {R^G}_{33} & {\Gamma^G}_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} {\Gamma^G}_1 \\ {\Gamma^G}_2 \\ {\Gamma^G}_3 \\ 1 \end{bmatrix} \tag{3.4}$$

O cálculo da posição do quadrirrotor em relação à marca visual de aterragem (referencial G) é essencial para a execução da aterragem, sendo esta posição posteriormente enviada para o nó *uav_landing*.

A pose do quadrirrotor, expressa através de R^G e Γ^G, é publicada para o tópico */vision_pose_estimate* que é subscrito pelo nó *uav_landing* e pelo *mavros*, garantido a distribuição de informação essencial para o processamento final da aterragem e para o processamento interno do módulo do piloto automático.

3.3.4.3. Aterragem

A fase final da implementação de aterragem visual autónoma consiste no nó *uav_landing*, que recebe a posição do quadrirrotor no referencial G com o objetivo de diminuir a distância entre a origem do referencial (marca visual de aterragem) e a posição do quadrirrotor, efetuando a aterragem. A Figura 37 representa o funcionamento do nó *uav_landing*.

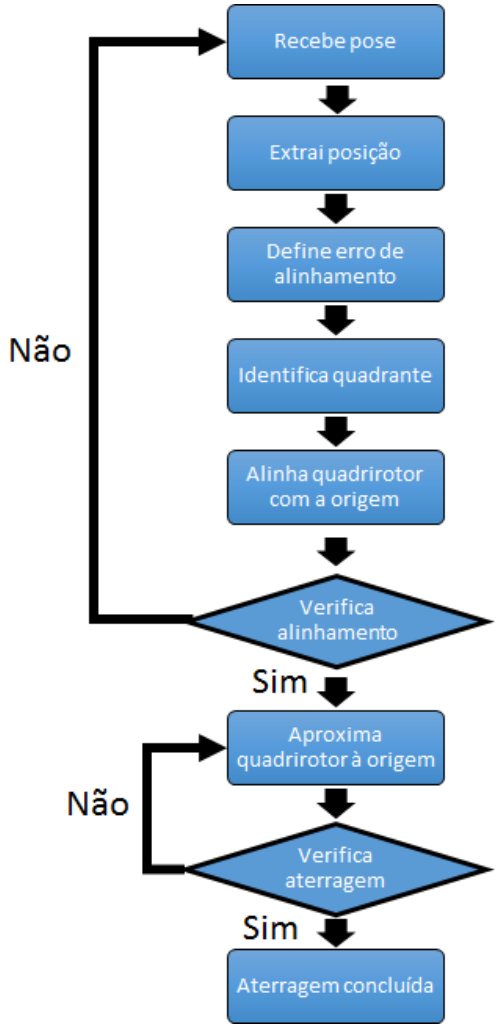

Figura 37 - Fluxograma do nó *uav_landing*.

A pose do quadrirrotor, transmitida pelo nó *uav_landing*, é recebida e utilizada para a extração da posição do quadrirrotor no referencial G.

De seguida é definido o erro de alinhamento permitido para se iniciar o alinhamento com a marca visual de aterragem. Para efeitos de implementação inicial e simulação foi definida uma área centrada na marca visual de aterragem com raio de 0,2m, em que o quadrirrotor efetua o alinhamento enquanto não for detetado na área. Este processo baseia-se na identificação do quadrante em o quadrirrotor se encontra, seguido pela redução da distância entre as suas coordenadas x e y, e a origem, a uma

velocidade correspondente a 10% da distância horizontal existente entre o quadrirrotor e o alvo, por iteração, garantindo uma aproximação gradual e estável.

O alinhamento do quadrirrotor implica a publicação nova posição calculada para o piloto automático, através do tópico */setpoint_position*, que por sua vez efetua o processamento e encaminhamento do quadrirrotor para a posição calculada. Enquanto o quadrirrotor não atingir a área de limite definida, repete este processo, atualizando a pose recebida de acordo com a nova visualização da marca visual, consequente da sua posição calculada.

Quando o quadrirrotor entra na área de limite definida, o nó *uav_landing* inicia o processo de aproximação. Este processo é executando apenas enquanto o quadrirrotor se encontrar alinhado com a marca visual de aterragem, a uma velocidade correspondente a 10% da distância vertical existente entre o quadrirrotor e o alvo, por iteração, garantindo uma aproximação gradual e estável. A aproximação é terminada no momento em que a distância é aproximada a nula, concluindo o processo de aterragem visual autónoma.

3.3.5. Operação do Controlador

A operação do controlador é efetuada através de um interface gráfico, desenvolvido através do ROS, que integra todas as funcionalidades desenvolvidas no módulo de comando e controlo, permitindo ao utilizador a execução de comandos de voo. Permite ainda iniciar a funcionalidade adicional de aterragem visual autónoma. O interface gráfico encontra-se representado na Figura 38.

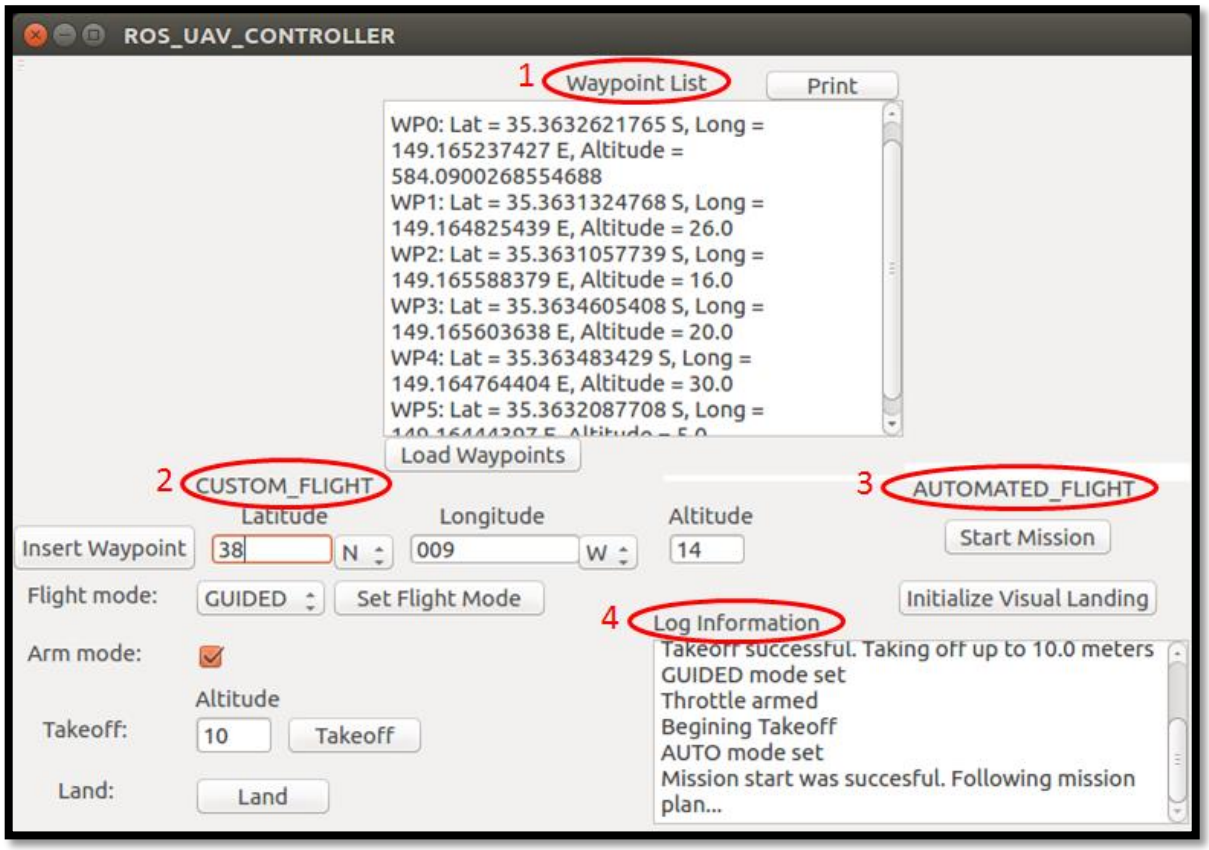

Figura 38 - Interface gráfico de operação do controlador.

O interface gráfico encontra-se organizado de acordo com os seguintes módulos:

1. *Waypoint List*;
2. CUSTOM_FLIGHT;
3. AUTOMATED_FLIGHT;
4. *Log Information*;

O módulo *Waypoint List* permite a visualização da lista de *waypoints* atribuída ao piloto automático, através da função "Print". Permite ainda enviar uma lista de *waypoints*, definida num ficheiro de texto, para o piloto automático através da função "Load Waypoints".

O módulo *CUSTOM_FLIGHT* integra o controlo individual de comandos de voo, permitindo inserir *waypoints* através da função "Insert Waypoint", que recebe um valor de latitude, de longitude e de altura. A função "Set Flight Mode" permite a seleção e atribuição de um modo de voo (GUIDED/AUTO/MANUAL/STABILIZE/TEST) do piloto automático. A função "Arm mode" permite armar/desarmar, o VANT, o que permite/inibe a descolagem. As funções "Takeoff" e "Land" permitem a descolagem e aterragem, respetivamente, sendo que é possível definir a altitude pretendida para a descolagem.

O módulo *AUTOMATED_FLIGHT* permite a inicialização automática do VANT, que se traduz no desempenho de funções de descolagem, seleção do modo de voo automático (AUTO), no seguimento de *waypoints* atribuídos ao piloto automático e na aterragem do VANT. Permite ainda a inicialização de funcionalidades adicionadas ao controlador, sendo que neste caso permite a inicialização do módulo de aterragem visual autónoma do controlador.

Por fim, o módulo *Log Information* apresenta informação relativa à execução de funções definidas nos restantes módulos, indicando a ocorrência de erros e apresentando o estado de funções executadas.

Capítulo 4. Validação

4.1. Introdução

Este capítulo descreve o processo de validação do controlador desenvolvido através da definição de meios e cenários de teste. Inicialmente, apresenta o *software* (subcapítulo 4.2) e *hardware* (subcapítulo 4.3) utilizados para a realização dos testes. De seguida, é apresentada a proposta de validação do controlador, dividida em dois subcapítulos. O subcapítulo 4.4.1 apresenta o cenário de validação do módulo de comando e controlo, que avalia o tempo de transmissão de comandos de voo para o piloto automático. Por fim, o subcapítulo 4.4.2 apresenta o cenário de validação do módulo de aterragem visual autónoma, que avalia o erro associado à estima da pose do VANT, bem como o erro associado ao alinhamento e aproximação do VANT à marca visual de aterragem.

4.2. Software

A fase inicial de implementação e testes ao controlador foi efetuada com recurso a simuladores de GCSs e de pilotos automáticos de VANTs.

Para simular a GCS foi utilizado o programa *ArduPilot Mega Planner 2*, que consiste numa aplicação, de edição-livre, que simula uma GCS e que se encontra adaptada pilotos automáticos que utilizam o protocolo MAVLink (e.g. *ArduPilot Mega* e PX4/*Pixhawk*). O *ArduPilot Mega Planner 2* permite a visualização e monitorização do comportamento do VANT durante o voo, configuração e calibração do piloto automático para a integração de comportamento autónomo, e a atribuição de missões ao piloto automático a partir de *waypoints*.

Quanto ao piloto automático, foi utilizado o simulador *Software-in-the-Loop* (SITL), que integra as funcionalidades do simulador *ArduPilot*, com a adição da capacidade de abstração de *hardware*. O SITL permite a simulação de um quadrirrotor, sem qualquer tipo de *hardware*, utilizando o protocolo MAVlink para a receção de comandos de voo a partir da GCS.

4.3. *Hardware*

Numa fase inicial a implementação do controlador, assim como os testes, foram executados num computador portátil *Toshiba Satellite P750*, com um processador *Intel i7-2630QM* de 2 gigahertz*,* uma memória de 4 gigabytes e uma câmara integrada de 1.3 megapixéis.

4.4. Validação do Controlador

A validação do controlador define os cenários de teste associados aos dois módulos desenvolvido para o controlador: Comando e Controlo, e Aterragem Visual Autónoma. Para além da descrição dos testes a efetuar, são referidos alguns dos resultados expectáveis.

4.4.1. Módulo de Comando e Controlo

O cenário de validação associado ao módulo de comando e controlo é focado na medição dos tempos de transmissão de comandos de voo para o piloto automático através do controlador. O grau de sucesso da transmissão de comandos de voo, por parte do controlador, é medido de acordo com a comparação com os tempos de transmissão de comandos de voo através dos utilitários (*mavwp*, *mavcmd* e *mavsys*) do nó *mavros*. Os testes compreendem a medição dos tempos de transmissão de comandos de voo relativos à introdução de *waypoints*, aterragem e descolagem, e escolha do modo de voo, sendo efetuadas trinta medições para cada comando. É expectável que controlador demore mais tempo a executar os comandos de voo, uma vez que, de uma forma geral, implica processamento adicional. Contundo, a diferença de tempo deverá possuir um impacto irrelevante no desempenho dos comandos de voo selecionados.

4.4.2. Módulo de Aterragem Visual Autónoma

A validação do módulo de aterragem visual autónoma compreende o cálculo do erro associado às coordenadas x e y durante a fase de alinhamento com a marca visual de aterragem, e do erro associado à coordenada z durante a fase de aproximação descendente à marca visual de aterragem.

A validação do alinhamento com a marca visual é efetuada através da comparação entre os valores de obtidos a partir do módulo de aterragem visual autónoma, com valores de referência correspondentes à distância real da câmara até ao centro da marca visual, simulando o alinhamento do VANT (Figura 39). A execução dos testes foi efetuada com recurso a uma marca visual de aterragem (subcapítulo 3.3.4.1.) que consiste num "H" inserido numa circunferência de raio igual a 8,3 cm. De modo a avaliar o sucesso da implementação, foi definido um parâmetro de validação que corresponde a um erro obtido inferior à distância de 8,3 cm.

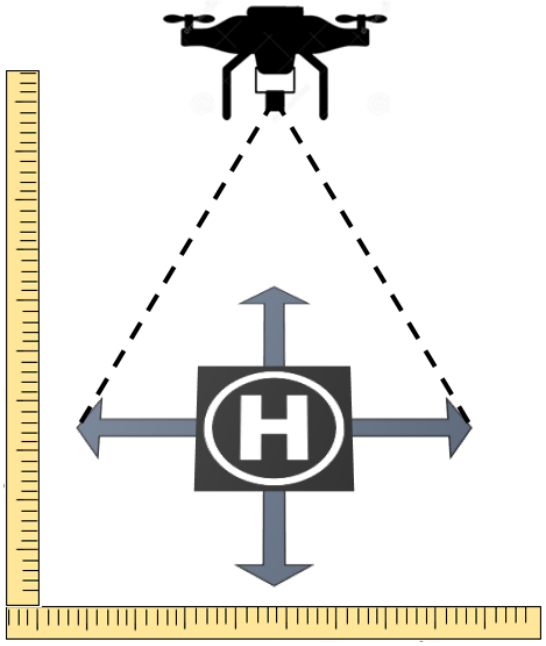

Figura 39 - Validação do alinhamento do VANT com a marca visual de aterragem.

Para uma coordenada z fixa (igual a 80 cm), foi testado o cálculo da posição da câmara em relação ao referencial com origem na marca visual de aterragem. Como tal foram definidas nove posições distintas (O, A, B, C, D, E, F, G, H) para a marca visual de

aterragem, dispostas numa estrutura de suporte com comprimento igual a 48 cm e largura igual a 44 cm (Figura 40).

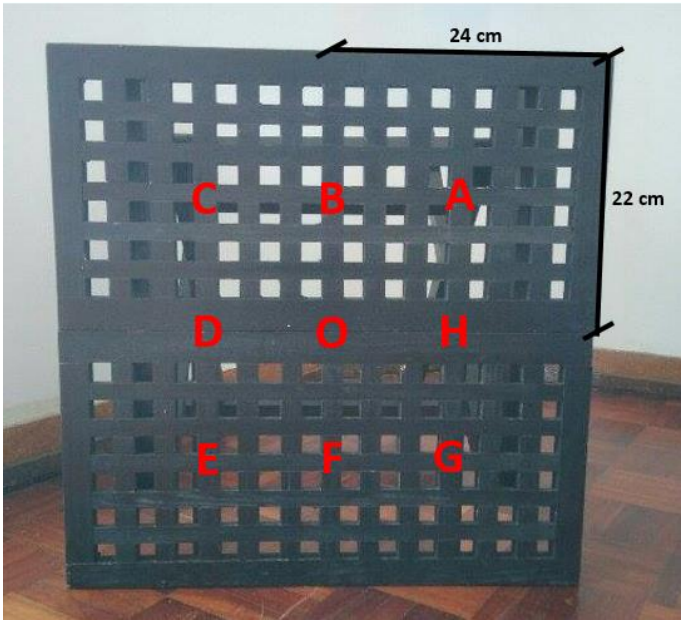

Figura 40 - Estrutura e posições de testes utilizadas.

A validação da aproximação descendente à marca visual é efetuada através da comparação entre os valores de obtidos a partir do módulo de aterragem visual autónoma, com valores de referência correspondentes à distância real da câmara até ao centro da marca visual, simulando a aproximação do VANT (Figura 41).

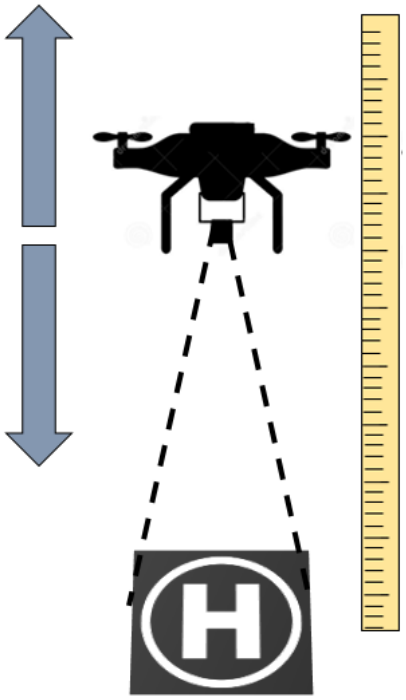

Figura 41 - Validação da aproximação descendente do VANT à marca visual de aterragem.

Para coordenadas xy fixas (correspondentes à posição "O" representada na Figura 40), foi testado o cálculo da coordenada z para diferentes distâncias entre o alvo e a câmara (116, 110, 100, 90, 80, 70, 60, 50, 40 e 30 cm).

Em ambos os testes, o cálculo do erro é obtido através da comparação entre as distâncias euclidianas experimental e real, calculadas entre o VANT e a marca visual de aterragem. Cada valor analisado corresponde à média de cinquenta iterações distintas, calculadas para cada posição, ou seja, o valor analisado para a distância entre o à marca visual e a câmara na posição O, corresponde à média de cinquenta valores calculados para a posição O. É expectável que o rigor das medições seja essencialmente afetado pela aproximação efetuada no cálculo da matriz de parâmetros intrínsecos da câmara, pelas condições de luminosidade, pelo erro induzido pela inexistência de uma estrutura de suporte fixo que limite movimentos indesejados da câmara e pelo erro derivado da inexistência de uma calibração prévia da câmara. O ambiente utilizado para a realização dos testes corresponde a um ambiente interior de uma casa.

Capítulo 5. Análise e Discussão de Resultados

5.1. Introdução

Este capítulo apresenta os resultados obtidos nos testes descritos no capítulo da validação (capítulo 4), assim como uma interpretação de resultados. Inicialmente, são analisados resultados de testes ao desempenho do módulo de comando e controlo (subcapítulo 5.2.), em particular, do desempenho do controlador ao enviar comandos de voo como: carregamento de *waypoints*, descolagem, aterragem e escolha do modo de voo. Por fim, é analisado o desempenho do módulo de aterragem visual autónoma (subcapítulo 5.3.) durante o processo de cálculo da pose do quadrirrotor (câmara), a partir da marca visual de aterragem. São analisados resultados correspondentes à fase de alinhamento e à fase descendente do quadrirrotor.

5.2. Módulo de Comando e Controlo

De acordo com o subcapítulo 4.4.1., foram testados os tempos de transmissão de comandos de voo, por parte do controlador, para o piloto automático.

Os primeiros testes correspondem à transmissão de uma lista composta por cinco *waypoints*. O gráfico da Figura 42 representa os resultados obtidos para a execução de trinta testes, nomeadamente, os valores respetivos aos tempos de transmissão (em segundos) do controlador e do utilitário *mavwp* (do nó *mavros*), representando ainda a diferença de desempenho expressa pelo valor absoluto do erro associado a cada teste e a regressão linear do erro para os trinta testes. Verifica-se uma diferença média de 925 milissegundos entre o tempo de transmissão de comandos de voo por parte do controlador e o de transmissão de comandos de voo por parte do utilitário *mavwp* (Tabela 3). A magnitude do erro obtido deve-se essencialmente à quantidade de processamento adicional que se encontra associado à operação através do interface gráfico. Tendo em conta a natureza do comando de voo testado, esta diferença não resulta em nenhum impacto operacional significativo para o voo, ou para o controlo do VANT.

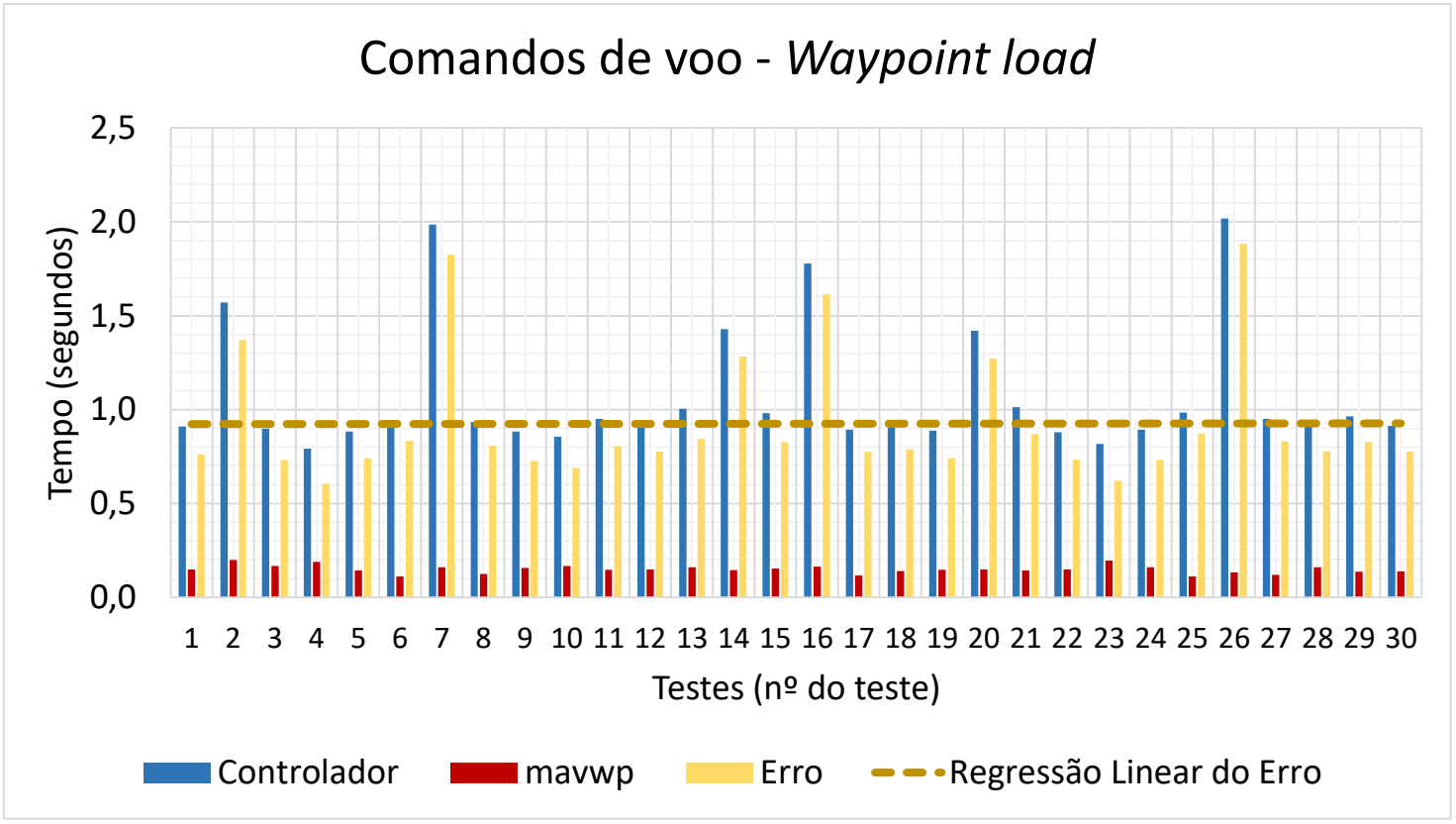

Figura 42 - Resultados dos testes efetuados para a transmissão de *waypoints*.

Tabela 3 - Média e mediana obtida nos testes de transmissão de *waypoints*.

Comandos de voo - *Waypoint Load*		
	Média (segundos)	Mediana (segundos)
Controlador	1,074	0,935
mavwp	0,149	0,148
Erro	0,925	0,787

Os testes seguintes correspondem à transmissão do comando de voo de descolagem e aterragem.

Os gráficos das Figura 43 e Figura 44 representam os resultados obtidos para a execução de trinta testes, nomeadamente, os valores respetivos aos tempos de transmissão (em segundos) do controlador e do utilitário *mavcmd* (do nó *mavros*), representando ainda a diferença de desempenho expressa pelo valor absoluto do erro associado a cada teste e a regressão linear do erro para os trinta testes.

No caso da descolagem, verifica-se uma diferença média de 6 milissegundos entre o tempo de transmissão de comandos de voo por parte do controlador e o de transmissão de comandos de voo por parte do utilitário *mavcmd* (Tabela 4). Tendo em conta o erro reduzido, os resultados obtidos são positivos no sentido em que o tempo adicionado pelo acréscimo de processamento associado ao controlo efetuado através do interface gráfico é ínfimo. O facto de o comando de voo ser transmitido diretamente para o módulo de comunicação, através do serviço */mavros/command/takeoff*, ao invés de ser utilizado um utilitário do mavros como intermediário, também contribui para o melhoramento do desempenho do controlador.

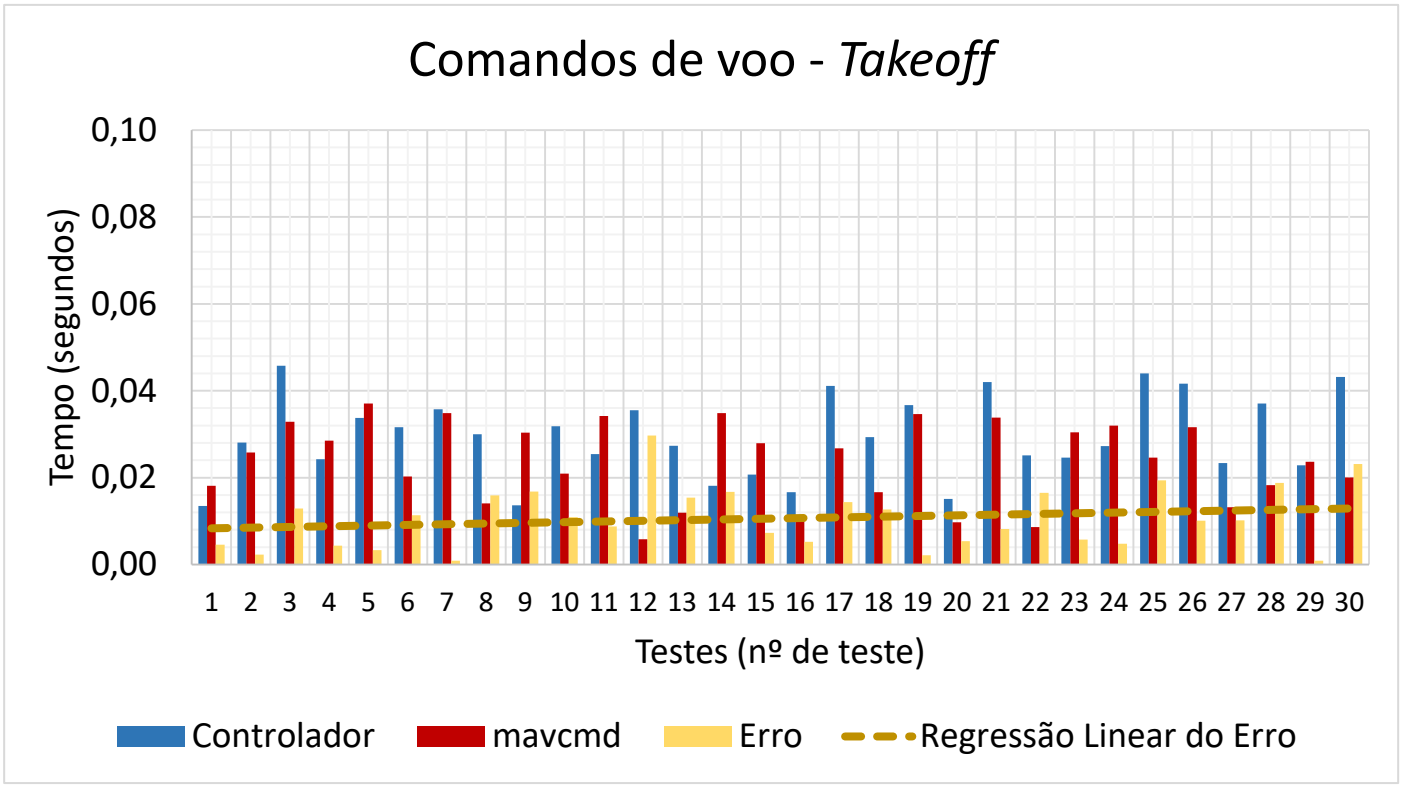

Figura 43 - Resultados dos testes efetuados para a transmissão do comando de voo de descolagem.

Tabela 4 - Média e mediana obtida nos testes de transmissão do comando de voo de descolagem.

Comandos de voo - *Takeoff*		
	Média (segundos)	Mediana (segundos)
Controlador	0,029	0,029
mavwp	0,024	0,025
Erro	0,005	0,004

Quanto à aterragem, verifica-se uma diferença média de 5 milissegundos entre o tempo de transmissão de comandos de voo por parte do controlador e o de transmissão de comandos de voo por parte do utilitário *mavcmd* (Tabela 5). À semelhança dos testes anteriores, os resultados obtidos são positivos no sentido em que o tempo adicionado pelo acréscimo de processamento associado ao controlo efetuado através do interface gráfico é ínfimo. O facto de o comando de voo ser transmitido diretamente para o módulo de comunicação, através do serviço

/mavros/command/landing, ao invés de ser utilizado um utilitário do *mavros* como intermediário, também contribui para o melhoramento do desempenho do controlador.

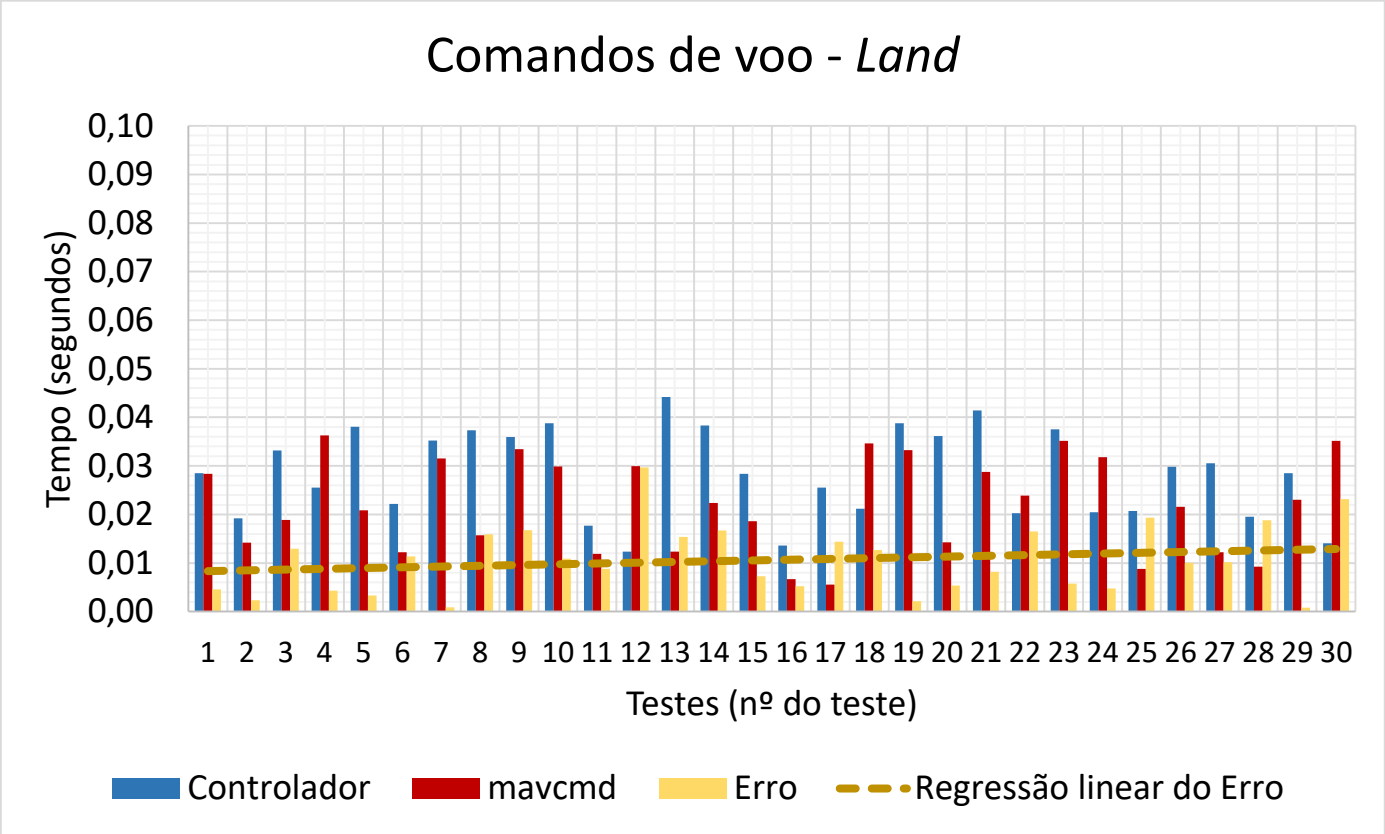

Figura 44 - Resultados dos testes efetuados para a transmissão do comando de voo de aterragem.

Tabela 5 - Média e mediana obtida nos testes de transmissão do comando de voo de aterragem.

Comandos de voo - *Land*		
	Média (segundos)	Mediana (segundos)
Controlador	0,028	0,029
mavwp	0,022	0,022
Erro	0,006	0,007

A escolha do modo de voo para GUIDED concluí a sequência de testes efetuados ao módulo de comando e controlo. O gráfico da Figura 45 representa os resultados obtidos para a execução de trinta testes, nomeadamente, os valores respetivos aos

tempos de transmissão (em segundos) do controlador e do utilitário *mavsys* (do nó *mavros*), representando ainda a diferença de desempenho expressa pelo valor absoluto do erro associado a cada teste e a regressão linear do erro para os trinta testes. Surpreendentemente, o controlador demonstrou um desempenho superior ao utilitário *mavsys*, sendo verificada uma diferença média de 32 milissegundos entre o tempo de transmissão de comandos de voo por parte do controlador e o de transmissão de comandos de voo por parte do utilitário *mavsys* (Tabela 6). O facto de o comando de voo ser transmitido diretamente para o módulo de comunicação, através do serviço /mavros/set_mode, ao invés de ser utilizado um utilitário do *mavros* como intermediário, contribuiu claramente para o desempenho superior do controlador.

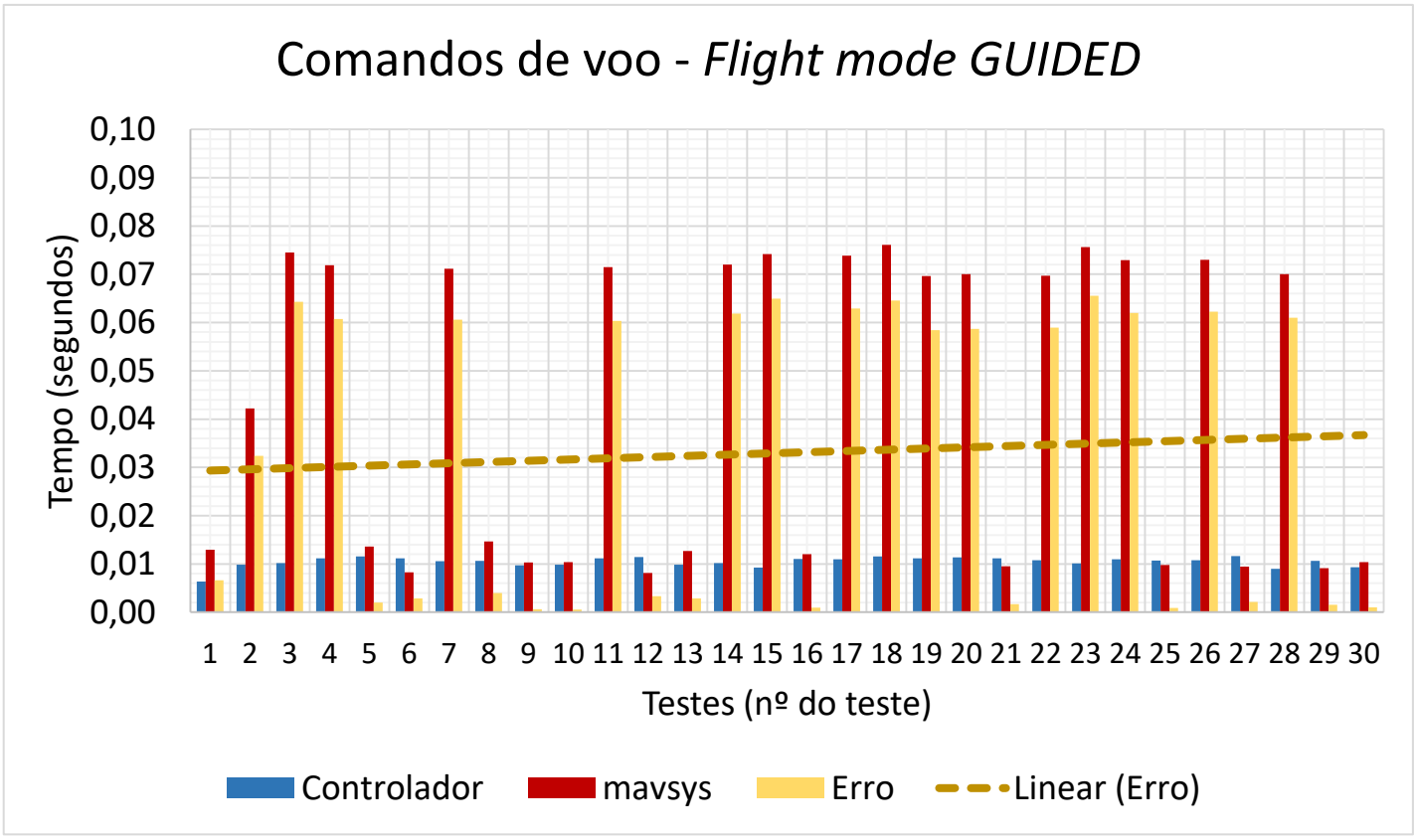

Figura 45- Resultados dos testes efetuados para a transmissão do comando de voo de escolha do modo de voo.

Tabela 6 - Média e mediana obtida nos testes de transmissão do comando de voo de escolha do modo de voo.

Comandos de voo - *Flight mode GUIDED*		
	Média (segundos)	Mediana (segundos)
Controlador	0,010	0,011
mavwp	0,043	0,056
Erro	0,032	0,045

5.3. Módulo de Aterragem Visual Autónoma

De acordo com o subcapítulo 4.4.2., foi calculado o erro entre a distância, à marca visual de aterragem, real e calculada pelo controlador, durante a fase de alinhamento e durante a fase de aproximação descendente.

O teste da fase de alinhamento compreende o cálculo das coordenadas xy, bem como a distância euclidiana entre o ponto de origem (ponto O) e a localização da marca visual de aterragem, relativa apenas às coordenadas xy. A Figura 46 retrata os diversos testes efetuados à fase de alinhamento, para as diferentes posições do alvo.

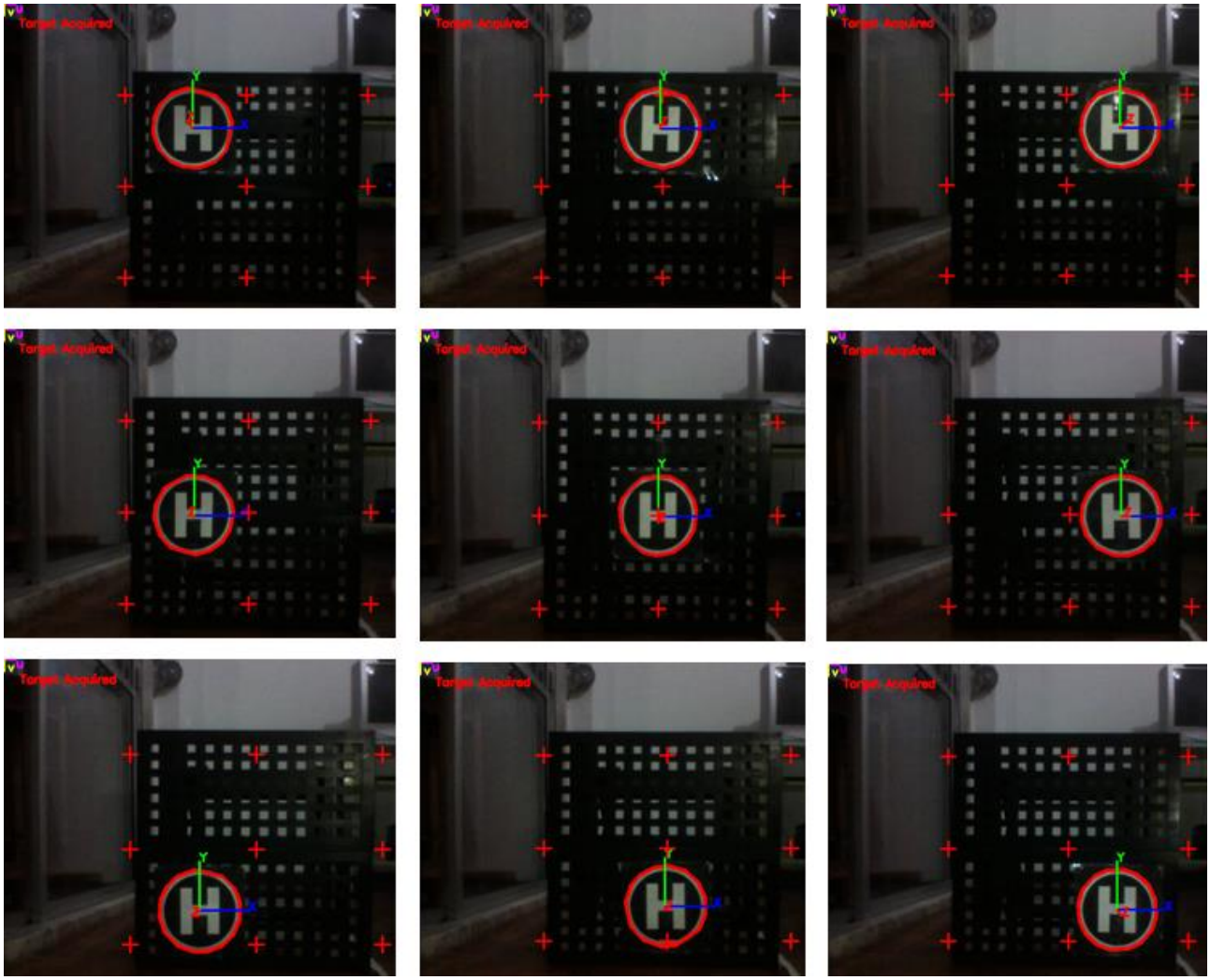

Figura 46 - Testes da fase de alinhamento para nove posições distintas.

A sequência dos testes efetuados corresponde às diferentes posições de teste, começando na posição O, e terminando na posição H, de acordo com a sequência: O-A-B-C-D-E-F-G-H. A Figura 47 representa os resultados obtidos para o cálculo da distância entre a câmara e a marca visual, relativa ao eixo do *x*, através do nó *pose_estimation*, onde é possível verificar o erro entre a distância real e a distância calculada. De acordo com os testes efetuados, verificou-se um erro mínimo de 0%, correspondente ao teste na posição O, e um erro máximo de 68,4%, correspondente ao teste na posição H, e uma média de erro de 22%, sendo que estas percentagens são relativas ao valor do raio (8,3 cm) da circunferência exterior ao "H" (da marca visual de aterragem).

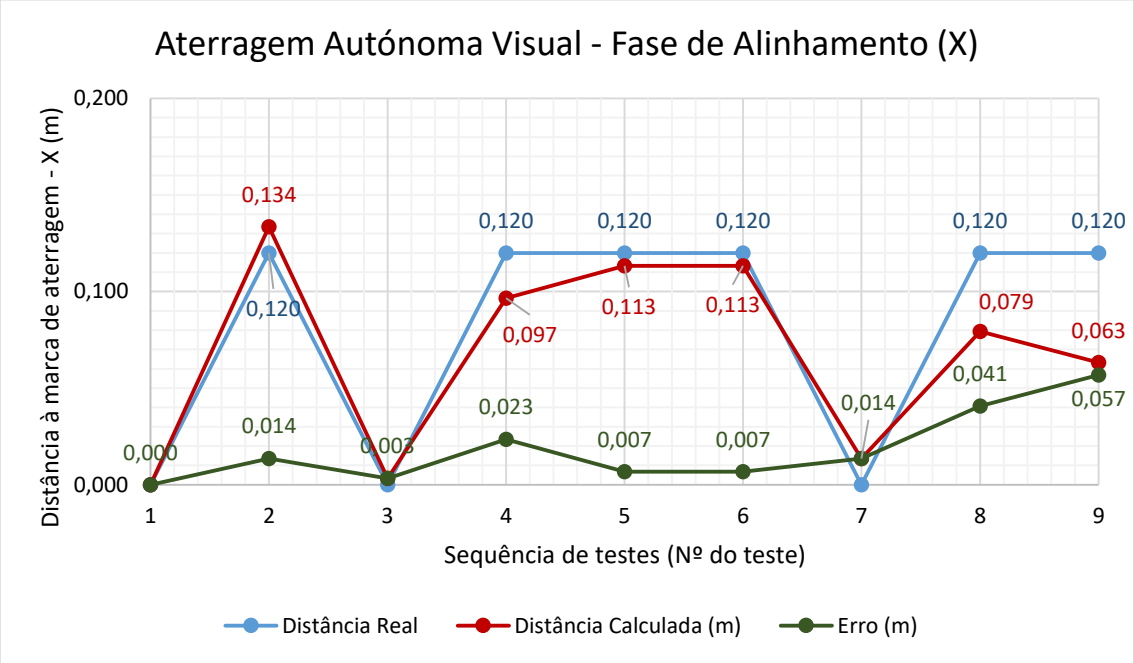

Figura 47 - Resultados dos testes efetuados na fase de alinhamento, relativo à coordenada *x*.

A Figura 48 representa os resultados obtidos para o cálculo da distância entre a câmara e a marca visual, relativa ao eixo do *y*, através do nó *pose_estimation*, onde é possível verificar o erro entre a distância real e a distância calculada. De acordo com os testes efetuados, verificou-se um erro mínimo de 2,75%, correspondente ao teste na posição F, e um erro máximo de 57,6%, correspondente ao teste na posição O, e uma média de erro de 25%, sendo que estas percentagens são relativas ao valor do raio (8,3 cm) da circunferência exterior ao "H" (da marca visual de aterragem).

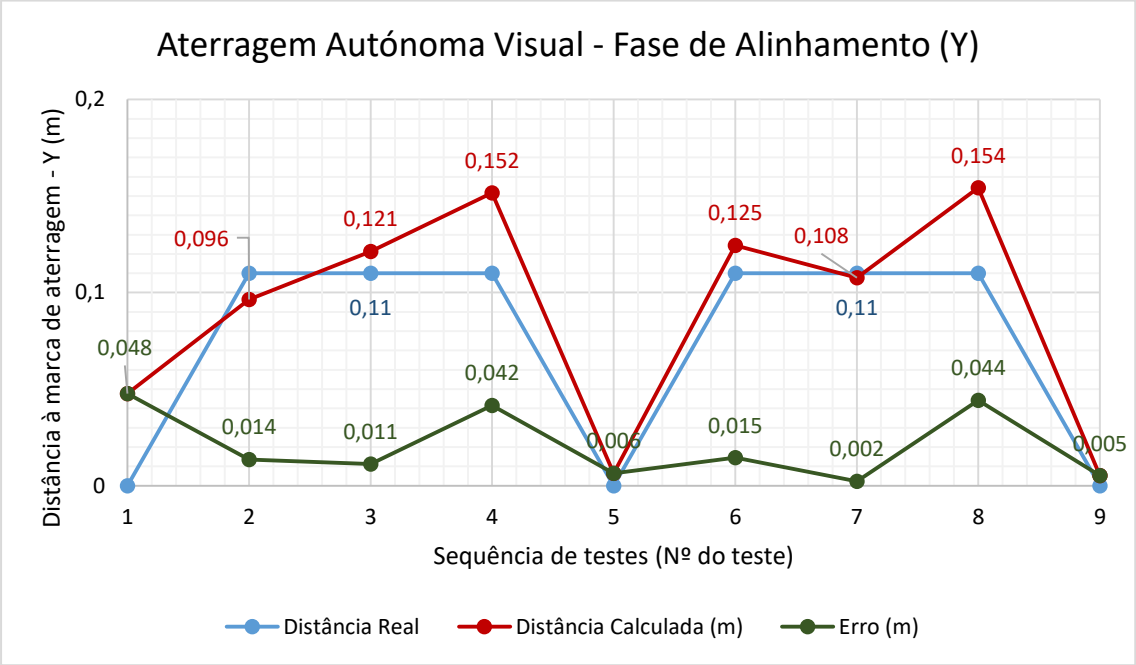

Figura 48 - Resultados dos testes efetuados na fase de alinhamento, relativo à coordenada *y*.

A Figura 49 representa os resultados obtidos para o cálculo da distância entre a câmara e a marca visual, relativa ao plano *xy*, através do nó *pose_estimation*, onde é possível verificar o erro entre a distância real e a distância calculada. De acordo com os testes efetuados, verificou-se um erro mínimo de 1,7%, correspondente ao teste na posição F, e um erro máximo de 68,2%, correspondente ao teste na posição H, e uma média de erro de 21,3%, sendo que estas percentagens são relativas ao valor do raio (8,3 cm) da circunferência exterior ao "H" (da marca visual de aterragem). Apesar da média de erro obtida para a fase de alinhamento ser da ordem dos 21,3% (1,8 cm), os resultados obtidos foram positivos, uma vez que nunca atingiram a ordem dos 100%, o que implicaria a aterragem num ponto exterior à circunferência desenhada na marca visual.

Com o erro obtido, seria possível garantir um alinhamento do VANT com um ponto correspondente ao interior da circunferência da marca visual.

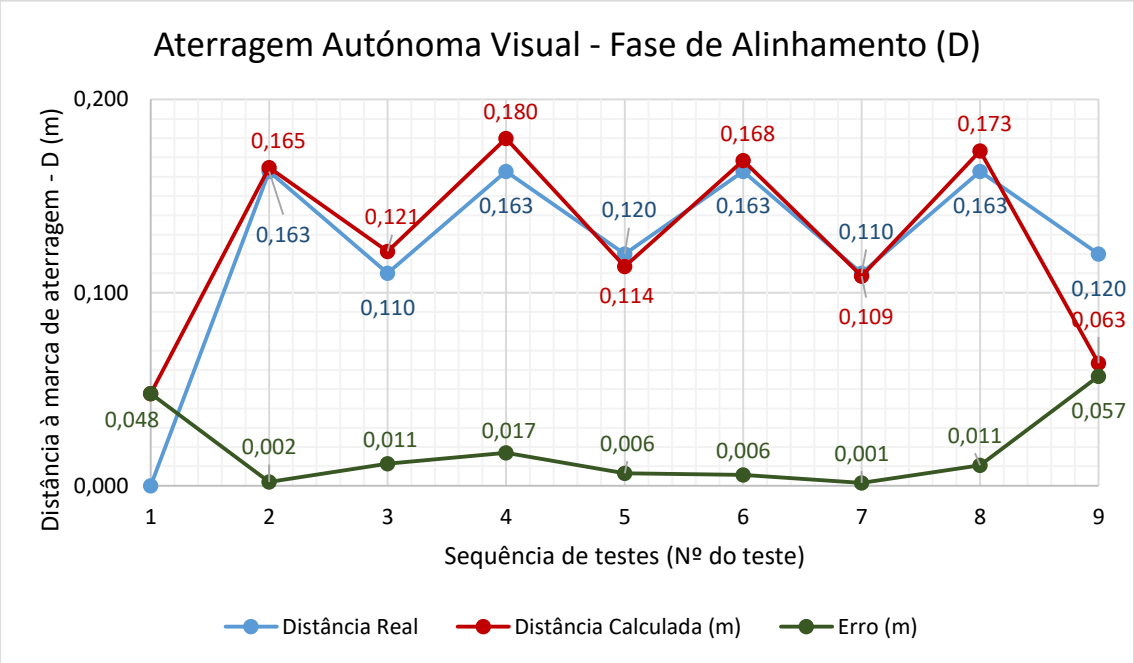

Figura 49 - Resultados dos testes efetuados na fase de alinhamento, relativo à distância entre a marca de aterragem e a câmara, no plano *xy*.

No que toca à fase de aproximação descendente, a sequência dos testes efetuados corresponde à posição O (Figura 50) detetada a distâncias diferentes no eixo do z, começando a uma distância z = 116 cm, e terminando a uma distância z = 30 cm, de acordo com a sequência: 116, 110, 100, 90, 80, 70, 60, 50, 40 e 30 cm. Os limites máximo e mínimo foram definidos de acordo com a capacidade do controlador para a deteção da marca visual, uma vez que para $z > 116$ cm a marca não é detetável, e para $z < 30$ cm a marca visual excede os limites da imagem detetada pela câmara.

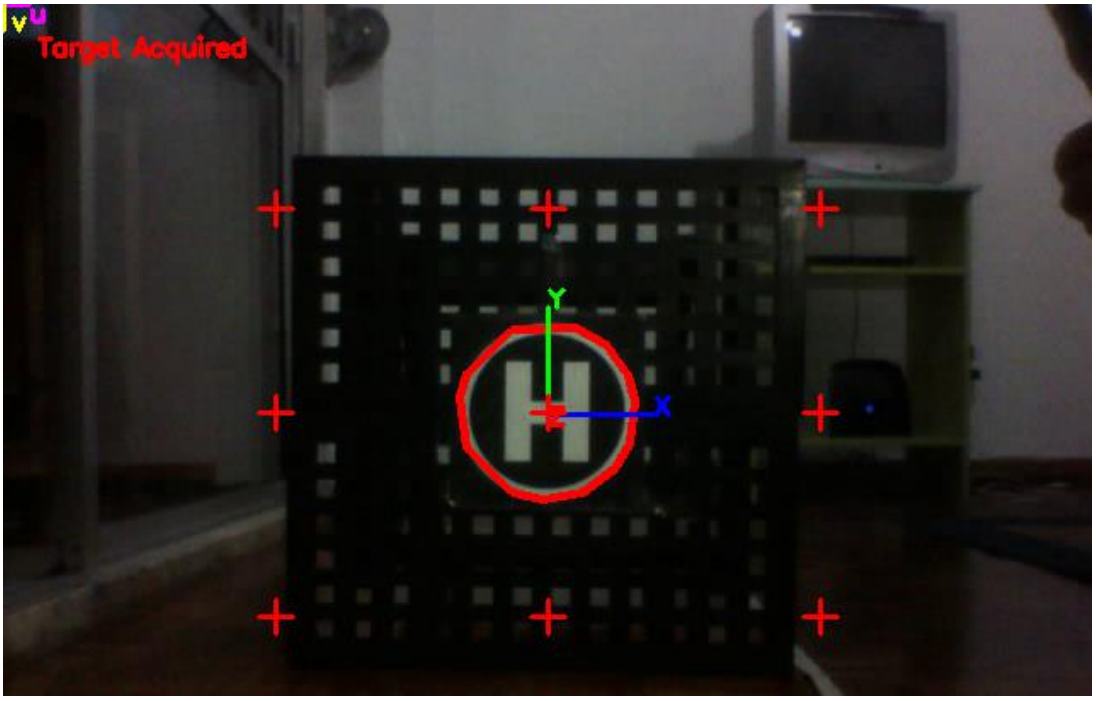

Figura 50 - Testes da fase de aproximação descendente para distâncias distintas distintas.

A Figura 51 representa os resultados obtidos para o cálculo da distância entre a câmara e a marca visual, relativa ao eixo do z, através do nó *pose_estimation*, onde é possível verificar o erro entre a distância real e a distância calculada. De acordo com os testes efetuados, verificou-se um erro mínimo de 6%, correspondente à distância de 50 cm, e um erro máximo de 28,1%, correspondente à distância de 116 cm, e uma média de erro de 16%, sendo que estas percentagens são relativas ao valor do raio (8,3 cm) da circunferência exterior ao "H" (da marca visual de aterragem). O erro médio obtido corresponde a 1,3 cm, o que se traduz num conjunto de resultados positivos.

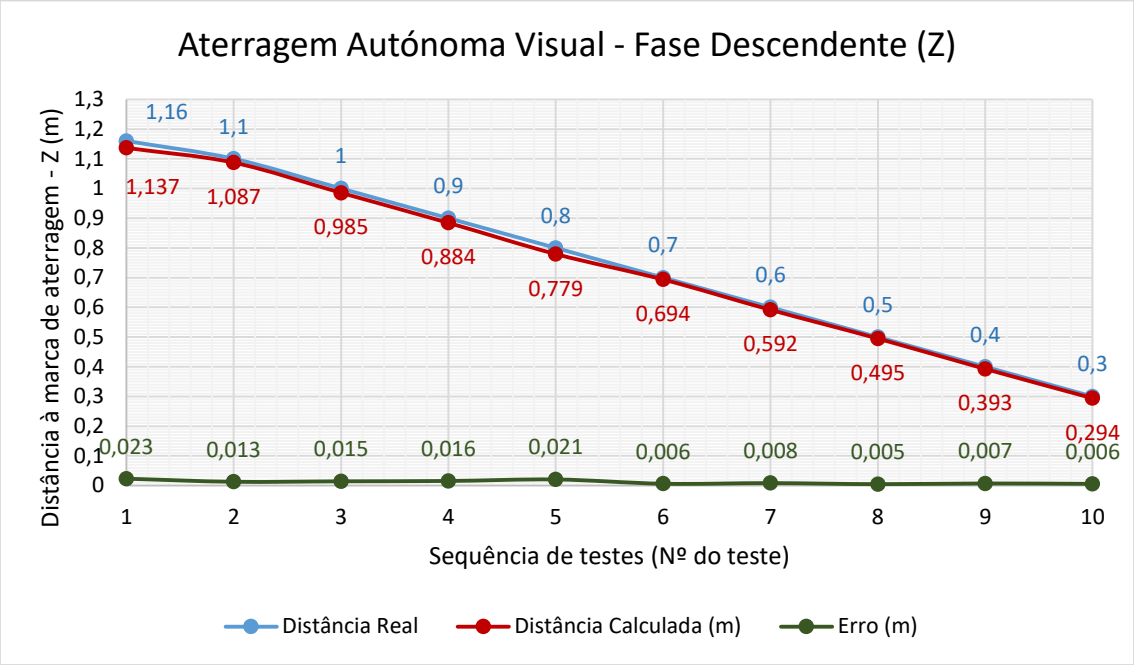

Figura 51 - Resultados dos testes efetuados na fase descendente, relativo à coordenada *z*.

Conclusão

Através de um estudo inicial de conceitos essenciais para a realização da dissertação e da definição de uma metodologia de investigação, foi realizada a implementação de um sistema de controlo externo para um piloto automático de um VANT, com recurso a uma norma de referência que promove interoperabilidade e permite a adição de novas funcionalidades.

Durante esta fase de modelação e implementação do controlador, foram verificadas algumas limitações que afetaram o rigor dos resultados obtidos, como a falta de calibração completa da câmara utilizada para a execução dos testes, que condiciona a aquisição dos parâmetros intrínsecos da câmara e a compensação da distorção gerada pela câmara. Adicionalmente, a inexistência de filtragem temporal, que permite melhorar o desempenho do sistema através da implementação de um algoritmo, que se baseia num conjunto de medições anteriores, da pose da câmara, com o objetivo de calcular valores estimados que se aproximem aos valores reais obtidos, gerando um mecanismo de previsão de resultados relevante para a implementação do sistema de controlo no VANT. Quanto à deteção da marca visual de aterragem, advém uma limitação adicional relativa à simplicidade do algoritmo de identificação da marca, que num ambiente com diversos elementos distintos presentes na imagem demonstra-se propício à deteção de falsos positivos. Por fim, a principal limitação consiste na incapacidade da realização de testes do sistema de controlo externo integrado num VANT, a operar diretamente sobre o piloto automático e a ser controlado por um operador em terra, num ambiente de teste controlado e a bordo de um navio de guerra.

De modo a mitigar o impacto destas limitações, torna-se necessário definir algumas recomendações. Nomeadamente, a calibração da câmara (a ser implementada no VANT) com base no algoritmo proposto por Sturm (1999) e através da função da biblioteca *cv2.calibrateCamera* (Bradski, 2013); a implementação de filtragem temporal com base num filtro *Kalman unscented* que permite solucionar a relação não-linear entre a pose estimada e medição prevista da pose (Julier, 1997) (Janabi-Sharif, 2010)

(Santos, 2014); quanto à otimização da deteção da marca visual de aterragem, a implementação de um algoritmo de deteção da marca visual convencional (“H” inserido numa circunferência) encontra-se implementado por Sanchez-Lopez (2013), no âmbito da aterragem em navios, servindo como um ponto de partida para o desenvolvimento de um algoritmo mais robusto. Outra alternativa para o algoritmo atual de deteção da marca visual baseia-se na implementação do código de edição-livre *OpenTLD*, que demonstra resultados promissores na área de processamento de imagens para a deteção e identificação de objetos (Nebehay, 2012).

Apesar da maioria das limitações serem relativas à funcionalidade adicionada de aterragem visual autónoma, os resultados obtidos foram positivos e promissores, sendo cumpridos os objetivos secundários e o objetivo principal proposto, que corresponde à criação de um controlador externo que amplifica e expande as capacidades de um VANT, com recurso a uma norma de referência (ROS), promovendo a interoperabilidade de sistemas.

O produto final desta dissertação resulta num estudo de interoperabilidade, no âmbito de VNTs (em particular VANTs), demonstrado através do desenvolvimento prático de um sistema de controlo externo para um piloto automático. Este estudo contribui diretamente para o desenvolvimento de VNTs na Marinha Portuguesa, apelando à criação de sistemas de controlo externo, interoperáveis, desenvolvidos em ROS. Por sua vez, o desenvolvimento desta tecnologia apela à investigação de VNTs no âmbito da Marinha Portuguesa, promovendo benefícios associados à integração de VNTs como uma capacidade operacional.

Trabalho Futuro

A propostas de trabalho futuro, baseado no trabalho desenvolvido nesta dissertação, consistem nas seguintes sugestões:

- Implementação do controlador num *Raspberry Pi*, com o objetivo de testar as funcionalidades do controlador num sistema de um VANT real, a bordo de um navio de guerra;
- Adição de novas funcionalidades ao controlador, através de pacotes ROS, como por exemplo um tradutor entre protocolos de comunicação distintos e integração de *software* que permita a operação de novos sensores integrados no VANT;
- Implementação de sistemas de controlo externos noutro tipo de VNTs, desenvolvidos em ROS e no âmbito da Marinha Portuguesa, com o objetivo de integrar funcionalidades comuns e partilhadas entre diversos VNTs, promovendo a interoperabilidade de sistemas;

Bibliografia

AERO DRUM. (2016). RC Blimps - Aerial Surveillance and Photography Systems. Obtido 30 de Maio de 2017, de http://rc-zeppelin.com/outdoor-rc-blimps.html

AUSTIN, R. (2010). *Unmanned Aircraft Systems: UAV Desing, Development and Deployment*. (I. Moir, A. Seabridge, & R. Langton, Eds.), *Aerospace Series* (first edit). John Wiley and Sons Ltd. Obtido de http://kms1.isn.ethz.ch/serviceengine/Files/ISN/122037/ichaptersection_singledocument/e961ebb2-ae79-4d3f-b085-340a2cb43e2f/en/Chapt.4.pdf

BENDEA, H., BOCCARDO, P., DEQUAL, S., TONOLO, F. G., MARENCHINO, D., e PIRAS, M. (2008). Low cost UAV for post-disaster assessment. *Proceedings of The XXI Congress of the International Society for Photogrammetry and Remote Sensing Beijing China 311 July 2008*, vol.*XXXVII*, pp. 1373–1380. Obtido de http://www.isprs.org/congresses/beijing2008/proceedings/8_pdf/14_ThS-20/37.pdf

BLOOMFIELD, R. S. (1999). Institute for Telecommunication Sciences - Online Telecom Glossary and other publications. Obtido de https://www.its.bldrdoc.gov/pubs/pubs.php

BORSHCHOVA, IRYNA; O'YOUNG, S. (2016). Visual servoing for autonomous landing of a multi-rotor UAS on a moving platform. *Journal of Unmanned Vehicle Systems, Canadian Science Publishing*, vol.*172*, no.1, pp. 1–78.

BRADSKI, G., e KAEHLER, A. (2013). *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, Inc.

BUTCHER, N., e STEWART, A. (2014). *Securing the MAVLink Communication Protocol for Unmanned Aircraft Systems*. Auburn University.

CAI, G., CHE, B. M., e LEE, T. H. (2011). *Unmanned Rotorcraft Systems* (1ª Edição). London: Springer-Verlag London.

CANNY, J. (1986). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.*8*, no.6, pp. 679–698.

CARREIRA, T. G. (2013). *Quadcopter Automatic Landing on a Docking Station*. Instituto Superior Técnico.

CESETTI, A., FRONTONI, E., MANCINI, A., ZINGARETTI, P., e LONGHI, S. (2010). A Vision-Based Guidance System for UAV Navigation and Safe Landing using Natural Landmarks. *Journal of Intelligent and Robotic Systems*, vol.*57*, pp. 233–257. URL: https://doi.org/10.1007/s10846-009-9373-3

CHAVES, S. M., WOLCOTT, R. W., e EUSTICE, R. M. (2015). NEEC Research: Toward GPS-denied Landing of Unmanned Aerial Vehicles on Ships at Sea. *Naval Engineering Journal*, vol.*127*, pp. 23–35.

CHROBOCINSKI, P., MAKRI, E., ZOTOS, N., STERGIOPOULOS, C., e BOGDOS, G. (2012). DARIUS project: Deployable SAR integrated chain with unmanned systems. *2012 International Conference on Telecommunications and Multimedia, TEMU 2012*, pp. 220–226. URL: https://doi.org/10.1109/TEMU.2012.6294722

CLAPPER, J. R., YOUNG, J. J., CARTWRIGHT, J. E., e GRIMES, J. G. (2007). Unmanned Systems Roadmap (2007-2032).

COOMBES, M., MCAEE, O., CHEN, W.-H., e RENDER, P. (2012). Development of an autopilot system for rapid prototyping of high level control algorithms. Em *Proceedings of the 2012 UKACC International Conference on Control* (pp. 292–297). Cardiff.

COSIC, J., CURKOVIC, P., KASAC, J., e STEPANIC, J. (2013). Interpreting Development of Unmanned Aerial Vehicles using Systems Thinking. *Interdisciplinary Description of Complex Systems*, vol.*11*, no.1, pp. 143–152. URL: https://doi.org/10.7906/indecs.11.1.12

DEFENSE, U.-D. OF. (2013). Unmanned Systems Integrated Roadmap. *Department of Defense*, no.2038, pp. 1–168.

EISENBEISS, H. (2004). A Mini Unmanned Aerial Vehicle (UAV): System overview and image acquisition. Em *International Workshop on «Processing and Visualization using High-Resolution Imagery»*. Pitsanulok, Thailand.

ELKAIM, G. H. (2012). Principles of Guidance , Navigation and Control of UAVs, pp. 1–30.

EMEPC - ESTRUTURA DE MISSÃO PARA A EXTENSÃO DA PLATAFORMA CONTINENTAL. (2009). *Continental Shelf Submission of Portugal*.

EMEPC - ESTRUTURA DE MISSÃO PARA A EXTENSÃO DA PLATAFORMA CONTINENTAL. (2017). Mapa «Portugal é Mar». Obtido 30 de Maio de 2017, de https://www.emepc.pt/pt/kit-do-mar/projetos/mapa

ERIKSSON, M., e RINGMAN, P. (2013). *Launch and recovery systems for unmanned vehicles onboard ships . A study and initial concepts*. KTH - Vetenskap Och Konst.

ERMAKOV, V. (2014). mavros - Package Summary. Obtido 25 de Maio de 2017, de http://wiki.ros.org/mavros#mavros.2BAC8-Plugins.setpoint_position

ESWAR, S. (2015). *Noise Reduction and Image Smoothing Using Gaussian Blur*. California State University Northridge.

FENG, D., XU, G., WANG, B., TIAN, Y., e YE, Y. (2013). Optimum Design of the Cooperation Objective for Computer Vision-based UAV autonomous Landing. *Advanced Materials Research*, vol.*720*, pp. 1221–1227. URL: https://doi.org/10.4028/www.scientific.net/AMR.718-720.1221

FLEMING, M., BRANNEN, S., MOSHER, A., ALTMIRE, B., METRICK, A., BOYLE, M., e SAY, R. (2015). *Unmanned Systems in Homeland Security*. Falls Church, Vancouver.

FLETCHER, B. (2000). Autonomous Vehicles and the Net-Centric Battlespace. Em *International Unmanned Undersea Vehicle Symposium*. Newport.

FORSYTH, D. A., e PONCE, J. (2002). *Computer Vision: A Modern Approach* (1ª Edição). Prentice Hall.

FULLER, B., KOK, J., KELSON, N., e GONZALEZ, F. (2014). Hardware Design and Implementation of a MAVLink Interface for an FPGA-Based Autonomous UAV Flight Control System, pp. 2–4.

GALLEGO, G., e YEZZI, A. (2015). A compact formula for the derivative of a 3-D rotation in exponential coordinates. *Journal of Mathematical Imaging and Vision*, vol.*51*,no.3, pp. 378–384.

GARRATT, M., POTA, H., LAMBERT, A., ECKERSLEY-MASLIN, S., e FARABET, C. (2009). Visual Tracking and LIDAR Relative Positioning for Automated Launch and Recovery of an Unmanned Rotorcraft from Ships at Sea. *American Society of Naval Engineers*. URL: https://doi.org/10.1111/j.1559-3584.2009.00194.x

GRIFFIS, C., e SCHNEIDER, J. (2009). *Unmanned Aircraft System Propulsion Systems Technology Survey*. Embry-Riddle Aeronautical University.

GUPTA, S. G., GHONGE, M. M., e JAWANDHIYA, P. M. (2013). Review of Unmanned Aircraft System. *International Journal of Advanced Research in Computer Engineering & Technology*, vol.*2*,no.4, pp. 2278–1323.

HARTLEY, R., e ZISSERMAN, A. (2004). *Multiple View Geometry in Computer Vision* (Segunda). Cambridge University Press.

HEIGHT TECH. (2016). HT-8 C180. Obtido de http://heighttech.com/en/products/ht-8-c180/

HIMSS. (2016). What is Interoperability. Obtido 3 de Setembro de 2016, de http://www.himss.org/library/interoperability-standards/what-is-interoperability

HUANG, H.-M. (2008). Autonomy Levels for Unmanned Systems (ALFUS) Framework Volume I : Terminology National Institute of Standards and Technology. *Framework*, vol.*1*, pp. 1–46.

IEEE. (1990). *IEEE Standard Glossary of Software Engineering Terminology*. Standards Coordinating Comitee of the Computer Society of IEEE - Standards Board.

JANABI-SHARIF, F., e MAREY, M. (2010). A Kalman-Filter-Based Method for Pose Estimation in Visual Servoing. *IEEE TRANSACTIONS ON ROBOTICS*, vol.*26*,no.5, pp. 939–947.

JOVANOVIC, M., e STARCEVIC, D. (2008). Software Architecture for Ground Control Station for Unmanned Aerial Vehicle. Em *Proc. 10th IEEE Int. Conf. on Computer Modeling and Simulation*. Cambridge, UK.

JULIER, S. J., e UHLMANN, J. K. (1997). A New Extension of the Kalman Filter to Nonlinear Systems. Em *Proc. of AeroSense: The 11th Int. Symp. on Aerospace/Defence Sensing, Simulation and Controls*.

KING, J. (2016). *Visual Servoing for a Quadrotor Unmanned Aerial Vehicle (UAV)*. University of Toronto Institute for Aerospace Studies.

KOCH, V. (2016). Rotations in 3D Graphics and the Gimbal Lock. *IEEE Okanagan*. Okanag.

KOWALK, W. (2006). *CRC Cyclic Redundancy Check Analysing and Correcting Errors*. Universität Oldenburg Fachbereich Informatik.

KUIPERS, J. B. (1999). *Quaternions and Rotation Sequences*. Princeton: Princeton University Press.

LEE, D., RYAN, T., e KIM, H. J. (2012). Autonomous Landing of a VTOL UAV on a Moving Platform Using Image-based Visual Servoing. Em *2012 IEEE International Conference on Robotics and Automation* (pp. 3–8). Minnesota.

LEITNER, J. (2015). ROS Hands-On Intro/Tutorial. Em *Robotic Vision Summer School 2015*.

LEPETIT, V., e PASCAL, F. M. (2009). EPnP: An Accurate O(n) Solution to the PnP Problem. *International Journal Of Computer Vision*, vol.*81*, pp. 155–166.

LI, R., LIU, J., ZHANG, Y., e HANG, Y. (2014). LIDAR/MEMS IMU integrated navigation (SLAM) method for a small UAV in indoor environments. Em *014 DGON Inertial Sensors and Systems (ISS)* (pp. 1–15). Karlsruhe.

LUGO, J. (2011). *Autonomous Landing of a Quadrotor UAV Using Vision and Infrared*

Markers for Pose Estimation. Technische Universität Dortmund.

MAHONY, R., KUMAR, V., e CORKE, P. (2012). Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *Robotics Automation Magazine - IEEE*, vol.*3*,no.19, pp. 20–32.

MARINHA PORTUGUESA. (2010). *Portugal, uma nação marítima*.

MARQUES, M. (2015). STANAG 4586 – Standard Interfaces of UAV Control System (UCS) for NATO UAV Interoperability. *NATO Science & Technology Organization*, pp. 1–14.

MARQUES, M. M., LOBO, V., NUNES, M. DE F., BATISTA, R., ALMEIDA, J., RIBEIRO, R., e BERNARDINO, A. (2016a). Oil Spills Detection: Challenges addressed in the scope of the SEAGULL project. Em *MTS/IEEE OCEANS 2016* (pp. 1–6). Monterey.

MARQUES, M. M., PARREIRA, R., LOBO, V., MARTINS, A., MATOS, A., CRUZ, N., ALMEIDA, J. M., ALVES, J. C., SILVA, E., BEDKOWSKI, J., MAJEK, K., PELKA, M., MUSIALIK, P., FERREIRA, H., DIAS, A., FERREIRA, B., AMARAL, G., FIGUEIREDO, A., ALMEIDA, R., SILVA, F., SERRANO, D., MORENO, G., DE CUBBER, G., BALTA, H., e BEGLEROVIC, H. (2016b). Use of multi-domain robots in search and rescue operations - Contributions of the ICARUS team to the euRathlon 2015 challenge. Em *MTS/IEEE OCEANS 2016 - Shanghai* (pp. 1–7). Shanghai. URL: https://doi.org/10.1109/OCEANSAP.2016.7485354

MCKEEGAN, N. (2011). Variable-wing prototype points to the future of UAVs. Obtido 30 de Maio de 2017, de http://newatlas.com/ucsd-flapping-wing-uav/18809/

MELLINGER, D., MICHAEL, N., e KUMAR, V. (2010). Trajectory generation and control for precise aggressive maneuvers with quadrotor. Em *Proceedings of the International Symposium on Experimental Robotics*.

NATO. (2006). Interoperability for joint operations, pp. 1-9. Obtido de http://www.nato.int/docu/interoperability/interoperability.pdf

NEBEHAY, G. (2012). *Robust Object Tracking Based on Tracking-Learning-Detection*.

Technischen Universität Wien.

NEHME, C., CRANDALL, J. W., e CUMMINGS, M. L. (2006). An Operator Function Taxonomy for Unmanned Aerial Vehicle Missions. Em *12TH ICCRTS «Adapting C2 to the 21st Century» An*. Cambridge, UK.

NEWS, S. W. (2016). Hydroid Introduces New Generation REMUS 100 AUV. Obtido 30 de Maio de 2017, de http://subseaworldnews.com/2016/03/14/hydroid-introduces-new-generation-remus-100-auv/

NORTHROP GRUMMAN. (2017). Fire Scout Unprecedented Persistent Situational Awareness. Obtido 30 de Maio de 2017, de http://www.northropgrumman.com/Capabilities/FireScout/Pages/default.aspx?utm_source=PrintAd&utm_medium=Redirect&utm_campaign=FireScout+Redirect

PAGANO, P., CANDELA, L., e CASTELLI, D. (2013). Data Interoperability. *Data Science Journal*, vol.*12*, pp. 4.

PEFFERS, K., TUUNANEN, T., ROTHENBERGER, M., e CHATTERJEE, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems 24 (3)*, pp. 45–77.

PERVIN, E., e WEBB, J. A. (1982). *Quaternions in computer vision and robotics*. Carnegie-Mellon University.

PESSOA, F. (1934). Mar Português. Em *Mensagem*. Lisboa: Parceria António Maria Pereira.

PI, E. R. (2015). *Implementation of a 3D pose estimation algorithm*. Universitat Politecnica de Catalunya.

PRINCE, S. J. (2012). *Computer vision: models, learning, and inference*. Cambridge University Press.

QINETIC. (2016). Dragon Runner. Obtido 30 de Maio de 2016, de https://www.qinetiq-na.com/products/unmanned-systems/dragon-runner/

QUIGLEY, M., GERKEY, B., e SMART, W. D. (2015). *Programming Robots with ROS A Practical Introduction to the Robot Operating System*. *Journal of Chemical Information and Modeling* (Vol. 53). URL: https://doi.org/10.1017/CBO9781107415324.004

RAZA, S., e GUEAIEB, W. (2010). Intelligent flight control of an autonomous quadrotor. *Robotics - Motion Control*.

REKER, N., TROXELL, D., e TROY, D. (2015). *Universal UAV Payload Interface*. California Polytechnic State University. Obtido de http://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1173&context=cpesp

ROMERO, A. M. (2014). ROS: Concepts. Obtido 22 de Maio de 2017, de http://wiki.ros.org/ROS/Concepts

ROSA, G. C., MARQUES, M. M., e LOBO, V. (2016). Unmanned Aerial Vehicles in the Navy: Its Benefits. *Scientific Bulletin of Naval Academy*, vol.*19*,no.1, pp. 39–43. URL: https://doi.org/10.21279/1454-864X-16-I1-007

ROYAL AUSTRALIAN AIR FORCE. (2017). Heron. Obtido de http://www.airforce.gov.au/Technology/Aircraft/Heron/?RAAF-U3cQ7cNqUl7hOR9akHK4KUQKnbbWmZnX

SANCHEZ-LOPEZ, J. (2013). Toward visual autonomous ship board landing of a vtol uav. Em *International Conference on Unmanned Aircraft Systems (ICUAS)* (pp. 779–788). Atlanta. URL: https://doi.org/10.1109/ICUAS.2013.6564760

SANTOS, N. P. (2014). *Sistema de Visão para Aterragem Automática de UAVs*. Instituto Superior de Engenharia de Lisboa.

SERRANO, D., CHROBOCINSKI, P., CUBBER, G. DE, MOORE, D., LEVENTAKIS, G., e GOVINDARAJ, S. (2015). ICARUS and DARIUS approaches towards interoperability - Two complementary projects that cover the full spectrum of interoperability issues for the integration of unmanned platforms in Search and Rescue operations. Em *TARP*

RISE Workshop, Proceedings of the NATO STO Lecture Series SCI-271. Lisbon.

SKRZYPIETZ, B. T. (2012). Unmanned Aircraft Systems for Civilian Missions. *Brandenburg Institute for Society and Security Policy Paper*,no.1, pp. 1–28.

STEVENS, B. L., e LEWIS, F. L. (2003). *Aircraft control and simulation* (Segunda). John Wiley & Sons.

STURM, P. F., e MAYBANK, S. J. (1999). On plane-based camera calibration: A general algorithm, singularities, applications. Em *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

SUNNY. (2014). About SUNNY: Overview. Obtido 10 de Março de 2017, de http://www.sunnyproject.eu/about.aspx#overview

SUZUKI, S. (1985). Topological structural analysis of digitized binary images by border following. Em *Computer Vision, Graphics, and Image Processing* (1st ed., pp. 32–46).

UPDATE, D. (2006). Silver Marlin Autonomous Unmanned Surface Vehicle (USV) Elbit Systems.

WADE, R. (2006). Joint architecture for unmanned systems. Em *44th Annual Targets, UAVs & Range Operations Symposium & Exhibition*.

WEAVER, J. N., FRANK, D. Z., SCHWARTZ, E. M., e ARROYO, A. A. (2013). Uav Performing Autonomous Landing on USV Utilizing the Robot Operating System. *ASME District F - Early Career Technical Conference (ECTC) 2013*, vol.*12*, pp. 119–124. Obtido de http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.401.2520

WZOREK, M., LAND, D., DOHERTY, P., LINK, S.-, e LAND, D. (2006). GSM Technology as a Communication Media for an Autonomous Unmanned Aerial Vehicle. Em *Proceedings of the 21st Bristol International UAV Systems Conference (UAVS)* (p. 41). Bristol.

XIE, J., AL-EMRANI, F., GU, Y., WAN, Y., e FU, S. (2016). UAV-carried Long-distance Wi-Fi

Communication Infrastructure. Em *AIAA Infotech @ Aerospace, AIAA SciTech Forum*.

Apêndice A – *main_controller*

```
#!/usr/bin/env python
# ============================================
# main_controller.py
#
# Permite ao operador a transmissão de comandos de voo
# para o piloto automático, e a inicialização da aterragem visual
autónoma
# ============================================

# Dependencias
#=========================================================
import os
import time
import sys
import fileinput

import rospy
import rospkg
import mavros

from python_qt_binding import loadUi, QtWidgets
from python_qt_binding.QtCore import Qt, qWarning, Signal
from python_qt_binding.QtWidgets import QMainWindow, QFrame

from mavros_msgs.msg import WaypointList

from flight_command_node import setFlightMode, setLandMode,
setArmMode, setDisarmMode, setTakeoffMode, setShowWPs,
globalPositionCallback, getWPs, setLoadWPs

from subprocess import call
#=========================================================

# Classe e Funcoes que definem a GUI utilizada para controlar o
sistema

class MyWindow(QMainWindow):
    def __init__(self):
        super(MyWindow, self).__init__()
        rp = rospkg.RosPack()
        ui_file = os.path.join(rp.get_path('image_conv'), 'resource',
'mainwindow.ui')
    loadUi(ui_file, self)
        self.show()


self.arm_check.clicked[bool].connect(self._handle_arm_check_clicked)

self.takeoff_button.clicked[bool].connect(self._handle_takeoff_button_
clicked)

self.land_button.clicked[bool].connect(self._handle_land_button_clicke
d)
```

```
self.set_flight_mode_button.clicked[bool].connect(self._handle_set_fli
ght_mode_button_clicked)

self.print_wp_button.clicked[bool].connect(self._handle_print_wp_butto
n_clicked)

self.start_mission_button.clicked[bool].connect(self._handle_start_mis
sion_button_clicked)

self.init_visual_landing_button.clicked[bool].connect(self._handle_ini
t_visual_landing_button_clicked)

self.load_wp_button.clicked[bool].connect(self._handle_load_wp_button_
clicked)

self.insert_wp_button.clicked[bool].connect(self._handle_insert_wp_but
ton_clicked)


    def _handle_arm_check_clicked(self, checked):
    if checked:
        setArmMode()
        self.log_text_browser.append("Throttle armed")
    else:
        setDisarmMode()
        self.log_text_browser.append("Throttle disarmed")

    def _handle_takeoff_button_clicked(self, checked):
    start_time = time.time()
    altitude = float(self.takeoff_input.text())
    setTakeoffMode(altitude)
    self.log_text_browser.append("Takeoff successful. Taking off up to
" + str(altitude) + " meters")
    print ("takeoff - tempo de execucao: %s segundos" % (time.time() -
start_time))

    def _handle_land_button_clicked(self, checked):
    start_time = time.time()
    setLandMode()
    self.log_text_browser.append("Land mode set")
    print ("land - tempo de execucao: %s segundos" % (time.time() -
start_time))

    def _handle_set_flight_mode_button_clicked(self, checked):
    start_time = time.time()
    mode = str(self.flight_mode.currentText())
    setFlightMode(mode)
    self.log_text_browser.append(mode + " mode set")
    print ("flightmode - tempo de execucao: %s segundos" %
(time.time() - start_time))

    def _handle_print_wp_button_clicked(self, checked):
    start_time = time.time()
    setShowWPs()

    self.waypoint_text_browser.setPlainText(" ")
```

```
    rp = rospkg.RosPack()
    wps_file = os.path.join(rp.get_path('image_conv'), 'resource',
'waypoint_list_2.txt')

    while True:
        try:
                txt = open(wps_file)
        except IOError:
            self.log_text_browser.append("Ficheiro nao foi
encontrado")
            continue
        break

    txt_array = txt.readlines()
    new_txt_array = txt_array

    for i, line in enumerate(txt_array):

        line = line.split()
        new_txt_array[0] = "Waypoint List: "
        if len(line) > 3:

            if(float(line[8]) < 0):
                hem_sphere = " S"
            elif(float(line[8])  > 0):
                hem_sphere = " N"
            else:
                hem_sphere = "  "

            if(float(line[9])  < 0):
                merid = " W"
            elif(float(line[9])  > 0):
                merid = " E"
            else:
                merid = " "
            line[8] = abs(float(line[8]))
            line[9] = abs(float(line[9]))

            new_txt_array[i] = "WP" + str(line[0]) + ": Lat = "+
str(line[8]) + hem_sphere + ", Long = " + str(line[9]) + merid + ",
Altitude = " + str(line[10])
            self.waypoint_text_browser.append(str(new_txt_array[i]))

    print ("printwp - tempo de execucao: %s segundos" % (time.time() -
start_time))

    def _handle_start_mission_button_clicked (self, checked):
    setFlightMode("GUIDED")
    self.log_text_browser.append("GUIDED mode set")
    time.sleep(1)

    setArmMode()
    self.log_text_browser.append("Throttle armed")
    time.sleep(1)

    setTakeoffMode(10.0)
```

```
        self.log_text_browser.append("Begining Takeoff")
        time.sleep(4)

        setFlightMode("AUTO")
        self.log_text_browser.append("AUTO mode set")

        self.log_text_browser.append("Mission start was succesful.
Following mission plan...")

    def _handle_init_visual_landing_button_clicked(self, checked):
        setFlightMode("GUIDED")
        call(["roslaunch", "image_conv", "visual_landing.launch"])
        self.log_text_browser.append("Started Visual Landing...")

    def _handle_load_wp_button_clicked(self, checked):
        start_time = time.time()
        setLoadWPs()
        self.log_text_browser.append("Waypoint list successfully loaded!")
        print ("wpload - tempo de execucao: %s segundos" % (time.time() -
start_time))

    def _handle_N_S_box(self, _latitude):

        pos = str(self.N_S_box.currentText())

        if (pos == "N"):
            _longitude = abs(_latitude)
        elif(pos == "S"):
            _longitude = - _latitude

        return _latitude

    def _handle_E_W_box(self, _longitude):

        pos = str(self.E_W_box.currentText())

        if(pos == "E"):
            _longitude = abs(_longitude)
        elif(pos == "W"):
            _longitude = - _longitude

        return _longitude

    def _handle_insert_wp_button_clicked(self, checked):
        start_time = time.time()
        latitude = float(self.lat_line_edit.text())
        longitude = float(self.long_line_edit.text())
        altitude = int(self.alt_line_edit.text())

        latitude = self._handle_N_S_box(latitude)
        longitude = self._handle_E_W_box(longitude)

        rp = rospkg.RosPack()
        wps_file_path = os.path.join(rp.get_path('image_conv'),
'resource', 'waypoint_list_2.txt')

        with open(wps_file_path, 'r') as wps_file:
```

```
        for i, lines in enumerate(wps_file):
                pass
        index = i
    wps_file.close()

    with open(wps_file_path, 'a') as wps_file:

        line = str(index) + "\t" + str(0) + "\t" + str(3) + "\t" +
str(16) + "\t" + str(0) +"\t"+ str(0) +"\t"+ str(0) +"\t"+ str(0)
+"\t"+ str(latitude)+ "\t" + str(longitude) +"\t"+ str(altitude) +
"\t" +  str(1) + "\n"
            wps_file.write(line)
    wps_file.close()


    self.log_text_browser.append("Waypoint inserted")
    print ("insertwp - tempo de execucao: %s segundos" % (time.time()
- start_time))

# Funcao main=============================================================
if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    window = MyWindow()
    sys.exit(app.exec_())
```

Apêndice B – *flight_command_node*

```
#!/usr/bin/env python
# ============================================
# flight_command_node.py
#
# Script que integra as funcoes do no main_controller.py
# ============================================

# Dependencias
#=========================================================
import os
import rospy
import rospkg
import mavros

from std_msgs.msg import String
from sensor_msgs.msg import NavSatFix
from mavros_msgs.srv import *
from mavros_msgs.msg import WaypointList
from subprocess import call
#=========================================================
# variavel global
latitude =0.0
longitude=0.0

# Funcao que escolhe o modo de voo
def setFlightMode(mode):
    rospy.wait_for_service('/mavros/set_mode')
    try:
        flightModeService = rospy.ServiceProxy('/mavros/set_mode',
mavros_msgs.srv.SetMode)
        isModeChanged = flightModeService(custom_mode=mode)
    except rospy.ServiceException, e:
        print "service set_mode call failed: %s. %s Mode could not be
set. Check that GPS is enabled" %(e, mode)

# Funcao que faz aterragem
def setLandMode():
    rospy.wait_for_service('/mavros/cmd/land')
    try:
        landService = rospy.ServiceProxy('/mavros/cmd/land',
mavros_msgs.srv.CommandTOL)
        isLanding = landService(altitude = 0, latitude = 0, longitude
= 0, min_pitch = 0, yaw = 0)
    except rospy.ServiceException, e:
        print "service land call failed: %s. The vehicle cannot land
"%e

# Funcao que arma o veiculo
def setArmMode():
    rospy.wait_for_service('/mavros/cmd/arming')
    try:
        armService = rospy.ServiceProxy('/mavros/cmd/arming',
mavros_msgs.srv.CommandBool)
```

```
        armService(True)
    except rospy.ServiceException, e:
        print "Service arm call failed: %s"%e

# Funcao que desarma o veiculo
def setDisarmMode():
    rospy.wait_for_service('/mavros/cmd/arming')
    try:
        armService = rospy.ServiceProxy('/mavros/cmd/arming',
mavros_msgs.srv.CommandBool)
        armService(False)
    except rospy.ServiceException, e:
        print "Service arm call failed: %s"%e

# Funcao que descola o veiculo
def setTakeoffMode(altitude):
    rospy.wait_for_service('/mavros/cmd/takeoff')
    try:
        takeoffService = rospy.ServiceProxy('/mavros/cmd/takeoff',
mavros_msgs.srv.CommandTOL)
        takeoffService(altitude = float(altitude), latitude = 0,
longitude = 0, min_pitch = 0, yaw = 0)
    except rospy.ServiceException, e:
        print "Service takeoff call failed: %s"%e

# Funcao que mostra os waypoints
def setShowWPs():
    try:
    rp = rospkg.RosPack()
        wps_file = os.path.join(rp.get_path('image_conv'), 'resource',
'waypoint_list_2.txt')
        open(wps_file, 'w').close()

    call(["rosrun", "mavros", "mavwp", "show"])
    call(["rosrun", "mavros", "mavwp", "dump", wps_file])
    except rospy.ServiceException, e:
        print "Service takeoff call failed: %s"%e

# Funcao que apaga os waypoints do piloto automatico
def setClearWPs():
    rospy.wait_for_service('/mavros/mission/clear')
    try:
    clearWPsService = rospy.ServiceProxy('/mavros/mission/clear',
mavros_msgs.srv.WaypointClear)
    clearWPsService()
    except rospy.ServiceException, e:
        print "Service takeoff call failed: %s"%e

# Funcao que carrega os waypoints no piloto automatico
def setLoadWPs():
    setClearWPs()

    try:
    rp = rospkg.RosPack()
        wps_file = os.path.join(rp.get_path('image_conv'), 'resource',
'waypoint_list_2.txt')
    print (wps_file)
```

```
    call(["rosrun", "mavros", "mavwp", "load", wps_file])
    except rospy.ServiceException, e:
        print "Service takeoff call failed: %s"%e

# Funcao que mostra waypoints
def getWPs():
    global waypoint_list
    return waypoint_list

# Funcao que mostra localizacao GPS
def globalPositionCallback(globalPositionCallback):
    global latitude
    global longitude
    latitude = globalPositionCallback.latitude
    longitude = globalPositionCallback.longitude


# Funcao main=================================================
if __name__ == '__main__':
    rospy.init_node('flight_command_node', anonymous=True)
    rospy.Subscriber("/mavros/global_position/raw/fix", NavSatFix,
globalPositionCallback)

    myLoop()
```

Apêndice C – *detect_marker*

```
#!/usr/bin/env python
# ============================================
# detect_marker.py
#
# Programa que deteta e analisa contornos de objetos presentes nas
imagens provenientes do no usb_cam,
# de modo a detetar a marca visual de aterragem, e a calcular as suas
respetivas coordenadas
# no plano da imagem, expressas em pixeis.
# ============================================

# Dependencias
#=========================================================
import os
import rospkg
import argparse
import numpy as np
import cv2
import rospy
from std_msgs.msg import Header
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import time
from image_conv.msg import Coordinates
from geometry_msgs.msg import Quaternion, Point, PoseStamped, Pose
import math
#=========================================================

def init_detect_marker():
    rospy.init_node('detect_marker', anonymous=True)
    rospy.Subscriber("/usb_cam/image_raw", Image, detect_marker)
    rospy.spin()

def detect_marker(camera):
    pub = rospy.Publisher('target_coordinates', Coordinates,
queue_size=1)
    coordinates = Coordinates()
    frame = bridge.imgmsg_to_cv2(camera, desired_encoding="bgra8")
    status = "No Targets"

    output = frame.copy()

    # Pre-processamento de imagem
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (7, 7), 0)
    edged = cv2.Canny(blurred, 50, 150)

    # Encontra contornos
    cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[-2]

    cv2.line(frame, (310, 240), (330, 240), (0,0,255), 2)
    cv2.line(frame, (320, 230), (320, 250), (0,0,255), 2)
#8
```

```
    cv2.line(frame, (310, 120), (330, 120), (0,0,255), 2)
    cv2.line(frame, (320, 110), (320, 130), (0,0,255), 2)
#2
    cv2.line(frame, (310, 360), (330, 360), (0,0,255), 2)
    cv2.line(frame, (320, 350), (320, 370), (0,0,255), 2)
#7
    cv2.line(frame, (150, 120), (170, 120), (0,0,255), 2)
    cv2.line(frame, (160, 110), (160, 130), (0,0,255), 2)

#4
    cv2.line(frame, (150, 240), (170, 240), (0,0,255), 2)
    cv2.line(frame, (160, 230), (160, 250), (0,0,255), 2)
#1
    cv2.line(frame, (150, 360), (170, 360), (0,0,255), 2)
    cv2.line(frame, (160, 350), (160, 370), (0,0,255), 2)
#9
    cv2.line(frame, (470, 120), (490, 120), (0,0,255), 2)
    cv2.line(frame, (480, 110), (480, 130), (0,0,255), 2)
#6
    cv2.line(frame, (470, 240), (490, 240), (0,0,255), 2)
    cv2.line(frame, (480, 230), (480, 250), (0,0,255), 2)
#3
    cv2.line(frame, (470, 360), (490, 360), (0,0,255), 2)
    cv2.line(frame, (480, 350), (480, 370), (0,0,255), 2)

    # analisa cada contorno
    for c in cnts:
        # aproxima o contorno
        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.01 * peri, True)

        ((x, y), radius) = cv2.minEnclosingCircle(c)
        M = cv2.moments(c)


        # avança no caso do raio do circulo for superior ao estipulado
        if radius > 25:

            # cria uma bounding box para definir a relacao de aspeto
            (x, y, w, h) = cv2.boundingRect(approx)
            aspectRatio = w / float(h)

            # obtem a solidez do contorno original
            area = cv2.contourArea(c)
            hullArea = cv2.contourArea(cv2.convexHull(c))
            if float(hullArea) != 0:
                solidity = area / float(hullArea)

            keepDims = w > 100 and h > 100
            keepSolidity = solidity > 0.9
            keepAspectRatio = aspectRatio >= 0.8 and aspectRatio <=
1.2

            # testa o contorno
            if keepDims and keepSolidity and keepAspectRatio:

                cv2.drawContours(frame, [approx], -1, (0, 0, 255), 4)
```

```
                status = "Target Acquired"

                header.stamp = rospy.Time.now()
                header.frame_id = "usb_cam"

                coordinates.x = x
                coordinates.y = y
                coordinates.width = w
                coordinates.height = h
                coordinates.header = header

                pub.publish(coordinates)

                rospy.loginfo(coordinates)


#============================TESTE================================
                x1 = x
                y1 = y

                x2 = x + w
                y2 = y + h

                x0 = x + (w)/2
                y0 = y + (h)/2

                width = w/2
                heigth = h/2

                image_points = np.array([
                            (x1, y1),     # canto superior esquerdo
                            (x2, y1),     # canto superior direito
                            (x1, y2),     # canto inferior esquerdo
                            (x2, y2),     # canto inferior direito
                            (x0,y0),      # centro
                                ], dtype="double")

                    model_points = np.array([
                            (-0.083, 0.083, 0), # canto superior
esquerdo
                            (0.083, 0.083, 0),  # canto superior
direito
                            (-0.083, -0.083, 0),# canto inferior
esquerdo
                            (0.083, -0.083, 0), # canto inferior
direito
                            (0, 0, 0)          # centro

                                ])

                 # Parametros intrisecos da camera (distancias focais
e centro da imagem)
                    focal_length = 640
                    center = (640 / 2, 480 / 2)
                # Define a matriz de parametros intrisecos da camera
                    camera_matrix = np.array([[focal_length, 0,
center[0]], [0, focal_length, center[1]], [0, 0, 1]], dtype="double")
```

```
                # Assume distorcao focal como nula
                    dist_coeffs = np.zeros((4, 1))
                # solvePnP obtem as matrizes de rotacao e translacao
(pose) do alvo, relativas ao referencial da camera
                    (success, rotation_vector, translation_vector) =
cv2.solvePnP(model_points, image_points, camera_matrix, dist_coeffs,

cv2.SOLVEPNP_EPNP)

                    print "Target Pose"
                    print "> Rotation Vector:\n
{0}".format(rotation_vector)
                    print "> Translation Vector:\n
{0}".format(translation_vector)

################################Camera Coord
                (rotation_matrix,_) = cv2.Rodrigues(rotation_vector)

                # Aplicar a matriz transposta na matriz de rotacao
                new_rotation_matrix = rotation_matrix.transpose()
                (new_rotation_vector,_) =
cv2.Rodrigues(new_rotation_matrix)

                new_translation_vector = - np.dot(new_rotation_matrix,
translation_vector)


                print " Camera position (G frame):\n
{0}".format(new_translation_vector)

                file_path = os.path.join(rp.get_path('image_conv'),
'scripts', 'resultados.txt')
                euc_dist=
math.sqrt(new_translation_vector[0]*new_translation_vector[0] +
new_translation_vector[1]*new_translation_vector[1])
                with open(file_path, 'a') as f:

                    f.write("X= {0}
\n".format(new_translation_vector[0]) +"Y= {0}
\n".format(new_translation_vector[1])+"D= {0} \n".format(euc_dist) )

                f.close()



                g_frame_points = np.array([(0.1, 0.0, 0.0),(0.0, 0.1,
0.0),(0.0, 0.0, 0.1),(0.0, 0.0, 0.0)])
                (g_frame, jacobian) =
cv2.projectPoints(g_frame_points, rotation_vector, translation_vector,
camera_matrix, dist_coeffs)

                    p0 = (int(g_frame[3][0][0]),
int(g_frame[3][0][1]))
                    px = (int(g_frame[0][0][0]),
int(g_frame[0][0][1]))
```

```
                    py = (int(g_frame[1][0][0]),
int(g_frame[1][0][1]))
                    pz = (int(g_frame[2][0][0]),
int(g_frame[2][0][1]))


                    cv2.line(frame, p0, px, (255,0,0), 2) #X - AZUL
                cv2.putText(frame, "X", px, cv2.FONT_HERSHEY_SIMPLEX,
0.5,(255,0,0), 2)

                    cv2.line(frame, p0, py, (0,255,0), 2) #Y - VERDE
                cv2.putText(frame, "Y", py, cv2.FONT_HERSHEY_SIMPLEX,
0.5,(0,255,0), 2)

                    cv2.line(frame, p0, pz, (0,0,255), 2) #Z -
VERMELHO
                cv2.putText(frame, "Z", pz, cv2.FONT_HERSHEY_SIMPLEX,
0.5,(0,0,255), 2)
#================================TESTE=====================================

                # desenha no centro da marca

                M = cv2.moments(approx)
                (cX, cY) = (int(M["m10"] / M["m00"]), int(M["m01"] /
M["m00"]))
                (startX, endX) = (int(cX - (w * 0.15)), int(cX + (w *
0.15)))
                (startY, endY) = (int(cY - (h * 0.15)), int(cY + (h *
0.15)))

                cv2.line(frame, (0, 0), (endX-cX, 0), (255, 0, 255),
2)
                cv2.putText(frame, "u", (endX-cX, 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5,(255, 0, 255), 2)

                cv2.line(frame, (0, 0), (0, endY-cY), (0, 255, 255),
2)
                cv2.putText(frame, "v", (5, endY-cY),
cv2.FONT_HERSHEY_SIMPLEX, 0.5,(0, 255, 255), 2)

                cv2.line(frame, (310, 240), (330, 240), (0,0,255), 2)
                cv2.line(frame, (320, 230), (320, 250), (0,0,255), 2)

                cv2.putText(frame,"x =" +
str(new_translation_vector[0]), (5, 420), cv2.FONT_HERSHEY_SIMPLEX,
0.5,(255,0,0), 2)
                cv2.putText(frame,"y =" +
str(new_translation_vector[1]), (5, 440), cv2.FONT_HERSHEY_SIMPLEX,
0.5,(0,255,0), 2)
                cv2.putText(frame,"z =" +
str(new_translation_vector[2]), (5, 460), cv2.FONT_HERSHEY_SIMPLEX,
0.5,(0,0,255), 2)



    cv2.putText(frame, status, (20, 30), cv2.FONT_HERSHEY_SIMPLEX,
0.5,
```

```
            (0, 0, 255), 2)


        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF

        # para o loop quando se carrega na tecla 'q'
        if key == ord("q"):
            rospy.signal_shutdown("Processo terminado com sucesso!")
            print "Processo terminado com sucesso!"
# Funcao main=================================================
if __name__ == '__main__':
    try:
        rp = rospkg.RosPack()
        bridge = CvBridge()
        header = Header()
        init_detect_marker()
    except rospy.ROSInterruptException:
        pass
```

Apêndice D – *pose_estimation*

```
#!/usr/bin/env python
# ============================================
# pose_estimation.py
#
# Programa que recebe captura de video e coordenadas do alvo
detectado,
# por forma a determinar a pose do UAV e calcular as suas coordenadas
relativas ao alvo.
# De seguida envia as suas coordenadas e versores de rotacao para o
MAVROS (topico: /vision_pose_estimation)
# ============================================

# Dependencias
#=========================================================
import cv2
import numpy as np
import rospy
import math
import time
from cv_bridge import CvBridge
from sensor_msgs.msg import Image
from image_conv.msg import Coordinates
from geometry_msgs.msg import Quaternion, Point, PoseStamped, Pose
from std_msgs.msg import Header
#import message_filters
from message_filters import TimeSynchronizer, Subscriber,
ApproximateTimeSynchronizer
#=========================================================

# Funcoes
#=========================================================
def init_pose_estimation():
    # Inicia node
    rospy.init_node('pose_estimation', anonymous=True)
    #rospy.Subscriber("/usb_cam/image_raw", Image, pose_estimation)
    #  Subscricao a topicos de modo a receber informacao de outros
nodes
    image_sub = Subscriber("/usb_cam/image_raw", Image)
    coord_sub = Subscriber("/target_coordinates", Coordinates)

    # Sincroniza a recepcao dos dois topicos e procede a chamar a
funcao "pose_estimation"
    ts = ApproximateTimeSynchronizer([image_sub, coord_sub], 10,1)
    rospy.loginfo(ts.registerCallback(pose_estimation))
    ts.registerCallback(pose_estimation)

    # Impede o programa de encerrar ate ser determinado pelo
utilizador
    rospy.spin()

def pose_estimation(image,coord):
    # Escreve na consola o conteudo da variavel "coord", entre outras
informacoes
    rospy.loginfo(coord)
```

```
    # Converte imagens utilizadas pelo ROS para imagens utilizadas por
OpenCv
    im = bridge.imgmsg_to_cv2(image, desired_encoding="passthrough")
    # Obtem a resolucao da imagem
    size = im.shape

    # Coordenadas do centro e dos quatro cantos do alvo detectado, na
imagem
    x1 = coord.x
    y1 = coord.y

    x2 = coord.x + coord.width
    y2 = coord.y + coord.height

    x0 = (coord.x + coord.x + coord.width)/2
    y0 =(coord.y + coord.y + coord.height)/2

    width = coord.width/2
    heigth = coord.height/2

    # Guarda num vector as varias coordenadas do alvo
    image_points = np.array([
                                (x1, y1),     # canto superior
esquerdo
                                (x2, y1),     # canto superior direito
                                (x1, y2),     # canto inferior
esquerdo
                                (x2, y2),     # canto inferior direito
                                (x0,y0),      # centro
                            ], dtype="double")

    # Coordenadas 3D do modelo utilizado como alvo (em metros)
    model_points = np.array([
                                (-0.7, 0.155, 0), # canto superior
esquerdo
                                (0.7, 0.155, 0),  # canto superior
direito
                                (-0.7, -0.155, 0),# canto inferior
esquerdo
                                (0.7, -0.155, 0), # canto inferior
direito
                                (0, 0, 0)         # centro

                            ])

    # Parametros intrisecos da camera (distancias focais e centro da
imagem)
    focal_length = size[1]
    center = (size[1] / 2, size[0] / 2)
    # Define a matriz de parametros intrisecos da camera
    camera_matrix = np.array([[focal_length, 0, center[0]], [0,
focal_length, center[1]], [0, 0, 1]], dtype="double")

    print "Focal lenght is:", focal_length
    print "Size is:", size[1], size[0]
    print "Camera Matrix :\n {0}".format(camera_matrix)
```

```
    # Assume distorcao focal como nula
    dist_coeffs = np.zeros((4, 1))
    # solvePnP obtem as matrizes de rotacao e translacao (pose) do
alvo, relativas ao referencial da camera
    (success, rotation_vector, translation_vector) =
cv2.solvePnP(model_points, image_points, camera_matrix, dist_coeffs,

flags=cv2.SOLVEPNP_ITERATIVE)

    print "Target Pose"
    print "> Rotation Vector:\n {0}".format(rotation_vector)
    print "> Translation Vector:\n {0}".format(translation_vector)

    # Converte as matrizes de rotacao e translacao anteriores em
matrizes de rotacao e translacao relativas ao referencial do alvo
    (tc_rotation, tc_translation) =
target_to_camera_pose(rotation_vector,translation_vector)


    # Converte a matriz de rotacao num Quaternion porque apenas e
possivel a publicacao para MAVROS neste formato
    quaternion = euler_to_quaternion(tc_rotation)
    print "> Quaternion:\n {0}".format(quaternion)

    # Obtem as coordenadas da camera relativas ao referencial do alvo
(mapping from world coordinates to camera coordinates)
    camera_coordinates = get_camera_real_point(tc_rotation,
tc_translation)

    camera_point.x = camera_coordinates[0]
    camera_point.y = camera_coordinates[1]
    camera_point.z = camera_coordinates[2]

    print camera_point

    g_frame_points = np.array([(0.3, 0.0, 0.0),(0.0, 0.3, 0.0),(0.0,
0.0, 0.3),(0.0, 0.0, 0.0)])

    (g_frame, jacobian) = cv2.projectPoints(g_frame_points,
rotation_vector, translation_vector, camera_matrix, dist_coeffs)

    print "G_frame:\n {0}".format(g_frame)
    print int(g_frame[0][0][0])


    # Publica para MAVROS a posicao do UAV e respectivo Quaternion de
rotacao
    header.stamp = rospy.Time.now()
    header.frame_id = "uav_pose"

    pose.position = camera_point
    pose.orientation = quaternion

    uav_pose.pose = pose
    uav_pose.header = header
```

```
    pub = rospy.Publisher('/mavros/vision_pose/pose', PoseStamped,
queue_size=1)

    pub.publish(uav_pose)
    r = rospy.Rate(1)
    rospy.loginfo(uav_pose)
    r.sleep()

    p0 = (int(g_frame[3][0][0]), int(g_frame[3][0][1]))
    px = (int(g_frame[0][0][0]), int(g_frame[0][0][1]))
    py = (int(g_frame[1][0][0]), int(g_frame[1][0][1]))
    pz = (int(g_frame[2][0][0]), int(g_frame[2][0][1]))


    cv2.line(im, p0, px, (255,0,0), 2) #X - AZUL
    cv2.line(im, p0, py, (0,255,0), 2) #Y - VERDE
    cv2.line(im, p0, pz, (0,0,255), 2) #Z - VERMELHO

# Converte euler pose para quaternion pose
def euler_to_quaternion(rotation):

    roll = rotation[0]
    pitch = rotation[1]
    yaw = rotation[2]

    q0 = math.cos(yaw * 0.5)
    q1 = math.sin(yaw * 0.5)
    q2 = math.cos(roll * 0.5)
    q3 = math.sin(roll * 0.5)
    q4 = math.cos(pitch * 0.5)
    q5 = math.sin(pitch * 0.5)

    quat.w = q0 * q2 * q4 + q1 * q3 * q5
    quat.x = q0 * q3 * q4 - q1 * q2 * q5
    quat.y = q0 * q2 * q5 + q1 * q3 * q4
    quat.z = q1 * q2 * q4 - q0 * q3 * q5

    return quat

def target_to_camera_pose(rotation, translation):

    (rotation_matrix,_) = cv2.Rodrigues(rotation)

    # Aplicar a matriz transposta na matriz de rotacao
    new_rotation_matrix = rotation_matrix.transpose()
    (new_rotation_vector,_) = cv2.Rodrigues(new_rotation_matrix)

    new_translation_vector = - np.dot(new_rotation_matrix,
translation)

    print " Rotation Matrix:\n {0}".format(rotation_matrix)
    print " Translation Vector:\n {0}".format(new_translation_vector)

    return new_rotation_vector, new_translation_vector

def get_camera_real_point(rotation, translation):
    (rotation_matrix,_) = cv2.Rodrigues(rotation)
```

```
    # Aplicar a matriz transposta na matriz de rotacao
    new_rotation_matrix = rotation_matrix.transpose()

    camera_coordinates = translation

    print " Camera coordinates:\n {0}".format(camera_coordinates)

    return camera_coordinates

# Funcao main=================================================
if __name__ == '__main__':
    bridge = CvBridge()
    quat = Quaternion()
    camera_point = Point()
    pose = Pose()
    uav_pose = PoseStamped()
    header = Header()

    init_pose_estimation()
```

Apêndice E – *uav_landing*

```
#!/usr/bin/env python
# ================================================
# uav_landing.py
#
# executa o alinhamento do VANT com a marca visual de aterragem,
# e subsequente aterragem no centro da marca visual
# ================================================


# Dependencias
#=========================================================
import numpy as np
import rospy
import math
import time
from std_msgs.msg import Header
from image_conv.msg import Coordinates
from geometry_msgs.msg import Quaternion, Point, PoseStamped, Pose
from std_msgs.msg import Header
#=========================================================

# Funcoes
#=========================================================
def init_uav_landing():
    rospy.init_node('uav_landing', anonymous=True)
    rospy.Subscriber("/mavros/vision_pose/pose", PoseStamped,
uav_landing)
        rospy.spin()

def uav_landing(uav_pose):


    uav_position = uav_pose.pose.position
    x = uav_position.x
    y = uav_position.y
    z = uav_position.z


    quad_1 = x > 0 and y > 0
    quad_2 = x < 0 and y > 0
    quad_3 = x < 0 and y < 0
    quad_4 = x > 0 and y < 0

    off_target = abs(x) > 0.02 or abs(y) > 0.02
    on_target = abs(x) < 0.02 and abs(y) < 0.02 and z > 0.02

    timeout = time.time() + 10

    if (abs(x) > 0.02 or abs(y) > 0.02):
        while True:
        #enquanto off_target:
            time.sleep(1)
            dx = abs(x/10)
            dy = abs(y/10)
```

```
            #faz alinhamento de acordo com o quadrante em que o VANT
se encontra:
            print "Not on target"
            if (quad_1):
                x = x - dx
                y = y - dy
            elif (quad_2):
                x = x + dx
                y = y - dy
            elif (quad_3):
                x = x + dx
                y = y + dy
            elif (quad_4):
                x = x - dx
                y = y + dy

            if (abs(x) < 0.02 and abs(y) < 0.02 and z > 0.02):
                landing_status = uav_descent(uav_pose, x, y, z)
                print landing_status
                break
            print "Alligning: x = %r y = %r z = %r" % (x,y,z)

            if(time.time() > timeout):
                break

def uav_descent(uav_pose, x,y,z):
    on_target = abs(x) < 0.02 and abs(y) < 0.02 and z > 0.02

    while on_target:

        dz = abs(z/10)
        z = z - dz
        time.sleep(1)
        print "Descending: x = %r y = %r z = %r" % (x,y,z)

        publish_pose(uav_pose, x, y, z)

        if (z <= 0.3):
            status = "Landed"
            break
    return status

def publish_pose(uav_pose, _x, _y, _z):
    pub = rospy.Publisher('/mavros/setpoint_position/local',
PoseStamped, queue_size=1)

    header.stamp = rospy.Time.now()
    header.frame_id = "target_position"

    target_pose.pose.orientation = uav_pose.pose.orientation
    target_pose.pose.position.x = _x
    target_pose.pose.position.y = _y
    target_pose.pose.position.z = _z
    target_pose.header = header

    pub.publish(target_pose)
```

```
#=========================================================
# Funcao main
#=========================================================
if __name__ == '__main__':
    target_pose = PoseStamped()
    header = Header()

    init_uav_landing()
```

Apêndice F – Artigo apresentado na conferência "MTS/IEEE OCEANS '17"

Interoperability of Unmanned Systems in Military Maritime Operations: Developing a controller for unmanned aerial systems operating in maritime environments

Rodolfo Santos Carapau, Alexandre Valerio Rodrigues, Mario Monteiro Marques & Vitor Lobo
Centro de Investigação Naval (CINAV)
Marinha de Guerra Portuguesa (MGP)
Almada, Portugal

Fernando Coito
Faculdade de Ciencias e Tecnologias (FCT)
Universidade Nova de Lisboa
Lisboa, Portugal

Abstract— **The current evolution of unmanned systems has unlocked a valuable opportunity to extend military operational capabilities in maritime environments, through the performance of missions such as: maritime reconnaissance, surveillance, patrolling, and search and rescue. To further develop this potential, the implementation of an interoperable structure that contains several unmanned vehicles cooperating and interacting to achieve a common goal, is essential. This paper aims at presenting the benefits of the use of unmanned systems, the benefits of the developing interoperability between unmanned vehicles, and at presenting a practical implementation of an on-board unmanned aerial vehicle (UAV) controller that allows the addition of several new functionalities to the UAV, like performing visual autonomous landing on warships. The development of the controller explores the potential of the Robotic Operating System, which provides a variety of libraries, tools and functionalities, as well as the capability of hardware abstraction, and it promotes and supports the development of new technologies. This work will provide a step further on the development of unmanned systems associated to the Portuguese Navy, and also to become a bedrock for future development.**

Keywords— ***Interoperability; Unmanned Aerial Vehicle; Robotic Operating System; Visual Autonomous Landing.***

I. Introduction and Motivation

The importance of maritime environments can be demonstrated through its great value for global economics. This value resides in commercial trading routes, transportation, commercial fishery, and off-shore oil platforms, all of which translate in great revenue [1]. As such, there is a permanent need to guarantee the welfare of these activities. The development of unmanned vehicles (UxVs) has unlocked the potential to perform tasks of maritime reconnaissance, surveillance, patrolling, search and rescue (SAR), and maritime security overall [2]. The use of unmanned aerial vehicles (UAVs) to perform such tasks, comes as a valuable resource in terms of financial and human costs, when compared to its manned counterparts. This potential can be enhanced by deploying multiple UAVs, or even other UxVs, which promotes the study of interoperability – a paramount concept that defines the quality of the joint performance of different systems.

The creation of interoperable systems provides certain benefits like: the reduction of operational costs and complexity; reduction of compatibility issues; successful cooperation and interaction between different systems; promotes the creation and the growth of heterogeneous structures; promotes joint collaboration and the creation of joint technology; and enhances the operational capability of a system. These benefits motivate the work developed on this paper, through the creation of a controller that uses the Robotic Operating System (ROS) as a main structure of software development, which allows for the integration of multiple functionalities, it promotes interoperability between other ROS structures on other UxVs, and it guarantees future development, since ROS keeps growing as a reference for robotic development. By implementing a ROS controller on-board an UAV, it enhances the operational capability of the UAV without tampering with the pre-existing automatic pilot module. This benefit can be validated by implementing new functionalities to the UAV, like a protocol communication bridge between the North Atlantic Treaty Organization (NATO) Standardization Agreement (STANAG) 4586 and the Micro Air Vehicle Communication Protocol (MAVLink), or in the case of this wok, a visual autonomous landing functionality to operate with warships.

As such, this work focuses on achieving the following objectives:

- Present the importance of UxVs for military applications;
- Relate the concept of interoperability with unmanned systems;

- Present the benefits of interoperability associated to UxVs;
- Create an on-board ROS controller for an UAV;
- Enhance the UAV's potential through the implementation of visual autonomous landing.

This paper is organized according to the following structure: section II provides some examples of projects that developed the use of UAV in military maritime environments, projects that developed interoperability between unmanned systems to perform certain tasks and missions, and work related to visual autonomous UAV landing associated to the use of ROS; section III presents a conceptual overview of the system developed, beginning by describing the system global architecture, then describing the controller modules and functionalities, and then concluding with an introduction to the vision based landing approach; section IV describes the validation process of this work, beginning with the command module validation, and advancing to the visual autonomous landing validation; finally a conclusions section that reviews the study as a whole.

II. State of The Art

When discussing the use of UAVs in military environments, it is crucial to reference some of the projects that developed the implementation of this technology, and validated its benefits. Both SEAGULL and Smart UNattended airborne sensor Network for detection of vessels used for cross border crime and irregular entrY (SUNNY) are valid examples of such projects.

The SEAGULL project aims at developing an "intelligent system to support maritime situation awareness based on UAVs". The project addresses UAV collision detection and avoidance methods, as well as optical sensors which contribute to the situational awareness of maritime tasks like detection and geo-referencing of oil spill or hazardous and noxious substances, maritime tracking systems, recognizing behavioral patterns, and monitoring parameters and indicators of good environmental status. It is a project developed by the Portuguese Navy and Air Force, Critical Software, *Instituto Superior Técnico* and *Faculdade de Engenharia da Universidade do Porto*, and motivated by the current Portuguese maritime search and rescue area, as well as the current Exclusive Economic Zone and continental platform extension [3] [4].

SUNNY also explores solutions for border patrolling and surveillance issues. Project SUNNY focuses on developing and implementing new and comprehensive solutions for intelligent surveillance of borders (maritime and land), and for the detection of border crossing and illegal entries through the use of a network of different on-board UAV sensors. As such, SUNNY develops the use of UAVs to perform detection, tracking, recognition, surveillance and patrolling tasks, while integrating a variety of sensors operating in the same system [5].

The advantages that are connected to the use of UAVs can be further developed through the addition of other UxVs, promoting interoperability. As stated in [6], interoperability is defined as the capability a system has to work with other systems without great limitations or additional effort of the user, which can be made possible through the use of standards. By defining a reference model (standard), it is possible to work with different entities on common ground, which is one of the goals of the work described on this paper. This is a capability that can be extended, not only to several UAVs, but also to other UxVs like unmanned ground vehicles (UGVs), unmanned surface vehicles (USVs) and unmanned underwater vehicles (UUVs). Building an interoperable system results in the reduction of operational costs and complexity, in a network built with common protocols, efficiency on accessing and searching for data in a common network, and in standards to connect new applications and functions to a pre-established interoperable structure. The development of interoperability and standards relies on projects that focus on the cooperation and interaction between several UxVs and the operator, to perform tasks associated to military intervention, like: SAR, reconnaissance, patrolling, payload transportation, among other tasks. These were some of the goals of projects like Deployable Search and Rescue Integrated Chain with Unmanned Systems (DARIUS) and Integrated Components for Assisted Rescue and Unmanned Search Operations (ICARUS).

The DARIUS project focuses on the intervention of UxVs, from multiple agencies, in various environments where SAR scenarios are being played, contributing to the development of interoperability between UxVs, and determining the requirements for future SAR UxVs. The development of DARIUS aims to benefit its users by enabling the control of multiple UxVs, sharing of UxVs (as well as collected information) across different users within the same operation, and the integration of UxVs in command and control and communication chains. DARIUS's organization defines three separate operational levels (coordination, tactical and execution) that work together to provide a response to crisis scenarios (earthquake, forest fire, maritime disaster, among others) through the intervention of multiple agencies in a multi-national level. Overall, DARIUS aims at developing a fast and easily deployable crisis response using UxVs and rescue systems, integrating UxVs in existing command and control platforms, establishing a communication network between the whole system while using UxVs for communication relaying, developing UxVs navigation capabilities through harsh environments, and providing adequate interaction between the UxVs, its payloads, the emergency first responders and the crisis victims [7].

Project ICARUS also develops the use of UxVs in SAR Operations. It focuses on developing unmanned SAR technologies for detecting, locating and rescue human victims. Associated with this objective, is the creation of a set of tools for unmanned SAR that would be accessible to the majority of the end-users. The ICARUS system establishes a network

between the control stations and the UxVs so that vehicles can communicate with each other and with the control stations, in order to promote autonomous navigation, victim detection and identification, and information flux, which allows the base station to process the incoming data into a geographical information, and thus allowing the creation of a situational map that promotes situational awareness and decision-making. ICARUS uses UAVs, UGVs and USVs that integrate ROS and Joint Architecture for Unmanned Systems (JAUS) controllers, which connect to the command and control module through JAUS's bridges, which achieves interoperability and promotes the use of JAUS has a standard. As a whole, ICARUS aims at developing cooperative UxV tools for unmanned SAR, developing sensors to detect human presence, cooperation and interaction between unmanned SAR devices, developing an interoperable wireless communication network, integration of unmanned SAR tools in command and control systems dedicated to SAR, and developing a training and support system for unmanned SAR and SAR teams. By doing so, ICARUS is developing a valuable resource for crisis management, reducing the impact of a crisis upon a community. As such, ICARUS was internationally recognized, receiving the Multi-Robot coordination award in the multi-domain robotics competition, EuRathlon 2015 [8].

When it comes to the implementation of UAV visual autonomous landing, [9] and [10] serve as examples of development using ROS to achieve image based landing. For this paper, [11] provided guidance on the approach to take when developing the visual autonomous landing module, that validates the ROS implementation.

III. System Overview

This section presents the overall system, and the methodology followed to its creation.

A. System Architecture

The system is a typical unmanned aerial system, composed by the vehicle and a Ground Control Station (GCS). The innovation lies within the UAV, with the addition of an on-board controller. This controller establishes a communication link with the GCS so that the operator can control the UAV through the on-board controller. The controller than performs its commands by communicating with the auto-pilot module via the MAVLink protocol. In order to perform autonomous visual landing, the controller receives information from its visual sensor, which allows the controller to detect a visual marker and perform an autonomous landing on the marker.

Figure 1 shows the outline of the system previously described.

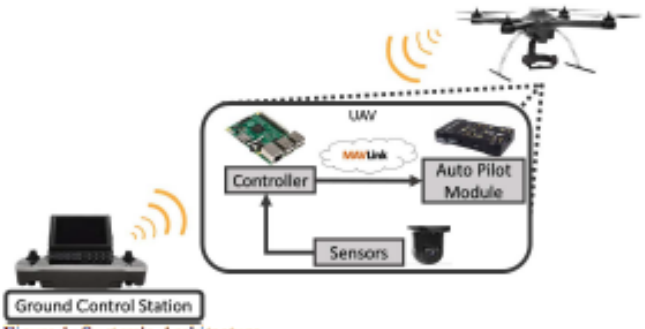

Figure 1: System's Architecture.

B. Controller

The external controller is an on-board controller, which is a Raspberry Pi 2 model B, running Debian and programs built upon the ROS Kinetic version.

ROS is an open-source framework that allows writing robotic software with the aim of simplifying the task of developing complex and detailed robotic systems. The creation of the ROS provides a variety of libraries, tools and functionalities with the capability of hardware abstraction that promote and support the development of new technologies. ROS simplifies tasks such as memory and process management, parallel processing, and makes possible the creation and connection of several packages that have different functions, since control drivers for sensors and peripherals, interfaces, image and sound processing, to a variety of other applications. The development of ROS has also created a community that grows on a daily basis through the submission of the users own projects and packages, as such, these contributes can be re-used by different users to develop new projects, without the need of developing functionalities that already exist. This framework uses file system resources organized in packages and stacks, and a computation graph that brings together concepts like nodes, topics, messages, services and bags, in order to provide the necessary tools to build, structure and develop a new (or already existing) project. Overall, ROS provides a modular system that can easily develop simple and complex structures, also it supports a variety of programing languages making it a flexible and versatile framework, and it is an open-source system that associates its own development to the contributes provided by its community [12].

The controller's architecture is described on Figure 2:

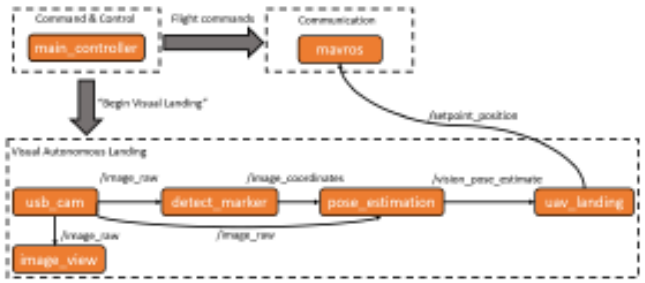

Figure 2: On-board controller architecture.

The system is composed by the following ROS nodes:

- **usb_cam** – provides raw image from the USB camera;
- **image_view** – displays the image captured by the USB camera, allowing visualization;
- **detect_marker** – detects and analyses contours within the camera image in order to detect a visual marker and provide its coordinates;
- **pose_estimator** – estimates the pose of the UAV using the image coordinates provided by "detect_marker" and visual analysis tools to obtain 3D coordinates of the UAV related to the marker's position;
- **uav_landing** – aligns the UAV with the marker's position, and then begins the descent while aligned with the marker, thus performing the landing;
- **mavros** – provides a communication driver for the autopilot with the MAVLink protocols. It generates MAVLink messages to the auto-pilot, with the information provided by the previous nodes;
- **main_controller** – allows the user to issue flight commands to the auto-pilot, and it initiates the visual autonomous landing.

The development of the controller began with the creation of a script that could issue commands to the auto-pilot, and thus establishing the bedrock for the development of new functionalities under ROS. The script ran as ROS node ("main_controller") that could issue flight commands like: "takeoff", "landing", "set flight mode", "insert waypoint", etc. These commands were called as services and published to topics subscribed by the preexisting "mavros" node. The "mavros" node would then convert the commands to MAVLink messages and sent them to the auto-pilot.

MAVlink is a fast and efficient communication protocol for UAVs that translates in a lightweight message library for communications between the vehicle and the GCS. The messages are made of packages, which contain encoded bytes divided in headers, payload and error correction, and sent through a transductor. Also, it is a protocol that is widely supported by the majority of autopilot's and GCS's software (p.e. Ardupilot, Pixhawk, QGroundControl, APM Planner, among others). Although it fails to have the complexity and confidentiality of other military communication protocols, it stands as a popular, reliable and accessible protocol that provides message creation and customization [13].

After the development of the "main_controller" node, began the creation of nodes that could perform visual autonomous landing, in order to validate the ROS implementation. The preexisting "usb_cam" publishes a topic containing raw image data displayed as video, this topic is subscribed by the custom new node "detect_marker", which reads the video (frame by frame) detecting the contours on each image. When the circular contour of the visual marker is detected, the node reads the image coordinates (in pixels) of the marker and it publishes this information to the "pose_estimator" node. The "pose_estimator" node subscribes to the "/image_coordinates" topic, which contains the image coordinates of the detected visual marker. By using the OpenCV "solvePnP()" function, which uses as inputs the image coordinates, 3-dimensional model coordinates (in meters) of the marker, and internal camera parameters, the node obtains the visual marker's pose, which is used to obtain the camera's pose and 3D position when related to the marker's center position as the reference point. This information is then published to the "uav_landing" node, which firstly aligns the UAV with the visual marker by reducing the x and y coordinates of the UAV to zero (since the center of the marker is origin coordinate). When the UAV is aligned, the node begins the descending phase that reduces the z coordinate, at a steady rate, in order to land the UAV. This landing approach takes in consideration that the camera is located directly under the UAV and pointing downwards. The aligning and descending procedures can only be executed by the auto-pilot by publishing this information to the "/setpoint_position" topic, which is subscribed by "mavros", converted to a MAVlink message, and sent to the auto-pilot to execute. Figure 3 describes the interactions between the controller, the camera and the auto-pilot.

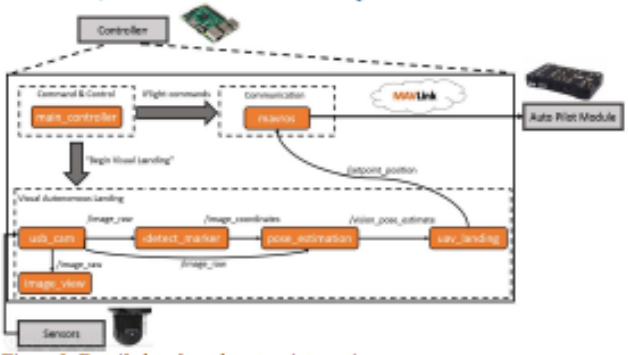

Figure 3: Detailed on-board system interactions.

This implementation uses the advantages provided by ROS to improve the UAV's operational capability and autonomy.

The following sub-section develops the concepts and implementation of the visual autonomous landing process.

C. *Visual Autonomous Landing*

This sub-section describes the process of implementing the autonomous landing functionality through the use of a visual marker.

The initial approach to this process was based on the type of landing marker chosen. Since the goal was to perform autonomous landing using the information provided by visual sensors, the initial approach was focused on the study of fiducial markers. Fiducial markers are natural or artificial marks, which allow for the detection and identification of reference points within an image, when detected in the visual scope of an imaging system [14]. These markers can be categorized by different types of existing techniques: augmented reality markers, cooperative targets, infrared markers, natural landmarks and visual cues. In this study, the

chosen marker was a visual cue similar to the classic helicopter landing marker (Figure 4).

Figure 4: Visual marker for autonomous landing.

The circular outer contour of this marker simplifies the marker detection process since there are no visual differences when looking at it from different perspectives, whereas other shapes could prove an additional challenge.

The marker detection and identification process is applied on each frame of the video provided by the on-onboard camera, and it starts with pre-processing the image to grayscale, since the color of the image is of no concern, followed by image blurring and edge detection, to remove high frequency noise and to reveal the outlines of the objects within the image, respectively. The edge detection provides an edge map of the image, which is used to detect the contours of the objects within the image. After detecting the contours, each contour is tested in order to determine if it matches the shape of a circle. When a match is confirmed, the pixel coordinates of the center of the marker are calculated, along with other four points which form a bounding box around the circular outline. These coordinates are then forwarded to the node which performs pose estimation.

The pose of an object describes its orientation and position according to a certain reference point. As such, obtaining the pose of an object implies obtaining a linear position (translation) and an angular position (rotation). Translation is based on the linear movement transformation when changing positions, so it can represented by (1), whereas rotation describes the angular transformation around the X, Y and Z axes, which results in three degrees of freedom that can be represented as Euler angles (roll, pitch and yaw), as shown in (2).

$$T = (X, Y, Z)[\text{m}] \tag{1}$$

$$R = (\phi, \theta, \psi)[\text{rad}] \tag{2}$$

The pinhole camera model [15] provides a valuable approach to computer vision since it describes the relation between 3D world coordinates and its projections onto the image plane through a perspective transformation (Figure 5).

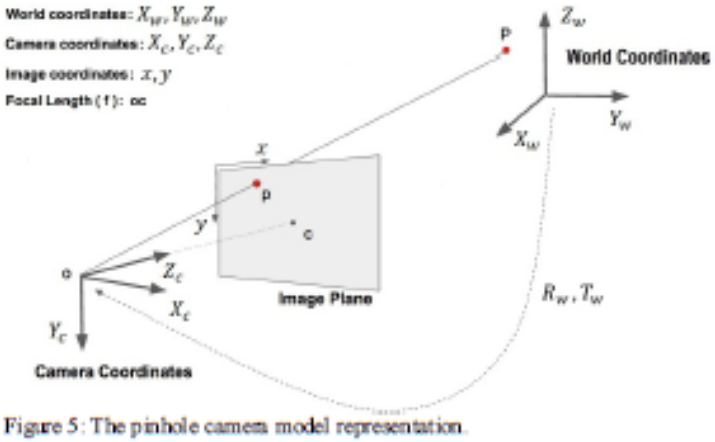

Figure 5: The pinhole camera model representation.

The previous model can also be represented according to equation (3),

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} [R|T] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3}$$

where:

- **(X,Y,Z)** are the coordinates of a 3D point in the world coordinate space;
- **(x,y)** are the coordinates of the projection point in pixels;
- **[R | T]** is the joint rotation-translation matrix, also known as, matrix of extrinsic parameters;
- **(c_x, c_y)** is a reference point, typically at the image center;
- **(f_x, f_y)** are the focal lengths expressed in pixels;
- **s** is a scale factor, related to the image unknown depth.

According to the previous model, the marker pose can be estimated by solving the Perspective-n-Point (PnP) problem, then leading to the estimation of the UAV's pose [16]. This problem can be solved when the intrinsic parameters related to the calibration of the camera are known, and when the location of n 3D points of an object and its corresponding 2D image projections are known. Figure 6 displays the camera reference that originates from the UAV, and the world reference originated from the marker's geometric center, as well as the rotation and translation (pose) matrixes obtained as solutions of the PnP problem.

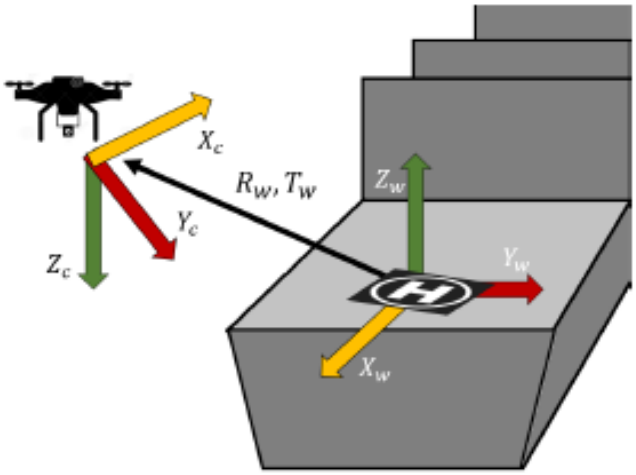

Figure 6: Camera and World references considered.

After obtaining the pose (R_w, T_w) of the UAV, the "pose_estimator" calculates the coordinates of the UAV referenced to the marker's position (being the center the origin point) and world coordinates. It then publishes these coordinates to the "uav_landing" node that performs the UAV alignment and descent as described in the previous sub-section.

IV. Validation

The successful validation of this work consists on validating two separate components: the controller flight commands, and the visual autonomous landing.

In order to validate the controller, it must be able to successfully receive and pass flight commands on both simulated and real environments:

- **Simulated environment** – This test evaluates the capability and performance of the controller node to pass flight commands to a quadcopter simulated with "Software in the Loop" (SITL) and "APM Mission Planner".
- **Real environment** – This test has two different phases. The first phase is intended to evaluate the capability and performance of the controller node when passing flight commands to a physically connected "Pixhawk" autopilot, while receiving the commands from a wireless connected GCS. The final phase seeks to perform the controller evaluation on a fully functional UAV during takeoff, flight and landing phases.

The validation of the visual autonomous landing begins with tests related to the coordinates obtained from visual pose estimation. The first test evaluates the alignment capability requiring the camera to be at a known fixed position, while the target is moved below the camera, varying its (x,y) coordinates in a measured area, providing reference points to compare with the estimated coordinates (Figure 7).

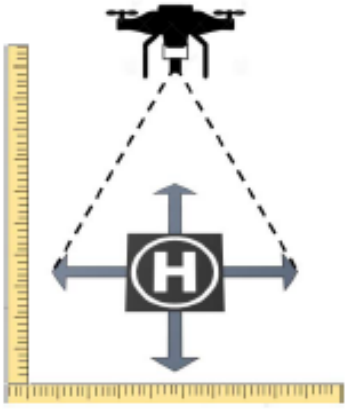

Figure 7: Representation of the alignment test (real test uses the camera only).

The second test evaluates the performance of the descending phase, by comparing the estimated z coordinate values with the measured values, while by approximating the camera to the target along a fixed (x,y) position (Figure 8).

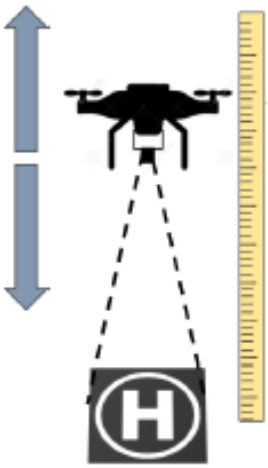

Figure 8: Representation of the alignment test (real test uses the camera only).

After validating the visual coordinates estimate, the next step is based on running the whole visual autonomous landing module (Section III – B.) with SITL and "APM Mission Planner". In this test, the primary focus is to verify if the simulators receive the setpoint positions previously calculated. Passing the test, the next test completes the simulated tests with the verification of the performance of the visual autonomous landing module when physically connected to a "Pixhawk" autopilot, while being prompted by a wireless connected GCS.

The final validation tests, evaluate the performance of the UAV while in a controlled land environment, and on-board a warship in an uncontrolled environment.

V. Conclusions

This work focuses on presenting the benefits of developing unmanned systems applied to maritime military environments, as well as the advantages associated to the development of interoperability between UxVs, and on developing an UAV controller that extends its capabilities.

Although the modules of the controller have been developed, it still hasn't been subjected to the validation process, previously described. Assuming it performs well during this process, it will provide versatility and flexibility to an UAV's performance when integrated on a naval force, and it will open an opportunity to further develop its capabilities.

As a whole, this paper describes the integration of a ROS external controller on an UAV, providing new functionalities with the automatic pilot abstraction. This innovation allows the use of libraries with high level of abstraction and ample support of multiple sensors to the automatic pilot, guaranteeing a stable and safe flight. Therefore, it amplifies the operational capability of an UAV, without the need of rewriting the automatic pilot.

Future work includes the development of a more robust visual detection method like OpenTLD [17], the detection of different types of fiducial markers, and the addition of different functionalities to the on-board controller.

References

[1] M. Stopford, *Maritime Economics*, Second edi. New York: Routledge - Taylor & Francis Group, 2003.

[2] R. Austin, *Unmanned Aircraft Systems: UAV Desing, Development and Deployment*, First edit. John Wiley and Sons Ltd, 2010, pp. 9–15.

[3] Mario Monteiro Marques, V. Lobo, R. Batista, J. Viegas, A. P. Aguiar, J. E. Silva, J. Borges de Sousa, M. F. Nunes, R. A. Ribeiro, A. Bernardino, and J. S. Marques, 'Oil Spills Detection: Challenges addressed in the scope of the SEAGULL project', in *MTS/IEEE OCEANS 2016*, Monterey, 2016 pp. 1–6.

[4] Mario Monteiro Marques, P. Dias, N. Santos, V. Lobo, R. Batista, D. Salgueiro, R. Ribeiro, J. Marques, A. Bernardino, M. Griné, M. Taiana, M. Nunes, E. Pereira, J. Morgado, A. Aguiar, M. Costa, J. Silva, A. Ferreira, J. Sousa, "Unmanned Aircraft Systems in Maritime Operations: Challenges addressed in the scope of the SEAGULL project," in *MTS/IEEE OCEANS 2015*, Washington, 2015, pp. 1–6.

[5] SUNNY, 'About SUNNY: Overview', 2014. [Online]. Available: http://www.sunnyproject.eu/about.aspx#overview. [Accessed: 10-Mar-2017].

[6] IEEE, 'IEEE Standard Glossary of Software Engineering Terminology', Standards Coordinating Comitee of the Computer Society of IEEE - Standards Board, 1990.

[7] P. Chrobocinski, E. Makri, N. Zotos, C. Stergiopoulos, and G. Bogdos, 'DARIUS project: Deployable SAR integrated chain with unmanned systems', *2012 Int. Conf. Telecommun. Multimedia, TEMU 2012*, 2012, pp. 220–226.

[8] Mario Monteiro Marques, R. Parreira, V. Lobo, A. Martins, A. Matos, N. Cruz, J. M. Almeida, J. C. Alves, E. Silva, j. Bedkowski, K. Majek, M Pelka, P. Musialik, H. Ferreira, A. Dias, B. Ferreira, G. Amaral, A. Figueiredo, R. Almeida, F. Silva, D. Serrano, G. Moreno, G. D. Cubber, H Balta, H. Beglerovic, S. Govindaraj, J. M. Sanchez, and . Tosa, 'Use of multi-domain robots in search and rescue operations - Contributions of the ICARUS team to the euRathlon 2015 challenge', in *MTS/IEEE OCEANS 2016*. Shangai, 2016, pp. 1-7.

[9] J. N. Weaver, D. Z. Frank, E. M. Schwartz, and A. A. Arroyo, 'Uav Performing Autonomous Landing on USV Utilizing the Robot Operating System', in *ASME Dist. F - Early Career Tech. Conf. 2013*, vol. 12, 2013, pp. 119–124.

[10] I. S. Borshchova, S. O'Young, 'Visual servoing for autonomous landing of a multi-rotor UAS on a moving platform', in *Journal of Unmanned Vehicle Systems*, vol. 5, no. 1, 2017, pp. 13–26.

[11] J. L. Sanchez-Lopez, S. Saripalli, P. Campoy, J. Pestana, and C. Fu 'Toward visual autonomous ship board landing of a VTOL UAV', in *2013 International Conference on Unmanned Aircraft Systems, ICUAS 2013 - Conference Proceedings*, Atlanta, 2013, pp. 779–788.

[12] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*, vol. 53., 2015.

[13] QGroundControl, 'MAVLink Micro Air Vehicle Communication Protocol'. [Online]. Available: MAVLink Micro Air Vehicle Communication Protocol. [Accessed: 01-Jan-2017].

[14] A. Andziulis, D. Drungilas, V. Glazko, and E. Kiseliovas, 'Resource saving approach of visual tracking fiducial marker recognition for unmanned aerial vehicle', in *Adv. Electr. Electron. Eng.*, vol. 13, no. 4, 2015, pp. 359–366.

[15] P. Strum, 'Pinhole Camera Model', in *Computer Vision: A Reference Guide*, K. Ikeuchi, Ed. Springer US, 2014, pp. 610–613.

[16] P. Moreno-Noguer, F.;Lepetit, V.; Fua, 'Accurate Non-Iterative O(n) Solution to the PnP Problem', *in Comput. Vision, 2007. ICCV 2007. IEEE 11th Int. Conf.*, 2007, pp. 1–8.

[17] G. Nebehay, 'Robust Object Tracking Based on Tracking-Learning-Detection', Technischen Universität Wien, 2012.

Apêndice G – Artigo apresentado na conferência "Sea-Conf 2017"

"Mircea cel Batran" Naval Academy Scientific Bulletin, Volume XX – 2017 – Issue 1
Published by "Mircea cel Batran" Naval Academy Press, Romania // The journal is indexed in: PROQUEST / DOAJ / DRJI / JOURNAL INDEX / I2OR / SCIENCE LIBRARY INDEX / Google Scholar / Crossref / Academic Keys / ROAD Open Access / OAJI / Academic Resources / Scientific Indexing Services / SCIPIO / JIFACTOR

Unmanned Aerial Systems in Military Environments: The Benefits of Interoperability

Rodolfo Santos CARAPAU[1]
Alexandre Valério RODRIGUES[1]
Mario Monteiro MARQUES[1]
Victor LOBO[1]
[1] CINAV, Portuguese Navy Research Center, Almada, Portugal
E-mail: santos.carapau@marinha.pt

Abstract: *Nowadays, the use of Unmanned Aerial Vehicles (UAVs) is a growing presence in both civilian and military environments, which has resulted in an opportunity to explore this technology, its benefits and how they can be improved. This paper aims to present a study focused on the impact of UAVs in military environments and how interoperability can further develop the benefits of the use of unmanned systems. It presents the importance and motivation for the use of UAVs, developing to a description of an UAV and its supporting structure. Afterwards, the study presents the primary military UAV applications, as well as studies that have been conducted to develop UAV capabilities in performing tasks such as surveillance, reconnaissance, search and rescue, and hazardous materials detection. Following the study of UAV in military scenarios, an approach to interoperability of unmanned systems is presented: its concept, and a project that has proved its reliability, converging to the benefits of interoperability. Overall this study hopes to improve awareness regarding unmanned systems and how it can play a key role for the future of military technology.*
Keywords: *Interoperability, Unmanned Aerial Vehicle, Military.*

I. INTRODUCTION AND MOTIVATION

The creation and development of Unmanned Aerial Vehicles (UAVs) has provided a valuable opportunity to develop tasks such as search and rescue (SAR), surveillance, reconnaissance, inspection, patrolling, hazardous materials detection, among many other tasks. Particularly in military scenarios, the danger is more significant, which motivates the use of UAVs to perform certain missions, since its use promotes the safeguard of human lives. This is one of the main motives that has promoted innovation regarding unmanned systems (UxS), and the development of applications for UAVs. For example, for a Navy, the creation and development of UAV applications such as buoy deployment and monitoring, missile decoy, communication relaying, and protection against coastal attacks prove to be a valuable asset [1]. Other applications regarding the detection of hazardous materials, SAR, surveillance and border patrolling, have been the target of projects like GammaEx, SEAGULL, and *Protection of European Seas and borders through the intelligent use of surveillance* (PERSEUS), involving military entities.

Although the deployment of an UAV grants an edge in the performance of a mission, the deployment of several different unmanned systems working together would result in an improvement of the operational capability. The successful achievement of a feat of such dimension can be made possible through the integration of interoperable systems that work together to achieve common objectives. This involves the creation of standard software and protocols that support unmanned systems in order to guarantee smooth and reliable services and information exchanges. Projects like *Deployable SAR Integrated Chain with Unmanned Systems* (DARIUS) have implemented such concepts to develop crisis emergency response systems that synergize UxS with control systems and multi-national emergency services.

According to this motivation, this study aims to achieve the following objectives:

- Demonstrate the importance of UAV for military applications;
- Relate the concept of interoperability with unmanned systems;
- Explaining the benefits of interoperability associated to the use of unmanned systems;

This paper is organized according to the following structure: section II provides an overall view of UAVs operating in military environments by defining "UAV" and the basic architecture and components of an Unmanned Aerial system (UAS), presenting a timeline of UAV development motivated by military conflicts, followed by the enumeration of several military tasks that UAVs can perform, and finally presenting projects that have developed these capabilities; section III combines the concept of interoperability with unmanned systems by presenting definitions of interoperability, and technology/projects that motivate interoperability between unmanned systems, in order to determine the primary benefits of interoperability; finally a conclusions section that reviews that study as a whole.

DOI: 10.21279/1454-864X-17-I1-000

II. UAVs IN MILITARY ENVIRONMENTS

UAVs are vehicles that operate by air, don't carry an onboard pilot or crew, and can be controlled by an onboard electronic equipment, or by a ground control station, through the use of waypoints, pre-established goals, or through manual teleoperation [2], [3]. They can be classified according to their weight and size (small, medium and heavy), operating altitude (low, medium or high) and design (fixed wing, rotary wing, flapping wing and blimps) [1] [4]. An UAV is part of an Unmanned Aerial System, which can be divided in three main parts:

- Command and Control includes the ground control station (GCS), communication subsystem, launch and recovery, and support equipment;
- Datalink establishes a communication link (uplink: land-to-air, downlink: air-to-land) between the communication subsystems of the ground control station and the vehicle;
- The Vehicle includes the payload, a navigation subsystem, sensors, a communication subsystem, power and propulsion.

The creation and development of UAVs dates back to the Nineteenth Century, and were always enhanced by military conflicts. Events such as World War I and II, Cold War, Vietnam War and Gulf War served as catalysts for the development of unmanned systems. Figure 1 displays a timeline describing the creation and evolution of UAVs, matching the main historical developments of UAVs with the time period of the previously mention conflicts, and mentioning the most recent development of small and micro UAVs [5] [6].

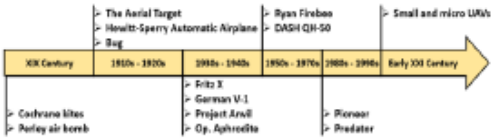

Figure 1 - Timeline of UAV evolution.

In military environments, the advantage of using the operating capability UAVs provide, can be demonstrated through a variety of missions and tasks that UAVs can perform. According to Rosa [7], such tasks can be:

- Intelligence and reconnaissance;
- Mine countermeasures;
- Anti-submarine Warfare;
- Inspection/identification;
- Oceanography/hydrography;
- Communication;
- Navigation;
- Payload delivery;
- Enemy Influence Activities;
- Time critical strike;
- Maritime Security;
- Surface Warfare;
- Special Operations Forces Support;
- Electronic Warfare;
- Maritime Interdiction Operations Support;
- Aerial Warfare;
- Transport cargo or passengers;
- Extraction;
- Insertion;
- Surveillance;
- Search and rescue;
- Analysis of damage attack;
- Border patrol.

In order to develop the capabilities and performance of UAVs, when executing some of the previous tasks, projects like GammaEx, SEAGULL, and Perseus, were created.

The GammaEx project is being conducted by the Portuguese Navy and Army, IST (*Instituto Superior Técnico*), I-SKYEX and ISQ (*Instituto de Soldadura e Qualidade*), and it aims to develop UASs to integrate an emergency response system to chemical, biological, radiological and nuclear (CBRN) threats or incidents. It involves the creation of UAVs (Figure 2) that carry CBRN sensors, and that are compliant with the ATEX (*ATmosphères EXplosibles*) directives. This system focuses on the deployment of GammaEX UAVs (Figure 2) in maritime and land scenarios, being a CBRN onboard a ship or in an off-shore platform, or inside a warehouse or any other building [8].

Figure 2 - GammaEX UAVs: a M6 hexacopter on the left and a tricopter on the right [8].

Another effort in developing UAVs is the SEAGULL project. SEAGULL is a project developed by the Portuguese Navy and Air Force, Critical Software, IST and FEUP (*Faculdade de Engenharia da Universidade do Porto*), and it focuses on developing intelligent systems to support maritime situation awareness based on unmanned aerial vehicles. Motivated by the current Portuguese maritime search and rescue area, as well as the current EEZ (Exclusive Economic Zone) and continental platform extension, the project aims to create an intelligent maritime surveillance systems by outfitting UAVs with optical sensors to perform: detection and geo-referencing of oil spills or hazardous and

DOI: 10.21279/1454-864X-17-I1-000

noxious substances, system tracking (vessels, shipwrecks, lifeboats, and other), recognizing behavioral patterns, and monitoring parameters and indicators of good environmental status. The implementation of this project provides an edge to the Portuguese Navy and Air Force when performing patrolling missions, SAR and surveillance on maritime environments due to the extended area coverage that the SEAGULL UAV (Figure 3) provides (at a small cost) [9].

Figure 3 - Portuguese Air Force ANTEX-M UAV used in the SEAGULL project [9].

Regarding UAVs operating in maritime environments, the *Protection of European Seas and borders through the intelligent use of surveillance* (PERSEUS) project also improves the use UAVs. PERSEUS is an INDRA project, financed by the European Union (EU) 7th *Framework Programme for Research and Technological Development* project that supports maritime border control and that aims to create and develop an EU maritime surveillance system that integrates national and communitarian installations, as well as new technologies. PERSEUS counts with the collaboration of several entities, some being non-European countries and agencies, military entities, NATO (North Atlantic Treaty Organization), and the International Maritime Organization. In its maritime validation scenarios, PERSEUS has deployed the tactical UAS ATLANTE (Figure 4 (a)) (with a connection between the GCS and the PERSEUS system) to enhance the surveillance awareness picture, and the ANTEX-M X02 UAV (Figure 4 (b)) to enhance detection, identification and tracking capabilities, which proves to be a contribution to the growth of the operating capability and reliability of UAVs [10].

Figure 4 - UAVs deployed in PERSEUS: on the left (a) the ATLANTE UAV; on the right (b) the ANTEX-M X02 UAV [10].

Overall, UAVs prove advantageous when deployed in military environments, since they reduce the risk of endangering human lives; involve less costs when compared to their manned counterparts; several vehicles can be deployed at the same time without the need of multiple pilots; its performance isn't affected by dull tasks; and prove a valuable extension to a unit's operational capability. This potential can be further developed when paired with other unmanned systems (UxS) (ground, surface, and underwater), which promotes the study of interoperability between unmanned systems.

III. INTEROPERABILITY OF UxS

The implementation of the concept of interoperability in a system allows for the development and expansion of its capabilities through the interaction with multiple other systems, which perform different tasks, in order to effectively achieve a common goal. This is a broad concept that can be applied in a variety of fields of study. According to NATO, interoperability is defined as the ability that two or more nation's forces have to train, exercise and execute, effectively, assigned missions and tasks [11]. Regarding UxS, the United States of America (USA) Department of Defense (DoD) [12] defines the concept has the ability that systems, units and forces have to provide and receive services from other systems, units and forces, in order to develop the joint operation effectiveness. The *Institute of Electrical and Electronics Enginners* (IEEE) [13] and the *National Institute of Standards and Technology* [14] have common interpretation of the definition of interoperability, which translates on the ability that different system's parts, or components, have to operate together effectively and successfully, obtaining a result that translates on a user's effort reduction. The *National Institute of Standards and Technology*'s definition adds that interoperability can be categorized in levels, types and degrees, and both entities add that interoperability can be achieved through the use of standards.

Standards are documents, software, hardware, etc., that define characteristics, parameters and rules of communication and interaction of a product or service. The creation of standards allow for the development of protocols, structures, and architectures that promote common operation, prevents incompatibility issues, and create a workspace that enhances the development of a system. Although the implementation of a standard may prove a challenge, its use allows for a reduction of time

DOI: 10.21279/1454-864X-17-I1-000

and costs associated to technological development, reduction of risk associated to the use of alternate technology, process productivity and effectiveness growth, and the use of standards motivates its own development, promoting reliability and guaranteeing its quality.

Regarding the development of UxS, there are several protocols and software that demonstrate the potential to be established as standards, due to their current use and functionalities. For example, the *NATO Standardization Agreement 4586* (STANAG 4586), *Micro Air Vehicle Communication Protocol* (MAVlink), and the *Robotic Operating System* (ROS).

The development of interoperability and standards has been achieved through projects that rely on cooperation and interaction between several UxS and the operator, to perform tasks associated to military intervention, like: search and rescue, reconnaissance, patrolling, payload transportation, among other tasks. These were some of the goals of projects like *Deployable SAR Integrated Chain with Unmanned Systems* (DARIUS).

The DARIUS project focuses on the intervention of unmanned systems, from multiple agencies, in various environments where SAR scenarios are being played, contributing to the development of interoperability between unmanned systems, and determining the requirements for further SAR unmanned systems. The development of DARIUS aims to benefit its users by enabling the control of multiple unmanned platforms, sharing of unmanned platforms (as well as collected information) across different users within the same operation, and the integration of unmanned systems in command and control and communication chains. DARIUS's organization defines three separate operational levels (coordination, tactical and execution) that work together to provide a response to crisis scenarios (earthquake, forest fire, maritime disaster, among others) through the intervention of multiple agencies in a multi-national level, and thus following the system architecture defined in Figure 5.

Figure 5 - DARIUS system architecture [15].

Overall, DARIUS aims at developing a fast and easily deployable crisis response using UxS and rescue systems, integrating UxS in existing command and control platforms, establishing a communication network between the whole system while using UxS for communication relaying, developing UxS navigation capabilities through harsh environments, and providing adequate interaction between the UxS, its payloads, the emergency first responders and the crisis victims [15].

According to the technological efforts that support interoperability and the projects that thrive upon it, it is important to state that the main benefits interoperability provide are:

- Reduction of operational costs and complexity;
- Reduction of compatibility issues;
- Successful cooperation and interaction between different systems;
- Promotes the creation and the growth of heterogeneous structures;
- Promotes joint collaboration and the creation of joint technology;
- Enhances the operational capability of a system (p.e. NATO).

Although interoperability provides several benefits, there are some limitations caused by standards with low accessibility and challenges related to information security requirements.

IV. CONCLUSIONS

The creation and development of UAVs has provided an opportunity for worldwide military entities to extend their operational capability while performing a number of different tasks and missions. This opportunity has motivated the creation of projects that studied the potential of UAVs, resulting in the creation of more UAVs and on the identification of requirements to implement, according to the task at hand. Regarding the advantages that the use of UAVs provide, it is crucial to further explore and develop these

DOI: 10.21279/1454-864X-17-I1-000

benefits through the creation of systems that promote interaction and cooperation between UAVs and other unmanned vehicles. Fortunately, the projects that delved in these concepts have proven successful, which further motivates the creation of new projects. Through the study of the capabilities of an UAV and the growing impact this technology has caused in military environments, this paper has resumed the potential of UAVs used to perform certain tasks, and has elevated the benefits of interoperability in the development of technology regarding unmanned systems.

BIBLIOGRAPHY

[1] R. Austin, *Unmanned Aircraft Systems*. 2010.
[2] H. Bendea, P. Boccardo, S. Dequal, F. G. Tonolo, D. Marenchino, and M. Piras, 'Low cost UAV for post-disaster assessment', *Proc. XXI Congr. Int. Soc. Photogramm. Remote Sens. Beijing China 311 July 2008*, vol. XXXVII, pp. 1373–1380, 2008.
[3] J. Cosic, P. Curkovic, J. Kasac, and J. Stepanic, 'Interpreting Development of Unmanned Aerial Vehicles using Systems Thinking', *Interdiscip. Descr. Complex Syst.*, vol. 11, no. 1, pp. 143–152, 2013.
[4] S. G. Gupta, M. M. Ghonge, and P. M. Jawandhiya, 'Review of Unmanned Aircraft System', *Int. J. Adv. Res. Comput. Eng. Technol.*, vol. 2, no. 4, pp. 2278–1323, 2013.
[5] J. F. Keane and S. S. Carr, 'A Brief History of Early Unmanned Aircraft', *John Hopkins APL Tech. Dig.*, vol. 32, no. 3, pp. 558–571, 2013.
[6] J. D. Blom, *Unmanned Aerial Systems: a historical perspective*. 2010.
[7] G. C. Rosa, M. M. Marques, and V. Lobo, 'Unmanned Aerial Vehicles in the Navy: Its Benefits', *Sci. Bull. Nav. Acad.*, vol. 19, no. 1, pp. 39–43, 2016.
[8] M. M. Marques, V. Lobo, J. Gouveia-Carvalho, A. J. M. N. Baptista, J. Almeida, C. Matos, R. S. Carapau, and A. V. Rodrigues, 'CBRN remote sensing using Unmanned Aerial Vehicles : Challenges addressed in the scope of the GammaEx project regarding hazardous materials and environments', *6th Int. Conf. Risk Anal. Cris. Response*, 2017.
[9] M. M. Marques, V. Lobo, R. Batista, J. Viegas, A. P. Aguiar, J. E. Silva, J. Borges de Sousa, M. F. Nunes, R. A. Ribeiro, A. Bernardino, and J. S. Marques, 'An Unmanned Aircraft System for Maritime Operations : System architecture , automatic detection , and sense and avoid subsystems', pp. 1–13.
[10] INDRA, 'PERSEUS - PERSEUS Demonstarion Campaigns', 2014.
[11] NATO, 'Interoperability for joint operations', no. July, p. 9, 2006.
[12] U.-D. of Defense, 'Unmanned Systems Integrated Roadmap', *Dep. Def.*, no. 2038, p. 168, 2013.
[13] IEEE, 'IEEE Standard Glossary of Software Engineering Terminology', Standards Coordinating Comitee of the Computer Society of IEEE - Standards Board, 1990.
[14] H.-M. Huang, 'Autonomy Levels for Unmanned Systems (ALFUS) Framework Volume I : Terminology National Institute of Standards and Technology', *Framework*, vol. I, no. October, pp. 0–46, 2008.
[15] P. Chrobocinski, E. Makri, N. Zotos, C. Stergiopoulos, and G. Bogdos, 'DARIUS project: Deployable SAR integrated chain with unmanned systems', *2012 Int. Conf. Telecommun. Multimedia, TEMU 2012*, pp. 220–226, 2012.

DOI: 10.21279/1454-864X-17-I1-000