UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Sistema autônomo para supervisão de missão e segurança de voo em VANTs

Jesimar da Silva Arantes

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP
SÃO CARLOS

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Jesimar da Silva Arantes

Sistema autônomo para supervisão de missão e segurança de voo em VANTs

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional. *EXEMPLAR DE DEFESA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Claudio Fabiano Motta Toledo

USP – São Carlos
Abril de 2019

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

A662s

Arantes, Jesimar da Silva
Sistema autônomo para supervisão de missão e segurança de voo em VANTs / Jesimar da Silva Arantes; orientador Claudio Fabiano Motta Toledo. -- São Carlos, 2019.
292 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2019.

1. Veículos Aéreos Não Tripulados. 2. Sistema Autônomo. 3. Arquitetura Embarcada. 4. Planejamento de Missão. 5. Replanejamento de Rota. I. Toledo, Claudio Fabiano Motta, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Jesimar da Silva Arantes

Autonomous system for mission control and flight safety in UAVs

Doctoral dissertation submitted to the Institute of Mathematics and Computer Sciences – ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *EXAMINATION BOARD PRESENTATION COPY*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Claudio Fabiano Motta Toledo

USP – São Carlos
April 2019

À minha família,
meu pai Jésus e minha mãe Sirley,
que estiveram sempre presentes, me apoiando, incentivando e
ajudando nos momentos em que mais precisei.

AGRADECIMENTOS

À Universidade de São Paulo (USP) e ao Instituto de Ciências Matemáticas e de Computação (ICMC) por me proporcionarem um ambiente de estudo adequado à minha formação. À Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processo número 2015/23182-2, e à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pelo apoio financeiro durante a realização deste trabalho. Aos professores da USP, em especial, aos professores do ICMC, pelos ensinamentos transmitidos.

Ao professor Claudio Toledo, pela orientação do trabalho, paciência durante nossas longas discussões e ensinamentos transmitidos, que foram de grande relevância para a realização deste trabalho e para meu crescimento profissional. Ao professor Onofre Trindade pelas inúmeras dicas e discussões sobre VANTs, que auxiliaram muito na realização deste trabalho. Ao professor Eduardo Simões, pelo grande auxílio nos experimentos realizados. Ao grupo de pesquisa, pelas inúmeras reuniões e apresentações, que serviram para ampliação do meu conhecimento nas mais diversas áreas da ciência. À Veronica Vannini, André Missaglia, Fabiano Paulelli, Marcelo Duchêne, Bárbara Castanheira, Rafael Sanches, Fernanda Guidotti, Pedro Natali e Rafael Pinho pelo auxílio prestado no desenvolvimento deste trabalho.

À todos os integrantes do Laboratório de Computação Reconfigurável (LCR), pelo apoio, companheirismo e amizade. E aos ex-colegas de apartamento Francisco de Assis e Julian Mariño, pelas muitas discussões, que contribuíram e auxiliaram nas minhas pesquisas.

Agradeço, sobretudo, ao meu irmão Márcio, pelas inumeráveis dicas e discussões, que auxiliaram a realização deste trabalho, além da constante amizade.

Por fim, agradeço à minha esposa, Vívian, por fazer parte deste momento em minha vida e pelas muitas dicas e correções ao longo do texto.

Este trabalho somente foi possível, pois estive apoiado sobre os ombros desses gigantes. Muito obrigado por tudo!!!

“ Se você tem uma maçã e eu tenho uma maçã e nós trocamos as maçãs, então você e eu ainda teremos uma maçã. Mas se você tem uma ideia e eu tenho uma ideia e nós trocamos essas ideias, então cada um de nós terá duas ideias. ”

George Bernard Shaw

RESUMO

ARANTES, J. S. **Sistema autônomo para supervisão de missão e segurança de voo em VANTs** . 2019. 292 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2019.

O presente documento tem por objetivo apresentar a tese desenvolvida no programa de doutorado em Ciência da Computação e Matemática Computacional do ICMC/USP. Esta tese aborda o desenvolvimento de sistemas autônomos, de baixo custo, para supervisão de missão e segurança de voo em Veículos Aéreos Não Tripulados (VANTs). A supervisão da missão é assegurada através da implementação de um sistema do tipo *Mission Oriented Sensor Array* (MOSA), responsável pelo adequado cumprimento da missão. A segurança de voo é garantida pelo sistema *In-Flight Awareness* (IFA), que visa monitorar o funcionamento da aeronave. Os assuntos missão e segurança são complexos e os sistemas MOSA e IFA foram idealizados e desenvolvidos de forma independente, fundamentando-se na ideia de separação de interesses. O desenvolvimento desses sistemas foi baseado em dois modelos de referência: MOSA e IFA, propostos pela literatura. Em trabalhos anteriores da literatura, alguns sistemas do tipo MOSA e IFA foram propostos para situações específicas de missão. Numa outra abordagem, esta tese propõe um único sistema MOSA e IFA capaz de se adequar a um conjunto distinto de missões. Neste trabalho, foi desenvolvida toda arquitetura de comunicação que integra os sistemas MOSA e IFA. No entanto, apenas esses dois sistemas não são suficientes para fazer a execução da missão com segurança, necessitando-se de um sistema capaz de se comunicar com o Piloto Automático (AP) do VANT. Logo, um sistema capaz de enviar requisições e comandos ao AP foi também implementado. Através desses três sistemas, missões autônomas com desvio de obstáculos puderam ser realizadas sem intervenção humana, mesmo diante de situações críticas ao voo. Assegurar os aspectos de segurança e missão pode se tornar conflitante durante o voo, pois em situações emergenciais deve-se abortar a missão. Diferentes estratégias para planejamento e replanejamento de rotas, baseadas em computação evolutiva e heurísticas, foram desenvolvidas e integradas nos sistemas MOSA e IFA. Os sistemas, aqui propostos, foram validados em quatro etapas: (i) experimentos com o simulador de voo FlightGear; (ii) simulações com a técnica *Software-In-The-Loop* (SITL); (iii) simulações com a técnica *Hardware-In-The-Loop* (HITL); (iv) voos reais. Na última etapa, os sistemas foram embarcados em dois modelos de VANTs, desenvolvidos pelo grupo de pesquisa. Durante a experimentação, alguns modelos de pilotos automáticos (APM e Pixhawk), computadores de bordo (Raspberry Pi 3, Intel Edison e BeagleBone Black), planejadores de missão e replanejadores de rotas emergenciais foram avaliados. Ao todo, três planejadores de rotas e oito replanejadores são suportados pela plataforma autônoma. O sistema autônomo desenvolvido permite alterar missões com diferentes características de hardware e de software de forma fácil e transparente, sendo, desse modo, uma arquitetura com características *plug and play*.

Palavras-chave: Veículos Aéreos Não Tripulados, Sistema Autônomo, Arquitetura Embarcada, Planejamento de Missão, Replanejamento de Rota .

ABSTRACT

ARANTES, J. S. **Autonomous system for mission control and flight safety in UAVs** . 2019. 292 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2019.

This document aims to present the thesis developed in the doctoral program in Computer Science and Computational Mathematics at ICMC/USP. This thesis addresses the development of low-cost autonomous systems for mission supervision and flight safety in Unmanned Aerial Vehicles (UAVs). The mission supervision is ensured through the implementation of a Mission Oriented Sensor Array (MOSA) system, which is responsible for the proper fulfillment of the mission. The flight safety is guaranteed by the In-Flight Awareness (IFA) system, which aims to monitor the aircraft operation. The mission and safety issues are complex, and the MOSA and IFA systems were idealized and developed independently, based on the idea of separation of concerns. The development of these systems was based on two reference models: MOSA and IFA, proposed in the literature. In previous works of the literature, some MOSA and IFA systems have been proposed for specific mission situations. In another approach, this thesis proposes a single MOSA and IFA system capable of adapting to a distinct set of missions. All the communication architecture that integrates the MOSA and IFA systems were developed in this work. However, only these two systems are not sufficient to carry out the mission safely; a system that can communicate with the AutoPilot (AP) of the UAV its also needed. In this way, a system that is capable of sending commands and requests to the AP was implemented in this work. Through these three systems, autonomous missions with a diversion of obstacles could be carried out without human intervention, even in critical situations to the flight. Ensuring the safety and mission aspects can become conflicting during the flight because in hazards situations the mission must be aborted. Different strategies for path planning and path replanning, based on evolutionary computation and heuristics, were developed and integrated into the MOSA and IFA systems. The systems proposed here were validated in four stages: (i) experiments with FlightGear flight simulator; (ii) simulations using Software-In-The-Loop (SITL); (iii) simulations using Hardware-In-The-Loop (HITL); (iv) real flights. In the last stage, the systems were embedded in two models of UAVs, developed by the research group. During the experiment were evaluated some models of autopilots (APM and Pixhawk), companion computers (Raspberry Pi 3, Intel Edison and BeagleBone Black), mission planners and emergency route planners. In all, three route planners and eight replanners are supported by the autonomous platform. The developed autonomous system allows changing missions with different hardware and software characteristics in an easy and transparent way, being, therefore, an architecture with Plug and play characteristics.

Keywords: Unmanned Aerial Vehicle, Autonomous System, Embedded Architecture, Mission Planning, Route Replanning .

LISTA DE ILUSTRAÇÕES

Figura 1 – Classificação dos veículos não tripulados. 46
Figura 2 – Classificação dos VANTs conforme: massa, altitude, distância e tempo de voo. 47
Figura 3 – Exemplos de VANTs de acordo com sua categoria. 48
Figura 4 – Veículos aéreos não tripulados utilizados. 49
Figura 5 – Ângulos de rotação e eixos principais do VANT. 50
Figura 6 – Estações de controle de solo avaliadas. 52
Figura 7 – Simuladores de voo avaliados. 54
Figura 8 – Pilotos automáticos avaliados. 56
Figura 9 – Computadores embarcados avaliados. 59
Figura 10 – Componentes aviônicos utilizados. 61
Figura 11 – Esquema da arquitetura de simulação SITL montada. 64
Figura 12 – Esquema da arquitetura de simulação HITL montada. 65
Figura 13 – Esquema da arquitetura de simulação HITL com CC. 66
Figura 14 – Classificação das arquiteturas de controle em sistemas robóticos. 67
Figura 15 – Arquitetura utilizada no sistema de busca e resgate no deserto. 82
Figura 16 – Arquitetura utilizada no sistema de controle da trajetória. 82
Figura 17 – Arquitetura utilizada no sistema de pulverização de pragas agrícolas. 83
Figura 18 – Arquitetura utilizada no sistema para evitar colisões com obstáculos. 84
Figura 19 – Arquitetura utilizada no sistema de tomada de decisão para pulverização. . . 85
Figura 20 – Fluxograma para sistema de tomada de decisão durante a missão do VANT. 85
Figura 21 – Arquitetura utilizada no sistema de monitoramento de ruído ambiental. . . . 86
Figura 22 – Arquitetura utilizada no sistema de detecção e desvio de obstáculos. 86
Figura 23 – Diagrama funcional básico da arquitetura MOSA. 93
Figura 24 – Processo sequencial para evitar ou mitigar acidentes com o VANT. 97
Figura 25 – Fluxograma de decisão do sistema IFA^{2}S de alto nível. 98
Figura 26 – Cenário geral do problema abordado. 104
Figura 27 – Cenário ilustrativo do plano de missão gerenciado pelos sistemas IFA e MOSA. 105
Figura 28 – Visão geral da arquitetura proposta. 116
Figura 29 – Hardware dos VANTs utilizados nos experimentos. 117
Figura 30 – Módulos e comunicação entre os sistemas MOSA, IFA, S2DK e UAV-GCS. 118
Figura 31 – Fluxo de execução para o módulo de controle da missão no MOSA. 119
Figura 32 – Fluxo de execução para o módulo do gerenciador de segurança no IFA. . . . 120
Figura 33 – Fluxograma de execução dos sistemas MOSA e IFA. 121

Figura 34 – Serviços e operações disponíveis através do sistema S2DK. 122
Figura 35 – Tipos de *waypoints* definidos na arquitetura proposta. 123
Figura 36 – Detecção, diagnóstico e tomada de decisão do sistema IFA. 123
Figura 37 – Fluxo de transição dos modos de voo do sistema autônomo implementado. . 125
Figura 38 – Sistemas de comunicação disponíveis no VANT utilizado. 127
Figura 39 – Comunicação e interação entre os módulos/componentes do CC, AP e PC. . 127
Figura 40 – Trocas de mensagens de comunicação entre o MOSA e o IFA. 128
Figura 41 – Estratégias implementadas executando os métodos em paralelo. 135
Figura 42 – Arquitetura do sistema embarcado no módulo de replanejamento de rotas. . 135
Figura 43 – Arquitetura do sistema embarcado no módulo de planejamento de rotas. . . 136
Figura 44 – Conversão entre sistemas de ângulos da aeronáutica e matemática. 138
Figura 45 – Protocolo especificado para criação do mapa e definição da missão. 139
Figura 46 – Fluxo de avaliação da arquitetura de hardware e software P&P. 141
Figura 47 – Conexões de hardware entre o PC e CC para fazer os experimentos HITL. . 142
Figura 48 – Conexões de hardware entre o CC, AP e PC para fazer os experimentos reais. 143
Figura 49 – Separação de interesses dos sistemas MOSA e IFA em nível de hardware. . 144
Figura 50 – *Framework* ProOF utilizado no desenvolvimento dos métodos. 145
Figura 51 – Esquema arquitetural do software UAV-Mission-Creator baseado em E/S. . . 148
Figura 52 – Resumo dos softwares instalados para execução do sistema autônomo proposto. 149
Figura 53 – Interface gráfica do software UAV-Manager. 150
Figura 54 – Resumo das formas de execução do sistema autônomo proposto. 150
Figura 55 – Interface gráfica do software UAV-GCS. 151
Figura 56 – Diagrama de comunicação do UAV-GCS com IFA, MOSA, S2DK e OD. . . 153
Figura 57 – Simulações utilizando instâncias de mapas artificiais. 154
Figura 58 – Número de avaliações por instância no método HGA4m. 165
Figura 59 – Tamanho da rota por instância no método HGA4m. 165
Figura 60 – Número de avaliações por instância no método MPGA4s. 166
Figura 61 – Local de pouso de ambas as arquiteturas de hardware para o MPGA4s. . . . 167
Figura 62 – Resultado do estudo de caso em um cenário do mundo real no Campus 2-USP. 168
Figura 63 – Resultado do estudo de caso em um cenário do mundo real utilizando o GA4s. 173
Figura 64 – Resultados obtidos no estudo de caso com simulação SITL. 175
Figura 65 – Resultados obtidos após o IFA detectar falha e fazer o pouso emergencial. . 176
Figura 66 – Resultados obtidos após o IFA detectar uma falha crítica. 176
Figura 67 – Visão geral dos quatro cenários utilizados na validação de arquitetura. . . . 179
Figura 68 – Resultados das rotas obtidas no cenário Case-I. 179
Figura 69 – Resultados das rotas obtidas no cenário Case-II. 181
Figura 70 – Resultados das rotas obtidas no cenário Case-III. 182
Figura 71 – Resultados das rotas obtidas no cenário Case-IV. 184
Figura 72 – Trajetória 3D seguida pelo VANT. 185

Figura 73 – Fluxo de avaliação da arquitetura nos experimentos com voo real. 187
Figura 74 – Comparação de desempenho entre os *compunions computers*. 189

LISTA DE QUADROS

Quadro 1 – Modos de voo suportados pelos pilotos automáticos APM e Pixhawk. . . . 58
Quadro 2 – Escala de graus de automação de Sheridan adaptada a aeronaves. 72
Quadro 3 – Características autônomas implementadas pelos pilotos automáticos. 73
Quadro 4 – Comparação entre as implementações dos sistemas: MOSA Figueira x MOSA Arantes. 95
Quadro 5 – Comparação entre as implementações dos sistemas: IFA Mattei x IFA Arantes. 102
Quadro 6 – Interface de requisições GET entre os sistemas MOSA/IFA e o S2DK. . . . 155
Quadro 7 – Interface de requisições/ações POST entre os sistemas MOSA/IFA e o S2DK. 156
Quadro 8 – Propriedades do arquivo de saída com dados da missão (`log-aircraft.csv`). 217

LISTA DE ALGORITMOS

Algoritmo 1 – Evolução Diferencial para segurança 134
Algoritmo 2 – Multi-Start para segurança 134

LISTA DE TABELAS

Tabela 1 – Especificações técnicas das aeronaves iDroneAlpha, iDroneBeta, Rascal 110 e Ararinha. 50
Tabela 2 – Comparações entre as estações de controle de solo avaliadas. 53
Tabela 3 – Características dos modelos de aeronaves disponíveis no FightGear. 54
Tabela 4 – Comparações entre os simuladores avaliados. 55
Tabela 5 – Comparações entre os pilotos automáticos avaliados. 57
Tabela 6 – Comparações entre os *companion computers* avaliados. 59
Tabela 7 – Classificação dos níveis de autonomia em sistemas robóticos. 71
Tabela 8 – Comparando contribuições baseadas em algumas características. 88
Tabela 9 – Resumo da representação do problema de planejamento/replanejamento de rotas. 113
Tabela 10 – Preços e pesos médios dos equipamentos utilizados na arquitetura de hardware. 129
Tabela 11 – Conjunto de comandos de teclado e de voz permitidos para controlar o VANT. 152
Tabela 12 – Conjunto de instâncias de mapas utilizado nos experimentos. 153
Tabela 13 – Valores das constantes do VANT utilizadas nas equações. 158
Tabela 14 – Parâmetros do piloto automático utilizados nas equações e nos experimentos. 158
Tabela 15 – Resumo dos principais códigos desenvolvidos neste trabalho. 160
Tabela 16 – Configurações utilizadas nos métodos HGA4m e MPGA4s. 164
Tabela 17 – Teste de Kruskal-Wallis para o método HGA4m. 166
Tabela 18 – Resumo dos locais de pouso do MPGA4s em ambas as arquiteturas de hardware. 167
Tabela 19 – Teste de Kruskal-Wallis para o método MPGA4s 168
Tabela 20 – Diferentes simulações efetuadas utilizando SITL para validar as rotas. . . . 169
Tabela 21 – Configurações utilizadas nos métodos HGA4m e MPGA4s. 170
Tabela 22 – Resultados obtidos após avaliar diferentes estratégias em mapas artificiais. . 171
Tabela 23 – Teste de Kruskal-Wallis para arquitetura e situação crítica. 171
Tabela 24 – Teste de Dunn para comparação múltipla em mapas artificiais. 172
Tabela 25 – Resultados obtidos após avaliar diferentes estratégias no estudo do caso. . . 173
Tabela 26 – Diferentes simulações efetuadas utilizando SITL para validar as rotas do GA4s. 174
Tabela 27 – Diferentes simulações realizadas utilizando SITL para validar a arquitetura. 177
Tabela 28 – Síntese dos resultados avaliando o tempo de escrita dos *waypoints* no AP. . . 177
Tabela 29 – Resumo dos experimentos realizados por estudo de caso. 186
Tabela 30 – Resumo do tempo de processamento dos métodos. 187
Tabela 31 – Configurações utilizadas nos experimentos HITL executados. 188

Tabela 32 – Valores de ângulos de *roll* e *pitch* e número de satélites capturados. 189
Tabela 33 – Conjunto de componentes de software utilizado nos experimentos. 190
Tabela 34 – Conjunto de componentes de hardware utilizado nos experimentos. 191
Tabela 35 – Comparação entre os planejadores e replanejadores utilizados. 191

LISTA DE ABREVIATURAS E SIGLAS

2D	**2 D**imensões ou Bidimensional
3D	**3 D**imensões ou Tridimensional
3DR	**3D R**obotics
Acel.	**Acel**erômetro
ADS-B	(***A**utomatic **D**ependent **S**urveillance - **B**roadcast*)
ANAC	**A**gência **N**acional de **A**viação **C**ivil
AP	Piloto Automático (***A**uto**P**ilot*)
API	Interface de Programação de Aplicação (***A**pplication **P**rogramming **I**nterface*)
APM	(***A**rdu**P**ilot **M**ega*)
ARP	**A**eronave **R**emotamente **P**ilotada (*Remote Piloted Aircraft*)
AUV	Veículo Submarino Autônomo (***A**utonomous **U**nderwater **V**ehicle*)
B&C	(***B**ranch and **C**ut*)
Baro.	**Barô**metro
BBB	***B**eagle**B**one **B**lack*
BD	**B**anco de **D**ados
CC	(***C**ompanion **C**omputer*)
CCQSP4m	(***C**hance-**C**onstrained **Q**ualitative **S**tate **P**lan for **m**ission*)
CEP	Processamento de Eventos Complexos (***C**omplex **E**vent **P**rocessing*)
CNPP	Problema de Planejamento de Caminho Não Convexo com *Chance-constraint* (***C**hance-constraint **N**on-convex **P**ath-planning **P**roblem*)
CSV	Valores Separados por Vírgula (***C**omma-**S**eparated **V**alues*)
CTOL	Decolagem e Aterrissagem Convencional (***C**onventional **T**ake-**O**ff and **L**anding*)
DDD	Entediantes, Sujas e Perigosas (***D**ull, **D**irty and **D**angerous*)
DE4s	Evolução Diferencial para segurança (***D**ifferential **E**volution for **s**ecurity*)
E/S	**E**ntrada e **S**aída
EKF	Filtro de Kalman Estendido (***E**xtended **K**alman **F**ilter*)
ESC	Controlador Eletrônico de Velocidade (***E**lectronic **S**peed **C**ontroller*)
ESWA	(***E**xpert **S**ystems **W**ith **A**pplications*)
EUA	**E**stados **U**nidos da **A**mérica
FAA	Administração Federal de Aviação (***F**ederal **A**viation **A**dministration*)
FG	**F**light**G**ear

FMEA	Análise do Modo e Efeito de Falha (***F**ailure **M**odes and **E**ffect **A**nalysis*)
FTA	Análise de Árvore de Falhas (***F**ault **T**ree **A**nalysis*)
GA	Algoritmo Genético (***G**enetic **A**lgorithm*)
GA-GA-4s	Algoritmo Genético e Algoritmo Genético para segurança (***G**enetic **A**lgorithm and **G**enetic **A**lgorithm for **s**ecurity*)
GA-GH-4s	Algoritmo Genético e Heurística Gulosa para segurança (***G**enetic **A**lgorithm and **G**reedy **H**euristic for **s**ecurity*)
GA4s	Algoritmo Genético para segurança (***G**enetic **A**lgorithm for **s**ecurity*)
GCS	Estação de Controle de Solo (***G**round **C**ontrol **S**tation*)
GECCO	(***G**enetic and **E**volutionary **C**omputation **C**onference*)
GH	Heurística Gulosa (***G**reedy **H**euristic*)
GH4s	Heurística Gulosa para segurança (***G**reedy **H**euristic for **s**ecurity*)
Giro.	**Giro**scópio
GISA	**G**rupo de **I**nteresse em **S**istemas **A**utônomos e Aplicações
GNSS	Sistema Global de Navegação por Satélite (***G**lobal **N**avigation **S**atellite **S**ystem*)
GPIO	Entrada e Saída de Propósito Geral (***G**eneral **P**urpose **I**nput **O**utput*)
GPL	Licença Pública Geral (***G**eneral **P**ublic **L**icense*)
GPS	Sistema de Posicionamento Global (***G**lobal **P**ositioning **S**ystem*)
GUI	Interface Gráfica do Usuário (***G**raphical **U**ser **I**nterface*)
HALE	Alta Altitude, Longa Resistência (***H**igh **A**ltitude, **L**ong **E**ndurance*)
HCCQSP	(***H**ybrid **C**hance-**C**onstrained **Q**ualitative **S**tate **P**lanning*)
HD	Disco Rígido (***H**ard **D**isk*)
HGA4m	Algoritmo Genético Híbrido para missão (***H**ybrid **G**enetic **A**lgorithm for **m**ission*)
HITL	(***H**ardware-**I**n-**T**he-**L**oop*)
HTTP	Protocolo de Transferência de Hipertexto (***H**yper**T**ext **T**ransfer **P**rotocol*)
IA	**I**nteligência **A**rtificial
ICAS	(***I**nternational **C**ouncil of the **A**eronautical **S**ciences*)
ICMC	**I**nstituto de **C**iências **M**atemáticas e **C**omputação
ICTAI	(***I**nternational **C**onference on **T**ools with **A**rtificial **I**ntelligence*)
IFA	Consciência em Voo (***I**n-**F**light **A**wareness*)
IFA^{2}S	(***I**n-**F**light **A**wareness **A**ugmentation **S**ystems*)
IMU	Unidade de Medição Inercial (***I**nertial **M**easurement **U**nit*)
IoD	Internet dos Drones (***I**nternet **o**f **D**rones*)
IoT	Internet das Coisas (***I**nternet **o**f **T**hings*)
IP	Protocolo de Internet (***I**nternet **P**rotocol*)
JSON	Notação de Objetos JavaScript (***J**ava**S**cript **O**bject **N**otation*)
KML	Linguagem de Marcação Keyhole (***K**eyhole **M**arkup **L**anguage*)

LALE	Baixa Altitude, Longa Resistência (***L**ow **A**ltitude, **L**ong **E**ndurance*)
LASE	Baixa Altitude, Curta Resistência (***L**ow **A**ltitude, **S**hort **E**ndurance*)
LCR	**L**aboratório de **C**omputação **R**econfigurável
LED	Diodo Emissor de Luz (***L**ight **E**mitting **D**iode*)
LGPL	(***L**esser **G**eneral **P**ublic **L**icense*)
LIDAR	Detecção e Medição de Distância por Luz (***LI**ght **D**etection **A**nd **R**anging*)
Lin.	**Lin**ux
LOWAS	(***L**IDAR **O**bstacle **W**arning and **A**voidance **S**ystem*)
Mac.	**Mac** OS X
Mag.	**Mag**netômetro
MALE	Média Altitude, Longa Resistência (***M**edium **A**ltitude, **L**ong **E**ndurance*)
MAV	(***M**icro **A**erial **V**ehicle*)
MAVLink	(***M**icro **A**ir **V**ehicle **Link***)
MBD	Projeto Baseado em Modelo (***M**odel-**B**ased **D**esign*)
MDD	Desenvolvimento Orientado a Modelos (***M**odel **D**riven **D**evelopment*)
MORSE	(***M**odular **O**pen**R**obots **S**imulation **E**ngine*)
MOSA	Arranjos de Sensores Orientados à Missão (***M**ission **O**riented **S**ensor **A**rray*)
MPGA4s	Algoritmo Genético Multi-Populacional para segurança (***M**ulti-**P**opulation **G**enetic **A**lgorithm for **s**ecurity*)
MS4s	Multi-Start para segurança (***M**ulti-**S**tart for **s**ecurity*)
MSFS	(***M**icro**S**oft **F**light **S**imulator*)
N/A	**N**ão **A**plicável (*Not Applicable*)
N/D	**N**ão **D**isponível (*Not Available*)
NFZ	Zona de Exclusão Aérea (***N**o-**F**ly **Z**one*)
OD	**O**racle **D**rone
OODA	Observação, Orientação, Decisão e Ação (***O**bservation, **O**rientation, **D**ecision, **A**ction*)
P&P	(***P**lug and **P**lay*)
PC	Computador Pessoal (***P**ersonal **C**omputer*)
PLIM	**P**rogramação **L**inear **I**nteira-**M**ista (*Mixed-Integer Linear Programming*)
PNLIM	**P**rogramação **N**ão **L**inear **I**nteira-**M**ista (*Mixed-Integer NonLinear Programming*)
ProOF	(***Pro**fessional **O**ptimization **F**ramework*)
PWM	Modulação por Largura de Pulso (***P**ulse **W**idth **M**odulation*)
QGC	**QG**round**C**ontrol
RAM	Memória de Acesso Aleatório (***R**andom **A**ccess **M**emory*)
RC	Controle de Rádio (***R**adio **C**ontrol*)
RGB	Vermelho, Verde e Azul (***R**ed, **G**reen and **B**lue*)

ROS	Sistema Operacional de Robôs (***R**obot **O**perating **S**ystem*)
ROV	Veículo Submarino Operado Remotamente (***R**emotely **O**perated Underwater **V**ehicle*)
RPi	**R**aspberry **Pi**
RTL	Retorne ao Lançamento (***R**eturn **T**o **L**aunch*)
RTOS	Sistema Operacional de Tempo Real (***R**eal-**T**ime **O**perating **S**ystem*)
S2DK	(***S**ervices to **D**rone**K**it*)
SD	(***S**ecure **D**igital*)
SdI	**S**eparação **d**e **I**nteresse
SEE	**S**istemas **E**mbarcados e **E**volutivos
Sim.	**Sim**ulador
SisVANT	**Sis**tema de **V**eículo **A**éreo **N**ão **T**ripulado (*Unmanned Aerial Systems*)
SITL	(***S**oftware-**I**n-**T**he-**L**oop*)
SO	**S**istema **O**peracional (*Operational System*)
SOA	Arquitetura Orientada a Serviços (***S**ervice-**O**riented **A**rchitecture*)
SSC	Departamento de **S**istemas de **C**omputação
SSH	(***S**ecure **Sh**ell*)
SSI	(***S**mart **S**ensor **I**nterface*)
SSN	Rede de Sensores Sonoros (***S**ound **S**ensor **N**etwork*)
SSP	(***S**mart **S**ensor **P**rotocol*)
STPA	Teórico - Sistêmico de Análise de Processos (***S**ystem **T**heoretic **P**rocess **A**nalysis*)
SUAS	(***S**mall **U**nmanned **A**ircraft **S**ystem*)
TCP	Protocolo de Controle de Transmissão (***T**ransmission **C**ontrol **P**rotocol*)
TUGV	Veículo Terrestre Não Tripulado Teleoperado (***T**ele-operated **U**nmanned **G**round **V**ehicle*)
UDP	Protocolo de Datagrama de Usuário (***U**ser **D**atagram **P**rotocol*)
UGV	Veículo Terrestre Autônomo (***U**nmanned **G**round **V**ehicle*)
USP	**U**niversidade de **S**ão **P**aulo
USV	Veículo de Superfície Não Tripulado (***U**nmanned **S**urface **V**ehicle*)
VAANT	**V**eículo **A**éreo **A**utônomo **N**ão **T**ripulado
VANT	**V**eículo **A**éreo **N**ão **T**ripulado (*Unmanned Aerial Vehicle*)
VMNT	**V**eículo **M**arinho **N**ão **T**ripulado (*Unmanned Marine Vehicles*)
VNT	**V**eículo **N**ão **T**ripulado (*Unmanned Vehicle*)
VSNT	**V**eículo **S**ubmarino **N**ão **T**ripulados (*Unmanned Underwater Vehicle*)
VTNT	**V**eículo **T**errestre **N**ão **T**ripulado (*Unmanned Ground Vehicles*)
VTOL	Decolagem e Aterrissagem Vertical (***V**ertical **T**ake-**O**ff and **L**anding*)
Wi-Fi	Fidelidade Sem Fio (***Wi**reless **Fi**delity*)
Win.	**Win**dows
XML	Linguagem de Marcação Extensível (*e**X**tensible **M**arkup **L**anguage*)

LISTA DE SÍMBOLOS

Constantes —

$\mathcal{R}_E$ — Raio equatorial do planeta Terra (em metros)

$\mathcal{R}_M$ — Raio meridional do planeta Terra (em metros)

$\mathcal{R}_T$ — Raio médio do planeta Terra (em metros)

$\mathcal{F}$ — Fator de conversão de coordenadas cartesianas para geográficas ou de metros para graus

$\mathcal{E}_1$ — Percentagem de tempo extra aferida na execução de RTL

$\mathcal{E}_2$ — Percentagem da distância máxima de alcance no voo

$\mathcal{K}_T^E$ — Eficiência energética média de tempo de voo (em % de bateria/segundo)

$\mathcal{K}_N^E$ — Eficiência energética média de navegação (em % de bateria/metro)

$\mathcal{K}_{N_{hrz}}^E$ — Eficiência energética média da navegação na horizontal (em % de bateria/metro)

$\mathcal{K}_{N_{up}}^E$ — Eficiência energética média da navegação vertical para cima (em % de bateria/metro)

$\mathcal{K}_{N_{down}}^E$ — Eficiência energética média da navegação vertical para baixo (em % de bateria/metro)

Parâmetros —

v_{hrz} — Velocidade de deslocamento horizontal [parâmetro do AP] (em metros/segundo)

v_{up} — Velocidade de deslocamento vertical para cima [parâmetro do AP] (em metros/segundo)

v_{down} — Velocidade de deslocamento vertical para baixo [parâmetro do AP] (em metros/segundo)

h_{rtl} — Altitude de realização da operação de RTL [parâmetro do AP] (em metros)

Δt — Intervalo de tempo ou fator de discretização do tempo (em segundos)

T — Horizonte de tempo utilizado no planejador/replanejador (em segundos)

$\hat{x}_0$ — Posição inicial esperada do VANT

$\hat{x}_{goal}$ — Posição objetivo esperada do VANT

m — Massa do VANT incluindo *payload* (em quilograma)

$\bar{\lambda}$ — Coordenada geográfica de longitude de um ponto de referência (em graus)

$\bar{\gamma}$ — Coordenada geográfica de latitude de um ponto de referência (em graus)

σ_A — Desvio padrão associado a posição inicial

σ_B — Desvio padrão associado ao ruído aplicado a posição

$\mathbb{A}$ — Matriz 4×4 para o estado do VANT utilizada na transição de estados

$\mathbb{B}$ — Matriz 4×2 para o controle da dinâmica do VANT utilizada na transição de estados

Σ_{x_0} — Matriz 4×4 de covariância associada à incerteza da posição inicial do VANT

Σ_{ω_t} — Matriz 4×4 de covariância aplicada na transição de estado

Variáveis — ——

$\mathbf{x}_t$ — Estado da aeronave no instante t

$\mathbf{u}_t$ — Controle aplicado a aeronave no instante t

$\mathbf{x}_0$ — Estado da aeronave no instante inicial 0

$\mathbf{x}_T$ — Estado da aeronave no instante final T

$\mathbf{x}_{goal}$ — Posição do objetivo (ponto-alvo) especificado pelo projetista

p — Posição da aeronave em coordenadas cartesianas

p_x — Posição no eixo x da aeronave ao longo da linha do equador (em metros)

p_y — Posição no eixo y da aeronave ao longo da linha do meridiano de Greenwich (em metros)

v — Velocidade na horizontal da aeronave (em metros/segundo)

v_x — Velocidade no eixo x do VANT (em metros/segundo)

v_y — Velocidade no eixo y do VANT (em metros/segundo)

a — Aceleração na horizontal do VANT (em metros/segundo^2)

a_x — Aceleração no eixo x do VANT (em metros/segundo^2)

a_y — Aceleração no eixo y do VANT (em metros/segundo^2)

α — Ângulo de orientação (direção) na horizontal da aeronave (em radianos)

ε — Variação angular (rotação) do VANT em torno do eixo z (em radianos)

α_{math} — Ângulo de orientação (*heading*) baseada no ângulo da matemática (em graus)

α_{aero} — Ângulo de orientação (*heading*) baseada no ângulo da aeronáutica (em graus)

d_c — Distância crítica percorrida durante o replanejamento de rotas (em metros)

ϕ — Ângulo de orientação de *roll* (em radianos)

θ — Ângulo de orientação de *pitch* (em radianos)

λ — Coordenada geográfica de longitude da aeronave (em graus)

γ — Coordenada geográfica de latitude da aeronave (em graus)

p_{lat}^{uav} — Posição de latitude (coordenada geográfica) atual do VANT (em graus)

p_{lng}^{uav} — Posição de longitude (coordenada geográfica) atual do VANT (em graus)

p_{altrel}^{uav} — Posição de altitude relativa atual do VANT (em metros)

p_{lat}^{home} — Posição de latitude (coordenada geográfica) do *home* do VANT (em graus)

p_{lng}^{home} — Posição de longitude (coordenada geográfica) do *home* do VANT (em graus)

p_{altrel}^{home} — Posição da altitude relativa do *home* do VANT (em metros)

B_{level} — Nível atual da bateria do VANT (em porcentagem de bateria)

D_{hrz} — Distância euclidiana na horizontal do VANT até o *home* (em metros)

B_{rtl}^{E} — Consumo de combustível estimado para fazer o RTL (em porcentagem de bateria)

D_{max}^{E} — Distância máxima estimada que o VANT consegue alcançar (em metros)

T_{rtl}^{E} — Tempo estimado para fazer a operação de RTL (em segundos)

T_{max}^{E} — Tempo máximo estimado de voo (em segundos)

L — Comprimento do arco da circunferência (em metros)

τ — Ângulo da circunferência (em graus)

ω_t — Ruído Gaussiano branco aditivo (perturbação ou incerteza) no instante t

F_{Ψ} — Função de transição de estados não linear para uma situação crítica do tipo Ψ

F_D — Força do arrasto (*drag*) associada à resistência do ar do VANT (em newton)

$Configurações$ — ——————————————————————————————

$\mathcal{M}$ — Representa o mapa, o plano da missão e a altitude de cruzeiro, em que $\mathcal{M} = \langle M_{map}, M_{goal}, M_{alt} \rangle$

$\mathcal{H}$ — Representa a configuração da arquitetura de hardware, em que $\mathcal{H} = \langle H_{ap}, H_{cc}, H_{sensor}, H_{actuator} \rangle$

$\mathcal{S}$ — Representa a configuração da arquitetura de software, em que $\mathcal{S} = \langle S_{mosa}, S_{ifa} \rangle$

$\mathcal{S}^*$ — Representa a solução ótima ou próxima do ótimo encontrada para o problema

M_{map} — Representa o mapa da missão, em que $M_{map} = \langle \mathbf{r}, \mathbf{h}, \mathbf{c} \rangle$

M_{goal} — Representa o plano da missão ou conjunto de pontos-alvo, em que $M_{goal} = \langle w^1, w^2, \cdots \rangle$

M_{alt} — Representa a altitude de voo cruzeiro da missão (em metros), em que $M_{alt} \in \mathbb{R}_+^*$

H_{ap} — AP utilizado na missão, em que $H_{ap} \in \{APM, Pixhawk\}$

H_{cc} — CC utilizado na missão, em que $H_{cc} \in \{IntelEdison, RaspberryPi, BeagleBoneBlack\}$

H_{sensor} — Lista com sensores da missão, em que $H_{sensor} = [CameraRGB, Sonar, Temperature]$

$H_{actuator}$ — Lista com atuadores da missão, em que $H_{actuator} = [Buzzer, Spraying, Parachute, LED]$

S_{mosa} — Representa o sistema MOSA, em que $S_{mosa} = \langle \mathcal{P}_P, \mathcal{L}_P, \mathcal{T}_{SC_P}, \Delta_P \rangle$

S_{ifa} — Representa o sistema IFA, em que $S_{ifa} = \langle \mathcal{P}_R, \mathcal{L}_R, \mathcal{T}_{SC_R}, \Delta_R, \mathcal{W}_R \rangle$

$\mathcal{P}_P$ — Representa o planejador de rotas utilizado, em que $\mathcal{P}_P \in \{HGA4m, CCQSP4m, FixedRoute4m\}$

$\mathcal{L}_P$ — Local de execução do planejador de rotas, em que $\mathcal{L}_P \in \{onboard, offboard\}$

$\mathcal{T}_{SC_P}$ — Critério de parada do planejador baseado em tempo (em segundos), em que $\mathcal{T}_{SC_P} \in \mathbb{R}^*_+$

Δ_P — Nível de risco assumido pelo planejador, em que $\Delta_P \in [0, \frac{1}{2}]$ e $\Delta_P \in \mathbb{R}$

$\mathcal{P}_R$ — Representa o replanejador de rotas utilizado, em que $\mathcal{P}_R \in \{GA4s, MPGA4s, DE4s, GH4s, MS4s, GA\text{-}GA\text{-}4s, GA\text{-}GH\text{-}4s, Pre\text{-}Planned4s\}$

$\mathcal{L}_R$ — Local de execução do replanejador de rotas, em que $\mathcal{L}_R \in \{onboard, offboard\}$

$\mathcal{T}_{SC_R}$ — Critério de parada do replanejador baseado em tempo (em segundos), em que $\mathcal{T}_{SC_R} \in \mathbb{R}^*_+$

Δ_R — Nível de risco assumido pelo replanejador, em que $\Delta_R \in [0, \frac{1}{2}]$ e $\Delta_R \in \mathbb{R}$

$\mathcal{W}_R$ — Número máximo de *waypoints* utilizado pelo replanejador de rotas, em que $\mathcal{W}_R \in \mathbb{N}$

$\mathbf{r}$ — Conjunto de regiões convexas, em que $\mathbf{r} = \{r^1, r^2, \cdots\}$

$\mathbf{h}$ — Conjunto de altitudes das regiões convexas (em metros), em que $\mathbf{h} = \{h^1, h^2, \cdots\}$

$\mathbf{c}$ — Conjunto de classificações das regiões, em que $\mathbf{c} = \{c^1, c^2, \cdots\}$ e $c \in \Phi$

$\mathbb{O}^n_j$ — j-ésima região convexa que representa um obstáculo, em que $\mathbb{O}^n_j \in \mathbf{r}$

$\mathbb{O}^p_j$ — j-ésima região convexa que representa uma região penalizadora, em que $\mathbb{O}^p_j \in \mathbf{r}$

$\mathbb{O}^b_j$ — j-ésima região convexa que representa uma região bonificadora, em que $\mathbb{O}^b_j \in \mathbf{r}$

t_{video} — Tempo de filmagem de uma região qualquer (em segundos), em que $t_{video} \in \mathbb{N}$

$n_{picture}$ — Quantidade de fotografias retiradas em sequência, em que $n_{picture} \in \mathbb{N}$

$d_{picture}$ — Tempo (*delay*) entre duas fotografias em sequência (em segundos), em que $d_{picture} \in \mathbb{N}$

$Classificações$ — ————————————————————————————————————

Ψ — Conjunto de situações críticas que podem ocorrer na aeronave, em que $\Psi = \{\Psi_m, \Psi_b, \Psi_{s1}, \Psi_{s2}\}$

Ψ_b — Situação crítica na bateria, em que $\Psi_b \in \Psi$

Ψ_m — Situação crítica no motor, em que $\Psi_m \in \Psi$

Ψ_{s1} — Situação crítica na semi-asa direita, em que $\Psi_{s1} \in \Psi$

Ψ_{s2} — Situação crítica na semi-asa esquerda, em que $\Psi_{s2} \in \Psi$

Φ — Conjunto de regiões do mapa, em que $\Phi = \{\Phi^n, \Phi^o, \Phi^p, \Phi^b, \Phi^s, \Phi^r\}$

Φ^n — Região do tipo não navegável, em que $\Phi^n \in \Phi$

Φ^o — Região do tipo obstáculo, em que $\Phi^o \in \Phi$

Φ^p — Região do tipo penalizadora, em que $\Phi^p \in \Phi$

Φ^b — Região do tipo bonificadora, em que $\Phi^b \in \Phi$

Φ^s — Região do tipo cênica, em que $\Phi^s \in \Phi$

Φ^r — Região do tipo remanescente, em que $\Phi^r \in \Phi$

Índices — ——————————————————————————————————————-

t — Índice que controla o instante de tempo, em que $t \in [0, T]$ e $t \in \mathbb{N}$

j — Índice que controla as regiões do mapa, em que $j \in [0, |\mathbf{r}|]$ e $j \in \mathbb{N}$

SUMÁRIO

1 INTRODUÇÃO 39
1.1 Contextualização 39
1.2 Motivação e Justificativas 41
1.3 Hipótese 42
1.4 Objetivos 42
1.5 Contribuições e Limitações 43
1.6 Organização do Texto 44

2 CONCEITOS FUNDAMENTAIS 45
2.1 Considerações Iniciais 45
2.2 Veículos Aéreos Não Tripulados 45
2.3 Componentes Básicos de um VANT 50
2.4 Estações de Controle de Solo 51
2.5 Simuladores de Voo 53
2.6 Pilotos Automáticos 55
2.7 Companion Computers 57
2.8 Aviônicos 60
2.9 Sensores e Atuadores da Aeronave 61
2.10 DroneKit 62
2.11 Simulação Software-In-The-Loop 63
2.12 Simulação Hardware-In-The-Loop 64
2.13 Arquiteturas de Controle de Robôs 66
2.14 Localização, Mapeamento e Navegação 67
2.15 Tolerância à Falhas 68
2.16 Autonomia em Sistemas Robóticos 71
2.17 Considerações Finais 72

3 REVISÃO BIBLIOGRÁFICA 75
3.1 Considerações Iniciais 75
3.2 Trabalhos Relacionados 75
3.3 Arquiteturas Propostas na Literatura 81
3.4 Considerações Finais 89

4 SISTEMAS MOSA E IFA 91

4.1 Considerações Iniciais 91
4.2 Sistema MOSA 91
4.3 Diferenças entre as Implementações do MOSA 94
4.4 Sistema IFA 96
4.5 Diferenças entre as Implementações do IFA 100
4.6 Considerações Finais 101

5 PROBLEMA ABORDADO 103
5.1 Considerações Iniciais 103
5.2 Descrição Geral do Cenário 103
5.3 Problema de Planejamento de Missão e Segurança 105
5.4 Modelagem do Problema 108
5.5 Considerações Finais 113

6 ARQUITETURA DE HARDWARE E SOFTWARE 115
6.1 Considerações Iniciais 115
6.2 Arquitetura de Hardware do VANT 116
6.3 Arquitetura de Software do VANT 117
6.4 Sistemas de Comunicação do VANT 126
6.5 Arquitetura de Baixo Custo 128
6.6 Considerações Finais 130

7 METODOLOGIA 131
7.1 Considerações Iniciais 131
7.2 Planejamento e Replanejamento de Rota do VANT 131
7.3 Conversões Utilizadas nos Planejadores/Replanejadores 136
7.4 Protocolo para Especificação do Mapa e da Missão 138
7.5 Avaliação da Arquitetura 140
7.6 Separação de Interesses 143
7.7 Framework ProOF 144
7.8 Considerações Finais 145

8 SISTEMAS DESENVOLVIDOS 147
8.1 Considerações Iniciais 147
8.2 Sistema UAV-Mission-Creator 147
8.3 Sistema UAV-Manager 149
8.4 Sistema UAV-GCS 150
8.5 Sistema UAV-S2DK 155
8.6 Sistema UAV-IFA 156
8.7 Sistema UAV-MOSA 159
8.8 Considerações Finais 160

9 RESULTADOS 163
9.1 Considerações Iniciais 163
9.2 Resultados do Artigo do GECCO 2017 163
9.3 Resultados do Artigo do ICTAI 2017 169
9.4 Resultados do Artigo do ICAS 2018 174
9.5 Resultados do Artigo do ESWA 2019 177
9.6 Resultados Finais da Tese 186
9.7 Considerações Finais 192

10 CONSIDERAÇÕES FINAIS 193
10.1 Avaliação Geral 193
10.2 Contribuições Alcançadas 194
10.3 Produção Científica 195
10.4 Dificuldades Enfrentadas 195
10.5 Trabalhos Futuros 196

REFERÊNCIAS 197

GLOSSÁRIO 209

APÊNDICE A ARQUIVOS DE ENTRADA E SAÍDA DE DADOS . . 213

APÊNDICE B ARQUITETURA EMBARCADA BASEADA EM ALGORITMOS GENÉTICOS 219

APÊNDICE C AVALIAÇÃO DE PLATAFORMAS DE HARDWARE PARA REPLANEJAMENTO DE ROTA 229

APÊNDICE D ARQUITETURA CONJUNTA DOS SISTEMAS MOSA E IFA ORIENTADA A SERVIÇOS 239

APÊNDICE E ARQUITETURA EMBARCADA DOS SISTEMAS MOSA E IFA 251

CAPÍTULO

1

INTRODUÇÃO

“ Não se espante com a altura do voo. Quanto mais alto, mais longe do perigo. Quanto mais você se eleva, mais tempo há de reconhecer uma pane. É quando se está próximo do solo que se deve desconfiar. ”

Santos Dumont

1.1 Contextualização

A presente proposta de doutorado visa o desenvolvimento de sistemas embarcados autônomos de baixo custo, voltados à execução de missão e à segurança de voo para Veículos Aéreos Não Tripulados (VANTs). Essas aeronaves estão se tornando cada vez mais populares, devido ao barateamento de tecnologias e dispositivos eletrônicos e à miniaturização dos componentes físicos necessários para a construção de aeronaves de pequeno porte (CHIARAMONTE, 2018). Com essa crescente utilização de VANTs, alguns questionamentos foram levantados em relação aos riscos envolvidos em sua operação sobre regiões povoadas. Assim, para sua integração ao espaço aéreo, a probabilidade de falha deve ser menor ou igual a aceita para a aviação geral (FAA, 2012). Nesse contexto, uma adequada execução da missão e uma constante avaliação de possíveis falhas são aspectos que devem ser considerados ao elaborar um sistema embarcado para VANTs.

O grupo de Sistemas Embarcados e Evolutivos (SEE) do Departamento de Sistemas de Computação (SSC) do ICMC/USP tem desenvolvido trabalhos com VANTs, que remetem a preocupações com execução da missão e com segurança de voo, em que se destacam o *Mission Oriented Sensor Array* (MOSA) e o *In-Flight Awareness* (IFA). O sistema supervisor da missão, MOSA, permite um adequado acompanhamento de toda a missão em execução pelo VANT (FIGUEIRA, 2016). O sistema supervisor da segurança em voo, IFA, monitora em tempo real o funcionamento da aeronave (MATTEI, 2015).

A literatura apresenta poucos trabalhos focados em arquiteturas/sistemas autônomos de

baixo custo para missão e segurança de VANTs, como o MOSA e o IFA. Dessa forma, uma arquitetura que garanta uma menor intervenção humana e, consequentemente, maior nível de autonomia para a aeronave foi proposta nesta tese integrando esses dois sistemas. Após uma revisão da literatura, observamos que várias arquiteturas apresentam foco no desenvolvimento de sistemas específicos para determinado tipo de aplicação, como em Brown, Estabrook e Franklin (2011), Prodan *et al.* (2013), Xue *et al.* (2016), Morozov e Janschek (2016), Ramasamy *et al.* (2016), Alsalam *et al.* (2017), Boubeta-Puig *et al.* (2018), Chiaramonte (2018). Assim, esta tese buscou preencher a lacuna citada, introduzindo uma arquitetura de propósito geral e com um nível de resiliência capaz de mitigar falhas, garantindo um adequado nível de autonomia para a aeronave.

A motivação para este trabalho está na oportunidade de convergir diversos resultados obtidos recentemente nas pesquisas em VANT, conduzidas pelo grupo SEE, dentro do SSC/ICMC/USP. Uma primeira versão do sistema IFA foi estabelecida em Mattei (2015). Da mesma forma, a definição do sistema MOSA foi estabelecida em Figueira (2016). Algoritmos voltados ao planejamento de missões, envolvendo Algoritmos Genéticos (GAs, do inglês *Genetic Algorithms*) combinados com resolução de modelos de Programação Linear Inteira-Mista (PLIM), foram desenvolvidos em Arantes (2017). Por fim, algoritmos voltados ao replanejamento de rotas para pouso emergencial, aplicando GA, Heurística Gulosa (GH, do inglês *Greedy Heuristic*) e modelos de PLIM, foram desenvolvidos em Arantes (2016).

Trabalhos anteriores sobre MOSA e IFA não apresentam uma implementação embarcada dos sistemas, focando mais no estabelecimento dos conceitos de monitoramento de missão e segurança. Além disso, nenhum trabalho abordou o funcionamento conjunto de tais sistemas. Dessa maneira, um sistema embarcado com as arquiteturas MOSA e IFA foi aqui desenvolvido e aplicado em dois VANTs de asa rotativa, nomeados iDroneAlpha e iDroneBeta. Essas aeronaves são quadricópteros e foram construídos durante este doutorado.

A autora Figueira (2016) destaca em seu trabalho que existem diversos MOSAs, cada um desses é específico para uma aplicação. A presente tese pretende desenvolver um único sistema MOSA, que seja modular e flexível o suficiente para atender diferentes aplicações apenas habilitando ou desabilitando os recursos que a aplicação utilizará. Como esperado, o sistema MOSA proposto não irá abarcar todos as aplicações possíveis, ou seja, todos os MOSAs possíveis. No entanto, pretende-se permitir que novos módulos possam ser facilmente integrados ao MOSA aqui proposto, abrangendo assim outras aplicações que dependam, por exemplo, de novos sensores ou atuadores. Da mesma forma, conforme ressaltado em Mattei (2015), não existe apenas um sistema IFA possível. O IFA aqui desenvolvido também será feito de forma modular e flexível para que diversos sensores possam ser habilitados ou desabilitados conforme a aplicação utilizada e os sensores disponíveis. Os sistemas desenvolvidos são distribuídos com código-fonte aberto, o que facilita também a integração de novos recursos e funcionalidades.

Sobre o sistema MOSA dois conceitos importantes podem ser estabelecidos:

- **MOSA Não-Adaptativo**: este sistema não consegue alterar o comportamento da missão durante o voo. Dessa forma, o MOSA não consegue contornar imprevistos ocorridos durante a missão. Após ocorrer um imprevisto, a única ação que pode ser executada é abortar a missão.

- **MOSA Adaptativo**: este sistema, por sua vez, consegue alterar o comportamento da missão durante o voo. Nesse sistema, o plano de voo pré-definido pode ser alterado dinamicamente durante o voo. Dessa maneira, o MOSA, baseado em novas variáveis, consegue contornar imprevistos através da reconfiguração da missão. Dentre as ações de reconfiguração permitidas têm-se: refazer a missão, trocar a missão, atualizar o plano de voo, ou ainda, abortar a missão.

O trabalho de Figueira (2016) abordou exclusivamente o sistema MOSA não-adaptativo. A presente tese trata ambos os tipos de sistemas: MOSA adaptativo e MOSA não-adaptativo.

1.2 Motivação e Justificativas

Os índices de acidentes e fatalidades aeronáuticos caíram nos últimos anos, como resultado direto do aprendizado acumulado a partir da investigação de suas causas (SAFETY, 2018). Já o número de acidentes envolvendo VANTs tem aumentado, conforme destacado em EASA (2017), devido, principalmente, a sua inserção no mercado de forma não controlada. Nos Estados Unidos da América (EUA), em 2016, o índice de acidentes da aviação geral foi de $5,93 \cdot 10^{-5}$, o que significa $5,93$ acidentes a cada 100 mil horas de voo. O número de acidentes com vítimas foi de $9,98 \cdot 10^{-4}$, portanto, pode-se concluir que, aproximadamente um sexto dos acidentes aéreos geram vítimas (NTSB, 2018). Esses números são relativamente baixos, quando comparados aos acidentes envolvendo transporte automotivo, que nos EUA geraram em 2016 cerca de 90 vezes mais vítimas que a aviação geral[1].

Conforme destacado em Mattei (2015), uma aeronave como o UAV Hermes 450 (aeronave de 2 milhões de dólares[2]) apresenta um acidente a cada 833 horas, já uma aeronave como o UAV Desert Hawk (aeronave de 224.000 dólares[3]) apresenta um acidente a cada 56 horas. Ao analisar o UAV Hermes 450, percebe-se que ele sofre um acidente a uma frequência de 20,2 vezes mais que uma aeronave tripulada. Já o UAV Desert Hawk sofre um acidente a uma frequência de 301,1 vezes mais que uma aeronave tripulada. Esses índices indicam que um VANT não tem a mesma confiabilidade de uma aeronave tripulada e indicam também uma tendência de que quanto menor o custo do VANT, menor a sua confiabilidade. Aeronaves com custos ainda menores sofrem acidentes a cada poucas horas de operação. Esse fato justifica o desenvolvimento de sistemas autônomos para segurança em voo, como o IFA. Não existe,

[1] <https://www.ntsb.gov/investigations/data/Pages/Data_Stats.aspx>
[2] <https://en.wikipedia.org/wiki/Elbit_Hermes_450>
[3] <http://www.deagel.com/Cannons-and-Gear/Desert-Hawk_a002338001.aspx>

obviamente, risco nulo na operação de uma aeronave, o que se busca é a manutenção de índices baixos o suficiente para sua aceitação e a busca incessante por sua redução contínua (MATTEI, 2015).

Empresas, como Amazon, Google, Microsoft e Uber têm investido na construção de aeronaves autônomas para realização de missões. A Amazon está, atualmente, desenvolvendo um sistema de entrega de encomendas, utilizando VANTs, chamado Amazon Prime Air, que promete realizar entregas com segurança em até 30 minutos (AMAZON, 2018b). A Google, em resposta a esse serviço, criou também um projeto para entrega de encomendas, chamado Wing (GOOGLE, 2019). Os projetos mencionados necessitam de um sistema supervisor de missão com segurança capaz de realizar a entrega no local, levando em consideração toda a complexidade do ambiente. O fato de se ter diversas empresas e grupos de pesquisa trabalhando na automação de VANTs motivam e justificam o desenvolvimento de sistemas, como o MOSA e IFA.

1.3 Hipótese

Esta pesquisa visa validar as seguintes hipóteses:

- É possível construir um VANT de baixo custo, que execute missões de forma autônoma, diante situações severas.
- É possível construir um VANT que faça tomadas de decisão *onboard* com as plataformas de hardware existentes.
- É possível construir uma arquitetura que consiga tratar diversas aplicações diferentes com poucas modificações no sistema.
- É possível construir uma arquitetura utilizando separação de interesse entre a missão e a segurança.

1.4 Objetivos

O principal objetivo deste trabalho é a proposição de uma arquitetura para VANTs de baixo custo e o desenvolvimento de sistemas relacionados, que apresentem as seguintes características:

- **Propósito geral**: diferentes missões podem ser incorporadas ao VANT sem necessidade de mudanças relevantes na arquitetura definida.
- **Resiliência**: os sistemas definidos na arquitetura devem prevenir a propagação de erros ou alterar seu comportamento visando o correto funcionamento da aeronave.

- **Autonomia**: sistemas que permitam que a aeronave opere com o menor nível possível de intervenção humana.

Nesse contexto, estabelecemos como meta atingir os seguintes objetivos específicos:

1. Implementar um sistema de controle de missão, baseado no MOSA, que inclua comportamentos reativos. Por exemplo, algoritmos que quando incorporados ao MOSA possibilitem tomadas de decisão em tempo real, levando a alteração da missão original a partir de novos dados obtidos pelo sensoriamento da aeronave.

2. Implementar um sistema de segurança de voo, baseado no IFA, capaz de lidar de forma robusta com diferentes tipos de falhas na aeronave. Por exemplo, algoritmos de replanejamento que quando incorporados, permitam o pouso da aeronave, em caso de situação crítica, assim como, sistemas para detecção e prevenção de falhas.

3. Possuir a capacidade de trocar módulos da arquitetura de forma transparente, tornando a arquitetura *Plug and Play* (P&P), na qual componentes, como diferentes modelos de aeronaves, pilotos automáticos, computadores de bordo, planejadores de missão e replanejadores de rotas emergenciais, possam ser trocados sem grandes alterações no hardware e software.

4. Avaliar o comportamento da arquitetura desenvolvida em experimentos utilizando simulador de voo, simulação *Software-In-The-Loop* (SITL), simulação *Hardware-In-The-Loop* (HITL) e voos reais.

1.5 Contribuições e Limitações

As contribuições esperadas com o desenvolvimento da presente tese são: (i) a obtenção de uma arquitetura de hardware e software para VANTs de baixo custo que possibilite a aeronave ter um alto grau de autonomia; (ii) a capacidade de executar algoritmos de forma rápida e com habilidade de realizar tomadas de decisão com intervenção mínima do piloto em solo; (iii) a divulgação do sistema autônomo de forma *open-source*, permitindo que toda a comunidade possa fazer uso desse sistema; (iv) a obtenção de uma plataforma embarcada para testes capaz de incorporar: algoritmos de planejamento de missão; algoritmos de replanejamento de rotas; algoritmos de função de sensores e coleta de dados em *logs* de voos para extração de conhecimento.

Este trabalho apresenta também alguns pontos que podem limitar o seu desenvolvimento, como: (i) a aeronave precisa conhecer o cenário de voo, já que qualquer eventual mudança no cenário pode afetar os resultados dos algoritmos de planejamento; (ii) a dificuldade inerente à realização de experimentos reais em campo, pois se trata de uma tarefa complexa que demanda tempo e disponibilidade de recursos financeiros.

1.6 Organização do Texto

A organização deste documento foi subdividida da seguinte forma:

- **Capítulo 2:** expõe uma revisão dos principais conceitos envolvidos;
- **Capítulo 3:** apresenta uma revisão dos principais trabalhos relacionados;
- **Capítulo 4:** apresenta os modelos de referência MOSA e IFA;
- **Capítulo 5:** define o problema abordado e sua modelagem;
- **Capítulo 6:** apresenta a arquitetura do sistema proposto;
- **Capítulo 7:** mostra a metodologia a ser utilizada;
- **Capítulo 8:** apresenta os sistemas desenvolvidos neste trabalho;
- **Capítulo 9:** mostra os resultados experimentais obtidos;
- **Capítulo 10:** encerra esta tese apontado as conclusões e os trabalhos futuros;
- **Apêndice A:** apresenta um conjunto de arquivos de entrada e saída utilizados;
- **Apêndice B:** apresenta um artigo completo publicado na conferência GECCO, em 2017;
- **Apêndice C:** mostra um artigo completo publicado na conferência ICTAI, em 2017;
- **Apêndice D:** expõe um artigo completo publicado na conferência ICAS, em 2018;
- **Apêndice E:** apresenta um artigo completo submetido no periódico ESWA, em 2019.

CAPÍTULO

2

CONCEITOS FUNDAMENTAIS

" A resposta certa, não importa nada: o essencial é que as perguntas estejam certas. "

Mario Quintana

2.1 Considerações Iniciais

Este capítulo apresentará os conceitos fundamentais necessários a compreensão deste trabalho. O capítulo está dividido em 15 seções que são: Veículos Aéreos Não Tripulados (VANTs); componentes básicos de um VANT; estações de controle de solo; simuladores de voo; pilotos automáticos; *companion computers*; aviônicos; sensores e atuadores da aeronave; Dronekit; simulações *software-in-the-loop*; simulações *hardware-in-the-loop*; tipos de arquiteturas de controle; localização, mapeamento e navegação; tolerância à falhas; e autonomia em sistemas robóticos. Os temas escolhidos para este capítulo têm como objetivo fornecer informações ao leitor, até mesmo àquele pouco conhecedor de temas aeronáuticos, para a compreensão desta tese.

2.2 Veículos Aéreos Não Tripulados

A Agência Nacional de Aviação Civil (ANAC) define VANT ou *drone* como sendo "Aeronave projetada para operar sem piloto a bordo e que não seja utilizada para fins meramente recreativos" (ANAC, 2012a, pág. 3). Essas aeronaves podem ser controladas à distância por meios eletrônicos e computacionais, por pessoas e por piloto automático (LEITE, 2013, pág. 4). Segundo ANAC (2012b, pág. 3), os VANTs necessitam de infraestrutura remota para sua operação por não possuírem piloto a bordo. Essa infraestrutura compõe o que se denomina Sistema de Veículo Aéreo Não Tripulado (SisVANT). O SisVANT é formado pelo veículo aéreo, os componentes necessários à decolagem, voo e pouso do veículo, os meios utilizados na realização da missão, a estação de pilotagem remota, os meios para comunicações e controle, a carga útil, entre outros. Este trabalho se encaixa na categoria de SisVANT, em que será projetada

a arquitetura que permite a realização de voos autônomos, no entanto, a fim de obter uma maior simplicidade, a notação de VANT será utilizada nesta tese.

O contexto em que se insere os VANTs, analisado nesta tese, pode ser melhor compreendido a partir do panorama geral de veículos não tripulados, apresentado na classificação da Figura 1. Um Veículo Não Tripulado (VNT) é um veículo que pode andar, voar, navegar pelo mar ou pelo espaço sem piloto a bordo. Diante de cada uma dessas capacidades, podemos subdividir ainda mais esses veículos. Um VNT capaz de andar sobre a superfície da terra é um Veículo Terrestre Não Tripulado (VTNT). Os VTNTs podem ser subdivididos em autônomos e operados remotamente. Um veículo capaz de navegar sobre ou abaixo da superfície da água é um Veículo Marinho Não Tripulado (VMNT). Os VMNT podem ser subdivididos conforme operam sobre ou abaixo da água e de acordo com sua autonomia. Um VNT que pode navegar pelo espaço sideral é uma Espaçonave Não Tripulada. Um VNT que pode voar é um Veículo Aéreo Não Tripulado (VANT), pode ser subdividido em duas categorias principais: Aeronave Remotamente Pilotada (ARP) e Veículo Aéreo Autônomo Não Tripulado (VAANT). Tendo esse panorama em mente, esta tese delimita como objeto de pesquisa os VANTs com o objetivo de aumentar seu nível de autonomia, aproximando-se, assim, dos VAANTs. Conforme irá se observar, a arquitetura projetada nesta tese não é restrita ao ambiente de VANTs, podendo, dessa forma, ser adaptada aos contextos de VTNTs e VMNTs.

Figura 1 – Classificação dos veículos não tripulados.

Fonte: Adaptada de IndiaMart (2017).

A classificação apresentada agrupa os VNTs conforme o local em que operam: terra, ar, água e espaço. Ela faz o agrupamento também, de acordo com a sua autonomia: operados remotamente e autônomos. Muitas outras formas de classificação poderiam ser feitas, como por exemplo, subdivisão dos VNTs em autônomos, semi-autônomos e rádio controlados. Pensando nos VANTs, eles poderiam ser organizados baseados em: massa, dimensão, altitude de voo, distância de voo, tempo de voo, tipo de aplicação, tipo de sustentação, tipo de decolagem/pouso, como abordado em Weatherington e Deputy (2005), Gambold (2011), Carvalho, Bueno e Modesto (2014). A Figura 2 mostra uma classificação adaptada de Watts, Ambrosia e Hinkley

(2012), Heffner e Foucher (2017), que agrupa os VANTs conforme a massa, altitude de voo, distância de voo e tempo de voo. Nessa figura, a sigla HALE representa *High Altitude, Long Endurance*, a sigla MALE significa *Medium Altitude, Long Endurance*, LALE quer dizer *Low Altitude, Long Endurance*, LASE remete a *Low Altitude, Short Endurance*, SUAS representa *Small Unmanned Aircraft System* e MAV indica *Micro Aerial Vehicle*. Como exemplos de veículos nessas categorias temos: o RQ-4 Global Hawk que custa cerca de US$131,00 milhões, voa a altitude de 18.000 metros por mais de 34 horas, pesa 14.000 kg e atinge a distância de aproximadamente 22.000 km, sendo assim, classificado como HALE [1]; o MQ-1 Predator custa por volta de US$20,00 milhões, voa a altitude de 7.600 metros, capaz de voar mais de 24 horas, pesa 1.020 kg e alcança a distância de aproximadamente 1.100 km, sendo assim, um exemplo de MALE [2]; o ScanEagle custa cerca de US$3,20 milhões, voa a altitude de 4.572 metros por mais de 24 horas, pesa 18 kg e atinge a distância de aproximadamente 100 km, sendo assim, um LALE [3]; o RQ-20 Puma custa por volta de US$250,00 mil, voa a altitude de 152 metros por até 3 horas, pesa 6,3 kg e alcança a distância de aproximadamente 20 km classificado, portanto, como LASE [4]. Todas essas aeronaves citadas são utilizadas, principalmente, em aplicações militares. As Figuras 3a, 3b, 3c e 3d apresentam os VANTs mencionados.

A proposta apresentada nesta tese abrange as categorias de MAV e SUAS, no entanto pode ser expandida também as demais classificações. Os veículos dessas categorias, em geral, são mais baratos e são empregados em aplicações civis, os demais veículos são mais caros e costumam ser utilizados em aplicações militares. As Figuras 3e e 3f apresentam os VANTs Arara e Ararinha, pertencentes as categorias SUAS e MAV, respectivamente.

Figura 2 – Classificação dos VANTs conforme: massa, altitude, distância e tempo de voo.

Fonte: Adaptada de Watts, Ambrosia e Hinkley (2012), Heffner e Foucher (2017).

[1] <http://www.af.mil/About-Us/Fact-Sheets/Display/Article/104516/rq-4-global-hawk/>
[2] <http://www.af.mil/About-Us/Fact-Sheets/Display/Article/104469/mq-1b-predator/>
[3] <http://www.boeing.com/history/products/scaneagle-unmanned-aerial-vehicle.page>
[4] <https://www.avinc.com/uas/view/puma>

Figura 3 – Exemplos de VANTs de acordo com sua categoria.

(a) RQ-4 Global Hawk (HALE).

(b) MQ-1 Predator (MALE).

(c) ScanEagle (LALE).

(d) RQ-20 Puma (LASE).

(e) Arara (SUAS).

(f) Ararinha (MAV).

Fonte: Grumman (2019), AirforceTechnology (2019), Boeing (2019), AeroVironment (2019), ArmamentoDefesa (2019), GISA (2019).

Os VANTs vêm sendo utilizados em diversas áreas de aplicação como: agricultura de precisão (XUE *et al.*, 2016; ALSALAM *et al.*, 2017); monitoramento/inspeção (DOSHI *et al.*, 2015; HALLERMANN; MORGENTHAL; RODEHORST, 2015); busca e resgate (BROWN; ESTABROOK; FRANKLIN, 2011); mapeamento/levantamento (NEX; REMONDINO, 2014); sensoriamento remoto (ROYO *et al.*, 2017); controle de trajetória/auxílio a navegação (PRODAN *et al.*, 2013; RAMASAMY *et al.*, 2016); combate a incêndios (SUJIT; KINGSTON; BEARD, 2007); transporte de pacotes (AMAZON, 2018b); pesquisa acadêmica (KIM *et al.*, 2013); segurança pública; ecologia; cinematografia; atividades militares; entre outras.

Sobre a aplicação/missão executada pela aeronave dois conceitos importantes podem ser estabelecidos:

- **Aplicações Ativas**: são missões que necessitam que a aeronave interfira no meio ambiente coletando ou trocando algum tipo de material. Como exemplos dessas aplicações, podemos citar: pulverização sobre cultivo; combate a incêndios; busca e resgate; e lançamento de projéteis.

- **Aplicações Passivas**: são missões em que a aeronave não interfere no meio ambiente de forma direta, ou seja, coletando ou trocando algum tipo de material. As interferências são apenas indiretas como som e liberação de gases combustíveis. Como exemplos dessas aplicações podemos citar: filmagem/cinematografia; mapeamento/levantamento; sensoriamento remoto; e monitoramento/inspeção.

Os autores Pastor, Lopez e Royo (2007), Austin (2010) definiram algumas situações em

que os VANTs apresentam vantagens sobre as aeronaves tripuladas para a realização de missões denominadas "D-cube", ou ainda, *Dull, Dirty and Dangerous* (DDD). Abaixo, estão definidas cada uma das situações:

- **D - *Dull* [missões entediantes]**: em geral relacionadas com vigilância, de longa duração e que podem levar a tripulação da aeronave à fadiga e à perda de concentração, contribuindo para uma menor eficácia da missão;

- **D - *Dirty* [missões sujas]**: aquelas em que existem a possibilidade de contaminação, como por exemplo, o sobrevoo de uma usina nuclear com vazamento radioativo ou áreas afetadas por contaminação biológica. A pulverização de culturas agrícolas com produtos químicos tóxicos é uma área de aplicação em que VANTs têm exercido constantes contribuições;

- **D - *Dangerous* [missões perigosas]**: aquelas em que a tripulação é constantemente submetida à situações que colocam em risco vidas humanas, geralmente presente em aplicações militares.

Nesta pesquisa, foram utilizados três VANTs, dois de asa rotativa (iDroneAlpha e iDroneBeta) e outro de asa fixa (Rascal 110). A Figura 4 mostra quatro VANTs. O iDroneAlpha e iDroneBeta são quadricópteros utilizados nos experimentos de voos reais durante o doutorado. O Rascal 110 é um VANT de asa fixa utilizado durante os experimentos de simulação. O Ararinha é uma iniciativa aberta, que teve origem em um projeto no SSC/ICMC/USP e possui informações de montagem disponibilizadas no site do Grupo de Interesse em Sistemas Autônomos e Aplicações (GISA) em GISA (2017). A aeronave Rascal 110 foi utilizada nesta pesquisa dentro do simulador de voo, pois não existe uma modelagem do Ararinha no simulador de voo utilizado. As principais especificações técnicas das aeronaves iDroneAlpha, iDroneBeta, Rascal e Ararinha encontram-se na Tabela 1.

Figura 4 – Veículos aéreos não tripulados utilizados.

(a) iDroneAlpha. (b) iDroneBeta. (c) Rascal 110. (d) Ararinha.

Fonte: SIGPlanes (2017), MTBSAE (2017).

[5] N/D representa um dado Não Disponível

Tabela 1 – Especificações técnicas das aeronaves iDroneAlpha, iDroneBeta, Rascal 110 e Ararinha.

Característica/Aeronave	iDroneAlpha	iDroneBeta	Rascal 110	Ararinha
Frame	F450	S500	N/D[5]	N/D
Envergadura/Largura	0,57 m	0,60 m	2,79 m	1,90 m
Comprimento	0,55 m	0,58 m	1,92 m	1,15 m
Altura	0,32 m	0,32 m	0,61 m	0,42 m
Potência elétrica	N/D	N/D	1600 W	740 W
Peso	1,06 kg	1,14 kg	4,99 kg	2,83 kg
Payload	0,40 kg	0,40 kg	0,91 kg	0,60 kg
Tipo de Asa	Asa Rotativa	Asa Rotativa	Asa Fixa	Asa Fixa
Tempo de voo	7 minutos	7 minutos	23 minutos	15 minutos

Fonte: SIGPlanes (2017), MTBSAE (2017).

2.3 Componentes Básicos de um VANT

Os VANTs, em geral, são compostos por um conjunto de partes como: moldura ou *frame*, motor(es), hélice(s), controlador(es) do(s) motor(es), servo(s) motor(es) e bateria. O *frame* nada mais é do que o chassi da aeronave em que se encontraram os demais componentes. O motor, em geral, *brushless* de corrente contínua é o responsável pela propulsão necessária ao VANT. A hélice é a responsável pela sustentação da aeronave no ar. O controlador de motor, também chamado de *Electronic Speed Controller* (ESC), é o responsável por controlar motores do tipo *brushless*. Os servos motores, em geral, são utilizados em aeronaves de asa fixa para fazer o controle do *aileron*, leme e profundor. A bateria é a responsável por alimentar todo o sistema elétrico a bordo da aeronave.

Figura 5 – Ângulos de rotação e eixos principais do VANT.

(a) Aeronave de Asa Fixa. (b) Aeronave de Asa Rotativa.

Fonte: Adaptada de Novatel (2019), Reséndiz e Rivas-Araiza (2016).

Alguns outros componentes como piloto automático, computador de bordo, módulo GPS, bússola, módulo de telemetria, receptor do controle de rádio e *power module* também são importantes na composição do VANT, entretanto esses componentes serão melhor discutidos nas Seções 2.6, 2.7 e 2.8.

Além dos componentes destacados, é importante conhecer os principais eixos e tipos de movimentos de rotação do VANT. Podemos subdividir a aeronave em três eixos principais: Longitudinal, Lateral e Vertical. Sobre cada um desses eixos, a aeronave pode executar um movimento de rotação: *roll*, *pitch* e *yaw*. A Figura 5 mostra os ângulos de rotação do VANT, assim como seus eixos principais.

2.4 Estações de Controle de Solo

Uma Estação de Controle de Solo (GCS, do inglês *Ground Control Station*) é uma aplicação que executa sobre um computador, *tablet* ou celular localizado em solo, e se comunica com o VANT através de um sistema de telemetria *wireless* (ARDUPILOT, 2017b). Esse software possui uma interface em que se pode planejar, editar, salvar, carregar e acompanhar a missão do VANT em tempo real. A GCS pode também ser utilizada para carregar o *firmware* do piloto automático na controladora de voo, visualizar os *logs*, controlar o voo, atualizar os comandos e as configurações de parâmetros da aeronave. A maioria das estações inclui informações sobre o *status* do VANT e permite enviar comandos para a aeronave (PEREZ *et al.*, 2013). As estações Mission Planner, APM Planner 2, QGroundControl (QGC), MAVProxy e LibrePilot foram avaliadas neste trabalho e suas telas estão ilustradas na Figura 6.

Essas GCS foram analisadas e as principais características levantadas são listadas na Tabela 2. Todos os códigos-fonte são abertos e estão disponíveis para *download* no GitHub sob a Licença Pública Geral (GPL, do inglês *General Public License*). Todas as estações são multiplataforma, com exceção da Mission Planner, que é exclusiva para ambiente Windows (Win.). Todas as estações, exceto o MAVProxy, operam sobre avião (**P** - *Plane*), multirotor (**C** - *Copter*) e veículo terrestre (**R** - *Rover*). Dentre todas essas ferramentas, a Mission Planner é a que possui mais funcionalidades e opções disponíveis, além de ser a estação mais antiga. Outras características presentes em todas as estações estudadas são o fato de serem compatíveis com o protocolo *Micro Air Vehicle Link* (MAVLink) (MAVLINK, 2019), possuírem uma vasta comunidade de desenvolvedores ativos (contribuidores) e muitos *forks* dos projetos no GitHub indicando o potencial dessas GCSs.

A estação QGroundControl possui a vantagem de dar suporte a simulações *Software-In-The-Loop* (SITL) e *Hardware-In-The-Loop* (HITL). O Mission Planner descontinuou as simulações HITL, dando suporte apenas a SITL. A estação QGroundControl possui a desvantagem de dar suporte completo somente a pilotos automáticos Pixhawk. O Mission Planner dá suporte a pilotos automáticos Pixhawk e *ArduPilot Mega* (APM). Tanto o Mission Planner quanto o QGroundControl dão suporte aos dois principais *firmwares* existentes de piloto automático PX4 e Ardupilot.

Figura 6 – Estações de controle de solo avaliadas.

(a) Mission Planner.

(b) APM Planner 2.0.

(c) QGroundControl.

(d) MAVProxy.

(e) LibrePilot.

Fonte: Elaborada pelo autor.

Este trabalho utiliza o Mission Planner, principalmente, para fazer a atualização do *firmware* antes de executar os experimentos reais, devido a sua alta confiabilidade e quantidade de recursos. Nos experimentos SITL e HITL, serão utilizadas as estações QGroundControl e MAVProxy. Essa escolha se deu pelo fato de ambas possuírem as funcionalidades SITL e HITL

[6] <http://ardupilot.org/planner>
[7] <https://github.com/ArduPilot/MissionPlanner>
[8] <http://ardupilot.org/planner2>
[9] <https://github.com/ArduPilot/apm_planner>
[10] <http://qgroundcontrol.com>
[11] <https://github.com/mavlink/qgroundcontrol>
[12] <http://ardupilot.github.io/MAVProxy>
[13] <https://github.com/ArduPilot/MAVProxy>
[14] <https://www.librepilot.org>
[15] <https://github.com/librepilot/LibrePilot>

Tabela 2 – Comparações entre as estações de controle de solo avaliadas.

Característica/GCS	Mission Planner	APM Planner 2	QGroundControl	MAVProxy	LibrePilot
Ano de Lançamento	2010	2013	2013	2012	2015
Ano da Última Versão	2018	2018	2018	2019	2016
Versão Avaliada	1.3.44	2.0.24	3.2.4	1.6.1	16.09
Preço	Gratuito	Gratuito	Gratuito	Gratuito	Gratuito
Código-Fonte	Aberto	Aberto	Aberto	Aberto	Aberto
Licença	GPL	GPL	GPL	GPL	GPL
Linguagem	C#	C++ e C	C++ e C	Python	C e C++
Plataforma	Win.	Win., Lin., Mac.	Win., Lin., Mac. Android, iOS	Win., Lin., Mac.	Win., Lin., Mac.
Veículos Suportados	P - C - R	P - C - R	P - C - R	P	P - C - R
Firmwares Suportados	PX4 e Ardupilot	PX4 e Ardupilot	PX4 e Ardupilot	Ardupilot	N/D
Hardwares Suportados	Pixhawk e APM	Pixhawk e APM	Pixhawk	APM	CopterControl, Revolution, Revo Nano, CC3D, Atom, OPLink Mini e Sparky2
Contribuidores GitHub	62	83	150	50	70
Forks no GitHub	1378	376	1417	375	148
Site [Código]	Site[6] [GitHub[7]]	Site[8] [GitHub[9]]	Site[10] [GitHub[11]]	Site[12] [GitHub[13]]	Site[14] [GitHub[15]]

Fonte: Elaborada pelo autor.

bem implementadas e serem fáceis de utilizar. Além disso, elas foram projetadas para o ambiente Linux, plataforma utilizada pela maioria das ferramentas desta tese.

2.5 Simuladores de Voo

Uma simulação pode ser pensada como uma imitação de um processo do mundo real ao longo do tempo (BANKS, 2000). Um simulador de voo é um software que tenta recriar a realidade existente no voo de uma aeronave. Em geral, os simuladores incluem as seguintes características do ambiente: turbulência, mudança do clima e transição do tempo. Eles incluem também características do cenário, tendo boa parte da superfície terrestre mapeada, e da aeronave, apresentando uma grande quantidade de veículos modelados. Três simuladores de voo principais foram avaliados: FlightGear (FG), X-Plane e MicroSoft Flight Simulator (MSFS). Dois simuladores de robótica foram avaliados: MORSE e Gazebo.

O FlightGear é um simulador de voo, codificado principalmente em C++, embora algumas partes sejam desenvolvidas em C. Caracteriza-se por ser multiplataforma, código aberto, lançado com a licença GPL e utilizado em pesquisa acadêmica, educação e entretenimento (FLIGHTGEAR, 2017a; FLIGHTGEAR, 2018). Ele se caracteriza também por possuir uma grande variedade de aeronaves, aeroportos e cenários disponíveis para *download*. Seu desenvolvimento teve como foco principal o realismo e a possibilidade de uso em pesquisas, não sendo voltado para jogos. No entanto, o uso dessa ferramenta exige um conhecimento mínimo de pilotagem, uma vez que, manipular seus controles pode gerar uma dificuldade inicial. O motor de simulação SimGear é utilizado pelo FG, sendo um software para simulações independente do FG (FLIGHTGEAR, 2017b). A Tabela 3 apresenta os principais tipos de veículos presentes

no FG, num total de 391 aeronaves apenas duas são controladas por rádio. O Rascal 110 é uma dessas duas aeronaves descritos nessa tabela. Outra opção é o Cessna 172p que é uma aeronave de asa fixa e se destaca devido ao seu modelo de dinâmica ser bastante realista, todavia não é um VANT, mas sim uma aeronave tripulada. A Figura 7a apresenta uma tela do simulador FG.

Tabela 3 – Características dos modelos de aeronaves disponíveis no FightGear.

Descrição	Quantidade Disponíveis	% Modelos
Aeronave de Asa Fixa	327	83,6%
Aeronave de Asa Rotativa	32	8,1%
Planador	11	2,8%
Veículo Terrestre	8	2,0%
Ficção/Fantasia/Diversão	6	1,5%
Dirigível/Balão	5	1,2%
Controlado Remotamente	2	0,5%
Total	391	100,0%

Fonte: Adaptada de FlightGear (2017c).

Figura 7 – Simuladores de voo avaliados.

(a) FlightGear. (b) X-Plane. (c) Microsoft Flight Simulator.

Fonte: X-Plane (2017b), Microsoft (2017).

X-Plane, entre todos os simuladores de voo, é o único simulador, credenciado pela Administração Federal de Aviação (FAA, do inglês *Federal Aviation Administration*) para treinamento de pilotos (X-PLANE, 2017a). Esse simulador possui uma alta precisão no modelo de voo, uma grande base de dados de aeronaves e cenários para navegação (X-PLANE, 2017b). A Figura 7 (b) mostra a tela de interface do simulador X-Plane.

Microsoft Flight Simulator (MSFS) é um jogo de computador e também um simulador de voo (MICROSOFT, 2017). Na Figura 7 (c), é apresentada uma tela do simulador de voo MSFS. No ano de 2010 o nome desse simulador mudou para Microsoft Flight. Esse simulador permaneceu em desenvolvimento até 2012, quando foi definitivamente cancelado (WIKIPéDIA, 2017).

Os simuladores MORSE e Gazebo também foram brevemente avaliados. Eles foram projetados para robótica em geral, não sendo portanto, específicos para aeronaves. A Tabela 4 traz um resumo de algumas características dos simuladores estudados. Entre essas características se destacam: o ano de lançamento, o ano da última versão, a plataforma em que operam, o tipo de licença, a linguagem de programação utilizada, o preço, entre outras.

Tabela 4 – Comparações entre os simuladores avaliados.

Característica/Simulador	FG	X-Plane	MSFS	MORSE	Gazebo
Ano de Lançamento	1997	1993	1982	2011	2011
Ano da Última Versão	2018	2018	2012	2016	2018
Plataforma	Win., Lin., Mac., FreeBSD, Solaris	Win., Lin., Mac.	Win.	Lin., FreeBSD, Mac.	Win., Lin., Mac.
Licença	GPL	Comercial	Comercial	BSD	Apache 2.0
Código-Fonte	Aberto	Fechado	Fechado	Aberto	Aberto
Linguagem	C++ e C	C e C++	C, C++ e C#	C e Python	C++
Preço	Gratuito	US$60,00[16]	US$16,00[17]	Gratuito	Gratuito
Certificado pela FAA	Não	Sim	Não	Não	Não
Número de Aeronaves	391	40	24	3	6
Número de Aeroportos	+20.000	+18.000	+24.000	N/D	N/D
Tipo de Simulador	Sim. de voo	Sim. de voo	Sim. de voo	Sim. de robótica	Sim. de robótica

Fonte: Elaborada pelo autor.

Nesta tese foi utilizado, principalmente, o simulador FlightGear na etapa de experimentos SITL, pois é um simulador realístico, leve e simples de utilizar. As simulações conduzidas no FG utilizaram o Rascal 110, que é a aeronave com características mais próximas do Ararinha. Diversos modelos de planejadores e replanejadores de rotas, utilizados neste trabalho, foram modelados baseando-se no Ararinha. O X-Plane foi utilizado em alguns experimentos HITL iniciais, posteriormente, foi abandonado. Os experimentos realizados no X-Plane utilizaram o VANT HILStar17f (PIXHAWK, 2017a).

2.6 Pilotos Automáticos

Um Piloto Automático (AP, do inglês *AutoPilot*) é um controlador de voo capaz de seguir *waypoints* especificados pelo usuário, prover estabilização autônoma e seguir comandos de telemetria (ARDUPILOT, 2017c). De maneira geral, os APs contêm um conjunto de sensores internos como: acelerômetro (acel.), giroscópio (giro.), bússola ou magnetômetro (mag.) e barômetro (baro.). Existem no mercado diversos APs disponíveis. Neste trabalho, seis APs, ilustrados na Figura 8, foram avaliados. Os APs são: Ardupilot APM, Pixhawk versão 1, Naza M versão 2, Intel Aero, Pixhawk versão 2 e Erle-Brain 3. O Ardupilot APM é um projeto de hardware aberto que utiliza um processador AVR 2560 8 bits e suporta até 166 *waypoints*. O controlador de voo Pixhawk v1 é um projeto de hardware aberto que possui um ARM Cortex de 32 bits e suporta até 718 *waypoints*. O controlador de voo Naza M v2 é um projeto de hardware fechado fabricado pela empresa DJI. O Intel Aero é uma placa de computação embarcada que executa o sistema operacional Yocto-Linux e roda nativamente o ArduPilot. O Pixhawk v2 possui um ARM Cortex de 32 bits, além de incluir o suporte a vários sistemas de redundância como GPS, alimentação, acelerômetro, giroscópio, bússola e barômetro. Esse controlador pode incluir também uma placa Intel Edison integrada. O Erle-Brain 3 possui um ARM v8 de 64 bits em seu AP (PXFmini) integrado com um computador embarcado (Raspberry Pi). Um levantamento mais

[16] Preço obtido através do site http://www.x-plane.com/desktop/buy-it/
[17] Preço obtido através do site https://www.amazon.com

completo sobre diferentes plataformas de controlador de voo foi desenvolvido em Ebeid, Skriver e Jin (2017). Não existe, na literatura, uma diferença clara entre os termos piloto automático e controlador de voo, conforme ressaltado em Ebeid, Skriver e Jin (2017), DroneTrest (2015), dessa maneira, não será feita distinção entre tais termos nesta tese.

Figura 8 – Pilotos automáticos avaliados.

(a) APM v2.8. (b) Pixhawk v1. (c) Naza M v2. (d) Intel Aero.

(e) Pixhawk v2. (f) Erle-Brain v3.

Fonte: Ardupilot (2017a), Pixhawk (2017b), DJI (2017), Intel (2017a), Pixhawk (2017b), ErleRobotics (2017).

A Tabela 5 sintetiza algumas características dos APs avaliados como o tipo de projeto de hardware, a empresa que o desenvolve, o processador embarcado, os tipos de veículos suportados, o preço médio, o Sistema Operacional (SO), as dimensões, o peso, os sensores e as interfaces disponíveis. Com relação aos tipos de veículos suportados as seguintes notações são utilizadas: operam sobre avião (**P** - *Plane*), multirotor (**C** - *Copter*), veículo terrestre (**R** - *Rover*), antena *tracker* (**A** - *Antena*). O NuttX é um Sistema Operacional de Tempo Real (RTOS, do inglês *Real-Time Operating System*) presente na Pixhawk v1, v2 e Intel Aero. Os APs utilizados se comunicam através do protocolo MAVLink, desse modo, todos os envios de comandos da aplicação MOSA e IFA ou mesmo da GCS e do rádio utilizam esse protocolo. Nesta tese os APs Ardupilot APM e Pixhawk v1 são utilizados por serem mais baratos, facilidade de uso, por possuírem o hardware e código aberto e se adequarem bem ao presente projeto. Vale lembrar que as versões dos *firmwares* utilizadas na APM e Pixhawk são diferentes, a versão da APM é o Ardupilot 3.2.1, já a versão utilizada na Pixhawk é o Ardupilot 3.5.7 (ArduCopter). A APM não suporta versões acima da 3.2.1, pois não tem memória e recursos de hardware suficientes.

Os APs Intel Aero, Pixhawk v2 e Erle-Brain 3 além de fazerem o papel de controlador de voo também funcionam como um computador de bordo. Existem outras placas no mercado que atuam de forma semelhante, como por exemplo, o NAVIO+ que integra um AP e uma Raspberry Pi 2, o NAVIO2 que integra um AP e uma Raspberry Pi 3, o Erle-Brain 1 que integra um AP

[18] N/A representa um dado que Não se Aplica
[19] Preços obtidos através dos sites www.amazon.com, erlerobotics.com, software.intel.com

Tabela 5 – Comparações entre os pilotos automáticos avaliados.

Característica/AP	APM	Pixhawk v1	Naza M v2	Intel Aero	Pixhawk v2	Erle-Brain v3
Hardware	Aberto	Aberto	Fechado	Fechado	Aberto	Aberto
Empresa	DIY Drone	3DR	DJI	Intel	3DR	Erle Robotics
Processador	Atmel AVR ATmega-2560	ARM Cortex-M4	N/D	Intel Atom x7-Z8750	ARM Cortex-M4	ARM Cortex-A53
Processador	8 bits	32 bits	N/D	64 bits	32 bits	64 bits
Tipos Veículos	P - C - R - A	P - C - R	C	C	P - C - R	P - C - R
SO	N/A[18]	NuttX	N/D	NuttX e Yocto	NuttX e Yocto	Debian
Dimensão(mm)	70x45x13	81x50x15	45x32x18	88x63x20	94x43x31	95x70x24
Peso	31 g	38 g	27 g	60 g	75 g	100 g
Preço Médio[19]	US$27,00	US$65,00	US$100,00	US$399,00	US$198,00	US$234,00
Sensores	Giro., Acel., Mag., Baro.	Giro., Acel., Mag., Baro.	Giro., Acel., Baro.	Giro., Acel., Mag., Baro.	Giro., Acel., Mag., Baro.	Giro., Acel., Mag., Baro., Gravidade
Interfaces Disponíveis	3xUART, 1xI2C, 1xSPI, 1xμ-USB, 1xTelem, 1xADC, 30xPWM, 1xGPS	5xUART, 2xCAN, 1xI2C, 1xSPI, 1xμ-USB, 2xTelem, 1xADC, 14xPWM, 1xGPS, 1xμ-SD	N/D	3xUART, 1xCAN, 1xI2C, 1xμ-USB, 1xμ-SD, 1xμ-HDMI	5xUART, 2xCAN, 2xI2C, 1xSPI, 1xμ-USB, 2xTelem, 1xADC, 14xPWM, 2xGPS, 1xμ-SD	1xUART, 2xI2C, 4xUSB, 1xμ-USB, 1xADC, 12xPWM, 1xμ-SD, 1xEthernet, 1xHDMI

Fonte: Elaborada pelo autor.

e uma BeagleBone Black (BBB), o Erle-Brain 2 que integra um AP e uma Raspberry Pi 2, o PixHawk Fire Cape (PXF) é um AP que integra uma BBB.

O Quadro 1 apresenta os principais modos de voo suportados pelos pilotos automáticos APM e Pixhawk. Durante este trabalho, os seguintes modos foram utilizados durante os experimentos: `Guided`, `Auto`, `Loiter`, `AltHold`, `Stabilize`, `RTL`, `Land`, `AutoTune` e `Circle`. Durante os experimentos com voo autônomo, os modos de voo utilizados foram: `Guided` e `Auto`. O `Guided` é utilizado apenas na etapa de decolagem, já o `Auto` é utilizado durante a realização da missão.

A próxima seção descreve alguns computadores embarcados que se integram ao piloto automático provendo à aeronave a autonomia necessária para o voo.

2.7 Companion Computers

O termo *Companion Computer* (CC) pode ser definido como um computador a bordo da aeronave, que auxilia nas tomadas de decisão durante o voo. O CC, em geral, é um computador de placa única que possui em um único circuito impresso: processador, memória e interfaces de entrada e saída de dados. O computador de bordo se comunica e controla o piloto automático, utilizando os dados obtidos através do protocolo MAVLink para fazer uma tomada de decisão inteligente durante o voo. Como exemplo de processamento a ser realizado a bordo temos:

Quadro 1 – Modos de voo suportados pelos pilotos automáticos APM e Pixhawk.

Modo de voo	Tipo de VANT	Precisa de GPS	Descrição do modo de voo
`Guided`	*Copter* e *Plane*	Sim	Segue os *waypoints* definidos pela GCS
`Auto`	*Copter* e *Plane*	Sim	Executa uma missão pré-definida automaticamente
`Loiter`	*Copter* e *Plane*	Sim	Mantém o controle de altitude e posição
`Stabilize`	*Copter* e *Plane*	Não	Mantém auto-nivelamento dos eixos de *roll* e *pitch*
`AltHold`	*Copter*	Não	Mantém o controle de altitude e auto-nivelamento do *roll* e *pitch*
`RTL`	*Copter* e *Plane*	Sim	Retorna acima do local de decolagem, pode incluir o pouso
`Land`	*Copter* e *Plane*	Não	Reduz a altitude até atingir o nível do solo
`AutoTune`	*Copter* e *Plane*	Sim	Procedimento automatizado de calibração de *roll*, *pitch* e *yaw* para melhorar os *loops* de controle
`Circle`	*Copter* e *Plane*	Sim	Automaticamente circunda um ponto a frente do VANT
`Acro`	*Copter* e *Plane*	Não	Mantém controle de atitude, sem auto-nivelamento
`Drift`	*Copter*	Sim	Como o `Stabilize`, mas coordena *roll* e *yaw* como um *plane*
`Flip`	*Copter*	Não	Executa uma manobra de *flip* automaticamente
`PosHold`	*Copter*	Sim	Como no `Loiter`, mas com *roll* e *pitch* manual, quando os *sticks* não são centrados
`Throw`	*Copter*	Sim	Mantém a posição após uma decolagem de arremesso
`SmartRTL`	*Copter*	Sim	Como no `RTL`, mas traça caminho para chegar no *home*
`FlowHold`	*Copter*	Não	Controle de posição utilizando fluxo óptico
`Follow`	*Copter*	Sim	Segue outro veículo
`Manual`	*Plane*	Não	Movimento das superfícies de controle manual
`FlyByWireA`	*Plane*	Não	Os controles de *roll* e *pitch* seguem a entrada do *stick*, até definir limites
`FlyByWireB`	*Plane*	Sim	Como no `FlyByWireA`, mas com controle automático de altura e velocidade
`Cruise`	*Plane*	Sim	Como no `FlyByWireB`, mas com rastreamento do terreno e acompanhamento do terreno
`Training`	*Plane*	Não	Controle manual até o limite do *roll* e *pitch*

Fonte: Adaptada de ArduPilot (2019b), ArduPilot (2019c).

planejar a missão, replanejar a trajetória, retirar fotografia, gravar vídeo, efetuar aspersão, processar imagens, abortar o voo, disparar paraquedas, entre outros.

Atualmente, soluções utilizando *companion computer* estão sendo desenvolvidas e utilizadas por diversos grupos de pesquisa em todo o mundo. As principais comunidades/empresas envolvidas em seu desenvolvimento são: ardupilot.org [20], DroneKit [21], 3D Robotics (3DR) [22], FlytBase [23], DIY Drones [24] e DJI[25]. As principais placas de processamento embarcado utilizadas

[20] <http://ardupilot.org>
[21] <http://dronekit.io>
[22] <https://3dr.com>
[23] <https://flytbase.com>
[24] <http://diydrones.com>
[25] <https://www.dji.com>

como CC são: Intel Edison, Raspberry Pi, BeagleBone, ODroid, NVidia Jetson TX1, Intel Aero, FlytPOD, FlytPOD PRO, entre outras. A Figura 9 mostra as placas Intel Edison, Raspberry Pi 3, BeagleBone Black e ODroid XU4.

Figura 9 – Computadores embarcados avaliados.

(a) Intel Edison. (b) Raspberry Pi 3. (c) BeagleBone Black. (d) Odroid XU4.

Fonte: Intel (2017b), Pi (2017), BeagleBoard (2017), Hardkernel (2017).

Tabela 6 – Comparações entre os *companion computers* avaliados.

Característica/CC	Intel Edison	Raspberry Pi 3	BeagleBone Black	Odroid XU4
Ano de Lançamento	2014	2016	2013	2015
Processador	Intel Atom Z3480	ARM Cortex-A53	ARM Cortex-A8	ARM Cortex-A15 e Cortex-A7
Processador	32 bits	64 bits	32 bits	N/D
Número de Núcleos	*Dual-Core*	*Quad-Core*	*Dual-Core*	*Octa-Core*
Frequência da CPU	500 MHz	1.2 GHz	1.0 GHz	2.0 GHz
Memória RAM	1 GB	1 GB	512 MB	2 GB
Armazenamento eMMC	4 GB	N/D	4 GB	N/D
GPIOs	30	40	92	42
SO *Default*	Yocto	Raspbian	Debian	Ubuntu
WiFi Nativo	Sim	Sim	Sim	Não
Bluetooth Nativo	Sim	Sim	Sim	Não
Dimensão (mm)	127x72x12	85x56x17	86x56x10	82x58x22
Peso	45 g	42 g	40 g	60 g
Preço Médio[26]	US$85,00	US$34,00	US$57,00	US$69,00
Interfaces Disponíveis	1xUSB, 2xμ-USB, 1xμ-SD	4xUSB, 1xμ-USB, 1xμ-SD, 1xEthernet, 1xHDMI	1xUSB, 1xμ-USB, 1xμ-SD	3xUSB, 1xμ-SD, 1xEthernet, 1xHDMI
Cartão μ-SD Equipado	Scan Disk Ultra 64 GB classe 10	Scan Disk Ultra 16 GB classe 10	Scan Disk 16 GB classe 4	N/A

Fonte: Elaborada pelo autor.

A Tabela 6 mostra as configurações dos CC analisados, em que podemos ver o ano de lançamento, o processador, o número de núcleos, a frequência, a quantidade de memória RAM, a quantidade de GPIOs, o SO, o preço médio, entre outras. É importante ressaltar que o SO descrito é o sistema *default* da placa, no entanto outros sistemas operacionais podem ser instalados, desde que sejam leves e compatíveis com a arquitetura. A maioria desses SOs embarcados são baseados em Linux. Nas placas Intel Edison, Raspberry Pi e BeagleBone Black, utilizaram-se os seguintes SOs Yocto, Raspbian e Debian, respectivamente. O *General Purpose Input Output* (GPIO) é utilizado para entrada e saída de dados dos CC, podendo ser conectados a sensores e a atuadores.

[26] Preços médios obtidos através do site https://www.amazon.com

Todas as placas analisadas possuem suporte a cartão de memória micro SD. A Intel Edison está equipada com um cartão 64 GB classe 10, a Raspberry Pi 3 está equipada com um cartão 16 GB classe 10, e por fim, a BeagleBone Black está com um cartão 16 GB classe 4.

Este trabalho utiliza os *companion computers* Intel Edison, Raspberry Pi e BeagleBone Black, como forma de processamento a bordo do VANT. Essa escolha se deu basicamente pelo poder de processamento, facilidade de uso, documentação disponível, baixo preço, além de serem placas utilizadas pela literatura.

2.8 Aviônicos

Um conjunto de sistemas aviônicos estão presentes no VANT. Por aviônicos, queremos dizer: toda a eletrônica a bordo do avião como, por exemplo, piloto automático, computador de bordo, receptor do controle de rádio, módulo de telemetria e Sistema de Posicionamento Global (GPS, do inglês *Global Positioning System*). Os aviônicos AP e CC já foram tratados nas Seções 2.6 e 2.7, portanto, não serão destacados aqui.

O receptor do controle de rádio, ilustrado na Figura 10a, é responsável por receber os sinais enviados pelo controle de rádio e enviar os comandos para o AP que, em seguida, executará a manobra do comando recebido. O controle de rádio é utilizado para controlar o movimento (*throttle*) e a orientação (*roll*, *pitch* e *yaw*) da aeronave utilizando um sinal *Pulse Width Modulation* (PWM) (EBEID; SKRIVER; JIN, 2017). O controle de rádio (transmissor) utilizado nesta tese é o FlySky FS-i6 com 6 canais operando a 2.4 GHz. Já o receptor de controle de rádio é o FlySky FS-iA6 operando também a 2.4 GHz.

Outro aviônico importante é o módulo de telemetria *air* ilustrado na Figura 10b. Esse componente é responsável por receber/enviar comandos da GCS para a aeronave e da aeronave para a GCS. Dessa forma, é necessário que um módulo de telemetria *ground* esteja conectado ao computador em solo que executa o software da GCS. O protocolo de comunicação serial MAVLink é utilizado na transmissão dos dados entre a estação de solo e o VANT. O módulo de telemetria da 3DR, operando a 433 MHz e com taxa de transmissão de dados no ar de até 250 kbps, será utilizado.

O aviônico GPS é um tipo Sistema de Navegação Global por Satélite (GNSS, do ingês *Global Navigation Satellite System*). Esse componente é responsável por dar as coordenadas geográficas da aeronave e a sua orientação (três eixos) através de uma bússola (magnetômetro) interna incluída. O dispositivo GPS está ilustrado na Figura 10c. O módulo GPS ublox NEO-6M e NEO-M8N integrado com bússola é utilizado. O NEO-6M utiliza apenas sinais do sistema GPS americano, já o NEO-M8N utiliza, além dos sinais de GPS americano, sinais do sistema GLONASS, Galileo e BeiDou (UBLOX, 2014; UBLOX, 2016).

O componente *power module* está ilustrado na Figura 10d. Esse componente é res-

ponsável por alimentar o AP, fazer medições de voltagem e de consumo de corrente elétrica da bateria (ARDUPILOT, 2018). Com esse componente é possível configurar algumas ações, como *Return To Launch* (RTL) utilizando o Mission Planner ou QGroundControl. Ações como essa, são executadas pela aeronave caso a bateria esteja abaixo de um determinado limiar. Este equipamento é utilizado para aumentar a segurança a bordo da aeronave.

Por fim, tem-se o componente alarme de baixa voltagem da bateria ilustrado na Figura 10e. Esse componente auxilia o piloto e pessoas em solo a saber se o nível da bateria está abaixo de determinado limiar, caso esteja, um alarme sonoro é disparado. Através desse alarme, as pessoas em solo tem um *feedback* direto, avisando-as que a aeronave pode cair a qualquer momento devido a falta de energia. Outro recurso desse equipamento é um *display* visual indicando a tensão de cada uma das células da bateria.

Figura 10 – Componentes aviônicos utilizados.

(a) Receptor do controle de rádio. (b) Módulo de telemetria *air*. (c) GPS com bússola.

(d) *Power module*. (e) Alarme de bateria.

Fonte: FlySky (2017), 3DR (2017, 2017), Ardupilot (2018), Amazon (2018a).

2.9 Sensores e Atuadores da Aeronave

A aeronave, como mencionada anteriormente, possui diversos sensores internos acoplados ao AP e GPS, como: acelerômetro, giroscópio, barômetro, bússola e GPS. A função de cada um desses sensores na aeronave será discutida nessa seção, exceto o GPS, que foi discutido na seção anterior.

O acelerômetro e giroscópio são sensores inerciais do AP. O acelerômetro mede forças de aceleração e o giroscópio mede forças de rotação. Ao combinar essas medidas, o controlador de voo (AP) é capaz de calcular a atitude atual do VANT. A atitude é orientação no espaço, ou seja, o valor das medidas dos ângulos de *pitch*, *yaw*, *roll* e *heading* (DRONETREST, 2018).

O barômetro é um sensor de pressão utilizado para medir a altitude da aeronave. Esse sensor pode detectar a mudança na pressão do ar, quando o veículo se move apenas alguns centímetros (DRONETREST, 2018). Este dispositivo sofre influências da atmosfera, como posição do sol, nuvens, ventos, temperatura, umidade, vibração, entre outras. Com esse sensor, podemos capturar a altitude relativa e absoluta. Apesar do GPS também informar a altitude da aeronave, o barômetro é um sensor com precisão bem melhor que o GPS (GRAHAM, 2011).

O sensor bússola ou magnetômetro mede a força magnética, assim como uma bússola. Este sensor é importante para os VANTs multi-rotor, porque os sensores do acelerômetro e do giroscópio não são suficientes para permitir que o controlador de voo saiba em que direção o visor está voltado. Em aeronaves de asa fixa, isso é fácil, pois só se pode voar em uma direção (DRONETREST, 2018). Dessa maneira, muitas vezes esse sensor é combinado ao acelerômetro e giroscópio para dar um melhor valor de *pitch*, *yaw*, *roll* e *heading*.

O *power module*, apresentado na seção anterior, pode ser pensado também como um sensor da aeronave, uma vez que, mede a voltagem e o consumo de corrente elétrica da bateria. Outros três sensores foram incorporados ao CC, são eles: câmera RGB, sensor ultrassônico e sensor de temperatura. A câmera no espaço de cores RGB é utilizada para registrar o voo, mas também pode fornecer informações úteis para projetar algoritmos auxiliares, que podem detectar situações adversas e auxiliar durante o pouso e decolagem autônomo. O sensor ultrassônico incorporado poderá ser utilizado durante o pouso e decolagem para detectar a proximidade com o solo com mais precisão do que os atuais sensores barômetro e GPS. O sensor de temperatura poderá auxiliar na detecção de temperaturas elevadas na aeronave, auxiliando em tomadas de decisão emergenciais. Esses sensores serão melhor discutidos ao longo da tese. Alguns outros sensores que podem ser incluídos são: tubo de Pitot, ADS-B, entre outros.

O piloto automático possui um conjunto de sistemas de atuação, como por exemplo, LEDs e *buzzers* que indicam o *status* do AP. Dois sistemas de atuação foram incorporados ao CC, são eles: sistema de *feedback* baseado em *buzzer* e sistema de *feedback* baseado em LED. Tais sistemas serão melhor discutidos ao longo da tese. Alguns outros atuadores que podem ser incluídos são: sistema de paraquedas, sistema pulverizador, entre outros.

2.10 DroneKit

Dronekit é uma biblioteca *open source*, iniciada em 2015, pela empresa 3D Robotics (3DR) e pela comunidade científica para fazer a comunicação entre o computador de bordo e o piloto automático (DRONEKIT, 2018). Essa biblioteca provê uma abstração do protocolo

MAVLink, que executa no CC, permitindo enviar comandos ao AP e, assim, controlar o VANT. Dronekit pode ser utilizado para incorporar inteligência ao VANT e processar planejamento de rota, visão computacional e modelagem 3D durante o voo (DRONEKIT, 2018). O Dronekit é compatível com todos os veículos que utilizam MAVLink. Esta biblioteca pode ser executada em Linux, Mac OS X e Windows. Ela é distribuída através da licença Apache versão 2.0 e dá suporte às seguintes plataformas Python, Android e Cloud.

Visando um ambiente de testes dos códigos desenvolvidos para controle do VANT, foi criado o ambiente Dronekit-SITL também pela 3DR. Esse ambiente permite testar todos os comandos executados sobre essa API. O ambiente utiliza o simulador de piloto automático ArduPilot SITL, que interpreta os comandos MAVLink e os executa em um ambiente virtual. O Dronekit-SITL suporta atualmente os seguintes veículos: *plane*, *copter* e *rover*.

Um conjunto de trabalhos acadêmicos têm utilizado o Dronekit em que podemos destacar os seguintes Boubeta-Puig *et al.* (2018), Chiaramonte (2018), Perez *et al.* (2018), Yu *et al.* (2018), Yuan, Zhan e Li (2017), Psirofonia *et al.* (2017), Penserini *et al.* (2017), Cavalcante, Bessa e Cordeiro (2017), Choi *et al.* (2016), Zhou *et al.* (2015).

O protocolo MAVLink foi iniciado em 2009 e foi projetado para comunicação com veículos não tripulados de pequeno porte (MAVLINK, 2019). Este protocolo é bastante leve, muito difundido entre os usuários de VANT e é distribuído sob licença LGPL. As mensagens da comunicação são definidas em arquivos *Extensible Markup Language* (XML). O Dronekit utiliza uma implementação desse protocolo chamada, pymavlink[27].

A presente tese utiliza o Dronekit 2.9.1 em Python, executado sobre ambiente Linux. Desse modo, pode-se fazer o controle da aeronave de forma fácil e modularizada com abstração dos comandos MAVLink. O Dronekit-SITL utilizado é a versão 3.2.0.

2.11 Simulação Software-In-The-Loop

Uma simulação *Software-In-The-Loop* (SITL) permite executar missões de VANTs sem nenhum hardware físico. Desse modo, todo o ambiente, hardware da aeronave, hardware do piloto automático e sistema de comunicação são emulados em software (ARDUPILOT, 2017e). A simulação do veículo e do ambiente é feita por um simulador de voo convencional, por exemplo, o FlightGear, enquanto a simulação do hardware do AP e do sistema de comunicação é feita através do ambiente ArduPilot-SITL.

O ArduPilot-SITL possui uma ampla quantidade de simuladores de veículos incorporados e pode se conectar a vários simuladores externos. Dessa maneira, o SITL pode simular: aeronaves multi-rotor, aeronaves de asa fixa e veículos terrestres. A Figura 11 mostra o funcionamento interno da simulação SITL. Cinco componentes principais podem ser visualizados: aplicação

[27] <https://pypi.org/project/pymavlink>

Figura 11 – Esquema da arquitetura de simulação SITL montada.

Fonte: Adaptada de CentMesh (2017).

SITL, simulador de física, simulador de voo (opcional), MAVProxy e estação de controle de solo. A aplicação SITL é o mesmo que o ArduPilot-SITL e pode executar os controladores de voo ArduPlane e ArduCopter. O simulador de física utilizado é o JSBSim, que simula o modelo de dinâmica de voo da aeronave. O simulador de voo utilizado é o FlightGear que é apenas um renderizador gráfico para acompanhamento do voo, sendo um elemento opcional. O MAVProxy é uma estação de controle de solo para execução das simulações SITL, além de ser um nó *hub* para outras estações de controle de solo. Por fim, tem-se que outras GCS mais robustas podem ser conectadas ao MAVProxy para acompanhar o voo.

Duas formas de simulações SITL principais foram realizadas neste trabalho: uma sobre o ambiente ArduPilot-SITL[28], descrita acima, e outra sobre o ambiente DroneKit-SITL[29], bastante semelhante a forma anterior, no entanto não há integração com o FlightGear. A versão do *firmware* do AP utilizada ao fazer os experimentos SITL é o Ardupilot 3.3.0.

2.12 Simulação Hardware-In-The-Loop

Uma simulação *Hardware-In-The-Loop* (HITL) substitui a aeronave e o ambiente por um simulador de voo. Esse simulador deve possuir um modelo de dinâmica de aeronave de alta fidelidade e modelo de ambiente (ARDUPILOT, 2017d). Os componentes piloto automático, telemetria, receptor do controle de rádio, controle de rádio são todos hardwares físicos. Tais elementos de hardware devem estar configurados da mesma forma que para execução de um voo

[28] <http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html>
[29] <http://python.dronekit.io/develop/sitl_setup.html>

real. Esse tipo de simulação, assim como SITL, auxilia no *debug* de algoritmos reduzindo o risco de uma queda do VANT em testes reais.

Figura 12 – Esquema da arquitetura de simulação HITL montada.

Fonte: Elaborada pelo autor.

A Figura 12 apresenta um esquema da arquitetura HITL desenvolvida neste doutorado. Tem-se um computador executando a estação QGroundControl com um módulo telemetria *ground*. Em outro computador tem-se o simulador de voo X-Plane conectado com o hardware do piloto automático. Integrado ao AP tem-se o módulo de telemetria *air* e o receptor do controle de rádio. Ainda nessa figura, percebemos dois modos de comunicação: um utilizando o controle de rádio a 2.4 GHz e um utilizando a telemetria a 433 MHz. Nessa figura, pode-se pensar que o primeiro computador simula todo o hardware da aeronave e o ambiente, entretanto todo o sistema aviônico dessa aeronave é composto por um hardware físico. O segundo computador funciona como uma GCS, em que monitora-se a missão executada.

Um modelo de simulação HITL estudado foi mostrado acima, em que o AP, o receptor de rádio, o rádio e a telemetria são hardwares físicos. Nesse modelo, não foi incorporado o CC como um hardware físico. Outro modelo de simulação HITL, montado nesta tese, considera, por exemplo, um CC (Raspberry Pi) e dois computadores conectados via Wi-Fi. A Figura 13 apresenta esse segundo modelo de arquitetura HITL. Pode-se pensar, nesse modelo, como uma simulação HITL junto com um SITL, uma vez que o AP é emulado em software. Mais detalhes sobre esse segundo modelo, envolvendo placas como Raspberry Pi, Intel Edison e BeagleBone Black, serão apresentados na Seção 7.5.

Figura 13 – Esquema da arquitetura de simulação HITL com CC.

Fonte: Elaborada pelo autor.

2.13 Arquiteturas de Controle de Robôs

Existem diversas arquiteturas de controle em sistemas robóticos encontradas na literatura, algumas que podemos destacar são: arquitetura reativa, arquitetura deliberativa, arquitetura híbrida. Abaixo, encontra-se uma descrição de cada uma dessas arquiteturas baseada em Jung *et al.* (2005).

- **Arquitetura Reativa**: possui como características principais: a percepção-ação; o modelo sensorial; e a reação imediata. Nessa arquitetura, o robô não tem um objetivo maior estabelecido (planejamento ou plano de ações), ele apenas percebe as informações do ambiente e reage a elas. Esta arquitetura segue o modelo de percepção e controle. Um sistema reativo é bastante útil para implementar comportamentos, como: desviar de obstáculos (reage a presença de um obstáculo) e seguir um objeto (acompanhar um elemento guia).

- **Arquitetura Deliberativa**: possui como características principais: o pré-planejamento; a sequência de ações; e a memória, mapa, planejamento. Nessa arquitetura, o robô tem apenas um plano a ser seguido, porém esse modelo não aceita imprevistos. Esta arquitetura segue o modelo de planejamento e controle. Um sistema deliberativo é bastante útil para implementar comportamentos, como: traçar uma rota (encontrar um caminho entre dois pontos) e executar um plano de ações (sequência de ações para apagar um incêndio).

- **Arquitetura Híbrida**: possui como características principais: planejamento; decisão; deliberação; e reação. Nessa arquitetura, o robô tem um plano global de ações a ser seguido e possui, também, a capacidade de reagir a imprevistos. Esta arquitetura segue o

modelo de percepção, planejamento e controle. Um sistema híbrido é bastante útil para implementar comportamentos, como: traçar uma rota reagindo a eventuais imprevistos no ambiente ou no veículo.

Figura 14 – Classificação das arquiteturas de controle em sistemas robóticos.

Fonte: Adaptada de Osório (2014).

A Figura 14 apresenta uma esquema da classificação dessas arquiteturas de controle. A arquitetura proposta nesta tese se encaixa como uma arquitetura híbrida ao apresentar um comportamento tanto deliberativo quanto reativo. O aspecto deliberativo é alcançado com o sistema MOSA efetuando o planejamento da missão. Desse modo, um objetivo global é estabelecido e esse planejamento busca atingi-lo. O aspecto reativo é obtido ao ficar monitorando informações via GCS e então decidir trocar/atualizar o plano da missão com o sistema MOSA adaptativo. O sistema IFA também entra no sistema como uma forma de reação, ao decidir atualizar a rota de voo, baseado em dados dos sensores internos, para uma rota emergencial, ou fazer um pouso vertical, ou ainda, um RTL.

Ao se pensar na execução de missões pré-planejadas, utilizando ferramentas como Mission Planner, apenas o aspecto deliberativo é obtido. Dessa maneira, o VANT apenas faz o controle dos motores de forma a seguir os *waypoints*, uma vez que a aeronave não é capaz de reagir e atualizar o plano da missão.

2.14 Localização, Mapeamento e Navegação

Na robótica móvel existem três grandes problemas que são localização, mapeamento e navegação (WOLF *et al.*, 2009). O primeiro problema busca descobrir, utilizando algum sistema de medida/métrica, qual a localização do robô. O segundo problema busca fazer o robô conhecer/mapear o cenário em que está inserido, identificando, por exemplo, onde estão os obstáculos. Por fim, o problema de navegação busca fazer o robô planejar uma rota e executá-la sem colidir com obstáculos.

No contexto abordado nesta tese, a aeronave realiza seus voos em espaço aberto (*outdoor*), consequentemente, o primeiro problema já está bem resolvido através do GPS. Esse equipamento

obtém a localização da aeronave com um erro de precisão, em geral, é inferior a 1,5 metros. Existem outros trabalhos como Teulière, Marchand e Eck (2015), Wang *et al.* (2013) que fazem a localização de VANTs em ambientes *indoor* (sem GPS), todavia esse tipo de cenário foge ao escopo desta tese.

O segundo problema é o mapeamento do ambiente, este trabalho o resolve através do carregamento do mapa no computador de bordo da aeronave. Esse mapa contém, previamente cadastradas, as regiões de interesse da missão conforme estabelecido pelo projetista. Essa estratégia funciona bem, pois o espaço em que a aeronave sobrevoa, em geral, é esparso, ou seja, possui poucos obstáculos. A Seção 7.4 detalha melhor como é feita a etapa de mapeamento do cenário em que serão realizados os voos.

Este trabalho preocupa-se em resolver uma parte do terceiro problema, que é a navegação. De acordo com Lin, Mu e Takase (2006), o processo de navegação é composto pelos seguintes componentes: mapeamento, localização, planejamento de rotas e desvio de obstáculos. Este trabalho busca construir um sistema que gerencie diversos aspectos da aeronave e que faça o planejamento de rotas com desvio de obstáculos de forma autônoma. A plataforma autônoma proposta lida também com outras questões, além da navegação, mapeamento e localização. Algumas das questões tratadas são: gerenciamento da missão (retirar fotografias, fazer filmagem da região, fazer pulverização) e gerenciamento de segurança (evitar danos a pessoas em solo e a aeronave).

2.15 Tolerância à Falhas

Uma falha é definida como uma mudança inesperada, que leva a um comportamento indesejável do sistema (MAGRABI; GIBBENS, 2000). Em geral, a literatura, como em Weber (2003), apresenta três conceitos importantes que são: falha, erro e defeito. A falha é definida como a causa física ou algorítmica do erro. Por sua vez, a falha pode levar o sistema ao estado de erro. Finalmente, um defeito é definido como um desvio da especificação e é obtido a partir do estado de erro. Falhas estão associadas ao universo físico, erros ao universo da informação e defeitos ao universo do usuário (WEBER, 2003).

A partir desses conceitos, pode-se pensar em construir um sistema que seja capaz de se recuperar de falhas, ou seja, um sistema em que, mesmo que ocorra uma falha, não permita que ela leve o comportamento do sistema a um estado de defeito. A área de Tolerância à Falhas lida exatamente com isso. Segundo Weber (2003), uma das principais maneiras de se implementar a tolerância à falhas é através de redundância, que pode ser obtida através de:

- Redundância de informação;
- Redundância de tempo;

- Redundância de hardware;
- Redundância de software.

A redundância de informação pode ser provida por códigos de correção de erros. Esta pode ser implementada através do aumento do número de bits, sendo portanto, uma forma de redundância de informação (WEBER, 2003). A redundância de tempo repete a computação no tempo. Dessa maneira, evita o custo de hardware adicional, aumentando o tempo necessário para realizar um processamento. É utilizada em sistemas em que o tempo não é crítico, ou em que o processador trabalha com ociosidade (WEBER, 2003).

A redundância de hardware aplicada ao contexto de VANTs pode ser obtida das seguintes formas: redundância de AP, de GPS, de IMU, de acelerômetro, de barômetro, de giroscópio, de bateria, de computador de bordo, entre outras. O autor Andersson (2013) apresenta um sistema de tripla redundância de APs, mais especificadamente o EasyPilot 3.0, aplicado a um VANT de asa fixa. O trabalho Kim e Sukkarieh (2003) apresenta um sistema com redundância de GPS, utilizando dois dispositivos em uma aeronave de asa fixa de pequeno porte. O trabalho de Osman *et al.* (2006) integra um sistema com redundância da Unidade de Medição Inercial (IMU, do inglês *Inertial Measurement Unit*) ao utilizar dois dispositivos IMU, trabalhando em conjunto.

Atualmente, já existem algumas soluções comerciais com redundância de AP, como o GEMINI[30]. Este sistema funciona com dois APs trabalhando em conjunto, cada piloto tem sua própria IMU, GPS e bússola. Caso o AP mestre apresente problemas, então o sistema troca automaticamente para o AP escravo. Esse sistema suporta no máximo 256 *waypoints* e apenas veículos multi-rotores. O preço desse sistema é de aproximadamente US$1919,00[31].

Outra solução de mercado para AP com redundância é o MicroPilot MP2128^{3X}[32]. Neste caso, temos uma redundância tripla em uma única placa. Este sistema conta com um arranjo triplo redundante de sistemas similares de software e hardware. O sistema possui um mecanismo de votação robusto utilizado para seleção de piloto automático. Se qualquer um dos três sistemas falhar, os dois restantes assumem, oferecendo um arranjo de redundância dupla. Se um dos outros dois sistemas falhar, o terceiro assume. Um mecanismo adicional também está incluído para supervisionar esses três sistemas. O preço desse produto é bastante alto com valores a partir de US$14.355,00 até US$21.975,00[33].

Algumas outras soluções de mercado com redundância de hardware são encontradas no drone DJI Matrice 600 Pro[34], que incorpora redundância tripla de GPS e IMU, entretanto, tais soluções possuem um alto custo, cerca de US$8.599,00. Os pilotos automáticos Pixhawk

[30] <https://www.zerotech.com/en/GEMINIen.html>
[31] <https://www.fpvmodel.com/zero-uav-gemini-autopilot-dual-redundancy-autopilot-for-multi-rotors_g480.html>
[32] <https://www.micropilot.com/products-mp21283x.htm>
[33] <https://www.micropilot.com/packages-mp2128g.htm>
[34] <https://www.spreadingwingsstore.com.au/products/dji-matrice-600-pro-drone-with-triple-redundancy-gps-imu>

suportam redundância com até dois GPS conectados, um deles utilizando a entrada serial[35]. O Pixhawk 2 suporta nativamente dois GPS conectados. Nessas placas, existem dois modos de configuração, um utilizando o GPS com maior número de satélites visíveis e outro mais recente, que combina os dados de ambos os dispositivos GPS. O Pixhawk 2 suporta redundância também ao incorporar 3 acelerômetros, 3 giroscópios, 3 magnetômetros e 2 barômetros[36],[37].

Alguns dispositivos para redundância de bateria também já estão disponíveis no mercado como o Auvidea P10[38], que permite a incorporação de duas baterias com características diferentes. Essa redundância visa aumentar o tempo de voo e garantir a alimentação em caso de falha de uma delas. O preço desse dispositivo é de aproximadamente US$49,00[39].

Outra tecnologia para aumentar a segurança é um receptor *Automatic Dependent Surveillance - Broadcast* (ADS-B), que pode se conectar as placas APM e Pixhawk, através da entrada de telemetria 2, e auxilia a evitar colisões com aeronaves tripuladas. O sistema de hardware é capaz de detectar aeronaves comerciais a uma distância de 160 km de raio. Ele implementa a ideia de *sense and avoid* para operações de VANTs no espaço aéreo. O produto é feito pela empresa uAvionix[40] e custa cerca de US$249,00. A FAA está exigindo que todas as aeronaves tripuladas sejam equipadas com o transmissor ADS-B até o ano 2020. O receptor de ADS-B, chamado pingRX[41], recebe a transmissão ADS-B da aeronave automaticamente. Quando uma aeronave atinge uma distância especificada do drone, o piloto em solo recebe um alerta de áudio e vídeo por meio de um *link* de telemetria (ARDUPILOT, 2019a).

As limitações de peso, consumo de energia e preço tornam aspectos, como redundância de hardware, difíceis de serem incorporados, como destacado em Spinka, Holub e Hanzalek (2011). Devido a essas limitações, a presente tese não incorpora nenhum aspecto de redundância de hardware no sistema.

As técnicas de redundância de software podem ser obtidas das seguintes formas: programação n-versões, blocos de recuperação e verificação de consistência.

A programação n-versões, também chamada programação diversitária, é uma técnica de redundância utilizada para obter tolerância à falhas em software (WEBER, 2003). A partir de um problema a ser solucionado são implementadas diversas soluções alternativas, sendo a resposta do sistema determinada por votação. Os sistemas são executados paralelamente.

A técnica blocos de recuperação é semelhante a programação n-versões, porém nessa técnica programas secundários só serão executados caso o programa primário (anterior) apresente um erro. Essa estratégia envolve um teste de aceitação. Os programas são executados

[35] <http://ardupilot.org/copter/docs/common-installing-3dr-ublox-gps-compass-module.html>
[36] <http://www.proficnc.com/>
[37] <https://docs.px4.io/en/flight_controller/pixhawk-2.html>
[38] <https://auvidea.com/pdb/>
[39] <https://www.indiegogo.com/projects/redundant-power-for-drones-and-uavs#/>
[40] <https://uavionix.com>
[41] <https://uavionix.com/products/pingrx>

sequencialmente e testados um a um até que um passe no teste de aceitação. Essa estratégia tolera até n-1 falhas.

A presente tese implementa algumas técnicas de tolerância à falhas em software que não trazem nenhum custo e peso adicional. Essas técnicas foram implementadas dentro sistema IFA, por exemplo, caso o sistema MOSA apresente alguma falha interna ou de comunicação o IFA assume o controle e efetua um pouso emergencial. Nesta tese, dois métodos de programação n-versões foram implementados para o pouso emergencial e exploram características de redundância de software. Os resultados desses estudos são apresentados no artigo Apêndice C. Tais características de redundância no sistema IFA serão exploradas nos capítulos seguintes.

2.16 Autonomia em Sistemas Robóticos

Os veículos aéreos não tripulados podem ter diferentes níveis de autonomia, como, por exemplo, aeronaves remotamente pilotadas (ARPs) ou totalmente autônomas (VAANTs). No entanto, essa simples divisão é pobre, uma vez que os VANTs reais podem oferecer graus intermediários de autonomia. Por exemplo, um ARP pode possuir uma operação de retorno à base de forma autônoma, ou ainda, uma operação de seguir um objeto autonomamente.

A Tabela 7 apresenta uma classificação quanto a autonomia alcançada em sistemas robóticos (OSóRIO, 2014). Ela faz a subdivisão desses sistemas em: autônomo, semi-autônomo, tele-operado e autômato, baseando-se nos mecanismos de percepção, decisão e ação. Nessa tabela, os agentes responsáveis por acionar os mecanismos do sistema robótico são representados por **H** e **R**, que significam Humano e Robótico, respectivamente. Apesar dessa tabela trazer alguns aspectos interessantes sobre a autonomia, ela, ainda, é uma classificação simplista e necessita de níveis mais detalhados.

Tabela 7 – Classificação dos níveis de autonomia em sistemas robóticos.

	Percepção	Decisão	Ação
Categoria	Sensores	Processamento	Motores
Autômato	-	-	**R**
Tele-Operado	**H**	**H**	**R**
Semi-Autônomo	**H/R**	**H/R**	**R**
Autônomo	**R**	**R**	**R**

Fonte: Osório (2014).

Uma escala mais completa e composta por 10 níveis de automação foi concebida em Sheridan e Verplank (1978) e adaptada a aeronaves por Mattei (2015), como visto no Quadro 2. Nessa escala, os níveis progridem do 1, representando operação manual, ao 10, caracterizando total autonomia. Os níveis 2 a 4 estão centrados em quem toma as decisões, o humano ou o computador, já os níveis 5 a 9 estão centrados em como executar essa decisão (PROUD; HART; MROZINSKI, 2003).

Quadro 2 – Escala de graus de automação de Sheridan adaptada a aeronaves.

Grau	Nível	Descrição
Alto	10	**Total Autonomia**: A aeronave toma todas as decisões, age de forma autônoma e ignora o operador humano.
	9	A aeronave informa ao operador, somente se ela decidir que deve.
	8	A aeronave informa ao operador, somente se ele pedir.
	7	A aeronave executa as ações automaticamente e depois informa ao operador.
	6	A aeronave fornece um breve período de tempo ao operador para que ele possa vetar uma ação, antes que ela seja automaticamente executada.
	5	A aeronave envia uma sugestão ao operador e a executa após aprovação.
	4	A aeronave sugere uma alternativa de ação ao operador.
	3	A aeronave analisa e seleciona algumas alternativas de ação ao operador.
	2	A aeronave oferece um conjunto completo de alternativas de ação ao operador.
Baixo	1	**Operação Manual**: A aeronave não oferece nenhuma ajuda ao operador, que deverá tomar todas as decisões e ações.

Fonte: Mattei (2015), Sheridan e Verplank (1978).

Os pilotos automáticos dos VANTs atuais já possuem algumas funções implementadas que executam operações autônomas específicas, como as listadas no Quadro 3. O recurso da navegação baseada em *waypoints* descrito pode ser associado a um robô seguidor de linha. Utilizando os recursos listados nesse quadro, principalmente, a navegação baseada em *waypoints*, dão a aeronave a capacidade de executar uma missão completa de maneira autônoma, como por exemplo, fazer retirada de fotografias sobre uma determinada região. Todavia, a aeronave não é capaz de desviar de obstáculos não previstos, ou ainda, fazer uma atualização de sua missão em pleno voo. Esta tese busca avançar nesse sentido, ao incorporar um computador de bordo que executa um sistema de missão e segurança, utilizando técnicas de inteligência artificial para aumentar o nível de autonomia da aeronave.

2.17 Considerações Finais

Este capítulo apresentou os principais conceitos sobre Veículos Aéreos Não Tripulados, equipamentos aviônicos, ferramentas e tecnologias relacionados a esta tese. Todos esses conceitos dão suporte para a compreensão dos capítulos seguintes.

O próximo capítulo apresentará uma revisão bibliográfica dos principais trabalhos relacionados à presente tese e que darão suporte a criação da arquitetura proposta.

Quadro 3 – Características autônomas implementadas pelos pilotos automáticos.

Característica	Modo de voo	Descrição
Navegação Baseada em *Waypoints*	`Auto`	A aeronave segue, autonomamente, os *waypoints* especificados através do GPS.
Seguir um Objeto	`Follow`	A aeronave segue, autonomamente, algum objeto, utilizando dados de GPS ou processamento de imagens.
Orbitar um Objeto	`Circle`	A aeronave circula autonomamente e continuamente em um determinado ponto.
Retornar para Lançamento	`RTL`	A aeronave voa de volta ao ponto de decolagem. Em geral, ela sobe antes para evitar colisões com árvores ou prédios.
Executar Acrobacias	`Flip`	A aeronave executa, autonomamente, acrobacias, como *rolls* e *loops*.
Manter Altitude	`AltHold`	A aeronave mantém, autonomamente, a altitude através do barômetro.
Decolar/Take-off	-	A aeronave decola autonomamente até uma altitude padrão especificada.
Pousar/Landing	`Land`	A aeronave pousa autonomamente em uma determinada posição.
Pairar/Hover	-	A aeronave flutua autonomamente, mantendo sua posição por um determinado tempo.
Failsafe	-	A aeronave executa, autonomamente, o pouso automático ou RTL após perder do sinal de controle.

Fonte: Elaborada pelo autor.

CAPÍTULO

3

REVISÃO BIBLIOGRÁFICA

“ Se fui capaz de ver mais longe, é porque me apoiei em ombros de gigantes. ”

Isaac Newton

3.1 Considerações Iniciais

Este capítulo apresentará uma revisão bibliográfica dos principais trabalhos relacionados a esta tese, algumas arquiteturas propostas na literatura e, por fim, um paralelo entre essas arquiteturas e a proposta neste trabalho.

3.2 Trabalhos Relacionados

A tese descrita em Figueira (2016) apresenta os conceitos envolvidos na arquitetura do MOSA e sua estrutura básica. Esse sistema faz a integração de vários componentes, como sensores, processador e hardware de comunicação, além de apresentar processamento em tempo real dos dados. Ele é responsável por controlar o percurso e os sensores do VANT durante toda a realização da missão.

O sistema MOSA é utilizado em Figueira *et al.* (2015) numa aplicação envolvendo a geração automática de mapas temáticos para monitoramento ambiental, cujo objetivo é identificar eventos de interesse, baseados em atividade sonora, como o disparo de armas de fogo ou uso de motosserras em uma floresta. Para isso, a integração dos dados da câmera térmica, localizada no VANT, e o conjunto de microfones, posicionados sobre o solo, é realizada com o intuito de localizar o evento. Além disso, MOSA gerencia toda a missão do VANT, que inclui: voar até a origem do evento e realizar coleta de dados. Nesse trabalho, o sistema é simulado utilizando Matlab Simulink, ou seja, ele não foi de fato embarcado numa aeronave.

O conceito do IFA está relacionado a um sistema supervisor que monitora todos os aspectos de segurança da aeronave, conforme proposto em Mattei (2015). Esse monitoramento é feito através de um conjunto de sensores que verifica o funcionamento dos principais componentes da

aeronave, visando mitigar acidentes ou, simplesmente, atualizar o plano de voo. Uma proposta de implementação do sistema IFA, chamada *In-Flight Awareness Augmentation Systems* (IFA^{2}S), é descrita em Mattei *et al.* (2015), apresentando como seria sua integração a uma aeronave do mundo real.

As aplicações reportadas em Figueira *et al.* (2015) e Mattei *et al.* (2015) não implementaram de fato um sistema embarcado e voos reais não foram realizados. Os autores em Figueira *et al.* (2015) apenas ilustram os conceitos do sistema MOSA, na aplicação mencionada, utilizando Matlab Simulink já em Mattei (2015), outra simulação bastante simples, integrando o Labview (linguagem de programação baseada em fluxo de dados) ao simulador de voo X-Plane, é realizada para ilustrar o uso do IFA.

Não há um sistema que integre MOSA e IFA, reportado na literatura até o momento. Também não há sistemas embarcados, efetivamente desenvolvidos, que permitam o uso de tais conceitos combinados a outros sistemas em uma plataforma para VANTs versátil e resiliente. Esses são alguns dos pontos a serem tratados neste doutorado.

No trabalho de Pires (2014), um protocolo de comunicação entre o sistema MOSA e a aeronave foi desenvolvido, chamado de *Smart Sensor Protocol* (SSP). Esse protocolo foi avaliado utilizando a ferramenta UPPAAL, para verificação de sistemas de tempo real. Foram feitos, também, experimentos em um ambiente controlado utilizando duas placas Arduíno e dois computadores. A arquitetura do protocolo SSP é formada por quatro elementos principais: um processador de missão, um *middleware* SSP para o processador, uma aeronave não tripulada e um *middleware* SSP para a aeronave. Apesar desse protocolo estar documentado, o mesmo não está disponibilizado e não se tem uma versão funcional dele. Dessa forma, durante esse doutorado, um protocolo próprio será criado a fim de se fazer a comunicação entre os sistemas MOSA, IFA e a aeronave, como descrito na Seção 6.4.

Existem muitas aplicações utilizando técnicas de Inteligência Artificial (IA) com sistemas de VANT, como relatado em Gonzalez *et al.* (2016), Ramirez-Atencia *et al.* (2017), Strupka, Levchenkov e Gorobetz (2017). Nos sistemas, avaliados em Gonzalez *et al.* (2016), para detecção de fauna silvestre uma detecção automatizada é proposta utilizando aquisição de imagens térmicas e processamento de vídeo. Os problemas de planejamento de missão e replanejamento são abordados por Ramirez-Atencia *et al.* (2017), em que são aplicadas técnicas de IA para reduzir a carga de trabalho dos operadores. Os métodos propostos enfatizam as melhorias alcançadas nas operações de VANT das estações de controle de solo. As vantagens da IA e dos sistemas de controle da autonomia dos VANTs são discutidas em Strupka, Levchenkov e Gorobetz (2017).

Uma falha crítica na aeronave, detectada por um sistema com o IFA, pode levar a um replanejamento da missão ou a sua interrupção. Nesse caso, algoritmos que permitam pousar a aeronave em segurança ou refazer o plano de voo deverão ser integrados. Existem poucos trabalhos enfatizando o pouso de aeronaves diante de situação crítica, como reportados em Meuleau *et al.* (2009), Meuleau *et al.* (2011), Li (2013), NASA (2017). Nos dois primeiros

trabalhos, considera-se o pouso emergencial de aeronaves tripuladas de grande porte utilizando um algoritmo baseado no A*. Trabalhos tratando o pouso emergencial de VANTs são ainda mais escassos. Os trabalhos de Li (2013), Li, Chen e Li (2014) tratam o pouso emergencial, no entanto não são bem definidas as falhas e não são apresentados testes rigorosos considerando tais falhas.

Um sistema para pouso emergencial foi desenvolvido pela NASA (2017), chamado Safe2Ditch, e é capaz de efetuar o pouso da aeronave em caso de situações críticas, ocorridas na bateria ou no motor. Esse sistema executa sobre um processador embarcado efetuando o gerenciamento da saúde do VANT e detectando possíveis falhas através de um auto-diagnóstico. Quando uma falha é detectada, a aeronave é conduzida, utilizando conhecimento do local de voo, e ao se aproximar da região desejada ela utiliza visão computacional para selecionar a região mais adequada ao pouso emergencial.

O trabalho de Kim *et al.* (2013) trata o pouso totalmente autônomo de um VANT de asa fixa, em que ele possui sensores, como câmera e GPS. Esse trabalho difere do aqui proposto ao efetuar o pouso sobre uma rede de recuperação e não sobre regiões adequadas ao pouso. Os autores em Kim *et al.* (2013) efetuam o pouso utilizando visão computacional, e não utilizam dados de GPS com regiões previamente cadastradas, como se pretende utilizar nesta tese. O replanejamento de caminho para VANTs é abordado por Evers *et al.* (2014) de modo que novos alvos possam se tornar disponíveis durante a execução da missão do VANT. Uma heurística de replanejamento de rotas é proposta por eles para definir uma trajetória que melhore a chance de alcançar alvos previstos e imprevistos. No entanto, esses trabalhos não resolvem o problema de replanejamento de caminho sob situação crítica.

Resultados obtidos pelo autor desta tese, durante seu mestrado, são reportados em Arantes *et al.* (2015), Arantes (2016), em que o pouso de VANTs, considerando situações críticas, foi tratado. Um total de sete métodos para o planejamento da rota de pouso é comparado: uma Heurística Gulosa (GH), dois métodos baseados em Algoritmos Genéticos (GA), dois métodos baseados em Algoritmos Genéticos Multi-Populacionais (MPGA) e dois métodos baseados em Programação Linear Inteira-Mista (PLIM). Um gerador de mapas foi desenvolvido, permitindo a criação de 600 cenários diferentes para avaliação dos métodos. Experimentos utilizando o simulador de voo FlightGear (FG) também foram executados. Uma das contribuições deste projeto de doutorado foi a integração de sistemas que permitam o replanejamento de rota para execução da missão ou para realização de um pouso de emergência. O foco nesse desenvolvimento está centrado em um replanejamento autônomo de trajetória, visando maior segurança e autonomia no processo de tomada de decisão pela aeronave durante a execução da missão.

Além do replanejamento da trajetória, procedimentos que permitam mitigar falhas na aeronave precisam ser implementados. Segundo os autores em Morozov e Janschek (2016), soluções baseadas em hardware apresentam a vantagem de serem mais seguras e capazes de lidar diretamente com as falhas, porém trazem um custo financeiro elevado e podem deteriorar

com o decorrer do tempo. Por outro lado, soluções baseadas em software podem não permitir a correção de erros, todavia são capazes de detectar antecipadamente as falhas no sistema. A desvantagem está em um elevado tempo de execução do sistema, o que pode demandar um maior consumo de memória e, consequentemente, levar à degradações de desempenho dos algoritmos de controle. Por isso, uma implementação baseada em software para detectar falhas, que busca evitar degradação de desempenho, será apresentada.

Dependendo do tipo de falha detectada, um ajuste dos parâmetros do sistema pode não ser suficiente ou pode indicar um problema sério o bastante para demandar atitudes mais drásticas, como replanejar a rota para retornar à base, replanejar a rota para executar um pouso de emergência ou desligar o motor seguido pelo acionamento do paraquedas. Um sistema redundante, envolvendo os métodos GH e GA executando em paralelo, foi desenvolvido durante esse doutorado em Arantes *et al.* (2017).

A tese, em Arantes (2017), abordou o problema de planejamento de missão para VANTs. O autor elaborou um conjunto de algoritmos e modelos para tratar tais problemas, dentre os algoritmos propostos destacam-se o *Hybrid Genetic Algorithm for mission* (HGA4m) e o *Chance-Constrained Qualitative State Plan for mission* (CCQSP4m). O presente trabalho pretende incorporar tais planejadores, HGA4m e CCQSP4m, dentro do sistema MOSA e aplicá-los em voos reais, o que não foi feito no trabalho de Arantes (2017).

O algoritmo evolutivo, HGA4m, apresentado em Arantes *et al.* (2016), define um planejamento de caminho para que os VANTs executem missões em um ambiente não convexo com incertezas. O HGA4m combina um Algoritmo Genético Multi-Populacional (MPGA) de Toledo *et al.* (2009) com o grafo de visibilidade de Kuwata (2003), resolvendo modelos de programação linear para encontrar caminhos. O HGA4m é aplicado para encontrar rotas para um conjunto de 50 mapas e seus resultados são comparados com uma abordagem exata e heurística, com resultados competitivos obtidos dentro de um curto tempo computacional.

O hardware do VANT pode ser projetado para integrar diferentes sistemas embarcados utilizando *companion computers* (CC), como o Intel Edison, o Raspberry Pi, o BeagleBone, o ODroid, o Intel Aero, entre outros. O CC é uma maneira eficiente de vincular os sistemas embarcados ao piloto automático, em que um computador, baseado em Linux, pode ser utilizado para executar uma ferramenta, como o Dronekit. Os autores em Li *et al.* (2017) desenvolvem um *framework* com o Raspberry Pi para realizar processamento de imagens para detecção de objetos, enquanto o rastreamento de imagens com o computador embarcado ODroid XU4 é feito por Kurt e Altuğ (2017). Um BeagleBone Black é utilizado por Velasquez, Argueta e Mazariegos (2016) com processamento de imagens *onboard* no VANT para executar missões em uma plantação de cana-de-açúcar. Um aplicação para vigilância de multidões com VANT e *Internet of Things* (IoT) é descrito em Motlagh, Bagaa e Taleb (2016), em que uma plataforma de VANT é configurada com um piloto automático, Raspberry Pi e câmera de vídeo. As imagens adquiridas são processadas utilizando algoritmos de reconhecimento de faces.

Uma arquitetura é descrita em Brown, Estabrook e Franklin (2011) para sistemas de VANTs que executam missões de resgate de caminhantes perdidos no deserto, em que imagens obtidas pela câmera acoplada à aeronave são processadas. Uma arquitetura focada em sistemas de pulverização contra pragas agrícolas, proposta em Xue *et al.* (2016), permite a execução automática da aspersão sobre a plantação. Um sistema para controle da trajetória, proposto em Prodan *et al.* (2013), baseia-se em controle preditivo e considera pertubações externas ao voo. Uma arquitetura para controle de trajetória, combinando hardware e software, desenvolvida em Ramasamy *et al.* (2016), permite ao VANT executar manobras para evitar obstáculos e outras aeronaves em tempo real. Uma arquitetura para aplicações de detecção de pragas e pulverização, baseada em visão computacional, é proposta em Alsalam *et al.* (2017). O trabalho de Boubeta-Puig *et al.* (2018) implementa uma solução para a aplicação de monitoramento de poluição sonora ambiental produzida pela decolagem e aterrissagem de aeronaves próximas a aeroportos. Em Chiaramonte (2018) é apresentado um sistema para detecção e desvio de obstáculos aplicados em VANTs, utilizando uma câmera monocular. Os trabalhos mencionados apresentam arquiteturas voltadas a uma aplicação específica. Esta tese pretende contribuir com a proposição e desenvolvimento de uma arquitetura mais versátil, que englobe mais de uma aplicação.

Uma Arquitetura Orientada a Serviços (SOA) para VANTs foi apresentada em Amenyo *et al.* (2014) com o objetivo de automatizar o controle de mosquitos vetores de doenças, como a malária e a dengue. Um sistema de vigilância, controle, supressão e eliminação de vetores é introduzido. Os autores propõem uma solução de software, em que o Piloto Automático, chamado MedizDroids, é especializado para o problema abordado. Ao contrário do trabalho desenvolvido aqui, o trabalho em Amenyo *et al.* (2014) não apresenta um sistema de geração automática de rotas para ambientes externos, nem um sistema de segurança, nem distingue o sistema específico da missão (controle) do de controle de voo (AP) dificultando a utilização em outras missões. O projeto MedizDroids está, atualmente, na fase de modelagem computacional e representação de conhecimento para suportar a automação e integração das tarefas específicas da aplicação.

O trabalho em Koubaa e Qureshi (2018) desenvolveu um sistema na nuvem para o controle de um ou vários *drones* aplicados a missões de vigilância, rastreando a posição GPS do dispositivo a ser seguido. A arquitetura proposta é baseada, principalmente, em *Internet of Drones* (IoD) e todo o processamento, controle de missão, coleta de dados e monitoramento de VANT são alocados na aplicação na nuvem. Apesar de algumas semelhanças com esta tese, a proposta de Koubaa e Qureshi (2018) não possui planejamento de rotas para desvio de obstáculos e não possui um sistema de vigilância da saúde da aeronave.

Os autores em Schrage (1999) apresentam uma arquitetura *Plug-and-Play* com o objetivo de desenvolver métodos para sistemas dinâmicos e complexos para VANTs inteligentes. Isso é feito a partir de um controlador de nível médio que troca/permuta entre os módulos de controle

de falhas, controle de voo e reconfiguração de voo no VANT. Uma arquitetura *Plug-and-Play*, baseada em componentes de tempo real, é descrita em Ippolito, Pisanich e Al-Ali (2005) para o desenvolvimento de sistemas embarcados. A arquitetura visa facilitar e acelerar o processo de prototipação e desenvolvimento de hardware, em que não é imposta qualquer conceituação dos periféricos conectados.

Uma arquitetura cognitiva é proposta em Gunetti, Dodd e Thompson (2010) para VANTs. Ela utiliza memória paralela e associativa, deliberação baseada em preferências, manutenção de crenças, decomposição de objetivos e adaptação através de uma generalização da experiência. Esses aspectos fornecem uma arquitetura robusta para voos autônomos. Existem três agentes nomeados, como planejador, executor e gerenciador de missão, que permitem executar ações, como análise de alvo, ataque de alvo, órbita em uma posição, entre outras, dentro do software de simulação.

Outras arquiteturas de software foram criadas especificamente para VANT, como apresentado em Briggs (2012). Comportamentos deliberativos e reativos, incerteza, riscos e flexibilidade são alguns dos requisitos necessários para robôs móveis. Além desses requisitos, é conveniente que os VANTs executem automaticamente manobras de alto desempenho, adaptem dinamicamente sua rota, detectem e evitem ameaças, identifiquem e compensem falhas e controlem o processo de pouso e decolagem.

O presente doutorado introduz uma arquitetura para VANTs visando preencher diversas lacunas encontradas na literatura em projetos de desenvolvimento de plataformas de hardware e software. A ideia é desenvolver uma plataforma que permita a integração de sistemas, como MOSA (FIGUEIRA *et al.*, 2015), IFA (MATTEI, 2015), algoritmos de planejamento de missão (ARANTES *et al.*, 2016; ARANTES, 2017), algoritmos de replanejamento de rotas (ARANTES *et al.*, 2015; ARANTES, 2016), controle de trajetória (PRODAN *et al.*, 2013; RAMASAMY *et al.*, 2016) com o propósito de executar diferentes tipos de missão com segurança e autonomia (BROWN; ESTABROOK; FRANKLIN, 2011; XUE *et al.*, 2016; ALSALAM *et al.*, 2017).

Os conceitos de MOSA e IFA foram definidos para serem tratados em conjunto, entretanto nenhum trabalho foi desenvolvido até o momento nesse sentido. O presente doutorado visa preencher essa lacuna, implementando um sistema integrado que monitora a missão e a segurança. Os algoritmos desenvolvidos serão avaliados, inicialmente, em simuladores de voo, sendo embarcados em uma próxima etapa para sua validação em testes reais. Conforme mencionado no capítulo anterior, simulações *Hardware-In-The-Loop* (HITL) combinam sistemas simulados em software com hardware físico. As simulações HITL têm facilitado o desenvolvimento em numerosos campos, incluindo engenharia aeronáutica (CAI *et al.*, 2009), aeroespacial (FRITZ *et al.*, 2015), automotiva, sistemas de energia, manufatura e robótica.

A tarefa de projetar e testar algoritmos de controle de VANTs é uma tarefa complexa, onerosa e que pode gerar riscos à propriedades e pessoas. Simulações nem sempre são suficientes para testar os algoritmos, pois não levam em consideração todos os aspectos como funciona-

mento do controlador, ruídos dos sensores e problemas associados aos atuadores. Nesse sentido, simulações HITL surgiram de forma a combinar partes do sistema em hardware com partes em software. Os autores em Cai *et al.* (2009) apresentam um *framework* de simulação HITL para um VANT de asa rotativa de pequena escala, mais especificamente, o Raptor 90. Nesse trabalho são apresentados, em linhas gerais, os elementos que compõem tal ambiente de simulação, como hardware *onboard*, controlador de voo, estação de terra e um software de integração. Comparações entre as simulações HITL e os voos reais são feitas. No entanto, esse trabalho apresenta como grande limitação o pequeno número de experimentos, apenas três, utilizados na validação do *framework* e resultados apenas qualitativos. No trabalho de Gans *et al.* (2009) é apresentado um ambiente para executar simulações HITL para VANT utilizando um sistema de controle de visão, que incorpora câmera, aviônicos e túnel de vento. A plataforma desenvolvida possui um VANT com piloto automático *onboard* e uma câmera interligados ao software de realidade virtual.

As simulações *Software-In-The-Loop* (SITL) também têm facilitado bastante o desenvolvimento de sistemas de VANTs. Na área de VANTs, simulações SITL têm se mostrado melhor que simulações HITL, sendo uma tendência mais atual. Algumas ferramentas como o Mission Planner têm descontinuado as simulações HITL e têm mantido apenas as simulações SITL. Isso justifica o fato de termos utilizado mais simulações SITL, como apresentadas na Figura 11, do que as simulações HITL, como da Figura 12. As simulações HITL, apresentadas na Figura 13, são utilizadas nesse trabalho, uma vez que combinam aspectos de SITL e HITL (aqui o HITL é basicamente composto pelo CC e possíveis sensores/atuadores acoplados a ele).

3.3 Arquiteturas Propostas na Literatura

Como mencionado, anteriormente, os autores em Brown, Estabrook e Franklin (2011) descrevem uma arquitetura para VANTs, chamada WiND, que poderia ser utilizada em missões de resgate de caminhantes perdidos no deserto. Nessa arquitetura, um microprocessador é responsável pelo controle de três partes do VANT: piloto automático, sistema de navegação e sistema de processamento de imagem. A arquitetura de hardware utiliza o AP Paparazzi, integrado com a placa de processamento embarcado PandaBoard. A Figura 15 apresenta um diagrama da arquitetura utilizada.

Um sistema de controle preditivo, descrito em Prodan *et al.* (2013), é capaz de controlar a trajetória da aeronave sob perturbações externas durante o voo. Esse sistema é apresentado na Figura 16 e possui dois módulos principais, em que o mais rápido é executado embarcado no VANT, enquanto o mais lento é executado a partir da estação terrestre. O sistema de bordo controla a dinâmica da aeronave, minimizando os erros durante a execução da trajetória, e o sistema da estação terrestre planeja toda a rota.

Ainda em Prodan *et al.* (2013), a arquitetura proposta foi utilizada em uma aplicação

Figura 15 – Arquitetura utilizada no sistema de busca e resgate no deserto.

Fonte: Brown, Estabrook e Franklin (2011).

que efetua a retirada de fotografias e as envia para a GCS. Os experimentos foram realizados utilizando diversas aeronaves de asa fixa com o AP Piccolo II. Algumas desvantagens dessa arquitetura, em comparação a nossa, são: o alto custo da plataforma de hardware; o processamento executado na GCS (*offboard*); a decolagem e o pouso executados de forma manual, utilizando o controle de rádio; e o fato de não possuir nenhum sistema gerenciador de segurança.

Figura 16 – Arquitetura utilizada no sistema de controle da trajetória.

Fonte: Prodan *et al.* (2013).

Os autores em Xue *et al.* (2016) apresentam uma arquitetura de hardware e software para navegação e pulverização automática em campos de cultivo. Um esquema dessa arquitetura é mostrado na Figura 17. Nesse trabalho, ambos os sistemas de controle de voo e controle de pulverização estão embarcados em um helicóptero não tripulado, mas o planejamento do trajeto é feito a partir da estação terrestre. A aeronave é controlada de acordo com coordenadas determinadas previamente em solo, ou seja, a rota foi pré-planejada. A estação de solo reporta

os dados da missão ao controle de voo, que segue os *waypoints* e envia um sinal para acionar o sistema de controle de pulverização. O microcontrolador MSP430 foi utilizado para realizar a aspersão sobre o campo de cultivo. O *status* em tempo real do sistema de pulverização é enviado de volta ao computador de bordo, que reporta à estação terrestre.

Figura 17 – Arquitetura utilizada no sistema de pulverização de pragas agrícolas.

Fonte: Xue *et al.* (2016).

Uma arquitetura para hardware e software, chamada *LIDAR Obstacle Warning and Avoidance System* (LOWAS), é descrita em Ramasamy *et al.* (2016) para aplicações com VANTs. LOWAS utiliza tecnologia *Light Detection and Ranging* (LIDAR), que emprega sensores laser e componentes eletro-óptico de direcionamento da aeronave, que proporcionam alta resolução e precisão angular em diversas condições operacionais. O sistema é do tipo *sense and avoid* e consegue realizar um acompanhamento em tempo real da navegação, facilitando o tratamento de erros e incertezas durante a execução da trajetória. Assim, LOWAS passa a ser um sistema de auxílio à navegação, concebido para detectar obstáculos terrestres e aéreos potencialmente perigosos na trajetória de voo, fornecendo os avisos necessários à execução de manobras de evasão. Como exemplos de obstáculos evitados, tem-se: linhas de transmissão de energia, árvores e construções civis. A Figura 18 ilustra a arquitetura LOWAS.

As interações com o piloto na estação de solo envolvem *links* de comunicação, que permitem a comunicação do LOWAS com a estação de solo e com o sistema de gerenciamento de tráfego aéreo. Os dados de telemetria precisam ser trocados entre o VANT e a estação de solo, permitindo rastreamento de veículos, controle de missão e atualizações do perfil da missão. LOWAS emprega três algoritmos: predição da trajetória futura, cálculo das colisões potenciais com os obstáculos detectados e geração de trajetórias para evitar colisão. Como desvantagens dessa arquitetura, podemos citar: o alto custo da plataforma de hardware; o processamento executado na GCS (*offboard*); e o fato da arquitetura ser específica para uma aplicação.

Um sistema genérico, baseado na visão computacional, é relatado em Alsalam *et al.*

Figura 18 – Arquitetura utilizada no sistema para evitar colisões com obstáculos.

Fonte: Ramasamy *et al.* (2016).

(2017) para controlar um VANT em aplicações, como pulverização em campos de cultivo, detecção de pragas, entre outros. A Figura 19 mostra a arquitetura desse sistema. Nesses cenários, os pontos de destino atuais podem ser alterados e as correções de caminho podem ser executadas de forma autônoma durante o voo. O nível de autonomia proposto para o sistema a bordo permite detectar a localização de ervas daninhas ou pragas. Isso leva a pulverização de apenas um local em vez de todo o campo de cultivo. Esse método é aplicado na área de agricultura de precisão para identificação de erva daninha, através da utilização de uma câmera acoplada ao sistema.

Ainda em Alsalam *et al.* (2017), o sistema de tomada de decisões embarcado codifica o *loop* de Observação, Orientação, Decisão e Ação (OODA), e o hardware é projetado para controlar um quadricóptero. Um fluxograma do OODA está ilustrado na Figura 20. Um *companion computer*, o Odroid U3 +, foi integrado com piloto automático Pixhawk e um Arduíno. O *Robotic Operating System* (ROS) foi utilizado durante a implementação do sistema para desenvolver nós para navegação, câmera e detecção.

A tomada de decisão em tempo real também é suportada pela arquitetura de VANT, proposta por Boubeta-Puig *et al.* (2018), que integra o Processamento de Eventos Complexos (CEP, do inglês *Complex Event Processing*) a uma Arquitetura Orientada a Serviços (SOA, do inglês *Service-Oriented Architecture*). O sistema lida com situações críticas em que o CEP processa e avalia um grande número de eventos em tempo real. O SOA permite que o sistema lide com dados de diferentes origens e domínios. A arquitetura de hardware emprega um VANT DJI F-450, um piloto automático ArduPilot APM 2.6, um CC Raspberry Pi 2, um Arduíno Uno e um conjunto de sensores e atuadores.

Figura 19 – Arquitetura utilizada no sistema de tomada de decisão para pulverização.

Fonte: Alsalam *et al.* (2017).

Figura 20 – Fluxograma para sistema de tomada de decisão durante a missão do VANT.

Fonte: Alsalam *et al.* (2017).

O trabalho de Boubeta-Puig *et al.* (2018) considera uma aplicação de monitoramento de poluição sonora ambiental produzida pela decolagem e aterrissagem de aeronaves próximas a aeroportos. Esse trabalho utiliza a biblioteca Dronekit e apresenta alguns mecanismos de segurança adicional, durante a realização da missão, como fazer RTL e pousar na vertical. O RTL é executado caso a bateria esteja abaixo de um determinado limiar. A plataforma possui, também, um sistema de verificação de invasão de espaço aéreo restrito, como aeroportos.

Figura 21 – Arquitetura utilizada no sistema de monitoramento de ruído ambiental.

Fonte: Adaptada de Boubeta-Puig *et al.* (2018).

O autor Chiaramonte (2018) apresenta um sistema para detecção e desvio de obstáculos aplicados em VANTs, utilizando uma câmera monocular. Nesse contexto, esse sistema pode ser dividido em dois subsistemas: o de *sense* ou detecção, que é voltado ao estudo de técnicas para detectar possíveis obstáculos e; o de *avoidance* ou desvio, que é relacionado a execução de manobras evasivas para evitar possíveis colisões. Nesse trabalho, é assumido que a aeronave possui um sistema gerenciador de missão ou um plano de voo pré-estabelecido. A arquitetura geral desse sistema é apresentada na Figura 22, em que os dados do VANT e da câmera são transmitidos para um computador em solo. O sistema conta com módulos que fazem a detecção do obstáculo, estimativa da posição do obstáculo e, então, decide se atualiza a trajetória da aeronave desviando de possíveis obstáculos.

Figura 22 – Arquitetura utilizada no sistema de detecção e desvio de obstáculos.

Fonte: Adaptada de Chiaramonte (2018).

O trabalho de Chiaramonte (2018) utiliza a biblioteca Dronekit para comunicação com o piloto automático do VANT. Todo o processamento de imagens e a atualização da rota é

executado em solo (*offboard*), pois não foi embarcado na aeronave nenhum computador de bordo. Além dos experimentos simulados, alguns experimentos reais também foram conduzidos, utilizando o quadricóptero 3DR Solo equipado com o AP Pixhawk. .

O presente trabalho introduz uma arquitetura genérica, com uma separação clara entre sistemas de execução de missão e consciência. Esta não é uma abordagem usual, como analisado em Prodan *et al.* (2013), Figueira *et al.* (2015), Ramasamy *et al.* (2016), Xue *et al.* (2016), Alsalam *et al.* (2017), em que os sistemas propostos são mais comprometidos com um tipo específico de missão. Nosso objetivo é avançar propondo uma arquitetura para VANTs inteligentes, conforme apresentado nos trabalhos acima, mas com sistemas resilientes e controle de falhas similares àqueles em Mattei (2015), Ramasamy *et al.* (2016), Boubeta-Puig *et al.* (2018), Chiaramonte (2018).

Embora o trabalho em Gunetti, Dodd e Thompson (2010) apresente semelhanças com nosso estudo, todo o desenvolvimento dos autores é dedicado a um sistema de simulação de voo, não apresentando testes reais. Os autores Ippolito, Pisanich e Al-Ali (2005) focam no desenvolvimento da comunicação entre as partes, mas seu trabalho não possui planejamento de missão e segurança de voo. Em Alsalam *et al.* (2017) também é apresentado um sistema orientado a missão, porém nenhum sistema de segurança de voo ou de análise da aeronave é proposto, assim como as arquiteturas propostas em Briggs (2012).

A Tabela 8 fornece uma comparação entre o nosso trabalho e os da literatura, avaliando se o nível de autonomia inclui uma clara separação entre os sistemas de execução da Missão e Segurança. Também é considerado se os experimentos são baseados em simulações (virtuais) ou em voos reais e se o núcleo (planejamento de rotas) da arquitetura proposta é executado *onboard* ou *offboard*. Avaliamos se a arquitetura suporta rotas pré-planejadas, se os componentes podem ser adicionados *Plug and Play* (P&P) e se a arquitetura foi desenvolvida pensando na Separação de Interesse (SdI). Finalmente, tentamos estimar se a arquitetura proposta é de baixo custo, com base nos recursos tecnológicos empregados.

Tabela 8 – Comparando contribuições baseadas em algumas características.

Ano	Autores	Missão	Segurança	Ambiente	*Onboard*	*Offboard*	Pré-Planej.	P&P	SdI	Baixo Custo
2010	Gunetti *et al.*	Sim	Não	Virtual	Não	Sim	N/D	N/D	N/D	N/D
2011	Brown *et al.*	Sim	Não	Real/Virtual	Sim	Não	Sim	Não	Não	Não
2013	Prodan *et el.*	Sim	Não	Real/Virtual	Não	Sim	N/D	Não	Não	Não
2015	Mattei	Não	Sim	Virtual	Não	Não	Sim	Sim	Não	N/A
2016	Figueira	Sim	Não	Virtual	Não	Não	Sim	Sim	Não	N/A
2016	Ramasamy *et al.*	Não	Sim	Virtual	Não	Sim	Sim	Não	Não	Não
2016	Xue *et al.*	Sim	Não	Real	Não	Não	Sim	Não	Não	Não
2017	Alsalam *et al.*	Sim	Não	Real/Virtual	Sim	Não	Sim	Não	Não	Sim
2018	Boubeta-Puig *et al.*	Sim	Sim	Real/Virtual	Sim	Não	Sim	Não	Não	Sim
2018	Chiaramonte	Não	Sim	Real/Virtual	Não	Sim	Sim	Não	Não	Não
2019	Arantes, J. S.	Sim	Sim	Real/Virtual	Sim	Sim	Sim	Sim[1]	Sim	Sim

Fonte: Elaborada pelo autor.

[1] Avaliado apenas em nível lógico

3.4 Considerações Finais

Este capítulo apresentou os principais trabalhos relacionados, que servirão de base para o desenvolvimento da arquitetura proposta na presente tese.

O próximo capítulo apresentará os sistemas MOSA e IFA detalhadamente, destacando as diferenças entre as propostas originais e a versão implementada neste trabalho.

CAPÍTULO

4

SISTEMAS MOSA E IFA

" A ciência nunca resolve um problema sem criar pelo menos outros dez. "

George Bernard Shaw

4.1 Considerações Iniciais

O presente capítulo apresentará de maneira mais detalhada os conceitos e ideias definidas por Figueira (2016) e Mattei (2015) durante suas teses de doutorado, em que as arquiteturas dos sistemas MOSA e IFA foram definidas e estruturadas. Esses trabalhos serviram como modelos de referência e guiaram toda a implementação dos sistemas MOSA e IFA na presente tese.

4.2 Sistema MOSA

Esta seção traz as principais ideias do sistema *Mission Oriented Sensor Array* (MOSA), inicialmente, idealizadas pelo prof. Onofre Trindade Júnior e estruturadas e reportadas no trabalho de doutorado de Figueira (2016). Esse trabalho foca na definição de um modelo de referência para o desenvolvimento de sensores inteligentes, orientados à missões específicas. Nesse trabalho, os principais objetivos são a modelagem de uma arquitetura de referência; uma implementação de referência para sistemas MOSA; e a geração de mapas temáticos através de sensores de imagem e sonoros. A autora destaca a necessidade de haver separação lógica e física entre o sistema de segurança IFA (sistema crítico em segurança) e o sistema de missão MOSA (sistema não crítico em segurança).

Segundo Figueira (2016), a importância da integração de múltiplos sensores ao MOSA, visa a geração de informação pronta para uso, em tempo real, a partir do processamento embarcado dos dados coletados. A autora salienta outro fator importante que motiva e impulsiona o desenvolvimento de sistemas MOSAs: a redução do grande volume de dados nos enlaces de comunicação. O sistema MOSA deve controlar a rota a ser percorrida em cada missão específica,

enviando comandos de navegação para o AP da aeronave. Essas rotas são planejadas em solo como parte do processo de planejamento de missão.

A autora traz a ideia de múltiplos sistemas MOSAs, sendo um para cada missão. A seguir são destacadas as premissas básicas para o desenvolvimento de MOSAs:

- A aeronave é apenas um meio de transporte para os sensores de missão. A definição e controle da missão estão dentro do sistema MOSA;
- O MOSA determina o caminho de navegação de voo e define a maioria dos parâmetros de voo. A aeronave pode, por razões de segurança, não seguir os comandos do MOSA e, eventualmente, terminar o voo;
- A aeronave pode fornecer dados de voo ou de posicionamento para o sistema MOSA;
- Versatilidade e flexibilidade (adequação da matriz de sensores a missões específicas);
- Uma interface padrão entre a carga útil, *payload*, (parte não crítica) e a aeronave/controle de voo (parte crítica);
- Funcionalidades *plug and play*, que facilitam a interoperabilidade (entre sensor e aeronave);
- Processamento de dados em tempo real, evitando a transmissão de grandes quantidades de dados brutos;
- Realização de múltiplas missões em paralelo, utilizando as mesmas fontes de dados. Por exemplo, mapear matas nativas e detectar plantação de maconha ou outros ilícitos.

A autora Figueira (2016), no desenvolvimento da arquitetura MOSA, utilizou técnicas de Desenvolvimento Orientado a Modelos (MDD, do inglês *Model Driven Development*) que empregam blocos padrões de hardware e software através de ferramentas de desenvolvimento e simulação. O sistema MOSA foi desenvolvido em Matlab/Simulink. O Matlab é um ambiente para computação numérica, visualização e programação, desenvolvido pela empresa MathWorks[1]. O Simulink é um ambiente de diagrama de blocos para simulação de múltiplos domínios e Projeto Baseado em Modelo[2] (MBD, do inglês *Model-Based Design*).

É destacado em Figueira (2016), que um sistema básico de MOSA precisa de pelo menos um sensor e módulos de hardware e software para possibilitar: (i) a comunicação do MOSA com a aeronave; (ii) a aquisição de dados e seu pré-processamento; (iii) a fusão de dados e a geração do produto final, como por exemplo, mapas temáticos.

A autora Figueira (2016) ilustra, na Figura 23, uma proposta de diagrama funcional básico da arquitetura MOSA e o inter-relacionamento entre os componentes do sistema. Módulos com bordas tracejadas são considerados opcionais. Este mesmo diagrama pode variar em

[1] <https://www.mathworks.com/products/matlab.html>
[2] <https://www.mathworks.com/help/simulink/index.html?s_tid=srchtitle>

complexidade e número de componentes, dependendo de cada aplicação em particular. Diversos sensores podem integrar um sistema MOSA. Os dados provenientes desses sensores são adquiridos e pré-processados por módulos específicos de hardware e/ou software. Ela salienta também a possibilidade da integração de sensores, como GPS, IMU, câmeras fotográficas, câmera de vídeo em diferentes regiões do espectro, *scanners* a laser, radares e sensores acústicos.

Figura 23 – Diagrama funcional básico da arquitetura MOSA.

Fonte: Figueira (2016).

Figueira (2016) destaca que a comunicação do sistema MOSA com o restante do SisVANT se dá primariamente pela interface SSP/SSI (*Smart Sensor Protocol*/*Smart Sensor Interface*). Isso inclui a comunicação com as estações de controle situadas em solo. Ela apresenta uma taxonomia de operações essenciais, em que são descritas diversas funções presentes no Matlab/Simulink. Dentre as funções descritas têm-se: 41 funções para processamento digital de imagens (18 funções para pré-processamento, 17 funções para realce, 6 funções para classificação), 43 funções para visão computacional (5 funções de aquisição de vídeo, 16 funções para processamento de vídeo, 16 funções para análise de vídeo, 6 funções para saída de vídeo), 5 funções para fusão de dados, 5 funções para banco de dados e 5 funções para sistemas de informação geográfica.

Figueira (2016) propõe dois estudos de caso práticos a partir da modelagem de um MOSA específico, visando a geração automática em tempo real de mapas temáticos. O primeiro estudo de caso conduzido, baseia-se em uma aplicação civil de monitoramento ambiental, em que gravadores de áudio de uma Rede de Sensores Sonoros (SSN, do inglês *Sound Sensor Network*), localizados em solo, foram utilizados para registrar de forma inteligente os sons existentes na área monitorada. Os sons do ambiente foram obtidos através de sistemas gravadores digitais que armazenam o tempo e a intensidade dos sons detectados por microfones, permitindo a

reconstrução da distribuição de intensidade do sinal e da frequência ao longo do tempo pelo processador de sinais da SSN. Nesse experimento, os "nós"da SSN coletam sons e enviam, através de um rádio modem, dados pré-processados, quando o VANT sobrevoa a área na qual a rede está locada. A modelagem teve como principal objetivo definir um sistema capaz de auxiliar as atividades de monitoramento ambiental.

O segundo estudo de caso conduzido baseia-se em uma aplicação militar em que uma operação de busca e confirmação de alvos de um Pelotão de Cavalaria Mecanizado foi feita. Foi dado enfoque na identificação dos seguintes elementos da atividade militar: (i) Tropas: militares reunidos em formação de combate; (ii) Viaturas: veículos militares no terreno (blindados, carros, motos). (iii) Armas de fogo: acompanhamento ou inspeção de qualquer elemento que realize disparos (canhões, metralhadoras, pistolas). No contexto desse experimento, foram utilizadas imagens termais e amostras de sons referentes à área de estudo. A modelagem teve como principal objetivo obter um sistema capaz de realizar o reconhecimento de alvos, coletando, gerenciando e processando dados em tempo real. O quadricóptero Orbis foi utilizado como plataforma para transporte dos sensores do MOSA durante o experimento real.

Por fim, a autora Figueira (2016) destaca que a partir dos estudos de caso, a arquitetura apresentou, de forma consistente, características essenciais para a implementação de novos sistemas de sensoriamento incluindo modularidade, flexibilidade e adaptabilidade.

4.3 Diferenças entre as Implementações do MOSA

A seguir serão apresentadas algumas diferenças entre a implementação do MOSA proposto por Figueira (2016) e a implementação feita no presente doutorado.

Figueira (2016) dá um destaque maior nos aspectos específicos da missão, como aquisição de imagens e identificação de alvos. Ela não dá enfoque ao planejamento/replanejamento de rota. A presente tese dá um maior destaque ao sistema MOSA sobre este segundo aspecto. Toda missão demanda um planejamento ou replanejamento de rota em menor ou maior nível, o que atende ao aspecto mais geral do sistema MOSA proposto nesta tese.

Apesar de destacar a importância da separação de interesse (separação lógica e física) entre os sistemas MOSA e IFA, a autora não apresenta isso em seu trabalho. O presente trabalho fez apenas a separação lógica dos sistemas MOSA e IFA, todavia para que haja a separação física dos sistemas, poucas modificações têm que ser feitas.

Através dessa separação, os seguintes aspectos são melhorados: modularidade, reusabilidade, configurabilidade, flexibilidade, manutenibilidade, confiabilidade e segurança. Em contrapartida, os seguintes aspectos são prejudicados: desempenho, tempo de desenvolvimento, custo de desenvolvimento e de equipamentos.

Outra divergência entre os trabalhos é a comunicação do MOSA com o restante do

Quadro 4 – Comparação entre as implementações dos sistemas: MOSA Figueira x MOSA Arantes.

Categoria	Descrição	MOSA Figueira	MOSA Arantes
Características	Modelo de referência	✓	
	Implementação do MOSA	✓	✓
	Taxonomia de blocos de software	✓	
	Taxonomia de blocos de hardware	✓	
	Função para processamento de imagens	✓	
	Função para processamento de vídeo	✓	
	MOSA não adaptativo	✓	✓
	MOSA adaptativo		✓
	Integração do MOSA com IFA		✓
	Integração do AP com CC		✓
Áreas de aplicação	Monitoramento ambiental (mapas temáticos)	✓	
	Busca de alvos (mapas temáticos)	✓	
	Imageamento aéreo (fotos e vídeos)		✓
	Planejamento/replanejamento de rotas		✓
Algoritmos de planejamento	Estratégia FixedRoute4m		✓
	Algoritmo HGA4m		✓
	Algoritmo CCQSP4m		✓
Formas de planejamento	Missão pré-planejada	✓	✓
	Planejamento *offboard* (na GCS)		✓
	Planejamento *onboard* (no VANT)		✓
Forma de experimento	Experimento com VANT simulado	✓	✓
	Experimento com VANT real	✓	✓
Simulador utilizado	Simulador FlightGear		✓
Computadores avaliados	PC - Computador Pessoal	✓	✓
	CC - Intel Edison		✓
	CC - Raspberry Pi		✓
	CC - BeagleBone Black		✓
Pilotos automáticos avaliados	APM		✓
	Pixhawk		✓
Sist. operacionais avaliados	Linuxs		✓
	Windows	✓	✓
	Mac OS		✓
Escala do mapa	Avaliação em mapas médios	✓	
	Avaliação em mapas pequenos		✓

Fonte: Elaborada pelo autor.

VANT, pois a autora afirma que esta ocorre através da interface SSP/SSI. No entanto, o protocolo de comunicação SSP/SSI foi desenvolvido em Pires (2014) e não apresenta uma implementação funcional disponível. Dessa maneira, no sistema aqui proposto, a maioria das comunicações utiliza o protocolo HTTP *request* (através dos métodos GET e POST) e TCP (através de *socket*). As mensagens e as informações trocadas foram também definidas neste trabalho.

É importante ressaltar que, apesar de Figueira (2016) mencionar em vários pontos que a identificação de alvos é em tempo real, a implementação apresentada em seu trabalho não

o faz desse modo. Isso ocorre, pois o VANT, ao sobrevoar a região de interesse, captura o vídeo. A análise e a identificação dos alvos, no vídeo, ocorre posteriormente em laboratório a partir de um computador pessoal (PC, do inglês *Personal Computer*). A análise do vídeo é em tempo real, contudo foi feita em um PC. No entanto, tal processamento pode não conseguir ser executado em tempo real em um computador embarcado, como a Intel Edison, Raspberry Pi, BeagleBone Black, entre outros. Desse modo, é necessária a avaliação de tal processamento sobre tais plataformas embarcadas para averiguar a viabilidade de execução em tempo real dos sistemas.

Figueira (2016) destaca que seu sistema MOSA possui aspectos de modularidade, flexibilidade e adaptabilidade. Entretanto, o sistema se apresentou pouco modular ao ter que criar diferentes MOSAs para as diferentes missões; pouco flexível ao ter que reestruturar completamente a rota de voo quando se troca a missão; e pouco adaptável ao não fazer o recálculo de rota em voo e também ao não dar suporte em tempo real ao processamento embarcado a bordo do VANT. O sistema proposto em Figueira (2016) possui os três aspectos mencionados acima, já o sistema proposto nesta tese, preenche todas essas lacunas.

Apesar de Figueira (2016) apresentar uma boa base sobre o sistema MOSA, seu trabalho apresenta as seguintes limitações: os resultados, por se tratarem de aplicações bem específicas, são difíceis de se comparar; muitas diferenças no sistema estruturado e o implementado; poucas missões foram testadas; poucos experimentos conduzidos; e muitas das características teóricas descritas não foram respeitadas como sistema de tempo real e aquisição de dados em tempo real.

A presente tese visa desenvolver e embarcar o sistema descrito em Figueira (2016). O Quadro 4 sintetiza algumas diferenças substanciais entre o sistema MOSA proposto por Figueira (2016) (MOSA Figueira) e o sistema MOSA implementado nesta tese (MOSA Arantes).

4.4 Sistema IFA

Esta seção traz as ideias do sistema *In-Flight Awareness* (IFA), que foram, inicialmente, idealizadas, assim como o MOSA, pelo prof. Onofre Trindade Júnior e reportadas no trabalho de doutorado de Mattei (2015). Conforme ressaltado nesse trabalho, o conceito de IFA estabelece um modelo para a obtenção de consciência situacional em aeronaves não tripuladas dando-lhe maior autonomia. A partir de eventos ocorridos interna ou externamente à aeronave, o sistema verifica os sintomas, através dos dados dos sensores embarcados, ambientais e de tráfego aéreo e toma decisões levando em consideração as capacidades da aeronave. As origens dos eventos são identificadas para evitar ou mitigar acidentes, mediante um algoritmo de análise e decisão. O objetivo da abordagem é maximizar a segurança da plataforma aérea através da detecção de falhas, do diagnóstico, da capacidade de auto-recuperação e do replanejamento de rotas, considerando obstáculos, riscos e recompensas envolvidos no processo decisório (MATTEI, 2015).

O objetivo geral do trabalho de Mattei (2015) é criar um modelo de referência para o aumento da consciência situacional em voo, IFA, visando melhorar a segurança de voo de VANTs. Os objetivos específicos estabelecidos foram a validação dos conceitos em um ambiente de simulação com estudo de caso. Assim, foi criado um sistema IFA simulado e adaptado às limitações de uma aeronave real. Este sistema visa aumentar a segurança aérea e respeitar as capacidades da aeronave em termos de interface com seu AP e sistemas embarcados. O autor em Mattei (2015) ressalta que, embora o IFA considere a presença humana no *loop* de comando e reconheça a importância de manter a equipe de solo atualizada da situação do voo, seu foco é a automação e aumento da consciência e da inteligência da plataforma.

Na Figura 24, o autor Mattei (2015) traz um exemplo de um processo que, a partir de um evento iniciador, apresenta uma sequência de passos para a execução de ações que evitem ou mitiguem um acidente aéreo. Inicialmente, os eventos gerados pelo ambiente e pela própria aeronave são lidos através de diversos sensores. A seguir, os sintomas são identificados através de algoritmos específicos para tal. Em seguida, utilizando os sintomas identificados, algoritmos baseados em uma árvore de decisão podem calcular a gravidade de um determinado evento para a segurança de voo, então são definidas as ações e, finalmente, executam-se as ações necessárias.

Figura 24 – Processo sequencial para evitar ou mitigar acidentes com o VANT.

Fonte: Adaptada de Mattei (2015).

Mattei (2015) utilizou a metodologia *System Theoretic Process Analysis* (STPA) para aumentar da segurança de voo como alternativa às metodologias tradicionais *Failure Modes and Effect Analysis* (FMEA) e *Fault Tree Analysis* (FTA). O STPA aborda de modo mais completo a relação entre os diferentes elementos do sistema aéreo uma vez que inclui fatores humanos, de hardware, de software e de tomada de decisão. O STPA é uma metodologia de análise de perigos e risco, que investiga os possíveis acidentes de maneira sistêmica durante o desenvolvimento do projeto. Dessa forma, esta abordagem permite criar os requisitos para as etapas iniciais do desenvolvimento de um sistema (MATTEI, 2015).

Uma diferenciação entre o sistema IFA conceitual (modelo de referência) e a implementação do IFA, chamado de *In-Flight Awareness Augmentation System* (IFA^{2}S) é realizada em Mattei (2015). O IFA^{2}S é capaz de melhorar a segurança de voo através do modelo de referência IFA. O IFA foi desenvolvido para tornar VANTs, não só mais conscientes sobre si e sobre o mundo ao seu redor, mas também para torná-los capazes de realizar ações mitigadoras de maneira autônoma, visando evitar acidentes ou reduzir suas consequências.

A metodologia de projeto e a falta de um piloto dentro do avião podem explicar porque

os VANTs têm, hoje, as taxas de falha observadas nos primeiros anos da aviação mundial. Ao contrário de pilotos de aeronaves tripuladas, os pilotos dos VANTs não têm uma boa compreensão das condições do ambiente interno e externo e das regras para atendimento de aeronavegabilidade. Quando a bordo, os seres humanos têm a sua disposição seus cinco sentidos, que complementam os dados fornecidos pelos sensores da aeronave e contribuem diretamente no processo de controle dela (MATTEI, 2015).

O trabalho de Mattei (2015) traz uma ideia geral dos algoritmos que devem ser definidos para a integração do IFA^{2}S nas aeronaves. Uma vez que uma ameaça para a segurança de voo é identificada, boas decisões dependem do diagnóstico correto dos sintomas, tendo como base as informações de sensores, dados armazenados e fontes externas. Um fluxograma de decisão de alto nível é apresentado na Figura 25. Nesse fluxograma, é possível perceber que se uma situação de risco for identificada pelo sistema, será verificado se a missão pode continuar ou não. Se essa ameaça estiver nas proximidades da aeronave, uma ação urgente pode ser necessária. Esse é o caso de uma possível colisão no ar com outra aeronave, uma condição meteorológica perigosa ou um problema gerado em solo, por exemplo, fogo em uma floresta. Se não for o caso, o sistema IFA verifica se a aeronavegabilidade está comprometida ou não e, se possível, comanda o retorno da aeronave à base. Se a aeronavegabilidade estiver comprometida e se o voo puder continuar em uma configuração degradada, o IFA também comanda retorno para a base. Todas as outras possibilidades levam a um pouso de emergência controlado ou, até mesmo, a uma abertura do paraquedas. Como se pode notar, um sistema IFA^{2}S é definido não somente em função da configuração particular do VANT, mas também do seu perfil de missão.

Figura 25 – Fluxograma de decisão do sistema IFA^{2}S de alto nível.

Fonte: Mattei (2015).

Em seu trabalho, Mattei (2015) discute teoricamente a dimensão do VANT, os sensores que estarão a bordo e a missão que a aeronave executará. Ele define que utilizará um VANT com

menos de 25 kg, voltado ao sensoriamento remoto de áreas agrícolas. A aeronave voa em baixa altitude sobre uma área de operação bem delimitada. O autor ilustra que o sistema de comunicação pode monitorar a presença de outras aeronaves na área de operações através de protocolos específicos, como o ADS-B. Para monitorar o sistema propulsivo, além da tensão e corrente da bateria, será também verificada a sua temperatura. Detectores especiais de fumaça podem identificar problemas elétricos antes de ocorrência de fogo interno. Algoritmos especializados podem calcular a autonomia e comparar com a distância da base para alertar o piloto, seja humano ou eletrônico, sobre a necessidade de retorno automático. Acelerômetros e outros sensores podem verificar as vibrações e rotações existentes nos diversos eixos e alertar ao piloto sobre situações críticas ou presença de condições meteorológicas adversas.

Um dos objetivos do trabalho de Mattei (2015) é apresentar um ambiente de simulação em que diferentes situações operacionais possam ser testadas e soluções automatizadas possam ser criadas. Dessa forma, ele utilizou o simulador X-Plane por apresentar uma fácil integração com o LabVIEW, em que se executa o IFA^{2}S e o controle de navegação. A programação da máquina de estado do IFA^{2}S foi feita em LabVIEW e é composta por duas partes distintas: diagrama de blocos e interface. O diagrama de blocos define a lógica do IFA^{2}S. A interface estabelece a iteração com o usuário e o controle da simulação. O trabalho de Mattei (2015) orienta que a implementação do IFA^{2}S esteja integrada com o AP, no entanto, é preciso que o IFA esteja em hardware separado do AP, de modo que os conceitos dos sistemas fiquem separados (separação de interesse) e minimizem os erros de implementação do sistema.

A seguir serão descritos alguns aspectos incorporados pelo sistema IFA de Mattei (2015): (i) funcionalidade para tratar colisões em voo, em que aeronaves intrusas foram criadas e suas posições foram constantemente comparadas com as da aeronave que executa o IFA. Caso estas distâncias se tornem menores que um limiar, uma ação do IFA^{2}S pode ser efetuada. (ii) funcionalidade para nivelamento automático da aeronave, caso as superfícies de *roll* e de *picth* excedam um determinado limiar de ângulo. Se um limiar maior for ultrapassado, então uma ação de abortar o voo é executada e o paraquedas é aberto. (iii) funcionalidade para evitar o sobrevoo sobre zona proibida. (iv) funcionalidade do sistema para identificação de falhas internas na aeronave baseada em: tensão da bateria, temperatura da bateria e corrente no sistema aviônico. (v) funcionalidade para recuperação automática de altitude, caso a aeronave atinja um limite de altitude menor que um limiar, o modo de recuperação de altitude é iniciado e a aeronave sobe automaticamente. (vi) funcionalidade para pouso emergencial através do algoritmo MPGA4s. (vii) conjunto de ações definidas e avaliadas, como retorno à base, pouso de emergência e abertura de paraquedas. (viii) conjunto de situações meteorológicas diferentes criadas no simulador de voo e avaliadas como: variações sobre o turbulência, velocidade de vento e taxa de *climb*.

4.5 Diferenças entre as Implementações do IFA

Esta seção apresenta algumas diferenças entre a implementação do IFA proposto por Mattei (2015) e a implementação feita na presente tese.

Um dos pontos em que os trabalhos se diferem é com relação ao ambiente em que foram avaliados. O presente trabalho fez a avaliação sobre um VANT simulado e um VANT real, ao contrário de Mattei (2015), que fez o trabalho simulado considerando uma aeronave tripulada.

O presente trabalho apresenta apenas o sistema de comunicação ar-terra, já em Mattei (2015) é destacado, além da comunicação ar-terra, sistemas de comunicação ar-ar através da interação entre dois ou mais VANTs cooperando informações. O autor ilustra essa comunicação através do uso de equipamentos como o sistema ADS-B, no entanto um dispositivo como esse é muito caro, em geral, acima de US$2.000,00 [3], não sendo, então, viável em VANTs de baixo custo. Outro sistema de identificação de aeronaves deve ser pensado para aeronaves pequenas e de baixo custo.

No trabalho de Mattei (2015), a implementação do sistema IFA foi feita em LabVIEW, uma ferramenta da National Instruments de uso bastante comum na indústria, todavia uma série de fatores complicam a reutilização do código gerado nesse ambiente. Um dos fatores é que o LabVIEW é um ambiente pesado para ser instalado em um computador de bordo (necessitando de um processador equivalente ao Intel i5, 256 MB de RAM e 108 MB de armazenamento[4]), além de ser um software proprietário com licenças que variam de R$349,00/ano a R$28.080,00[5]. Existem versões para Linux e Mac OS apenas para planos mais caros, o que acaba por limitar o uso dessa ferramenta. Para sanar essas limitações, toda a implementação, desta tese de doutorado, foi feita em uma linguagem de programação que pudesse ser embarcada.

Apesar do trabalho de Mattei (2015) dar uma boa base sobre o sistema IFA, o mesmo apresenta as seguintes limitações: utilizou uma aeronave tripulada (simulado) e não um VANT; poucos mapas de missão foram testados; poucos experimentos foram conduzidos; e nenhum experimento real foi executado.

Conforme ressaltado por Mattei (2015, pág. 95), seu trabalho não definiu um sistema embarcado para aumentar a segurança de voo, somente criou as bases e os conceitos para que isso seja feito em um trabalho futuro. A presente tese visa justamente embarcar o sistema proposto por tal autor. O Quadro 5 sintetiza algumas diferenças substanciais entre os sistemas IFA propostos em Mattei (2015) (IFA Mattei) e o implementado nesta tese (IFA Arantes).

[3] <https://www.gulfcoastavionics.com/category/145-install-ads-b.aspx>
[4] <http://www.ni.com/white-paper/53740/en/>
[5] <http://www.ni.com/pt-br/shop/labview/select-edition.html>

4.6 Considerações Finais

Este capítulo apresentou os sistemas MOSA e IFA, propostos por Figueira (2016) e Mattei (2015) que serviram de base para a implementação efetuada nesta tese. Foi feita uma comparação destacando as diferenças entre os sistemas MOSA e IFA de tais autores e os propostos nesta tese.

O próximo capítulo apresentará o problema a ser abordado.

Quadro 5 – Comparação entre as implementações dos sistemas: IFA Mattei x IFA Arantes.

Categoria	Descrição	IFA Mattei	IFA Arantes
Características	Modelo de referência	✓	
	Implementação do IFA	✓	✓
	Detecta colisão com aeronaves no Ar	✓	
	Detecta colisão com solo	✓	
	Pouso emergencial em região atrativa	✓	✓
	Retorno à base	✓	✓
	Condições meteorológicas ruins ao voo	✓	✓
	Integração do IFA com MOSA		✓
	Integração do AP com CC		✓
	Avaliação em aeronaves tripuladas	✓	
	Avaliação em VANT		✓
Áreas de aplicação	Estudos de caso (sensoriamento remoto)	✓	
	Imageamento aéreo (fotos e vídeos)		✓
	Planejamento/replanejamento de rotas		✓
Algoritmos de replanejamento	Algoritmo MPGA4s	✓	✓
	Algoritmo GA4s		✓
	Algoritmo DE4s		✓
	Algoritmo GH4s		✓
	Algoritmo MS4s		✓
	Algoritmo GA-GA-4s		✓
	Algoritmo GA-GH-4s		✓
	Algoritmo Pre-Planned4s		✓
Formas de replanejamento	Replanejamento *offboard* (na GCS)	✓	✓
	Replanejamento *onboard* (no VANT)		✓
Forma de experimento	Experimento SITL	✓	✓
	Experimento HITL		✓
	Experimento com voo real		✓
Simulador utilizado	Simulador X-Plane	✓	
	Simulador FlightGear		✓
Computadores avaliados	PC - Computador Pessoal	✓	✓
	CC - Intel Edison		✓
	CC - Raspberry Pi		✓
	CC - BeagleBone Black		✓
Pilotos automáticos avaliados	APM		✓
	Pixhawk		✓
Sist. operacionais avaliados	Linuxs		✓
	Windows	✓	✓
	Mac OS		✓
Escala do mapa	Avaliação em mapas grandes	✓	
	Avaliação em mapas pequenos		✓

Fonte: Elaborada pelo autor.

CAPÍTULO

5

PROBLEMA ABORDADO

“ Para quê preocuparmo-nos com a morte? A vida tem tantos problemas que temos de resolver primeiro. ”

Confúcio

5.1 Considerações Iniciais

O problema abordado apresentará, inicialmente, uma descrição geral do cenário. Em seguida, as etapas envolvidas durante a realização da missão com segurança serão ilustradas através de um exemplo. Por fim, será apresentada a modelagem matemática efetuada para esse problema, seguida de dois modelos estocásticos, o primeiro para o planejamento e o segundo para o replanejamento de rotas.

5.2 Descrição Geral do Cenário

Antes de estabelecermos as ideias e conceitos por trás dos voos autônomos, vamos considerar um cenário como o apresentado na Figura 26a. Nesse cenário, podemos identificar um conjunto de elementos como: três regiões povoadas, um aeroporto comercial, uma montanha, um pico, duas florestas, três árvores, um lago e três pistas para VANTs. Após a identificação desses elementos no cenário, um mapeamento pode ser feito, como mostrado na Figura 26b. O mapeamento efetuado consiste basicamente no estabelecimento de um conjunto de polígonos convexos e suas respectivas classificações. Ainda na Figura 26b, podemos perceber que um dos povoados foi colocado sobre um polígono com altura ilimitada ($h = \infty$) e foi classificado como uma região não navegável, também chamada de Zonas de Exclusão Aérea ou *No-Fly Zones* (NFZ). As duas regiões povoadas mais acima no cenário foram colocadas sobre um mesmo polígono convexo a fim de obter uma maior simplificação no mapeamento. O aeroporto também foi considerado uma região não navegável. Todas as pistas para VANTs foram rotuladas como regiões bonificadoras. O lago foi considerado uma região penalizadora. Todas as florestas e

árvores foram consideradas regiões de obstáculos. O pico foi classificado também como um obstáculo. E por fim, a montanha foi rotulada como uma região cênica.

O mapeamento feito é importante para delimitar o espaço em que o VANT poderá sobrevoar, além de auxiliar nas etapas de tomadas de decisão a bordo da aeronave. Todas as regiões, exceto as NFZs, possuem altitude limitada, possibilitando o sobrevoo, desde que a altitude de voo cruzeiro seja superior a altura dos polígonos. Dado esse tratamento para o cenário do problema, o planejamento de trajetória passa a ser executado em uma região não convexa devido a presença de NFZs e obstáculos.

Figura 26 – Cenário geral do problema abordado.

(a) Cenário inicial. (b) Cenário mapeado.

Fonte: Elaborada pelo autor.

Nós consideramos neste trabalho que todo o mapa é conhecido, porém atualizações do mapa poderiam ser realizadas durante o voo via enlace de comunicação com um operador em solo, ou ainda por meio dos sensores da própria aeronave. A Figura 26b mostra algumas regiões que foram mapeadas e suas classificações/rótulos. A lista completa de todas as possíveis classificações aceitas são apresentadas a seguir e foram baseadas em Arantes (2016).

1. **Região Não Navegável (Φ^n):** A aeronave não pode sobrevoar e pousar nas regiões deste conjunto, conhecidas como NFZ. Por exemplo, aeroportos, cidades e bases militares.

2. **Região de Obstáculo (Φ^o):** A aeronave não pode sobrevoar e pousar nas regiões deste conjunto, exceto se estiver fazendo RTL. Por exemplo, florestas e montanhas.

3. **Região Penalizadora (Φ^p):** A aeronave pode sobrevoar regiões deste conjunto, mas não é desejável que ela pouse sobre elas. Por exemplo, lagos e rios.

4. **Região Bonificadora (Φ^b):** A aeronave pode sobrevoar e é desejado que ele pouse numa das regiões deste conjunto. Por exemplo, planícies e campos com plantações rasteiras.

5. **Região Cênica (Φ^s):** A aeronave deve sobrevoar a região deste conjunto, pois ela representa uma região de interesse. Por exemplo, florestas de interesse e objetos de interesse.

6. **Região Remanescente (Φ^r):** A aeronave pode sobrevoar e pousar nessas regiões. Este conjunto representa áreas restantes que não foram classificados para o pouso, sendo assim, não há restrições de voo ou pouso.

5.3 Problema de Planejamento de Missão e Segurança

Abordamos o problema de executar missões de forma autônoma com um VANT, em que a aeronave deve tomar decisões sobre a missão e garantir um voo seguro. A Figura 27 dá um exemplo do tipo de situação a ser tratada.

Figura 27 – Cenário ilustrativo do plano de missão gerenciado pelos sistemas IFA e MOSA.

(a) Exemplo-I: A rota inicial contendo o plano de voo.

(b) Exemplo-II: A rota após a atualização no plano de voo.

(c) Exemplo-III: A rota após uma falha e replanejamento.

Fonte: Elaborada pelo autor.

Plano da Missão: O VANT deve iniciar seu voo no ponto **A** (decolagem), prosseguir para os pontos-alvo **B**, **C** e concluir no ponto **D** (pouso). A sequência de pontos determina algumas regiões cênicas (**B** e **C**) a serem sobrevoadas pelo VANT que irá tirar fotografias de tais regiões como ilustrado na Figura 27a. O cenário é conhecido antecipadamente e conta com três zonas de exclusão aérea, NFZ, (aeroporto, cidade e base militar), um obstáculo (montanha), quatro regiões cênicas (florestas) e quatro regiões bonificadoras (pistas de VANT, áreas planas como campos de cultivo). As regiões bonificadoras representam locais seguros para pousar a aeronave, se ocorrer uma situação crítica. Na realização dessa missão, supomos um VANT quadricóptero com piloto automático Pixhawk, Raspberry Pi (*companion computer*), uma câmera específica e um *buzzer* de longo alcance. A altitude de voo cruzeiro é de 50 metros.

Sistema MOSA: O sistema gerenciador de missão planeja a rota com o objetivo de minimizar o consumo de combustível da aeronave entre os pontos-alvo **A** e **B**, e entre **B** e **C**. Em seguida, o sistema supervisiona a execução da rota pelo piloto automático, como ilustrado na Figura 27a e

o período correto para tirar fotos sobre cada região cênica. No entanto, as pessoas na estação de solo podem decidir incluir duas novas regiões cênicas quando o VANT estiver se movendo para o ponto-alvo **D**. O sistema de missão embarcado receberá os novos pontos de destino **D** e **E**, e ele irá planejar novamente a rota como mostra a Figura 27b.

Sistema IFA: Suponhamos agora que o VANT apresente uma falha como um superaquecimento da bateria. O sistema de consciência em voo detecta essa falha entre os pontos-alvo **C** e **D** e decide abortar a missão. O sistema de execução de missão é abortado e um replanejamento de caminho é executado para pousar com segurança a aeronave. Enquanto o sistema procura por uma nova rota de pouso de emergência, uma distância crítica é percorrida (d_c). A nova trajetória é, agora, supervisionada pelo sistema de consciência de voo, como ilustrado na Figura 27c, e deve pousá-la sobre uma região plana.

A partir do exemplo anterior da Figura 27, podemos estabelecer que estamos tratando um problema de execução de missão para VANTs em um ambiente estático e não-convexo. Assim, as regiões não navegáveis definem restrições não-convexas que devem ser satisfeitas durante a execução da missão. A aeronave deve sobrevoar as regiões cênicas, que são regiões de permanência, sendo possível pousar sobre as áreas planas, aqui chamadas regiões bonificadoras. As áreas planas podem ser utilizadas para pouso quando uma falha é detectada. Existem incertezas relacionadas a perturbações internas (dinâmica do VANT) e externas (turbulência). As incertezas relacionadas ao ambiente e à própria aeronave foram tratadas com base na abordagem de alocação de risco, conforme descrito em Blackmore, Ono e Williams (2011). Esses autores estudam o problema de planejamento de caminho não-convexo com restrições de probabilidade, nomeado em inglês *Chance-constraint Non-convex Path-planning Problem* (CNPP).

A rota geralmente pode ser definida utilizando alguma ferramenta de planejamento, como o Mission Planner ou o QGroundControl, e o piloto automático (AP) segue uma rota previamente planejada. Entretanto, como mencionado, o presente trabalho trata do problema de planejar e replanejar rotas a bordo. Isso é feito aplicando-se os planejadores descritos em Arantes *et al.* (2016), Arantes *et al.* (2017a), que levam em consideração a abordagem de alocação de risco para fazer o desvio de obstáculos sob incertezas a partir dos estudos descritos em Blackmore, Ono e Williams (2011).

Um VANT, no contexto do CNPP, deve alcançar a posição objetivo partindo da posição inicial, evitando regiões não navegáveis e pode sobrevoar florestas e montanhas. Todavia, as incertezas relacionadas com o ambiente e a dinâmica do VANT podem desviar a aeronave para alcançar uma região não navegável ou, até, atingir uma montanha. Portanto, um certo nível de risco é assumido através das *chance-constraints* no CNPP e o problema passa a ser encontrar um caminho através da otimização de uma medida, como a distância, sem exceder o nível de risco assumido.

Os autores em Blackmore, Ono e Williams (2011) descreveram esse problema utilizando um modelo de Programação Não Linear Inteira-Mista (PNLIM), porém meta-heurísticas que

resolvem modelos de Programação Linear Inteira-Mista (PLIM) foram propostas em Arantes (2017). Um algoritmo genético combinado com o grafo de visibilidade, nomeado de *Hybrid Genetic Algorithm for mission* (HGA4m), foi utilizado para resolver esse problema em Arantes *et al.* (2016). A Figura 27a mostra um cenário, semelhante aos trabalhos de Blackmore, Ono e Williams (2011), Arantes *et al.* (2016), no qual o problema de planejamento de missão foi abordado.

A Figura 27c ilustra um cenário em que o problema de replanejamento de rota ocorre a partir da detecção de uma situação crítica. Em Arantes *et al.* (2015), o replanejamento de rota é definido com a mesma *chance-constraint* e dentro de um ambiente não convexo de Blackmore, Ono e Williams (2011), Arantes *et al.* (2016). Situações críticas são consideradas em Arantes *et al.* (2015), uma vez que podem impactar na capacidade de voo da aeronave (dinâmica do VANT). Um método chamado *Multi-Population Genetic Algorithm for security* (MPGA4s) foi aplicado para encontrar uma trajetória de pouso emergencial. De maneira geral, o MPGA4s busca encontrar uma trajetória que pouse a aeronave sobre regiões bonificadores (por exemplo, planícies) minimizando o impacto contra construções e pessoas no solo.

A fim de conseguir executar a missão com segurança e dar à aeronave um alto grau de autonomia, precisamos incorporar na aeronave um computador de bordo. Este *companion computers* (CC) executará o planejamento de missão e replanejamento de rotas *onboard* (*online*), utilizando técnicas de Inteligência Artificial (IA) para evitar colisões com obstáculos. Existem vários CC utilizados na literatura como Intel Edison, Raspberry Pi (RPi), BeagleBone, Odroid XU4, entre outros. Este trabalho utiliza as placas Intel Edison, Raspberry Pi 3 e BeagleBone Black (BBB) para fazer o processamento embarcado.

As falhas das aeronaves ou situações de ameaças também são enfrentadas durante o voo autônomo. Nós chamamos de situação crítica qualquer falha interna ou externa ao VANT que coloque em risco a sua aeronavegabilidade. Durante uma situação crítica, uma ação de tomada de decisão deve ser feita pelo sistema proposto. Assim, situações críticas foram intencionalmente inseridas para avaliar o comportamento e robustez do sistema. Uma falha no motor, superaquecimento da bateria, baixo nível de bateria ou baixo nível de combustível são exemplos de possíveis situações críticas. Para alguns desses problemas, já existem soluções simples, como fazer um retorno ao local de lançamento (RTL) ou disparar o paraquedas. No entanto, o combustível pode não ser suficiente para o RTL, ou o pouso com paraquedas pode danificar a aeronave. Nesse caso, se possível, um pouso de emergência pode ser executado para minimizar as chances de danos. O sistema IFA desenvolvido é o responsável por fazer essas tarefas que lidam com os aspectos críticos da aeronave.

As entradas para o nosso sistema podem ser fornecidas pelo projetista da missão. Ele deve descrever o local de lançamento da aeronave, os pontos-alvo da missão (*waypoints*), os possíveis locais de pouso, os obstáculos e as NFZ. Os obstáculos já são previamente conhecidos, o que evita o processamento de imagens a bordo. Mesmo assim, um sistema de processamento de

imagens que coleta informações do ambiente pode ser adicionado à nossa arquitetura. Todas as tomadas de decisão, como, por exemplo, decidir qual caminho percorrer para atingir os objetivos da missão fazendo o desvio de obstáculos são feitas de maneira autônoma. O sistema IFA é capaz de garantir que a aeronave durante o voo tome decisões rápidas em caso de emergência, na ocorrência uma falha crítica.

5.4 Modelagem do Problema

Nós podemos modelar o problema acima descrito da seguinte forma:

Definição 1: (Problema de Navegação Autônomo). Dado a ênupla $\langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle$, o sistema deve encontrar uma solução $\mathcal{S}^*$, em que:

- $\mathcal{M} = \langle M_{map}, M_{goal}, M_{alt} \rangle$ é o mapa, o plano da missão e a altitude de voo cruzeiro a ser seguida pelo VANT;
- $\mathcal{H} = \langle H_{ap}, H_{cc}, H_{sensor}, H_{actuator} \rangle$ é a configuração da arquitetura de hardware para o problema;
- $\mathcal{S} = \langle S_{mosa}, S_{ifa} \rangle$ é a configuração da arquitetura de software para o problema;
- $\mathcal{S}^*$ é a solução ótima ou próxima do ótimo encontrada para o problema.

Definição 2: (Missão). $\mathcal{M} = \langle M_{map}, M_{goal}, M_{alt} \rangle$, em que:

- $M_{map} = \langle \mathbf{r}, \mathbf{h}, \mathbf{c} \rangle$ é o mapa da missão, em que:
  - $\mathbf{r} = \{r^1, r^2, \cdots\}$ representa um conjunto de regiões, em que r é uma região convexa;
  - $\mathbf{h} = \{h^1, h^2, \cdots\}$ representa a altura de cada uma das regiões;
  - $\mathbf{c} = \{c^1, c^2, \cdots\}$ representa a sua classificação, em que $c \in \{\Phi^n, \Phi^o, \Phi^p, \Phi^b, \Phi^s, \Phi^r\}$.
- $M_{goal} = \langle w^1, w^2, \cdots \rangle$ é o conjunto de pontos-alvo da missão a serem alcançados pelo VANT, em que $w = (p_x, p_y)$;
- M_{alt} é a altura de voo cruzeiro do VANT durante a execução da missão.

Definição 3: (Arquitetura de Hardware). $\mathcal{H} = \langle H_{ap}, H_{cc}, H_{sensor}, H_{actuator} \rangle$, em que:

- H_{ap} é o piloto automático (AP) utilizado durante a missão. Existem dois APs suportados atualmente: $H_{ap} \in \{APM, Pixhawk\}$;
- H_{cc} é o *companion computer* (CC) embarcado no VANT. Existem três CC suportados atualmente: $H_{cc} \in \{IntelEdison, RaspberryPi, BeagleBoneBlack\}$;

- H_{sensor} é uma lista com sensores específicos instalados na aeronave para realização da missão. Os seguintes sensores são atualmente suportados: $H_{sensor} = [CameraRGB, Sonar, Temperature]$;

- $H_{actuator}$ é uma lista com atuadores específicos instalados na aeronave para realização da missão. Os seguintes atuadores são atualmente suportados: $H_{actuator} = [Buzzer, Spraying, Parachute, LED]$.

Definição 4: (Arquitetura de Software). $\mathcal{S} = \langle S_{mosa}, S_{ifa} \rangle$, em que:

- $S_{mosa} = \langle \mathcal{P}_P, \mathcal{L}_P, \mathcal{T}_{SC_P}, \Delta_P \rangle$ é o sistema MOSA, em que:
  - $\mathcal{P}_P$ representa o planejador de rotas empregado. Existem três planejadores atualmente disponíveis: $\mathcal{P}_P \in \{HGA4m, CCQSP4m, FixedRoute4m\}$;
  - $\mathcal{L}_P$ é o local de execução do método de planejamento, em que $\mathcal{L}_P \in \{onboard, offboard\}$;
  - $\mathcal{T}_{SC_P}$ tem o tempo disponível (em segundos) para aguardar uma rota do planejador de caminho (critério de parada do método);
  - Δ_P é o nível de risco assumido pelo planejador de rotas.

- $S_{ifa} = \langle \mathcal{P}_R, \mathcal{L}_R, \mathcal{T}_{SC_R}, \Delta_R, \mathcal{W}_R \rangle$ é o sistema IFA, em que:
  - $\mathcal{P}_R$ representa o replanejador de rotas empregado. Existem oito replanejadores atualmente suportados: $\mathcal{P}_R \in \{GA4s, MPGA4s, DE4s, GH4s, MS4s, GA\text{-}GA\text{-}4s, GA\text{-}GH\text{-}4s, Pre\text{-}Planned4s\}$;
  - $\mathcal{L}_R$ é o local de execução do método de replanejamento, em que $\mathcal{L}_R \in \{onboard, offboard\}$;
  - $\mathcal{T}_{SC_R}$ tem o tempo disponível (em segundos) para aguardar uma rota do replanejador de caminho (critério de parada do método);
  - Δ_R é o nível de risco assumido pelo replanejador de rotas;
  - $\mathcal{W}_R$ representa o número máximo de *waypoints* que pode ser utilizado pelo replanejador de rotas.

Por exemplo, baseado no problema descrito na Figura 27a, o mapa da missão $M_{map} = \langle \mathbf{r}, \mathbf{h}, \mathbf{c} \rangle$ tem as seguintes regiões:

$$\mathbf{r} = \{pistavant_1, pistavant_2, planicie_1, planicie_2, aeroporto, cidade, basemilitar, montanhas, floresta_1, floresta_2, floresta_3, floresta_4\}$$

as alturas das regiões são:

$$\mathbf{h} = \{0,0,0,0,\infty,\infty,\infty,20,10,10,10,10\},$$

e as classificações/rótulos das regiões são:

$$\mathbf{c} = \{\Phi^b,\Phi^b,\Phi^b,\Phi^b,\Phi^n,\Phi^n,\Phi^n,\Phi^o,\Phi^s,\Phi^s,\Phi^s,\Phi^s\}$$

Os *waypoints* da missão são $M_{goal} = \{A,B,C,D\}$ e a altitude de cruzeiro é $M_{alt} = 50,0$ metros. Os seguintes equipamentos de hardware foram utilizados: $H_{ap} = Pixhawk$, $H_{cc} = RaspberryPi$, $H_{sensor} = [CameraRGB]$ e $H_{actuator} = [Buzzer]$. Vamos supor também que as seguintes configurações de software sejam definidas pelo usuário: $\mathcal{P}_P = HGA4m$, $\mathcal{P}_R = GA4s$, $\mathcal{L}_P = offboard$, $\mathcal{L}_R = onboard$, $\mathcal{T}_{SC_P} = 10$ segundos, $\mathcal{T}_{SC_R} = 1$ segundo, $\Delta_P = \Delta_R = 1.0\%$ e $\mathcal{W}_R = 30$ *waypoints*. O problema na Figura 27a pode ser resumido através da Equação 5.1 apresentada a seguir:

$$Ex.I \therefore \langle \mathcal{M},\mathcal{H},\mathcal{S}\rangle = \begin{cases} \mathcal{M} = \begin{cases} M_{map} = \langle \mathbf{r},\mathbf{h},\mathbf{c}\rangle = \begin{cases} \mathbf{r} = \{pistavant_1, pistavant_2, planicie_1, planicie_2, \\ \quad aeroporto, cidade, basemilitar, montanhas, \\ \quad floresta_1, floresta_2, floresta_3, floresta_4\} \\ \mathbf{h} = \{0,0,0,0,\infty,\infty,\infty,20,10,10,10,10\} \\ \mathbf{c} = \{\Phi^b,\Phi^b,\Phi^b,\Phi^b,\Phi^n,\Phi^n,\Phi^n,\Phi^o,\Phi^s,\Phi^s,\Phi^s,\Phi^s\} \end{cases} \\ M_{goal} = \{A,B,C,D\} \\ M_{alt} = 50.0 \end{cases} \\ \mathcal{H} = \begin{cases} H_{ap} = Pixhawk \\ H_{cc} = RaspberryPi \\ H_{sensor} = [CameraRGB] \\ H_{actuator} = [Buzzer] \end{cases} \\ \mathcal{S} = \begin{cases} S_{mosa} = \langle \mathcal{P}_P, \mathcal{L}_P, \mathcal{T}_{SC_P}, \Delta_P\rangle = \langle HGA4m, offboard, 10, 1.0\%\rangle \\ S_{ifa} = \langle \mathcal{P}_R, \mathcal{L}_R, \mathcal{T}_{SC_R}, \Delta_R, \mathcal{W}_R\rangle = \langle GA4s, onboard, 1.0, 1.0\%, 30\rangle \end{cases} \end{cases} \tag{5.1}$$

Os aspectos mencionados acima serão devidamente descritos por duas formulações matemáticas estocásticas. A primeira formulação, introduzida por Blackmore, Ono e Williams (2011) e utilizada em Arantes *et al.* (2016), resolve o problema de planejamento de rota ($\mathcal{P}_P$).

$$Minimizar \quad \sum_t \mathbf{u}_t \cdot \mathbf{u}_t \tag{5.2}$$

sujeito a:

$$\mathbf{x}_T = \mathbf{x}_{goal} \tag{5.3}$$

$$\mathbf{x}_{t+1} = \mathbb{A}\mathbf{x}_t + \mathbb{B}\mathbf{u}_t + \omega_t \qquad \forall(t) \tag{5.4}$$

$$\mathbf{x}_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_{x_0}), \quad \omega_t \sim \mathcal{N}(0, \Sigma_{\omega_t}) \qquad \forall(t) \tag{5.5}$$

$$Pr\left[\bigwedge_j \bigwedge_t \mathbf{x}_t \notin \mathbb{O}_j^n\right] \geq 1 - \Delta_P \tag{5.6}$$

No modelo acima, o vetor de estado $\mathbf{x}_t$ tem as posições do veículo p e velocidades v, ambos no eixo (x,y), enquanto que o vetor de controle $\mathbf{u}_t$ possui aceleração, no eixo (x,y), definida pelas expressões 5.7.

$$\mathbf{x}_t := [p_x \; p_y \; v_x \; v_y]^T, \quad \mathbf{u}_t := [a_x \; a_y]^T \tag{5.7}$$

As entradas incluem a posição inicial $\hat{x}_0$, a posição objetivo $\hat{x}_{goal}$, as matrizes de controle da dinâmica do VANT $\mathbb{A}$ e $\mathbb{B}$ (Equação 5.8), o horizonte de planejamento $t = 0, 1, \cdots T$, o intervalo de tempo Δt, as incertezas externas ω_t e o nível de risco assumido no planejador Δ_P. As variáveis de decisão são os estados do VANT $\mathbf{x}_t$ e os controles $\mathbf{u}_t$ aplicados em cada instante de tempo t. A Função Objetivo 5.2 é o produto escalar dos controles que é minimizado durante o planejamento da missão. Essa medida pode estar relacionada ao consumo de combustível ou a distância percorrida. O estado do veículo $\mathbf{x}_T$ deve atingir a posição objetivo $\mathbf{x}_{goal}$ no final dessa missão através da Restrição 5.3. A transição de estados do veículo está descrita na Restrição 5.4, em que o primeiro estado $\mathbf{x}_0$ segue uma distribuição Gaussiana do estado inicial esperado $\hat{x}_0$ com matriz de covariância Σ_{x_0}, mostrada na Equação 5.9. Existe um ruído Gaussiano branco aditivo ω_t com matriz de covariância Σ_{ω_t}, expressa na Equação 5.9 aplicado a cada transição de estado. A Restrição 5.6 define que os estados do veículo $\mathbf{x}_t$ devem estar fora de todos os obstáculos $\mathbb{O}_j^n$ em todo tempo t. A expressão $Pr[\cdot] \geq 1 - \Delta_P$ reporta que a probabilidade do veículo estar fora de todos os obstáculos deve ser maior ou igual a $1 - \Delta_P$. Assim, a probabilidade de falha deve ser menor que Δ_P.

$$\mathbb{A} = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbb{B} = \begin{pmatrix} \Delta t^2/2 & 0 \\ 0 & \Delta t^2/2 \\ \Delta t & 0 \\ 0 & \Delta t \end{pmatrix} \tag{5.8}$$

$$\Sigma_{x_0} = \begin{pmatrix} \sigma_A^2 & 0 & 0 & 0 \\ 0 & \sigma_A^2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad \Sigma_{\omega_t} = \begin{pmatrix} \sigma_B^2 & 0 & 0 & 0 \\ 0 & \sigma_B^2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{5.9}$$

A seguir é descrita uma formulação matemática estocástica, introduzida por Arantes *et al.* (2015), que resolve o problema de replanejamento de rota ($\mathcal{P}_R$) sobre situações críticas.

$$Minimizar\ Pr\Big(\bigvee_j \mathbf{x}_T \in \mathbb{O}_j^p\Big) - Pr\Big(\bigvee_j \mathbf{x}_T \in \mathbb{O}_j^b\Big) \tag{5.10}$$

sujeito a:

$$\mathbf{x}_{t+1} = F_\Psi(\mathbf{x}_t, \mathbf{u}_t) + \omega_t \qquad \forall(t) \tag{5.11}$$

$$\mathbf{x}_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_{x_0}), \quad \omega_t \sim \mathcal{N}(0, \Sigma_{\omega_t}) \qquad \forall(t) \tag{5.12}$$

$$Pr\left[\bigwedge_j \bigwedge_t \mathbf{x}_t \notin \mathbb{O}_j^n\right] \geq 1 - \Delta_R \tag{5.13}$$

$$\mathbf{x}_{t+1} = F_\Psi(\mathbf{x}_t, \mathbf{u}_t) \Leftrightarrow \begin{bmatrix} p_{t+1}^x \\ p_{t+1}^y \\ v_{t+1} \\ \alpha_{t+1} \end{bmatrix} = \begin{bmatrix} p_t^x + v_t \cdot cos(\alpha_t) \cdot \Delta t + a_t \cdot cos(\alpha_t) \cdot (\Delta t)^2/2 \\ p_t^y + v_t \cdot sen(\alpha_t) \cdot \Delta t + a_t \cdot sen(\alpha_t) \cdot (\Delta t)^2/2 \\ v_t + a_t \cdot \Delta t - \frac{F_D}{m} \cdot \Delta t \\ \alpha_t + \varepsilon_t \cdot \Delta t \end{bmatrix} \tag{5.14}$$

No modelo acima, o vetor de estado $\mathbf{x}_t$ tem as posições do veículo p (no eixo (x,y)), velocidades horizontal v e ângulo α, enquanto que o vetor de controle $\mathbf{u}_t$ possui aceleração a e variação angular ε, como definido pelas expressões 5.15.

$$\mathbf{x}_t := [p_x\ p_y\ v\ \alpha]^T, \quad \mathbf{u}_t := [a\ \varepsilon]^T \tag{5.15}$$

As entradas incluem a posição inicial $\hat{x}_0$ no momento da falha, a função de transição de estados não linear F_Ψ, o horizonte de planejamento $t = 0, 1, \cdots T$, o intervalo de tempo Δt, as incertezas externas ω_t e o nível de risco assumido no replanejador Δ_R. As variáveis de decisão são os estados do VANT $\mathbf{x}_t$ e os controles $\mathbf{u}_t$ aplicados em cada instante de tempo t. A Função Objetivo 5.10 define punições sobre a chance de pousar a aeronave sobre uma região penalizadora $\mathbb{O}_j^p$ e recompensas sobre a chance de pousar sobre regiões bonificadoras $\mathbb{O}_j^b$. A função de transição dos estados do veículo está descrita pela Restrição 5.11 e pela Equação 5.14, em que o primeiro estado $\mathbf{x}_0$ segue uma distribuição Gaussiana do estado esperado $\hat{x}_0$ com matriz de covariância Σ_{x_0}. O valor F_D modela a força de arrasto associada à resistência do ar. Um ruído Gaussiano branco ω_t com matriz de covariância Σ_{ω_t} é aplicado a cada transição. A Restrição 5.13 define que os estados do veículo devem estar fora dos obstáculos $\mathbb{O}_j^n$ em todo tempo t.

Acima foram apresentados dois modelos estocásticos. O primeiro mostrou uma modelagem que calcula rota entre dois pontos **A** e **B**, minimizando o consumo de combustível. O

método HGA4m possui uma modelagem semelhante a apresentada. O segundo apresentou uma modelagem que retorna uma rota emergencial, minimizando as chances de colidir com construções, buscando pousar sobre regiões atrativas. Os métodos GA4s, MPGA4s e DE4s possuem uma modelagem semelhante a essa. A Tabela 9 mostra uma representação dos problemas de planejamento e replanejamento de rotas. Ambos os problemas são resolvidos por técnicas de IA baseadas em computação evolutiva. Os genes em cada um dos modelos é obtido pelo controle $\mathbf{u}_t$ aplicado. O genótipo é a representação matemática da solução e o fenótipo é a solução encontrada no mundo real para o problema.

Tabela 9 – Resumo da representação do problema de planejamento/replanejamento de rotas.

Problema	Gene	Genótipo	Fenótipo	Função de *Fitness*
Planejamento de Rotas	Acelerações desejadas ($\mathbf{u}_t := [a_x\ a_y]^T$)	Vetor de *waypoints*	Rota da missão	Minimiza o gasto de combustível da aeronave
Replanejamento de Rotas	Aceleração e variação angular desejadas ($\mathbf{u}_t := [a\ \varepsilon]^T$)	Vetor de *waypoints*	Rota de pouso emergencial	Maximiza a chance de pousar em uma região bonificadora

Fonte: Elaborada pelo autor.

Ambos os modelos de planejador e replanejador calculam a rota em um espaço bidimensional (2D). Os pontos-alvo M_{goal} são especificados em um espaço 2D. Toda a missão é realizada em uma altitude de cruzeiro especificada pelo parâmetro M_{alt}. Dessa forma, a presente modelagem deve utilizar uma aeronave do tipo *Vertical Take-off and Landing* (VTOL), como, por exemplo, um quadricóptero. Aeronaves do tipo *Conventional Take-off and Landing* (CTOL), como o Ararinha, para serem suportadas na plataforma autônoma, necessitam de algumas alterações relacionadas ao momento do pouso e decolagem.

5.5 Considerações Finais

Este capítulo descreveu o cenário do problema abordado em maiores detalhes, em que um exemplo de planejamento de missão com situação crítica foi descrito e foram definidos os conjuntos de regiões mapeadas. A formalização do problema foi feita através da modelagem matemática apresentada. Assim, a partir do problema de execução da missão com segurança, descrito neste capítulo, uma arquitetura de hardware e software será elaborada visando fornecer à aeronave uma maior autonomia para tomadas de decisão.

CAPÍTULO

6

ARQUITETURA DE HARDWARE E SOFTWARE

" A imaginação é mais importante que a ciência, porque a ciência é limitada, ao passo que a imaginação abrange o mundo inteiro. "

Albert Einstein

6.1 Considerações Iniciais

O presente capítulo apresentará a arquitetura de hardware e software elaborada, além de destacar os sistemas de comunicação e a construção da arquitetura de baixo custo.

A Figura 28 dá uma visão geral da arquitetura, apresentando os sete principais componentes de hardware: piloto automático, atuadores, sensores, comunicação, alimentação, computador embarcado e computador em solo. O piloto automático (AP) é responsável pela navegação, mantendo a aeronave na rota planejada. Os atuadores são os mecanismos que permitem o deslocamento e a execução das manobras do VANT. Os sensores são responsáveis pela captura de informações do ambiente e pelo auxílio à navegação, missão e segurança. A comunicação é responsável por receber comandos do controle de rádio e enviar dados através do módulo de telemetria. A alimentação é responsável por fornecer energia para todo o sistema de hardware. O computador embarcado é incorporado e hospeda os sistemas que fornecem o nível desejado de autonomia: MOSA, IFA e S2DK. Finalmente, tem-se o computador em solo, em que o sistema UAV-GCS é responsável por enviar todas as atualizações para o CC. Todavia, ele deve funcionar mais como um sistema de suporte ao invés de um sistema de tomada de decisão.

As próximas seções detalharão toda a arquitetura.

Figura 28 – Visão geral da arquitetura proposta.

Fonte: Elaborada pelo autor.

6.2 Arquitetura de Hardware do VANT

O primeiro passo deste trabalho foi construir o VANT, chamado iDroneAlpha, apresentado na Figura 29a com os seguintes componentes: (a) *Companion Computer* (CC) Intel Edison; (b) piloto automático APM v2.8; (c) conjunto de motores (JMT D2212 920kV), ESC (Simonk 30A) e hélice (9x47); (d) módulo de GPS com bússola (Ublox NEO-6M); (e) receptor do controle de rádio (Flysky FS-iA6), telemetria; (f) bateria (Venom 3S 2200mAh 11.1V LiPo), *power module*; (g) controle de rádio (Flysky FS-i6).

Um voo controlado por rádio ou pré-planejado com os equipamentos descritos nos itens (b) a (g) pode ser executado. Com a inclusão do item (a), torna-se possível incorporar os sistemas de execução de missão e de segurança que forneçam o nível de autonomia desejado. Na Figura 29b, é apresentado o VANT iDroneBeta também construído e utilizado nos experimentos. Durante esse doutorado, outros dois VANTs foram construídos no Laboratório de Computação Reconfigurável (LCR), no ICMC/USP. O tempo de voo suportado pelo iDroneAlpha é de cerca de 7 minutos e suas dimensões são 55cm x 57 cm x 32 cm. Algumas informações dos VANTs

Figura 29 – Hardware dos VANTs utilizados nos experimentos.

(a) iDroneAlpha. (b) iDroneBeta.

Fonte: Elaborada pelo autor.

iDroneAlpha e iDroneBeta estão sintetizadas na Tabela 1.

6.3 Arquitetura de Software do VANT

A arquitetura de software do VANT é projetada para separar a execução da missão (MOSA) do sistema de gerenciamento de segurança/monitoramento da aeronave (IFA). Há também o sistema *Services to DroneKit* (S2DK), que inclui um conjunto de serviços desenvolvidos sobre a API do Dronekit. Por fim, tem-se o sistema UAV-GCS, que é uma estação de controle de solo desenvolvida para acompanhar e atualizar a missão. A Figura 30 mostra os quatro principais sistemas da arquitetura proposta.

Dentro dos sistemas MOSA e IFA, um conjunto de módulos foi implementado a fim de garantir o cumprimento efetivo da missão, levando em conta aspectos relacionados à segurança. Nesse contexto, MOSA e IFA são sistemas compostos por alguns módulos em comum: Aquisição de Dados, Comunicação, (Re)Planejamento de Rota e Tomada de Decisão.

A aquisição de dados solicita informações dos sensores da aeronave através do protocolo HTTP *request*. O módulo de comunicação permite que o MOSA e o IFA conversem entre si, bem como com a estação de controle de solo (GCS). O planejamento de rota possui os algoritmos responsáveis por definir as trajetórias. Depois que os módulos planejamento de rota ou replanejamento de rota retornam uma rota, ela é convertida para o formato *JavaScript Object Notation* (JSON) e enviada para a aeronave por meio de mensagens POST. Essas mensagens são convertidas em comandos MAVLink e enviadas para o AP, que passa a seguir a rota estabelecida.

A tomada de decisão executa diferentes tarefas em relação ao MOSA e ao IFA. Esse módulo é capaz de iniciar outro módulo externo ao MOSA, por exemplo, atualizar a trajetória

Figura 30 – Módulos e comunicação entre os sistemas MOSA, IFA, S2DK e UAV-GCS.

Fonte: Elaborada pelo autor.

após novos pontos-alvo terem sido definidos a partir da GCS. No IFA, o módulo de Tomada de Decisão executa ações apenas em relação à situações críticas. A título de exemplo, esse módulo pode fazer um replanejamento de caminho para pouso de emergência, ou realizar um *Return To Launch* (RTL), ou executar um pouso na vertical, ou ainda, acionar o paraquedas. Os módulos de planejamento de rota e replanejamento de rota são os que consomem mais processamento nessa arquitetura embarcada, uma vez que tais problemas são considerados de difícil solução por serem classificados como não convexos e não lineares. O Capítulo 7 apresentará todos os planejadores e replanejadores utilizados.

Todos os planejadores e replanejadores utilizados fazem uso de alocação de risco, como estabelecido em Blackmore, Ono e Williams (2011), Ono, Williams e Blackmore (2013). Em um estudo feito por Arantes *et al.* (2017a), estimou-se que o computador embarcado Intel Edison é da ordem de oito vezes mais lento que um computador PC tradicional. Dessa forma, foi desenvolvido o suporte para execução desses planejadores tanto *onboard* quanto *offboard*, a fim de acelerar os cálculos. Quando deseja-se fazer o cálculo *offboard*, uma requisição é enviada para o UAV-GCS com todos os parâmetros e dados necessários para que o planejador encontre a rota. O computador em solo faz o cálculo/processamento enviando a resposta/rota à aeronave. Os módulos Planeja Rota *Offboard* e Replaneja Rota *Offboard*, na Figura 30, são os responsáveis por enviar tais requisições para o computador de solo.

O MOSA possui o módulo Controle da Missão que supervisiona as tarefas relacionadas à execução da missão. A Figura 31 ilustra como esse módulo cuida da missão. Primeiro, o sistema é iniciado, criando alguns arquivos de *log* e lendo as entradas sobre a missão. Em seguida, quatro *threads* são executadas em paralelo. A primeira *thread* é responsável pela comunicação

com o IFA; a segunda envia dados à GCS; a terceira supervisiona a execução da missão; e a quarta executa ações, como o planejamento de uma nova rota. Depois de completar a missão, o sistema pousa a aeronave e desarma os motores, finalizando o fluxo de execução. Durante toda a execução da missão, esse sistema é capaz de fazer a retirada de fotografias, gravar vídeos, ou mesmo, acionar o *buzzer* após algum evento de interesse ocorrer.

Figura 31 – Fluxo de execução para o módulo de controle da missão no MOSA.

Fonte: Elaborada pelo autor.

O Controle de Segurança no IFA gerencia todos os subsistemas, como mostra a Figura 32. Nesse caso, a criação do arquivo de *log* é seguida pela calibração dos parâmetros do AP. Posteriormente, quatro *threads* também são ativadas: a primeira *thread* se comunica com o MOSA, a segunda se comunica com a GCS, a terceira supervisiona a segurança e a operação da aeronave e a quarta envia dados para a GCS. Se alguma situação crítica for detectada, o IFA assume o controle e uma ação relacionada à segurança da aeronave é executada. Por exemplo, se um pouso de emergência for exigido, o replanejador é chamado, a trajetória é definida e os *waypoints* são enviados para o AP. Nesta situação, o VANT será pousado pelo IFA, os motores serão desarmados e o sistema será desligado.

Com base no fluxo de execução dos sistemas MOSA e IFA das figuras 31 e 32, podemos definir um conjunto de ações que esses sistemas devem executar para cumprir a missão. A Figura 33 sintetiza, através de um fluxograma, o conjunto de ações nas quais o MOSA realiza a calibração dos sistemas e executa o planejamento de rota. Em seguida, a aeronave decola e sobe até alcançar a altitude de cruzeiro, segue os *waypoints* da missão e, finalmente, pousa. Em paralelo com o MOSA, o sistema IFA também executa a calibração do sistema (por exemplo, alterar alguns parâmetros do AP, como altitude do RTL) e monitora os sensores verificando possíveis situações críticas. No caso de acontecer alguma falha enquanto a aeronave estiver voando, uma nova rota é calculada, o alarme é disparado e a aeronave pousa, abortando a missão atual.

Figura 32 – Fluxo de execução para o módulo do gerenciador de segurança no IFA.

Fonte: Elaborada pelo autor.

O sistema IFA incorpora alguns recursos essenciais, caso alguma falha crítica ocorra, como: capacidade do sistema reagir em tempo real e fazer o recálculo de rota trazendo a aeronave até o solo com segurança; capacidade de reagir de maneira distinta a diferentes problemas ocorridos com ações, como executar um novo planejamento de caminho, fazer um RTL, um pouso na vertical ou disparar um alarme sonoro a fim de alertar as pessoas que se encontrarem nas proximidades.

Tanto o IFA quanto o MOSA podem acessar os dados da aeronave através do sistema *Services to DroneKit* (S2DK), mostrado na Figura 34. O S2DK é uma aplicação desenvolvida que utiliza o Dronekit como API. O Dronekit é uma biblioteca, que fornece recursos que facilitam o controle do VANT [1]. O Dronekit traduz os comandos desejados para o protocolo MAVLink que é entendido pelo AP. Os serviços implementados no S2DK respondem a solicitações HTTP *request* por meio dos métodos GET e POST. Esse tipo de comunicação ocorre em apenas uma direção. No entanto, ao contrário da comunicação *socket*, a aplicação não pode iniciar uma comunicação, mas apenas processá-la e respondê-la. Esse é um aspecto relevante, pois qualquer aplicação pode acessar os dados de sensores da aeronave, independentemente da linguagem de programação. O acesso desses dados ocorre através do método GET. Outra funcionalidade importante é a definição de ações a serem seguidas pelo VANT. As definições de ações são feitas através do método POST, em que os dados necessários para a tomada de decisão pelo AP são anexados ao cabeçalho da mensagem.

Através dos serviços fornecidos pelo método GET, é possível acessar dados do AP, barômetro, acelerômetro, GPS, bússola, *power module* e tubo de Pitot. Isso torna possível solicitar informações sobre latitude, longitude, altitude (relativa, absoluta), orientação, velocidade no solo,

[1] <http://dronekit.io/>

Figura 33 – Fluxograma de execução dos sistemas MOSA e IFA.

Fonte: Elaborada pelo autor.

velocidade no ar, *pitch*, *yaw*, *roll*, tensão, nível de bateria, entre outros. A Figura 34 esquematiza esses dados fornecidos pelo S2DK e o componente de hardware responsável por responder a essas solicitações. As informações do método GET e POST são entregues no formato JSON, o que facilita o processo de leitura/gravação.

Outro recurso fornecido pelo S2DK é a definição das ações executadas pela aeronave através do método POST. As seguintes ações são suportadas por este serviço: `set-mission`, `append-mission`, `set-waypoint`, `append-waypoint`, `set-heading`, `set-velocity`, `set-mode` e `set-parameter`. O `set-mission` permite mudar toda a missão, dessa maneira, a missão anterior é sobrescrita por uma nova missão no AP, durante o voo. Outra função disponível é a `append-mission`, que pode adicionar uma nova missão no final da missão atual durante o voo.

O S2DK disponibiliza um conjunto de funcionalidades para que outras aplicações possam utilizar e fazer um gerenciamento completo do voo do VANT. Os sistemas IFA e MOSA utilizam os recursos do S2DK ao criar um gerenciador inteligente que é capaz de fazer voos autônomos.

Os comandos `set-mission`, `append-mission`, `set-waypoint`, `append-waypoint`

Figura 34 – Serviços e operações disponíveis através do sistema S2DK.

Fonte: Elaborada pelo autor.

utilizam internamente, de alguma forma, um *waypoint*. Desse modo, precisamos definir formalmente o que é um *waypoint*. Um *waypoint* é aqui tratado como um local específico no espaço 3D, definido por uma coordenada $(lat,\ lng,\ alt)$. Uma rota ou missão possui, em geral, um conjunto de *waypoints* estabelecidos. Um único *waypoint* pode ser utilizado para diversos fins, como: navegação, decolagem, pouso, entre outros. Os seguintes tipos de *waypoints* estão disponíveis nos sistemas MOSA, IFA e S2DK: `TAKEOFF`, `GOTO`, `LAND`, `LAND_VERTICAL` e `RTL`. A Figura 35 apresenta um esquema deles e, a seguir, é apresentada uma breve descrição.

- **TAKEOFF**: *waypoint* que especifica o procedimento de decolagem da aeronave. Necessita de um argumento: a altitude;
- **GOTO**: *waypoint* que especifica o procedimento de navegação para uma determinada localização geográfica. Necessita de três argumentos: a latitude, a longitude e a altitude;
- **LAND**: *waypoint* que especifica o procedimento de pouso em uma determinada coordenada geográfica. Necessita de dois argumentos: a latitude e a longitude;
- **LAND_VERTICAL**: *waypoint* que especifica o procedimento de pouso na vertical na localização corrente. Não necessita de argumentos;
- **RTL**: *waypoint* que especifica o procedimento de RTL a partir da localização corrente. Não necessita de argumentos.

Todos os *waypoints* possuem os campos/argumentos latitude, longitude e altitude. No entanto, alguns valores não são necessários, dessa forma, são definidos como zero.

Um sistema robusto de detecção, diagnóstico e tomada de decisão foi implementado na plataforma IFA. Para detectar algum evento de comportamento anormal, sensores específicos ou

Figura 35 – Tipos de *waypoints* definidos na arquitetura proposta.

Fonte: Elaborada pelo autor.

sistema de software foram utilizados. Uma etapa de diagnóstico começa nos dados do sensor e do sistema, em seguida, uma ação deve ser feita pelo sistema de tomada de decisão. A Figura 36 mostra as etapas envolvidas desde a detecção até a tomada de decisão pelo sistema IFA. A título de exemplo, se a aeronave tiver um sensor de *Power Module* detectando o nível da bateria, um diagnóstico pode ser estabelecido a partir do nível da bateria. Um diagnóstico de bateria baixa se torna simples comparando o nível atual com um limite (limiar), em que o sistema executa uma ação de segurança quando o limite é violado. A ação tomada pelo sistema, nesse caso, é fazer o pouso emergencial com disparo de um alarme sonoro alertando as pessoas em solo que houve uma falha crítica e que aeronave irá pousar rapidamente.

Figura 36 – Detecção, diagnóstico e tomada de decisão do sistema IFA.

Fonte: Elaborada pelo autor.

Outros sensores/sistemas são mostrados na Figura 36, como GPS, Piloto Automático e Sistema de Previsão do Tempo, entre outros. Diferentes diagnósticos também podem ser avaliados através desses componentes. Por exemplo, uma condição de tempo ruim pode revelar que a aeronave deve abortar a missão atual e acionar uma ação de RTL. Outro exemplo, é utilizar o *status* do GPS como métrica para efetuar um pouso na vertical e soar o alarme, caso essa medida não seja 3D ou 4D *fix*. Um valor de *status* do GPS de 1D ou 2D *fix* indica que poucos

satélites estão sendo captados, dessa forma, a aeronave pode perder/errar a sua localização.

Uma das ideias originais do sistema IFA é o sistema gerenciador de falhas que pode ser representado por uma matriz bidimensional. Essa matriz possui dois componentes principais: eventos (diagnósticos) e ações (tomada de decisão). O mapeamento feito por essa matriz associa um dado evento a uma ação. A ideia era que essa matriz fosse evoluindo aos poucos através da necessidade e conhecimento do projetista. Assim, através desse aprendizado se construiriam as regras do sistema. Esse conjunto de regras aumenta a autonomia do sistema IFA para reagir a diversos eventos que podem levar o VANT a uma falha crítica. A Figura 36 mapeia todo o conhecimento de falhas obtido durante este trabalho, não utilizando a representação matricial, mas sim empregando um fluxo para tomada de decisão.

O Capítulo 9 descreve e avalia um conjunto de detecções, diagnósticos e ações diferentes. Vale ressaltar que o projeto da arquitetura de software permite adicionar ou substituir diferentes equipamentos de hardware e módulos de software, lembrando um sistema *Plug and Play* (P&P). Existe um arquivo de configurações que descreve os componentes de hardware existentes, quais componentes de software serão utilizados, qual é a aplicação a ser executada, entre outras configurações. Um fragmento desse arquivo pode ser visto em Apêndice A. A Equação 5.1 mostra alguns desses parâmetros configuráveis que tornam a arquitetura P&P. A arquitetura introduzida permite também a execução de diferentes tipos de missões, garante requisitos de segurança de voo em tempo real e fornece um maior nível de autonomia para a aeronave.

Os modos de voo utilizados pelo dronekit para a realização da missão autônoma, **Guided** e **Auto**, foram brevemente explicados no Quadro 1. A Figura 37 esquematiza o seu fluxo e explica o momento que cada um é utilizado. Inicialmente, a aeronave está em **Loiter** ou **Stabilize**, que são utilizados apenas para ligar a aeronave. Ao iniciar os sistemas MOSA e IFA, o modo é trocado para **Guided** e um plano de voo é estabelecido pelo MOSA. O **Guided** é utilizado apenas na etapa de decolagem. Por fim, o modo é chaveado para **Auto** após a aeronave atingir a altitude de cruzeiro, permanecendo nesse modo até concluir a missão. Como mencionado anteriormente, uma condição de tempo ruim ao voo pode levar o sistema IFA a chavear automaticamente para RTL. Vale ressaltar que o RTL não é um modo de voo, ele é um comando interno ao **Auto**, porém seu funcionamento é igual ao modo **RTL** definido pelo controle de rádio.

Um mecanismo de segurança adicional foi implementado dentro do sistema IFA a fim de permitir que o piloto em solo (usuário) aborte o sistema autônomo (sistemas MOSA e IFA). Esse mecanismo é acionado caso o piloto, usando o controle de rádio ou via GCS, coloque em modo **RLT** a aeronave. Dessa maneira, os sistemas IFA e MOSA seriam automaticamente encerrados e todo o controle passaria a ser obtido através do controle de rádio ou da GCS. Esse recurso de segurança foi implementado para que o sistema autônomo seja interrompido quando o piloto em solo desejar.

Os pilotos automáticos utilizados, como APM e Pixhawk, possuem mecanismos internos

implementados para lidar com falhas, chamados de *FailSafe*[2]. Como este trabalho trabalhará com falhas, em geral simuladas, esse recurso do AP (*FailSafe*) será desabilitado para que se tenha certeza que as tomadas de decisão feitas pelo AP estejam utilizando o sistema implementado e não as tomadas de decisão nativas do AP. Outro motivo para desabilitar o mecanismo de *FailSafe* é que ele muda o modo de voo toda vez que ocorre uma falha, assim, o sistema autônomo ficaria brigando com o sistema de tomada de decisão do piloto automático.

Figura 37 – Fluxo de transição dos modos de voo do sistema autônomo implementado.

Fonte: Elaborada pelo autor.

Como podemos observar nas Figuras 28 e 30, o sistema controlador da missão (MOSA) foi implementado de maneira genérica o suficiente para se comunicar com os dados obtidos por um sistema de processamento de imagem (câmera RGB) ou um sistema de pulverização. Tais sistemas podem ser integrados ao MOSA conjuntamente ou separadamente. O sistema controlador da missão supervisiona se, por exemplo, numa aplicação de pulverização de plantação, se o sistema de aspersão é acionado como esperado quando o VANT atinge a região alvo. No caso do uso de uma câmera, o sistema pode supervisionar, por exemplo, se a obtenção das imagens está ocorrendo.

Assim, diferente do que foi proposto na arquitetura descrita em Xue *et al.* (2016), o módulo de planejamento de rota está embarcado e faz parte do controle da missão, que passa a ser responsável por enviar os *waypoints* para o piloto automático. Isso se justifica, já que um mesmo algoritmo que planeja ou replaneja uma trajetória pode ser aplicado a diferentes missões, além de dar a aeronave uma maior autonomia. Em Xue *et al.* (2016), um microcontrolador é utilizado para controlar a aspersão sobre as plantações, enquanto nesta tese, esse controle utiliza um *companion computer*, que permite a aeronave fazer outras tarefas simultaneamente sem adicionar custos à aeronave.

Por último, a arquitetura definida em Ramasamy *et al.* (2016) não foca em uma aplicação específica, porém os autores propõem o desenvolvimento de sistemas mais elaborados para

[2] <http://ardupilot.org/copter/docs/failsafe-landing-page.html>

controle da execução da trajetória. Nesse caso, o planejamento da rota ocorre na estação de solo e o controle da trajetória é realizado parcialmente pelo sistema embarcado em tempo real. Os autores em Ramasamy *et al.* (2016), dada a tecnologia empregada, conseguem evitar obstáculos e executar manobras na aeronave com maior precisão e segurança. Sistemas semelhantes poderiam ser incorporados ao módulo IFA aqui proposto. Nesse sentido, se um obstáculo ou outra aeronave aparecesse durante a execução da missão, detectada por um sensor como o LIDAR, o sistema IFA tomaria o controle da trajetória da aeronave e executaria as manobras evasivas necessárias para evitar colisão, retornando o controle ao MOSA em seguida.

O sistema de detecção e desvio de obstáculo, proposto em Chiaramonte (2018), pode ser entendido, como dois módulos de segurança em voo. O módulo de detecção poderia ser incorporado dentro do sistema IFA, já o módulo de desvio não seria necessário, uma vez que o IFA possui módulos que são capazes de alterar/desviar a rota.

Conceitos relacionados a sistemas para controle de trajetórias, como descritos em Prodan *et al.* (2013), Ramasamy *et al.* (2016), e sistemas capazes de mitigar falhas, como apresentados em Morozov e Janschek (2016), Arantes *et al.* (2015), Arantes (2016), foram integrados à arquitetura proposta. O desenvolvimento de cada módulo passou por implementações em software e integração de hardware de modo similar às metodologias adotadas em Brown, Estabrook e Franklin (2011), Xue *et al.* (2016), Ramasamy *et al.* (2016), Morozov e Janschek (2016).

6.4 Sistemas de Comunicação do VANT

As opções de comunicação utilizadas estão ilustradas na Figura 38, como Controle de Rádio (RC, do inglês *Radio Control*), telemetria e Wi-Fi. A presença do RC em nossa proposta é apenas uma medida de segurança, já que o VANT voará autonomamente. O RC se comunica a 2.4 GHz com o seu receptor localizado no VANT. A telemetria é empregada nesse trabalho, para enviar dados de voo para a GCS, como por exemplo, o QGroundControl, e essa comunicação ocorre a 433 MHz. A conexão Wi-Fi ocorre entre a GCS e o CC embarcado no VANT. Para que a rede Wi-Fi esteja disponível em campo, pode-se utilizar diversas formas, como notebook ou celular, ambos no modo *hotspot*, ou, ainda, um roteador Wi-Fi. Todas as três opções foram avaliadas e efetuaram-se voos reais com todas elas. Dentre essas opções, estamos utilizando o roteamento através do celular. O projetista da missão se conectará ao computador de bordo através do protocolo *Secure Shell* (SSH).

A Figura 39 dá alguns detalhes sobre os sistemas de comunicação. O agente (usuário ou piloto) pode interagir diretamente com o AP através do RC. Se o agente quiser interagir com o CC, ele deve utilizar um terminal SSH. A comunicação com o VANT começa enviando um sinal para acionar os sistemas IFA e MOSA. A Raspberry Pi, a Intel Edison e a BeagleBone Black executam o sistema operacional Linux (Raspbian, Yocto e Debian, respectivamente). A API PyMAVLink é uma biblioteca para comunicação através do protocolo MAVLink. A API do

Figura 38 – Sistemas de comunicação disponíveis no VANT utilizado.

Fonte: Elaborada pelo autor.

Dronekit tem um conjunto de comandos de alto nível, que pode ser traduzido para MAVLink, permitindo que os sistemas controlem o VANT. O MAVProxy é uma GCS e um *proxy* que abre um conjunto de portas para comunicação entre diversas GCS e o AP. Outra maneira dele interagir com o AP é utilizando uma GCS.

Figura 39 – Comunicação e interação entre os módulos/componentes do CC, AP e PC.

Fonte: Elaborada pelo autor.

A comunicação entre os sistemas IFA e MOSA ocorre através do protocolo de rede TCP/IP, utilizando *sockets*. O sistema IFA é o servidor e o MOSA é o cliente na aplicação, indicando a ordem de inicialização de cada sistema.

A Figura 40 mostra detalhadamente como as mensagens são enviadas. Uma vez iniciado o sistema MOSA, ele informará ao IFA que está pronto (mensagem: `MOSA->IFA[INITIALIZED]`, leia-se o MOSA diz ao IFA que está inicializado). Em seguida, o IFA informa a localização do *home* da aeronave (*homelocation*), parâmetro importante durante o voo, (mensagem: `IFA->MOSA[HOMELOCATION][DATA...]`, leia-se o IFA diz ao MOSA o *homelocation* através dos dados enviados). Posteriormente, o IFA informa que o MOSA pode iniciar o cálculo da rota a ser seguida. Assim que esse cálculo termina, o MOSA informa ao IFA que a execução da rota já foi iniciada. Durante a missão, se o IFA detectar uma falha, o MOSA será parado e uma rota de emergência será calculada pelo IFA. Finalmente, se durante a execução da missão ocorrer alguma falha no MOSA, o IFA é avisado sobre tal ocorrência, iniciando uma rota de emergência. É através desse protocolo de mensagens simples que ocorre toda a comunicação entre os sistemas MOSA e IFA.

Figura 40 – Trocas de mensagens de comunicação entre o MOSA e o IFA.

Fonte: Elaborada pelo autor.

6.5 Arquitetura de Baixo Custo

Um dos objetivos deste trabalho é a construção de uma arquitetura de hardware de baixo custo. A seguir, na Tabela 10, estão descritos os principais equipamentos de hardware utilizados na construção dos VANTs. De maneira geral, os componentes descritos foram separados em componentes que voarão com a aeronave (*air*) e os componentes que ficarão em solo (*ground*).

Tabela 10 – Preços e pesos médios dos equipamentos utilizados na arquitetura de hardware.

Local de Uso	Equipamento	Modelo/Empresa	Preço[3]	Peso (g)
Air	*Frame* + Motores + ESCs + Hélices	iDroneAlpha	US$146,00	725 g
Air	Bateria	Venom 2200 mAh	US$10,79	145 g
Air	Piloto Automático	APM v2.8	US$27,00	31 g
Air	Piloto Automático	Pixhawk v1	US$65,00	38 g
Air	*Companion Computer*	Intel Edison	US$85,00	63 g
Air	*Companion Computer*	Raspberry Pi 3	US$34,00	92 g
Air	*Companion Computer*	BeagleBone Black	US$69,00	38 g
Air	GPS	uBlox Neo-6M	US$19,99	39 g
Air	Receptor de Rádio	FlySky FS-iA6	US$14,99	7 g
Air	Telemetria *Air*	Readytosky 3DR	US$11,98	21 g
Air	*Power Module*	HobbySoar	US$12,69	17 g
Air	Câmera Raspberry Pi	Module V2 - 8MP	US$28,50	4 g
Air	*Buzzer*	Cylewet	US$1,00	2 g
Air	Alarme de Bateria	DLFPV	US$5,59	9 g
Air	Conversor USB para TTL	PL2303	US$6,99	5 g
Air	Regulador de Tensão	LM2596	US$2,00	11 g
Ground	Controle de Rádio	FlySky FS-i6-M2	US$50,09	396 g
Ground	Telemetria *Ground*	Readytosky 3DR	US$11,98	21 g

Fonte: Elaborada pelo autor.

De forma geral, todos os softwares utilizados são gratuitos, com exceção do X-Plane. Entretanto, nesta pesquisa utiliza-se a versão trial desse simulador, não existindo custos associados a softwares. Assim, os únicos gastos do VANT são relativos ao custo de hardware. O custo total do iDroneAlpha é de US$374,60 e seu peso é de 1079 gramas. Esta lista de equipamentos não está totalmente completa, já que, em solo, ficarão, além do controle de rádio e telemetria *ground*, um computador (notebook) para acompanhar a missão, um celular utilizado como roteador Wi-Fi e no ar poderá existir paraquedas e outros atuadores/sensores a bordo da aeronave.

Assim, o objetivo de se propor uma arquitetura de baixo custo foi alcançado ao se construir um VANT utilizando aproximadamente US$375,00 considerando um peso por volta de 1,5 kg, incluindo carga útil (*payload*). Foi assumido neste trabalho que uma arquitetura de baixo custo deve ter custo operacional de no máximo US$1000,00. Desse modo, nossa arquitetura se encaixa nessa categoria. Consideramos também o aspecto de baixo peso, um valor abaixo de 5,0 kg é assumido. Pensando nesses limiares estabelecidos e na classificação proposta por Watts, Ambrosia e Hinkley (2012), Heffner e Foucher (2017), mostrada na Figura 2, esta tese abrange veículos da categoria *Micro Aerial Vehicle* (MAV), como já dito anteriormente.

[3] Preços obtidos, em geral, através dos sites www.amazon.com e hobbyking.com

6.6 Considerações Finais

Este capítulo apresentou a arquitetura proposta, destacando seus aspectos de hardware, de software, de comunicação e como as arquiteturas da literatura poderiam ser encaixadas neste trabalho. Nos aspectos relacionados ao hardware, são destacados os VANTs produzidos e utilizados nos experimentos: iDroneAlpha e iDroneBeta. Nos aspectos relacionados ao software, são destacados os sistemas IFA, MOSA, S2DK e UAV-GCS. Nos aspectos relacionados a comunicação, deu-se destaque às comunicações entre os dispositivos e como é feita a comunicação interna no software.

O capítulo seguinte apresentará o metodologia utilizada no desenvolvimento da arquitetura.

CAPÍTULO

7

METODOLOGIA

“ O ignorante afirma, o sábio duvida, o sensato reflete. ”

Aristóteles

7.1 Considerações Iniciais

O presente capítulo apresentará os planejadores e replanejadores de rotas utilizados. Posteriormente, será mostrado um protocolo definido para especificação do mapa e da missão a serem utilizados nos voos autônomos. Também serão apresentadas a forma de avaliação da arquitetura e a maneira como foram feitas as conexões de hardware nos experimentos *Hardware-In-The-Loop* (HITL) e nos voos reais. Por fim, o *framework* ProOF que foi utilizado para o desenvolvimento dos métodos será brevemente mostrado.

7.2 Planejamento e Replanejamento de Rota do VANT

A Figura 30, que apresenta a arquitetura de software dos sistemas MOSA e IFA, mostra o módulo de planejamento de rota. Esse módulo fica integrado ao sistema MOSA e dá suporte aos seguintes algoritmos: HGA4m, CCQSP4m e FixedRoute4m. Eles serão brevemente descritos a seguir:

- **HGA4m**: sigla para *Hybrid Genetic Algorithm for mission*. É um algoritmo genético (GA) combinado com a estratégia grafo de visibilidade para o problema de planejamento de caminho entre dois pontos. O HGA4m procura encontrar um caminho com alocação de risco para minimizar o combustível e evitar colisão com obstáculos. Esse algoritmo foi proposto em Arantes *et al.* (2016);

- **CCQSP4m**: sigla para *Chance Constraint Qualitative State Plan for mission*. É um algoritmo baseado em Programação Linear de Integração Mista (MILP) para planejamento de missão com vários pontos-alvo. O método procura encontrar um caminho com alocação

de risco para minimizar o combustível e evitar obstáculos. Esse algoritmo, proposto em Arantes (2017), é baseado no planejador de Ono, Williams e Blackmore (2013);

- **FixedRoute4m**: sigla para *Fixed Route for mission*. É uma estratégia simples/clássica para a execução de voo em VANT, em que uma rota pré-definida (em geral, feita manualmente) é carregada para ser executada pela aeronave. Essa estratégia se assemelha a definição/-setagem de *waypoints* utilizando alguma estação de controle de solo, como o Mission Planner ou QGroundControl. A principal diferença é que os *waypoints* são definidos no piloto automático, utilizando o sistema MOSA, poucos segundos antes de iniciar o voo. A trajetória não tem alocação de risco ou desvio de obstáculos.

Ainda na Figura 30, é apresentado o módulo de replanejamento de rota que fica integrado ao sistema IFA. Nesse módulo os seguintes algoritmos estão disponíveis: GA4s, MPGA4s, GH4s, DE4s, MS4s, GA-GH-4s, GA-GA-4s e Pre-Planned4s. Eles serão brevemente descritos a seguir:

- **GA4s**: sigla para *Genetic Algorithm for security*. É um GA simples para o problema de pouso emergencial de VANTs em caso de situação crítica. Este algoritmo busca pousar a aeronave sobre uma região bonificadora, com base no tipo de situação crítica, evitando obstáculos e com alocação de risco. Mais detalhes desse algoritmo são encontrados em Arantes *et al.* (2017b);

- **MPGA4s**: sigla para *Multi-Population Genetic Algorithm for security*. Também foi desenvolvido para lidar com pouso de emergência para VANTs na ocorrência de situação crítica, com as mesmas restrições definidas para o GA4s: região bonificadora, evitando obstáculos e alocação de risco. O GA4s e MPGA4s têm desempenho semelhante em termos de qualidade de solução e tempo de execução, conforme relatado em Arantes *et al.* (2017b);

- **GH4s**: sigla para *Greedy Heuristic for security*. É uma abordagem gulosa para o problema de pouso de emergência em eventualidade de ocorrer uma situação crítica. Mais detalhes desse algoritmo são encontrados em Arantes *et al.* (2017b);

- **DE4s**: sigla para *Differential Evolution for security*. É um algoritmo baseado em evolução diferencial (DE) simples para o problema de pouso emergencial de VANTs em caso de situação crítica, como GA4s e MPGA4s. A DE também é uma metaheurística, como o GA, que melhora as suas soluções ao longo das gerações. O DE4s foi desenvolvido por este trabalho e integrado no sistema proposto, baseado em Das e Suganthan (2011), Li e Liu (2010). O Algoritmo 1 descreve seu funcionamento em que, inicialmente, cria-se uma população de indivíduos que são inicializados e avaliados. Um processo de evolução até um critério de parada ser atingido é iniciado. Um indivíduo alvo (*indTarget*) é selecionado utilizando alguma pressão de seleção, por exemplo, utilizando torneio. Em seguida, três

indivíduos (*ind1, ind2, ind3*) são obtidos de maneira aleatória, ou seja, sem nenhuma pressão de seleção. Um conjunto de operadores é aplicado nos indivíduos, de forma a gerar novos indivíduos são eles: *difference*, *disturbance* e *crossover*. O indivíduo *indChild* é então avaliado e inserido na população no lugar do pior indivíduo. Por fim, a melhor solução encontrada é retornada pelo método após os critério de parada ser atingido;

- **MS4s**: sigla para *Multi-Start for security*. É uma abordagem heurística construtiva aleatória, baseada em Martí *et al.* (2016), aplicada ao problema de pouso de emergência na ocorrência de situação crítica. O MS4s foi desenvolvido por este trabalho e integrado no sistema proposto. O Algoritmo 2 descreve esse procedimento, em que, inicialmente, uma solução é criada, uma busca local é aplicada e então a solução é avaliada. Este processo é repetido até que se atinja o critério de parada do algoritmo e, então, a melhor solução é retornada;

- **GA-GH-4s**: sigla para *Genetic Algorithm and Greedy Heuristic for security*. É uma abordagem, baseada em comitê, que executa, em paralelo, um GA4s e um GH4s e retorna a melhor solução entre os dois métodos. Essa estratégia explora a técnica de programação n-versões, que é uma forma de tolerância à falhas (redundância) implementada em software. A Figura 41a mostra o fluxo de execução dessa estratégia. O GA-GH-4s foi desenvolvido por este trabalho e integrado no sistema proposto. Mais detalhes desse algoritmo são encontrados em Arantes *et al.* (2017) no Apêndice C;

- **GA-GA-4s**: sigla para *Genetic Algorithm and Genetic Algorithm for security*. É uma abordagem, baseada em comitê, que executa, em paralelo, dois GA4s e retorna a melhor solução entre os dois métodos. Essa estratégia também explora a técnica de programação n-versões. A Figura 41b apresenta a sequência de execução dessa estratégia. O GA-GA-4s foi desenvolvido por este trabalho e integrado no sistema proposto. Mais detalhes desse algoritmo são encontrados em Arantes *et al.* (2017) no Apêndice C;

- **Pre-Planned4s**: sigla para *Pre-Planned for security*. É uma estratégia em que são previamente calculadas as rotas emergenciais. Caso ocorra uma falha crítica, a rota com ponto inicial mais próximo a localização corrente do VANT é retornada.

Todos os algoritmos de pouso emergencial, acima citados, buscam pousar o VANT na região bonificadora, em geral, mais próxima, fazendo o desvio de obstáculo. Os algoritmos GA-GH-4s e GA-GA-4s, ao executarem em paralelo, conseguem extrair o máximo das plataformas de hardware atuais, que são, em geral, *multicore*. Alguns exemplos de computadores de bordo com mais de um núcleo são: Raspberry Pi, Intel Edison, BeagleBone Black e Odroid XU4. Dessa forma, depois que uma situação crítica é detectada, os métodos baseados em comitê são executados em paralelo utilizando dois núcleos e a melhor solução é retornada no final.

A Figura 42 detalha o módulo de replanejamento de rota do sistema IFA, apresentado na Seção 6.3. Após detectar uma falha, o sistema IFA aciona o módulo de replanejamento

Algoritmo 1 – Evolução Diferencial para segurança

```
 1: procedimento DE4S(map)
 2:     createPopulation(routes);                    ▷ Cria uma população de indivíduos em routes
 3:     initialize(routes, map);                     ▷ Inicializa esses indivíduos
 4:     evaluate(routes, map);                       ▷ Avalia esses indivíduos
 5:     repita
 6:         selectTournament(indTarget);             ▷ Seleciona um indivíduo com pressão de seleção
 7:         selectRandom(ind1, ind2, ind3);          ▷ Seleciona 3 indivíduos aleatoriamente
 8:         indDiff ← difference(ind1, ind2);        ▷ Aplica operador diferença
 9:         indDist ← disturbance(ind3, indDiff, weightDiff);   ▷ Aplica operador distúrbio
10:         indChild ← crossover(indDist, indTarget, rateCross); ▷ Aplica operador crossover
11:         evaluate(indChild, map);                 ▷ Avalia o indivíduo
12:         add(indChild);                           ▷ Adiciona o indivíduo no lugar do pior
13:     até reach(stoppingCriterion)                 ▷ Repita até atingir o critério de parada
14:     route* ← getBestRoute(routes);               ▷ Selecione a melhor rota encontrada
15:     retorna route*;                              ▷ Retorna a melhor rota
16: fim procedimento
```

Algoritmo 2 – Multi-Start para segurança

```
 1: procedimento MS4S(map)
 2:     f* ← ∞;                                      ▷ Inicializa f* com um valor de fitness ruim
 3:     repita
 4:         createIndividual(route);                 ▷ Cria um indivíduo em route
 5:         initialize(route, map);                  ▷ Inicializa esse indivíduo
 6:         localsearch(route);                      ▷ Executa uma busca local
 7:         f ← evaluate(route, map);                ▷ Avalia esse indivíduo e salva o resultado em f
 8:         se f < f* então                          ▷ Se melhor, atualiza
 9:             route* ← route;                      ▷ Atualiza a melhor rota
10:             f* ← f;                              ▷ Atualiza o melhor fitness
11:         fim se
12:     até reach(stoppingCriterion)                 ▷ Repita até atingir o critério de parada
13:     retorna route*;                              ▷ Retorna a melhor rota encontrada
14: fim procedimento
```

que envia as entradas: estado inicial, mapa, modelo de dinâmica, falha e configurações para o algoritmo de replanejamento. No módulo de replanejamento, qualquer uma das oito estratégias para pouso, acima descritas, podem ser selecionadas. A estratégia escolhida retornará as seguintes informações: conjunto de *waypoints*, local de pouso, probabilidade de pouso e *fitness*. Essas saídas retornam ao módulo de replanejamento de rota, que envia um comando ao AP para abortar a missão atual, enviando, em seguida, um comando com a nova rota.

O módulo de planejamento de rota do sistema MOSA é detalhado na Figura 43 de forma semelhante ao módulo de replanejamento. Inicialmente, o sistema MOSA aciona o módulo de planejamento que envia as entradas: estado inicial, pontos-alvo, mapa, modelo de dinâmica e configurações para o algoritmo de planejamento. No módulo de planejamento, qualquer uma das três estratégias acima descritas podem ser selecionadas. A estratégia escolhida retornará as

Figura 41 – Estratégias implementadas executando os métodos em paralelo.

(a) Combinando o GA4s e GH4s. (b) Duas execuções do GA4s.

Fonte: Elaborada pelo autor.

Figura 42 – Arquitetura do sistema embarcado no módulo de replanejamento de rotas.

Fonte: Elaborada pelo autor.

seguintes informações: conjunto de *waypoints*, risco assumido e *fitness*. Essas saídas retornam ao módulo de planejamento de rota, que envia um comando ao AP para executar a missão com a rota calculada.

Os métodos planejadores (HGA4m e CCQSP4m) e replanejadores (GA4s, MPGA4s, GH4s, DE4s, MS4s, GA-GH-4s e GA-GA-4s) são módulos de software completamente independentes dos sistemas MOSA e IFA. Ao se projetar a plataforma autônoma dessa forma, garante-se uma maior modularidade e flexibilidade. Assim, eles são executados através de chamadas de sistema.

Após o sistema IFA detectar uma falha crítica, ele envia em tempo real, para os replanejadores, os dados de posição, velocidade e orientação da aeronave. Esses dados são utilizados como entradas para calcular a rota emergencial. A posição da aeronave obtida pelo GPS é

Figura 43 – Arquitetura do sistema embarcado no módulo de planejamento de rotas.

Fonte: Elaborada pelo autor.

dada em coordenadas geográficas (latitude e longitude) e os replanejadores utilizam apenas coordenadas cartesianas (posição x, y), dessa forma, antes de enviar essas informações, deve-se converter para coordenadas cartesianas. A orientação da aeronave obtida pela bússola (*heading*) é representada baseada no sistema de ângulos da aeronáutica em que o norte representa 0º, no entanto o replanejador utiliza o sistema de ângulos da matemática. Dessa maneira, antes de enviar essa informação, ela deve ser convertida. A seção seguinte mostra como são feitas essas conversões.

Alguns outros parâmetros de entrada utilizados, em geral, pelos replanejadores de rotas são: o número de *waypoints* a ser utilizado ($\mathcal{W}_R$); o risco de colisão assumido (Δ_R); o critério de parada baseado no tempo de execução do método ($\mathcal{T}_{SC_R}$); e o local de execução do replanejamento ($\mathcal{L}_R$).

Os códigos-fonte dos planejadores de rotas (HGA4m e CCQSP4m) utilizados podem ser acessados no repositório [1]. Os códigos-fonte dos oito replanejadores de rotas desenvolvidos podem ser acessados no repositório [2].

7.3 Conversões Utilizadas nos Planejadores/Replanejadores

O Piloto Automático (AP) trabalha com o sistema de coordenadas geográficas, enquanto os (re)planejadores trabalham com coordenadas cartesianas. Dessa forma, a conversão do espaço de coordenadas geográficas para cartesianas é realizada de modo que os planejadores possam executar. Por fim, as coordenadas da rota são convertidas de coordenadas cartesianas para geográficas e enviadas ao AP para que ele possa seguir a rota. Para esse cálculo, considera-se

[1] <https://github.com/marcio-da-silva-arantes/ProOF>
[2] <https://github.com/jesimar/Path-Replanning>

a Terra um elipsoide conforme Hossomi (2015), em que a posição p_x (no eixo x) é medida na linha do equador e a posição p_y (no eixo y) é medida a partir do meridiano central. O valor da longitude (λ) é medido pelo ângulo entre o meridiano zero e o ponto medido expresso em graus, variando de $\pm 180^o$. O valor da latitude (γ) é medido pelo quanto se está próximo do polo ou equador ao longo de um meridiano, e é representado por um ângulo de $\pm 90^o$, em que 0° é o equador. Dessa maneira, vale a conversão:

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} (\lambda - \bar{\lambda}) \cdot \frac{\pi \cdot \mathcal{R}_E}{180} \cdot cos(\bar{\gamma} \cdot \frac{\pi}{180}) \\ (\gamma - \bar{\gamma}) \cdot \frac{\pi \cdot \mathcal{R}_M}{180} \end{bmatrix} \tag{7.1}$$

$$\begin{bmatrix} \lambda \\ \gamma \end{bmatrix} = \begin{bmatrix} p_x \cdot \frac{180}{\pi \cdot \mathcal{R}_E} \cdot \frac{1}{cos(\bar{\gamma} \cdot \frac{\pi}{180})} + \bar{\lambda} \\ p_y \cdot \frac{180}{\pi \cdot \mathcal{R}_M} + \bar{\gamma} \end{bmatrix} \tag{7.2}$$

$$\begin{aligned} \mathcal{R}_E &= 6.378.137 \\ \mathcal{R}_M &= 6.356.752 \end{aligned} \tag{7.3}$$

em que, $\mathcal{R}_E$ e $\mathcal{R}_M$ são os eixos maior (raio equatorial) e menor (raio meridional) da Terra, λ e γ são longitude e latitude medidos em graus, $\bar{\lambda}$ e $\bar{\gamma}$ são as coordenadas de longitude e latitude de um ponto de referência próximo ao ponto considerado. A Equação 7.1 mostra a conversão de coordenadas geográficas para cartesianas. A Equação 7.2 mostra a conversão de coordenadas cartesianas para geográficas. A Equação 7.3 mostra os valores de $\mathcal{R}_E$ e $\mathcal{R}_M$ utilizados nos cálculos. Certamente, esse tipo de conversão apresenta singularidades nos limites em que $\lambda = \pm 180^o$ e $\gamma = \pm 90^o$, quando fecha-se um arco ao redor do planeta. Esse caso, entretanto, foge do escopo das aplicações deste trabalho.

A Figura 44 mostra duas maneiras distintas de representação de ângulos. Na parte esquerda, vemos como o sistema aeronáutico considera os ângulos, em que 0° é o norte e o ângulo cresce em sentido horário. Já na parte direita da figura, temos a representação matemática dos ângulos, em que 90° é o norte e o ângulo cresce em sentido anti-horário. A Equação 7.4 mostra como é feita a conversão do ângulo de coordenadas aeronáuticas em coordenadas matemáticas sujeita as Restrições 7.5. Na Equação 7.4, o operador **mod** representa o módulo, ou seja, o resto da divisão. Esse tipo de conversão é utilizada, pois a orientação da aeronave (*heading*) é baseada no sistema de ângulos da aeronáutica, todavia o replanejador precisa desse valor utilizando o sistema de ângulos da matemática.

$$\alpha_{math} = (360 - \alpha_{aero} + 90) \textbf{ mod } 360 \tag{7.4}$$

$$\begin{aligned} 0 \leq \alpha_{aero} < 360 \\ 0 \leq \alpha_{math} < 360 \end{aligned} \tag{7.5}$$

Figura 44 – Conversão entre sistemas de ângulos da aeronáutica e matemática.

Fonte: Elaborada pelo autor.

7.4 Protocolo para Especificação do Mapa e da Missão

A especificação do cenário (mapa) e da missão é uma etapa bastante importante durante a realização do voo real. A ferramenta Google Earth foi utilizada por este trabalho para definir o mapa e a missão a ser cumprida pelo VANT. Este programa apresenta um modelo tridimensional do globo terrestre e possui um ambiente, que permite edições sobre o mapa (GOOGLE-EARTH, 2017). Dessa forma, um protocolo próprio foi definido para a criação do mapa e da missão utilizando esse ambiente. Na Figura 45, essa ferramenta foi utilizada para definir o cenário (mapa), os *waypoints* e o local de início de gravação de vídeo da missão. As definições dessas regiões e o plano da missão utilizam o menu à esquerda da figura. Essas informações especificadas pelo projetista da missão serão salvas em um formato apropriado. Posteriormente, os sistemas MOSA e IFA farão as leituras dessas informações, de maneira a seguir os *waypoints* e efetuar a gravação do vídeo.

Esse mapeamento prévio das regiões é importante para que o VANT conheça o cenário de voo, uma vez que não será feito o processamento de imagens a bordo para desvio de obstáculos. Na presente abordagem, todo o mapa estará na memória da aeronave. Em diversos trabalhos anteriores, como em Blackmore, Ono e Williams (2011), Ono, Williams e Blackmore (2013), Arantes *et al.* (2016), apenas cenários artificiais com coordenadas cartesianas foram analisados, já este trabalho traz cenários mais realistas.

A definição do mapa e da missão se dá através de um conjunto de *tags*/notações bem definidas. As seguintes notações foram estabelecidas, baseadas na Seção 5.2, a fim de mapear as regiões em que ocorrerão as missões:

map_nfz: também chamada de região não navegável (Φ^n), define as áreas que o VANT está estritamente proibido de sobrevoar ou pousar;

map_obstacle: também chamada de região de obstáculo (Φ^o), define as áreas que o VANT está

Figura 45 – Protocolo especificado para criação do mapa e definição da missão.

Fonte: Elaborada pelo autor.

proibido de sobrevoar ou pousar;

map_penalty: também chamada de região penalizadora (Φ^p), define as áreas que o VANT pode sobrevoar, mas não deve pousar;

map_bonus: também chamada de região bonificadora (Φ^b), define as áreas que o VANT pode sobrevoar e pousar;

map_scenic: também chamada de região cênica (Φ^s), define as áreas que a aeronave deve sobrevoar, representa uma região de interesse;

map_frontier: também chamada de região remanescente (Φ^r), define a fronteira da missão.

As seguintes notações/*tags* foram definidas a fim de registrar as atividades relacionadas ao plano da missão:

geo_base: define um ponto de referência utilizado para a transformação entre os sistemas de coordenadas geográficas e cartesianas. A longitude e latitude desse ponto de referência são os parâmetros: $\bar{\lambda}$ e $\bar{\gamma}$, utilizados nas Equações 7.1 e 7.2;

waypoint: define um ponto-alvo que a aeronave deve cumprir durante a sua missão. Os *waypoints* especificados serão utilizados pelo MOSA para estabelecer a rota a ser seguida pela aeronave;

cmd_picture: define o ponto em que uma fotografia deve ser retirada. O sistema MOSA ao atingir esse ponto efetua a retirada de fotografia;

cmd_photo_seq: define o ponto em que um conjunto de fotografias em sequência deve ser retirado. O sistema MOSA ao atingir esse ponto efetua uma quantidade $n_{picture}$ de fotografias com um *delay* ($d_{picture}$) entre elas. Os parâmetros $n_{picture}$ e $d_{picture}$ são especificados através de um arquivo de entrada;

cmd_video: define o ponto a partir do qual se inicia a filmagem da região por um tempo t_{video}. O parâmetro t_{video} é especificado através de um arquivo de entrada;

cmd_buzzer: define o ponto em que um disparo de apito do *buzzer* é efetuado;

cmd_spraying_begin: define o ponto a partir do qual se inicia a pulverização da região;

cmd_spraying_end: define o ponto que indica o término da pulverização.

Todas as regiões cadastradas devem ser polígonos convexos, adicionados no Google Earth através do recurso `Polígono`. A construção de regiões não convexas deve ser feita através de duas ou mais regiões não convexas. Todas as atividades do plano da missão devem ser adicionadas no Google Earth através do recurso `Marcador`. Foi estabelecido que essas *tags* devem ser utilizadas apenas no começo do nome da região ou da atividade do plano da missão. Dessa maneira, as seguintes notações são válidas: `waypoint_takeoff`, `cmd_video1`, `map_obstacle1`, entre outras.

Após o projetista especificar o mapa e o plano da missão, ele deve salvar essas informações em um arquivo em formato apropriado: o *Keyhole Markup Language* (KML). Esse arquivo será posteriormente lido por um programa que após executado irá gerar as entradas necessárias aos sistemas MOSA e IFA.

A justificativa da escolha dessa ferramenta para a definição do mapa e da missão se deve ao seu ambiente de edição, que atende requisitos como: simplicidade de utilização; capacidade de criação de polígonos e marcadores; capacidade de criação de linhas e caminhos; capacidade de salvar/carregar as informações em arquivo; e facilidade de interpretação/manipulação do arquivo KML.

7.5 Avaliação da Arquitetura

Uma forma de fazer a validação da robustez da arquitetura proposta é através de testes com diferentes componentes de hardware e software. A Figura 46 mostra como serão feitos esses testes, em que diferentes aeronaves, tipos de pilotos automáticos, *companion computers*, planejadores e replanejadores de rotas poderão ser trocados de modo transparente para o restante do sistema. Nesse sentido, o objetivo é que a arquitetura de hardware e software desenvolvida consiga se adaptar a esses diferentes equipamentos e softwares sem grandes alterações no sistema. Esse chaveamento de componentes da aeronave é o que se estamos chamando de arquitetura *Plug and Play* (P&P).

Figura 46 – Fluxo de avaliação da arquitetura de hardware e software P&P.

Fonte: Elaborada pelo autor.

Além de permitir a fácil troca de planejadores/replanejadores, objetivou-se durante o desenvolvimento que outros usuários do sistema conseguissem desenvolver e integrar novos algoritmos ao sistema MOSA e IFA de modo fácil e rápido.

Durante a validação da arquitetura, três diferentes formas de experimentos foram conduzidas são elas: `SITL`, `HITL` e `REAL_FLIGHT`.

- **SITL**: A arquitetura proposta foi avaliada através de experimentos *Software-In-The-Loop*. Em geral, essa forma de avaliação é a mais fácil de se conduzir e útil na calibração e ajustes dos sistemas MOSA e IFA e de seus módulos. A Seção 2.11 apresentou alguns outros detalhes sobre essa técnica;

- **HITL**: A arquitetura proposta foi avaliada também através de experimentos *Hardware-In-The-Loop*, em que o comportamento dos sistemas no computador de bordo foram testados. Esse tipo de experimentação, em geral, é efetuado após um conjunto de experimentos SITL, em que o sistema funciona perfeitamente. Através da avaliação HITL, os seguintes problemas podem ser detectados para posterior correção: problema na comunicação entre os sistemas no CC e na GCS; problema devido ao menor desempenho da plataforma; e problema devido ao sistema estar executando em uma plataforma de hardware diferente, com diferentes sistemas instalados. A Figura 47 apresenta um diagrama com as conexões de hardware dos experimentos HITL efetuados. Na Figura 47a, tem-se a conexão da Raspberry Pi 3 com o computador de solo (GCS), em que são realizados os experimentos. As Figuras 47b e 47c apresentam, respectivamente, como estão dispostas as conexões na Intel Edison e na BeagleBone Black. A Seção 2.12 explorou alguns outros detalhes sobre essa técnica;

Figura 47 – Conexões de hardware entre o PC e CC para fazer os experimentos HITL.

(a) Conexão da GCS e a Raspberry Pi 3. (b) Conexão da GCS e a Intel Edison.

(c) Conexão da GCS e a BeagleBone Black.

Fonte: Elaborada pelo autor.

- **REAL_FLIGHT**: A plataforma autônoma proposta foi avaliada em experimentos com voos reais. Essa etapa, em geral, é a última a ser conduzida. Esse tipo de experimento é conduzido após os sistemas e algoritmos serem avaliados em SITL e HITL e tudo funcionar perfeitamente. Através da avaliação REAL_FLIGHT, os seguintes problemas, que não ocorriam na etapas anteriores, podem ser detectados: problema na decolagem devido à inclinação do solo; problema na decolagem/missão devido às condições de vento; problema de perda de sinal do Wi-Fi devido à distância do VANT e da GCS; problema de perde/ganho de altitude anormal devido à imprecisão do barômetro (os sensores utilizados nos experimentas SITL e HITL não possuem tais problemas por serem simulados). A Figura 48 esquematiza o diagrama com as conexões de hardware dos experimentos REAL_-FLIGHT. A Figura 48a mostra as conexões do computador de solo (GCS), com a Raspberry Pi 3 e com a APM. As figuras restantes mostram as diferentes configurações de hardware para outros pilotos automáticos e computadores de bordo avaliados.

Figura 48 – Conexões de hardware entre o CC, AP e PC para fazer os experimentos reais.

(a) Conexão da GCS, Raspberry Pi e APM.

(b) Conexão da GCS, Raspberry Pi e Pixhawk.

(c) Conexão da GCS, Intel Edison e APM.

(d) Conexão da GCS, Intel Edison e Pixhawk.

(e) Conexão da GCS, BeagleBone Black e APM.

(f) Conexão da GCS, BeagleBone e Pixhawk.

Fonte: Elaborada pelo autor.

7.6 Separação de Interesses

Segundo Sommerville (2011), a Separação de Interesses (SdI) é um princípio-chave em projeto e implementação de grandes softwares. Isso significa que o software deve ser organizado de tal forma que cada elemento do programa faça apenas uma coisa. Podemos pensar que a SdI é uma abordagem para desenhar uma solução ou serviço de TI que divide o problema em partes que podem ser resolvidas de maneira independente. Esta abordagem separa "o quê" será feito do "como" será feito (ITIL, 2019).

Uma pergunta que poderia ser feita é por que desenvolver os sistemas MOSA e IFA de forma separada e não unificada? A resposta para isso é a separação de interesses. Ao manter os dois sistemas separados, garante-se um maior nível de modularidade, flexibilidade, reusabili-

dade, confiabilidade, organização de conceitos e tolerância a erros. No contexto que estamos trabalhando, podemos pensar na SdI de forma lógica e física entre os sistemas MOSA e IFA.

Os sistemas MOSA e IFA foram pensados para serem sistemas distintos, sendo assim, eles precisam se comunicar de maneira organizada. Durante os experimentos conduzidos nesta tese, verificou-se que o MOSA necessita de um hardware com maior poder de processamento. Dessa maneira, pode ser interessante ter duas placas de processamento embarcado a bordo da aeronave. A Figura 49 ilustra uma visão geral de como os sistemas MOSA e IFA poderiam trabalhar em conjunto utilizando duas placas de processamento embarcado. Apesar de não ter sido feito nenhum teste com duas placas, todo o sistema foi estruturado para que suporte trabalhar desse modo. Portanto, o presente trabalho avaliou apenas a separação lógica dos sistemas, já a separação física foi somente descrita.

Figura 49 – Separação de interesses dos sistemas MOSA e IFA em nível de hardware.

Fonte: Elaborada pelo autor.

7.7 Framework ProOF

A presente seção descreve brevemente o *framework* de otimização utilizado no desenvolvimento dos métodos de planejamento e replanejamento de rota. O *Professional Optimization Framework* (ProOF) é um ambiente que permite a codificação de problemas e métodos de otimização mono e multi-objetivo. O ProOF oferece suporte para o desenvolvimento de métodos exatos, heurísticos, meta-heurísticos e híbridos (ARANTES, 2014).

A Figura 50 apresenta a interface gráfica do ambiente de otimização ProOF. A interface,

após carregar o problema e o método programado, permite ao usuário definir a configuração dos parâmetros utilizados nos experimentos.

Figura 50 – *Framework* ProOF utilizado no desenvolvimento dos métodos.

Fonte: Elaborada pelo autor.

Os métodos de replanejamento de rota utilizados nesta tese foram todos programados dentro desse *framework*, na linguagem Java. Uma das grandes vantagens desse ambiente é o fato de possuir uma série de métodos de otimização e problemas já definidos. Outra vantagem é a facilidade de fazer testes com diferentes parâmetros de forma automatizada, o que acelera a geração de resultados. O código-fonte desse *framework* pode ser acessado no repositório[3].

7.8 Considerações Finais

Este capítulo apresentou brevemente os planejadores e replanejadores de rotas utilizados; o protocolo definido para especificação do mapa e da missão; e como foi feita a avaliação da arquitetura. A arquitetura aqui proposta é híbrida, pois possui características deliberativas e reativas, como explicado na Seção 2.13. A arquitetura é P&P para integração de diferentes componentes de hardware e software, como mostrado na Seção 6.3. A arquitetura faz a separação de interesse no nível lógico entre missão e segurança, como mostrado na Seção 6.3.

O capítulo seguinte apresentará alguns detalhes dos sistemas desenvolvidos.

[3] <https://github.com/marcio-da-silva-arantes/ProOF>

CAPÍTULO

8

SISTEMAS DESENVOLVIDOS

“ Descobrir consiste em olhar para o que todo mundo está vendo e pensar uma coisa diferente. ”

Roger Von Oech

8.1 Considerações Iniciais

O presente capítulo apresentará alguns detalhes sobre as implementações dos sistemas desenvolvidos durante esta tese. Os principais sistemas desenvolvidos são: UAV-Mission-Creator, UAV-Manager, UAV-GCS, UAV-S2DK, UAV-IFA e UAV-MOSA. As versões implementadas dos sistemas são caracterizadas pelo termo "UAV"antes do seu nome, por exemplo, o sistema UAV-MOSA é nossa implementação do sistema MOSA. Todos esses programas/sistemas integram o sistema autônomo proposto nesta tese, que será chamado de UAV-Toolkit e está disponível do repositório no Github [1] sobre a licença *General Public License* (GPL) v3.0.

8.2 Sistema UAV-Mission-Creator

O sistema UAV-Mission-Creator foi desenvolvido para auxiliar no projeto de missões e mapas utilizando a ferramenta Google Earth e encontra-se disponível no repositório [2]. Esse sistema foi desenvolvido em Java e é capaz de produzir um conjunto de arquivos de saída, que serão utilizados na plataforma autônoma. O sistema utiliza um arquivo com o mapa e o plano da missão, feito no Google Earth, conforme protocolo definido na Seção 7.4, e o transforma em um conjunto de arquivos necessários aos algoritmos de planejamento de caminho (HGA4m, CCQSP4m) e replanejamento de caminho (MPGA4s, GA4s, GH4s, DE4s, etc.). A ideia geral desse sistema pode ser melhor compreendida, analisando a Figura 51, que é baseada em arquivos de Entrada e Saída (E/S).

[1] <https://github.com/jesimar/UAV-Toolkit/>
[2] <https://github.com/jesimar/UAV-Toolkit/tree/master/UAV-Mission-Creator/>

Figura 51 – Esquema arquitetural do software UAV-Mission-Creator baseado em E/S.

Fonte: Elaborada pelo autor.

- Arquivos de Entrada:
  - **config.properties**: um arquivo de configuração para definição de um conjunto de propriedades importantes internas do UAV-Mission-Creator;
  - **mission.kml**: um arquivo com o mapa e o plano da missão, planejado utilizando o Google Earth.
- Arquivos de Saída:
  - **map.sgl**: um arquivo contendo a especificação do mapa com as regiões de obstáculos. Utilizado pelo algoritmo HGA4m;
  - **map-full.sgl**: um arquivo contendo a especificação do mapa com as regiões de obstáculos, bonificadores e penalizadores. Utilizado pelos algoritmos GA4s, DE4s, MPGA4s, GH4s, MS4s, GA-GA-4s e GA-GH-4s;
  - **wpts.txt**: um arquivo contendo a especificação do plano da missão. Utilizado pelo algoritmo HGA4m;
  - **mission-ccqsp.sgl**: um arquivo contendo a especificação do mapa e do plano da missão. Utilizado pelo algoritmo CCQSP4m;
  - **geoBase.txt** um arquivo contendo a especificação de um ponto de referência (base) em coordenadas geográficas utilizado na conversão entre os sistemas de coordenadas cartesianas e geográficas, e vice-versa. Utilizado pelos sistemas IFA e MOSA;
  - **featureMission.txt**: um arquivo contendo a especificações de ações a serem executadas durante a realização da missão, como: retirada de fotografias, filmagem, pulverização, acionamento do *buzzer*, entre outras. Utilizado pelo sistema MOSA.

Com base nos dois arquivos de entrada, acima especificados, pode-se produzir os seis arquivos de saída que serão utilizados, por sua vez, como entradas na plataforma autônoma (sistemas IFA e MOSA).

8.3 Sistema UAV-Manager

O UAV-Manager é um programa desenvolvido para facilitar a instalação e execução do sistema autônomo proposto nesta tese. Esse sistema foi desenvolvido em Java e encontra-se disponível no repositório [3]. A Figura 52 apresenta um resumo dos softwares que devem ser instalados para a execução do sistema autônomo aqui proposto. Este figura mapeia os softwares necessários dependendo da forma de execução da plataforma, que pode ser: SITL, HITL e REAL_FLIGHT. Pode-se perceber que alguns softwares devem ser instalados apenas na GCS e outros apenas no CC.

Figura 52 – Resumo dos softwares instalados para execução do sistema autônomo proposto.

Fonte: Elaborada pelo autor.

A Figura 53 apresenta três telas da interface gráfica (GUI, do inglês *Graphical User Interface*) do sistema UAV-Manager. A primeira, na Figura 53a, mostra como essa ferramenta auxilia na instalação dos softwares mínimos necessários à execução da plataforma autônoma. A Figura 53b possui uma tela em que se pode executar o sistema autônomo utilizando a técnica *Software-In-The-Loop*. Por fim, na Figura 53c, é mostrado um conjunto de terminais abertos com o sistema em execução.

Uma das principais vantagens dessa ferramenta é que ela facilita a instalação de alguns dos softwares necessários. Outra vantagem é que ela também facilita a execução do sistema autônomo, no entanto ela está disponível apenas para o modo de execução SITL. Para fazer experimentos HITL e voos reais, necessita-se executar o sistema autônomo através do terminal

[3] <https://github.com/jesimar/UAV-Toolkit/tree/master/UAV-Manager/>

Figura 53 – Interface gráfica do software UAV-Manager.

(a) Tela de instalação do sistema. (b) Tela de execução do sistema. (c) Tela com sistema executando.

Fonte: Elaborada pelo autor.

de comandos. A Figura 54 apresenta três maneiras diferentes (SITL, HITL e REAL_FLIGHT) de se executar o sistema através do terminal.

Figura 54 – Resumo das formas de execução do sistema autônomo proposto.

Fonte: Elaborada pelo autor.

8.4 Sistema UAV-GCS

O sistema UAV-GCS é uma Estação de Controle em Solo desenvolvida para acompanhamento, visualização e interação com os sistemas MOSA, IFA e o AP. Esse sistema foi desenvolvido em Java e encontra-se disponível no repositório [4]. Essa estação desenvolvida não tem por objetivo concorrer com outras GCS disponíveis, como o QGroundControl e o Mission Planner. O UAV-GCS é necessário, principalmente, para interagir com os sistemas MOSA e IFA, o que não é possível com as outras estações disponíveis no mercado. Dessa forma, o usuário do sistema pode executar outra estação de controle qualquer para acompanhar a execução da missão.

[4] <https://github.com/jesimar/UAV-Toolkit/tree/master/UAV-GCS/>

A Figura 55 mostra três interfaces GUI desse software. Na Figura 55a, vemos uma tela com os componentes de hardware utilizados no experimento. Essa tela ajuda no *debug* das configurações especificadas pelo usuário, antes de realizar o voo/simulação. Na Figura 55b, temos um modo de plotar o mapa da missão de maneira bastante simples. A Figura 55c é bastante semelhante a anterior, porém o mapa é plotado utilizando o API do Google Maps, o que permite ter uma ideia melhor do cenário/mapa ao fundo. Apesar de mais simples, o mapa da Figura 55b possui mais recursos de plotagem e navegação. As formas de plotagem de mapa das Figuras 55b e 55c são invertidas em relação ao eixo y.

Figura 55 – Interface gráfica do software UAV-GCS.

(a) Tela com *status* do hardware. (b) Tela com mapa simples. (c) Tela com Google Maps.

Fonte: Elaborada pelo autor.

O UAV-GCS apresenta, na esquerda da sua interface, um conjunto de recursos/funções disponíveis. Entre eles se destacam:

- Inserção de Falhas:
  - Simular falha baseada em condições de tempo ruim levando a um RTL;
  - Simular falha de bateria baixa levando a um pouso forçado com alocação de risco;
  - Simular falha no sistema IFA levando a um pouso na vertical.
- Comandos de Ação:
  - Retirar uma foto utilizando a câmera;
  - Retirar uma sequência de fotos utilizando a câmera;
  - Gravar um vídeo utilizando a câmera;
  - Acionar o apito utilizando o *buzzer*;
  - Acionar o alarme utilizando o *buzzer*;
  - Abrir o paraquedas;
  - Iniciar a pulverização;
  - Parar a pulverização;

  - Ligar o LED;
  - Piscar o LED de maneira intermitente.

- Troca de Comportamento:
  - Trocar o comportamento do VANT durante o voo baseado em arquivo de entrada;
  - Trocar o comportamento do VANT durante o voo para voar em formato de círculo;
  - Trocar o comportamento do VANT durante o voo para voar em formato de triângulo;
  - Trocar o comportamento do VANT durante o voo para voar em formato de quadrado.

- Formas de Controlar do VANT:
  - Utilizar comandos de teclado para controlar a aeronave;
  - Utilizar comandos de voz para controlar a aeronave.

Alguns dos recursos listados acima não estão disponíveis no menu esquerdo da Figura 55, pois o VANT não possuía alguns dos sensores ou atuadores necessários. Esse menu é construído de forma dinâmica, baseado nos componentes de hardware da aeronave.

Como mencionado anteriormente, um dos recursos disponíveis é a capacidade de controlar a aeronave através de comandos de teclado e voz. Um resumo dos comandos aceitos estão listados na Tabela 11. Esses recursos de comandos de teclado e voz não tornam a plataforma autônoma, apenas adicionam no sistema um modo diferente de executar uma missão controlada. Futuramente, esse recurso pode ser utilizado para controlar o VANT, tendo uma camada de segurança adicional em relação ao voos rádio controlados.

Tabela 11 – Conjunto de comandos de teclado e de voz permitidos para controlar o VANT.

Ação do Comando	Comando de Teclado (Teclar)	Comando de Voz (Dizer)	Fator de Deslocamento (Configurável)
decolar o VANT	*enter*	*takeoff*	3 metros
pousar o VANT	*backspace*	*land*	-
mover para cima	*page up*	*up*	1 metro
mover para baixo	*page down*	*down*	1 metro
mover para esquerda	seta para esquerda	*left*	3 metros
mover para direita	seta para direita	*right*	3 metros
mover para frente	seta para cima	*forward*	3 metros
mover para trás	seta para baixo	*back*	3 metros

Fonte: Elaborada pelo autor.

A Figura 56 detalha alguns dos recursos do UAV-GCS com enfase na comunicação com os sistemas UAV-IFA, UAV-MOSA, UAV-S2DK e Oracle Drone. O sistema Oracle Drone (OD) é uma plataforma na nuvem que recebe os dados de voo em tempo real e foi desenvolvido por Guidoti (2019). O OD funciona como uma GCS na internet permitindo que múltiplos usuários

utilizem essa estação simultaneamente. O UAV-GCS, caso configurado, pode salvar os dados de voo diretamente no Banco de Dados (BD) do OD na internet.

Figura 56 – Diagrama de comunicação do UAV-GCS com IFA, MOSA, S2DK e OD.

Fonte: Elaborada pelo autor.

A Tabela 12 mostra um conjunto de instâncias de mapas utilizados neste trabalho, algumas foram geradas artificialmente outras manualmente (instâncias reais). As instâncias artificiais foram geradas a partir da aplicação de um algoritmo baseado em Blackmore, Ono e Williams (2011), que encontra-se disponível nesse repositório [5]. As instâncias reais foram geradas de forma manual utilizando o Google Earth e a ferramenta UAV-Mission-Creator. Todas os mapas utilizados estão disponíveis no repositório [6]. Os arquivos KML utilizados para projetar o mapa e o plano da missão podem ser acessados no repositório [7].

Tabela 12 – Conjunto de instâncias de mapas utilizado nos experimentos.

Tipo de Mapa/Instância	Tipo de Planejador	Quantidade de Instâncias
Artificial	Planejamento de Rota	50
	Replanejamento de Rota	600
Real	Planejamento de Rota	9
	Replanejamento de Rota	10

Fonte: Elaborada pelo autor.

[5] <https://github.com/jesimar/UAV-Tools/tree/master/CreateMapArtificial/>
[6] <https://github.com/jesimar/UAV-Toolkit/tree/master/Instances/>
[7] <https://github.com/jesimar/UAV-Toolkit/tree/master/Missions/KML/>

Os arquivos de instâncias de mapas estão disponíveis em três formatos diferentes. São eles: .SGL, .JSON, .XML.

- **SGL**: formato de entrada utilizado pelo *Framework* ProOF. Esse é o padrão de mapa mais leve e compacto, todavia também é o mais difícil de entender;
- **JSON**: formato de entrada do mapa é, em geral, fácil de entender e ler. Esse padrão é mais pesado que o SGL, ocupa em média o quádruplo do SGL em disco (HD);
- **XML**: formato de entrada do mapa é, em geral, fácil de ler. Esse padrão é mais pesado que o SGL e JSON, em geral, ocupa o dobro do tamanho do JSON em disco (HD).

Os trabalhos desenvolvidos por Li (2010), Blackmore, Ono e Williams (2011), Ono, Williams e Blackmore (2013), Arantes (2016), Arantes (2017), Ariu *et al.* (2017) utilizam apenas instâncias artificiais, com polígonos quadriláteros regulares, e simulação SITL na sua avaliação. O sistema autônomo aqui proposto permite avaliar todos os algoritmos desenvolvidos nesses trabalhos, mesmo que a sua forma de geração de instâncias tenha sido feita de modo diferente. Nesse sentido, a Figura 57 mostra duas missões desenvolvidas em Arantes (2017), Arantes (2016). A Figura 57a mostra uma missão que foi portada do trabalho Arantes (2017) para o sistema autônomo aqui proposto. Já Figura 57b mostra uma missão que foi portada do trabalho Arantes (2016) para o nosso sistema autônomo. Através desse recurso permite-se fazer novas avaliações de missões anteriormente desenvolvidas utilizando técnicas como SITL, HITL e REAL_FLIGHT.

Figura 57 – Simulações utilizando instâncias de mapas artificiais.

(a) Tela com simulação sobre mapa artificial baseado em missão de Arantes (2017).

(b) Tela com simulação sobre mapa artificial baseado no problema de Arantes (2016).

Fonte: Elaborada pelo autor.

8.5 Sistema UAV-S2DK

A Seção 6.3 deu uma visão geral do sistema *Services to DroneKit* (S2DK). Esse sistema provê um conjunto de serviços para o MOSA e o IFA. Esses serviços permitem, em geral, acessar/enviar dados ao AP, como destacado na Figura 34. Os serviços mencionados são acessados através do protocolo HTTP *request* utilizando métodos GET e POST. Esse serviço não pode iniciar uma interação com os sistemas MOSA e IFA, ele apenas responde à requisições feitas por tais sistemas, assemelhando-se a um servidor *web*. A presente seção descreve com mais detalhes as requisições que podem ser feitas ao UAV-S2DK que foi desenvolvido na linguagem Python.

Quadro 6 – Interface de requisições GET entre os sistemas MOSA/IFA e o S2DK.

Requisição	Exemplo de Resposta JSON	Informações dos Campos
get-gps	{"gps": [-22.0059726, -47.8986881]}	[latitude, longitude]
get-barometer	{"barometer": [0.0, 870.0]}	[alt_rel, alt_abs]
get-battery	{"bat": [12.587, 0.0, 100]}	[tensão, corrente, nível]
get-attitude	{"att": [-0.0018, 1.9062, -0.0016]}	[pitch, yaw, roll]
get-velocity	{"vel": [-0.02, -0.05, 0.0]}	[vel_x, vel_y, vel_z]
get-gpsinfo	{"gpsinfo": [3, 10, 121, 65535]}	[fix, n_sat., eph, epv]
get-heading	{"heading": 110}	unidade: º
get-groundspeed	{"groundspeed": 2.21}	unidade: m/s
get-airspeed	{"airspeed": 1.53}	unidade: m/s
get-mode	{"mode": "STABILIZE"}	tipo: string
get-system-status	{"system-status": "STANDBY"}	tipo: string
get-armed	{"armed": false}	tipo: booleano
get-is-armable	{"is-armable": true}	tipo: booleano
get-ekf-ok	{"ekf-ok": true}	tipo: booleano
get-next-waypoint	{"next-waypoint": 5}	tipo: inteiro
get-count-waypoint	{"count-waypoint": 28}	tipo: inteiro
get-distance-to-home	{"distance-to-home": 6.50}	unidade: m
get-dist-to-wpt-curr	{"distance-to-wpt-current": 4.32}	unidade: m
get-home-location	{"home-location": [-22.0059726, -47.8986881, 870.0]}	[lat, lng, alt_abs]
get-parameters	{"parameters": "Key:WPNAV_RADIUS Value:80.0; Key:WPNAV_SPEED_DN Value:150.0; Key:RTL_ALT_FINAL Value:0.0; ... Key:BATT_CURR_PIN Value:12.0; "}	-
get-all-sensors	{"all-sensors": [-22.00597, -47.89869, 0.0, 870.0, 0.72, 1.93, 0.02, 109, 0.0, 0.0, 3, 10, 121, 65535, [-0.01, -0.04, 0.05], 0, 0, 0.1, 3.5, "STABILIZE", "STANDBY", false, true, true]}	-

Fonte: Elaborada pelo autor.

O Quadro 6 apresenta um conjunto de requisições (usando GET), que podem ser pedidos pelos sistemas MOSA e IFA ao sistema S2DK. Na primeira coluna temos o nome da requisição

e na segunda coluna temos um exemplo de resposta obtida em formato JSON. A terceira coluna apresenta algumas informações sobre a requisição.

Quadro 7 – Interface de requisições/ações POST entre os sistemas MOSA/IFA e o S2DK.

Requisição/Ação	Exemplo de Argumento JSON	Informações dos Campos
set-mission	{"mission":[{"action":"takeoff", "lat":0.0, "lng":0.0, "alt":2.0}, {"action":"goto", "lat":-22.00587424, "lng":-47.89874454, "alt":3.0}, {"action":"goto", "lat":-22.00587325, "lng":-47.89854124, "alt":5.0}]}	{[ação, lat, lng, alt_rel]}
append-mission	{"mission":[{"action":"goto", "lat":-22.00582424, "lng":-47.89874454, "alt":3.0}, {"action":"goto", "lat":-22.00587325, "lng":-47.89855124, "alt":5.0}, {"action":"goto", "lat":-22.00604778, "lng":-47.89854005, "alt":2.0}, {"action": "landv", "lat":0.0, "lng":0.0, "alt":0.0}]}	{[ação, lat, lng, alt_rel]}
set-waypoint	{"waypoint":{"action":"takeoff", "lat":0.0, "lng":0.0, "alt":2.0}}	[ação, lat, lng, alt_rel]
append-waypoint	{"waypoint":{"action":"goto", "lat":-22.00604778, "lng":-47.89853005, "alt":2.0}}	[ação, lat, lng, alt_rel]
set-parameter	{"parameter":{"key":"RTL_-ALT","value":800.0}}	[chave, valor]
set-heading	{"heading":{"value":193, "typeDirection":"ccw", "typeAngle":"relative"}}	unidade: º
set-velocity	10.0	tipo: double
set-mode	"GUIDED"	tipo: string

Fonte: Elaborada pelo autor.

O Quadro 7 mostra o conjunto de ações (usando POST), que pode ser efetuado pelos sistemas MOSA e IFA ao S2DK. Na primeira coluna temos o nome da ação requerida e na segunda coluna temos um exemplo de argumento, especificado em formato JSON, necessário a realização da ação. A terceira coluna apresenta algumas informações sobre a requisição.

8.6 Sistema UAV-IFA

Uma implementação do sistema IFA foi desenvolvida, chamada UAV-IFA, e está disponível no repositório [8]. Esse sistema foi desenvolvido em Java e provê uma camada de segurança às missões executadas. Dentre as bibliotecas utilizadas nesse sistema, se destaca a Lib-UAV. Essa biblioteca foi desenvolvida durante a presente tese e contém um conjunto de funções

[8] <https://github.com/jesimar/UAV-Toolkit/tree/master/UAV-IFA/>

comuns (genéricas) aos sistemas UAV-IFA, UAV-MOSA e UAV-UAV-GCS. As ideias centrais da Lib-UAV são a reusabilidade de código e separação de conceitos.

Inicialmente, quando o sistema IFA é executado, um conjunto de parâmetros de voo do AP é definido pelo IFA. Esses parâmetros estão especificados no arquivo, chamado config-param.properties, que está disponível no Apêndice A. Alguns exemplos de parâmetros que podem ser alterados são: `RTL_ALT`, `WPNAV_RADIUS`, `WPNAV_SPEED`, `WPNAV_SPEED_UP` e `WPNAV_SPEED_DN`.

Dois arquivos de *logs* são gerados, em formato Valores Separados por Vírgula (CSV, do inglês *Comma-Separated Values*) quando o sistema IFA executa. Esses arquivos são o log-aircraft.csv e o log-overhead-ifa.csv. Um exemplo de cada um deles pode ser encontrado no Apêndice A.

O sistema IFA calcula algumas métricas, durante o voo, de forma a dar a aeronave uma maior inteligência/autonomia em suas tomadas de decisão. A Equação 8.1 retorna a distância euclidiana na horizontal do VANT até o local de decolagem (*home*). Nessa equação, o numerador apresenta a distância medida em graus, mesma unidade dos argumentos de entrada. Através a divisão pelo fator $\mathcal{F}$, garante-se a transformação de graus para metros. A Equação 8.2 dá o tempo estimado para fazer a operação de RTL. A Equação 8.3 retorna o consumo de bateria estimado para fazer uma operação de RTL. A Equação 8.4 apresenta a distância máxima estimada que o VANT consegue voar com a atual quantidade de bateria. A Equação 8.5 apresenta o tempo máximo estimado que o VANT consegue voar tendo em vista sua atual quantidade de bateria. Os tempos T^E_{rtl} e T^E_{max} são medidos em segundos. As distâncias D_{hrz} e D^E_{max} são medidas em metros. O consumo de combustível estimado em B^E_{rtl} é medido em porcentagem da bateria.

$$D_{hrz} = \frac{1}{\mathcal{F}} \cdot \sqrt{(p^{uav}_{lat} - p^{home}_{lat})^2 + ((p^{uav}_{lng} - p^{home}_{lng}) \cdot cos(\frac{\pi \cdot p^{home}_{lat}}{180})^2} \tag{8.1}$$

$$T^E_{rtl} = (1 + \mathcal{E}_1) \cdot \left(\frac{|p^{uav}_{altrel} - h_{rtl}|}{v_{up}} + \frac{D_{hrz}}{v_{hrz}} + \frac{|h_{rtl} - p^{home}_{altrel}|}{v_{down}} \right) \tag{8.2}$$

$$B^E_{rtl} = |p^{uav}_{altrel} - h_{rtl}| \cdot \mathcal{K}^E_{N_{up}} + D_{hrz} \cdot \mathcal{K}^E_{N_{hrz}} + |h_{rtl} - p^{home}_{altrel}| \cdot \mathcal{K}^E_{N_{down}} \tag{8.3}$$

$$D^E_{max} = \mathcal{E}_2 \cdot \frac{v_{hrz} \cdot B_{level}}{\mathcal{K}^E_{N_{hrz}}} \tag{8.4}$$

$$T^E_{max} = \frac{B_{level}}{\mathcal{K}^E_T} \tag{8.5}$$

Os valores definidos das constantes $\mathcal{E}$, $\mathcal{K}^E_T$ e $\mathcal{K}^E_N$ foram obtidos durante experimentos reais e simulados conduzidos durante esse tese. Esses valores foram encontrados para as aeronaves

iDroneAlpha e iDroneBeta. Caso o VANT utilizado seja muito diferente das aeronaves descritas, devido a aspectos físicos e elétricos, novos valores devem ser calculados para que as estimativas sejam corretas. Os valores D^E_{max} e T^E_{max} estão linearmente relacionados, ou seja, quanto maior for o tempo de voo maior será a distância alcançada.

Tabela 13 – Valores das constantes do VANT utilizadas nas equações.

Constante		Valor	Unidade de Medida	Tipo de Voo	Descrição
	$\mathcal{F}$	$0,000009$	°/metro	Real e Sim.	Fator de conversão de metros para graus
$\mathcal{E}$	$\mathcal{E}_1$	$0,20$	-	Real e Sim.	Percentagem de tempo extra aferida na execução de RTL
	$\mathcal{E}_2$	$0,70$	-	Real e Sim.	Percentagem da distância máxima de alcance no voo
$\mathcal{K}^E_T$	$\mathcal{K}^E_T$	$0,151$	% de bateria/segundo	Real	Eficiência energética média de tempo de voo
	$\mathcal{K}^E_T$	$0,216$	% de bateria/segundo	Simulado	
$\mathcal{K}^E_N$	$\mathcal{K}^E_{N_{hrz}}$	$0,077$	% de bateria/metro	Real	Eficiência energética média da navegação horizontal
	$\mathcal{K}^E_{N_{hrz}}$	$0,110$	% de bateria/metro	Simulado	
	$\mathcal{K}^E_{N_{up}}$	$0,481$	% de bateria/metro	Real	Eficiência energética média da navegação vertical para cima
	$\mathcal{K}^E_{N_{up}}$	$0,500$	% de bateria/metro	Simulado	
	$\mathcal{K}^E_{N_{down}}$	$0,337$	% de bateria/metro	Real	Eficiência energética média da navegação vertical para baixo
	$\mathcal{K}^E_{N_{down}}$	$0,450$	% de bateria/metro	Simulado	

Fonte: Elaborada pelo autor.

Tabela 14 – Parâmetros do piloto automático utilizados nas equações e nos experimentos.

Parâmetro	Valor	Unidade de Medida	Descrição
v_{hrz}	$2,00$	metros/segundo	Velocidade de deslocamento horizontal
v_{up}	$0,50$	metros/segundo	Velocidade de deslocamento vertical para cima
v_{down}	$0,50$	metros/segundo	Velocidade de deslocamento vertical para baixo
h_{rtl}	20 (em geral)	metros	Altitude de realização do RTL

Fonte: Elaborada pelo autor.

Variáveis:

- p^{uav}_{lat}: posição da latitude atual do VANT;
- p^{uav}_{lng}: posição da longitude atual do VANT;
- p^{uav}_{altrel}: posição da altitude relativa atual do VANT;
- p^{home}_{lat}: posição da latitude do ponto de decolagem do VANT (*home*);
- p^{home}_{lng}: posição da longitude do ponto de decolagem do VANT (*home*);
- p^{home}_{altrel}: posição da altitude relativa do ponto de decolagem do VANT (*home*), em geral é 0 metro;

- B_{level}: nível atual da bateria do VANT.

A Equação 8.6 pode ser utilizada para converter um valor de arco em graus terrestre para metros. Essa equação é uma derivação simples da equação da circunferência, $C = 2 \cdot \pi \cdot R$. Entretanto, não há interesse no comprimento total da circunferência, mas em uma fração dela ($\frac{\tau}{360}$). Na Equação 8.6, L é o comprimento do arco da circunferência (em metros), $\mathcal{R}_T$ é o raio do círculo (raio da Terra em metros), τ é o ângulo percorrido na circunferência (em graus). A constante $\mathcal{F}$ que converte os valores de metros para graus, e vice-versa, foi obtida utilizando o raio da Terra, igual a $\mathcal{R}_T = 6.366.197$ metros, veja a Equação 8.7.

$$L = \pi \cdot \mathcal{R}_T \cdot \frac{\tau}{180} \tag{8.6}$$

$$\mathcal{F} = \frac{180}{\pi \cdot \mathcal{R}_T} = \frac{90^\circ}{10.000.000\ metros} \tag{8.7}$$

Os valores das variáveis T^E_{rtl}, B^E_{rtl}, D^E_{max} e T^E_{max} são calculados constantemente pelo sistema IFA, dentro do módulo que monitora a segurança da aeronave. Caso uma condição de tempo ruim ocorra e o sistema IFA decida acionar o mecanismo de RTL, como ilustrado na Figura 36, a seguinte condição $B_{level} > B^E_{rtl}$ é verificada. Essa condição analisa se o nível atual da bateria é maior que o nível estimado de bateria necessária para fazer o RTL, caso seja então a operação de RTL é normalmente acionada. Todavia, caso essa condição não seja satisfeita, uma mensagem dizendo que não há bateria suficiente é reportada ao piloto em Terra e a ação de pouso na vertical é executada automaticamente. Esse mecanismo de segurança foi implementado de maneira a impedir que a aeronave caia durante a execução de um RTL em que o sistema já havia, previamente, verificado não ser possível de concluir.

8.7 Sistema UAV-MOSA

Uma implementação do sistema MOSA, chamada UAV-MOSA, foi desenvolvida e encontra-se disponível no repositório [9]. Ela foi desenvolvida em Java e objetiva executar toda a missão especificada pelo projetista. Dentre as bibliotecas utilizadas nesse sistema, se destaca a Lib-UAV.

Um arquivo de *log* é gerado em formato CSV quando o sistema MOSA executa. Esse arquivo é o log-overhead-mosa.csv e um exemplo dele pode ser encontrado no Apêndice A.

Todos os planejadores e replanejadores utilizados retornam uma rota discreta formada por um conjunto finito de pontos. No entanto, a rota seguida pela aeronave é uma trajetória contínua formada por infinitos pontos. Quanto maior o fator de discretização utilizado, melhor podemos

[9] <https://github.com/jesimar/UAV-Toolkit/tree/master/UAV-MOSA/>

aproximar a rota contínua por uma rota discreta. Um alto fator de discretização, em geral, leva os algoritmos a um alto custo computacional, como demostrado em Arantes (2017). Um problema não relacionado ao algoritmo planejador de rotas é a capacidade de armazenamento de *waypoints* pelo AP. A APM suporta até 166 *waypoints*, já a Pixhawk suporta até 718 *waypoints*. Dessa maneira, a Pixhawk suporta rotas com até 4,3 vezes mais *waypoints* que a APM.

Outro fator que limita os pilotos automáticos no quesito quantidade de *waypoints* é o tempo de escrita/armazenamento dos mesmos no AP. Em um estudo feito nesta tese, verificou-se que o tempo de escrita aumenta linearmente com a quantidade de *waypoints*. Devido a esses dois fatores limitantes, capacidade de armazenamento e tempo de escrita dos *waypoints*, um método simplificador de *waypoints* foi implementado. O simplificador, de forma geral, remove os pontos colineares baseado em um limiar.

8.8 Considerações Finais

Este capítulo apresentou os principais sistemas desenvolvidos durante esta tese de doutorado. Os sistemas implementados possuem como características: ser *plug and play*; multiplataforma (testado em Linux, Windows e Mac OS X); suportar arquitetura x86 e ARM; suportar os pilotos automáticos APM e Pixhawk; suportar os *companion computers* Raspberry Pi, Intel Edison e BeagleBone Black; suportar missões com MOSA não-adaptativo e adaptativo. A Tabela 15 resume algumas informações sobre esses programas. Em uma análise direta sobre a memória consumida por todo o sistema autônomo, verificou-se o consumo de cerca de 90 a 100 MB de RAM nas placas Raspberry Pi, Intel Edison e BeagleBone Black. Com relação ao gasto no disco rígido (HD) pelo sistema autônomo, necessita-se de pelo menos 57 MB de espaço livre executando do repositório [10]. Com relação ao processamento sobre os CC, uma análise direta revelou que a aplicação que mais consome processamento é o MavProxy, seguido por UAV-S2DK, UAV-IFA e UAV-MOSA, esses dois últimos sem estar executando os (re)planejadores.

Tabela 15 – Resumo dos principais códigos desenvolvidos neste trabalho.

Sistema/Programa	Linguagem	Classes	Pacotes	Métodos	Linhas de Código
UAV-Mission-Creator	Java	21	7	182	2.972
UAV-Manager	Java	9	3	44	1.301
UAV-GCS	Java	33	11	180	7.387
UAV-S2DK	Python	7[11]	1[12]	44	788
UAV-IFA	Java	18	6	87	3.416
UAV-MOSA	Java	10	5	75	2.804
Lib-UAV	Java	87	12	558	8.751
Total	-	185	45	1.170	27.419

Fonte: Elaborada pelo autor.

[10] <https://github.com/jesimar/UAV-Embedded/>

O próximo capítulo apresentará os resultados obtidos.

[11] Leia-se arquivos e não classes
[12] Leia-se pasta e não pacote

CAPÍTULO

9

RESULTADOS

“ Você nunca sabe que resultados virão da sua ação. Mas se você não fizer nada, não existirão resultados. ”
Mohandas Karamchand Gandhi

9.1 Considerações Iniciais

O presente capítulo apresentará os resultados computacionais obtidos durante as etapas de experimentação. A Seção 9.2, mostra os resultados dos experimentos desenvolvidos e publicados no *Genetic and Evolutionary Computation Conference* (GECCO), em 2017. A Seção 9.3 reporta os resultados experimentais desenvolvidos e publicados no *IEEE International Conference on Tools with Artificial Intelligence* (ICTAI), em 2017. A Seção 9.4 expõe os resultados obtidos e publicados no *International Council of the Aeronautical Sciences* (ICAS), em 2018. A Seção 9.5 mostra os resultados computacionais desenvolvidos e submetidos ao *Expert Systems With Applications* (ESWA), em 2019. Uma descrição mais detalhada pode ser encontrada nos artigos anexados a este documento. Por fim, alguns experimentos finais estão reportados na Seção 9.6.

9.2 Resultados do Artigo do GECCO 2017

Esta seção discute os resultados computacionais obtidos em dois experimentos desenvolvidos e publicados no GECCO 2017. O artigo está disponível no Apêndice B. O primeiro experimento objetiva avaliar o desempenho do método *Hybrid Genetic Algorithm for mission* (HGA4m) e do *Multi-Population Genetic Algorithm for security* (MPGA4s) quando executados sobre as plataformas de hardware Intel Edison e Intel i5. Os modelos dos métodos HGA4m e do MPGA4s foram brevemente apresentados no Capítulo 5. O segundo experimento é um estudo de caso mais específico, em que simulações são conduzidas sobre a arquitetura do sistema proposto descrita no Capítulo 6. Os resultados descritos nesta seção foram baseados na aeronave Ararinha e simulados utilizando o Rascal 110, cujas configurações estão na Tabela 1. A Tabela 16 mostra

os valores dos parâmetros para o HGA4m e MPGA4s utilizados. Esses valores são os mesmos valores utilizados em Arantes *et al.* (2016), Arantes *et al.* (2015).

Tabela 16 – Configurações utilizadas nos métodos HGA4m e MPGA4s.

Parâmetro	Valor HGA4m	Valor MPGA4s
número de populações	3	3
tamanho da população	3×13	3×13
taxa de crossover	5	$0,5$
taxa de mutação	$0,7$	$0,75$
critério de parada ($\mathcal{T}_{SC}$)	10 segundos	1 segundo
elitismo	Sim	Sim

Fonte: Elaborada pelo autor.

Utilizando o gerador proposto em Blackmore, Ono e Williams (2011), 40 instâncias de mapas foram gerados aleatoriamente. Esses mapas foram empregados nos experimentos com HGA4m, em que rotas para o planejamento da missão foram criadas. Os arquivos de instância dos mapas utilizados no planejador estão disponíveis no *link* [1]. A avaliação do método MPGA4s foi conduzida sobre um conjunto de 60 mapas, em que havia 10 mapas para cada tipo de instância, como descrito em Arantes *et al.* (2015). Os arquivos de instância dos mapas utilizados no replanejador estão disponíveis no *link* [2]. Os primeiros resultados foram obtidos após a execução do HGA4m por 10 vezes sobre cada um dos 40 mapas. O HGA4m foi executado em duas plataformas de hardware. A primeira delas foi a plataforma de computação Intel Edison com processador *dual-core* com 500 MHz, 1 GB de memória RAM e sistema operacional Linux - Yocto. A segunda foi um Computador Pessoal (PC) regular com processador Intel i5 com 1.8 GHz, 4 GB de memória RAM, sistema operacional Linux - Ubuntu 16.04. Duas métricas principais são comparadas: número de avaliações de *fitness* e tamanho da rota retornado. A Figura 58 mostra o número de avaliações de *fitness* médios sobre 10 execuções em cada instância.

Conforme esperado, o número de avaliações é reduzido quando se executa o HGA4m sobre a Edison. Isso significa que o HGA4m explora menos o espaço de busca ao executar sobre um CC do que ao ser executado em um PC regular. Entretanto, a qualidade da solução em termos de tamanho da rota é praticamente a mesma, como mostrado pela Figura 59. O tamanho da rota retornada com a Edison foi similar a alcançada sobre a plataforma PC, o que indica um consumo de combustível/bateria equivalente. Logo, a segunda métrica indica que a qualidade das rotas retornadas é mantida quando o HGA4m é executado sobre uma plataforma de hardware inferior.

O teste não-paramétrico de Kruskal-Wallis foi realizado para medir estatisticamente os resultados das Figuras 58 e 59, como mostrado na Tabela 17. Os critérios de independência,

[1] <https://github.com/jesimar/UAV-Toolkit/tree/master/Instances/Artificial/Path-Planning/GECCO-2017>

[2] <https://github.com/jesimar/UAV-Toolkit/tree/master/Instances/Artificial/Path-Replanning/GECCO-2017>

Figura 58 – Número de avaliações por instância no método HGA4m.

Fonte: Elaborada pelo autor.

Figura 59 – Tamanho da rota por instância no método HGA4m.

Fonte: Elaborada pelo autor.

homocedasticidade e normalidade devem ser satisfeitos para a aplicação de testes paramétricos. Se pelo menos um critério não for satisfeito, devemos utilizar testes não paramétricos. O teste de Kolmogorov-Smirnov foi realizado e mostrou que os dados das Figuras 58 e 59 não seguem uma distribuição normal. Há uma diferença significativa entre os resultados retornados pelo PC i5 e pela Edison, como pode ser visto por $p\text{-}value < 0,0001$ em relação ao critério número de avaliação. No entanto, não há diferença significativa detectada quanto ao critério de tamanho da rota.

A seguir é avaliado o MPGA4s sobre ambas plataformas, PC i5 e Edison, para resolver um conjunto de 60 mapas. A Figura 60 mostra o número de avaliações (avaliações da função de *fitness*).

Tabela 17 – Teste de Kruskal-Wallis para o método HGA4m.

Métrica	Arquitetura	*p-value*
Número de Avaliações	PC i5 Edison	$< 0,0001$
Tamanho da Rota	PC i5 Edison	$0,185$

Fonte: Elaborada pelo autor.

Figura 60 – Número de avaliações por instância no método MPGA4s.

Fonte: Elaborada pelo autor.

Um total de 805 avaliações em média da função de *fitness* é executado pelo MPGA4s na plataforma Edison, enquanto 34.440 avaliações ocorrem sobre PC i5. A plataforma PC executa aproximadamente 42 vezes mais avaliações do que a Edison, mas a qualidade do desempenho mantém-se satisfatória, como mostra a Figura 61.

O local de pouso pode ser pensado como uma maneira direta de medir a qualidade da solução. A Figura 61 ilustra os locais onde o pouso ocorreu. Esse local é obtido através das rotas do MPGA4s de ambas as arquiteturas avaliadas. Existem seis conjuntos de instâncias (I1, I2, $\cdots$ I6), em que o MPGA4s é avaliado levando em conta quatro situações críticas descritas em Arantes *et al.* (2015): falha no motor (Ψ_m), superaquecimento da bateria (Ψ_b), problema na semi-asa direita (Ψ_{s1}) e problema na semi-asa esquerda (Ψ_{s2}). Um total de 10 mapas diferentes foram avaliados. A cor azul indica o local de pouso em região bonificadora, cor branca indica pouso em região remanescente e cor cinza escuro indica violação da restrição de não navegabilidade. Essa imagem revela uma grande similaridade entre os locais de pouso para as rotas retornadas em ambas arquiteturas de hardware.

A Tabela 18 apresenta numericamente os resultados da Figura 61. Pode-se observar que cerca de 75,4% das soluções pousaram em uma região segura para rotas retornados pela Edison, contra 85,8% na plataforma PC. As soluções do PC executaram apenas 10,4% melhor do que as

Figura 61 – Local de pouso de ambas as arquiteturas de hardware para o MPGA4s.

Fonte: Elaborada pelo autor.

soluções embarcadas e o número de violações da condição de não navegabilidade nas rotas de ambas as plataformas é praticamente o mesmo.

Os resultados acima revelam que uma arquitetura de hardware, como a Edison, consegue salvar o VANT em 75,4% das situações críticas que causariam sua inevitável queda. Os resultados também mostram que, mesmo que fosse possível configurar todo o hardware de uma plataforma de PC na aeronave, a melhora alcançada seria de cerca de 10,4% nas chances de salvar a aeronave.

Tabela 18 – Resumo dos locais de pouso do MPGA4s em ambas as arquiteturas de hardware.

Região de Pouso	PC i5	Edison
bonificadora	206 (85,8%)	181 (75,4%)
remanescente	19 (7,9%)	43 (17,9%)
penalizadora	0 (0,0%)	0 (0,0%)
no-fly zone	15 (6,2%)	16 (6,6%)
total	240 (100%)	240 (100%)

Fonte: Elaborada pelo autor.

O teste de Kruskal-Wallis foi realizado para medir estatisticamente os resultados da Figura 60 e os resultados são mostrados na Tabela 19. O teste não paramétrico foi justificado,

uma vez que esses dados também não seguem uma distribuição normal, informação obtida pelo teste de Kolmogorov-Smirnov. Existe uma diferença significativa entre o desempenho do MPGA4s quando se compara as plataformas de hardware.

Tabela 19 – Teste de Kruskal-Wallis para o método MPGA4s

Métrica	Arquitetura	*p-value*
Avaliações	PC i5 Edison	$< 0,0001$

Fonte: Elaborada pelo autor.

Um estudo de caso também foi avaliado, em que uma aplicação do mundo real para planejamento de missão e segurança é conduzida. Nesse experimento, o desempenho do sistema é também comparado quando executado sobre as plataformas Edison e PC i5. A Figura 62a mostra os resultados obtidos.

Figura 62 – Resultado do estudo de caso em um cenário do mundo real no Campus 2-USP.

(a) Rotas obtidas pelo HGA4m e MPGA4s. (b) Rota obtida pelo HGA4m utilizando SITL.

Fonte: Elaborada pelo autor.

A rota completa planejada com o método HGA4m, em que o VANT deve executar todo o plano da missão, é representada por uma linha verde. Essa rota é extremamente similar em ambas as arquiteturas e não apresenta diferenças significativas. Uma análise dessa rota mostra que a missão é executada sem violação das regiões não navegáveis.

A Figura 62a também mostra pontos rotulados por **F1**, **F2**, **F3**, **F4** e **F5**, que são lugares onde uma falha crítica foi detectada. O sistema proposto é avaliado quando uma falha ocorre

Tabela 20 – Diferentes simulações efetuadas utilizando SITL para validar as rotas.

Arquitetura	Método Avaliado	Descrição	*Link Web*
PC i5	HGA4m	Rota completa sem falha	<https://youtu.be/D22r8qZ4Wmo>
Edison	HGA4m	Rota completa sem falha	<https://youtu.be/O-xMC51w5ec>
PC i5	HGA4m e MGPA4s	Rota parcial com falha em **F5**	<https://youtu.be/Vzt6sZgFhL0>
Edison	HGA4m e MGPA4s	Rota parcial com falha em **F5**	<https://youtu.be/rTtg3ANmNvw>

Fonte: Elaborada pelo autor.

em cada um desses pontos. O superaquecimento da bateria é assumido como falha, forçando o sistema IFA a realizar um pouso de emergência. Nesse caso, uma linha azul indica a rota emergencial retornada pela Edison. A linha vermelha indica a rota emergencial retornada pela plataforma PC. Todas as rotas são capazes de trazer com segurança a aeronave ao solo, pousando sobre regiões bonificadoras, sem voar sobre regiões não navegáveis.

Um conjunto de simulações SITL foi utilizado para validar as rotas, em que o hardware do piloto automático é simulado. A Figura 62b apresenta os resultados obtidos quando a missão é executada sem qualquer falha. As trajetórias seguidas pelo VANT durante essas simulações apresentam um pequeno desvio da rota planejada pelo HGA4m.

Os vídeos, na Tabela 20, mostram quatro de doze experimentos conduzidos utilizando SITL. É possível ver que a interação entre os módulos embarcados funciona adequadamente e que a diferença no poder de processamento entre a Edison e PC não compromete os experimentos. A explicação para esse fato é que os métodos HGA4m e MPGA4s são técnicas de IA, baseadas em computação evolutiva, que consomem pouco processamento e são capazes de encontrar soluções próximas do ótimo em um curto tempo computacional.

9.3 Resultados do Artigo do ICTAI 2017

Esta seção discute os resultados computacionais obtidos nos experimentos desenvolvidos e publicados no ICTAI 2017. O artigo está disponível no Apêndice C. Esses experimentos avaliam o desempenho de quatro métodos (GH4s, GA4s, GA-GH-4s e GA-GA-4s) executados sobre as plataformas de hardware Intel Edison e Intel i5. Um estudo de caso foi avaliado através de simulações SITL sobre a arquitetura do sistema proposto. Os experimentos foram projetados e calibrados com base no VANT de asa fixa Ararinha e simulados com o Rascal 110, cujas configurações estão descritas na Tabela 1.

As configurações do GA4s são baseadas em Arantes *et al.* (2017a) e estão descritas na Tabela 21. Nesses experimentos, duas situações críticas foram avaliadas: problema na bateria (Ψ_b) e no motor (Ψ_m).

Um total de 30 mapas artificiais foi avaliado e pode ser acessado no *link*[3]. Os métodos

[3] <https://github.com/jesimar/UAV-Toolkit/tree/master/Instances/Artificial/Path-Replanning/

Tabela 21 – Configurações utilizadas nos métodos HGA4m e MPGA4s.

Parâmetro	Valor GA4s
tamanho da população	39
tamanho do torneio	3
taxa de crossover	$0,5$
taxa de mutação	$0,7$
critério de parada ($\mathcal{T}_{SC}$)	tempo
elitismo	Sim

Fonte: Elaborada pelo autor.

são avaliados em duas plataformas de hardware diferentes que são: Intel Edison e Intel i5. A comparação entre duas arquiteturas de computador diferentes é relevante, uma vez que os resultados obtidos pelo Intel i5 podem ser vistos como um limite superior para a qualidade da solução do computador embarcado. As estratégias baseadas em computação evolutiva são executadas 10 vezes para cada mapa, enquanto o GH4s é executado apenas uma vez, visto que é uma heurística determinística.

A Tabela 22 apresenta os resultados alcançados para cada abordagem (GH4s, GA4s, GA-GH-4s e GA-GA-4s). Esses resultados dão as taxas de sucesso, quando a aeronave efetua um pouso seguro (pouso dentro de uma região bonificadora). Também é mostrado o tempo utilizado no critério de parada (parâmetro $\mathcal{T}_{SC_R}$) do algoritmo genético, em que a execução da GH4s não foi previamente configurada com limite de tempo, pois é uma heurística determinística de rápida execução. As colunas Ψ_b e Ψ_m representam a taxa de pouso com sucesso diante uma falha na bateria e no motor, respectivamente. A coluna Média indica a média das colunas Ψ_b e Ψ_m. A coluna Tempo indica a duração do processamento em milissegundos (ms). Nessa tabela a cor verde salienta os melhores resultados, a cor vermelha os piores, a cor azul e negrito destacam alguns valores mencionados no texto.

Os resultados dos métodos, que executam no computador Intel i5, são competitivos com base nos valores relatados. O VANT conseguiu pousar com segurança, em média de 83,2% até 88,8% dos casos, exceto pela GH4s que pousa a aeronave 73,3% das vezes. Por outro lado, a GH4s precisa apenas de 41 milissegundos, em média, para retornar um novo caminho. Além disso, o aumento no limite de tempo melhora os resultados globais, apenas 5,6% (de 83,2% até 88,8%), mesmo quando executado quatro vezes mais (de 250 ms até 1000 ms).

Nos resultados obtidos com a Intel Edison, a taxa de sucesso diminui um pouco e o tempo computacional aumenta conforme esperado. A GH4s mantém sua qualidade, pois é uma heurística determinista. No entanto, o tempo de execução aumenta de 41 ms na Intel i5 para 347 ms na Intel Edison, o que significa uma perda de desempenho em torno de 8,4 vezes. As taxas de sucesso dos outros métodos mudam de 67,3% para 82,3%, indicando uma diferença em torno de

Tabela 22 – Resultados obtidos após avaliar diferentes estratégias em mapas artificiais.

		PC - Intel i5				CC - Intel Edison			
Métodos	$\mathcal{T}_{SC_R}$ (ms)	Ψ_b	Ψ_m	Média	Tempo	Ψ_b	Ψ_m	Média	Tempo
GH4s	N/A	86,7%	60,0%	73,3%	41	86,7%	60,0%	**73,3%**	347
GA4s	250	96,3%	72,0%	84,2%	250	72,3%	62,3%	67,3%	250
GA4s	500	99,0%	73,0%	86,0%	500	84,7%	65,3%	75,0%	500
GA4s	1000	98,3%	75,7%	87,0%	1000	91,0%	68,0%	79,5%	1000
GA-GH-4s	250	95,3%	71,0%	**83,2%**	250	89,3%	62,7%	**76,0%**	309
GA-GH-4s	500	100,0%	72,3%	86,2%	500	90,3%	65,7%	78,0%	510
GA-GH-4s	1000	99,3%	76,0%	87,7%	1000	92,3%	69,0%	80,7%	1000
GA-GA-4s	250	99,3%	72,3%	85,8%	250	81,0%	65,0%	73,0%	250
GA-GA-4s	500	99,7%	73,3%	86,5%	500	89,7%	68,3%	79,0%	500
GA-GA-4s	1000	99,7%	78,0%	88,8%	1000	94,7%	70,0%	82,3%	1000
Média	-	97,4%	72,4%	84,9%	-	87,2%	65,6%	76,4%	-

Fonte: Elaborada pelo autor.

15% com base na estratégia, bem como o limite de tempo associado. As estratégias baseadas em comitê superam a GH4s para a maioria dos casos, enquanto elas são melhores do que GA4s quando comparados dentro do mesmo limite de tempo de execução. A estratégia GA-GH-4s retorna uma taxa de pouso bem sucedido em 76,0% em pouco tempo (250 ms), enquanto a estratégia GA-GA-4s tem melhores taxas de sucesso variando de 500 ms para 1000 ms.

O teste não paramétrico de Kruskal-Wallis foi aplicado para avaliar a diferença significativa entre os resultados reportados na Tabela 23. Em primeiro lugar, é feita uma comparação entre as taxas de sucesso obtidas pelos métodos executados com os computadores Intel i5 e Edison. Os resultados mostram uma diferença significativa, uma vez que p-value $< 0,05$. A segunda análise compara se há uma diferença significativa entre as taxas de sucesso reportadas para falhas de bateria e motor. Há também uma diferença significativa entre o desempenho dos métodos, quando se lida com falha de bateria e motor.

Tabela 23 – Teste de Kruskal-Wallis para arquitetura e situação crítica.

Métrica	Atributo	*p-value*	Grupos	
Arquitetura	Intel i5	$0,003$	A	
	Intel Edison			B
Situação Crítica	Falha na Bateria	$< 0,0001$	A	
	Falha no Motor			B

Fonte: Elaborada pelo autor.

A Tabela 24 apresenta os resultados do teste de Dunn para comparação múltipla. Os piores métodos são GA4s e GA-GA-4s, com 250 ms, e GA4s, com 500 ms em execução na Edison, enquanto não há diferença significativa entre as outras estratégias. Assim, com base na avaliação estatística feita, não podemos garantir uma diferença significativa entre as taxas de sucesso ao executar a maioria das estratégias utilizando um *companion computer*, como Intel

Edison ou um computador pessoal, como Intel i5.

Tabela 24 – Teste de Dunn para comparação múltipla em mapas artificiais.

Arquitetura	Métodos	$\mathcal{T}_{SC_R}$ (ms)	Grupos			
Intel Edison	GA4s	250	D			
	GA-GA-4s	250	D	C		
	GA4s	500	D	C	B	
	GH4s	-		C	B	A
	GA-GH-4s	250		C	B	A
	GA4s	1000		C	B	A
	GA-GH-4s	500		C	B	A
	GA-GA-4s	500		C	B	A
	GA-GH-4s	1000		C	B	A
	GA-GA-4s	1000		C	B	A
Intel i5	GH4s	-		C	B	A
	GA4s	250		C	B	A
	GA-GH-4s	250		C	B	A
	GA4s	500			B	A
	GA-GA-4s	250			B	A
	GA-GH-4s	500				A
	GA-GA-4s	500				A
	GA4s	1000				A
	GA-GH-4s	1000				A
	GA-GA-4s	1000				A

Fonte: Elaborada pelo autor.

Um conjunto de simulações foi executado a partir de um estudo de caso sobre um cenário para imageamento aéreo. A Figura 63 mostra a rota seguida pelo VANT durante a missão. Existem várias falhas rotuladas como **F1**, **F2**, **F3** e **F4** que foram simuladas, em que o sistema IFA detectou essas falhas. Em seguida, o algoritmo de replanejamento é executado pelo limite de tempo definido anteriormente como critério de parada. Esse tempo faz com que o VANT percorra a distância crítica d_c. Nesses experimentos, a velocidade do VANT no instante das falhas é de 24 m/s e, após um segundo (como limite de tempo), a aeronave atingiu $d_c = 24$ metros para longe do ponto da falha. A nova rota é enviada para o piloto automático e a missão atual é interrompida.

As rotas brancas, na Figura 63, representam o caminho replanejado para a falha da bateria, enquanto as rotas pretas representam a falha do motor. Essas trajetórias foram determinadas aplicando a estratégia GA-GA-4s. A aeronave foi capaz de pousar em uma região bonificadora em quase todos os casos em que a única exceção foi a falha no motor (**F4**).

Em seguida, um total de 10 execuções é feito para cada método e cada falha e os resultados são mostrados na Tabela 25. O método com pior desempenho é o GH4s, que salva a aeronave apenas 12,5%. Os outros métodos retornam soluções melhores, principalmente quando executados por 1000 ms. Suas taxas de sucesso atingem valores de 71,3% a 75,0%. De acordo com esses resultados, o método com a melhor taxa de sucesso é o GA-GA-4s.

A Tabela 26 apresenta algumas das simulações utilizando SITL para validar as rotas, geradas pelo GA-GA-4s, após uma falha crítica na bateria. Essa estratégia foi executada na Intel

Figura 63 – Resultado do estudo de caso em um cenário do mundo real utilizando o GA4s.

Fonte: Elaborada pelo autor.

Tabela 25 – Resultados obtidos após avaliar diferentes estratégias no estudo do caso.

		Intel i5				Intel Edison			
Métodos	$\mathcal{T}_{SC_R}$ (ms)	Ψ_b	Ψ_m	Média	Tempo	Ψ_b	Ψ_m	Média	Tempo
GH4s	N/A	25,0%	0,0%	12,5%	22	25,0%	0,0%	12,5%	192
GA4s	250	100,0%	50,0%	75,0%	250	35,0%	37,5%	36,3%	250
GA4s	500	100,0%	50,0%	75,0%	500	62,5%	50,0%	56,3%	500
GA4s	1000	100,0%	67,5%	83,8%	1000	95,0%	50,0%	72,5%	1000
GA-GH-4s	250	100,0%	50,0%	75,0%	250	37,5%	37,5%	37,5%	234
GA-GH-4s	500	100,0%	50,0%	75,0%	500	67,5%	50,0%	58,8%	481
GA-GH-4s	1000	100,0%	75,0%	87,5%	1000	92,5%	50,0%	**71,3%**	992
GA-GA-4s	250	100,0%	50,0%	75,0%	250	50,0%	45,0%	47,5%	250
GA-GA-4s	500	100,0%	50,0%	75,0%	500	85,0%	50,0%	67,5%	500
GA-GA-4s	1000	100,0%	75,0%	87,5%	1000	100,0%	50,0%	75,0%	1000
Média Final	-	92,5%	51,8%	72,1%	-	65,0%	42,0%	53,5%	-

Fonte: Elaborada pelo autor.

Edison. As simulações mostram que o piloto automático do VANT é capaz de seguir a rota de pouso emergencial enviada pelo método.

Tabela 26 – Diferentes simulações efetuadas utilizando SITL para validar as rotas do GA4s.

Simulação	Descrição	*Link Web*
Simulação 1	Falha crítica na bateria em **F1**	<https://youtu.be/lPYJ6nVCBRs>
Simulação 2	Falha crítica na bateria em **F2**	<https://youtu.be/k38GBjM5jXU>
Simulação 3	Falha crítica na bateria em **F3**	<https://youtu.be/SQebmNOnUkg>
Simulação 4	Falha crítica na bateria em **F4**	<https://youtu.be/E6L2kQBF-ps>
Simulação 5	Rota completa sem falha crítica	<https://youtu.be/DxWcQVyJtFQ>

Fonte: Elaborada pelo autor.

9.4 Resultados do Artigo do ICAS 2018

Esta seção discute os resultados computacionais obtidos nos experimentos desenvolvidos e publicados no ICAS 2018. O artigo está disponível no Apêndice D. Nesse trabalho, foi implementada uma Arquitetura Orientada a Serviços (SOA), junto com os sistemas MOSA e IFA, de forma a dar à aeronave maior autonomia de voo. Esse trabalho apresenta uma versão desses sistemas com destaque para a comunicação entre eles. Esses resultados avaliam o desempenho da nossa arquitetura ao executar os sistemas MOSA e IFA sobre diferentes tipos de falhas, como as falhas esquematizadas na Figura 36.

Um conjunto de experimentos simulados foi realizado para avaliar nossa arquitetura. Eles foram calibrados com base em um quadricóptero. O cenário avaliado possui dimensões de 50m x 36m. As simulações são executadas utilizando a técnica *Software-In-The-Loop* (SITL).

O computador utilizado nos experimentos é um Intel i7 com 2.50 GHz, 16 GB de RAM e sistema operacional Linux - Ubuntu 17.04. Os métodos de planejamento/replanejamento de rotas utilizados foram HGA4m e MPGA4s, desenvolvidos em Arantes *et al.* (2016), Arantes *et al.* (2015), respectivamente. Os parâmetros dos métodos de planejamento/replanejamento de rotas utilizados são, praticamente, os mesmos reportados na Tabela 16. O HGA4m executou por um tempo limite de 4,0 segundos (parâmetro $\mathcal{T}_{SC_P}$), e o risco assumido de violar uma NFZ pelo método foi de 1% (parâmetro Δ_P). O MPGA4s executou por um tempo limite de 1,0 segundo (parâmetro $\mathcal{T}_{SC_R}$), fazendo uso de até 25 *waypoints* (parâmetro $\mathcal{W}_R$) e o risco assumido pelo método foi de 1% (parâmetro Δ_R). A altitude de cruzeiro da missão foi de 3,0 metros (parâmetro M_{alt}).

Nesse experimento, será considerada a missão mostrada na Figura 64a, em que a arquitetura criada será avaliada sem considerar a ocorrência de falha crítica. Nesse estudo, cinco simulações foram realizadas e suas rotas são apresentadas nessa figura. As rotas em vermelho representam as rotas calculadas pelo método HGA4m. É interessante notar que entre os pontos P_6 e P_7, duas rotas passaram entre os obstáculos e três rotas passaram abaixo dos obstáculos. Observe na imagem que algumas rotas estão sobrepostas. As rotas azuis representam a trajetória percorrida de fato pela aeronave (obtida através do GPS) tentando seguir a rota planejada em vermelho. Em geral, os dispositivos de GPS possuem um erro de precisão associado a cada

tipo de equipamento e quantidade de satélites capturados. Dessa forma, embora esse dispositivo forneça uma localização específica, a mesma deve ser questionada. Em geral, os aparelhos GPS apresentam um erro menor que 1,5 metros de raio. A Figura 64b mostra um *flow tube*, representando uma região de incerteza com 1,5 metros de raio, em que o VANT poderá estar localizado. Analisando essa figura, percebemos que durante toda a trajetória, mesmo com o erro do GPS a aeronave não viola os obstáculos nem a NFZ.

Figura 64 – Resultados obtidos no estudo de caso com simulação SITL.

(a) Resultado sem falha crítica. (b) Resultado com *flow tube* de erro do GPS.

Fonte: Elaborada pelo autor.

As falhas descritas na Figura 36 que possuem o diagnóstico *status* AP *Critical*, *Emergency* e *PowerOff*, e *status* do GPS diferente de 3D e 4D *Fix* apesar de implementadas não foram avaliadas. A justificativa para isso é a dificuldade da inserção dessas falhas em ambiente simulado. Já as falhas: bateria baixa, falha no MOSA, falha no IFA e condição de tempo ruim ao voo foram avaliadas três vezes cada.

A primeira falha crítica estudada está mostrada na Figura 65a. A estrela representa o local em que houve falha crítica. Dessa maneira, podemos ver que a aeronave abortou a rota da missão e nos três casos avaliados o VANT pousou na área segura (base), fazendo o desvio dos obstáculos.

Outro experimento avaliou a detecção de falha crítica no sistema MOSA. A tomada de decisão feita para esse tipo de falha é executar um replanejamento de rotas no ar abortando a missão original, uma vez que o sistema MOSA apresente alguma falha interna. A Figura 65b mostra três simulações de falhas do sistema MOSA em diferentes locais da rota. Em apenas um dos casos avaliados, a aeronave não conseguiu pousar sobre a área segura.

De forma similar ao experimento anterior, algumas falhas no sistema IFA foram analisadas. Nesse caso, a tomada de decisão deveria ser abrir o paraquedas, uma vez que não se poderia efetuar nenhum replanejamento de rota, já que o IFA apresentou problemas. A Figura 66a mostra três simulações, em que nos pontos F_1, F_2 e F_3, ocorreu uma falha crítica no sistema IFA. O

Figura 65 – Resultados obtidos após o IFA detectar falha e fazer o pouso emergencial.

(a) Resultados com falha de bateria baixa. (b) Resultados com falha no sistema MOSA.

Fonte: Elaborada pelo autor.

pouso da aeronave, utilizando o dispositivo de paraquedas, ocorreu logo após a detecção da falha crítica pelo sistema.

Um último experimento envolvendo falhas é mostrado na Figura 66b. Nós podemos observar as trajetórias percorridas pela aeronave após o sistema IFA ser informado sobre a ocorrência de condições meteorológicas ruins ao voo. Desse modo, foi disparada a ação de retornar a base (RTL). Nesse exemplo, a altura configurada do RTL foi de 8 metros, o que fez que o VANT fosse capaz de passar por cima de obstáculos de 5 metros de altura, encontrados no cenário.

Figura 66 – Resultados obtidos após o IFA detectar uma falha crítica.

(a) Resultados com falha no sistema IFA e então abrir o paraquedas. (b) Resultados com condições ruins ao voo e então fazer o RTL.

Fonte: Elaborada pelo autor.

Algumas das simulações descritas anteriormente nas Figuras 64 e 66b podem ser assisti-

das através de vídeos destacados na Tabela 27.

Tabela 27 – Diferentes simulações realizadas utilizando SITL para validar a arquitetura.

Métodos Avaliados	Descrição	*Link Web*
HGA4m	Rota completa sem falha crítica	<https://youtu.be/kh_mH3KcHe4>
HGA4m e MPGA4s	Rota parcial com bateria baixa	<https://youtu.be/WQm3tn7gMs4>
HGA4m e MPGA4s	Rota parcial com falha no MOSA	<https://youtu.be/SYJopMU1Ehc>
HGA4m e Abrir Paraquedas	Rota parcial com falha no IFA	<https://youtu.be/-nawxwiTkiY>
HGA4m e RTL	Rota parcial com tempo ruim	<https://youtu.be/aYOpoKobXmk>

Fonte: Elaborada pelo autor.

Juntamente com o estudo da arquitetura, um estudo sobre o tempo de escrita (gravação) de *waypoints* (rota) no AP foi feito. A Tabela 28 apresenta os resultados desse estudo, em que se percebe que o tempo de gravação cresce linearmente com o número de *waypoints*. Essa análise revela que, em geral, gasta-se cerca de 0,1 segundo para setar/armazenar um *waypoint* no AP. A quantidade máxima de *waypoints* suportados no piloto automático Pixhawk é 718.

Tabela 28 – Síntese dos resultados avaliando o tempo de escrita dos *waypoints* no AP.

	PC - Intel i7		CC - Raspberry Pi 3	
Nº de *Waypoints*	Tempo	Tempo/*Waypoint*	Tempo	Tempo/*Waypoint*
5 *waypoints*	0,7 segundo	0,140	0,6 segundo	0,120
10 *waypoints*	1,1 segundo	0,110	1,0 segundo	0,100
20 *waypoints*	2,1 segundos	0,105	2,0 segundos	0,100
40 *waypoints*	3,8 segundos	0,095	3,9 segundos	0,097
80 *waypoints*	7,3 segundos	0,091	6,8 segundos	0,085
160 *waypoints*	14,9 segundos	0,093	18,2 segundos	0,114
320 *waypoints*	32,8 segundos	0,103	31,2 segundos	0,097
640 *waypoints*	62,9 segundos	0,098	64,3 segundos	0,100
718 *waypoints*	73,5 segundos	0,102	80,5 segundos	0,112
Média	-	0,104	-	0,103

Fonte: Elaborada pelo autor.

9.5 Resultados do Artigo do ESWA 2019

Esta seção discute os resultados computacionais obtidos nos experimentos desenvolvidos e submetidos ao ESWA 2019. O artigo está disponível no Apêndice E. Esses experimentos avaliam o desempenho completo da plataforma autônoma, executando sobre diferentes computadores de bordo e pilotos automáticos.

Dois tipos de experimentos foram conduzidos na avaliação desse trabalho, o primeiro baseado em simulação e o segundo baseado em voos reais. As simulações são executadas utilizando a técnica *Hardware-In-The-Loop* (HITL), com um total de 50 experimentos realizados. Para os voos reais, foram utilizados os quadricópteros apresentados nas Figuras 29a e 29b, em

que oito experimentos foram realizados. Os *companion computers* utilizados nos experimentos possuem as seguintes configurações:

- Placa Raspberry Pi 3: processador *quad-core* ARM Cortex-A53 com 1.2 GHz, 1 GB de RAM e sistema operacional Linux - Raspbian;
- Placa Intel Edison: processador *dual-core* Intel Atom com 500 MHz, 1 GB de RAM e Linux - Yocto;
- Placa BeagleBone Black: processador *dual-core* ARM Cortex-A8 com 1.0 GHz, 512 MB de RAM e Linux - Debian.

O computador utilizado na GCS é um notebook Dell Inspiron com processador Intel i7, 2.50 GHz, 16 GB de RAM e Linux - Ubuntu. O algoritmo *Branch and Cut* (B&C), disponível no *solver* IBM ILOG CPLEX 12.2, é utilizado pelos métodos de planejamento de rotas HGA4m e CCQSP4m. O CPLEX foi instalado no computador de solo e na placa Intel Edison, enquanto a Raspberry Pi e a BeagleBone Black não têm o CPLEX instalado, pois suas arquiteturas são ARM.

A Figura 67 mostra uma visão geral de quatro diferentes cenários avaliados nesse trabalho. Case-I ilustra uma missão simples na qual o VANT decola, segue uma missão pré-definida, retirando um conjunto de fotografias, e, finalmente, retorna ao ponto inicial onde pousa. Nesse *case* não existem obstáculos, mas temos duas regiões atrativas ao pouso. Case-II mostra uma missão com um conjunto de obstáculos artificiais a serem evitados, em que a aeronave deve iniciar o voo em um ponto e aterrissar em outro, realizando fotografias na região. Nesse *case* todo o planejamento de rotas deve ser feito pelo sistema embarcado. Case-III apresenta uma missão com um grande obstáculo real a ser evitado e duas regiões bonificadoras. Case-IV ilustra uma missão mais complexa, na qual o sistema precisará mudar seu comportamento durante o voo. Nesse *case*, o cenário tem dois obstáculos a serem evitados e duas regiões cênicas a serem alcançadas. Um conjunto de situações críticas foram inseridas para todos os *cases*, então o comportamento de reação do sistema é avaliado.

A Figura 68 mostra os resultados para o Case-I. Um total de 12 experimentos foram realizados com 10 simulações HITL e dois voos reais. O VANT deve iniciar o voo no ponto-alvo **A** e alcançar os pontos-alvo **B**, **C**, **D**, e assim por diante, até finalizar em **K**. Dois CCs foram avaliados: Raspberry Pi 3 e Intel Edison. A Figura 68a mostra as rotas de voos sem falhas críticas. Por outro lado, a Figura 68b apresenta as rotas após uma falha de bateria ter sido detectada pelo sistema IFA, assim, o VANT deve pousar o mais rápido possível. A missão é abortada e o sistema IFA define um pouso de emergência, executando o algoritmo DE4s. As estrelas na Figura 68b marcam os pontos em que a falha foi detectada. A aeronave se desloca um pouco até que a nova rota seja calculada. A Figura 68c possui rotas quando uma condição de tempo ruim é simulada. Nessa situação, o sistema IFA decide executar o comando RTL seguindo os

Figura 67 – Visão geral dos quatro cenários utilizados na validação de arquitetura.

Fonte: Elaborada pelo autor.

procedimentos ilustrados na Figura 36. Em cinco de seis falhas de bateria simuladas, o sistema IFA conseguiu aterrissar o VANT dentro de uma região bonificadora (veja a Figura 68b). A Equação 9.1 dá um exemplo das configurações da ênupla $\langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle$ empregadas por um dos 12 experimentos realizados para o Case-I.

Figura 68 – Resultados das rotas obtidas no cenário Case-I.

(a) Resultado sem falha crítica. (b) Resultado com bateria baixa. (c) Resultado com tempo ruim.

Fonte: Elaborada pelo autor.

$$
CaseI \therefore \langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle = \begin{cases} \mathcal{M} = \begin{cases} M_{map} = \langle \mathbf{r}, \mathbf{h}, \mathbf{c} \rangle & \text{veja Figura 68} \\ M_{goal} = \{A, B, C, D, E, F, G, H, I, J, K\} \\ M_{alt} = 5.0 \end{cases} \\ \mathcal{H} = \begin{cases} H_{ap} = APM \\ H_{cc} = IntelEdison \\ H_{sensor} = [CameraRGB] \\ H_{actuator} = [Buzzer] \end{cases} \\ \mathcal{S} = \begin{cases} S_{mosa} = \langle \mathcal{P}_P, \mathcal{L}_P, \mathcal{T}_{SC_P}, \Delta_P \rangle = \langle FixedRoute4m, onboard, \varnothing, \varnothing \rangle \\ S_{ifa} = \langle \mathcal{P}_R, \mathcal{L}_R, \mathcal{T}_{SC_R}, \Delta_R, \mathcal{W}_R \rangle = \langle DE4s, onboard, 1.0, 1.0\%, 30 \rangle \end{cases} \end{cases} \tag{9.1}
$$

A Figura 69 mostra os resultados para o Case-II. Nesse cenário, 18 experimentos foram conduzidos no total com 16 experimentos executando simulações HITL e dois voos reais. Nesse cenário, o VANT deve iniciar o voo no ponto-alvo **A** e passar para os pontos-alvo **B**, **C**, **D** e **E**. Dois CCs foram avaliados: Raspberry Pi 3 e BeagleBone Black. As Figuras 69a e 69c mostram as rotas após um voo sem falhas críticas. Nas Figuras 69a e 69b, o planejador CCQSP4m foi acionado, enquanto o HGA4m é executado nas Figuras 69c e 69d. As Figuras 69b e 69d possuem as rotas quando uma falha no AP é detectada pelo sistema IFA. Nesse caso, a rota planejada é abortada e o VANT faz um pouso vertical. As estrelas sinalizam os pontos em que a falha foi detectada. A Equação 9.2 traz as configurações $\langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle$ utilizadas em um dos 18 experimentos realizados para o Case-II.

$$
CaseII \therefore \langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle = \begin{cases} \mathcal{M} = \begin{cases} M_{map} = \langle \mathbf{r}, \mathbf{h}, \mathbf{c} \rangle & \text{veja Figura 69} \\ M_{goal} = \{A, B, C, D, E\} \\ M_{alt} = 15.0 \end{cases} \\ \mathcal{H} = \begin{cases} H_{ap} = APM \\ H_{cc} = RaspberryPi \\ H_{sensor} = [CameraRGB] \\ H_{actuator} = [Buzzer] \end{cases} \\ \mathcal{S} = \begin{cases} S_{mosa} = \langle \mathcal{P}_P, \mathcal{L}_P, \mathcal{T}_{SC_P}, \Delta_P \rangle = \langle HGA4m, offboard, 4.0, 2.0\% \rangle \\ S_{ifa} = \langle \mathcal{P}_R, \mathcal{L}_R, \mathcal{T}_{SC_R}, \Delta_R, \mathcal{W}_R \rangle = \langle \varnothing, onboard, \varnothing, \varnothing, \varnothing \rangle \end{cases} \end{cases} \tag{9.2}
$$

A Figura 70 mostra os resultados para o Case-III. Um total de 18 experimentos foram realizados, sendo 16 com simulações HITL e dois com voos reais. O VANT deve iniciar o voo no ponto-alvo **A** e seguir para os pontos-alvo **B** e **C**, pousando em **A**. Dois CCs foram avaliados: Raspberry Pi 3 e Intel Edison. As Figuras 70a e 70d mostram as rotas obtidas a partir do voo sem

Figura 69 – Resultados das rotas obtidas no cenário Case-II.

(a) Resultado sem falha. $\mathcal{P}_P = CCQSP4m$.

(b) Resultado com falha no AP. $\mathcal{P}_P = CCQSP4m$.

(c) Resultado sem falha. $\mathcal{P}_P = HGA4m$.

(d) Resultado com falha no AP. $\mathcal{P}_P = HGA4m$.

Fonte: Elaborada pelo autor.

falhas críticas. Nas Figuras 70a, 70b e 70c, o planejador CCQSP4m foi executado, enquanto que o HGA4m é aplicado nos resultados das Figuras 70d, 70e e 70f. As Figuras 70b, 70c, 70e e 70f têm rotas após uma falha de bateria ter sido simulada. A rota planejada é abortada pelo sistema IFA e o VANT faz um replanejamento utilizando o algoritmo MPGA4s. Nas figuras, as estrelas sinalizam os pontos em que foi detectada a falha. Em 12 de 13 detecções de falhas de bateria, o sistema IFA foi capaz de aterrissar o VANT dentro da região bonificadora. A Equação 9.3 dá as configurações $\langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle$ para um dos 18 experimentos do Case-III.

Figura 70 – Resultados das rotas obtidas no cenário Case-III.

(a) Resultado sem falha crítica. $\mathcal{P}_P = CCQSP4m$.
(b) Resultado com bateria baixa. $\mathcal{P}_P = CCQSP4m$.
(c) Resultado com bateria baixa. $\mathcal{P}_P = CCQSP4m$.
(d) Resultado sem falha crítica. $\mathcal{P}_P = HGA4m$.
(e) Resultado com bateria baixa. $\mathcal{P}_P = HGA4m$.
(f) Resultado com bateria baixa. $\mathcal{P}_P = HGA4m$.

Fonte: Elaborada pelo autor.

$$
CaseIII \therefore \langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle = \begin{cases} \mathcal{M} = \begin{cases} M_{map} = \langle \mathbf{r}, \mathbf{h}, \mathbf{c} \rangle & \text{veja Figura 70} \\ M_{goal} = \{A, B, C, D\} \\ M_{alt} = 20.0 \end{cases} \\ \mathcal{H} = \begin{cases} H_{ap} = Pixhawk \\ H_{cc} = IntelEdison \\ H_{sensor} = [CameraRGB] \\ H_{actuator} = [Buzzer] \end{cases} \\ \mathcal{S} = \begin{cases} S_{mosa} = \langle \mathcal{P}_P, \mathcal{L}_P, \mathcal{T}_{SC_P}, \Delta_P \rangle = \langle CCQSP4m, onboard, 30.0, 2.0\% \rangle \\ S_{ifa} = \langle \mathcal{P}_R, \mathcal{L}_R, \mathcal{T}_{SC_R}, \Delta_R, \mathcal{W}_R \rangle = \langle MPGA4s, onboard, 1.0, 1.0\%, 30 \rangle \end{cases} \end{cases} \tag{9.3}
$$

A Figura 71 mostra os resultados para o Case-IV. Nesse cenário, 10 experimentos foram

realizados sendo oito simulações HITL e dois voos reais. O VANT deve iniciar o voo no ponto-alvo **A**, seguindo para os pontos-alvo **B**, **C**, **D** e **E**. Esse cenário avalia a mudança de comportamento do VANT durante a missão, utilizando o sistema gerenciador de missão, MOSA adaptativo. Primeiro, o VANT voa de **A**, evitando o obstáculo, para atingir o ponto **B**. No entanto, uma vez próximo ao ponto **B**, o sistema MOSA é informado pela GCS que algumas fotos devem ser tiradas no ponto **B**. Nesse ponto, o sistema MOSA reage e altera o comportamento anterior (vá direto para o ponto **C**), planejando uma nova rota descrita como uma forma circular em torno da região cênica 1. Ao concluir o novo comportamento, a aeronave se dirige para o ponto **C**, em que outro comportamento é enviado pela GCS. Nesse caso, o VANT deve executar, por exemplo, uma pulverização de inseticida seguindo uma trajetória específica (zigue-zague) na região cênica 2. Finalmente, ao chegar em **D**, uma nova rota é calculada para evitar obstáculos e aterrissar no ponto **E**. Dois CC foram avaliados: Raspberry Pi 3 e BeagleBone Black.

Nesse exemplo, dois comportamentos novos foram inseridos durante a realização da missão, tornando-a mais dinâmica. O sistema MOSA adaptativo permite que a missão possa ser alterada em tempo real. Nesse cenário, o algoritmo HGA4m foi utilizado para fazer o planejamento de rotas entre os pontos-alvo **A** e **B** e entre o **D** e **E**. Por se tratarem de comportamentos bem específicos, como mover-se em formato de círculo e fazer um zigue-zague para pulverização, dois outros algoritmos/técnicas foram projetadas.

A Figura 71a tem as rotas após um voo sem falhas críticas. A Figura 71b mostra as rotas quando uma condição de tempo ruim é detectada pelo sistema IFA e um RTL é executado. O RTL não planeja um caminho com alocação de risco, portanto, isso permite que o VANT sobrevoe o obstáculo. Durante o RTL, a aeronave atinge a altitude necessária para retornar diretamente e com segurança ao ponto de partida.

A Figura 71c apresenta as rotas após uma falha no piloto automático e um pouso vertical ser executado. Finalmente, a Figura 71d mostra as rotas quando ocorre uma falha no sistema MOSA. Diante disso, o sistema IFA assume o controle e replaneja o caminho utilizando o algoritmo GA4s. Pode ser visto na Figura 71d que uma das rotas não conseguiu pousar o VANT em uma região bonificadora. A Equação 9.4 tem as configurações $\langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle$ aplicadas em um dos 10 experimentos realizados para o Case-IV.

Figura 71 – Resultados das rotas obtidas no cenário Case-IV.

(a) Resultado sem falha crítica.

(b) Resultado com tempo ruim.

(c) Resultado com falha no AP.

(d) Resultado com falha no MOSA.

Fonte: Elaborada pelo autor.

$$
CaseIV \therefore \langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle = \begin{cases} \mathcal{M} = \begin{cases} M_{map} = \langle \mathbf{r}, \mathbf{h}, \mathbf{c} \rangle & \text{veja Figura 71} \\ M_{goal} = \{A, B, C, D, E\} \\ M_{alt} = 12.0 \end{cases} \\ \mathcal{H} = \begin{cases} H_{ap} = Pixhawk \\ H_{cc} = RaspberryPi \\ H_{sensor} = [CameraRGB] \\ H_{actuator} = [Buzzer] \end{cases} \\ \mathcal{S} = \begin{cases} S_{mosa} = \langle \mathcal{P}_P, \mathcal{L}_P, \mathcal{T}_{SC_P}, \Delta_P \rangle = \langle HGA4m, offboard, 4.0, 2.0\% \rangle \\ S_{ifa} = \langle \mathcal{P}_R, \mathcal{L}_R, \mathcal{T}_{SC_R}, \Delta_R, \mathcal{W}_R \rangle = \langle GA4s, onboard, 1.0, 1.0\%, 30 \rangle \end{cases} \end{cases} \tag{9.4}
$$

As trajetórias simuladas ou reais são próximas à planejada para todos os casos. Essas rotas foram obtidas utilizando dados dos voos reais realizados para cada um dos cenários. O

link[4] contém um vídeo com alguns dos voos realizados. A Figura 72 mostra uma visão 3D da rota para cada um desses casos.

Figura 72 – Trajetória 3D seguida pelo VANT.

Fonte: Elaborada pelo autor.

A Tabela 29 resume os resultados obtidos. Algumas medidas são relatadas, tais como: a distância do VANT até os obstáculos durante as simulações e os voos reais. Essa medida específica dá uma ideia sobre o nível de risco incorrido ao executar a rota planejada. Outro valor importante é a distância crítica percorrida pelo VANT, que significa a distância entre o ponto em que uma falha foi detectada e o local em que o VANT começa a executar a nova rota. Este valor dá uma ideia sobre o *overhead* entre o tempo de processamento da rota de emergência pelo sistema IFA e o tempo para escrevê-la/armazená-la no piloto automático.

A Tabela 30 resume o tempo de processamento dos métodos de planejamento/replanejamento utilizados. Todos os métodos de replanejamento foram executados a bordo da aeronave no CC (*onboard*). Os métodos de planejamento CCQSP4m e HGA4m foram executados *onboard* (*online*) quando a placa utilizada era a Intel Edison. Todavia, eles foram executados *offboard* na estação de controle de solo (GCS) quando as placas utilizadas eram Raspberry Pi ou BeagleBone Black. Um dos dados avaliados nessa tabela é o limite de tempo para a execução dos métodos de planejamento/replanejamento de rota. Outra medida na Tabela 30 é o tempo de processamento para encontrar a rota e enviá-la ao piloto automático.

Em geral, a placa com melhor desempenho ao analisar os replanejadores é a Raspberry Pi, seguida pela Intel Edison e pela BeagleBone Black. A execução mais rápida é alcançada pelo CCQSP4m quando se trata de planejadores. Os métodos de planejamento de caminho são

[4] <https://youtu.be/8vswz-1E8fA>

Tabela 29 – Resumo dos experimentos realizados por estudo de caso.

	Descrição	Case-I	Case-II	Case-III	Case-IV
Cenário	Dimensões dos cenários	125 m × 90 m	130 m × 95 m	140 m × 150 m	110 m × 75 m
	Número de obstáculos ($\|\Phi^o\|$)	0	5	1	2
	Número de regiões bonificadoras ($\|\Phi^b\|$)	2	2	3	2
	Número de regiões cênicas ($\|\Phi^s\|$)	0	0	0	2
	Altura dos obstáculos	-	20 m	25 m	15 m
HITL	Número de simulações HITL	10	16	16	8
	Comprimento total da rota planejada	462 m	215 m	379 m	324 m
	Tempo médio de simulação	226 s	171 s	240 s	225 s
	Distância mínima entre a rota planejada e obstáculo	-	1,5 m	2,7 m	1,9 m
	Distância mínima entre a posição do VANT e obstáculo	-	1,6 m	2,5 m	1,8 m
	Desvio máximo entre a rota do VANT e a rota planejada	1,0 m	0,8 m	0,9 m	0,9 m
Voo Real	Número de experimentos	2	2	2	2
	Comprimento total da rota planejada	462 m	215 m	379 m	324 m
	Tempo médio do experimento	228 s	175 s	316 s	267 s
	Distância mínima entre a rota planejada e obstáculo	-	1,6 m	2,7 m	1,9 m
	Distância mínima entre a posição do VANT e obstáculo	-	1,6 m	2,6 m	1,7 m
	Desvio máximo entre a rota do VANT e a rota planejada	1,6 m	1,3 m	1,4 m	1,7 m
Sit. Crítica	Número de experimentos com bateria baixa	6	0	13	0
	Número de experimentos com tempo ruim	3	0	0	3
	Número de experimentos com falha no AP	0	13	0	2
	Número de experimentos com falha no MOSA	0	0	0	2
Geral	Número de experimentos sem falha crítica	3	5	5	3
	Altitude de cruzeiro da aeronave (M_{alt})	5 m	15 m	20 m	12 m
	Média de *waypoints* da estratégia FixedRoute4m	11	-	-	-
	Média de *waypoints* do método HGA4m	0	85	78	0
	Média de *waypoints* do método CCQSP4m	0	31	31	0
	Média de *waypoints* da estratégia MOSA adaptativo	-	-	-	79
	Altitude do RTL (h_{rtl})	10,0 m	-	-	20,0 m
	Média da distância crítica percorrida no replanejamento (d_c)	10,3 m	-	9,2 m	8,5 m
	Número de vezes que pousou em uma região segura	5 de 6	-	12 de 13	1 de 2
	Replanejador de rotas utilizado	DE4s	-	MPGA4s	GA4s

Fonte: Elaborada pelo autor.

executados durante um tempo variável com base na configuração de arquitetura aplicada. O risco de violar NFZs para planejadores é definido como 2,0%. Os métodos de replanejamento de caminho são executados dentro de um limite de tempo de 1 segundo e o risco de violar NFZs é definido como 1,0%.

A realização das missões descritas foi possível utilizando a arquitetura de baixo custo proposta para todos os cenários. A mudança de missão utilizando nosso sistema é facilmente executada apenas trocando os arquivos de mapas e pontos-alvo da missão (*waypoints*).

9.6 Resultados Finais da Tese

A presente seção utiliza como base, os experimentos conduzidos e apresentados na Seção 9.5, de forma a detalhar mais esses experimentos.

Tabela 30 – Resumo do tempo de processamento dos métodos.

	$\mathcal{P}_P$	H_{cc}	$\mathcal{L}_P$	$\mathcal{T}_{SC_P}$	Média do Tempo de Processamento	Δ_P
Planejador	HGA4m	GCS	*offboard*	14,0 s	15,5 s	2,0%
	HGA4m	Intel Edison	*onboard*	90,0 s	99,0 s	2,0%
	CCQSP4m	GCS	*offboard*	4,0 s	5,0 s	2,0%
	CCQSP4m	Intel Edison	*onboard*	30,0 s	31,0 s	2,0%
	$\mathcal{P}_R$	H_{cc}	$\mathcal{L}_R$	$\mathcal{T}_{SC_R}$	Média do Tempo de Processamento	Δ_R
Replanejador	MPGA4s	Raspberry Pi 3	*onboard*	1,0 s	1,6 s	1,0%
	MPGA4s	Intel Edison	*onboard*	1,0 s	2,6 s	1,0%
	GA4s	Raspberry Pi 3	*onboard*	1,0 s	1,2 s	1,0%
	GA4s	BeagleBone Black	*onboard*	1,0 s	3,4 s	1,0%
	DE4s	Raspberry Pi 3	*onboard*	1,0 s	2,0 s	1,0%
	DE4s	Intel Edison	*onboard*	1,0 s	2,2 s	1,0%

Fonte: Elaborada pelo autor.

Figura 73 – Fluxo de avaliação da arquitetura nos experimentos com voo real.

Fonte: Elaborada pelo autor.

Uma das características desejadas da plataforma autônoma desenvolvida é ser *plug and play*. Desse modo, diferentes componentes de hardware e software podem ser chaveados/trocados sem grandes alterações no sistema. A Figura 73 detalha as configurações de hardware e software utilizadas nos oito experimentos reais conduzidos. A forma de se trocar os componentes de software é feita apenas alterando o(s) parâmetro(s) associado(s) no arquivo de entrada, descrito em Apêndice A. Para se trocar os componentes de hardware, deve-se também modificar o(s) parâmetro(s) nesse arquivo, além de trocar fisicamente os equipamentos desejados. As linhas coloridas sinalizam os componentes selecionados/utilizados em cada um dos oito experimentos.

A Tabela 31 detalha como foram definidas as 50 simulações HITL. Essa tabela mapeia os componentes de hardware e software utilizados na avaliação da plataforma autônoma, de forma semelhante a Figura 73. Nessa tabela foram sintetizados os cenários avaliados, o CC utilizado, o planejador/replanejador utilizado, o tipo de falha associada e o tipo de ação executada. Vale lembrar que em cada uma das simulações HITL executadas algum parâmetro foi alterado, de forma a avaliar o comportamento do sistema com os diversos módulos suportados.

Tabela 31 – Configurações utilizadas nos experimentos HITL executados.

Cenário	CC	Planejador	Replanejador	Tipo de Falha	Tipo de Ação
Case-I	Raspberry Pi 3	FixedRoute4m	-	Sem Falha	-
			-	Condição de tempo ruim	RTL
			DE4s	Bateria baixa-1	Replanejar rota
			DE4s	Bateria baixa-2	Replanejar rota
			DE4s	Bateria baixa-3	Replanejar rota
	Intel Edison	FixedRoute4m	-	Sem Falha	-
			-	Condição de tempo ruim	RTL
			DE4s	Bateria baixa-1	Replanejar rota
			DE4s	Bateria baixa-2	Replanejar rota
			DE4s	Bateria baixa-3	Replanejar rota
Case-II	Raspberry Pi 3	CCQSP4m	-	Sem Falha	-
			-	Falha no AP-1	Pousar na vertical
			-	Falha no AP-2	Pousar na vertical
			-	Falha no AP-3	Pousar na vertical
		HGA4m	-	Sem Falha	-
			-	Falha no AP-1	Pousar na vertical
			-	Falha no AP-2	Pousar na vertical
			-	Falha no AP-3	Pousar na vertical
	BeagleBone Black	CCQSP4m	-	Sem Falha	-
			-	Falha no AP-1	Pousar na vertical
			-	Falha no AP-2	Pousar na vertical
			-	Falha no AP-3	Pousar na vertical
		HGA4m	-	Sem Falha	-
			-	Falha no AP-1	Pousar na vertical
			-	Falha no AP-2	Pousar na vertical
			-	Falha no AP-3	Pousar na vertical
Case-III	Raspberry Pi 3	CCQSP4m	-	Sem Falha	-
			MPGA4s	Bateria baixa-1	Replanejar rota
			MPGA4s	Bateria baixa-2	Replanejar rota
			MPGA4s	Bateria baixa-3	Replanejar rota
		HGA4m	-	Sem Falha	-
			MPGA4s	Bateria baixa-1	Replanejar rota
			MPGA4s	Bateria baixa-2	Replanejar rota
			MPGA4s	Bateria baixa-3	Replanejar rota
	Intel Edison	CCQSP4m	-	Sem Falha	-
			MPGA4s	Bateria baixa-1	Replanejar rota
			MPGA4s	Bateria baixa-2	Replanejar rota
			MPGA4s	Bateria baixa-3	Replanejar rota
		HGA4m	-	Sem Falha	-
			MPGA4s	Bateria baixa-1	Replanejar rota
			MPGA4s	Bateria baixa-2	Replanejar rota
			MPGA4s	Bateria baixa-3	Replanejar rota
Case-IV	Raspberry Pi 3	M-Adaptive4m	-	Sem Falha	-
			GA4s	Falha no MOSA	Replanejar rota
			-	Falha no AP	Pousar na vertical
			-	Condição de tempo ruim	RTL
	BeagleBone Black	M-Adaptive4m	-	Sem Falha	-
			GA4s	Falha no MOSA	Replanejar rota
			-	Falha no AP	Pousar na vertical
			-	Condição de tempo ruim	RTL

Fonte: Elaborada pelo autor.

Uma análise de desempenho foi feita entre os três computadores de bordo utilizados: Raspberry Pi (RPi), Intel Edison (Edison) e BeagleBone Black (BBB). A Figura 74a mostra o tempo de processamento, em milissegundos, sobre diferentes critérios para os três CCs. Os critérios utilizados foram: `set-parameter`, `get-parameters`, `get-all-sensors` e `set-waypoint`. Esses critérios representam requisições feitas pelo IFA (executando no CC) para o AP. A função `set-parameter`, em geral, é chamada apenas algumas vezes na inicialização do sistema, apesar disso não ser restritivo. O comando `get-parameters` é chamado uma única

vez na inicialização. O critério `get-all-sensors` retorna todas as informações de sensores da aeronave, ele é chamado a uma frequência de 2 hz durante toda a execução dos sistemas MOSA e IFA e gasta apenas 9,28 ms (na RPi) para processar. Por fim, o critério `set-waypoint` é chamado toda vez que a missão for alterada para pouso na vertical ou para fazer um RTL.

Figura 74 – Comparação de desempenho entre os *compunions computers*.

(a) Comparação baseada no tempo de processamento. (b) Comparação baseada no número de vezes.

Fonte: Elaborada pelo autor.

Ao analisar esses critérios, percebemos que a placa com melhor desempenho é a RPi, seguida pela Edison e pela BBB. A Figura 74b apresenta o número de vezes que uma placa é mais rápida que outra de acordo com esses diferentes critérios. A RPi chega a ser até 11 vezes mais rápida que a BBB dependendo do critério analisado. O critério `get-all-sensors`, talvez seja, o mais confiável, pois foi gerado a partir milhares de amostras de dados. Dessa forma, a RPi é cerca de 7 vezes mais rápida que a BBB, já a RPi é aproximadamente de 2 vezes mais rápida que a Edison.

Tabela 32 – Valores de ângulos de *roll* e *pitch* e número de satélites capturados.

	AP	GPS	*Roll* (ϕ)		*Pitch* (θ)		Número de Satélites Visíveis		
Descrição	-	-	Máximo	Mínimo	Máximo	Mínimo	Média	Máximo	Mínimo
Case-I	APM	NEO-6M	20,14º	-15,02º	12,16º	-17,19º	10,0	10	10
Case-I	APM	NEO-6M	20,31º	-21,64º	28,05º	-22,09º	10,0	10	10
Case-II	APM	NEO-6M	16,53º	-12,63º	15,56º	-16,63º	8,9	9	8
Case-II	Pixhawk	NEO-M8N	13,32º	-16,98º	17,97º	-21,93º	18,6	19	17
Case-III	Pixhawk	NEO-M8N	13,32º	-8,96º	17,97º	-15,98º	17,5	18	15
Case-III	Pixhawk	NEO-M8N	11,23º	-11,51º	12,63º	-8,58º	13,0	13	12
Case-IV	Pixhawk	NEO-M8N	9,14º	-11,45º	12,56º	-9,91º	12,9	16	9
Case-IV	Pixhawk	NEO-M8N	12,39º	-8,37º	12,02º	-12,69º	17,9	18	17
Média	-	-	14,55º	-13,32º	16,12º	-15,63º	13,62	14,13	12,25
Máximo	-	-	**20,31º**	-8,37º	**28,05º**	-8,58º	18,63	**19**	17
Mínimo	-	-	9,14º	**-21,64º**	12,02º	**-22,09º**	8,96	9	**8**

Fonte: Elaborada pelo autor.

A Tabela 32 mostra alguns dados obtidos pelos sensores durante os experimentos realizados. Ao observar essa tabela vemos que a menor quantidade de satélites capturados foi 8 e a maior foi 19. De forma geral, os experimentos com o GPS NEO-M8N foram o que capturaram a maior quantidade de satélites. Outras informações importantes são os ângulos de *roll* (ϕ) e

pitch (θ). Os valores de *roll* ficaram no intervalo de -21,64º $\leq \phi \leq$ 20,31º, já os valores de *pitch* ficaram no intervalo de -22,09º $\leq \theta \leq$ 28,05º.

$$tomada\ de\ decisão\ = \begin{cases} pouso\ na\ vertical, & se\ ((|\phi| \geq 35)\ ou\ (|\theta| \geq 35)) \\ abrir\ paraquedas, & se\ ((|\phi| \geq 45)\ ou\ (|\theta| \geq 45)) \end{cases} \tag{9.5}$$

De acordo com a experiência obtida, verificou-se que um dispositivo de segurança adicional poderia ser implementado no sistema IFA, complementando o esquema de tomada de decisão mostrado na Figura 36. Esse dispositivo analisaria os valores de *roll* e *pitch* e caso ultrapasse um determinado limiar, uma tomada de decisão poderia ser feita, como um pouso na vertical, ou ainda, um disparo de paraquedas. A Equação 9.5 descreve essas possíveis tomadas de decisão caso o ângulo esteja fora dos intervalos sugeridos.

Tabela 33 – Conjunto de componentes de software utilizado nos experimentos.

				Tipo de Experimento		
Sistema	Módulo	Algoritmo/Estratégia	Local	SITL	HITL	REAL_FLIGHT
UAV-IFA	Replanejador	MPGA4s	onboard	Sim	Sim	Sim
		MPGA4s	offboard	Sim	Sim	Não
		GA4s	onboard	Sim	Sim	Não
		GA4s	offboard	Sim	Sim	Não
		GH4s	onboard	Sim	Sim	Não
		GH4s	offboard	Sim	Sim	Não
		DE4s	onboard	Sim	Sim	Não
		DE4s	offboard	Sim	Sim	Não
		MS4s	onboard	Sim	Sim	Não
		MS4s	offboard	Sim	Sim	Não
		GA-GA-4s	onboard	Sim	Sim	Não
		GA-GH-4s	onboard	Sim	Sim	Não
		Pre-Planned4s	onboard	Sim	Não	Não
UAV-MOSA	Planejador	FixedRoute4m	N/A	Sim	Sim	Sim
		HGA4m	onboard	Sim	Sim	Sim
		HGA4m	offboard	Sim	Sim	Sim
		CCQSP4m	onboard	Sim	Sim	Sim
		CCQSP4m	offboard	Sim	Sim	Sim
		M-Adaptive4m	onboard	Sim	Não	Não
		M-Adaptive4m	offboard	Sim	Sim	Sim

Fonte: Elaborada pelo autor.

A Tabela 33 mostra alguns dos algoritmos planejadores e replanejadores utilizados/implementados. Essa tabela destaca, principalmente, em qual tipo de experimento SITL, HITL ou REAL_FLIGHT eles foram avaliados.

A Tabela 34 mostra alguns dos componentes de hardware utilizados nos experimentos semelhantemente a tabela anterior mostrada. Nessa tabela é destacado, também, em qual tipo de experimento SITL, HITL ou REAL_FLIGHT esses equipamentos foram avaliados. Tanto a Tabela 33 quanto a Tabela 34 trazem um conjunto de características suportadas pela plataforma autônoma, em que são destacadas quais características e em quais tipos de experimentos foram avaliados.

Tabela 34 – Conjunto de componentes de hardware utilizado nos experimentos.

Componente		Tipo de Experimento		
Categoria	Nome do Componente	SITL	HITL	REAL_FLIGHT
Piloto Automático	APM	N/A	N/A	Sim
	Pixhawk	N/A	N/A	Sim
Companion Computer	Intel Edison	N/A	Sim	Sim
	Raspberry Pi 2	N/A	Sim	Sim
	Raspberry Pi 3	N/A	Sim	Sim
	BeagleBone Black	N/A	Sim	Não
Sensor	*Power Module*	Sim (Simulado)	Sim	Sim
	Câmera	Sim (Simulado)	Sim	Sim
	Temperatura	Sim (Simulado)	Sim	Não
	Sonar	Sim (Simulado)	Sim	Não
Atuador	*Buzzer*	Sim (Simulado)	Sim	Sim
	LED	N/A	Sim	Não
	Parachute	N/A	Não	Não
	Pulverizador	N/A	Não	Não

Fonte: Elaborada pelo autor.

Uma comparação dos planejadores e replanejadores de rotas utilizados na plataforma autônoma é apresentada na Tabela 35. Ao analisarmos essa tabela, percebemos que a estratégia clássica (FixedRoute4m), possui a vantagem do baixo tempo de processamento, diante das estratégias HGA4m e CCQSP4m. No entanto, ela não faz o desvio de obstáculos, que as aplicações atuais demandam. Dente as estratégias HGA4m e CCQSP4m, a mais interessante de se utilizar é a CCQSP4m devido ao seu menor tempo de processamento. Com relação aos métodos para pouso emergencial, os métodos baseados em computação evolutiva (GA4s, DE4s, MPGA4s) e suas versões combinadas (GA-GA-4s e GA-GH-4s) são os que possuem melhor qualidade de solução, além de possuir um baixo tempo computacional.

Tabela 35 – Comparação entre os planejadores e replanejadores utilizados.

Planejador	Estratégia	Local de Execução	Desvio de Obstáculo	Alocação de Risco	Nº *Cores* da CPU	Utiliza CPLEX	Tempo Processamento	Qualidade da Solução
HGA4m	Combinada[5]	*on/offboard*	Sim	Sim	1	Sim	Lento (≈10 a 100 s)	⋆⋆⋆
CCQSP4m	PLIM	*on/offboard*	Sim	Sim	1	Sim	Moderado (≈4 a 40 s)	⋆⋆⋆
FixedRoute4m	Clássica	*onboard*	Não	Não	1	Não	Super Ráp. (≈<0,1 s)	⋆
Replanejador	Estratégia	Local de Execução	Desvio de Obstáculo	Alocação de Risco	Nº *Cores* da CPU	Utiliza CPLEX	Tempo de Processamento	Qualidade da Solução
GA4s	Metaheurística	*on/offboard*	Sim	Sim	1	Não	Rápido (≈0,5 a 3 s)	⋆⋆⋆
MPGA4s	Metaheurística	*on/offboard*	Sim	Sim	1	Não	Rápido (≈0,5 a 3 s)	⋆⋆⋆
DE4s	Metaheurística	*on/offboard*	Sim	Sim	1	Não	Rápido (≈0,5 a 3 s)	⋆⋆⋆
GH4s	Heurística	*on/offboard*	Não	Não	1	Não	Muito Ráp. (≈<0,5 s)	⋆⋆
MS4s	Heurística	*on/offboard*	Sim	Sim	1	Não	Rápido (≈0,5 a 3 s)	⋆⋆
GA-GA-4s	Combinada[6]	*onboard*	Sim	Sim	2	Não	Rápido (≈0,5 a 3 s)	⋆⋆⋆
GA-GH-4s	Combinada[7]	*onboard*	Sim	Sim	2	Não	Rápido (≈0,5 a 3 s)	⋆⋆⋆
Pre-Planned4s	Determinística	*onboard*	Depende	Depende	1	Não	Muito Ráp. (≈<0,5 s)	⋆

Fonte: Elaborada pelo autor.

[5] Combinação de uma Metaheurística (GA) com Grafo de Visibilidade e solução de modelo de PLIM
[6] Redundância utilizando duas Metaheurísticas (GA)
[7] Redundância utilizando uma Metaheurística (GA) e uma Heurística (GH)

9.7 Considerações Finais

Este capítulo apresentou os resultados computacionais obtidos durante o doutorado, em que três artigos completos foram publicados em conferências internacionais e um artigo foi submetido em periódico internacional. Os resultados reportados evidenciam que a arquitetura autônoma de baixo custo proposta consegue executar os sistemas de missão e de segurança de forma satisfatória.

O próximo capítulo apresentará as considerações finais deste trabalho.

CAPÍTULO

10

CONSIDERAÇÕES FINAIS

“ O futuro dependerá daquilo que fazemos no presente. ”

Mohandas Karamchand Gandhi

10.1 Avaliação Geral

O presente trabalho de doutorado desenvolveu um sistema embarcado autônomo de baixo custo, voltado à execução de missão e à segurança de voo para Veículos Aéreos Não Tripulados (VANTs). A aeronave utilizada possui um piloto automático (AP) integrado com um *Companion Computer* (CC), o que possibilita que tomadas de decisão sejam executadas em tempo real durante o voo. A autonomia da aeronave foi alcançada ao embarcar os sistemas *Mission Oriented Sensor Array* (MOSA), *In-Flight Awareness* (IFA) e *Services to DroneKit* (S2DK) no computador de bordo.

Os VANTs, iDroneAlpha e iDroneBeta, foram construídos e possuem um AP que se comunica com o CC e seus periféricos. Durante esta pesquisa, diversos experimentos envolvendo simulações *Software-In-The-Loop* (SITL), *Hardware-In-The-Loop* (HITL) e voos reais foram executados para validar a arquitetura proposta. Um dos requisitos estabelecidos foi que a arquitetura fosse *plug and play*, suportando diversos módulos de hardware e software. Ao todo, dois APs (APM e Pixhawk) e três CCs (Raspberry Pi 3, Intel Edison e BeagleBone Black) foram avaliados.

Nos experimentos realizados, o sistema autônomo proposto foi capaz de supervisionar a execução da missão e a segurança de voo. Em cenários sem situação crítica ao voo, o MOSA foi capaz de garantir a realização da missão sem grandes desvios da rota planejada. Já em cenários em que ocorreram uma situação crítica, o sistema IFA também pode assumir o controle da aeronave e executar um comportamento de emergência, como pouso, RTL ou pouso vertical. Em todos os cenários, o planejamento de rota e o replanejamento de rota levaram em consideração a alocação de risco para evitar os obstáculos. O sistema IFA foi capaz de reagir às falhas durante o voo, sem incorrer em risco de colisão com tais obstáculos. As trajetórias obtidas nas simulações

de HITL, bem como as de voos reais foram similares. A arquitetura de hardware e software empregada funcionou satisfatoriamente durante todos os experimentos.

Este trabalho objetivou desenvolver uma arquitetura, com as seguintes características: propósito geral, resiliência e autonomia. A característica de propósito geral foi alcançada, uma vez que diferentes missões podem ser incorporadas ao VANT sem grandes alterações no sistema. A característica resiliência foi alcançada, tendo em vista que o sistema é capaz de prevenir a propagação de erros ou alterar seu comportamento visando o correto funcionamento da aeronave. Por fim, a característica autonomia foi alcançada, visto que o VANT é capaz de executar missões sem intervenção humana mesmo diante das situações críticas modeladas.

10.2 Contribuições Alcançadas

Durante a elaboração deste trabalho, algumas contribuições foram alcançadas.

- Adaptar e unificar os sistemas MOSA, IFA e planejadores de rotas em um única arquitetura embarcada na aeronave (ARANTES *et al.*, 2017a);
- Planejar e executar a missão de forma embarcada em tempo hábil, utilizando computadores de bordo de baixo custo (ARANTES *et al.*, 2017a);
- Reagir de forma inteligente a algumas falhas críticas e tomadas de decisão autônoma para minimizar danos/acidentes (ARANTES *et al.*, 2017; VANNINI *et al.*, 2018).
- Implementar todo o sistema em código-fonte aberto e modular, disponível no Github [1], em que os principais sistemas são:
  - **Sistema UAV-MOSA**: implementação do modelo de referência MOSA que provê a supervisão da missão;
  - **Sistema UAV-IFA**: implementação do modelo de referência IFA que provê mecanismos de segurança adicional;
  - **Sistema UAV-S2DK**: implementação que permite aos sistemas MOSA e IFA conversarem com o AP;
  - **Sistema UAV-GCS**: implementação que possibilita ao usuário interagir com os sistemas MOSA e IFA;
  - **Sistema UAV-Mission-Creator**: implementação que simplifica ao usuário a criação de missões.

[1] <https://github.com/jesimar/UAV-Toolkit/>

10.3 Produção Científica

Os seguintes artigos foram publicados ou submetidos durante a elaboração desta tese:

An Embedded System Architecture based on Genetic Algorithms for Mission and Safety Planning with UAV: Publicado na conferência *Genetic and Evolutionary Computation Conference* (GECCO), em 2017. *Qualis* A1. Resultado direto da presente tese de doutorado. Encontra-se no Apêndice B.

Evaluating Hardware Platforms and PathRe-Planning Strategies for the UAV Emergency Landing Problem: Publicado na conferência *IEEE International Conference on Tools with Artificial Intelligence* (ICTAI), em 2017. *Qualis* B1. Resultado direto da presente tese de doutorado. Encontra-se no Apêndice C.

Service-Oriented Architecture to Integrate Flight Safety and Mission Management Subsystems Into UAVs: Publicado no congresso *International Council of the Aeronautical Sciences* (ICAS), em 2018. Resultado direto da presente tese de doutorado. Encontra-se no Apêndice D.

Low-Cost Architecture for Autonomous Flight of UAVs Based on Mission Execution and Safety: Submetido no periódico *Expert Systems With Applications* (ESWA), em 2019. *Qualis* A1. Resultado direto da presente tese de doutorado. Encontra-se no Apêndice E.

10.4 Dificuldades Enfrentadas

Este trabalho abrangeu conhecimentos multidisciplinares, o que permitiu produzir resultados novos ao custo de lidar com diversas dificuldades, não só tecnológicas como também científicas, abaixo listadas:

- **Falta de Padrões**: existem diversos algoritmos de planejamento/replanejamento de rotas, que possuem arquivos de entrada com mapas e com configurações bem diferentes entre si. Os arquivos de saída com as rotas dos planejadores/replanejadores também não apresentam padrões bem definidos.

- **Regulamentação dos VANTs**: o voo de VANTs em regiões povoadas ou próximo à pessoas não é autorizado. Não existe uma regulamentação que autorize a realização de voos totalmente autônomos e que facilite o uso de VANTs para pesquisas acadêmicas. Dessa forma, a realização do voo demandou diversos esforços.

- **Integração de Hardware**: a principal dificuldade na integração de hardware foi estabelecer a comunicação do computador de bordo com o piloto automático. Outra dificuldade foi a montagem dos VANTs e configuração dos parâmetros do AP. Fazer a montagem e a

configuração do VANT para o voo real gerou alguns contratempos, como: problemas com a comunicação wifi; e com o sistema de alimentação do VANT.

- **Integração de Software**: as principais dificuldades na integração de software estão relacionadas ao alto grau de paralelismo e à sincronização dos sistemas MOSA e IFA. Outra dificuldade foi fazer a comunicação entre esses dois sistemas. A plataforma autônoma possui diversos módulos de software que se comunicam entre si, o que pode dificultar a compreensão do sistema e, consequentemente, a sua implementação.

10.5 Trabalhos Futuros

Lidar com o desenvolvimento de sistemas embarcados autônomos para VANTs é algo bastante amplo e contínuo. Existem inúmeras possibilidades de extensão da plataforma autônoma aqui proposta. Dessa maneira, os seguintes trabalhos poderão ser realizados a fim de dar continuidade ao que foi proposto:

- Fazer a separação de interesses entre os sistemas MOSA e IFA em nível físico;
- Validar a arquitetura utilizando outros CC, como o Odroid XU4 e a Raspberry Pi Zero;
- Validar a arquitetura utilizando outros AP, como o Pixhawk v2, Erle-Brain v3 e Naza M;
- Desenvolver e validar a plataforma autônoma em VANTs de asa fixa, como o Ararinha;
- Incorporar o desvio de obstáculos dinâmicos, baseado em visão computacional;
- Atualizar automaticamente a velocidade de navegação, subida e descida do VANT, caso o nível da bateria esteja abaixo de um determinado limiar;
- Executar o IFA, sistema crítico em segurança, em um CC com Sistema Operacional de Tempo Real (RTOS);
- Adaptar a plataforma autônoma ao contexto de veículos terrestres não tripulados;
- Prover melhorias no protocolo de especificação do mapa e plano da missão ampliando as possíveis ações a serem executadas pela aeronave, principalmente, em missões adaptativas;
- Incorporar novos algoritmos, como o *Hybrid Chance-Constrained Qualitative State Planning* (HCCQSP), na plataforma autônoma, de modo a resolver novos problemas, por exemplo combate a incêndios.
- Desenvolver novos algoritmos planejadores de missão que não utilizem modelos de PLIM e que tenham um baixo custo computacional.

REFERÊNCIAS

3DR. *3DR Site*. https://3dr.com/: [s.n.], 2017. Citado na página 61.

AEROVIRONMENT, A. **AV AeroVironment**. https://www.avinc.com/uas/view/puma: [s.n.], 2019. Citado na página 48.

AIRFORCETECHNOLOGY. **Airforce Technology**. https://www.airforce-technology.com/projects/predator-uav/: [s.n.], 2019. Citado na página 48.

ALSALAM, B. H. Y.; MORTON, K.; CAMPBELL, D.; GONZALEZ, F. Autonomous uav with vision based on-board decision making for remote sensing and precision agriculture. In: **2017 IEEE Aerospace Conference**. [S.l.: s.n.], 2017. p. 1–12. Citado nas páginas 40, 48, 79, 80, 84, 85 e 87.

AMAZON. **Amazon**. https://www.amazon.com/Battery-Voltage-Checker-Indicator-Helicopter/dp/B01DBZK7Y4/ref=pd_lpo_vtph_21_lp_img_2/134-1765715-0792759?_-encoding=UTF8&psc=1&refRID=CRAQMXTDRG2A3PY22VTR: [s.n.], 2018. Citado na página 61.

____. **Amazon Prime Air**. https://www.amazon.com/b?node=8037720011: [s.n.], 2018. Citado nas páginas 42 e 48.

AMENYO, J.-T.; PHELPS, D.; OLADIPO, O.; SEWOVOE-EKUOE, F.; JADOONANAN, S.; JADOONANAN, S.; TABASSUM, T.; GNABODE, S.; SHERPA, T. D.; FALZONE, M. *et al.* Medizdroids project: Ultra-low cost, low-altitude, affordable and sustainable uav multicopter drones for mosquito vector control in malaria disease management. In: IEEE. **IEEE Global Humanitarian Technology Conference (GHTC 2014)**. [S.l.], 2014. p. 590–596. Citado na página 79.

ANAC. **Instrução Suplementar - IS: IS n 21-002 Revisão A**. [S.l.], 2012. 21 p. Citado na página 45.

____. **Proposta de Instrução Suplementar, Intitulada Emissão de Certificado de Autorização de Voo Experimental para Sistemas de Veículo Aéreo Não Tripulado**. [S.l.], 2012. 5 p. Citado na página 45.

ANDERSSON, M. **Fault Diagnosis of a Fixed Wing UAV Using Hardware and Analytical Redundancy**. Tese (Master Thesis) — Linkopings Universitet, 2013. Citado na página 69.

ARANTES, J. d. S.; ARANTES, M. d. S.; TOLEDO, C. F. M.; JÚNIOR, O. T.; WILLIAMS, B. C. An embedded system architecture based on genetic algorithms for mission and safety planning with uav. In: ACM. **Proceedings of the Genetic and Evolutionary Computation Conference**. [S.l.], 2017. p. 1049–1056. Citado nas páginas 106, 118, 169 e 194.

____. Heuristic and genetic algorithm approaches for uav path planning under critical situation. **International Journal on Artificial Intelligence Tools**, World Scientific, v. 26, n. 01, p. 1760008, 2017. Citado na página 132.

ARANTES, J. S. **Planejamento de rota para VANTs em caso de situação crítica: Uma abordagem baseada em segurança**. Tese (Dissertação de Mestrado) — Universidade de São Paulo (USP), ago 2016. Sao Carlos, SP. Citado nas páginas 40, 77, 80, 104, 126 e 154.

ARANTES, J. S.; ARANTES, M. S.; MISSAGLIA, A. B.; SIMõES, E. V.; TOLEDO, C. F. M. Evaluating hardware platforms and path re-planning strategies for the uav emergency landing problem. In: **ICTAI**. [S.l.]: IEEE International Conference on Tools with Artificial Intelligence, 2017. p. 1–8. Citado nas páginas 78, 133 e 194.

ARANTES, J. S.; ARANTES, M. S.; TOLEDO, C. F. M.; WILLIAMS, B. C. A multi-population genetic algorithm for uav path re-planning under critical situation. In: **ICTAI**. [S.l.]: IEEE International Conference on Tools with Artificial Intelligence, 2015. p. 1–8. Citado nas páginas 77, 80, 107, 112, 126, 164, 166 e 174.

ARANTES, M. d. S.; ARANTES, J. d. S.; TOLEDO, C. F. M.; WILLIAMS, B. C. A hybrid multi-population genetic algorithm for uav path planning. In: **Proceedings of the Genetic and Evolutionary Computation Conference 2016**. New York, NY, USA: ACM, 2016. (GECCO '16), p. 853–860. ISBN 978-1-4503-4206-3. Disponível em: <http://doi.acm.org/10.1145/2908812.2908919>. Citado nas páginas 78, 80, 106, 107, 110, 131, 138, 164 e 174.

ARANTES, M. S. **Ambiente para desenvolvimento de métodos aplicados a problemas de otimização**. Tese (Dissertação de Mestrado) — Universidade de São Paulo (USP), abr 2014. São Carlos, SP. Citado na página 144.

ARANTES, M. S. **Hybrid Qualitative State Plan Problem e o Planejamento de Missão com VANTs**. Tese (Tese de Doutorado) — Universidade de Sao Paulo (USP), ago 2017. Sao Carlos, SP. Citado nas páginas 40, 78, 80, 107, 132, 154 e 160.

ARDUPILOT. **ArduPilot Open Source Autopilot**. http://ardupilot.org/: [s.n.], 2017. Citado na página 56.

____. **Choosing a Ground Station**. http://ardupilot.org/plane/docs/common-choosing-a-ground-station.html: [s.n.], 2017. Citado na página 51.

____. **Glossary**. http://ardupilot.org/planner/docs/common-glossary.html: [s.n.], 2017. Citado na página 55.

____. **HITL Simulators**. http://ardupilot.org/dev/docs/hitl-simulators.html: [s.n.], 2017. Citado na página 64.

____. **SITL Simulator**. http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html: [s.n.], 2017. Citado na página 63.

____. **Common Power Module**. http://ardupilot.org/copter/docs/common-3dr-power-module.html: [s.n.], 2018. Citado na página 61.

____. **ADS-B Receiver**. http://ardupilot.org/copter/docs/common-ads-b-receiver.html#common-ads-b-receiver: [s.n.], 2019. Citado na página 70.

____. **Flight Modes - Copter**. http://ardupilot.org/copter/docs/flight-modes.html: [s.n.], 2019. Citado na página 58.

____. **Flight Modes - Plane**. http://ardupilot.org/plane/docs/flight-modes.html: [s.n.], 2019. Citado na página 58.

ARIU, K.; FANG, C.; ARANTES, M.; TOLEDO, C.; WILLIAMS, B. Chance-constrained path planning with continuous time safety guarantees. In: **Workshops at the Thirty-First AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2017. Citado na página 154.

ARMAMENTODEFESA. **Armamento e Defesa**. http://armamentoedefesa.blogspot.com/2013/12/arara-ii-e-o-primeiro-drone-brasileiro.html: [s.n.], 2019. Citado na página 48.

AUSTIN, R. **Unmanned Aircraft Systems: UAV Design, Development and Deployment**. [S.l.]: John Wiley & Sons, 2010. v. 54. Citado na página 48.

BANKS, J. Discrete event simulation. In: **Winter Simulation Conference**. [S.l.: s.n.], 2000. Citado na página 53.

BEAGLEBOARD. **beagleboard.org**. https://beagleboard.org/black: [s.n.], 2017. Citado na página 59.

BLACKMORE, L.; ONO, M.; WILLIAMS, B. C. **Chance-Constrained Optimal Path Planning with Obstacles**. 2011. IEEE Press. Citado nas páginas 106, 107, 110, 118, 138, 153, 154 e 164.

BOEING. **Boeing**. https://www.boeing.com/defense/autonomous-systems/scaneagle/index.page: [s.n.], 2019. Citado na página 48.

BOUBETA-PUIG, J.; MOGUEL, E.; SÁNCHEZ-FIGUEROA, F.; HERNÁNDEZ, J.; PRECIADO, J. C. An autonomous uav architecture for remote sensing and intelligent decision-making. **IEEE Internet Computing**, IEEE, v. 22, n. 3, p. 6–15, 2018. Citado nas páginas 40, 63, 79, 84, 85, 86 e 87.

BRIGGS, F. Uav software architecture. **Infotech@Aerospace 2012**, 2012. Citado nas páginas 80 e 87.

BROWN, A.; ESTABROOK, J.; FRANKLIN, B. Sensor processing and path planning framework for search and rescue uav network. **Worcester Polytech Institute, MA**, 2011. Citado nas páginas 40, 48, 79, 80, 81, 82 e 126.

CAI, G.; CHEN, B. M.; LEE, T. H.; DONG, M. Design and implementation of a hardware-in-the-loop simulation system for small-scale uav helicopters. **Mechatronics**, Elsevier, v. 19, n. 7, p. 1057–1066, 2009. Citado nas páginas 80 e 81.

CARVALHO, J. R. H.; BUENO, S. S.; MODESTO, J. F. Sistemas aéreos não tripulados para o monitoramento e gestão de risco do bioma amazônico. **Computação Brasil - Veículos Autônomos Não Tripulados**, v. 24, p. 1 – 155, 2014. Citado na página 46.

Cavalcante, T. R. F.; Bessa, I. V. d.; Cordeiro, L. C. Planning and evaluation of uav mission planner for intralogistics problems. In: **2017 VII Brazilian Symposium on Computing Systems Engineering (SBESC)**. [S.l.: s.n.], 2017. p. 9–16. ISSN 2324-7894. Citado na página 63.

CENTMESH. **Software in the loop (SITL)**. https://www.ece.ncsu.edu/wireless/MadeInWALAN/CentMeshWikiBackup/tracwiki/wiki/HardWare/Drones/Autopilot/sitl.html: [s.n.], 2017. Citado na página 64.

CHIARAMONTE, R. B. **Detecção e desvio de obstáculos para veículos aéreos não tripulados usando visão monocular**. Tese (Tese de Doutorado) — Universidade de São Paulo (USP), nov 2018. São Carlos, SP. Citado nas páginas 39, 40, 63, 79, 86, 87 e 126.

Choi, H.; Geeves, M.; Alsalam, B.; Gonzalez, F. Open source computer-vision based guidance system for uavs on-board decision making. In: **2016 IEEE Aerospace Conference**. [S.l.: s.n.], 2016. p. 1–5. Citado na página 63.

DAS, S.; SUGANTHAN, P. N. Differential evolution: A survey of the state-of-the-art. **IEEE transactions on evolutionary computation**, IEEE, v. 15, n. 1, p. 4–31, 2011. Citado na página 132.

DJI. **DJI - NAZA-M V2**. http://www.dji.com/naza-m-v2: [s.n.], 2017. Citado na página 56.

DOSHI, A. A.; POSTULA, A. J.; FLETCHER, A.; SINGH, S. P. Development of micro-uav with integrated motion planning for open-cut mining surveillance. **Microprocessors and Microsystems**, Elsevier, v. 39, n. 8, p. 829–835, 2015. Citado na página 48.

DRONEKIT. **Dronekit by 3DR Robotics**. http://dronekit.io/: [s.n.], 2018. Citado nas páginas 62 e 63.

DRONETREST. **Beginners guide to drone autopilots (flight controllers) and how they work**. http://www.dronetrest.com/t/beginners-guide-to-drone-autopilots-flight-controllers-and-how-they-work/1380: [s.n.], 2015. Citado na página 56.

____. **DroneTrest**. http://www.dronetrest.com/t/beginners-guide-to-drone-autopilots-flight-controllers-and-how-they-work/1380: [s.n.], 2018. Citado na página 62.

EASA. **EASA Annual Safety Review 2017**. https://cdn.aviation-safety.net/airlinesafety/industry/reports/EASA-Annual-safety-review-2017.pdf, 2017. Citado na página 41.

EBEID, E.; SKRIVER, M.; JIN, J. A survey on open-source flight control platforms of unmanned aerial vehicle. In: IEEE. **Euromicro Symposium on Digital Systems Design**. [S.l.], 2017. Citado nas páginas 56 e 60.

ERLEROBOTICS. **Erle-Brain 3 - Erle Robotics**. http://erlerobotics.com/blog/erle-brain-3/: [s.n.], 2017. Citado na página 56.

EVERS, L.; BARROS, A. I.; MONSUUR, H.; WAGELMANS, A. Online stochastic uav mission planning with time windows and time-sensitive targets. **European Journal of Operational Research**, v. 238, n. 1, p. 348 – 362, 2014. ISSN 0377-2217. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0377221714002288>. Citado na página 77.

FAA, F. A. A. **FAA Modernization and Reform Act of 2012**. Washington, DC, USA, 2012. 296 p. Citado na página 39.

FIGUEIRA, N.; FREIRE, I.; TRINDADE, O.; OES, E. S. Mission-oriented sensor arrays and uavs - a case study on environmental monitoring. **ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences**, p. 305–312, 2015. Disponível em: <http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-1-W4/305/2015/>. Citado nas páginas 75, 76, 80 e 87.

FIGUEIRA, N. M. **Arranjos de sensores orientados à missão para a geração automática de mapas temáticos em VANTs**. Tese (Tese de Doutorado) — Universidade de Sao Paulo (USP), mar 2016. Sao Carlos, SP. Citado nas páginas 39, 40, 41, 75, 91, 92, 93, 94, 95, 96 e 101.

FLIGHTGEAR. **FlightGear Wiki**. http://wiki.flightgear.org/: [s.n.], 2017. Citado na página 53.

____. **SimGear - FlightGear Wiki**. http://wiki.flightgear.org/SimGear: [s.n.], 2017. Citado na página 53.

____. **Table of models - FlightGear Wiki**. http://wiki.flightgear.org/Table_of_models: [s.n.], 2017. Citado na página 54.

____. **Características - FlightGear - Simulador de Voo**. http://www.br.flightgear.org/sobre-flightgear/caracteristicas/: [s.n.], 2018. Citado na página 53.

FLYSKY. **FlySky Website)**. http://www.flysky-cn.com: [s.n.], 2017. Citado na página 61.

FRITZ, M.; WINTER, S.; FREUND, J.; PFLUEGER, S.; ZEILE, O.; EICKHOFF, J.; ROESER, H.-P. Hardware-in-the-loop environment for verification of a small satellite's on-board software. **Aerospace Science and Technology**, Elsevier, v. 47, p. 388–395, 2015. Citado na página 80.

GAMBOLD, K. A. Unmanned aircraft system access to national airspace. **Background Paper. Unmanned Experts LLC, Washington DC**, 2011. Citado na página 46.

GANS, N.; DIXON, W.; LIND, R.; KURDILA, A. A hardware in the loop simulation platform for vision-based control of unmanned air vehicles. **Mechatronics**, Elsevier, v. 19, n. 7, p. 1043–1056, 2009. Citado na página 81.

GISA. **Grupo de Interesse em SisVANTs e Aplicações**. gisa.icmc.usp.br/site: [s.n.], 2017. Citado na página 49.

____. **GISA**. http://gisa.icmc.usp.br/site/: [s.n.], 2019. Citado na página 48.

GONZALEZ, L. F.; MONTES, G. A.; PUIG, E.; JOHNSON, S.; MENGERSEN, K.; GASTON, K. J. Unmanned aerial vehicles (uavs) and artificial intelligence revolutionizing wildlife monitoring and conservation. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 16, n. 1, p. 97, 2016. Citado na página 76.

GOOGLE. **X - Wing**. https://x.company/projects/wing/: [s.n.], 2019. Citado na página 42.

GOOGLE-EARTH. **Google Earth**. https://www.google.com.br/intl/pt-PT/earth/: [s.n.], 2017. Citado na página 138.

GRAHAM, M. **GPS versus Barometric Altitude: The definitive answer**. http://xcmag.com/news/gps-versus-barometric-altitude-the-definitive-answer/: [s.n.], 2011. Citado na página 62.

GRUMMAN, N. **Northrop Grumman**. http://www.northropgrumman.com/Capabilities/Global Hawk/Pages/default.aspx: [s.n.], 2019. Citado na página 48.

GUIDOTI, F. P. **Proposta de uma Arquitetura Base para Integração de VANT**. Tese (Dissertação de Mestrado) — Universidade de São Paulo (USP), fev 2019. São Carlos, SP. Citado na página 152.

GUNETTI, P.; DODD, T.; THOMPSON, H. A software architecture for autonomous uav mission management and control. **AIAA Infotech@Aerospace 2010**, 2010. Citado nas páginas 80 e 87.

HALLERMANN, N.; MORGENTHAL, G.; RODEHORST, V. Unmanned aerial systems (uas)–case studies of vision based monitoring of ageing structures. In: **Proceedings of the International Symposium Non-Destructive Testing In Civil Engineering, Berlin, Germany**. [S.l.: s.n.], 2015. p. 15–17. Citado na página 48.

HARDKERNEL. **ODROID - Hardkernel**. http://www.hardkernel.com/main/main.php: [s.n.], 2017. Citado na página 59.

HEFFNER, K.; FOUCHER, S. **Machine Vision for Intelligent Drones: An Overview**. https://pt.slideshare.net/kheffner/machine-vision-for-intelligent-drones-an-overview: [s.n.], 2017. Citado nas páginas 47 e 129.

HOSSOMI, M. Y. B. **Framework para Desenvolvimento de Ambientes de Simulação e Controle**. Tese (TCC) — Universidade de São Paulo (USP), nov 2015. São Carlos, SP. Citado na página 137.

INDIAMART. **Unmanned Vehicles**. https://www.indiamart.com/rds/army-defence-quater-guards-stores-supply.html: [s.n.], 2017. Citado na página 46.

INTEL. **Intel Aero Compute Board**. https://software.intel.com/en-us/aero/enclosure-kit: [s.n.], 2017. Citado na página 56.

____. **The Intel Edison Module - IOT**. https://software.intel.com/en-us/iot/hardware/edison: [s.n.], 2017. Citado na página 59.

IPPOLITO, C. A.; PISANICH, G.; AL-ALI, K. Component-based plug-and-play methodologies for rapid embedded technology development. **American Institute of Aeronautics and Astronautics**, 2005. Citado nas páginas 80 e 87.

ITIL, P. A. G. **Separação de Interesses (SdI)**. https://www.pmgacademy.com/pt/glossario-itil/467-separacao-de-interesses-sdi: [s.n.], 2019. Citado na página 143.

JUNG, C. R.; OSÓRIO, F. S.; KELBER, C. R.; HEINEN, F. J. Computação embarcada: Projeto e implementação de veículos autônomos inteligentes. **Anais do CSBC**, v. 5, p. 1358–1406, 2005. Citado na página 66.

KIM, H.; KIM, M.; LIM, H.; PARK, C.; YOON, S.; LEE, D.; CHOI, H.; OH, G.; PARK, J.; KIM, Y. Fully autonomous vision-based net-recovery landing system for a fixed-wing uav. **Mechatronics, IEEE/ASME Transactions on**, v. 18, n. 4, p. 1320–1333, Aug 2013. ISSN 1083-4435. Citado nas páginas 48 e 77.

KIM, J.; SUKKARIEH, S. A baro-altimeter augmented ins/gps navigation system for an uninhabited aerial vehicle. In: **Int. Symp. on Satellite Navigation Technologys**. [S.l.: s.n.], 2003. Citado na página 69.

KOUBAA, A.; QURESHI, B. Dronetrack: Cloud-based real-time object tracking using unmanned aerial vehicles over the internet. **IEEE Access**, IEEE, v. 6, p. 13810–13824, 2018. Citado na página 79.

KURT, H. B.; ALTUĞ, E. Development of an autonomous uav platform for advanced research applications. In: ACM. **Proceedings of the 2017 International Conference on Mechatronics Systems and Control Engineering**. [S.l.], 2017. p. 29–32. Citado na página 78.

KUWATA, Y. **Real-time Trajectory Design for Unmanned Aerial Vehicles using Receding Horizon Control**. Tese (151 p.) — Massachusetts Institute of Technology, 2003. Citado na página 78.

LEITE, O. **Projeto de Lei Nº 5942**. [S.l.], 2013. 11 p. Citado na página 45.

LI, G.-y.; LIU, M.-g. The summary of differential evolution algorithm and its improvements. In: IEEE. **2010 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)**. [S.l.], 2010. v. 3, p. V3–153. Citado na página 132.

LI, H. X. **Kongming: A Generative Planner for Hybrid Systems with Temporally Extended Goals**. Tese (PhD thesis) — Massachusetts Institute of Technology (MIT), 2010. 237 p. Citado na página 154.

LI, P.; CHEN, X.; LI, C. Emergency landing control technology for uav. In: **Guidance, Navigation and Control Conference (CGNCC), 2014 IEEE Chinese**. [S.l.: s.n.], 2014. p. 2359–2362. Citado na página 77.

LI, S.; WAN, Y.; FU, S.; LIU, M.; WU, H. F. Design and implementation of a remote uav-based mobile health monitoring system. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. **SPIE Smart Structures and Materials+ Nondestructive Evaluation and Health Monitoring**. [S.l.], 2017. p. 101690A–101690A. Citado na página 78.

LI, X. A software scheme for uav's safe landing area discovery. **{AASRI} Procedia**, v. 4, p. 230 – 235, 2013. ISSN 2212-6716. 2013 {AASRI} Conference on Intelligent Systems and Control. Disponível em: <http://www.sciencedirect.com/science/article/pii/S221267161300036X>. Citado nas páginas 76 e 77.

LIN, W.; MU, C.; TAKASE, K. Path planning with topological map built with id tag and web camera. In: IEEE. **2006 International Conference on Mechatronics and Automation**. [S.l.], 2006. p. 1739–1744. Citado na página 68.

MAGRABI, S.; GIBBENS, P. Decentralised fault detection and diagnosis in navigation systems for unmanned aerial vehicles. In: IEEE. **Position Location and Navigation Symposium, IEEE 2000**. [S.l.], 2000. p. 363–370. Citado na página 68.

MARTÍ, R.; LOZANO, J. A.; MENDIBURU, A.; HERNANDO, L. Multi-start methods. **Handbook of Heuristics**, Springer, p. 1–21, 2016. Citado na página 133.

MATTEI, A. L. P. **Consciencia situacional em voo de sistemas aereos nao tripulados**. Tese (Tese de Doutorado) — Universidade de Sao Paulo (USP), ago 2015. Sao Carlos, SP. Citado nas páginas 39, 40, 41, 42, 71, 72, 75, 76, 80, 87, 91, 96, 97, 98, 99, 100 e 101.

MATTEI, A. L. P.; CUNHA, A. M.; DIAS, L. A. V.; EUPHRASIO, P. C. S.; TRINDADE, O.; TOLEDO, C. M. Ifa2s – in-flight awareness augmentation systems. In: **Information Technology - New Generations (ITNG), 2015 12th International Conference on**. [S.l.: s.n.], 2015. p. 95–100. Citado na página 76.

MAVLINK. **MAVLink Micro Air Vehicle Communication Protocol**. https://mavlink.io/en/: [s.n.], 2019. Citado nas páginas 51 e 63.

MEULEAU, N.; NEUKOM, C.; PLAUNT, C.; SMITH, D. E.; SMITH, T. The emergency landing planner experiment. In: **21st International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2011. Citado na página 76.

MEULEAU, N.; PLAUNT, C.; SMITH, D. E.; SMITH, T. B. An emergency landing planner for damaged aircraft. In: **IAAI**. [S.l.]: AAAI, 2009. Citado na página 76.

MICROSOFT. **Microsoft Flight**. http://www.microsoft.com/games/flight/: [s.n.], 2017. Citado na página 54.

MOROZOV, A.; JANSCHEK, K. Flight control software failure mitigation: Design optimization for software-implemented fault detectors. **IFAC-PapersOnLine**, v. 49, n. 17, p. 248 – 253, 2016. ISSN 2405-8963. Symposium on Automatic Control in Aerospace. Disponível em: <http://www.sciencedirect.com/science/article/pii/S2405896316315154>. Citado nas páginas 40, 77 e 126.

MOTLAGH, N. H.; BAGAA, M.; TALEB, T. Uav selection for a uav-based integrative iot platform. In: IEEE. **Global Communications Conference (GLOBECOM), 2016 IEEE**. [S.l.], 2016. p. 1–6. Citado na página 78.

MTBSAE. **MTB SAE - Serviços Aéreos Especiais**. www.mtbsae.com.br: [s.n.], 2017. Citado nas páginas 49 e 50.

NASA, A. **Safe2Ditch Technology**. 2017. Disponível em: <https://technology.nasa.gov/t2media/tops/pdf/LAR-TOPS-243.pdf>. Citado nas páginas 76 e 77.

NEX, F.; REMONDINO, F. Uav for 3d mapping applications: a review. **Applied geomatics**, Springer, v. 6, n. 1, p. 1–15, 2014. Citado na página 48.

NOVATEL. **Attitude - Pitch/Roll/Yaw**. https://www.novatel.com/solutions/attitude/: [s.n.], 2019. Citado na página 50.

NTSB, N. T. S. B. **Aviation Statistics**. https://www.ntsb.gov/investigations/data/Pages/aviation_-stats.aspx: [s.n.], 2018. Citado na página 41.

ONO, M.; WILLIAMS, B. C.; BLACKMORE, L. Probabilistic planning for continuous dynamic systems under bounded risk. **J. Artif. Int. Res.**, AI Access Foundation, USA, v. 46, n. 1, p. 511–577, jan 2013. ISSN 1076-9757. Disponível em: <http://dl.acm.org/citation.cfm?id=2512538.2512551>. Citado nas páginas 118, 132, 138 e 154.

OSMAN, A.; WRIGHT, B.; NOURELDIN, A.; EL-SHEIMY, N. Multi-sensor inertial navigation systems employing skewed redundant inertial sensors. In: **ION GNSS 19th International Technical Meeting of the Satellite Division**. [S.l.: s.n.], 2006. Citado na página 69.

OSóRIO, F. S. **Disciplina de Robôs Móveis Autônomos - SSC-5888**. [S.l.], 2014. Citado nas páginas 67 e 71.

PASTOR, E.; LOPEZ, J.; ROYO, P. Uav payload and mission control hardware/software architecture. **IEEE Aerospace and Electronic Systems Magazine**, IEEE, v. 22, n. 6, p. 3–8, 2007. Citado na página 48.

Penserini, L.; Tonucci, E.; Ippoliti, G.; Di Labbio, J. Development framework for drones as smart autonomous systems. In: **2017 8th International Conference on Information, Intelligence, Systems Applications (IISA)**. [S.l.: s.n.], 2017. p. 1–6. Citado na página 63.

PEREZ, D.; MAZA, I.; CABALLERO, F.; SCARLATTI, D.; CASADO, E.; OLLERO, A. A ground control station for a multi-uav surveillance system. **Journal of Intelligent & Robotic Systems**, Springer, v. 69, n. 1-4, p. 119–130, 2013. Citado na página 51.

Perez, E.; Winger, A.; Tran, A.; Garcia-Paredes, C.; Run, N.; Keti, N.; Bhandari, S.; Raheja, A. Autonomous collision avoidance system for a multicopter using stereoscopic vision. In: **2018 International Conference on Unmanned Aircraft Systems (ICUAS)**. [S.l.: s.n.], 2018. p. 579–588. ISSN 2575-7296. Citado na página 63.

PI, R. **Raspberry Pi - Teach, Learn, and Make with Raspberry Pi**. https://www.raspberrypi.org/: [s.n.], 2017. Citado na página 59.

PIRES, R. M. **Desenvolvimento de um mecanismo plug and play para o arranjo inteligente de sensores em sistemas aéreos não tripulados**. Tese (Dissertação de Mestrado) — Universidade de São Paulo (USP), mai 2014. São Carlos, SP. Citado nas páginas 76 e 95.

PIXHAWK. **Hardware in the Loop Simulation Setup**. https://pixhawk.org/users/hil: [s.n.], 2017. Citado na página 55.

____. **Pixhawk Flight Controller Hardware Project**. https://pixhawk.org/: [s.n.], 2017. Citado na página 56.

PRODAN, I.; OLARU, S.; BENCATEL, R.; SOUSA, J. B. de; STOICA, C.; NICULESCU, S.-I. Receding horizon flight control for trajectory tracking of autonomous aerial vehicles. **Control Engineering Practice**, v. 21, n. 10, p. 1334 – 1349, 2013. ISSN 0967-0661. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0967066113001020>. Citado nas páginas 40, 48, 79, 80, 81, 82, 87 e 126.

PROUD, R. W.; HART, J. J.; MROZINSKI, R. B. **Methods for determining the level of autonomy to design into a human spaceflight vehicle: a function specific approach**. [S.l.], 2003. Citado na página 71.

PSIROFONIA, P.; SAMARITAKIS, V.; ELIOPOULOS, P.; POTAMITIS, I. Use of unmanned aerial vehicles for agricultural applications with emphasis on crop protection: Three novel case—studies. **International Journal of Agricultural Science and Technology**, v. 5, p. 30–39, 2017. Citado na página 63.

RAMASAMY, S.; SABATINI, R.; GARDI, A.; LIU, J. {LIDAR} obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid. **Aerospace Science and Technology**, v. 55, p. 344 – 358, 2016. ISSN 1270-9638. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1270963816301900>. Citado nas páginas 40, 48, 79, 80, 83, 84, 87, 125 e 126.

RAMIREZ-ATENCIA, C.; RODRÍGUEZ-FERNÁNDEZ, V.; GONZALEZ-PARDO, A.; CAMACHO, D. New artificial intelligence approaches for future uav ground control stations. In: IEEE. **Evolutionary Computation (CEC), 2017 IEEE Congress on**. [S.l.], 2017. p. 2775–2782. Citado na página 76.

RESÉNDIZ, V. M. A.; RIVAS-ARAIZA, E. A. System identification of a quad-rotor in x configuration from experimental data. **Research in Computing Science**, v. 118, p. 77–86, 2016. Citado na página 50.

ROYO, P.; PASTOR, E.; BARRADO, C.; CUADRADO, R.; BARRAO, F.; GARCIA, A. Hardware design of a small uas helicopter for remote sensing operations. **Drones**, Multidisciplinary Digital Publishing Institute, v. 1, n. 1, p. 3, 2017. Citado na página 48.

SAFETY, A. **Aviation Safety Network**. https://aviation-safety.net/statistics/period/stats.php?cat=A1: [s.n.], 2018. Citado na página 41.

SCHRAGE, D. P. Software-enabledcontrol for intelligent uavs. **IEEE International Symposium on Computer Aided Control System Design**, 1999. Citado na página 79.

SHERIDAN, T. B.; VERPLANK, W. L. **Human and computer control of undersea teleoperators**. [S.l.], 1978. Citado nas páginas 71 e 72.

SIGPLANES. **SIG - Rascal 110**. www.sigplanes.com/SIG-Rascal-110-EG-ARF_p_232.html: [s.n.], 2017. Citado nas páginas 49 e 50.

SOMMERVILLE, I. **Engenharia de Software, 9a edição**. São Paulo, SP, Brasil: [s.n.], 2011. Citado na página 143.

SPINKA, O.; HOLUB, O.; HANZALEK, Z. Low-cost reconfigurable control system for small uavs. **IEEE Transactions on Industrial Electronics**, IEEE, v. 58, n. 3, p. 880–889, 2011. Citado na página 70.

STRUPKA, G.; LEVCHENKOV, A.; GOROBETZ, M. Automated situation analysis as next level of unmanned aerial vehicle artificial intelligence. In: SPRINGER. **International Conference on Applied Human Factors and Ergonomics**. [S.l.], 2017. p. 25–37. Citado na página 76.

SUJIT, P. B.; KINGSTON, D.; BEARD, R. Cooperative forest fire monitoring using multiple uavs. In: **2007 46th IEEE Conference on Decision and Control**. [S.l.: s.n.], 2007. p. 4875–4880. ISSN 0191-2216. Citado na página 48.

Teulière, C.; Marchand, E.; Eck, L. 3-d model-based tracking for uav indoor localization. **IEEE Transactions on Cybernetics**, v. 45, n. 5, p. 869–879, May 2015. ISSN 2168-2267. Citado na página 68.

TOLEDO, C.; FRANCA, P.; KIMMS, A.; MORABITO, R. A multi-population genetic algorithm approach to solve the synchronized and integrated two-level lot sizing and scheduling problem. **International Journal of Production Research**, v. 47, p. 3097–3119, 2009. ISSN 0020-7543. Citado na página 78.

UBLOX, D. **NEO-6 - u-blox 6 GPS Modules - Data Sheet**. [S.l.], 2014. Https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_(GPS.G6-HW-09005).pdf. Citado na página 60.

____. **NEO-M8 - u-blox M8 concurrent GNSS modules - Data Sheet**. [S.l.], 2016. Https://www.u-blox.com/sites/default/files/NEO-M8_DataSheet_%28UBX-13003366%29.pdf. Citado na página 60.

VANNINI, V.; ARANTES, J. d. S.; MATTEI, A. P.; FIGUEIRA, N.; ARANTES, M. d. S.; TOLEDO, C. F. M.; JUNIOR, O. T.; SAQUI-SANNES, P. d. Service-oriented architecture to integrate flight safety and mission management subsystems into uavs. 2018. Citado na página 194.

VELASQUEZ, L. C.; ARGUETA, J.; MAZARIEGOS, K. Implementation of a low cost aerial vehicle for crop analysis in emerging countries. In: IEEE. **Global Humanitarian Technology Conference (GHTC), 2016**. [S.l.], 2016. p. 21–27. Citado na página 78.

WANG, F.; CUI, J.-Q.; CHEN, B.-M.; LEE, T. H. A comprehensive uav indoor navigation system based on vision optical flow and laser fastslam. **Acta Automatica Sinica**, v. 39, n. 11, p. 1889 – 1899, 2013. ISSN 1874-1029. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1874102913600804>. Citado na página 68.

WATTS, A. C.; AMBROSIA, V. G.; HINKLEY, E. A. Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use. **Remote Sensing**, Molecular Diversity Preservation International, v. 4, n. 6, p. 1671–1692, 2012. Citado nas páginas 47 e 129.

WEATHERINGTON, D.; DEPUTY, U. Unmanned aircraft systems roadmap, 2005-2030. **Deputy, UAV Planning Task Force, OUSD (AT&L)**, 2005. Citado na página 46.

WEBER, T. S. Tolerância a falhas: conceitos e exemplos. 2003. Citado nas páginas 68, 69 e 70.

WIKIPéDIA. **Microsoft Flight - Wikipédia, a enciclopédia livre**. http://en.wikipedia.org/wiki/Microsoft_Flight: [s.n.], 2017. Citado na página 54.

WOLF, D. F.; SIMÕES, E.; OSÓRIO, F. S.; JUNIOR, O. T. Robótica móvel inteligente: Da simulação às aplicações no mundo real. In: SN. **Mini-Curso: Jornada de Atualização em Informática (JAI), Congresso da SBC**. [S.l.], 2009. p. 13. Citado na página 67.

X-PLANE. **FAA-Certified X-Plane**. http://www.x-plane.com/pro/certified/: [s.n.], 2017. Citado na página 54.

____. **X-Plane 11 is Here**. http://www.x-plane.com/: [s.n.], 2017. Citado na página 54.

XUE, X.; LAN, Y.; SUN, Z.; CHANG, C.; HOFFMANN, W. C. Develop an unmanned aerial vehicle based automatic aerial spraying system. **Computers and Electronics in Agriculture**, v. 128, p. 58 – 66, 2016. ISSN 0168-1699. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0168169916305282>. Citado nas páginas 40, 48, 79, 80, 82, 83, 87, 125 e 126.

YU, L.; LUO, C.; YU, X.; JIANG, X.; YANG, E.; LUO, C.; REN, P. Deep learning for vision-based micro aerial vehicle autonomous landing. **International Journal of Micro Air Vehicles**, SAGE Publications Sage UK: London, England, v. 10, n. 2, p. 171–185, 2018. Citado na página 63.

YUAN, Q.; ZHAN, J.; LI, X. Outdoor flocking of quadcopter drones with decentralized model predictive control. **ISA transactions**, Elsevier, v. 71, p. 84–92, 2017. Citado na página 63.

Zhou, W.; Nair, D.; Gunawan, O.; van Kessel, T.; Hamann, H. F. A testing platform for on-drone computation. In: **2015 33rd IEEE International Conference on Computer Design (ICCD)**. [S.l.: s.n.], 2015. p. 732–735. Citado na página 63.

GLOSSÁRIO

Aeronave de Asa Fixa: é um tipo de aeronave capaz de voar usando a asa fixada em seu chassi. Alguns exemplos dessas aeronaves são: Ararinha e Rascal.

Aeronave de Asa Rotativa: é um tipo de aeronave em que sua asa são hélices que giram em torno de um eixo vertical. Alguns exemplos dessas aeronaves são: helicópteros, quadricópteros e multicópteros.

Aileron: é um componente da aeronave, sendo o principal responsável pelo movimento de rotação *roll*.

Aviônicos: representa a toda a eletrônica a bordo da aeronave, como por exemplo, piloto automático, *companion computer*, receptor de rádio controle, módulo de telemetria e GPS.

Companion Computer (CC): é um computador a bordo da aeronave, que auxilia nas tomadas de decisão realizadas durante o voo.

Conventional Take-off and Landing (CTOL): é a aeronave que possui sistema de decolagem e aterrissagem convencional efetuado sobre pistas. Alguns exemplos dessas aeronaves são: Ararinha e Rascal.

Estação de Controle de Solo (GCS): é um software executado em um computador no solo que recebe informações de telemetria do VANT, exibe seu progresso e *status*, e transmite comandos para a aeronave.

Firmware do Piloto Automático: é o programa que controla o hardware do piloto automático determinando como ele irá operar.

Leme: é um componente da aeronave, sendo o principal responsável pelo movimento de rotação *yaw*.

Log de Voo: conjunto de informações, sobre o comportamento da aeronave, salva pelo piloto automático ou outro software durante o voo.

MAVLink: é um protocolo de comunicação utilizado nos pilotos automáticos de VANTs.

Missão: é todo o caminho que a aeronave deve percorrer em voo afim de cumprir seu objetivo. Uma missão pode ser representada por um conjunto de rotas, incluindo o acionamento de mecanismos, como retirada de fotografia e acionamento do pulverizador.

Piloto Automático (AP): equipamento acoplado ao VANT capaz de estabilizar o voo, manter a posição e seguir *waypoints*.

Pitch: representa o movimento de rotação do VANT ao redor do eixo lateral da aeronave.

Ponto-Alvo: é um local específico no espaço, definido por uma coordenada $(lat,\ lng,\ alt)$. Podemos pensar no ponto-alvo como um *waypoint* específico definido pelo projetista da missão.

Profundor: é um componente da aeronave, sendo o principal responsável pelo movimento de rotação *pitch*.

Return To Launch (RTL): é um modo de voo que leva a aeronave de volta ao local onde ela decolou, e então a pousa. Quando acionado faz a aeronave subir até uma altitude h_{rtl} configurado no piloto automático.

Roll: representa o movimento de rotação do VANT ao redor do eixo longitudinal da aeronave.

Rota ou Caminho: existem muitas maneiras de passar do ponto A ao ponto B. O caminho entre esses pontos é conhecido como rota. Uma rota é representada por uma série de pontos, chamados *waypoints*, conectados por linhas retas.

Simulador de Voo: é um software que tenta recriar a realidade existente no voo de uma aeronave simulando tanto o cenário quanto a aeronave.

Simulação HITL: permite executar missões de VANTs em que parte dos equipamentos são simulados e parte são físicos.

Simulação SITL: permite executar missões de VANTs sem nenhum hardware físico, dessa forma, todo o ambiente, hardware da aeronave, do piloto automático e sistema de comunicação são emulados.

Sistema de Telemetria: sistema de comunicação entre o computador em solo e a aeronave no ar capaz de fazer o monitoramento do voo.

Situação Crítica: é qualquer falha interna ou externa ao VANT que coloque em risco a sua aeronavegabilidade.

Throttle: representa a potência (aceleração) aplicada ao motor do VANT.

Trajetória: ao seguir uma rota, o caminho percorrido pode não ser exatamente o mesmo que a rota. O caminho real seguido é chamado de trajetória.

Unidade de Medida Inercial (IMU): é um conjunto combinado de giroscópios e acelerômetros que podem determinar a orientação e a estabilidade da aeronave.

Vertical Take-off and Landing (VTOL): é o veículo aéreo que possui sistema de decolagem e aterrissagem na vertical. Alguns exemplos desses veículos são: helicópteros, quadricópteros, multicópteros e balões a gás.

Waypoint: é um local específico no espaço, definido por uma coordenada $(lat,\ lng,\ alt)$. Uma rota ou missão possui, em geral, um conjunto de *waypoints* estabelecidos.

Yaw: representa o movimento de rotação do VANT ao redor do eixo vertical da aeronave.

APÊNDICE

A

ARQUIVOS DE ENTRADA E SAÍDA DE DADOS

O Código-fonte 1, nomeado de `config-param.properties`, é um exemplo de arquivo de entrada utilizado pelo sistema IFA. Esse tipo de arquivo possui o padrão chave-valor e, em geral, é utilizado para armazenar dados, parâmetros e configurações. A chave é uma representação única de uma propriedade/parâmetro. O IFA, após iniciar a sua execução, atualiza um conjunto de parâmetros de voo do AP, conforme configurado pelo projetista da missão. Essas configurações serão utilizadas pela aeronave durante todo o voo. Esses parâmetros são os mesmos aceitos pelo pilotos automáticos de códigos-fonte abertos, como APM e Pixhawk. Existem várias centenas de parâmetros possíveis de serem utilizados, a lista completa pode ser acessada no link [1].

```
#Formato: CHAVE=VALOR
RTL_ALT=2500.0
WPNAV_RADIUS=50.0
WPNAV_SPEED=200.0
WPNAV_SPEED_UP=50.0
WPNAV_SPEED_DN=50.0
LAND_SPEED=50.0
```

Código-fonte 1 – Arquivo de entrada para configuração dos parâmetros de voo utilizado no AP.

O Código-fonte 2, nomeado de `config-global.properties`, é um trecho contendo um exemplo de arquivo de entrada. Esse arquivo contém 25 parâmetros que podem ser alterados de forma a executar as missões autônomas. O arquivo original possui 116 parâmetros que podem ser definidos. Os parâmetros desse arquivo são utilizados pelos seguintes sistemas: UAV-IFA, UAV-MOSA, UAV-S2DK e UAV-GCS.

```
#Descrição da Propriedade: tipo de simulação
#Tipo de Dado: String    <->    [SITL, HITL, REAL_FLIGHT]
```

[1] <http://ardupilot.org/copter/docs/parameters.html>

```
3  prop.global.operation_mode=REAL_FLIGHT
4
5  #Descrição da Propriedade: tipo de AP usado nos experimentos
6  #Tipo de Dado: String  <->  [APM, PIXHAWK]
7  prop.global.type_ap=PIXHAWK
8
9  #Descrição da Propriedade: tipo de CC usado nos experimentos
10 #Tipo de Dado: String  <->  [EDISON, RASPBERRY, BEAGLE_BONE]
11 prop.global.type_cc=RASPBERRY
12
13 #Descrição da Propriedade: diretório com a missão a ser executada
14 prop.global.mission.dir=../Missions/Thesis/Scenery03
15
16 #Descrição da Propriedade: altitude relativa da missão (metros)
17 prop.global.mission.altitude_relative=12.0
18
19 #Descrição da Propriedade: IP do computador em que o IFA está executando
20 prop.global.comm.host_ifa=localhost
21
22 #Descrição da Propriedade: IP do computador em que o MOSA está executando
23 prop.global.comm.host_mosa=localhost
24
25 #Descrição da Propriedade: possui power module conectado no drone (CC)?
26 #Tipo de Dado: Booleano  <->  [TRUE, FALSE]
27 prop.hw.sensor.has_powermodule=TRUE
28
29 #Descrição da Propriedade: possui câmera conectada no drone (CC)?
30 #Tipo de Dado: Booleano  <->  [TRUE, FALSE]
31 prop.hw.sensor.has_camera=FALSE
32
33 #Descrição da Propriedade: possui buzzer conectada no drone (CC)?
34 #Tipo de Dado: Booleano  <->  [TRUE, FALSE]
35 prop.hw.actuator.has_buzzer=FALSE
36
37 #Descrição da Propriedade: diretório em que está o sistema da câmera
38 prop.hw.sensor.camera.dir=../Modules-Global/Camera/
39
40 #Descrição da Propriedade: duração do vídeo em segundos
41 prop.hw.sensor.camera.video.time=240
42
43 #Descrição da Propriedade: diretório em que está o sistema do buzzer
44 prop.hw.actuator.buzzer.dir=../Modules-Global/Buzzer/
45
46 #Descrição da Propriedade: pino de sinal usado para conectar o buzzer no CC
47 prop.hw.actuator.buzzer.pin=8
48
49 #Descrição da Propriedade: sistema executado pelo drone
```

```
50 #Tipo de Dado: String    <->    [REPLANNER, FIXED_ROUTE, CONTROLLER]
51 prop.ifa.global.system_exec=REPLANNER
52
53 #Descrição da Propriedade: local de execução do método
54 #Tipo de Dado: String    <->    [ONBOARD, OFFBOARD]
55 prop.ifa.replanner.local_exec=ONBOARD
56
57 #Descrição da Propriedade: replanejador de rotas usado pelo drone
58 #Tipo de Dado: String  <->  [DE4s, GH4s, GA4s, MPGA4s, MS4s, GA-GA4s, ...]
59 prop.ifa.replanner.method=GA4s
60
61 #Descrição da Propriedade: tempo de execução do método (em segundos)
62 prop.ifa.replanner.time_exec=1.0
63
64 #Descrição da Propriedade: número de waypoints usado no replanejador
65 prop.ifa.replanner.number_waypoints=30
66
67 #Descrição da Propriedade: risco alocado durante a missão (parâmetro delta)
68 prop.ifa.replanner.delta=0.01
69
70 #Descrição da Propriedade: sistema executado pelo drone
71 #Tipo de Dado: String    <->    [PLANNER, FIXED_ROUTE]
72 prop.mosa.global.system_exec=PLANNER
73
74 #Descrição da Propriedade: local de execução do método
75 #Tipo de Dado: String    <->    [ONBOARD, OFFBOARD]
76 prop.mosa.planner.local_exec=ONBOARD
77
78 #Descrição da Propriedade: método planejador de rotas usado pelo drone
79 #Tipo de Dado: String    <->    [HGA4m, CCQSP4m, M_ADAPTIVE4m]
80 prop.mosa.planner.method=HGA4m
81
82 #Descrição da Propriedade: tempo de execução do método entre cada rota
83 #Formato 1: time          Formato 2: [time1,time2,...timeN]
84 prop.mosa.planner.hga4m.time_exec=[4.0,4.0,4.0,4.0]
85
86 #Descrição da Propriedade: risco alocado durante a missão (parâmetro delta)
87 prop.mosa.planner.hga4m.delta=0.02
```

Código-fonte 2 – Arquivo de entrada para configuração dos parâmetros do sistema autônomo.

O Código-fonte 3, nomeado de `log-overhead-ifa.csv`, é um arquivo de saída de dados que contém um conjunto de informações de *overhead* de comunicação do IFA com AP. Esse arquivo é gerado pelo sistema IFA.

```
1 Type-of-Method;Requisition-URL;Time-In-MilliSeconds
2 POST;/set-parameter/;322
3 GET;/get-parameters/;208
```

```
GET;/get-home-location/;419
GET;/get-all-sensors/;27
GET;/get-distance-to-home/;60
...
GET;/get-all-sensors/;44
```

Código-fonte 3 – Arquivo de saída de *log* do *overhead* de comunicação do IFA com AP.

O Código-fonte 4, nomeado de `log-overhead-mosa.csv`, é um arquivo de saída de dados que contém um conjunto de informações de *overhead* de comunicação do MOSA com AP. Esse arquivo é gerado pelo sistema MOSA.

```
Type-of-Method;Requisition-URL;Time-In-MilliSeconds
GET;/get-parameters/;348
GET;/get-all-sensors/;32
GET;/get-distance-to-home/;11
POST;/set-mission/;16052
...
GET;/get-all-sensors/;37
```

Código-fonte 4 – Arquivo de saída de *log* do *overhead* de comunicação do MOSA com AP.

O Código-fonte 5, nomeado de `log-aircraft.csv`, é um arquivo de saída de dados que contém um conjunto de informações heterogêneas obtidas por diversos sensores e sistemas da aeronave. Esse arquivo é gerado pelo sistema IFA. O Quadro 8 sintetiza as informações dos campos desse arquivo.

```
date;hour;time;lat;lng;alt-rel;alt-abs;voltage-bat;current-bat;level-bat;
    pitch;yaw;roll;vel-x;vel-y;vel-z;fix-type;satellites-visible;eph;epv;
    heading;groundspeed;airspeed;next-wpt;count-wpt;dist-to-home;dist-to-
    current-wpt;mode;system-status;armed;is-armable;ekf-ok;type-failure;est-
    time-to-do-rtl;est-consump-bat-rtl;est-max-dist;est-max-time;dist-sonar;
    temperature-sensor
2018/11/19;15:50:12;5.7;-22.0017092;-47.9330039;-1.80;840.00;12.175;0.00;
    99.0;-0.0970;2.7140;0.0556;-0.02;0.01;0.02;3;10;146;65535;155.0;0.00;
    0.00;0;0;-1.00;-1.00;LOITER;STANDBY;false;false;false;NONE;87.30;13.44;
    1872.97;611.11;NONE;NONE
...
2018/11/19;15:52:11;1.8;-22.0017211;-47.9331374;4.80;850.00;10.317;12.60;
    98.0;-0.0747;2.9450;0.0138;-0.19;0.09;0.19;3;10;145;65535;169.0;0.16;
    0.16;1;12;-1.00;9.02;GUIDED;ACTIVE;true;false;false;NONE;77.58;12.45;
    1854.05;604.94;NONE;NONE
2018/11/19;15:52:11;18.0;-22.0017197;-47.9331384;4.90;850.00;10.221;13.56;
    97.0;-0.1737;3.0550;0.0723;0.02;0.01;-0.02;3;10;145;65535;169.0;0.10;
    0.10;1;12;-1.00;8.86;AUTO;ACTIVE;true;false;false;NONE;77.38;12.42;
    1835.14;598.77;NONE;NONE
...
```

Código-fonte 5 – Arquivo de saída contendo dados da missão obtidos pelos sensores e sistemas.

Quadro 8 – Propriedades do arquivo de saída com dados da missão (`log-aircraft.csv`).

Propriedade	Descrição	Intervalo	Unidade
date	Data do voo no formato AAAA/MM/DD	-	-
hour	Hora do voo no formato HH:MM:SS	$[00:00:00, 23:59:59]$	-
time	Tempo após inicialização do sistema	$[0, +\infty]$	segundos
lat	Latitude da aeronave (posição norte-sul)	$[-90, +90]$	graus
lng	Longitude da aeronave (posição leste-oeste)	$[-180, +180]$	graus
alt-rel	Altitude relativa da aeronave (em relação ao solo)	$[offset, +\infty]$	metros
alt-abs	Altitude absoluta da aeronave (em relação ao mar)	$[offset, +\infty]$	metros
voltage-bat	Voltagem atual da bateria	$[0, offset]$	volts
current-bat	Corrente elétrica atual da bateria	$[0, offset]$	amperes
level-bat	Nível percentual atual da bateria	$[0, 100]$	%
pitch	Ângulo de inclinação ao redor do eixo transversal	$[-\pi, +\pi]$	radianos
yaw	Ângulo de guinada ao redor do eixo vertical	$[-\pi, +\pi]$	radianos
roll	Ângulo de rolamento ao redor do eixo longitudinal	$[-\pi, +\pi]$	radianos
vel-x	Velocidade do VANT em relação a coordenada x	$[0, offset]$	m/s
vel-y	Velocidade do VANT em relação a coordenada y	$[0, offset]$	m/s
vel-z	Velocidade do VANT em relação a coordenada z	$[0, offset]$	m/s
fix-type	Tipo de fixação do GPS em dimensões capturadas	$\{0D, 1D, 2D, 3D\}$	-
satellites-visible	Quantidade de satélites capturados pelo GPS	$[0, +\infty]$	-
eph	Desvio padrão do erro de posição horizontal do GPS	$[0, +\infty]$	metros
epv	Desvio padrão do erro de posição vertical do GPS	$[0, +\infty]$	metros
heading	Ângulo de orientação em coordenadas aeronáuticas	$[0, 360]$	graus
groundspeed	Velocidade do VANT em relação ao solo	$[0, +\infty]$	m/s
airspeed	Velocidade do VANT em relação ao ar	$[0, +\infty]$	m/s
next-wpt	Número do *waypoint* a ser buscado pelo VANT	$[0, offset]$	-
count-wpt	Contador de *waypoints* totais salvos no AP	$[0, offset]$	-
dist-to-home	Distância do VANT ao *waypoint home*	$[0, +\infty]$	metros
dist-to-current-wpt	Distância do VANT ao *waypoint* atual	$[0, +\infty]$	metros
mode	Modo de voo atual do AP	{AUTO, GUIDED, STABILIZE, RTL, LOITER, etc.}	-
system-status	*Status* atual do AP	{UNINIT, BOOT, CALIBRATING, STANDBY, ACTIVE, CRITICAL, EMERGENCY, POWEROFF}	-
armed	O VANT está armado?	$\{true, false\}$	-
is-armable	O VANT é armável?	$\{true, false\}$	-
ekf-ok	O EKF do AP está ok?	$\{true, false\}$	-
type-failure	Tipo de falha crítica detectada	{NONE, SYSTEM_MOSA, SYSTEM_IFA, LOW_BATTERY, BAD_WEATHER, GPS, AP_CRITICAL, AP_EMERGENCY, AP_POWEROFF}	-
est-time-to-do-rtl	Estimativa do tempo de voo para realizar RTL	$[0, offset]$	segundos
est-consump-bat-rtl	Estimativa do consumo de bateria para realizar RTL	$[0, 100]$	%
est-max-dist	Estimativa da distância máxima percorrida	$[0, offset]$	metros
est-max-time	Estimativa do tempo máximo de voo	$[0, offset]$	segundos
dist-sonar	Distância obtida pelo sensor ultrassom	$[0, offset]$	metros
temperature-sensor	Temperatura obtida pelo sensor	$[0, 700]$	^{o}C

Fonte: Elaborada pelo autor.

APÊNDICE

B

ARQUITETURA EMBARCADA BASEADA EM ALGORITMOS GENÉTICOS

Artigo completo publicado na conferência *Genetic and Evolutionary Computation Conference* (GECCO), em 2017, com *qualis* A1 em ciência da computação. Nesse trabalho, foi feita uma análise de desempenho dos algoritmos de planejamento (HGA4m) e replanejamento (MPGA4s) de rotas de VANTs. Esses algoritmos foram embarcados no *companion computer* (CC) Intel Edison. Os resultados obtidos no CC foram comparados com um computador regular Intel i5. Esse trabalho apresenta uma primeira versão dos sistemas MOSA e IFA funcionando em conjunto. Os resultados obtidos nesse artigo foram avaliados através de simulações *Software-In-The-Loop* (SITL). Esse artigo é um resultado direto da presente proposta de doutorado.

An Embedded System Architecture based on Genetic Algorithms for Mission and Safety Planning with UAV

Jesimar da Silva Arantes
University of São Paulo - USP
São Carlos, São Paulo, Brazil
jesimar.arantes@usp.br

Márcio da Silva Arantes
University of São Paulo - USP
São Carlos, São Paulo, Brazil
marcio@icmc.usp.br

Claudio Fabiano Motta Toledo
University of São Paulo - USP
São Carlos, São Paulo, Brazil
claudio@icmc.usp.br

Onofre Trindade Júnior
University of São Paulo - USP
São Carlos, São Paulo, Brazil
otj@icmc.usp.br

Brian C. Williams
Massachusetts Institute of Technology - MIT
Cambridge, Massachusetts, USA
williams@mit.edu

ABSTRACT

The present paper describes an embedded system architecture, based on genetic algorithms, aiming safety mission execution by Unmanned Aerial Vehicles (UAVs). A two-dimensional non-convex environment is considered since obstacle avoidance happens. The embedded system integrates the Mission Oriented Sensor Array (MOSA) and In-Flight Awareness (IFA) systems, where MOSA is responsible for mission accomplishment and IFA stands for flight safety. The features of MOSA and IFA are combined under a platform that applies promising genetic algorithm approaches from literature to reach their goals. First, the genetic algorithms performance running from the embedded system is compared against their performance on a personal computer architecture. Next, the proposed system is evaluated in a real-world scenario using Software-In-The-Loop (SITL) technique. The computational results showed that the embedded system provides reliable results.

CCS CONCEPTS

•**Computer systems organization** → **Evolutionary robotics;** Robotic autonomy; •**Computing methodologies** → *Planning under uncertainty;*

KEYWORDS

Embedded System, Evolutionary Computation, Path Planning, Unmanned Aerial Vehicles

ACM Reference format:
Jesimar da Silva Arantes, Márcio da Silva Arantes, Claudio Fabiano Motta Toledo, Onofre Trindade Júnior, and Brian C. Williams. 2017. An Embedded System Architecture based on Genetic Algorithms for Mission and Safety Planning with UAV. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071302

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GECCO '17, Berlin, Germany
© 2017 ACM. 978-1-4503-4920-8/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3071178.3071302

1 INTRODUCTION

The present paper proposes an embedded system architecture for Unmanned Aerial Vehicle (UAV), based on Genetic Algorithms (GA), aiming mission execution and flight safety. The supervision of mission systems applies the concept of Mission Oriented Sensor Array (MOSA) described in [6], which is focused on those subsystems responsible for mission accomplishment. The supervision of safety systems applies the concept of In-Flight Awareness (IFA) that stands for ensuring flight safety by monitoring in real-time the aircraft operation as proposed by [10].

MOSA concept becomes possible to setup the aircraft for different missions by separating mission systems from other control systems. For instance, the flight navigation path can be included among MOSA attributes. In this case, the mission systems will sense if the aircraft follows the path, making adjustments in the trajectory when necessary. The present paper will include the evolutionary algorithm adapted from [2] within MOSA module, named here as Hybrid Genetic Algorithm for mission (HGA4m). The HGA4m defines a path planning that avoids obstacles within a safety margin of risk, executes it within a short computational time and minimizes fuel consumption. Moreover, the mission system module will be able to provide autonomously reliable trajectories.

On the other hand, IFA concept includes those systems that are monitoring the operation of aircraft. The IFA module takes control in case of any situation that puts the flight at risk, which can happen by internal (subsystems problems) or external (bad weather, intruder aircraft) conditions. The evolutionary algorithm adapted from [1], named here Multi-Population Genetic Algorithm for security (MPGA4s), will be integrated to IFA module. The MPGA4s was designed for path re-planning under critical situations, where the aim is to land the aircraft safely.

The flight occurs in non-convex environment, which means the presence of non-navigable regions. The MOSA and IFA systems along with HGA4m and MPGA4s are embedded in a hardware platform for UAVs. Their performance on such hardware platform is compared against a Personal Computer (PC) architecture. A simulation is also conducted to evaluate the embedded system behaviour using Software-In-The-Loop (SITL) technique.

The paper is organized as follows: section 2 describes some related work and section 3 defines the problem to be studied. The

proposed method is introduced in section 4 and computational results are reported in section 5. The conclusions of this work follow in section 6.

2 RELATED WORK

The ideas and concepts related to MOSA and its basic structure are reported in [5, 6]. A MOSA architecture can be designed to include sensors, processors and communication hardwares as well as to execute a real-time data processing. MOSA is in charge of the path planning execution as reported in [6]. The authors in [5] apply the MOSA concept as reference to design a system that automatically produce thematic maps by processing raw data collected from an array of sensors. A case study was reported with feasible results achieved, where MOSA managed the UAV mission that included to fly until the source of an event and gathering data. The systems designed were able to combine data from a thermal camera and on-the-ground microphone array.

IFA concept is defined in [10] as a supervisor system that monitors all aspects related to the safety of the aircraft. This monitoring is done through a set of sensors that verify the operation of the main components of the aircraft, aiming to mitigate accidents or simply update the flight plan. IFA establishes a model for obtaining situational awareness for unmanned aircraft regarding the operations of its systems as well as the surroundings environment. The aim is to identify autonomously a threat and effectively to avoid or mitigate a possible accident as described in [10]. The authors in [9] proposed an implementation for such concept, named In-Flight Awareness Augmentation System. This system was implemented as part of the autopilot software in a small fixed wing UAV, where it is reported an awareness improvement in safety based on a total of 100 flights.

The systems reported in [5] and [9] were not embedded on any platform, this work contributes in this sense. In addition, there is not a system integrating MOSA and IFA reported in the literature so far, the present paper describes such integration.

An architecture for UAV focused on spray systems against agricultural pests is proposed in [15] that allows the automatic execution of spraying on plantations. A system to control the trajectory is proposed in [12] based on predictive control and considering external perturbations to the flight. Another architecture for trajectory control, combining hardware and software, is developed in [13] which allows the UAV to perform maneuvers avoiding obstacles and other aircraft in real-time. The authors in [11] introduce an optimized implementation based on software to detect failures and avoid degradation of performance in embedded systems that includes UAVs systems. The architectures proposed in [11–13, 15] do not describe a similar division between mission execution systems and awareness systems as proposed here, which can provide more autonomy and resiliency to the UAV systems.

A set of critical situations for UAV flight are taking into account in [1]. These situations lead the UAV to execute a hard landing triggered by equipment failures or environmental situations, where the goal is to define a path that will land the aircraft without risk for people, properties and itself. A Multi-Population Genetic Algorithm, named here as MPGA4s, and a greed heuristic are proposed to define a landing trajectory by taking into account the type of critical situation. The methods are evaluated over a set of 600 maps, where the MPGA4s performs better. Experiments using FlightGear (FG) Flight Simulator are also executed.

The authors in [2] present an evolutionary algorithm, named here as HGA4m, which defines a path planning for UAVs to execute missions in a non-convex environment with uncertainties. HGA4m combines a multi-population genetic algorithm [14] with visibility graph [8] by solving linear programming models to find paths. HGA4m is applied to find paths for a set of 50 maps and its results are compared against an exact and heuristic approaches with competitive results achieved within a short computational time.

In the current paper, HGA4m is integrated to MOSA system while MPGA4s is added to IFA. The autonomous system produced is evaluated running under a hardware platform. The evaluations done here differ from those reported in [1, 2] since the proposed system will perform more complete missions. In addition, the results achieved are also validated by Software-In-The-Loop (SITL) technique.

Hardware-In-The-Loop (HITL) simulations usually combine simulated software systems with physical hardware. The authors in [4] present a HITL framework composed of on-board hardware, flight controller, ground station and integration software for a UAV helicopter. A framework is presented in [7] to run HITL simulations for UAV based on a vision control system. The developed platform has an UAV with on-board autopilot and a camera interconnected to virtual reality software. In the present paper, the simulations are executed using Software-In-The-Loop (SITL), where all aircraft hardware is emulated in software, similarly to the work described in [12].

To sum up, this paper contributes to the development of a more versatile architecture focused on mission execution and security. The idea is to develop a platform that enables the integration of systems like MOSA [5] and IFA [9], mission planning algorithms as HGA4m [2], path re-planning for emergency landing as MPGA4s [1], control of trajectory as described in [12, 13] and the execution of different types of mission as proposed in [15].

3 PROBLEM DESCRIPTION

Figure 1 illustrates how previous literature works lead to the problem approached by this paper. Figure 1 (a) shows a scenario from which the path planning problem was addressed in [3] and [2]. The authors in [3] introduced the Chance-constraint Non-convex Path-planning Problem (CNPP), where an UAV must reach a goal position from a start one by avoiding no-fly zones (NFZ) such as regular airport, populated regions, military bases or storm areas. It is possible to fly over forests and mountains, but uncertainties related to the environment and the UAVs dynamics can deviate the UAV to reach a NFZ or even to hit a mountain. Therefore, a certain level of risk is assumed by chance-constraints and the problem becomes to find a path by optimizing a measure, such as distance, without exceeding the level of risk. The authors in [3] described this problem using a nonlinear mixed-integer programming model, but heuristics are proposed that solve mixed-integer linear programming models. A genetic algorithm is combined with visibility graph to solve the same problem in [2].

Figure 1: Illustrative scenarios of the mission and path planning problem. (a) Example of path planning addressed in [3] and [2]. (b) Example of emergency path re-planning addressed in [1]. (c) Example of mission addressed in this paper.

Figure 1 (b) shows a scenario where the problem of path re-planning under a critical situation was addressed by [1]. The path re-planning is defined for the same chance-constraint and non-convex environment from [2, 3]. However, the type of critical situation is now considered since it may impact in the aircraft's flight ability (UAV dynamic). Thus, the regions are previously mapped and labeled as no-fly zones, where the UAV can not land; bonus regions, where the UAV can land; and penalty regions, where the UAV can land but suffering some damage. More details about this mapping of regions are available in [1]. The MPGA4s method was applied to find a landing trajectory.

Figure 1 (c) illustrates a scenario for mission planning that considers all those aspects together. In this case, path planning is demanded for different starts and goal positions, and it can be done during the flight. Also, during the flight, a critical situation may occur, leading the aircraft to perform an emergency landing. This is the scenario approached by the present paper.

The aspects mentioned above will be properly described by two stochastic mathematical formulations. The first formulation describes the CNPP as introduced by [3].

$$Minimize \quad \sum_t \mathbf{u}_t \cdot \mathbf{u}_t \tag{1}$$

subject to:

$$\mathbf{x}_T = \mathbf{x}_{goal} \tag{2}$$

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t + \omega_t \qquad \forall(t) \tag{3}$$

$$\mathbf{x}_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_{x_0}), \quad \omega_t \sim \mathcal{N}(0, \Sigma_{\omega_t}) \qquad \forall(t) \tag{4}$$

$$Pr\left[\bigwedge_j \bigwedge_t \mathbf{x}_t \notin \mathbb{O}_j^n\right] \geq 1 - \Delta \tag{5}$$

In the above model the state vector $\mathbf{x}_t$ has the vehicle's positions p and velocities v, while the control vector $\mathbf{u}_t$ has acceleration defined by expressions (6).

$$\mathbf{x}_t := [p_x \; p_y \; v_x \; v_y]^T, \quad \mathbf{u}_t := [a_x \; a_y]^T \tag{6}$$

The inputs are initial position ($\hat{x}_0$), goal position ($\hat{x}_{goal}$), dynamic control matrices A and B, time horizon ($t = 0, \cdots T$), time interval (Δt), external uncertainties (ω_t) and level of risk (Δ). The decision variables are UAV states ($\mathbf{x}_t$) and controls ($\mathbf{u}_t$) applied at each time step (t). The objective function (1) is the scalar product of the controls that is minimized during the mission planning. This measure may be related to fuel consumption or distance traveled. The vehicle state ($\mathbf{x}_T$) must reach the goal position ($\mathbf{x}_{goal}$) at the end of this mission by constraint (2). The vehicle state transitions is described in constraints (3), where the first state ($\mathbf{x}_0$) follows a Gaussian distribution of the initial expected state ($\hat{x}_0$) with covariance matrix (Σ_{x_0}). There is an additive Gaussian white noise (ω_t) with covariance matrix (Σ_{ω_t}) applied to each state transition. Constraint (5) stands that vehicle states ($\mathbf{x}_t$) must be out of obstacles ($\mathbb{O}_j^n$) for all time steps t. The expression $Pr[\cdot] \geq 1 - \Delta$ reports that the probability of the vehicle being outside of all obstacles must be greater than or equal to $1 - \Delta$. Thus, the probability of failure must be less than Δ.

The second formulation, introduced in [1], describes all constraints related to the path re-planning problem under critical situation.

$$Minimize \; Pr\left(\bigvee_j \mathbf{x}_T \in \mathbb{O}_j^p\right) - Pr\left(\bigvee_i \mathbf{x}_T \in \mathbb{O}_j^b\right) \tag{7}$$

subject to:

$$\mathbf{x}_{t+1} = F_\Psi(\mathbf{x}_t, \mathbf{u}_t) + \omega_t \qquad \forall(t) \tag{8}$$

$$\mathbf{x}_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_{x_0}), \quad \omega_t \sim \mathcal{N}(0, \Sigma_{\omega_t}) \qquad \forall(t) \tag{9}$$

$$Pr\left[\bigwedge_j \bigwedge_t \mathbf{x}_t \notin \mathbb{O}_j^n\right] \geq 1 - \Delta \tag{10}$$

In the above model the state vector $\mathbf{x}_t$ has the vehicle's positions p, velocities v and angle α while the control vector $\mathbf{u}_t$ has acceleration a and angular variation ε as defined by expressions (11).

$$\mathbf{x}_t := [p_x \; p_y \; v \; \alpha]^T, \quad \mathbf{u}_t := [a \; \varepsilon]^T \tag{11}$$

The inputs include initial position ($\hat{x}_0$) at the moment of failure, nonlinear state transition function (F_Ψ), time horizon ($t = 0, \cdots K$), time interval (Δt), external uncertainties (ω_t) and level of risk (Δ). The decision variables are UAV states ($\mathbf{x}_t$) and controls ($\mathbf{u}_t$) applied at each time step (t). The objective function (7) defines penalties on the chance of landing the aircraft over a penalty region ($\mathbb{O}_j^p$) and rewards for the chance of landing the UAV on bonus regions ($\mathbb{O}_j^b$). The transition function of the vehicle states is described by constraint (8), where the initial state ($\mathbf{x}_0$) follows a Gaussian distribution of the expected state ($\hat{x}_0$) with covariance matrix (Σ_{x_0}). A white Gaussian noise (ω_t) with covariance matrix (Σ_{ω_t}) is applied to each transition. The constraint (10) defines that the vehicle states should be outside the obstacles ($\mathbb{O}_j^n$) for all time step t.

Figure 2: Real world case study for mission and safety.

A case study is also conducted by the present paper in order to verify the joint operation of the mission planning with safe lading under critical situation. Figure 2 describes the scenario for this case study, where a set of waypoints is specified (green color). The aircraft start its route at point A and must go through each of these waypoints. The region highlighted by red color is the border of our safe region (navigable area). It is also shown a set of regions that was previously mapped. These regions are represented by convex polygons, where the gray color represents NFZ regions, blue color represents bonus regions (safe landing area) and the orange color represents penalty regions (landing area with some damage for UAV).

The scenario in Figure 2 allows us to give more details about how the problem is approached by this paper:

Mission Plan: *The UAV should start its flight at the point **A** and proceed to the goal points **B**, **C**, **D**, **E**, **F**, **G**, **H** and then conclude its flight at **A**. This sequence of points determines the whole mission to be followed by the UAV that will take pictures at each region. However, the next goal can be provided to the aircraft during the flight.*

The tasks of MOSA and IFA systems during this mission are described next:

MOSA System: *MOSA system is responsible for fulfilling the mission defined above. In this way, it first plans the route that minimizes the aircraft's fuel consumption between waypoints **A** and **B**. Next, the system oversees the implementation of the route by autopilot as well as the right instant to start taking pictures. At the same time, the system also defines the next route between waypoints **B** and **C**, and so on during the mission execution.*

IFA System: *During the mission, the internal states of the aircraft is being monitored by IFA system. The UAV may present a failure, such as battery overheating, due to unforeseen event on the aircraft. The supervisor system (IFA) is in charge to detect such failure and it may decide to abort the current mission. In this case, the UAV control executed by MOSA is interrupted and an algorithm for path re-planning is executed aiming to define a landing route.*

4 METHODS

The architecture of the embedded system proposed is shown in Figure 3.

Figure 3: Embedded system architecture in the aircraft.

The system has four main hardware components: actuators, autopilot, sensors and embedded processor. Actuators allow the displacement and maneuvers execution of the UAV; the sensors are responsible for capturing information from the environment aiming to aid navigation, mission execution and flight; the autopilot is responsible for navigation keeping the aircraft in the route; and the dual core embedded processor is dedicated to executing MOSA and IFA systems.

The main modules of MOSA are data acquisition, sensor fusion, mission control, camera control, communication control and path planning. The present paper focuses in the this last module where HGA4m is integrated. The main components of the IFA are data acquisition, sensor fusion, security control, decision making, communication control and path re-planning where MPGA4s is integrated. The path planning and re-planning modules are the ones that consume more processing in this embedded architecture, since they solve non-convex path planning problems.

Figure 4 illustrates the state machine of path planning module in MOSA system.

Figure 4: State machine of path planning module in MOSA.

Initially the subsystem is kept in *Waiting* state from which it can switch to *Processing Route* state, where HGA4m is triggered, changing its state to *Ready Route*, when the new path is sent to autopilot. The *Disabled* state can be reached always a critical situation happens, leading MOSA modules to abort the current processing. Figure 5 shows the state machine of path re-planning module in IFA system.

Figure 5: State machine of path re-planning module in IFA.

Path re-planning module starts from *Monitoring* state and changes to *Processing Route* state in case of a system failure. As soon as the route is returned by MPGA4s, the current state becomes *Ready Route* where the trajectory is sent to autopilot. Next, the system returns to *Monitoring* state. This module can enter the *Disabled* state since an event related to a severe fault happens. For instance, such severe fault can lead IFA to abort all systems to trigger the aircraft parachute.

5 COMPUTATIONAL RESULTS

This section discusses the computational results obtained in two experiments. The first experiment aims to evaluate HGA4m and MPGA4s performance when running under a hardware platform. The second experiment is a more specific case study where simulations are conducted with the proposed system architecture. Table 1 shows parameter values set for HGA4m and MPGA4s, which are practically the same reported in [1, 2]. The only exception is the stopping criterion for MPGA4s that has been changed from number of evaluations to time limit. This change is important to ensure that the method returns an emergency route within a short computational time. The crossover rate is 5.0 for HGA4m and 0.5 for MPGA4s, which means HGA4m and MPGA4s will generate, respectively, 195 and 19 new individuals (*crossRate*popSize*, see [1, 2]) at each generation. The mutation rates are seemingly high since the mutation operators designed are very subtle.

Table 1: Settings used in the HGA4m and MPGA4s method.

Parameters	Value HGA4m	Value MPGA4s
number of populations	3	3
population size	3x13	3x13
crossover rate	5	0.5
mutation rate	0.7	0.75
stopping criterion	10 sec	1 sec

The parameter settings $\hat{x}_0$, $\mathbf{x}_{goal}$, T, Δt, Δ, A, B, Σ_{x_0} and Σ_{ω_t} used in the mathematical models of the section 3 are the same used for HGA4m described in [2]. The same occurs for parameter settings $\hat{x}_0$, T, Δt, Δ, F_Ψ, Σ_{x_0} and Σ_{ω_t} in MPGA4s as defined in [1].

A total of 40 maps is randomly generated, using the generator proposed in [3], for the experiments with HGA4m. The evaluation of MPGA4s method is conducted over a set of 60 maps, where there are 10 maps for each type of instance as described in [1]. The experiments are designed and calibrated based on a fixed wing UAV called Ararinha. The parameters related to Ararinha are depicted in Table 2. During the experiments using SITL with FlightGear (FG) simulator, the Rascal 110 aircraft is used due to its similarity to the Ararinha.

Table 2: Technical specifications of the Ararinha and Rascal.

Component	Value Ararinha	Value Rascal 110
Wingspan	1.90m	2.79m
Length	1.15m	1.92m
Electric Power	740W	1600W
Weight	2.83kg	4.99kg
Payload	0.60kg	0.91kg
Endurance	15 minutes	23 minutes

5.1 Evaluating HGA4m

The results running HGA4m 10 times over each one of those 40 maps is reported in this section. HGA4m is executed in two different hardware platforms. The first is a dual core Intel Edison computing platform with 500 MHz, 1 GB RAM and Linux - Yocto operating

system. The second is a regular Personal Computer (PC) running under an Intel Core i5 processor with 1.8 GHz, 4 GB RAM and Linux - Ubuntu 16.04 operating system. Two main metrics are compared: number of fitness evaluations and length of the path returned. Figure 6 shows the number of fitness evaluations on average after 10 executions over each instance.

Figure 6: Number of evaluations by instance in the HGA4m.

The number of evaluations is reduced when running HGA4m under Edison as expected. This means that HGA4m exploits less the search space than running in a regular PC. However, the solution quality in terms of path length is practically the same as shown by Figure 7. This second metric indicates that the quality of the routes returned is maintained even running under an inferior hardware platform. Thus, even exploring less the search space under Edison, the path lengths returned were similar to those achieved under PC platform.

Figure 7: Path length by instance in the HGA4m method.

Table 3: Kruskal-Wallis test for HGA4m

Metric	Architecture	p-value
Evaluations	PC i5 Edison	< 0.0001
Path Length	PC i5 Edison	0.185

The Kruskal-Wallis non-paramtetric test is performed to measure statistically the results in Figures 6 and 7 as show on Table 3. There is a significant different between the results returned from PC i5 and Edison as can be seen by $p\text{-}value < 0.0001$ regarding evaluation criterion. However, there is no significant difference detected regarding length-path criterion.The criteria of independence, homoscedasticity and normality must be satisfied to apply parametric tests. If at least one criterion is not satisfied, we must use non-parametric tests. The Kolmogorov-Smirnov test was first performed and showed that the data from Figures 6 and 7 do not follow a normal distribution.

5.2 Evaluating MPGA4s

This section evaluates MPGA4s on both PC i5 and Edison platforms by solving the set of 60 maps. Figure 8 shows the number of evaluations (fitness function evaluations).

Figure 8: Number of evaluations by instance in the MPGA4s.

There are in average 805 fitness evaluations running MPGA4s with Edison platform, while 34,440 evaluations are performed running the PC platform. The PC platform executes around 42 times more evaluations than Edison, but the quality performance remains satisfactory as shown in Figure 9 and Table 4.

Figure 9: Landing sites in both architectures for MPGA4s.

The landing place is highlighted in [1] as a straight way to measure the solution quality. Figure 9 illustrates the locations where the landing place occurred from paths returned by MPGA4s in both architectures. There are six set of instances (I1, I2, $\cdots$ I6)

where MPGA4s is evaluated taking into account 4 critical situations described in [1]: motor failure (Ψ_m), battery overheating (Ψ_b), right wing problem (Ψ_{s1}) and left wing problem (Ψ_{s2}). The blue color indicates landing place in bonus region, light gray indicates landing in remaining region, and dark gray color indicates violation of the non-navigability restriction. This image reveals a great similarity between the landing sites for paths returned from both architectures.

Table 4 presents numerically the results of Figure 9. It can be seen that about 75.4% of the solutions landed on a safe region for paths returned by Edison, against 85.8% in PC platform. The PC solutions performed only 10.4% better than the embedded solutions and the number of violations of the non-navigability condition on paths from both platforms is practically the same.

The above results reveal that a hardware architecture as Edison becomes possible to save the aircraft in 75.4% of the critical situations that would cause its inevitable fall. The results also show that even if it were possible to configure all hardware from a PC platform in the aircraft, despite problems related to PC weight restrictions and energy consumption, the improvement achieved is around 10.4% on the chances to save the aircraft.

Table 4: Summarizing the landing sites of the MPGA4s.

Region	PC i5	Edison
bonus	206 (85.8%)	181 (75.4%)
remainder	19 (7.9%)	43 (17.9%)
penalty	0 (0.0%)	0 (0.0%)
no-fly zone	15 (6.2%)	16 (6.6%)
total	240	240

The Kruskal-Wallis test is performed again to measure statistically the results from Figure 8 and the results are showed in Table 5. The non-parametric test was justified since these data also do not follow a normal distribution by Kolmogorov-Smirnov test. There is a significant difference between MPGA4s performance comparing both hardware platform.

Table 5: Kruskal-Wallis test for MPGA4s

Metric	Architecture	p-value
Evaluations	PC i5 Edison	< 0.0001

Figure 10: Results of case study in a real world scenario.

Figure 11: Experiment with SITL to validating routes.

5.3 Results of the Case Study

The present section evaluates the system proposed in section 4, where a real world case study for mission and safety planning is conducted. In this experiments, the system performance is also compared when running under Edison and PC i5 platforms. Figure 10 shows the results achieved.

The complete route planned with HGA4m method is represented by a green line, where the UAV must fulfill the mission plan as defined in section 3. This route is extremely similar for both architectures and does not present significant changes. The analysis of this route shows that the mission is executed without violation of non-navigable regions.

Figure 10 also shows points labeled by **F1**, **F2**, **F3**, **F4** and **F5**, which are places where a critical fault has been detected. Thus, the proposed system is evaluated when a failure happens for each one of these points. The overheating of battery is assumed as failure, forcing the IFA system to perform an emergency landing. In this case, the blue line indicates the emergency landing paths returned by Edison. The red line indicates the emergency landing route from PC platform. All routes are able to safely bring the

Table 6: Different simulations performed using SITL to validate the trajectories obtained by HGA4m and MPGA4s methods.

Architecture Evaluated	Methods Evaluated	Description	Web Link
PC i5	HGA4m	Full route without failure critical	https://youtu.be/D22r8qZ4Wmo
Edison	HGA4m	Full route without failure critical	https://youtu.be/O-xMC51w5ec
PC i5	HGA4m and MGPA4s	Partial route with route of failure in **F5**	https://youtu.be/Vzt6sZgFhL0
Edison	HGA4m and MGPA4s	Partial route with route of failure in **F5**	https://youtu.be/rTtg3ANmNvw

aircraft to the ground, landing over bonus regions without to fly over no-navigable regions.

The Software-In-The-Loop (SITL) technique is used to validate these trajectories, where the hardware of autopilot is simulated. This type of experiment allows to verify the behavior of the aircraft in conditions closer to one found in real flight, since the maneuvers of the aircraft are more realistically executed. Figure 11 presents the result obtained when the mission is executed without any fail. The trajectories followed by the UAV during these simulations present a small deviation from the path planned by HGA4m. In this case, the path planned executing the system from Edison or PC platforms is extremely similar as already mentioned, thus, only one path planned is shown.

The videos in Table 6 show four of the 12 experiments conducted using SITL, including when failures happen, running the system from Edison. It is possible to see that the interaction between the embedded system modules works properly, where the time consumption running HGA4m and MPGA4s does not compromise the experiments.

6 CONCLUSIONS

This paper described and evaluated an embedded system architecture based on genetic algorithms, where HGA4m and MPGA4s were integrated within MOSA and IFA systems. First, the HGA4m and MPGA4s performances running under Edison and PC platforms are compared. The results achieved by the embedded architecture was very promising with a reduced loss of quality in the solutions found from Edison. Such result indicates the robustness of the evolutionary algorithms to find good solutions despite the limitation of the hardware platform. The complete embedded system integrating MOSA and IFA also returned promising results for the execution of a mission. The system implementation was evaluated in a case study by comparing the results also running under Edison and PC platforms. SITL simulations were performed in order to validate the routes obtained by the system. The experiment allowed to verify the behavior of the aircraft in conditions close to one found in an real flight. The simulation using Edison platform returned paths able to be executed with the maneuvers capability of the aircraft. The results presented in this work indicate that genetic algorithms, running under an embedded system, can provide reliable solutions for autonomous decision making during real UAV flights.

7 ACKNOWLEDGMENTS

This paper acknowledges Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), with financial support from projects 2015/23182-2, 2014/11331-0 and 2013/26091-2 and MIT-Brazil Seed Fund, project Study and Development of Methodologies for Applications in Unmanned Aerial Vehicles, with support from the Itaú Fund for research on sustainability in Latin America via the Sustainability Initiative at MIT Sloan. This paper acknowledges too Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) by financial support.

REFERENCES

[1] Jesimar da Silva Arantes, Márcio da Silva Arantes, Claudio Fabiano Motta Toledo, and Brian Charles Williams. 2015. A Multi-Population Genetic Algorithm for UAV Path Re-Planning under Critical Situation. In *Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conference on*. IEEE, 486–493.

[2] Márcio da Silva Arantes, Jesimar da Silva Arantes, Claudio Fabiano Motta Toledo, and Brian C Williams. 2016. A Hybrid Multi-Population Genetic Algorithm for UAV Path Planning. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*. ACM, 853–860.

[3] Lars Blackmore, Masahiro Ono, and Brian C. Williams. 2011. Chance-Constrained Optimal Path Planning with Obstacles. IEEE Press. (2011).

[4] Guowei Cai, Ben M Chen, Tong H Lee, and Miaobo Dong. 2009. Design and implementation of a hardware-in-the-loop simulation system for small-scale UAV helicopters. *Mechatronics* 19, 7 (2009), 1057–1066.

[5] N. Figueira, I. Freire, O. Trindade, and E. Simões. 2015. Mission-Oriented Sensor Arrays and UAVs - A Case Study on Environmental Monitoring. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* (2015), 305–312. DOI:http://dx.doi.org/10.5194/isprsarchives-XL-1-W4-305-2015

[6] N. Figueira, O. Trindade, A. L. P. Mattei, and L. Neris. 2013. Mission Oriented Sensor Arrays – An Approach towards UAS Usability Improvement in Practical Applications. In *5th European Conference for Aeronautics and Space Sciences (EUCASS)*.

[7] NR Gans, WE Dixon, R Lind, and A Kurdila. 2009. A hardware in the loop simulation platform for vision-based control of unmanned air vehicles. *Mechatronics* 19, 7 (2009), 1043–1056.

[8] Yoshiaki Kuwata. 2003. *Real-time Trajectory Design for Unmanned Aerial Vehicles using Receding Horizon Control*. 151 p. Massachusetts Institute of Technology.

[9] A. L. P. Mattei, A. M. Cunha, L. A. V. Dias, P. C. S. Euphrasio, O. Trindade, and C. M. Toledo. 2015. IFA2S – In-flight Awareness Augmentation Systems. In *Information Technology - New Generations (ITNG), 2015 12th International Conference on*. 95–100. DOI:http://dx.doi.org/10.1109/ITNG.2015.21

[10] A. L. P. Mattei, E. Fonseca, N. M. Figueira, O. Trindade, and F. Vaz. 2013. UAV In-Flight Awareness: A Tool to Improve Safety. In *5th European Conference for Aeronautics and Space Sciences (EUCASS)*. Munich.

[11] Andrey Morozov and Klaus Janschek. 2016. Flight Control Software Failure Mitigation: Design Optimization for Software-implemented Fault Detectors. *IFAC-PapersOnLine* 49, 17 (2016), 248 – 253. DOI:http://dx.doi.org/10.1016/j.ifacol.2016.09.043 Symposium on Automatic Control in Aerospace.

[12] Ionela Prodan, Sorin Olaru, Ricardo Bencatel, João Borges de Sousa, Cristina Stoica, and Silviu-Iulian Niculescu. 2013. Receding horizon flight control for trajectory tracking of autonomous aerial vehicles. *Control Engineering Practice* 21, 10 (2013), 1334 – 1349. DOI:http://dx.doi.org/10.1016/j.conengprac.2013.05.010

[13] Subramanian Ramasamy, Roberto Sabatini, Alessandro Gardi, and Jing Liu. 2016. {LIDAR} obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid. *Aerospace Science and Technology* 55 (2016), 344 – 358. DOI:http://dx.doi.org/10.1016/j.ast.2016.05.020

[14] C.F.M. Toledo, P.M. Franca, A. Kimms, and R. Morabito. 2009. A multi-population genetic algorithm approach to solve the synchronized and integrated two-level lot sizing and scheduling problem. *International Journal of Production Research* 47 (2009), 3097–3119.

[15] Xinyu Xue, Yubin Lan, Zhu Sun, Chun Chang, and W. Clint Hoffmann. 2016. Develop an unmanned aerial vehicle based automatic aerial spraying system. *Computers and Electronics in Agriculture* 128 (2016), 58 – 66. DOI:http://dx.doi.org/10.1016/j.compag.2016.07.022

APÊNDICE

C

AVALIAÇÃO DE PLATAFORMAS DE HARDWARE PARA REPLANEJAMENTO DE ROTA

Artigo completo publicado na conferência *IEEE International Conference on Tools with Artificial Intelligence* (ICTAI), em 2017, com *qualis* B1 em ciência da computação. Esse trabalho faz uma análise de desempenho de diferentes estratégias para replanejamento de rota em caso emergencial quando embarcados no *companion computer* (CC) Intel Edison. Os resultados obtidos no CC foram comparados com um computador regular Intel i5. As seguintes estratégias foram implementadas, analisadas e comparadas: GH4s, GA4s, GA-GA-4s e GA-GH-4s. Nesse estudo, foi destacada a arquitetura do sistema IFA, em que o replanejamento de rotas é efetuado. Os resultados obtidos foram avaliados através de simulações *Software-In-The-Loop* (SITL). Esse artigo é resultado direto da presente proposta de doutorado.

2017 International Conference on Tools with Artificial Intelligence

Evaluating Hardware Platforms and Path Re-Planning Strategies for the UAV Emergency Landing Problem

Jesimar da Silva Arantes, Márcio da Silva Arantes, André Badawi Missaglia,
Eduardo do Valle Simões and Claudio Fabiano Motta Toledo
University of São Paulo, USP, São Carlos, Brazil
Email: {jesimar.arantes, andrebm}@usp.br
Email: {marcio, claudio, simoes}@icmc.usp.br

***Abstract*—The present paper evaluates some strategies to apply path re-planning algorithms for emergency landing of Unmanned Aerial Vehicles (UAVs). The strategies proposed are based on executing only one path re-planning algorithm or a combination of them using an ensemble approach. The proposed methods are integrated to the security supervision system of the UAV, namely In-Fly Awareness (IFA), where decision-making algorithms are applying after a critical situation happens. The critical situations considered are battery overheating and engine failure. The performance of the proposed strategies for emergency landing is evaluated in a real world scenario using the Software-In-The-Loop (SITL) technique. Computational results reported show that these strategies are promising based on the success rate to land the UAV safely.**

***Keywords*—Evolutionary Computation; Unmanned Aerial Vehicles; Path Re-Planning; Embedded Applications**

I. Introduction

Unmanned Aerial Vehicles (UAVs) are broadly used in many real world applications such as agriculture, transportation, logistics, surveillance, among others [1]. Although UAVs are becoming more robust in terms of processing power, autopilot (AP), embedded sensors and flight time, their autonomy level can be improved through, e.g., decision-making capability during the flight.

The autonomy of UAVs may vary from a ground control system with a human pilot to a fully autonomous flight [2]. In the case of human pilot, the authors in [3] describe that a reliable communication is relevant, otherwise, the remote control can break away and the aircraft will not be able, e.g., to re-planning a mission under a critical situation.

This work intends to improve the UAV autonomy even facing critical situations, where the path re-planning problem for fixed wing UAVs is solved. This problem is based on the chance-constrained non-convex problem approached in [4], [5]. The proposed methods extends studies conducted by [6]–[8].

The path re-planning begins after a critical failure has been detected by the aircraft supervisor system, which is composed by a collection of sensors responsible to monitor the aircraft health and report any problem identified. Hence, after detecting a failure, an emergency landing must be performed aiming to avoid any damage to people, properties and aircraft itself.

The present paper is organized as follows: Section II presents some related work and Section III defines the problem to be studied. The proposed method is introduced in Section IV and computational results are reported in Section V. The conclusions of this work follow in Section VI.

II. Related Work

There are many applications of Artificial Intelligence (AI) techniques with UAV systems as reported in [9]–[11]. Systems for wildlife detection with UAV are evaluated in [9] where an automated detection is proposed by using thermal image acquisition and video processing. Mission planning and re-planning problems are approached by [10] applying AI techniques to reduce workload of operators. The methods proposed emphasize the improvements achieved for UAV operations from the ground control stations. The advantages of AI and control systems for UAVs autonomy are discussed in [11].

The present paper proposes autonomous systems to make decision about path re-planning based on the Chance-constrained Non-convex Path planning Problem (CNPP). The authors in [4], [5] introduce the CNPP and apply methods based on mathematical programming techniques to solve it. A hybrid method combining multi-population genetic algorithm with visibility graph is described in [7] to solve the CNPP, where improvements are reported planning trajectories from a large set of maps. Path re-planning for UAV is approached by [12] where new targets can become available during the UAV mission execution. A path re-planning heuristic is proposed to define a trajectory that will improve the chance to reach foreseen and unforeseen targets. However, these previous works do not solve path re-planning problem under critical situation.

The Emergency Landing Planner (ELP) is described in [13], [14], which assists aircraft pilots to find a best landing place. If a critical situation occurs, ELP applies an A* hybrid algorithm to execute a path re-planning. Computational tests were conducted using a flight simulator for large aircraft. A emergency landing planner for UAVs is introduced by [6], [8]

2375-0197/17/31.00 ©2017 IEEE
DOI 10.1109/ICTAI.2017.00144

with evolutionary algorithms and a greedy heuristic applied to find a safe landing path. The evolutionary algorithms are able to find good quality solutions, while the greedy heuristic returns feasible solutions within a short computational time.

The UAV hardware can be designed to integrate different embedded systems by using companion computers such as Intel Edison, Intel Aero, Raspberry Pi, BeagleBone, ODroid, among others. The companion computer is an efficient way to link the embedded systems with autopilot, where a Linux-based computer may be used to run a tool such as DroneKit-Python [1]. The authors in [15] develop a framework with Raspberry Pi to perform image processing for object detection, while image tracking with embedded computer ODroid XU4 is done by [16]. A BeagleBone Black is used by [17] with on-board image processing in UAV to execute missions over a sugar cane plantation. An application for crowd surveillance with UAV and Internet of Things (IoT) is described in [18], where the UAV platform is setup with an autopilot, Raspberry Pi and video camera. The acquired images are processed using face recognition algorithms. An evolutionary algorithm for emergency landing was implemented and evaluated in [19] using two different hardware platforms: a personal computer and an Intel Edison.

Mission Oriented Sensor Array (MOSA) and In-Fly Awareness (IFA) systems are employed in the present paper using as companion computer an Intel Edison platform. MOSA was introduced by [20] and it is responsible for mission execution by processing data in real-time and keeping the aircraft on track. IFA was proposed by [21] as a supervisor system responsible for detecting failures, taking control of the aircraft under critical situations. According to the website [2], the critical failures that more often cause UAVs to crash are engine, mechanical, electrical (battery) failures as well as pilot error, loss of link and bad weather. Thus, the present paper will include the path re-planning for emergency landing module inside IFA system.

III. Problem Description

Figure 1 illustrates the problem approached by this work. Let's suppose an UAV that executes aerial imaging mission over several crop fields in a farm, where there are also a house, warehouse, granary, two tractors, cattle shed and UAV runway. The whole imaging mission is previously planned on the ground such that the UAV autopilot is setup to follow waypoints. Thus, the flight is semi-autonomous since a human pilot is not needed to control the aircraft. The pilot is still needed to monitor the whole mission once unforeseen events may happen and the aircraft can not be able to deal with such unexpected or critical situation. However, the pilot may lose contact with the aircraft, as a consequence of the failure, or he/she can not be able to decide fast enough the best place to land the aircraft, damaging it during the emergency landing process. The present work proposes an autonomous decision-making device to avoid human intervention when a critical situations happens. The idea is to develop a system able to decide where to land the aircraft quickly and safely.

Fig. 1. Illustrative scenario of emergency path re-planning.

The critical situations approached are battery overheating and engine failure, which trigger the IFA system to take control of the aircraft, switching from the current path to an emergency landing. The path re-planning problem for critical situations approached here is similar to one addressed in [6], however, this work focuses on two hardware failures: battery and engine problems.

The scenario describe in Figure 1 is non-convex since the UAV can not fly above some regions that will be considered obstacles as proposed in [4]. Therefore, the house, warehouse and granary become no-fly regions. The regions with cattle shed, crop fields and two tractors are considered as penalty regions for landing. It means that the UAV may land there, but it will probably cause some damage. Finally, the runway was considered a bonus region which means the perfect place to land in case of critical situation. The black route in Figure 1 indicates the current mission being executed until a critical situation has been detected. At that point, the IFA system executes a re-planning algorithm and a new route is planned, while the aircraft is still flying a critical distance (d_c). The so-called critical distance is travelled until the safe route to land (red line) has been found. Hence, the remain original route (gray line) is aborted when the safe route is setup to the autopilot.

During the path re-planning, there are many constraints to be taking into account. These constraints will be better described by the stochastic mathematical model introduced in [6] and showed next.

$$Minimize\ Pr\Big(\bigvee_j \mathbf{x}_T \in \mathbb{O}_j^p\Big) - Pr\Big(\bigvee_i \mathbf{x}_T \in \mathbb{O}_j^b\Big) \quad (1)$$

subject to:

[1] http://ardupilot.org/copter/docs/common-companion-computers.html
[2] https://dronewars.net/drone-crash-database/

$$\mathbf{x}_{t+1} = F_{\Psi}(\mathbf{x}_t, \mathbf{u}_t) + \omega_t \qquad \forall(t) \tag{2}$$

$$\mathbf{x}_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_{x_0}), \quad \omega_t \sim \mathcal{N}(0, \Sigma_{\omega_t}) \qquad \forall(t) \tag{3}$$

$$Pr\left[\bigwedge_j \bigwedge_t \mathbf{x}_t \notin \mathbb{O}_j^n\right] \geq 1 - \Delta \tag{4}$$

The state vector $\mathbf{x}_t$ has the positions p of vehicle, velocities v and angle α, while the control vector $\mathbf{u}_t$ has acceleration a and angular variation ε as defined by expressions (5).

$$\mathbf{x}_t := [p_t^x \ p_t^y \ v_t \ \alpha_t]^T, \quad \mathbf{u}_t := [a_t \ \varepsilon_t]^T \tag{5}$$

The mathematical model inputs include initial position ($\hat{x}_0$) at the moment of failure, nonlinear state transition function (F_{Ψ}), time horizon ($t = 0, \cdots T$), time interval (Δt), external uncertainties (ω_t) and level of risk (Δ). The decision variables are UAV states ($\mathbf{x}_t$) and controls ($\mathbf{u}_t$) applied at each time step (t). The objective function (1) defines penalties on the chance of landing the aircraft over a penalty region ($\mathbb{O}_j^p$) and rewards on the chance of landing the UAV in bonus regions ($\mathbb{O}_j^b$). The transition function of the vehicle states is described by constraint (2), where the initial state ($\mathbf{x}_0$) follows a Gaussian distribution of the expected state ($\hat{x}_0$) with covariance matrix (Σ_{x_0}). A white Gaussian noise (ω_t) with covariance matrix (Σ_{ω_t}) is applied to each transition. The constraint (4) defines that the vehicle states should be outside the obstacles ($\mathbb{O}_j^n$) for all time step t.

The aircraft dynamics change when a critical situation occurs and such change has to be considered when planning the emergency landing. The notation Ψ_b means a battery problem, whereas Ψ_m indicates a engine failure. The state transition, given by equation 6, is applied when a battery problem happens (F_{Ψ_b}) and the equation 7 is setup for engine failure (F_{Ψ_b}). Equation 7 assumes the absence of acceleration since the engine does not work anymore. The dynamic functions allows to model the behaviour of the UAV. The component $\frac{F_t^d}{m}$ describes the drag force exerted on the aircraft, reducing at each instant its speed. The two-dimensional modelling aims to lead the aircraft to the bonus region. The position component in the third axis (p_t^z) is calculated afterwards by using a linear decay for the altitude.

$$F_{\Psi_b}(\mathbf{x}_t, \mathbf{u}_t) = \begin{bmatrix} p_t^x + v_t \cdot cos(\alpha_t) \cdot \Delta t + a_t \cdot cos(\alpha_t) \cdot (\Delta t)^2/2 \\ p_t^y + v_t \cdot sin(\alpha_t) \cdot \Delta t + a_t \cdot sin(\alpha_t) \cdot (\Delta t)^2/2 \\ v_t + a_t \cdot \Delta t - \frac{F_t^d}{m} \cdot \Delta t \\ \alpha_t + \varepsilon_t \cdot \Delta t \end{bmatrix} \tag{6}$$

$$F_{\Psi_m}(\mathbf{x}_t, \mathbf{u}_t) = \begin{bmatrix} p_t^x + v_t \cdot cos(\alpha_t) \cdot \Delta t \\ p_t^y + v_t \cdot sin(\alpha_t) \cdot \Delta t \\ v_t - \frac{F_t^d}{m} \cdot \Delta t \\ \alpha_t + \varepsilon_t \cdot \Delta t \end{bmatrix} \tag{7}$$

A real-world case study was done executing aerial imaging in a farm as shown by Figure 2. A mission was design using the Ground Control Station (GCS) Mission Planner, where a set of waypoints was established as depicted by the green route with length equal to 4,150m. In an ideal setting, the UAV takes off from the runway, executes its mission and returns to the runway. However, a fail can happen and an intelligent system is necessary to find quickly a safe place to land the aircraft. Therefore, a set of regions was previously mapped as illustrated in Figure 2, where there are three bonus regions (blue polygons), two no-fly regions (gray polygons) and three penalty regions (orange polygons) as proposed in [4], [5]. Nevertheless, non-convex regions can be modeled as a set of convex polygons as shown by the two polygons in orange that are side by side. The red region indicates the border of the mapped area with dimensions 675m x 940m. Mapping the region beforehand avoids to execute image processing in the UAV, which would consume a lot of computational power from a small aircraft.

Fig. 2. Real world case study for aerial imaging application.

IV. Methods

The embedded system architecture is based on one proposed in [19] and illustrated by Figure 3. The system has six main hardware components: actuators, autopilot, sensors, communication, power and embedded processor. The actuators allow maneuvers of the UAV; the sensors are responsible for capturing information from the environment to aid mission execution and flight; the autopilot is responsible for navigation by keeping the aircraft in the desired route; communication is responsible for receiving commands from the radio control and sending the aircraft data through the telemetry modem; the battery is responsible for powering the components throughout the flight; and the dual core embedded processor is dedicated to executing IFA and MOSA systems.

The main components of the IFA are data acquisition, sensor fusion, security control, decision making, communication

Fig. 3. Embedded system architecture in the aircraft.

Fig. 4. Embedded system architecture in the path re-planning module.

Fig. 5. Implemented strategies running methods in parallel. (a) Combining of GA e GH. (b) Two executions of GA.

control and path re-planning where a set of algorithms is integrated. The main modules of MOSA are data acquisition, sensor fusion, mission control, camera control, communication control and path planning. The present paper focuses in the path re-planning module of IFA, which is one that most consumes processing power in this embedded architecture. This happen once real-time decisions under a critical situation have to be made as soon as possible, where many paths have to be processed within a short time to return the best one.

Figure 4 details the path re-planning module of the IFA system. After detecting a failure, the IFA system triggers the path re-planning module by sending the inputs: Initial State, Map, Dynamic Model, Failure, and Config to the re-planning algorithm. In the path re-planning module, one of the following strategies can be selected: Greedy Heuristic (GH), Genetic Algorithm (GA), combination of GA executing in parallel (GA-GA) and combination of GA with GH executing in parallel (GA-GH). The selected strategy will output the following information: Set of Waypoints, Landing Site, Probability of Landing and Fitness. These outputs return to the path re-planning module that commands the autopilot to abort the current route (mission) and to follow the new one.

In Section V, the ensemble strategies will be compared against the single execution of GH and GA. Figure 5 shows the execution flow of the proposed ensemble strategies. After a critical situation has been detected, two methods are executed in parallel using both available cores and the best solution is returned at the end. Figure 5 (a) shows the ensemble strategy that uses a combination of GA and GH (GA-GH), while the method in Figure 5 (b) runs two GAs in parallel (GA-GA).

The Greedy Heuristic (GH) aims to return feasible solutions fast enough as described by Algorithm 1. This approach becomes relevant when a viable solution is more demanded than time-consuming better solutions. In lines 3-5, paths for each bonus regions are initialize and evaluated. A path is determined first by rotating the UAV to reach a bonus region. Next, the procedure defines a path as a straight trajectory to the bonus region. After a list of paths has been built for all bonus regions, the best path is returned. Details about GH are reported in [6].

Algorithm 1: Greedy Heuristic.

```
1 begin
2     EmergencyRoute listRoutes ← ∅;
3     foreach region in map.setBonusRegions do
4         EmergencyRoute route;
5         initialize(route, region);
6         evaluate(route, map);
7         listRoutes.add(route);
8     EmergencyRoute bestRoute ← getBestRoute(listRoutes);
9     return bestRoute;
```

Algorithm 2 describes the genetic algorithm (GA), which is based on standard genetic algorithm approaches. A population of paths is generated and evaluated, with the evolutionary process being executed next until a convergence criterion happens. During the evolutionary process, a total of $crossRate \times numIndividuals$ individuals are generated. A tournament operator selects two individuals (parents) for reproduction, and the new individual is created by crossover and mutation operators. If the new individual is better, it replaces the worst parent. A convergence occurs when new individuals

are not inserted into the population during the evolutionary process. In this case, the population is re-initialized by saving only its best individual. Details about GA are reported in [8].

Algorithm 2: Genetic Algorithm.

```
begin
    EmergencyRoute vectorRoutes[numIndividuals];
    for i = 1 to numIndividuals do
        EmergencyRoute route;
        initialize(route, map);
        evaluate(route, map);
        vectorRoutes.add(route);
    repeat
        repeat
            for i = 1 to crossRate × numIndividuals do
                EmergencyRoute ind1, ind2;
                select(ind1, ind2);
                EmergencyRoute child ← crossover(ind1, ind2);
                mutation(child);
                evaluate(child, map);
                vectorRoutes.add(child);
        until converge(vectorRoutes);
        restart(vectorRoutes);
    until reach(stoppingCriterion);
    EmergencyRoute bestRoute ← getBestRoute(vectorRoutes);
    return bestRoute;
```

V. Results

Table I shows the parameters of GA, which are applied here based on the parameter tuning reported in [19].

TABLE I
SETTINGS USED IN THE GA METHOD.

Parameter	Value	Parameter	Value
population size	39	*crossover rate*	0.5
tournament size	3	*mutation rate*	0.7
stopping criterion	time	*elitism*	yes

Two critical situations are evaluated as already mentioned: battery and engine failure. First, the convergence of the GA method is analyzed and, next, the methods are applied over an artificial set of 30 maps. These maps were generated using the same generator introduced in [6]. Finally, computational experiments are conducted from the real world scenario introduced in Section III. The methods are evaluated in two different hardware platforms. The first is a dual core Intel Edison computing platform with 500 MHz, 1 GB RAM and Linux - Yocto operating system. The second is a regular Personal Computer (PC) running under an Intel Core i5 processor with 1.8 GHz, 4 GB RAM and Linux - Ubuntu 16.04 operating system. The comparison between two different computer architectures is relevant since the results obtained by Intel i5 computer can be seen as an upper bound for the quality of the embedded computer solution. The experiments are designed and calibrated based on a fixed wing UAV named Ararinha as shown by Table II. The GA strategies are executed 10 times over each map, while GH is executed only one since it is a deterministic heuristic.

TABLE II
TECHNICAL SPECIFICATIONS OF THE ARARINHA.

Component	Value	Component	Value
Wingspan	1.90m	Weight	2.83kg
Length	1.15m	Payload	0.60kg
Electric Power	740W	Endurance	15 minutes

A. Convergence of the GA

Figure 6 shows the results executing the GA to find a path re-planning when a battery failure happens. This experiment was executed using Intel i5 personal computer. The battery overheating was chosen since it leads the UAV to land as soon as possible. The average, minimum and maximum values of the fitness function for all maps are depicted. The aim is to determine the maximum time ($time_{max}$) that a fitness convergence usually happens on those maps. The fitness values is improved less than 1% after 890ms, while this value is 2% between 690 and 890ms. The improvement is around 4% from 460ms to 690ms. It seems that running the GA for 690ms may be enough and extending such execution for more than 890ms may be unnecessary.

Fig. 6. Time analysis for convergence of the GA.

B. Results for artificially generated maps

The 30 maps evaluated have dimensions 1000m x 1000m and the critical situation always happens in the center of the map. Table III presents the results achieved for each approach (GH, GA, GA-GH e GA-GA). These results are the success rates, when the aircraft executes a safe landing. It is also shown the time limits (Stopping Criterion (SC)) evaluated, where GH execution is not previously setup with time limit since it is a deterministic method and fast enough. The columns Ψ_b and Ψ_m represent the success landing rate under battery and engine failure, respectively.

The results of the methods executing on Intel i5 computer are competitive based on the values reported. The UAV was able to land safely on average from 83.2% to 88.8% of the cases, except by GH that lands the aircraft 73.3% of the times. On the other hand, GH needs only 41.2 milliseconds on average to return a new path. Also, the increase in the time limit improves the overall results only 5.6% (from 83.2% to 88.8%), even when running four times longer. This makes

TABLE III
RESULTS OBTAINED AFTER EVALUATING DIFFERENT STRATEGIES IN ARTIFICIAL MAPS.

		Intel i5				Intel Edison			
Methods	SC (ms)	Ψ_b	Ψ_m	Avg Overall	Time (ms)	Ψ_b	Ψ_m	Avg Overall	Time (ms)
GH	-	86.7%	60.0%	73.3%	41	86.7%	60.0%	73.3%	347
GA	250	96.3%	72.0%	84.2%	250	72.3%	62.3%	67.3%	250
GA	500	99.0%	73.0%	86.0%	500	84.7%	65.3%	75.0%	500
GA	1000	98.3%	75.7%	87.0%	1000	91.0%	68.0%	79.5%	1000
GA-GH	250	95.3%	71.0%	83.2%	250	89.3%	62.7%	76.0%	309
GA-GH	500	100.0%	72.3%	86.2%	500	90.3%	65.7%	78.0%	510
GA-GH	1000	99.3%	76.0%	87.7%	1000	92.3%	69.0%	80.7%	1000
GA-GA	250	99.3%	72.3%	85.8%	250	81.0%	65.0%	73.0%	250
GA-GA	500	99.7%	73.3%	86.5%	500	89.7%	68.3%	79.0%	500
GA-GA	1000	99.7%	78.0%	88.8%	1000	94.7%	70.0%	82.3%	1000
Final Avg	-	97.4%	72.4%	84.9%	-	87.2%	65.6%	76.4%	-

sense once the improvements happen in the beginning of the execution as reported in Section V-A.

In the results achieved with Intel Edison, the success rate decreases and computational time increases as expected. GH keeps its quality since it is a deterministic heuristic. However, the execution time increases form 41ms in Intel i5 to 347ms in Intel Edison that means a loss of performance around 8.4 times. The success rates of the other methods change from 67.3% to 82.3% which means a difference around 15% based on the strategy as well as the time limit associated. The ensemble strategies outperform GH for the majority of instances, while they are better than GA when comparing within the same execution time limit. The strategy GA-GH returns a success landing rate of 76.0% within a short time (250ms), while GA-GA strategy has the better success rates running from 500ms to 1000ms.

The non-parametric Kruskal-Wallis test is applied to evaluate significant difference among the results reported in Table IV. First, it is done a comparison between the success rates achieved by running methods with Intel i5 and Edison computers. The results show a significant difference since p-value < 0.05. The second analysis compares if there is a significant difference among success rates reported for battery and engine failures. There is also a significant difference between methods performance when dealing with battery and engine failure.

TABLE IV
KRUSKAL-WALLIS TEST FOR ARCHITECTURE AND FAILURE

Metric	Attribute	p-value	Groups	
Architecture	Intel i5 Intel Edison	0.003	A	B
Failure	Battery failure Motor failure	< 0.0001	A	B

Table V reports the Dunn test results for multiple comparison. The worst methods are GA and GA-GA within 250ms, and GA with 500ms running on Edison, while there is no significant difference among the other strategies. Thus, based on the statistical evaluation done, we cannot guarantee a significant difference among success rates when running the majority of strategies using a companion computer as Intel Edison or a personal computer with Intel i5.

TABLE V
DUNN TEST FOR MULTIPLE COMPARISON IN ARTIFICIAL MAPS.

Architecture	Methods	SC (ms)	Groups			
Intel Edison	GA	250	D			
	GA-GA	250	D	C		
	GA	500	D	C	B	
	GH	-		C	B	A
	GA-GH	250		C	B	A
	GA	1000		C	B	A
	GA-GH	500		C	B	A
	GA-GA	500		C	B	A
	GA-GH	1000		C	B	A
	GA-GA	1000		C	B	A
Intel i5	GH	-		C	B	A
	GA	250		C	B	A
	GA-GH	250		C	B	A
	GA	500			B	A
	GA-GA	250			B	A
	GA-GH	500				A
	GA-GA	500				A
	GA	1000				A
	GA-GH	1000				A
	GA-GA	1000				A

C. Case study

Simulations are executed from the aerial imaging scenario introduced in Section III. Figure 7 shows the path to be followed by the UAV during a mission. There are several failures labeled as F1, F2, F3 and F4 that will be simulated, where IFA system detects these failures. Next, the re-planning algorithm is executed during the time limit previously defined as stopping criterion, which makes the UAV travels the critical distance d_c. In these experiments, the UAV speed during the failures is $24m/s$ and, after one second as time limit, the aircraft traveled $d_c = 24$ meters far away from the failure point. The new path is sent to the autopilot and the current mission is aborted.

The white paths in Figure 7 are path re-planned for battery failure, while the black paths represent engine failure. These trajectories are determined applying GA-GA strategy and the

TABLE VI
RESULTS OBTAINED AFTER EVALUATING DIFFERENT STRATEGIES IN THE STUDY CASE.

		Intel i5				Intel Edison			
Methods	SC (ms)	Ψ_b	Ψ_m	Avg Overall	Time (ms)	Ψ_b	Ψ_m	Avg Overall	Time (ms)
GH	-	25.0%	0.0%	12.5%	22	25.0%	0.0%	12.5%	192
GA	250	100.0%	50.0%	75.0%	250	35.0%	37.5%	36.3%	250
GA	500	100.0%	50.0%	75.0%	500	62.5%	50.0%	56.3%	500
GA	1000	100.0%	67.5%	83.8%	1000	95.0%	50.0%	72.5%	1000
GA-GH	250	100.0%	50.0%	75.0%	250	37.5%	37.5%	37.5%	234
GA-GH	500	100.0%	50.0%	75.0%	500	67.5%	50.0%	58.8%	481
GA-GH	1000	100.0%	75.0%	87.5%	1000	92.5%	50.0%	71.3%	992
GA-GA	250	100.0%	50.0%	75.0%	250	50.0%	45.0%	47.5%	250
GA-GA	500	100.0%	50.0%	75.0%	500	85.0%	50.0%	67.5%	500
GA-GA	1000	100.0%	75.0%	87.5%	1000	100.0%	50.0%	75.0%	1000
Final Avg	-	92.5%	51.8%	72.1%	-	65.0%	42.0%	53.5%	-

TABLE VII
DIFFERENT SIMULATIONS PERFORMED USING SITL TO VALIDATE THE TRAJECTORIES OBTAINED BY GA METHOD RUNNING IN THE INTEL EDISON.

Simulation	Description	Web Link	Average error (m)	Maximum error (m)
Simulation 1	Failure critical in the battery in **F1**	https://youtu.be/lPYJ6nVCBRs	2.9	9.0
Simulation 2	Failure critical in the battery in **F2**	https://youtu.be/k38GBjM5jXU	18.6	30.0
Simulation 3	Failure critical in the battery in **F3**	https://youtu.be/SQebmNOnUkg	11.5	19.0
Simulation 4	Failure critical in the battery in **F4**	https://youtu.be/E6L2kQBF-ps	8.0	18.0
Simulation 5	Full route without failure critical	https://youtu.be/DxWcQVyJtFQ	32.5	59.0

aircraft is able to land on a bonus region in almost all cases. The only failure that leads the aircraft to land on the forest is the engine failure (F4).

Fig. 7. Results of case study in a real world scenario using a GA method.

Next, a total of 10 executions are done for each method and each failure and the results are shown in Table VI. The method with the worst performance is GH that saves the aircraft only 12.5%. The other methods return better solutions mainly when executed for 1000ms. Their success rates reach values from 71.3% to 75.0%. According to these results, the method that should be embedded in the UAV is the GA-GA that reports the best success rate running on Intel i5 and Edison computers.

The Kruskal-Wallis test is applied and returns p-value $= 0.004$ from these results, so there is significant difference among methods. Table VIII has the groups defined by Dunn test for multiple comparisons. The worst methods are GH, GA, GA-GA, GA-GH executed within 250ms and there is no significant difference among the other methods.

TABLE VIII
DUNN TEST FOR MULTIPLE COMPARISON IN THE STUDY CASE.

Architecture	Methods	SC (ms)	Groups			
Intel Edison	GH	-	D			
	GA	250	D	C		
	GA-GA	250	D	C	B	
	GA-GH	250	D	C	B	
	GA	500	D	C	B	A
	GA-GH	500	D	C	B	A
	GA-GA	500	D	C	B	A
	GA-GH	1000		C	B	A
	GA	1000		C	B	A
	GA-GA	1000		C	B	A
Intel i5	GH	-	D			
	GA	250		C	B	A
	GA-GH	250		C	B	A
	GA-GA	250		C	B	A
	GA	500		C	B	A
	GA-GH	500		C	B	A
	GA-GA	500		C	B	A
	GA	1000			B	A
	GA-GH	1000				A
	GA-GA	1000				A

Table VII presents some of the simulations using Software-In-The-Loop (SITL) technique to validate the routes generated by the GA-GA, after a critical failure in the battery. SITL is combined with FlightGear simulator, where parameters of the fixed wing UAV named Ararinha are adjusted for the path re-planning algorithm. The paths are provided by the GA-GA

strategy running on Edison. The simulations show that the UAV autopilot is able to follow the emergency landing path sent by GA-GA. An analysis of the average and maximum error when executing the paths is also found in Table VII. The simulation 2 has the worst result for average (18.6m) and maximum errors (30.0m). A possible explanation for such error is that the route is planned for the aircraft Ararinha, while the SITL simulates the Rascal 110. The Rascal 110 was chosen because it is the only available UAV for SITL in the FlightGear.

VI. Conclusions

This paper evaluated different strategies to apply path re-planning algorithms for UAVs under a critical situation. A total of four strategies were evaluated, GH, GA, GA-GH and GA-GA, running on two different hardware architectures: a personal computer and a companion computer. A set of artificial scenarios was evaluated as well as a case study based on a real world scenario, considering an aerial imaging application. This case study validated the path re-planning using Software-In-The-Loop (SITL).

In both scenarios, there is a significant difference between results achieved by all methods with personal computer i5 and companion computer Edison. As expected, the personal computer leverages the strategies performance. However, the Dunn test for multiple comparison did not report a statistical significant difference among the majority of strategies running on Intel i5 or Edison, when the time limit is greater than 250ms. Also, there is a significant difference between results for battery and engine failures in the set of artificial map, where the methods performance are better for battery failure. This can be explained by the limitation brought by the engine failure since it is not possible to apply acceleration in the UAV as described by equation 7.

The number of executions from the set of artificial and real-world scenarios seems to be not enough to reach a statistical significant difference among results reported within time limit greater than 250ms. As future work, more than 10 executions will be considered to better verify if there is a statistical difference among the strategies. Also other methods, such as multi-population genetic algorithms proposed in [8] and differential evolution, will be combined using the ensemble approach.

Acknowledgment

This paper acknowledges São Paulo Research Foundation (FAPESP) for the financial support of projects 2015/23182-2 and 2014/11331-0. This work was supported by CEPID-CeMEAI (FAPESP 2013/07375-0).

References

[1] A. Arfaoui, "Unmanned aerial vehicle: Review of onboard sensors, application fields, open problems and research issues," *International Journal of Image Processing (IJIP)*, vol. 11, no. 1, p. 12, 2017.

[2] D. WR Jr, "Application of artificial intelligence techniques in uninhabited aerial vehicle flight," in *Digital Avionics Systems Conference, 2003. DASC'03. The 22nd*, vol. 2. IEEE, 2003, pp. 8–C.

[3] H. Chen, X. m. Wang, and Y. Li, "A survey of autonomous control for uav," in *2009 International Conference on Artificial Intelligence and Computational Intelligence*, vol. 2, Nov 2009, pp. 267–271.

[4] L. Blackmore, M. Ono, and B. C. Williams, "Chance-constrained optimal path planning with obstacles," *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1080–1094, 2011.

[5] M. Ono, B. C. Williams, and L. Blackmore, "Probabilistic planning for continuous dynamic systems under bounded risk," *J. Artif. Int. Res.*, vol. 46, no. 1, pp. 511–577, Jan. 2013.

[6] J. S. Arantes, M. S. Arantes, C. F. M. Toledo, and B. C. Williams, "A multi-population genetic algorithm for uav path re-planning under critical situation," in *Tools with Artificial Intelligence, IEEE 27th*, Nov 2015, pp. 486–493.

[7] M. S. Arantes, J. S. Arantes, C. F. M. Toledo, and B. C. Williams, "A hybrid multi-population genetic algorithm for uav path planning," in *Genetic and Evolutionary Computation Conference*, Jul 2016.

[8] J. S. Arantes, M. S. Arantes, C. F. M. Toledo, O. Trindade, and B. C. Williams, "Heuristic and genetic algorithm approaches for uav path planning under critical situation," Fev 2017.

[9] L. F. Gonzalez, G. A. Montes, E. Puig, S. Johnson, K. Mengersen, and K. J. Gaston, "Unmanned aerial vehicles (uavs) and artificial intelligence revolutionizing wildlife monitoring and conservation," *Sensors*, vol. 16, no. 1, p. 97, 2016.

[10] C. Ramirez-Atencia, V. Rodríguez-Fernández, A. Gonzalez-Pardo, and D. Camacho, "New artificial intelligence approaches for future uav ground control stations," in *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE, 2017, pp. 2775–2782.

[11] G. Strupka, A. Levchenkov, and M. Gorobetz, "Automated situation analysis as next level of unmanned aerial vehicle artificial intelligence," in *International Conference on Applied Human Factors and Ergonomics*. Springer, 2017, pp. 25–37.

[12] L. Evers, A. I. Barros, H. Monsuur, and A. Wagelmans, "Online stochastic uav mission planning with time windows and time-sensitive targets," *European Journal of Operational Research*, vol. 238, no. 1, pp. 348 – 362, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221714002288

[13] N. Meuleau, C. Plaunt, D. E. Smith, and T. B. Smith, "An emergency landing planner for damaged aircraft." in *IAAI*, K. Z. Haigh and N. Rychtyckyj, Eds. AAAI, 2009. [Online]. Available: http://dblp.uni-trier.de/db/conf/iaai/iaai2009.html#MeuleauPSS09

[14] N. Meuleau, C. Neukom, C. Plaunt, D. E. Smith, and T. Smith, "The emergency landing planner experiment," in *21st International Conference on Automated Planning and Scheduling*, 2011.

[15] S. Li, Y. Wan, S. Fu, M. Liu, and H. F. Wu, "Design and implementation of a remote uav-based mobile health monitoring system," in *SPIE Smart Structures and Materials+ Nondestructive Evaluation and Health Monitoring*. International Society for Optics and Photonics, 2017, pp. 101 690A–101 690A.

[16] H. B. Kurt and E. Altuğ, "Development of an autonomous uav platform for advanced research applications," in *Proceedings of the 2017 International Conference on Mechatronics Systems and Control Engineering*. ACM, 2017, pp. 29–32.

[17] L. C. Velasquez, J. Argueta, and K. Mazariegos, "Implementation of a low cost aerial vehicle for crop analysis in emerging countries," in *Global Humanitarian Technology Conference (GHTC), 2016*. IEEE, 2016, pp. 21–27.

[18] N. H. Motlagh, M. Bagaa, and T. Taleb, "Uav-based iot platform: A crowd surveillance use case," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 128–134, 2017.

[19] J. S. Arantes, M. S. Arantes, C. F. M. Toledo, O. T. Júnior, and B. C. Williams, "An embedded system architecture based on genetic algorithms for mission and safety planning with uav," in *Genetic and Evolutionary Computation Conference*, Jul 2017.

[20] N. Figueira, O. Trindade, A. L. P. Mattei, and L. Neris, "Mission oriented sensor arrays – an approach towards uas usability improvement in practical applications," in *5th European Conference for Aeronautics and Space Sciences (EUCASS)*, 2013.

[21] A. Mattei, E. Fonseca, N. Figueira, O. Trindade, and F. Vaz, "Uav in-flight awareness: A tool to improve safety," in *5TH European Conference for Aeronautics and Space Sciences (EUCASS)*, 2013.

APÊNDICE

D

ARQUITETURA CONJUNTA DOS SISTEMAS MOSA E IFA ORIENTADA A SERVIÇOS

Artigo completo publicado no congresso *International Council of the Aeronautical Sciences* (ICAS), em 2018. Nesse trabalho, foi implementada uma Arquitetura Orientada a Serviços (SOA), que provê serviços aos sistemas MOSA e IFA de forma a dar maior autonomia de voo à aeronave. Esse trabalho apresenta uma versão dos sistemas MOSA e IFA em que foi dado destaque a comunicação entre eles. Nesse trabalho, foi feito um conjunto de experimentos avaliando quatro falhas distintas das oito falhas implementadas no IFA. Os resultados obtidos foram avaliados através de simulações *Software-In-The-Loop* (SITL). Esse artigo é um resultado direto da presente proposta de doutorado.

SERVICE-ORIENTED ARCHITECTURE TO INTEGRATE FLIGHT SAFETY AND MISSION MANAGEMENT SUBSYSTEMS INTO UAVS

Veronica Vannini[a] , Jesimar da Silva Arantes[a] , André Pierre Mattei[b] , Nina Figueira[c] , Márcio da Silva Arantes[b] , Claudio Fabiano Motta Toledo[a] , Onofre Trindade Júnior[a] , Pierre de Saqui-Sannes[d]

[a]University of São Paulo, Av. Trabalhador São Carlense, 400 - São Carlos - SP - Brazil , [b]Senai Innovation Institute for Embedded Systems, Av. Luiz Boiteux Piazza, 1302 - Florianópolis - SC - Brazil , [c]Brazilian Army, Dept. of Science and Technology, Quartel-General do Exército - Bloco G - 3o Andar - Brasília - DF - Brazil , [d]Institut Supérieur de l'Aéronautique et de l'Espace, 10 Avenue Edouard Belin, 31400 - Toulouse - France

Keywords: *Unmanned Aerial Vehicle, Service-Oriented Architecture, Autonomous Flight, Embedded System*

Abstract

This paper presents the design and implementation of a Service-Oriented Architecture (SOA). This architecture tackle with the data exchange different elements onboard the Unmanned Aerial Vehicle (UAV) and the real-time safety-critical operation of In-Flight Awareness Augmentation System (IFA^{2}S) and non-safety critical Mission Oriented Sensors Array (MOSA), respectively to take care of flight safety and mission accomplishment. The IFA and MOSA systems implemented and evaluated using the Software-In-The-Loop (SITL) simulation technique. The case study conducted had several critical situations were assessed through the sensors, then a diagnosis was made, and finally, a decision was made as a safety measure.

1 Introduction

The recent advances in onboard systems of Unmanned Aerial Aircraft (UAV) aims both flight safety and mission management, where In-Flight Awareness Augmentation System (IFA^{2}S) [1] and Mission Oriented Sensors Array (MOSA) [2] are examples of systems focused on these aspects. In order to be integrated into an aircraft, these subsystems shall comply with many different requirements such as time constraints, share computational resources and share the data from several sensors [3]. In order to enable flexibility, improve data management efficiency and become easier future integration (either services or hardware), it is necessary to have an environment capable of dealing with different tasks, protocols, time constraints, and priorities. Service-Oriented Architecture (SOA) provides this environment using a middleware that enables information interchange between services on a loose coupling [7] and [8].

Although IFA^{2}S and MOSA have different purposes (flight safety improvement and mission accomplishment efficiency, respectively), most data used by both come from same sensors and they share most of the information. Both systems act as service providers and consume data from sensors, which are also considered as service providers. The subsystem IFA^{2}S is a safety critical, real-time system designed to semi-automatically (with human supervision) identify and avoid flight hazards and accidents. These problems come from either internal or external

causal factors. MOSA is a non-safety critical system designed to manage the mission accomplishment by decoupling the mission-oriented part of the system from the aircraft control systems (safety-critical).

The IFA^{2}S is a novel autonomous decision-maker onboard the aircraft aiming to improve flight safety. It makes the UAV more conscious (situational awareness increase) about its subsystems conditions (internal health), flight profile, intruders presence (other aircrafts), and surrounding conditions (ground and meteorological) by keeping pilots on the ground as system managers. It provides a platform that is Situational Awareness (SA), instead of relying on human pilots' perceptions. It allows the system to act as soon as it identifies a situation that potentially leads to an accident.

MOSA is an architecture created to improve mission management during the flight [2]. The integration of MOSA with a set of sensors provides information for specific applications. MOSA also manages mission accomplishment during flight and interfaces with the safety-critical part of the UAV (automatic pilot). In addition to the hardware, a MOSA system also includes the software necessary to carry out a mission, communicate with all sensors, and send/receive data to the aircraft.

The MOSA and IFA^{2}S systems have been designed in a modular way allowing that new planning and replanning algorithms may be added to the system without significant modifications [4]. Another feature of the platform is to be plug-in-play by allowing new sensors and hardware equipment to be easily coupled by changing only a few configuration files and a few system changes in some cases. Another advantage of the platform is the facility of making changes in the missions without a high cost of time. Although the MOSA and IFA^{2}S systems are general and can be used in several UAVs, they have been thought to run at low cost and low weight UAVs.

The present paper is organized as follows: Section 2 presents some related work and Section 3 defines the problem to be studied. The proposed SOA architecture is introduced in Section 4 and experimental results are reported in Section 5. The conclusion of this work follows in Section 6.

2 Related Work

A Service Oriented Architecture (SOA) was presented in [9] for UAVs with the objective of automating the control of mosquito that are vectors of diseases such as malaria and dengue. A system of surveillance, control, suppression and elimination of vectors is introduced. The authors propose a software solution, where the Auto Pilot (MedizDroids) is specialized for the problem approached. Unlike the work developed here, the work in [9] does not present an automatic route generation system for external environments, nor a safety system, nor it distinguishes the specific system from the mission (control) The control system (AP) becomes difficult to use the aircraft on other missions.

The work on [10] has developed a cloud system for the control of one or multiple drones applied to surveillance missions by tracking the GPS position of the device to be followed. The proposed architecture is mainly based on Internet-of-Drones (IoD), where the application in the ground only deals with Web services, the interface with the cloud and the human-aircraft interface. The application in the aircraft is reserved for the control of the web services, the interface with the cloud and hardware. All processing, mission control, data collection and UAV monitoring is allocated to the cloud application. Despite some similarities with the project proposed in this article, the proposal of [10] has no route planning for obstacle avoidance as it does not have an aircraft health surveillance system.

A mission-oriented architecture is presented in [11], which proposes a modular and generic system for the control of UAVs using computational vision in agriculture. The authors present a system that changes the current mission objective and implements corrections in the flight path after finding a new target. In this work, the companion computer Odroid U3+ was integrated with the AP Pixhawk and a Micro Arduino to control a quad-

copter. A mission-oriented system is also presented in [11], but no flight safety system or aircraft analysis or route planning is proposed.

3 Problem Description

One of the proposals of this work is to develop an autonomous UAV by incorporating in it an onboard computer, so it can autonomously perform missions. All mission management, safety and route is done on board the own aircraft.

Several sensors/systems placed aboard the aircraft permit the system to be capable of detecting flight-critical failures such as checking low battery level, GPS loss, AutoPilot (AP) failures. Thus, some decision-making needs to be made for aborting the current route and executing, for example, an emergency landing over a safe region, a vertical landing, a Return To Launch (RTL), or open the parachute as a last resort.

Although the platform operates autonomously, the mission designer should describe what he or she wants the aircraft to do such as informing the launch site, the mission's objective waypoints, the landing place, and where obstacles are in the scenario. All decision-making, for example, can decide how to achieve the objectives of the mission by making the deviation of obstacles. Thus, the MOSA system is able to do this autonomously. The IFA^2 system can ensure that the aircraft during the flight executes a fast emergency decision under any critical failure.

Figure 1 shows an example of the type of problem studied and how the MOSA and IFA^{2}S systems solve this problem autonomously. The mission plan to be executed and the tasks of the MOSA and IFA^{2}S systems are detailed next.

Mission Plan: The UAV should start its flight at the point P_1 (*takeoff*) and proceed to the goal points P_2, P_3, ..., P_6 and conclude it at P_7 (*landing*). This sequence of points determines the whole mission to be followed by the UAV that will take pictures at each region. The scenario has two obstacles and No-Fly Zones (NFZ), which give the boundaries of the aircraft's navigable environment.

Fig. 1 Illustrative scenario of the mission plan managed by IFA^{2}S and MOSA systems.

MOSA System: MOSA first plans the route that minimizes the aircraft's fuel consumption between waypoints P_1 and P_2, and between waypoints P_2 and P_3 and so on until P_6 and P_7. Next, the system oversees the execution of the route by autopilot (green line in Figure 1) as well as the right instant to start taking pictures over each waypoint.

IFA^{2}S System: The UAV may present a failure such as low battery. The supervisor system (IFA^{2}S) detects the failure and decides to abort the current mission. Therefore, the UAV control executed by MOSA is halted and an algorithm for path re-planning is executed. The new trajectory is now overseen by IFA^{2}S (red line in Figure 1) landing the aircraft over a safe region (blue region).

Based on the mission specified in Figure 1, we can define a set of actions that the MOSA and IFA^{2}S systems must execute to accomplish the mission. Figure 2 synthesizes through a flowchart the set of actions in which MOSA performs the calibration of the systems and performs the path planning between points P_1 to P_7. Next, the aircraft takeoff and rises until reaching cruising altitude, follows the waypoints of the mission (P_1 to P_7) and finally lands (P_7). In parallel with MOSA, the IFA^{2}S system also performs system calibration (for example, read AP parameters and get home location), and monitors the sensors for

Fig. 2 Execution flowchart of the MOSA and IFA^{2}S systems.

critical situations. If the aircraft is flying in the event of any failure, a new route is calculated, the sound of an alarm goes off, and the aircraft lands on that region aborting the current mission.

4 Service-Oriented Architecture

The design of the Real-Time Service-Oriented Architecture was accomplish by using Model Based Space System Engineering (MBSSE) with SysML as language for system definition [13], and the free software TTool [12]. The MBSSE is chosen for this project since it facilitates application of concurrent engineering in the early phases of the aerospace system life cycle, Phases 0, A, and B [14]. Phase 0 refers to mission analysis and needs identification. Phase A is a feasibility study containing possible system concepts and assess its technical and programmatic aspects. Phase B establishes a preliminary design definition by confirming the technical solutions using trade-off studies for the selected system concept.

The SOA is responsible for the integration of service providers (e.g. GPS, cameras, accelerometers, gyros and barometers) and high-level systems such as IFA^{2}S, MOSA, the autopilot service providers and data consumers. It is implemented by using a deterministic design (same input always leads to the same output) capable of respecting time constraints in order to carry out common features and data exchange.

The development of SOA by using a middleware can be seen on the architecture depicted by the SysML block diagram in Figure 3. This software architecture provides the autonomy necessary to make the platform more capable for both managing its mission (non safety-critical service) and avoiding dangerous situations (safety-critical service). The design controls services and provides are both resources and priorities necessary for task accomplishment.

The SOA middleware implementation explores similarities between services and makes it easier information access. The Resource Manager is the integration interface for sensors, IFA^{2}S, and MOSA and uses standard interface SSP/SSI (Smart Sensor Protocol/Smart Sensor Interface). MOSA, IFA^{2}S, and Reroute Planner use sensor's data for different purposes and the Resource Manager makes them available. Reroute Planner updates the mission route due to either an emergency or mission update. The Admission Controller supervises the access to both Reroute Planner and Resource Manager giving priority to IFA^{2}S in case of conflict with MOSA. IFA^{2}S is an event based service and has priority over MOSA when either sending orders or requesting data. The direct connection between IFA^{2}S and the flight control surfaces aims prioritize emergency landing in the case of autopilot failure.

The communication between the IFA^{2}S and MOSA systems occurs through the TCP/IP network protocol using sockets. The IFA^{2}S sys-

Fig. 3 SOA architecture in which the MOSA and IFA^{2}S systems were integrated into the UAV.

tem is the server and MOSA is the client in the application, indicating the order of initialization of each system. Figure 4 shows in detail how the messages are sent. Once the MOSA system started, it will inform the IFA^{2}S. Next, IFA^{2}S informs the home location of the aircraft (important parameter during the flight). Subsequently, the IFA^{2}S informs that MOSA can start the calculation of the route to follow. As soon as this calculation finishes, MOSA tells IFA^{2}S that the route execution has already started. During the mission, if the IFA^{2}S detects a fault, the MOSA is stopped and an emergency route is calculated by IFA^{2}S. Finally, if during the execution of the mission any failure occurs in MOSA, the IFA^{2}S is warned about such occurrence and initiates an emergency route.

A robust detection, diagnostic, and decision-making system were implemented on the IFA^{2}S platform. To detect some event of abnormal behavior in the system, specifics sensors or a software system must be used. A diagnostic step is initiated based on the sensing and system data. Lastly, basis in the diagnosis, an action must be done by the decision-making system. Figure 5 shows the steps involved from the sensing to the decision-making by the IFA^{2}S system. It is important to note that the platform described has not implemented some of the strategies that are to be made as future work. In the following example, based on Figure 5, it is assumed that the aircraft has a Power Module system capable of monitoring the system's current battery level. Through this sensor, low battery diagnosis becomes simple by comparing the current battery level with a threshold. In case the value obtained is smaller than the threshold, the system must take a security action. The possible actions taken by the system are to make the emergency landing, by firing an audible alarm alerting people on the ground that there has been a critical failure and that aircraft will land quickly.

In Figure 5, other sensors/systems were embedded on the platform such as GPS, Autopilot, MOSA and IFA^{2}S and Weather Forecast. Through these components, different diagnostics can also be evaluated. For instance, the location system may not have 3D fix, showing a GPS failure, and a vertical landing should be started immediately. A bad weather condition may reveal

Fig. 4 Communication system between MOSA and IFA^{2}S.

that the aircraft should abort the current mission and return to base (RTL). A set of different sensing, diagnostics, and actions have been evaluated and are described in the results section.

5 Experimental Results

A set of simulated experiments were conducted to evaluate our architecture. They were calibrated based on a quadcopter and the results are reported below. The scenario evaluated has dimensions of

Fig. 5 Detection, diagnosis, and decision making systems of IFA^{2}S.

50m x 36m. The simulations are executed using Software-In-The-Loop (SITL) technique.

The computer used in the experiments is an Intel i7 with 2.50 GHz, 16 GB RAM and Linux - Ubuntu 17.04 operating system. The route planning/replanning methods used were Hybrid Genetic Algorithm for the mission (HGA4m) and Multi-Population Genetic Algorithm for security (MPGA4s) developed in [6] and [5], respectively. The parameters of the route planning/replanning methods used are the same as reported in the literature [3]. The HGA4m ran within 4.0 seconds as time limit, and the assumed risk of violating an NFZ by the method was 1%. The MPGA4s performed within a time limit of 1.0 seconds, making use of up to 25 waypoints and the risk assumed by the method was 1%. The altitude of the mission was 3.0 meters.

In the first experiment, as the mission execution proposed in Section 3, it will evaluated the architecture created without considering the occurrence of a critical failure. In this study, five simulations were conducted, and their routes are presented in Figure 6. The red routes represent the routes calculated by the HGA4m method. It is interesting to note that between the points P_6 and P_7 two routes passed between the obstacles and three routes passed under the obstacles. Notice in the image that some routes are overlapping. The blue routes represent the trajectory traveled by the UAV (obtained through the GPS) trying to follow the planned route in red. In general, GPS devices have a precision error associated with each type of equipment and number of satellites captured. In this way, although this device gives a specific location, it must be questioned. In general, GPS devices have an error less than 1.5 meters radius. Figure 7 shows a flow tube representing a region of uncertainty with 1.5 meters radius at which the UAV can be located. Analyzing this figure, we realize that throughout the trajectory even with the GPS error, it does not violate the obstacles nor the NFZ.

The faults described in Figure 5 have the AP Status Critical/Emergency/PowerOff and 3D Fix diagnostics that, although implemented, were not evaluated since they are difficult to in-

Fig. 6 Results obtained in the case study with SITL simulation.

Fig. 7 Results obtained considering the routes covered with a GPS precision error flow tube.

clude in simulated environments. However, the Low Battery, MOSA's Failure, IFA^2 Failure and Bad Weather for Flight failures were fully evaluated by three times each.

The first critical fault studied is shown in Figure 8. The star represents the location where the critical fault occurred. In this way, we can see that the aircraft aborted the mission route and the UAV landed in the safe area (base) in the three evaluated cases doing the deviation of obstacles.

Another result was critical fault detection in the MOSA system. The decision made for this type of failure is to perform a replanning of routes during the flight and aborting the original mission since the system MOSA presented some internal fault. Figure 9 shows three fault simulations of the MOSA system at different locations on the route. In only one of the evaluated cases, the aircraft could not land in the safe area.

Similar to the previous experiment, some flaws in the IFA^{2}S system were analyzed. In this case, the decision-making should decide to open the parachute, once no route planning could be carried out since the IFA^{2}S presented problems. Figure 10 shows three simulations, where at F_1, F_2, and F_3 points a critical flaw occurred in the IFA^{2}S system. The landing of the aircraft using the parachute device occurred shortly after the critical failure detection by the system.

A final experiment involving failures is shown in Figure 11. It can be observed the trajectories crossed by the aircraft after the system IFA^{2}S be informed about the occurrence of bad meteorological conditions. In this way, the return to base (RTL) action was triggered. In our case, the height of the configured RTL was 8 meters, thus making the VANT overpass the obstacles found in the scenario that have 5 meters.

Some of the simulations previously described in the Figures 6 and 11 can be watched by videos highlighted in Table 1.

With the above study of the architecture, a study on the time of writing (recording) of waypoints in the AP was made. Table 2 presents the results of this study, in which it is noticed that the recording time increases linearly with the number of waypoints. This analysis reveals that, in general, it takes about 0.1 seconds to set/store a waypoint in the AP. The maximum amount of waypoints supported in AP as the pixhawk is 718.

The codes of the implemented MOSA and IFA^{2}S systems are available on the GitHub platform: *https://github.com/jesimar/UAV-Toolkit*.

6 Conclusions

This article describes and evaluated a low-cost service-oriented architecture for UAVs. This architecture was embedded on a quadcopter with an onboard computer in which several simulated experiments were performed. The MOSA software system was able to properly manage the

Table 1 Different simulations performed using SITL to validate the architecture.

Methods Evaluated	Description	Web Link
HGA4m	Full route without failure critical	https://youtu.be/kh_mH3KcHe4
HGA4m and MPGA4s (emergency landing)	Partial route with Low Battery	https://youtu.be/WQm3tn7gMs4
HGA4m and MPGA4s (emergency landing)	Partial route with MOSA's Failure	https://youtu.be/SYJopMU1Ehc
HGA4m and Open Parachute	Partial route with IFA^{2}S's Failure	https://youtu.be/-nawxwiTkiY
HGA4m and RTL	Partial route with Bad Weather	https://youtu.be/aYOpoKobXmk

Fig. 8 Results obtained after the IFA^{2}S detect the state of low battery and then make the emergency landing.

Fig. 9 Results obtained after a critical fault occurs in the MOSA system and then the IFA^{2}S makes the emergency landing.

Table 2 Summary of results evaluating waypoints writing time in AP.

Nº of Waypoints	PC Time	Seconds/Waypoint
5 waypoints	0.7 seconds	0.140
10 waypoints	1.1 seconds	0.110
20 waypoints	2.1 seconds	0.105
40 waypoints	3.8 seconds	0.095
80 waypoints	7.3 seconds	0.091
160 waypoints	14.9 seconds	0.093
320 waypoints	32.8 seconds	0.103
640 waypoints	62.9 seconds	0.098
718 waypoints	73.5 seconds	0.102

mission and route planning between waypoints safely, while avoiding obstacles. The IFA^2 system was able to react to failures that occurred internally/externally in the aircraft during the flight and managed to make the emergency landing. The quality of the routes obtained in SITL simulations was considered adequate since they avoided collisions with obstacles. The study of waypoint recording time in the AP reveals the importance of improving the (re)planners HGA4m and MGPA4s, since the updating time of in-flight routes increases linearly following the number of waypoints.

Future work intends to evaluate the service-oriented architecture proposed in real flights. It is also intended to evaluate different companions computers such as Intel Edison, Raspberry Pi and ODroid XU4 in the scenario illustrated here. It is also intended to run the IFA^{2}S system on a Real-Time Operating System (RTOS) computer as it is a critical security application.

Acknowledgment

This paper acknowledges São Paulo Research Foundation (FAPESP) for the financial support of projects 2015/23182-2 and 2014/11331-0. Research developed using computational resources of Centro de Ciências Matemáticas Aplicadas à Indústria (CeMEAI) supported by Fundação de Amparo à Pesquisa do Estado de São

Fig. 10 Results obtained after a critical failure occurs in the IFA^{2}S system and then the parachute system is triggered.

Fig. 11 Results obtained after detection of bad weather conditions on the flight and then doing the RTL.

Paulo (FAPESP) - CEPID-CeMEAI (FAPESP 2013/07375-0). This work was also supported by SENAI Innovation Institute for Embedded Systems.

References

[1] A. L. P. Mattei, A. M. d. Cunha, L. A. V. Dias, P. C. d. S. Euphrasio, O. Trindade and C. M. Toledo, *IFA2S – In-flight Awareness Augmentation Systems*. 12th International Conference on Information Technology - New Generations (ITNG), Las Vegas, 2015.

[2] N. M. Figueira, I. L. Freire, O. Trindade and E. Simões, *Mission-oriented sensor arrays and UAVs - A case study on environmental monitoring*.The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences, Toronto, Canada, 2015.

[3] J. d. S. Arantes, M. d. S. Arantes, C. F. M. Toledo, O. T. Júnior and B. C. Williams, *An embedded system architecture based on genetic algorithms for mission and safety planning with UAV*. Proceedings of the Genetic and Evolutionary Computation Conference, 2017.

[4] J. d. S. Arantes, V. Vaninni, M. d. S. Arantes, C. F. M. Toledo, *Low-Cost Hardware/Software Architecture for Autonomous Flight of UAVs*. International Conference on Intelligent Robots and Systems (IROS), 2018.

[5] J. d. S. Arantes, M. d. S. Arantes, C. F. M. Toledo, and B. C. Williams *A Multi-Population Genetic Algorithm for UAV Path Re-Planning under Critical Situation*. IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 2015.

[6] M. d. S. Arantes, J. d. S. Arantes, C. F. M. Toledo, and B. C. Williams, *A Hybrid Multi-Population Genetic Algorithm for UAV Path Planning*. Genetic and Evolutionary Computation Conference (GECCO), 2016.

[7] M. Panahi, W. Nie and K.-J. Lin, *The Design of Middleware Support for Real-Time SOA*. 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, 2011.

[8] D. Rodrigues, R. d. M. Pires, E. A. Marconato, C. Areias, J. C. Cunha, K. R. C. Branco and M. Vieira, *Service-Oriented Architectures for a Flexible and Safe Use of Unmanned Aerial Vehicles*. IEEE Intelligent Transportation Systems Magazine, vol. 9.1, pp. 97-109, 2017.

[9] J. T. Amenyo, D. Phelps, O. Oladipo, F. Sewovoe-Ekuoe, S. Jadoonanan, S. Jadoonanan, T. Tabassum, S. Gnabode, T. D Sherpa, M. Falzone, A. Hossain, and A. Kublal, *MedizDroids Project: Ultra-Low Cost, Low-Altitude, Affordable and Sustainable UAV Multicopter Drones For Mosquito Vector Control in Malaria Disease Management*. IEEE 2014 Global Humanitarian Technology Conference, 2014.

[10] A. Koubaa and B. Qureshi, *DroneTrack: Cloud-*

Based Real-Time Object Tracking Using Unmanned Aerial Vehicles Over the Internet. IEEE Access, vol. 6, pp. 13810-13824, 2018.

[11] B. H. Y. Alsalam, K. Morton, D. Campbell and F. Gonzalez, *Autonomous UAV with vision based on-board decision making for remote sensing and precision agriculture*. 2017 IEEE Aerospace Conference, Big Sky, MT, pp. 1-12, 2017.

[12] L. Apvrille, *TTool*. Telecom ParisTech, Campus SophiaTech, 2016. [Online]. Available: http://ttool.telecom-paristech.fr/. [Accessed 07 03 2018].

[13] S. Mazzini, E. Tronci, C. Paccagnini and X. Olive, *A Model-Based methodology to support the Space System Engineering (MBSSE)*. System Design & Simulation Workshop, Rome, 2010.

[14] European Space Agency, *Space project management*. ESA Requirements and Standards Division, Noordwijk, 2009.

[15] Smith J, Jones B and Brown J. *The title of the book*. 1st edition, Publisher, 2001.

[16] Smith J, Jones B and Brown J. The title of the conference paper. *Proc Conference title*, where it took place, Vol. 1, paper number, pp 1-11, 2001.

[17] Smith J, Jones B and Brown J. The title of the journal paper. *Journal Name*, Vol. 1, No. 1, pp 1-11, 2001.

Contact Author Email Address

Claudio Toledo, e-mail: *claudio@icmc.usp.br*

Copyright Statement

The authors confirm that they, and/or their company or organization, hold copyright on all of the original material included in this paper. The authors also confirm that they have obtained permission, from the copyright holder of any third party material included in this paper, to publish it as part of their paper. The authors confirm that they give permission, or have obtained permission from the copyright holder of this paper, for the publication and distribution of this paper as part of the ICAS proceedings or as individual off-prints from the proceedings.

APÊNDICE

E

ARQUITETURA EMBARCADA DOS SISTEMAS MOSA E IFA

Artigo completo submetido no periódico *Expert Systems With Applications* (ESWA), em 2019, com *qualis* A1 em ciência da computação. Nesse trabalho, foram implementados os sistemas MOSA e IFA de maneira a aumentar a autonomia de voo visando aspectos de missão e segurança. Ambos os sistemas foram embarcados nos *companion computers* Intel Edison, Raspberry Pi e BeagleBone Black. Foram avaliados também dois pilotos automáticos: APM e Pixhawk. Esse trabalho apresenta uma versão totalmente funcional dos sistemas MOSA e IFA executando sobre uma aeronave real (iDroneAlpha). Juntamente com os voos reais um conjunto de simulações *Hardware-In-The-Loop* (HITL) foi realizado para validar a arquitetura proposta. Esse artigo é um resultado direto da presente proposta de doutorado.

Low-Cost Architecture for Autonomous Flight of UAVs Based on Mission Execution and Safety

Jesimar da Silva Arantes[a,1,*], Veronica Vannini[a,2], Márcio da Silva Arantes[b,3], Onofre Trindade Junior[a,4], Claudio Fabiano Motta Toledo[a,5]

[a] *University of São Paulo, Institute of Mathematical and Computer Sciences, São Carlos - SP, Brazil*
[b] *Innovation Institute of Embedded Systems, Florianópolis - SC, Brazil*

Abstract

The present paper proposes a low-cost hardware and software architecture for an autonomous flight of Unmanned Aerial Vehicles (UAVs). The proposed architecture improves the level of autonomy for UAVs by combining mission oriented and in-flight awareness systems. The hardware and software architectures allow to executed different missions with reduced human intervention. The architecture is hybrid since it has deliberative and reactive behaviours. We evaluated the flexibility of the proposed autonomous platform with three different onboard computers and two different autopilots. The results reported come from Hardware-In-The-Loop (HITL) simulations and real flights using a quadcopter. A total of four case studies is described, where the UAV must fly autonomously within non-convex scenarios with obstacles and no-fly zones (NFZs). The introduced systems must also deal with critical situations during the flight, which can lead to abort the mission execution. The results achieved show a robust performance of the proposed architecture.

Keywords: Unmanned Aerial Vehicle, Autonomous Flight, Mission and Safety, Embedded System, Obstacle Avoidance

*Corresponding author
Email address: `jesimar.arantes@usp.br` (Jesimar da Silva Arantes)
[1]jesimar.arantes@usp.br
[2]veronica.vannini@usp.br
[3]marcio.arantes@sc.senai.br
[4]onofre.trindade.junior@gmail.com
[5]claudio@icmc.usp.br

1. Introduction

Nowadays, the technologies related to Unmanned Aerial Vehicles (UAVs) are more robust in terms of processing power, autopilot (AP), embedded sensors and flight time. These advances have leveraged the application of such aircraft to many real-world scenarios in agriculture, transportation, logistics, surveillance, among others Arfaoui (2017).

Expert systems have been applied to develop efficient UAV systems as reported in Pérez-Ortiz et al. (2016); Ramirez-Atencia et al. (2017); Strupka et al. (2017); Rodríguez-Fernández et al. (2017); Kaljahi et al. (2019). The level of autonomy for UAVs changes from a ground control system, with a human pilot in charge, to fully autonomous flight Dufrene (2003). It can be enhanced; e.g. by embedding decision-making capabilities to copy with changes in the environment and possible aircraft faults. The chance of failure for UAVs is expected to be less than the accepted for general aviation Wright (2014); ICAO (2013).

The present paper introduces a hardware and software architectures that improve the level of autonomy for low-cost UAVs. The focus is mission execution and flight safety with two main systems working together. There is a system in charge of the mission that autonomously plans and supervises its entire execution. There is also an in-flight awareness system that monitors the aircraft sensors to mitigate failures autonomously. In this context, the in-flight awareness system can take the aircraft control, from the mission execution system, under internal or external threat situations.

The features of our system architecture are validated regarding scenarios with non-convex regions, which are defined by obstacles and No-Fly Zones (NFZ). The obstacles can be mountains, forests, or other regions with a certain altitude that can be overflown. The NFZ are populated regions, airports, military zones or any other region where the UAV is forbidden to overfly.

The UAV must always keep a safe distance from obstacles and NFZs taking into account the uncertainties. The uncertainties can come from the internal

aircraft dynamic; e.g. its precise position, or external factors such as wind turbulence or updates in the scenario. The design of the autonomous systems has to deal efficiently with non-convex scenarios and their uncertainties aiming to reduced human intervention.

In the proposed architecture, the mission and in-flight awareness systems are executed from a Companion Computer (CC), and we evaluate these systems running under three CC: Raspberry Pi 3, Intel Edison and BeagleBone Black. The CC is embedded into a quadcopter and tested using two autopilots (APs): APM and Pixhawk. The architecture is evaluated based on a set of Hardware-In-The-Loop (HITL) simulations and real flights.

The present paper is organized as follows: Section 2 describes the main related work and Section 3 states the problem approached. The architecture is explained in Section 4 and the experimental results are reported by Section 5. The conclusion follows in Section 6.

2. Related Work

There are several works in literature proposing hardware/software architectures for UAV. A cognitive architecture is proposed in Gunetti et al. (2010) for UAVs using parallel and associative memory, preference-based deliberation, belief maintenance, goal decomposition and adaption via a generalization of experience. These aspects provide a robust architecture for autonomous flights. There are three agents named as planner, performer and mission manager which allow executing actions such as target analysis, target attack, orbit in a position, among others, within the simulation software.

A predictive control system is described in Prodan et al. (2013) for UAV trajectories under external perturbations during the flight. The control system presents two main modules, where the faster one runs embedded on the UAV while the slower is executed from the ground station. The on board system controls the UAV dynamics and the system from the ground station plans the whole trajectory. Another architecture for trajectory control of UAV is pre-

sented in Ramasamy et al. (2016) and named as Laser Obstacle Warning and Avoidance System (LOWAS). It is a sense and avoid architecture running three main algorithms for trajectory prediction, collision estimation of obstacles and collision avoidance.

Mission Oriented Sensors Array (MOSA) is introduced as a system responsible for trajectory execution in Figueira et al. (2013). MOSA can dynamically adapt to the mission by choosing sensors that best fit depending on the situation. It is a system focused on mission accomplishment. The authors in Figueira et al. (2015) apply MOSA as a reference to design a system that automatically generates thematic maps.

In-Flight Awareness (IFA) system is introduced in Mattei et al. (2013) as a supervisor system aiming safety through the mitigation of aircraft problems. The main idea is to identify a threat autonomously by monitoring the aircraft from a set of sensors. The first development of IFA, named In-Flight Awareness Augmentation System, is described in Mattei et al. (2015) as part of the autopilot software within a small fixed-wing UAV. An awareness improvement in safety is reported by the authors based on a total of 100 flight simulations.

The authors in Xue et al. (2016) present a hardware and software architecture for navigation and automatic spraying on crop fields. In this case, both the flight control and spraying control systems are embedded in an unmanned helicopter, but the path planning is done from the ground station. The ground station reports the mission data to the flight control, which follows waypoints and sends a signal to trigger the spraying control system. The real-time status of the spray system is sent back to the on board computer that reports it to the ground station.

A generic system, based on computer vision, is reported in Alsalam et al. (2017) to control UAV in applications such as spraying on crop fields, pest detection, among others. In these scenarios, the current target points can change and the path corrections are executed autonomously during the flight. The level of autonomy proposed for the on board system allows to detect weed or pest location. This leads to spray locally instead of through the whole crop field. The

embedded decision-making system codes the Observation, Orientation, Decision and Action (OODA) loop, and the hardware is designed to control a quadcopter. A companion computer Odroid U3+ is integrated with Pixhawk autopilot and a micro Arduino. Robotic Operating System (ROS) packages are also used to develop nodes for navigation, camera and detection.

On board real-time decision making is also supported by the UAV architecture proposed by Boubeta-Puig et al. (2018), which integrates Complex Event Processing (CEP) with an event driven Service-Oriented Architecture (SOA). The system handles critical situations where CEP processes and evaluates a large number of events in real time. The SOA enables the system to deal with data from different sources and domains. The hardware architecture employs a DJI F-450 UAV, ArduPilot APM 2.6 autopilot, a companion computer Raspberry Pi 2 and message queue telemetry transport.

The authors in Yan et al. (2018) introduce a hardware and software architecture for navigation system focused on redundancy. A dual navigation computer is used, and fault-tolerant filtering algorithm is applied. The hardware integrates two FPGA, high-performance DSP and several subsystems such as Strapdown Inertial Navigation System (SINS) and Global Navigation Satellite System (GNSS).

The present paper introduces a generic architecture with a clear separation between mission execution and awareness systems. This is not an usual approach as reviewed in Prodan et al. (2013); Figueira et al. (2015); Ramasamy et al. (2016); Xue et al. (2016); Alsalam et al. (2017), where the proposed systems are more committed to a specific type of mission. We aim to advance by proposing an architecture for intelligent UAVs, as reported by these previous works, but with resilient systems and fault control similar to those in Gunetti et al. (2010); Mattei et al. (2015); Ramasamy et al. (2016).

Although the work in Gunetti et al. (2010) presents similarities to our study, the entire development of the authors is dedicated to a flight simulation system. The previous works in Figueira et al. (2015); Mattei et al. (2015) do not embedded MOSA and IFA on any platform as described here. The MOSA system

in Figueira et al. (2015) was developed in Matlab and Simulink, and the IFA reported in Mattei et al. (2013) was developed in Labview. The first integration of MOSA and IFA systems is presented in Arantes et al. (2017a), but focused on path planning and path replanning validated only in simulations. Thus, several features related to hardware/software architecture, besides path planning, are approached in the current work aiming at a higher level of autonomy for the UAV.

Table 1 provides a comparison between our work and those from literature. We evaluate if the level of autonomy includes a clear separation between Mission execution and Security (flight safety) systems. It is also considered if the experiments are based on simulations (virtual) or real flights, and if the core of the proposed architecture runs onboard or offboard. Finally, we tried to estimate if the proposed architecture is low-cost, based on the necessary technology resources employed.

Table 1: Comparing contributions based on some features.

Reference	Mission	Security	Environment	Onboard	Offboard	Low-cost
Gunetti et al. (2010)	Yes	No	Virtual	No	Yes	-
Prodan et al. (2013)	Yes	No	Virtual/Real	No	Yes	Yes
Mattei et al. (2015)	No	Yes	Virtual	No	Yes	-
Figueira et al. (2015)	Yes	No	Virtual	No	Yes	-
Ramasamy et al. (2016)	No	Yes	Virtual	Yes	No	No
Xue et al. (2016)	Yes	No	Real	Yes	No	Yes
Alsalam et al. (2017)	Yes	No	Real/Virtual	Yes	Yes	Yes
Yan et al. (2018)	No	Yes	Real	Yes	No	Yes
Boubeta-Puig et al. (2018)	Yes	No	Real/Virtual	Yes	Yes	Yes
Our Proposal	Yes	Yes	Real/Virtual	Yes	Yes	Yes

3. Problem Description

We approach the problem of executing missions autonomously with a UAV, where the aircraft makes decisions about the mission and ensures a safe flight by itself. Figure 1 gives an example of the type of situation to be handled.

(a) Example-I: The initial route containing the flight plan.

(b) Example-II: The route after updating in flight plan.

(c) Example-III: The route after failure detection and path replanning.

Figure 1: Illustrative scenario of the mission plan managed by IFA and MOSA systems.

Mission plan: The UAV should start its flight at the point A (*takeoff*), proceed to the target points B, C, and conclude at **D** (*landing*) point. The sequence of points determines some scenic regions to be overflown by the UAV that will take pictures of such regions in Figure 1a. The scenario is known in advance and it has three No-Fly Zones (NFZ) (airport, city and military base), one obstacle (mountain), four scenic regions (forests) and four bonus regions (runways of UAV, flat areas such as crop fields). The bonus regions represent safe places to land the aircraft, if a critical situation occurs. In carrying out this mission, let's assume a quadcopter with pixhawk (auto-pilot), raspberry pi (companion computer), a specific camera and a long-range buzzer. The flight altitude is 50 meters.

Mission execution system: The mission system plans the route, e.g. to minimize the aircraft's fuel consumption between A and B, and between B and C. Next, the system oversees the execution of the route by autopilot in Figure 1a as well as the right period to take pictures over each scenic region. However, the people in the ground station may decide to include two new scenic regions when the UAV is moving to target point D. The embedded mission system will receive the new target points D and E, and it will re-plan the route as shown by Figure 1b.

In-flight awareness system: Let's suppose now the UAV has a failure such as battery overheating. The in-flight awareness system detects such failure between target points C and D, and it decides to abort the mission. The mission execution system is aborted, and a path replanning is executed to land safely the aircraft. The new trajectory is now overseen by the in-flight awareness system, as in Figure 1c, that must land the aircraft over a flat region.

From the previous example in Figure 1, we are approaching a mission execution problem for UAVs within a static and non-convex environment. Thus, the no-fly regions define non-convex constraints that must be satisfied during the mission execution. The aircraft must overfly the scenic regions, which are stay-in regions, and it is possible to land over the flat areas, called bonus regions. There are uncertainties related to internal (UAV dynamic) and external disturbances (turbulence). The uncertainty here is addressed based on the risk allocation approach as described in Blackmore et al. (2011) for the Chance-Constrained Path Planning Problem. The route can usually be defined using some planner tool such as Mission Planner or QGroundControl, and the auto-pilot (AP) follows a previously planned route. However, as mentioned, the present paper deals with the problem of planning and replanning paths onboard. This is done by applying the planners described in Arantes et al. (2016, 2017a), which take into account the risk allocation approach for obstacle avoidance under uncertainties.

Moreover, we need to incorporate into the aircraft an onboard computer that performers the online path planning. There are several companion computers reported in literature as Intel Edison, Raspberry Pi, Odroid XU4, BeagleBone, among others. This work uses an Intel Edison, Raspberry Pi 3 and BeagleBone Black boards.

Aircraft failures or threat situations are also tackled during the autonomous flight. We named as critical situation any internal or external fault to the UAV that endangers the aircraft's airworthiness. During a critical situation, a decision-making action must be made by the proposed system. Thus, critical situations were intentionally inserted to evaluate the behavior and robustness of

the system. An engine failure, battery overheating and low battery or fuel level are examples of possible critical situations. For some of these problems, there are already simple solutions such as returning to the launch (RTL) or shooting the parachute. However, the fuel cannot be enough for RTL, or the landing with parachute can damage the aircraft. In this case, if possible, an emergency landing can be executed to minimize the chance of damage.

We can model the problem described above as follows:

Definition 1: (Autonomous Navigation Problem). Give $\langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle$, the system must find a solution $\mathcal{S}^*$, where:

- $\mathcal{M} = \langle M_{map}, M_{goal}, M_{alt} \rangle$ is the map and the mission plan to be followed by the UAV;
- $\mathcal{H} = \langle H_{ap}, H_{cc}, H_{sensor}, H_{actuator} \rangle$ is the configuration of the hardware architecture for the problem;
- $\mathcal{S} = \langle S_{mosa}, S_{ifa} \rangle$ is the configuration of the software architecture for the problem;
- $\mathcal{S}^*$ is the near-optimal or optimal solution found for the problem.

Definition 2: (Mission). $\mathcal{M} = \langle M_{map}, M_{goal}, M_{alt} \rangle$, where:

- $M_{map} = \langle \mathbf{r}, \mathbf{c} \rangle$ is the map of the mission, where:
  - $\mathbf{r} = \{r^1, r^2, \cdots\}$ represents the set of regions;
  - $\mathbf{c} = \{c^1, c^2, \cdots\}$ represents its classification, where $c \in \{bonus, nfz, obstacle, scenic\}$;
- $M_{goal} = \{w^1, w^2, \cdots\}$ is the set of target points of the mission to be reached by the UAV, where $w = (p_x, p_y)$;
- M_{alt} is the flight altitude of the UAV during the mission accomplishment.

Definition 3: (Hardware Architecture). $\mathcal{H} = \langle H_{ap}, H_{cc}, H_{sensor}, H_{actuator} \rangle$, where:

- H_{ap} is the autopilot (AP) employed during the mission. There are two AP currently supported: $H_{ap} \in \{APM, Pixhawk\}$;
- H_{cc} is the companion computer (CC) embedded within the UAV. There are three CC currently supported: $H_{cc} \in \{IntelEdison, RaspberryPi, BeagleBone\}$;
- H_{sensor} is a list of the sensors installed in the aircraft to accomplish a specific mission. The following sensors are currently supported: $H_{sensor} = [CameraRGB, Sonar, Temperature]$;
- $H_{actuator}$ is a list of the specific actuators installed in the aircraft to accomplish the mission. The following actuators are supported: $H_{actuator} = [Buzzer, Spraying, Parachute]$.

Definition 4: (Software Architecture). $\mathcal{S} = \langle S_{mosa}, S_{ifa} \rangle$, where:

- $S_{mosa} = \langle \mathcal{P}_P, \mathcal{L}_P, \mathcal{T}_{SC_P}, \Delta_P \rangle$ is the MOSA system, where:
  - $\mathcal{P}_P$ represents the path planner employed. There are three planner currently available in the system: $\mathcal{P}_P \in \{HGA4m, CCQSP4m, FixedRoute4m\}$;
  - $\mathcal{L}_P$ is the places to execute the system with $\mathcal{L}_P \in \{onboard, offboard\}$;
  - $\mathcal{T}_{SC_P}$ has the available time (in seconds) to wait for a route from the path planner;
  - Δ_P is the level of risk assumed by the path planner.
- $S_{ifa} = \langle \mathcal{P}_R, \mathcal{L}_R, \mathcal{T}_{SC_R}, \Delta_R \rangle$ is the IFA system, where:
  - $\mathcal{P}_R$ represents the path replanner employed. There are three replanner currently supported: $\mathcal{P}_R \in \{GA4s, MPGA4s, DE4s\}$;
  - $\mathcal{L}_R$ is the place to execute the system with $\mathcal{L}_R \in \{onboard, offboard\}$;
  - $\mathcal{T}_{SC_R}$ has the available time (in seconds) to wait for a route from the path replanner;

– Δ_R is the level of risk assumed by the path replanner.

For example, based on the problem described in Figure 1a, the mission map $M_{map} = \langle \mathbf{r}, \mathbf{c} \rangle$ has the following regions:

$$\mathbf{r} = \{runway_1, runway_2, flatarea_1, flatarea_2, airport, city, mountains, \\ militarybase, forest_1, forest_2, forest_3, forest_4\}$$

which are labeled as

$$\mathbf{c} = \{bonus, bonus, bonus, bonus, nfz, nfz, nfz, nfz, scenic, scenic, \\ scenic, scenic\},$$

respectively. The waypoints of the mission are $M_{goal} = \{A, B, C, D\}$ and the cruise altitude is $M_{alt} = 50.0$ meters. The following hardware equipment was used: $H_{ap} = Pixhawk$, $H_{cc} = RaspberryPi$, $H_{sensor} = [CameraRGB]$ and $H_{actuator} = [Buzzer]$. The following software configuration could be set by the user: $\mathcal{P}_P = HGA4m$, $\mathcal{P}_R = GA4s$, $\mathcal{L}_P = offboard$, $\mathcal{L}_R = onboard$, $\mathcal{T}_{SC_P} = 10$ sec, $\mathcal{T}_{SC_R} = 1$ sec and $\Delta_P = \Delta_R = 1.0\%$. The problem in Figure 1a can be summarized as follows:

$$
ExampleI \therefore \langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle = \begin{cases} \mathcal{M} = \begin{cases} M_{map} = \langle \mathbf{r}, \mathbf{c} \rangle = \begin{cases} \mathbf{r} = \{runway_1, runway_2, flatarea_1, \\ \quad flatarea_2, airport, city, militarybase, \\ \quad mountains, forest_1, forest_2, forest_3, \\ \quad forest_4\} \\ \mathbf{c} = \{bonus, bonus, bonus, bonus, nfz, nfz, \\ \quad nfz, nfz, scenic, scenic, scenic, scenic\} \end{cases} \\ M_{goal} = \{A, B, C, D\} \\ M_{alt} = 50.0 \end{cases} \\ \mathcal{H} = \begin{cases} H_{ap} = Pixhawk \\ H_{cc} = RaspberryPi \\ H_{sensor} = [CameraRGB] \\ H_{actuator} = [Buzzer] \end{cases} \\ \mathcal{S} = \begin{cases} S_{mosa} = \langle \mathcal{P}_P, \mathcal{L}_P, \mathcal{T}_{SC_P}, \Delta_P \rangle = \langle HGA4m, offboard, 10, 1.0\% \rangle \\ S_{ifa} = \langle \mathcal{P}_R, \mathcal{L}_R, \mathcal{T}_{SC_R}, \Delta_R \rangle = \langle GA4s, onboard, 1.0, 1.0\% \rangle \end{cases} \end{cases}
$$

To become this paper self-content, we describe next the stochastic mathematical formulation, introduced by Blackmore et al. (2011), for the problem solved by the Path Planner ($\mathcal{P}_P$).

$$
Minimize \quad \sum_t \mathbf{u}_t \cdot \mathbf{u}_t \tag{1}
$$

subject to:

$$
\mathbf{x}_T = \mathbf{x}_{goal} \tag{2}
$$

$$
\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t + \omega_t \qquad \forall(t) \tag{3}
$$

$$
\mathbf{x}_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_{x_0}), \quad \omega_t \sim \mathcal{N}(0, \Sigma_{\omega_t}) \qquad \forall(t) \tag{4}
$$

$$Pr\left[\bigwedge_j \bigwedge_t \mathbf{x}_t \notin \mathbb{O}_j^n\right] \geq 1 - \Delta_P \tag{5}$$

The state vector $\mathbf{x}_t$ has the vehicle's positions p and velocities v, while the control vector $\mathbf{u}_t$ has acceleration defined by expressions (6).

$$\mathbf{x}_t := [p_x \ p_y \ v_x \ v_y]^T, \quad \mathbf{u}_t := [a_x \ a_y]^T \tag{6}$$

The inputs are initial position ($\hat{x}_0$), goal position ($\hat{x}_{goal}$), dynamic control matrices A and B, time horizon ($t = 0, \cdots T$), time interval (Δt), external uncertainties (ω_t) and level of risk (Δ_P). The decision variables are UAV states ($\mathbf{x}_t$) and controls ($\mathbf{u}_t$) applied at each time step (t). The objective function (1) is the scalar product of the controls that is minimized during the mission planning. This measure may be related to fuel consumption or distance traveled. The vehicle state ($\mathbf{x}_T$) must reach the goal position ($\mathbf{x}_{goal}$) at the end of this mission by constraint (2). The vehicle state transitions are described in constraints (3), where the first state ($\mathbf{x}_0$) follows a Gaussian distribution of the initial expected state ($\hat{x}_0$) with covariance matrix (Σ_{x_0}). There is an additive Gaussian white noise (ω_t) with covariance matrix (Σ_{ω_t}) applied to each state transition. Constraint (5) stands that vehicle states ($\mathbf{x}_t$) must be out of obstacles ($\mathbb{O}_j^n$) for all time steps t. The expression $Pr[\cdot] \geq 1 - \Delta_P$ reports that the probability of the vehicle being outside of all obstacles must be greater than or equal to $1 - \Delta_P$. Thus, the probability of failure must be less than Δ_P.

A stochastic mathematical formulation, introduced by Arantes et al. (2015), for the problem solved by the Path Replanner ($\mathcal{P}_R$) is described next.

$$Minimize \ Pr\Big(\bigvee_j \mathbf{x}_T \in \mathbb{O}_j^p\Big) - Pr\Big(\bigvee_i \mathbf{x}_T \in \mathbb{O}_j^b\Big) \tag{7}$$

subject to:

$$\mathbf{x}_{t+1} = F_\Psi(\mathbf{x}_t, \mathbf{u}_t) + \omega_t \qquad \forall(t) \tag{8}$$

$$\mathbf{x}_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_{x_0}), \quad \omega_t \sim \mathcal{N}(0, \Sigma_{\omega_t}) \qquad \forall(t) \tag{9}$$

$$Pr\left[\bigwedge_j \bigwedge_t \mathbf{x}_t \notin \mathbb{O}_j^n\right] \geq 1 - \Delta_R \tag{10}$$

The state vector $\mathbf{x}_t$ has the vehicle's positions p, velocities v and angle α, while the control vector $\mathbf{u}_t$ has acceleration a and angular variation ε as defined by expressions (11).

$$\mathbf{x}_t := [p_x \ p_y \ v \ \alpha]^T, \quad \mathbf{u}_t := [a \ \varepsilon]^T \tag{11}$$

The inputs include initial position ($\hat{x}_0$) at the moment of failure, nonlinear state transition function (F_Ψ), time horizon ($t = 0, \cdots K$), time interval (Δt), external uncertainties (ω_t) and level of risk (Δ_R). The decision variables are UAV states ($\mathbf{x}_t$) and controls ($\mathbf{u}_t$) applied at each time step (t). The objective function (7) defines penalties on the chance of landing the aircraft over a penalty region ($\mathbb{O}_j^p$) and rewards for the chance of landing the UAV on bonus regions ($\mathbb{O}_j^b$). The transition function of the vehicle states is described by constraint (8), where the initial state ($\mathbf{x}_0$) follows a Gaussian distribution of the expected state ($\hat{x}_0$) with covariance matrix (Σ_{x_0}). A white Gaussian noise (ω_t) with covariance matrix (Σ_{ω_t}) is applied to each transition. The constraint (10) defines that the vehicle states should be outside the obstacles ($\mathbb{O}_j^n$) for all time step t.

4. Hardware/Software Architecture

Figure 2 gives an overview of the proposed architecture. It shows the basic components: UAV, Companion Computer and UAV-GCS (Ground Control Station). We assume a basic drone to run the proposed platform, which must have a simple autopilot model, some actuators, basic sensors and communication devices. The Companion Computer (CC) goes on board and hosts the systems that provide the desired level of autonomy: MOSA, IFA and S2DK. The UAV-GCS is responsible for sending all updates to the CC, but it shall work more

as a support system instead of a decision-making system. The next subsections detail the whole architecture.

Figure 2: Architecture overview.

4.1. UAV Hardware

The first step of this work was to build the UAV shown by Figure 3 with the following components: (a) Companion Computer (CC) Intel Edison; (b) autopilot APM v2.8; (c) engine, ESC and propeller; (d) GPS and compass module; (e) radio control receiver, telemetry; (f) battery, power module; (g) radio control.

It is possible to execute a radio controlled or pre-planned flight with the equipment described from items (b) to (g). The inclusion of item (a) becomes possible to embed the in-flight awareness and mission execution systems that provide the desired level of autonomy. One of the goals is to build a low-cost

Figure 3: Hardware of the UAV used in the experiments.

hardware architecture. The total cost of the UAV in Figure 3 is approximately US$500.00 and its weight is around 1.1kg with 0.4kg payload.

4.2. UAV Software Architecture

The UAV software architecture is designed to separate mission execution from aircraft monitoring systems. The mission system is based on the Mission Oriented Sensor Array (MOSA) from Figueira et al. (2015), while the monitoring is based on the In-Flight Awareness (IFA) system from Mattei et al. (2015). There is also the Services to Dronekit (S2DK) system which includes a set of services developed on the Dronekit API in Python. Figure 4 shows the three main systems of our architecture.

MOSA and IFA are compounded by the following common modules: Data acquisition, Communication, Path Planning and Decision-Making. Data Acquisition requests information from the sensors through the HTTP protocol. The Communication module allows MOSA and IFA to talk to each other as well as to the Ground Control Station (GCS). Path Planning has the algorithms responsible for defining trajectories. After the Path Planning or Path Replanning modules return a route, it is translated to JSON format and sent to the aircraft via POST messages. These messages are converted into MAVLink commands and sent next to the AutoPilot (AP).

Figure 4: The main systems: MOSA, IFA and S2DK.

Decision-Making executes different tasks regarding MOSA and IFA. This module is capable of starting another external module in MOSA, e.g. updating the trajectory after new target points have being defined from the GCS. In the IFA, the Decision-Making module takes decisions only regarding critical situations. For example, the Decision-Making module in IFA can trigger the parachute, the path replanning for emergency landing or the Return To Lunch (RTL) action.

MOSA has the Mission Manager module that oversees tasks related to the mission execution. Figure 5 illustrates how this module takes care of the mission. First, The system is initiated by creating some log files and reading the inputs about the mission. Next, four threads are executed. The first thread is responsible for communicating with the IFA, the second sends data to GCS, the third one supervises the execution of the mission, and the fourth performs actions such as planning a new route. After completing the mission, the system lands the aircraft and disarms the engines ending the execution flow.

Figure 5: Flow of execution for mission manager module in MOSA.

Security Manager in IFA controls all subsystems as shown by Figure 6. In this case, the log file creation is followed by the calibration of AP parameters. Next, four threads are also activated: the first thread communicates with MOSA, the second communicates with the GCS, the third sends data to GCS, and the fourth supervises the safety and operation of the aircraft. If any critical situation is detected, the IFA assumes control, and an action related to aircraft safety takes place. For example, if an emergency landing is demanded, the planer is called, the trajectory is defined, and the waypoints are sent to the AP. In this situation, the UAV will be put down by the IFA, the motors will be disarmed and the system will shut down.

Both IFA and MOSA can access the aircraft data through the S2DK system shown by Figure 7. The S2DK is an application coded in Python language from the API of dronekit. The dronekit is a library providing features that facilitate the control of the UAV[6]. The dronekit translates the desired commands into MAVLink protocol understood by the AP. The services implemented in S2DK respond HTTP requests through the GET and POST methods. This type of

[6]http://dronekit.io/

Figure 6: Flow of execution for security manager module in IFA.

communication occurs in only one direction. However, unlike the socket, the application can not start a communication but just processing and answering it. This is a relevant aspect since any application can access the sensor data of the aircraft regardless of the programming language. The access occurs through the GET method. Another important feature is the definition of actions to be followed by the UAV. The definitions of actions are done through the POST method, where the data needed for decision making by the AP is attached to the header of the message.

Through the services provided by the GET method, it is possible to access data from AP, barometer, accelerometer, GPS, compass and power module. This becomes possible to request information about latitude, longitude, altitude (relative, absolute), groundspeed, airspeed, pitch, yaw, roll, voltage, level, among others. In Figure 7, we have the data provided by S2DK and the hardware component responding the request. The GET and POST method information is delivered in JavaScript Object Notation (JSON) format, which makes it easy for the read/write process.

Another feature provided by S2DK is the definition of actions performed by

Figure 7: Services/operations available through the S2DK system.

the aircraft through the POST method. The following actions are supported by this service: set-mission, append-mission, set-waypoint, append-waypoint, set-heading, set-velocity, set-parameter and set-mode. The set-mission allows to change the whole mission, with the previous mission overwritten by a new one in the AP during flight. Another function available is the append-mission that can add a new mission at the end of the current mission also during the flight.

A robust detection, diagnostic, and decision-making system were implemented on the IFA platform based on Vannini et al. (2018). To detect some event of abnormal behavior, specific sensors or software system must be used. A diagnostic step begins on the sensing and system data, after which an action must be done by the decision-making system. Figure 8 shows the steps from the sensing to the decision-making by the IFA system. For example, if the aircraft has a Power Module system sensing the battery level, a diagnosis can be established from the battery level. A low battery diagnosis becomes simple by comparing the current level with a threshold, where the system takes a security action when the threshold is violated.

Other sensors/systems are shown in Figure 8 such as GPS, Autopilot and

Figure 8: Detection, diagnosis, and decision making systems of IFA.

Weather Forecast. Different diagnostics can also be evaluated through these components. For example, a bad weather condition may reveal that the aircraft should abort the current mission, and a RTL action is triggered. The Section 5 describes and evaluates a set of different sensing, diagnostics, and actions. It is worth mentioning that the design of the software architecture allows to replace or add different hardware equipment and software modules, resembling a plug and play system as illustrated by *Example I* in Section 3.

4.3. UAV Path (Re)Planning

Figure 4 showed the path planning module integrated into the MOSA system. In this module, the following path planning algorithms are available: HGA4m, CCQSP4m and FixedRoute4m. They will be briefly described next:

- **HGA4m**: acronym for Hybrid Genetic Algorithm (GA) for the mission. It is a GA combined with visibility graph strategy for the path planning problem between two points. HGA4m seeks to find a path with risk allocation to minimize fuel and avoid obstacles. It was proposed in Arantes et al. (2016);

- **CCQSP4m**: acronym for Chance Constraint Qualitative State Plan for the mission. It is a Mixed-Integer Linear Programming (MILP) based algorithm for mission planning with several target points. The method seeks

to find a path with risk allocation to minimize fuel and avoid obstacles. It is based on the planner described in Ono et al. (2013);

- **FixedRoute4m**: It is a simple strategy for flight execution in UAV, where a predefined route is loaded to be executed by the aircraft. This strategy resembles trajectories with waypoints returned by Mission Planner or QGroundControl tools. The main difference is that the waypoints are set on autopilot, through MOSA system, few seconds before the flight starting. The trajectory has no risk allocation or deviation of obstacles.

In the path replanning module, the following algorithms are available: GA4s, MPGA4s and DE4s. They are briefly described next:

- **GA4s**: acronym for Genetic Algorithm for security. It is a standard GA for the emergency landing problem of UAVs in case of a critical situation. This algorithm seeks to land the aircraft over a bonus region, based on the type of critical situation, by avoiding obstacles and with risk allocation. Details about this algorithm are found in Arantes et al. (2017b);

- **MPGA4s**: acronym for Multi-Population Genetic Algorithm (MPGA) for security. It was also developed to deal with emergency landing for UAVs under a critical situation, with the same constraints defined for GA4s: bonus region, avoiding obstacles and risk allocation. GA4s and MPGA4s have similar performance in terms of solution quality and execution time as reported in Arantes et al. (2017b);

- **DE4s**: acronym for Differential Evolution for security. It is a simple DE for the emergency landing problem of UAVs in case of a critical situation such as GA4s and MPGA4s. DE4s was developed by this current work and integrated in the proposed system.

The architecture proposed here has deliberative and reactive characteristics. Deliberative aspects can be identified by doing mission and path planning using HGA4m, CCQSP4m or FixedRoute4m methods where a global objective is

established. Reactive aspects are given when a fault is identified and the aircraft reacts to it. The algorithms GA4s, MPGA4s and DE4s are proposed as reactive procedures. However, RTL or vertical landing are other possible reactive actions available in our system.

4.4. UAV Communication Systems

The communication options used are illustrated in Figure 9 such as radio control, telemetry and Wi-Fi. The presence of the radio control in our proposal is only a security measure since the UAV will autonomously fly. Telemetry is employed here to send flight data to the GCS, while the Wi-Fi connection occurs between GCS and the CC embedded in the UAV. The mission designer will connect to the onboard computer through the Secure Shell (SSH) protocol. Figure 10 gives some details about the communication systems.

Figure 9: Communication systems available in the UAV used.

The Agent (user or pilot) can interact directly with the AP through the radio control. If the Agent wants to interact with the CC, she/he must use a SSH terminal. The communication with the UAV begins by sending a signal to trigger both IFA and MOSA systems. The Raspberry Pi, Intel Edison and BeagleBone Black run Linux operating system (Raspbian, Yocto and Debian,

Figure 10: Forms of communication and interaction among the modules/components of the CC, AP and solo computer.

respectively). The API PyMAVLink is a library for communication through the protocol MAVLink. The Dronekit API has a high-level command set that is translated into the MAVLink, enabling the systems to control the UAV.

5. Experimental Results

Two types of experiments were conducted, one based on simulation and other on real flights. The simulations are executed using Hardware-In-The-Loop (HITL) technique with a total of 50 experiments performed. The real flights use the quadcopter presented in Figure 3 in which eight experiments were performed. The companion computers have the following configurations:

- Raspberry Pi 3 board: quad-core ARM Cortex-A53 processor with 1.2GHz, 1GB RAM and Linux - Raspbian operating system.

- Intel Edison board: a dual-core Intel Atom processor with 500MHz, 1GB RAM and Linux - Yocto.

- BeagleBone Black board: dual-core ARM Cortex-A8 processor with 1.0GHz, 512MB RAM and Linux - Debian.

.

The computer used in GCS is a notebook Dell Inspiron with Intel i7 processor, 2.50GHz, 16GB RAM and Linux - Ubuntu. The Branch and Cut (B&C) algorithm, available in solver IBM ILOG CPLEX 12.2, is used by path planning methods HGA4m and CCQSP4m. CPLEX was installed on the GCS notebook and Intel Edison board, while Raspberry Pi and BeagleBone Black do not have CPLEX installed since their architectures are ARM.

Figure 11: Overview of the four scenarios used in architecture validation.

Figure 11 shows an overview of four different scenarios evaluated in this

paper. Case-I illustrates a simple mission in which the drone takes off, follows a predefined mission taking a set of pictures and finally returns to the initial point. There are no obstacles, but we have one bonus region besides the take off place. Case-II shows a mission with a set of artificial obstacles to be avoided, where the aircraft must start the flight at one point and land on another. Case-III has the mission with a real obstacle to be avoided and two bonus regions. Case-IV illustrates the more complex mission in which the system will need to change its behavior during the flight. In this case, the scenario has two obstacles to be avoided and two scenic regions to be reached. A set of critical situations are inserted for all cases, so the reaction behavior of the system is evaluated.

Figure 12 shows the results for Case-I. A total of 12 experiments were conducted with 10 HITL simulations and two real flights. The UAV should start the flight at point A and reach points B, C, D and so on until finalizing in K. Two CC were evaluated: Raspberry Pi 3 and Intel Edison. Figure 12a has the routes from a flight without critical failures. On the other hand, Figure 12b shows the routes after a low battery failure has been detected by the IFA system, and the UAV must land as soon as possible. The mission is aborted, and the IFA system defines an emergency landing by executing the algorithm DE4s. The stars in Figure 12b tick the points where the fault was detected. The aircraft moves a little until the new route has been calculated. Figure 12c has routes when a bad weather condition is simulated. In this case, IFA system decides to execute the RTL command following the procedures illustrate in Figure 8. In five out of six low battery failures simulated, the IFA system was able to land the UAV within a bonus region (see Figure 12b). Equation 12 gives an example of the settings $\langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle$ employed by one of the 12 experiments performed for Case-I.

(a) Result without failure. (b) Result with low battery. (c) Result with bad weather.

Figure 12: Results of routes obtained in scenario Case-I.

$$
CaseI \therefore \langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle = \begin{cases} \mathcal{M} = \begin{cases} M_{map} = \langle \mathbf{r}, \mathbf{c} \rangle & \text{see Figure 12} \\ M_{goal} = \{A, B, C, D, E, F, G, H, I, J, K\} \\ M_{alt} = 5.0 \end{cases} \\ \mathcal{H} = \begin{cases} H_{ap} = APM \\ H_{cc} = IntelEdison \\ H_{sensor} = [CameraRGB] \\ H_{actuator} = [Buzzer] \end{cases} \\ \mathcal{S} = \begin{cases} S_{mosa} = \langle \mathcal{P}_P, \mathcal{L}_P, \mathcal{T}_{SC_P}, \Delta_P \rangle = \langle FixedRoute4m, onboard, \varnothing, \varnothing \rangle \\ S_{ifa} = \langle \mathcal{P}_R, \mathcal{L}_R, \mathcal{T}_{SC_R}, \Delta_R \rangle = \langle DE4s, onboard, 1.0, 1.0\% \rangle \end{cases} \end{cases} \tag{12}
$$

Figure 13 shows the results for Case-II. In this scenario, 18 experiments were conducted in total, where 16 experiments have executed HITL simulation and two executed real flights. In this scenario the UAV should start the flight at point A and move on to points B, C, D and E. Two CC were evaluated: Raspberry Pi 3 and BeagleBone Black. Figures 13a and 13c shows routes after a flight without critical failures. In Figures 13a and 13b, the planner CCQSP4m was trigged, while HGA4m is executed in Figure 13c and 13d. Figures 13b and 13d have the routes when a fault in the AP is detected by the IFA system. In this case, the planned route is aborted and the UAV makes a vertical landing. The stars shows the points where the fault was detected. Equation 13 brings the settings $\langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle$ used in one of the 18 experiments performed for Case-II.

(a) Result without failure.

$\mathcal{P}_P = CCQSP4m$.

(b) Result AP failure.

$\mathcal{P}_P = CCQSP4m$.

(c) Result without failure.

$\mathcal{P}_P = HGA4m$.

(d) Result AP failure.

$\mathcal{P}_P = HGA4m$.

Figure 13: Results of routes obtained in scenario Case-II.

$$
CaseII \therefore \langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle = \begin{cases} \mathcal{M} = \begin{cases} M_{map} = \langle \mathbf{r}, \mathbf{c} \rangle & \text{see Figure 13} \\ M_{goal} = \{A, B, C, D, E\} \\ M_{alt} = 15.0 \end{cases} \\ \mathcal{H} = \begin{cases} H_{ap} = APM \\ H_{cc} = RaspberryPi \\ H_{sensor} = [CameraRGB] \\ H_{actuator} = [Buzzer] \end{cases} \\ \mathcal{S} = \begin{cases} S_{mosa} = \langle \mathcal{P}_P, \mathcal{L}_P, \mathcal{T}_{SC_P}, \Delta_P \rangle = \langle HGA4m, offboard, 4.0, 2.0\% \rangle \\ S_{ifa} = \langle \mathcal{P}_R, \mathcal{L}_R, \mathcal{T}_{SC_R}, \Delta_R \rangle = \langle \varnothing, onboard, \varnothing, \varnothing \rangle \end{cases} \end{cases} \tag{13}
$$

Figure 14 shows the results for Case-III. A total of 18 experiments were conducted where 16 with HITL simulations and 2 with real flights. The UAV should start the flight at point A and move on to points B and C, landing at A. Two CC were evaluated: Raspberry Pi 3 and Intel Edison. Figures 14a and 14d show routes obtained from the flight without critical failures. In Figures 14a, 14b and 14c, the planner CCQSP4m was run, while the HGA4m is applied in the results of Figures 14d, 14e and 14f. The figures 14b, 14c, 14e and 14f have routes after a low battery failure has been simulated. The planned route is aborted by IFA system and the UAV does a replaning using the MPGA4s algorithm. In 12 out of 13 detections of low battery failures, the IFA system was able to land the UAV within bonus region. Equation 14 gives the settings $\langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle$ for one of the 18 experiments of Case-III.

(a) Result without failure. $\mathcal{P}_P = CCQSP4m$.

(b) Result low battery. $\mathcal{P}_P = CCQSP4m$.

(c) Result low battery. $\mathcal{P}_P = CCQSP4m$.

(d) Result without failure. $\mathcal{P}_P = HGA4m$.

(e) Result low battery. $\mathcal{P}_P = HGA4m$.

(f) Result low battery. $\mathcal{P}_P = HGA4m$.

Figure 14: Results of routes obtained in scenario Case-III.

$$
CaseIII \therefore \langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle = \begin{cases} \mathcal{M} = \begin{cases} M_{map} = \langle \mathbf{r}, \mathbf{c} \rangle & \text{see Figure 14} \\ M_{goal} = \{A, B, C, D\} \\ M_{alt} = 20.0 \end{cases} \\ \mathcal{H} = \begin{cases} H_{ap} = Pixhawk \\ H_{cc} = IntelEdison \\ H_{sensor} = [CameraRGB] \\ H_{actuator} = [Buzzer] \end{cases} \\ \mathcal{S} = \begin{cases} S_{mosa} = \langle \mathcal{P}_P, \mathcal{L}_P, \mathcal{T}_{SC_P}, \Delta_P \rangle = \langle CCQSP4m, onboard, 30.0, 2.0\% \rangle \\ S_{ifa} = \langle \mathcal{P}_R, \mathcal{L}_R, \mathcal{T}_{SC_R}, \Delta_R \rangle = \langle MPGA4s, onboard, 1.0, 1.0\% \rangle \end{cases} \end{cases} \tag{14}
$$

Figure15 shows the results for Case-IV. In this scenario, 10 experiments were carried out where 8 executed HITL simulations and two real flights. The UAV should start the flight at point A following points B, C, D, and E. This scenario evaluates the change of behavior for the UAV during the mission. First, the UAV flies from A, avoiding the obstacle, to reach point B. However, once near point B, the MOSA system is informed from GCS that some pictures must be taken at point B. At this point, the MOSA system reacts and changes the previous behavior (go directly to point C), by planning a new route depicted as a circular shape around scenic region 1. When completing the new behavior, the aircraft heads towards point C where another behavior is sent by GCS. In this case, the UAV must execute, for example, an insecticide spraying following a specific trajectory on scenic region 2. Finally, upon arriving in D, a new route is calculated to avoid obstacles and landing at point E. Two CC were evaluated: Raspberry Pi 3 and BeagleBone Black.

Figure 15a has the routes after a flight without critical failures. Figure 15b shows the routes when a bad weather is detected by the IFA system and RTL is performed. The RTL does not plan a path with risk allocation; therefore, this allows the UAV to fly over the obstacle. During the RTL, the aircraft reaches the altitude required to return directly and safely to starting point.

Figure 15c presents routes after an autopilot failure and a vertical landing have been triggered. Finally, Figure 15d shows the routes when a failure in the MOSA system happens. In this case, IFA system takes control and re-plans the path with GA4s algorithm. It can be seen in Figure 15d that one of the routes can not find a solution that lands the UAV in a bonus region. Equation 15 has the settings $\langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle$ applied in one of the 10 experiments performed for Case-IV.

(a) Result without failure.

(b) Result bad weather.

(c) Result AP failure.

(d) Result MOSA failure.

Figure 15: Results of routes obtained in scenario Case-IV.

$$
CaseIV \therefore \langle \mathcal{M}, \mathcal{H}, \mathcal{S} \rangle = \begin{cases} \mathcal{M} = \begin{cases} M_{map} = \langle \mathbf{r}, \mathbf{c} \rangle & \text{see Figure 15} \\ M_{goal} = \{A, B, C, D, E\} \\ M_{alt} = 12.0 \end{cases} \\ \mathcal{H} = \begin{cases} H_{ap} = Pixhawk \\ H_{cc} = RaspberryPi \\ H_{sensor} = [CameraRGB] \\ H_{actuator} = [Buzzer] \end{cases} \\ \mathcal{S} = \begin{cases} S_{mosa} = \langle \mathcal{P}_P, \mathcal{L}_P, \mathcal{T}_{SC_P}, \Delta_P \rangle = \langle HGA4m, offboard, 4.0, 2.0\% \rangle \\ S_{ifa} = \langle \mathcal{P}_R, \mathcal{L}_R, \mathcal{T}_{SC_R}, \Delta_R \rangle = \langle GA4s, onboard, 1.0, 1.0\% \rangle \end{cases} \end{cases} \tag{15}
$$

The simulated or real trajectories are close to the planned one for all cases, except by Figure 14a in Case-III, where the real flight trajectory is more distant from the one planned by CCQSP4m. These routes were obtained using data from the real flights performed for each of the scenarios. The link [7] displays a video with some of the flights. Figure 16 shows a 3D view for each of those cases.

Table 2 summarizes the results achieved. It is reported some measures such as UAV distance from obstacles during simulations and real flights. This specific measure gives an idea about the level of risk incurred when executing the planned route. Another important value is the critical distance traveled by the UAV, which means the distance between the point where a failure has been detected and the UAV begins to execute a new route. This value gives an idea about the overhead between the processing time of the emergency route by IFA system and the time to write it on autopilot.

Table 3 has the processing time of the planning/replanning methods used. All replanning methods were performed onboard the aircraft in CC (online). The planning methods CCQSP4m and HGA4m were executed onboard (online) of

[7] https://youtu.be/8vswz-1E8fA

Figure 16: Trajectory followed by 3D UAV.

the Intel Edison. However, they were executed offline within the Ground Control Station (GCS), when Raspberry Pi or BeagleBone Black were set as CC. One of the evaluated data is the time limit for execution of the path (re)planning methods. Other measure in Table 3 is the processing time for finding the route and seeding it to the autopilot.

In general, the best performing board for replanners is Raspberry Pi, followed by Intel Edison and BeagleBone Black. The fastest execution is reached by the CCQSP4m when concerning planners. The path planning methods run during a variable time based on the architecture configuration applied. The risk of violating NFZs for planners is set as 2.0%. The path replanning methods run within a time limit of 1 sec, and the risk of violating NFZs is set as 1.0%.

The accomplishment of the described missions was possible using the low-cost architecture proposed for all scenarios. The change of missions using our system is easily done just switching the maps files and objective waypoints of the mission. All codes developed in this work are available through the GitHub platform at the link [8].

[8] https://github.com/jesimar/UAV-Toolkit/

Table 2: Summary of the experiments.

	Description	Case-I	Case-II	Case-III	Case-IV
Scenario	Dimensions of scenarios	125m x 90m	130m x 95m	140m x 150m	110m x 75m
	Number of obstacles	0	5	1	2
	Number of bonus region	2	2	3	2
	Number of scenic region	0	0	0	2
	Height of obstacles	-	20m	25m	15m
HITL	Number of simulations HITL	10	16	16	8
	Total route planned length	462m	215m	379m	324m
	Average time of simulation	226s	171s	240s	225s
	Minimum distance between planned route and obstacle	-	1.5m	2.7m	1.9m
	Minimum distance between UAV position and obstacle	-	1.6m	2.5m	1.8m
	Maximum deviation between UAV route and planned route	1.0m	0.8m	0.9m	0.9m
Real Flight	Number of experiments	2	2	2	2
	Total route planned length	462m	215m	379m	324m
	Average time of experiment	228s	175s	316s	267s
	Minimum distance between planned route and obstacle	-	1.6m	2.7m	1.9m
	Minimum distance between UAV position and obstacle	-	1.6m	2.6m	1.7m
	Maximum deviation between UAV route and planned route	1.6m	1.3m	1.4m	1.7m
Critical Sit.	Number of experiments with low battery	6	0	13	0
	Number of experiments with bad weather	3	0	0	3
	Number of experiments with AP failure	0	13	0	2
	Number of experiments with MOSA failure	0	0	0	2
General	Number of experiments without failure	3	5	5	3
	Cruise altitude of the aircraft (M_{alt})	5m	15m	20m	12m
	Average waypoints of the FixedRoute4m strategy	11	-	-	-
	Average waypoints of the HGA4m method	0	85	78	0
	Average waypoints of the CCQSP4m method	0	31	31	0
	Average waypoints of the adaptive MOSA strategy	-	-	-	79
	RTL altitude	10m	-	-	20m
	Average critical distance traveled in the replanning (d_c)	10.3m	-	9.2m	8.5m
	Number of times that land in the safety area	5 of 6	-	12 of 13	1 of 2
	Path replanner used	DE4s	-	MPGA4s	GA4s

6. Conclusion

This paper introduced a low-cost hardware/software architecture for UAVs. This architecture was embedded in a quadcopter with a companion computer. Several experiments using HITL simulations and real flights were reported to validate the systems on this architecture. The autonomous system proposed

Table 3: Summary of the processing time of the methods.

	$\mathcal{P}_P$	H_{cc}	$\mathcal{L}_P$	$\mathcal{T}_{SC_P}$	Average Processing Time	Δ_P
Planner	HGA4m	GCS	Offline	14.0s	15.5s	2.0%
	HGA4m	Intel Edison	Online	90.0s	99.0s	2.0%
	CCQSP4m	GCS	Offline	4.0s	5.0s	2.0%
	CCQSP4m	Intel Edison	Online	30.0s	31.0s	2.0%
	$\mathcal{P}_R$	H_{cc}	$\mathcal{L}_R$	$\mathcal{T}_{SC_R}$	Average Processing Time	Δ_R
Replanner	MPGA4s	Raspberry Pi 3	Online	1.0s	1.6s	1.0%
	MPGA4s	Intel Edison	Online	1.0s	2.6s	1.0%
	GA4s	Raspberry Pi 3	Online	1.0s	1.2s	1.0%
	GA4s	BeagleBone Black	Online	1.0s	3.4s	1.0%
	DE4s	Raspberry Pi 3	Online	1.0s	2.0s	1.0%
	DE4s	Intel Edison	Online	1.0s	2.2s	1.0%

here was able to oversee the execution of the mission and the operation of the aircraft by using two separated main systems: MOSA and IFA. In scenarios without critical situation, MOSA was able to guarantee the mission accomplishment without large deviations from the planned route. If there is a critical situation, the IFA system was also able to take control of the aircraft and to execute an emergency behaviour (emergency landing, RTL or vertical landing). In all scenarios, the path planning and path replanning had to take the risk allocation into account for obstacle avoidance. The IFA system was able to react to the failures during the flight, without incurring a risk of collision with such obstacles. The trajectories obtained in HITL simulations, as well as those from real flights are similar. The hardware employed works satisfactorily during all experiments. Thus, the current status of the proposed architecture already allows to embedded it in other UAVs. As future work, we will validate the proposed hardware/software architecture in fixed-wing UAVs. The current architecture will also be exhaustively evaluated over more complex scenarios and

it will incorporate vision-based navigation to dynamically react to unforeseen obstacles. Finally, we also intend to validate the architecture using fixed-wing aircraft with other companion computers such as Odroid XU4 and Raspberry Pi Zero.

Acknowledgments

This paper acknowledges São Paulo Research Foundation (FAPESP) for the financial support of projects 2015/23182-2 and 2014/11331-0. This work was supported by CEPID-CeMEAI (FAPESP 2013/07375-0).

References

Alsalam, B. H. Y., Morton, K., Campbell, D., & Gonzalez, F. (2017). Autonomous uav with vision based on-board decision making for remote sensing and precision agriculture. In *2017 IEEE Aerospace Conference* (pp. 1–12). doi:10.1109/AERO.2017.7943593.

Arantes, J. S., Arantes, M. S., Toledo, C. F. M., Júnior, O. T., & Williams, B. C. (2017a). An embedded system architecture based on genetic algorithms for mission and safety planning with uav. In *Genetic and Evolutionary Computation Conference*.

Arantes, J. S., Arantes, M. S., Toledo, C. F. M., Trindade, O., & Williams, B. C. (2017b). Heuristic and genetic algorithm approaches for uav path planning under critical situation. *International Journal on Artificial Intelligence Tools, IEEE 27th*, .

Arantes, J. S., Arantes, M. S., Toledo, C. F. M., & Williams, B. C. (2015). A multi-population genetic algorithm for uav path re-planning under critical situation. In *Tools with Artificial Intelligence, IEEE 27th* (pp. 486–493).

Arantes, M. S., Arantes, J. S., Toledo, C. F. M., & Williams, B. C. (2016). A hybrid multi-population genetic algorithm for uav path planning. In *Genetic and Evolutionary Computation Conference - GECCO*.

Arfaoui, A. (2017). Unmanned aerial vehicle: Review of onboard sensors, application fields, open problems and research issues. *International Journal of Image Processing (IJIP)*, *11*, 12.

Blackmore, L., Ono, M., & Williams, B. C. (2011). Chance-constrained optimal path planning with obstacles. *IEEE Transactions on Robotics*, *27*, 1080–1094.

Boubeta-Puig, J., Moguel, E., Sánchez-Figueroa, F., Hernández, J., & Preciado, J. C. (2018). An autonomous uav architecture for remote sensing and intelligent decision-making. *IEEE Internet Computing*, *22*, 6–15.

Dufrene, W. R. (2003). Application of artificial intelligence techniques in uninhabited aerial vehicle flight. In *Digital Avionics Systems Conference, 2003. DASC'03. The 22nd* (pp. 8–C). IEEE volume 2.

Figueira, N., Freire, I., Trindade, O., & Simões, E. (2015). Mission-oriented sensor arrays and uavs - a case study on environmental monitoring. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, (pp. 305–312). URL: `http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-1-W4/305/2015/`. doi:`10.5194/isprsarchives-XL-1-W4-305-2015`.

Figueira, N., Trindade, O., Mattei, A. L. P., & Neris, L. (2013). Mission oriented sensor arrays – an approach towards uas usability improvement in practical applications. In *5th European Conference for Aeronautics and Space Sciences (EUCASS)*.

Gunetti, P., Dodd, T., & Thompson, H. (2010). A software architecture for autonomous uav mission management and control. *AIAA Infotech@Aerospace 2010*, .

ICAO, I. C. A. O. (2013). *Doc 9859: Safety Management Manual (SMM)*. University Street, Montréal, Quebec, Canada (edition 3 ed.). 251 p.

Kaljahi, M. A., Shivakumara, P., Idris, M. Y. I., Anisi, M. H., Lu, T., Blumenstein, M., & Noor, N. M. (2019). An automatic zone detection

system for safe landing of uavs. *Expert Systems with Applications*, *122*, 319 – 333. URL: http://www.sciencedirect.com/science/article/pii/S0957417419300272. doi:https://doi.org/10.1016/j.eswa.2019.01.024.

Mattei, A., Fonseca, E., Figueira, N., Trindade, O., & Vaz, F. (2013). Uav in-flight awareness: A tool to improve safety. In *5TH European Conference for Aeronautics and Space Sciences (EUCASS)*. Munich.

Mattei, A. L. P., Cunha, A. M., Dias, L. A. V., Euphrasio, P. C. S., Trindade, O., & Toledo, C. M. (2015). IFA2S – in-flight awareness augmentation systems. In *Information Technology - New Generations (ITNG), 2015 12th International Conference on* (pp. 95–100). doi:10.1109/ITNG.2015.21.

Ono, M., Williams, B. C., & Blackmore, L. (2013). Probabilistic planning for continuous dynamic systems under bounded risk. *J. Artif. Int. Res.*, *46*, 511–577.

Prodan, I., Olaru, S., Bencatel, R., de Sousa, J. B., Stoica, C., & Niculescu, S.-I. (2013). Receding horizon flight control for trajectory tracking of autonomous aerial vehicles. *Control Engineering Practice*, *21*, 1334 – 1349. URL: http://www.sciencedirect.com/science/article/pii/S0967066113001020. doi:http://dx.doi.org/10.1016/j.conengprac.2013.05.010.

Pérez-Ortiz, M., Peña, J. M., Gutiérrez, P. A., Torres-Sánchez, J., Hervás-Martínez, C., & López-Granados, F. (2016). Selecting patterns and features for between- and within- crop-row weed mapping using uav-imagery. *Expert Systems with Applications*, *47*, 85 – 94. URL: http://www.sciencedirect.com/science/article/pii/S0957417415007472. doi:https://doi.org/10.1016/j.eswa.2015.10.043.

Ramasamy, S., Sabatini, R., Gardi, A., & Liu, J. (2016). LIDAR obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid. *Aerospace Science and Technology*, *55*, 344 – 358. URL: http:

//www.sciencedirect.com/science/article/pii/S1270963816301900. doi:http://dx.doi.org/10.1016/j.ast.2016.05.020.

Ramirez-Atencia, C., Rodríguez-Fernández, V., Gonzalez-Pardo, A., & Camacho, D. (2017). New artificial intelligence approaches for future uav ground control stations. In *Evolutionary Computation (CEC), 2017 IEEE Congress on* (pp. 2775–2782). IEEE.

Rodríguez-Fernández, V., Menéndez, H. D., & Camacho, D. (2017). Analysing temporal performance profiles of uav operators using time series clustering. *Expert Systems with Applications*, *70*, 103 – 118. URL: http://www.sciencedirect.com/science/article/pii/S0957417416305851. doi:https://doi.org/10.1016/j.eswa.2016.10.044.

Strupka, G., Levchenkov, A., & Gorobetz, M. (2017). Automated situation analysis as next level of unmanned aerial vehicle artificial intelligence. In *International Conference on Applied Human Factors and Ergonomics* (pp. 25–37). Springer.

Vannini, V., Arantes, J. d. S., Mattei, A. P., Figueira, N., Arantes, M. d. S., Toledo, C. F. M., Trindade Junior, O., & Saqui-Sannes, P. d. (2018). Service-oriented architecture to integrate flight safety and mission management subsystems into uavs. In *International Council of the Aeronautical Sciences - ICAS*.

Wright, D. (2014). Drones: Regulatory challenges to an incipient industry. *Computer Law & Security Review*, *30*, 226 – 229.

Xue, X., Lan, Y., Sun, Z., Chang, C., & Hoffmann, W. C. (2016). Develop an unmanned aerial vehicle based automatic aerial spraying system. *Computers and Electronics in Agriculture*, *128*, 58 – 66. URL: http://www.sciencedirect.com/science/article/pii/S0168169916305282. doi:http://dx.doi.org/10.1016/j.compag.2016.07.022.

Yan, F., Li, S., & Chen, Q. (2018). Hardware and software design of highly reliable integrated navigation system for long-endurance uav. In *2018 Chinese Control And Decision Conference (CCDC)*. IEEE.

ICMC USP
SÃO CARLOS