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

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

ICMC USP SÃO CARLOS

Jesimar da Silva Arantes

Sistema autônomo para supervisão de missão e segurança de voo em VANTs [1]

Monografia apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, para o Exame de Qualificação, como parte dos requisitos para obtenção do título de Doutor em Ciências – Ciências de Computação e Matemática Computacional.

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Claudio Fabiano Motta Toledo

USP – São Carlos
Agosto de 2017

[1] Trabalho realizado com auxílio parcial da CAPES e da FAPESP processo número 2015/23182-2

Jesimar da Silva Arantes

Autonomous system for mission control and flight safety in UAVs [2]

Monograph submitted to the Institute of Mathematics and Computer Sciences – ICMC-USP, as part of the qualifying exam requisites of the of the Doctorate Program in Computer Science and Computational Mathematics.

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Claudio Fabiano Motta Toledo

USP – São Carlos
August 2017

[2] Trabalho realizado com auxílio parcial da CAPES e da FAPESP processo número 2015/23182-2

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) e à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro durante a realização desse trabalho. Aos professores da USP, em especial, aos professores do ICMC pelos ensinamentos transmitidos.

Ao professor Claudio Toledo pela orientação, paciência, amizade e ensinamentos, que foram de grande relevância para a realização deste trabalho e para meu crescimento profissional. Ao grupo de pesquisa pelas inúmeras reuniões e apresentações que serviram para ampliação do meu conhecimento nas mais diversas áreas da ciência. Ao professor Onofre Trindade pelas inúmeras dicas e discussões a respeito de VANTs. Ao professor Eduardo Simões pelo grande auxílio nos experimentos envolvendo VANTs. Ao Marcelo Duchêne, Fabiano Paulelli, Veronica Vannini, Bárbara Castanheira e André Missaglia pela 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, por ventura, 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 noiva 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** . 2017. 137 p. Monografia (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

O presente documento tem por objetivo apresentar a tese desenvolvida no Programa de Doutorado em Ciência da Computação e Matemática Computacional do ICMC/USP. Essa tese abordará o desenvolvimento de sistemas autônomos para supervisão de missão e segurança de voo em Veículos Aéreos Não-Tripulados (VANTs). A supervisão da missão será assegurada através da implementação de um sistema do tipo *Mission Oriented Sensor Array* (MOSA), responsável pelo adequado cumprimento da missão. A segurança de voo será garantida pelo sistema *In-Flight Awareness* (IFA), que visa monitorar o funcionamento da aeronave. A integração dos sistemas mencionados demandará um estudo aprofundado dessas arquiteturas, além do desenvolvimento e estabelecimento de toda a arquitetura de comunicação entre tais sistemas. Assegurar os aspectos de segurança e missão podem se tornar conflitantes durante o voo, pois em situações emergenciais deve-se abortar a missão. Diferentes estratégias para (re)planejamento de rotas baseadas em computação evolutiva e heurísticas foram desenvolvidas e integradas nos sistemas MOSA e IFA. Esses sistemas serão validados em quatro etapas: (i) experimentos com o simulador de voo FlightGear; (ii) simulações com *Software-In-The-Loop* (SITL); (iii) simulações com *Hardware-In-The-Loop* (HITL); (iv) voos reais. Na última etapa, os sistemas serão embarcados em dois modelos de VANTs desenvolvidos pelo grupo de pesquisa.

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** . 2017. 137 p. Monografia (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

This document aims to present the thesis developed in the Doctoral Program in Computer Science and Computational Mathematics at ICMC/USP. This thesis will address the development of autonomous systems for mission control and flight safety in Unmanned Aerial Vehicles (UAVs). The mission control will be ensured through the implementation of Mission Oriented Sensor Array (MOSA) system, which is responsible for the proper fulfillment of mission. The flight safety will be guaranteed by the In-Flight Awareness (IFA) system, which is responsible for monitoring the aircraft operation. The whole operation of the mentioned systems will require the development of algorithms able to guarantee the proposed functionality. Different strategies for path (re)planning based on evolutionary computation and heuristics were developed and integrated in the MOSA and IFA systems. These systems will be validated in four steps: (i) simulations with flight simulator FlightGear; (ii) simulations using Software-In-The-Loop (SITL); (iii) simulations using Hardware-In-The-Loop (HITL); (iv) real flights, where the systems will be embedded in two models of UAVs developed by the research group.

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

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG	Algoritmo Genético (*Genetic Algorithm*)
AGMP	Algoritmo Genético Multi-Populacional (*Multi-Population Genetic Algorithm*)
ANAC	Agência Nacional de Aviação Civil
AP	Piloto Automático (*AutoPilot*)
APM	ArduPilot Mega
ARP	Aeronave Remotamente Pilotada (*Remote Piloted Aircraft*)
AUV	Veículo Submarino Autônomo (*Autonomous Underwater Vehicle*)
CC	(*Companion Computer*)
CNPP	Problema de Planejamento de caminho Não-convexo com Chance-constraint (*Chance-constraint Non-convex Path-planning Problem*)
CP	Critério de Parada
FAA	Administração Federal de Aviação (*Federal Aviation Administration*)
FG	FlightGear
GCS	Estação de Controle de Solo (*Ground Control Station*)
GISA	Grupo de Interesse em Sistemas Autônomos e Aplicações
GPIO	Entrada e Saída de Propósito Geral (*General Purpose Input Output*)
GPL	Licença Pública Geral (*General Public License*)
GPS	Sistema de Posicionamento Global (*Global Positioning System*)
HG	Heurística Gulosa (*Greedy Heuristic*)
HGA4m	Algoritmo Genético Híbrido para missão *(*Hybrid Genetic Algorithm for mission)
HITL	(*Hardware-In-The-Loop*)
ICMC	Instituto de Ciências Matemáticas e Computação
IFA	Consciência em Voo (*In-Flight Awareness*)
IFA^2S	(*In-Flight Awareness Augmentation Systems*)
IMU	Unidade de Medição Inercial (*Inertial Measurement Unit*)
IoT	Internet das Coisas (*Internet of Things*)
LCR	Laboratório de Computação Reconfigurável
LIDAR	Detecção e Medição de Distância por Luz (*LIght Detection And Ranging*)
Lin.	Linux
Mac.	Mac OS
MAVLink	(*Micro Air Vehicle Link*)

MOSA	Arranjos de Sensores Orientados à Missão (*Mission Oriented Sensor Array*)
MPGA4s	Algoritmo Genético Multi-Populacional para segurança (*Multi-Population Genetic Algorithm for security*)
MSFS	Simulador de Voo MicroSoft (*MicroSoft Flight Simulator*)
N/A	Não Aplicável ou Não Disponível (*Not Applicable* ou *Not Available*)
NFZ	Zona de Exclusão Aérea (*No-Fly Zone*)
PC	Computador Pessoal (*Personal Computer*)
PLIM	Programação Linear Inteira-Mista (*Mixed-Integer Linear Programming*)
PNLIM	Programação Não Linear Inteira Mista (*Mixed-Integer NonLinear Programming*)
RC	Rádio Controlado (*Radio Control*)
ROV	Veículo Submarino Operado Remotamente (*Remotely Operated Underwater Vehicle*)
SEE	Sistemas Embarcados e Evolutivos
SisVANT	Sistema de Veículo Aéreo Não-Tripulado (*Unmanned Aerial Systems*)
SITL	(*Software-In-The-Loop*)
SO	Sistema Operacional (*Operational System*)
SSC	Departamento de Sistemas de Computação
TUGV	Veículo Terrestre Não-Tripulado Teleoperado (*Tele-Operated Unmanned Ground Vehicle*)
UGV	Veículo Terrestre Autônomo (*Unmanned Ground Vehicle*)
USP	Universidade de São Paulo
USV	Veículo de Superfície Não-Tripulado *Unmanned Surface Vehicle*
UUV	Veículo Submarino Não-Tripulados (*Unmanned Underwater Vehicle*)
VAANT	Veículo Aéreo Autônomo Não-Tripulado
VANT	Veículo Aéreo Não-Tripulado (*Unmanned Aerial Vehicle*)
VMNT	Veículo Marinho Não-Tripulado (*Unmanned Marine Vehicles*)
VNT	Veículo Não-Tripulado (*Unmanned Vehicle*)
VTNT	Veículo Terrestre Não-Tripulado (*Unmanned Ground Vehicles*)
Win.	Windows

SUMÁRIO

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 voltados à execução de missão e segurança de voo para Veículos Aéreos Não-Tripulados (VANTs). Devido a crescente utilização de VANTs, alguns questionamentos foram levantados sobre os 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) no ICMC/USP tem desenvolvido trabalhos com VANTs que remetem a preocupações com execução da missão e 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).

Todavia, até onde foi pesquisado, não há na literatura uma arquitetura capaz de integrar conceitos como MOSA, IFA e outros sistemas que garantam uma menor intervenção humana e, consequentemente, maior nível de autonomia para a aeronave. Observa-se que várias arquiteturas apresentam foco no desenvolvimento de sistemas específicos para determinado tipo de aplicação (BROWN; ESTABROOK; FRANKLIN, 2011; XUE *et al.*, 2016; MOROZOV; JANSCHEK, 2016; RAMASAMY *et al.*, 2016). Assim, esta tese pretende preencher a lacuna citada, introdu-

zindo uma arquitetura de propósito geral, com um nível de resiliência capaz de mitigar falhas e garantir um adequado nível de autonomia para a aeronave.

A motivação para o projeto 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 (AGs) 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 envolvendo AGs, Heurística Gulosa (HG) e modelos de PLIM foram desenvolvidos em Arantes (2016).

Os trabalhos 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 aborda o funcionamento conjunto de tais sistemas. Dessa forma, um sistema embarcado com as arquiteturas MOSA e IFA será desenvolvido e aplicado em um VANT de asa fixa, chamado Ararinha, e também num VANT de asa rotativas, chamado iDroneAlpha. O Ararinha é o primeiro projeto de VANT desenvolvido no SSC/ICMC/USP com instruções para a sua construção e uso disponibilizadas no site do Grupo de Interesse em Sistemas Autônomos e Aplicações (GISA) [1]. O iDroneAlpha é um quadricóptero montado durante este doutorado, visando sua integração a um computador embarcado com um piloto automático.

1.2 Objetivos

O principal objetivo deste trabalho é a proposição de uma arquitetura para VANTs e o desenvolvimento dos sistemas relacionados, que devem apresentar 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: os sistemas irão permitir que a aeronave opere com o menor nível possível de intervenção humana.

Nesse contexto, o presente projeto estabeleceu 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 serão incorporados ao MOSA possibilitando

[1] www.gisa.icmc.usp.br

tomadas de decisão em tempo real que levem 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 permitam o pouso da aeronave em caso de situação crítica ou a minimização de erros na execução da trajetória, serão incorporados, assim como, sistemas para detecção e prevenção de falhas.

3. Avaliar o comportamento da arquitetura desenvolvida em experimentos utilizando simulador de voo, *Software-In-The-Loop* (SITL), *Hardware-In-The-Loop* (HITL) e, principalmente, voos reais com VANTs de asa fixa, como o Ararinha, e de asa rotativa, como o iDroneAlpha.

1.3 Contribuições e Limitações

As contribuições esperadas com o desenvolvimento da presente tese são: obtenção de um VANT com alto grau de autonomia, com capacidade de executar algoritmos de forma rápida e com habilidade de realizar tomadas de decisão com intervenção mínima do piloto em solo. Deseja-se que o sistema desenvolvido, ao realizar o planejamento/replanejamento de rotas, minimize o consumo de combustível e possíveis danos em caso de situação crítica. Uma situação crítica abrange situações que levem o replanejamento da rota para execução de um pouso de emergência. Outra contribuição esperada é o desenvolvimento de uma plataforma embarcada para testes como: (i) algoritmos de planejamento de missão; (ii) algoritmos de replanejamento de rotas; (iii) algoritmos de função de sensores e (iv) 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 trata-se de uma tarefa complexa que demanda tempo e disponibilidade de recursos financeiros.

1.4 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:** define o cenário do problema abordado;

- **Capítulo 5:** apresenta a metodologia a ser utilizada e a arquitetura desenvolvida;
- **Capítulo 6:** mostra os resultados preliminares obtidos;
- **Capítulo 7:** mostra o cronograma a ser seguido neste trabalho;
- **Capítulo 8:** encerra esta tese apontado as conclusões e os próximos passos da pesquisa.
- **Apêndice A:** apresenta um artigo completo publicado na conferência GECCO 2017.
- **Apêndice B:** apresenta um artigo completo publicado na conferência ICTAI 2017.
- **Apêndice C:** apresenta um artigo completo publicado na revista IJAIT 2017.

CAPÍTULO

2

CONCEITOS FUNDAMENTAIS

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

Mario Quintana

2.1 Considerações Iniciais

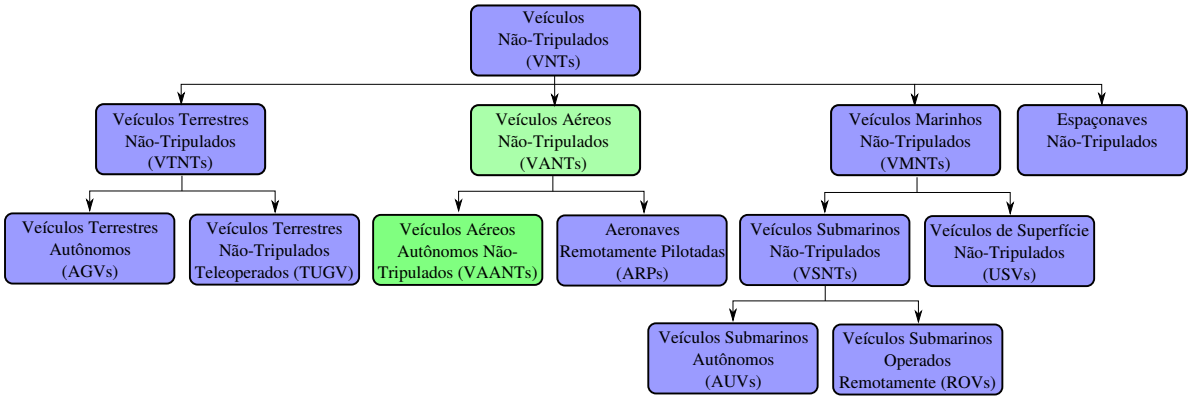
Este capítulo apresentará os conceitos fundamentais necessários para a compreensão deste trabalho como: Veículos Aéreos Não-Tripulados (VANTs), estações de controle de solo, simuladores de voo, pilotos automáticos, *companion computers*, aviônicos, simulações *software-in-the-loop* e simulações *hardware-in-the-loop*.

2.2 Veículos Aéreos Não-Tripulados

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), sendo 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, se aproximando, 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).

Essa 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 VNTs em autônomos, semi-autônomos e rádio controlados. Pensando nos VANTs, eles poderiam ser organizados baseados em: massa, dimensão, alcance, resistência, 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 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.

Os VANTs vêm sendo utilizados em diversas áreas de aplicação como: agricultura de precisão, monitoramento/inspeção, busca e resgate, mapeamento/levantamento, segurança pública, ecologia, cinematografia, pesquisa acadêmica, entre outras.

Durante essa pesquisa, dois VANTs de asa fixa principais serão utilizados: o Ararinha e o Rascal 110. A Figura 2 mostra esses dois VANTs. O Ararinha é uma iniciativa aberta que teve

origem em um projeto no ICMC/USP e possui informações de montagem em GISA (2017). Essa aeronave será utilizada em experimentos de voos reais durante o doutorado. A aeronave Rascal 110 será utilizada nessa pesquisa dentro do simulador de voo, pois não existe uma modelagem do Ararinha no simulador de voo utilizado. As principais especificações técnicas das aeronaves Ararinha e Rascal encontram-se na Tabela 1.

Figura 2 – Veículos aéreos não-tripulados utilizados.

(a) Ararinha. (b) Rascal 110.

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

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

Componente	Ararinha	Rascal 110
Envergadura	1,90 m	2,79 m
Comprimento	1,15 m	1,92 m
Potência elétrica	740 W	1600 W
Peso	2,83 kg	4,99 kg
Payload	0,60 kg	0,91 kg
Tempo de voo	15 minutos	23 minutos

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

2.3 Estações de Controle de Solo

Uma Estação de Controle de Solo, do inglês *Ground Control Station* (GCS), é uma aplicação executada sobre um computador, *tablet* ou celular localizado em solo, que 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, acompanhar a missão pelo VANT em tempo real. A GCS pode também ser utilizada para carregar o *firmware* do piloto automático na controladora de voo, controlar o voo, visualizar os *logs*, 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, LibrePilot e MAVProxy foram avaliadas neste trabalho e estão ilustradas na Figura 3.

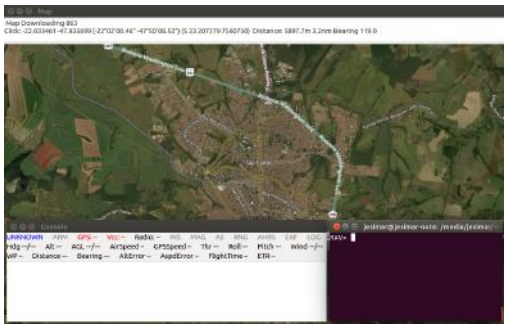
Figura 3 – Estações de controle de solo avaliadas.

(a) Mission Planner.

(b) APM Planner 2.0.

(c) QGroundControl.

(d) LibrePilot.



(e) MAVProxy.

Fonte: Elaborada pelo autor.

Essas GCS foram analisadas e as principais características levantadas são listadas nas tabelas 2 e 3. Todos os códigos fontes são abertos e estão disponíveis para *download* no GitHub sob a Licença Pública Geral, do inglês *General Public License* (GPL). Todas as estações são multiplataforma, com exceção do Mission Planner, que é exclusivo 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, o 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) (MAVLINK, 2017), possuírem uma vasta comunidade de desenvolvedores ativos (contribuidores) e muitas informações (*forks*) disponíveis 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 somente a pilotos automáticos da Pixhawk. O Mission Planner dá suporte a pilotos automáticos da 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.

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

GCS	Plataforma	Código	Licença	Linguagem	Contrib. [Forks]	Site [Code]	Veículos
Mission Planner	Win.	Aberto	GPL	C#	43 [1004]	[1] [2]	P - C - R
APM Planner 2	Win., Lin., Mac.	Aberto	GPL	C++	71 [307]	[3] [4]	P - C - R
QGroundControl	Win., Lin., Mac.	Aberto	GPL	C++	112 [802]	[5] [6]	P - C - R
LibrePilot	Win., Lin., Mac.	Aberto	GPL	C++	65 [100]	[7] [8]	P - C - R
MAVProxy	Win., Lin., Mac.	Aberto	GPL	Python	44 [253]	[9] [10]	P

Fonte: Elaborada pelo autor.

Tabela 3 – Comparações entre as estações de controle de solo avaliadas.

GCS	Lançamento	Última Versão	Preço	*Firmwares* Suportados	Hardwares Suportados
Mission Planner	2010	2017	Gratuito	PX4 e Ardupilot	Pixhalk e APM
APM Planner 2	2013	2017	Gratuito	PX4 e Ardupilot	Pixhalk e APM
QGroundControl	2013	2017	Gratuito	PX4 e Ardupilot	Pixhalk
LibrePilot	2015	2016	Gratuito	N/A	Revolution, Atom, Sparky2
MAVProxy	2012	2017	Gratuito	Ardupilot	APM

Fonte: Elaborada pelo autor.

Esta tese utilizará o Mission Planner, principalmente, para fazer os experimentos em voos 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 bem implementadas e serem fáceis de utilizar. Além disso, elas foram projetadas para o ambiente Linux, plataforma utilizada pela maioria das ferramentas utilizadas nesta tese.

[1] http://ardupilot.org/planner
[2] https://github.com/ArduPilot/MissionPlanner
[3] http://ardupilot.org/planner2
[4] https://github.com/ArduPilot/apm_planner
[5] http://qgroundcontrol.com
[6] https://github.com/mavlink/qgroundcontrol
[7] https://www.librepilot.org
[8] https://github.com/librepilot/LibrePilot
[9] http://ardupilot.github.io/MAVProxy
[10] https://github.com/ArduPilot/MAVProxy

2.4 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, tendo uma grande quantidade de veículos modelados. Três simuladores principais foram avaliados: FlightGear (FG), X-Plane e Microsoft Flight Simulator (MSFS).

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). 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 4 apresenta os principais tipos de veículos presentes no FG, num total de 327 aeronaves de asa fixa e 2 aeronaves Rádio Controladas (RC). A aeronave Rascal 110 é um dos dois veículos RC 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, no entanto, não é um VANT, mas sim uma aeronave tripulada. A Figura 4 (a) apresenta uma tela do simulador FG.

Tabela 4 – Características dos modelos de aeronaves disponíveis no FightGear.

Sigla	Descrição	Qtd. Disponíveis	% Modelos
A	Aeronave de Asa Fixa	327	83,6%
H	Aeronave de Asa Rotativa	32	8,1%
AG	Planador	11	2,8%
V	Veículo Terrestre	8	2,0%
F	Ficção/Fantasia/Diversão	6	1,5%
AZ	Dirigível/Balão	5	1,2%
RC	Controlado Remotamente	2	0,5%
Total	-	391	100%

Fonte: Adaptada de FlightGear (2017c).

X-Plane, entre todos os simuladores de voo, é o único simulador, credenciado pela Administração Federal de Aviação, do inglês *Federal Aviation Administration* (FAA), 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 4 (b) mostra a tela de interface do simulador X-Plane.

Figura 4 – Simuladores de voo avaliados.

(a) FlightGear.

(b) X-Plane.

(c) Microsoft Flight Simulator.

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

Microsoft Flight Simulator (MSFS) é um jogo de computador e também um simulador de voo (MICROSOFT, 2017). Na Figura 4 (c), é apresentada uma tela do simulador de voo Flight Simulator. 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).

A Tabela 5 traz um resumo de algumas características dos simuladores de voos 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 e o preço.

Tabela 5 – Comparações entre os simuladores de voo avaliados.

Simulador	Lançamento	Última Versão	Plataforma	Licença	Linguagem	Preço
FG	1997	2017	Win., Lin., Mac., FreeBSD, Solaris	GPL	C/C++	Gratuito
X-Plane	1993	2017	Win., Lin., Mac.	Comercial	C/C++	US$60,00[11]
MSFS	1982	2012	Win.	Comercial	C/C++/C#	US$16,00[12]

Fonte: Elaborada pelo autor.

Nesta tese será utilizado, principalmente, o simulador FlightGear na etapa de experimentos SITL. O X-Plane será utilizado na etapa de experimentos HITL. As simulações conduzidas no FG utilizarão o VANT Rascal 110. Os experimentos realizados no X-Plane utilizarão o VANT HILStar17f (PIXHAWK, 2017a).

2.5 Pilotos Automáticos

Um Piloto Automático, do inglês AutoPilot (AP), é um controlador de voo capaz de seguir *waypoints* especificados pelo usuário, prover estabilização autônoma e seguir comandos de telemetria (ARDUPILOT, 2017c). De forma geral, os APs contêm um conjunto de sensores internos como: acelerômetro, giroscópio, bússola (magnetômetro) e barômetro. Existem no mercado diversos APs disponíveis. Neste trabalho, quatro APs, ilustrados na Figura 5, foram avaliados. Os APs são: Ardupilot APM, Pixhawk versão 1, Pixhawk versão 2 e Naza M versão 2.

[11] Preço retirado do site http://www.x-plane.com/desktop/buy-it/
[12] Preço retirado do site https://www.amazon.com

O Ardupilot APM utiliza um processador AVR 2560 8 bits. O controlador de voo Pixhawk v1 possui um ARM Cortex de 32 bits. O Pixhawk v2 possui um ARM Cortex de 32 bits e, além de incluir o suporte a vários sistemas de redundância como GPS, alimentação, acelerômetro, giroscópio, bússola e barômetro, inclui também uma placa Intel Edison integrada. O controlador de voo Naza-M v2 é um projeto de hardware fechado fabricado pela empresa DJI.

Figura 5 – Pilotos automáticos avaliados.

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

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

A Tabela 6 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 e o preço médio. Nesta tese os APs Ardupilot APM e Pixhawk v1 serão utilizados por serem mais baratos e se adequarem bem ao presente projeto.

Tabela 6 – Comparações entre os pilotos automáticos avaliados.

Piloto Automático	Hardware	Empresa	Processador	Veículos Suportados	Preço Médio[13]
Ardupilot APM	Aberto	Ardupilot	AVR 2560 - 8 Bits	P - C - R	US$27,00
Pixhawk v1	Aberto	3DR	ARM Cortex - 32 Bits	P - C - R	US$65,00
Pixhawk v2	Aberto	3DR	ARM Cortex - 32 Bits	P - C - R	US$198,00
Naza M v2	Fechado	DJI	N/A	C	US$100,00

Fonte: Elaborada pelo autor.

2.6 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 se comunica e controla o piloto automático, usando 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: planejamento de missão, replanejamento de trajetória, retirada de fotografia, iniciar gravação de vídeo, fazer processamento de imagens a bordo, abortar voo, disparo do paraquedas, entre outros.

Atualmente, soluções usando *companion computer* estão sendo desenvolvidas e utilizadas por diversos grupos de pesquisa em todo o mundo. As principais comunidades/empresas

[13] Preços retirados do site https://www.amazon.com

envolvidas em seu desenvolvimento são ardupilot.org [14], DroneKit [15], 3DR [16], FlytBase [17]. As principais placas de processamento embarcado utilizadas pelo CC são: Intel Edison, Raspberry Pi, ODroid, BeagleBone, BeaglePilot (projeto de Piloto Automático dentro de uma BeagleBone Black), NVidia TX1 (usando a placa J120), Intel Aero, FlytPOD, FlytPOD PRO, entre outras. A Figura 6 mostra as placas Intel Edison, Raspberry Pi, ODroid e BeagleBone Black.

Figura 6 – Computadores embarcados avaliados.

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

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

A Tabela 7 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, o Sistema Operacional (SO) e o preço médio.

Este trabalho pretende utilizar os *companion computers* Intel Edison, Raspberry Pi e ODroid como forma de processamento a bordo do VANT. Essa escolha se deu basicamente pelo poder de processamento, documentação disponível e preço.

Tabela 7 – Comparações entre os *companion computers* avaliados.

CC	Ano	Processador	Núcleos	Frequência	RAM	SO	Preço [18]
Intel Edison	2014	Intel Atom - 64 bits	*Dual-Core*	500 MHz	1 GB	Yocto	US$85,00
Raspberry Pi 3	2016	ARMv8 Cortex-A53 - 64 bits	*Quad-Core*	1.2 GHz	1 GB	Raspbian	US$34,00
Odroid XU4	2015	Cortex-A15 e Cortex-A7	*Octa-Core*	2 GHz	2 GB	Ubuntu	US$69,00
BeagleBone Black	2013	ARM Cortex-A8 - 32 bits	*Single-Core*	1.0 GHz	512 MB	Debian	US$57,00

Fonte: Elaborada pelo autor.

2.7 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 (AP), *Companion*

[14] http://ardupilot.org
[15] http://dronekit.io
[16] https://3dr.com
[17] https://flytbase.com
[18] Preços médios retirados do site https://www.amazon.com

Computer (CC), receptor de rádio controle, módulo de telemetria e Sistema de Posicionamento Global, do inglês *Global Positioning System* (GPS). Os aviônicos AP e CC já foram tratados nas seções 2.5 e 2.6, dessa forma, não serão destacados aqui.

O receptor de rádio controle ilustrado na Figura 7 (a) é responsável por receber os sinais enviados pelo rádio controle e enviar os comandos para o AP, que em seguida executará a manobra do comando recebido. O rádio controle (transmissor) utilizado nesta tese é o FlySky FS-i6 com 6 canais operando a 2.4 GHz. Já o receptor de rádio controle é o FlySky FS-iA6 operando também a 2.4 GHz.

Um outro aviônico importante é o módulo de telemetria *air* ilustrado na Figura 7 (b). 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 módulo de telemetria da 3DR, operando a 433 MHz e com taxa de transmissão de dados no ar de até 250 kbps, será utilizado.

Por fim, o aviônico GPS está ilustrado na Figura 7 (c). 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 módulo GPS ublox NEO-6M com bússola será utilizado.

Figura 7 – Componentes aviônicos utilizados neste trabalho.

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

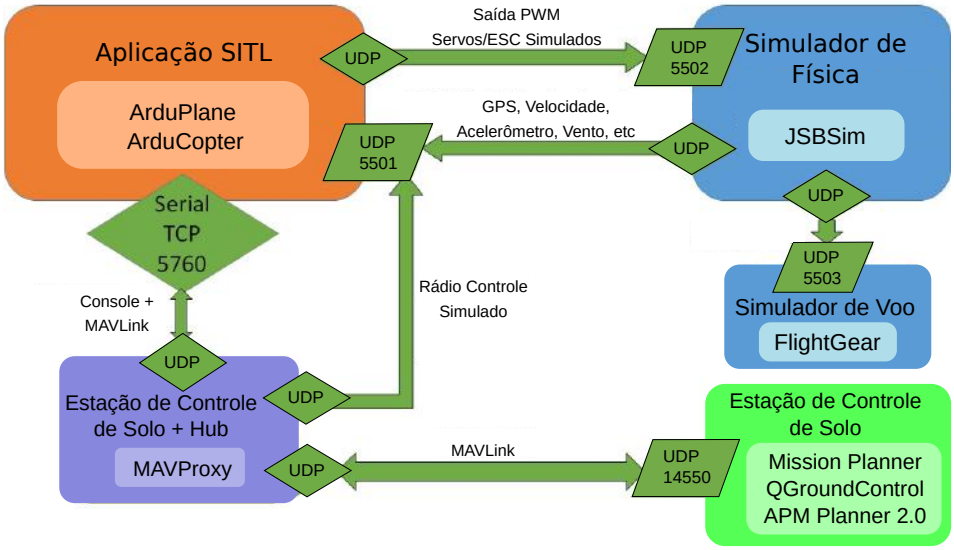
Fonte: FlySky (2017), 3DR (2017, 2017).

2.8 Simulação Software-In-The-Loop

Uma simulação *Software-In-The-Loop* (SITL) permite executar missões de VANTs sem nenhum hardware físico. Dessa forma, todo o ambiente, hardware da aeronave, hardware do piloto automático e sistema de comunicação são emulados em software (ARDUPILOT, 2017e). A simulação da aeronave 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.

O ArduPilot possui uma ampla quantidade de simuladores de veículos incorporados e pode se conectar a vários simuladores externos. Dessa forma, o SITL pode simular: aeronaves multi-rotor, aeronaves de asa fixa e veículos terrestres. A Figura 8 mostra o funcionamento interno da simulação SITL. Cinco componentes principais podem ser visualizados: software SITL, simulador de física, simulador de voo, MAVProxy e estação de controle de solo. O software SITL é o mesmo que o ArduPilot e pode executar os pilotos automáticos 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.

Figura 8 – Esquema da arquitetura de simulação SITL montada.



Fonte: Adaptada de CentMesh (2017).

Duas simulações SITL principais serão realizadas neste projeto: uma sobre o ambiente ArduPilot[19] e outra sobre o ambiente DroneKit-SITL[20].

2.9 Simulação Hardware-In-The-Loop

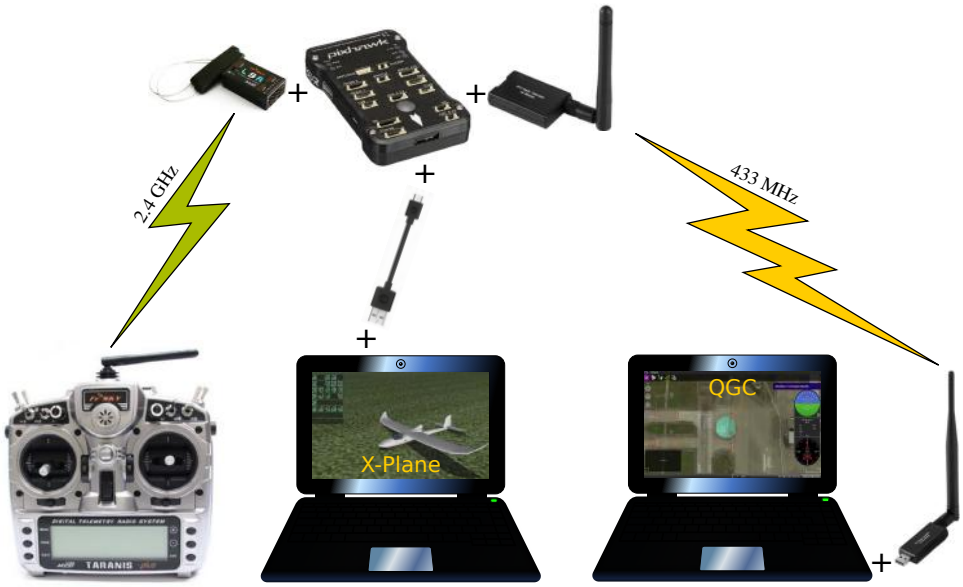
Uma simulação *Hardware-In-The-Loop* (HITL) substitui a aeronave e o ambiente por um simulador de voo. Esse simulador deve possuir um modelo de dinâmica de aeronave de alta fidelidade e modelo de ambiente (ARDUPILOT, 2017d). Os componentes piloto automático,

[19] http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html
[20] http://python.dronekit.io/develop/sitl_setup.html

telemetria, receptor de rádio controle, controle de rádio são todos hardwares físicos. Tais elementos de hardwares devem estar configurados da mesma forma que para execução de um voo real. Esse tipo de simulação, assim como SITL, auxilia no *debug* de algoritmos reduzindo o risco de uma queda do VANT em testes reais.

A Figura 9 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 de rádio controle. Ainda nessa figura, percebemos duas formas de comunicação: uma usando o rádio controle a 2.4 GHz e uma usando a telemetria a 433 MHz. Nessa figura, pode-se pensar que o primeiro computador descrito funciona como uma estação de controle de solo, em que monitora-se a missão. O segundo computador simula todo o hardware da aeronave e o ambiente, no entanto todo o sistema aviônico dessa aeronave é composto por um hardware físico.

Figura 9 – Esquema da arquitetura de simulação HITL montada.



Fonte: Elaborada pelo autor.

2.10 Considerações Finais

Este capítulo apresentou os principais conceitos envolvidos com veículos aéreos não-tripulados, estações de controle de solo, simuladores de voo, pilotos automáticos, *companion computers*, aviônicos, simulações *software-in-the-loop* e simulações *hardware-in-the-loop*. Todos esses conceitos são importantes e dão suporte para a compreensão dos capítulos seguintes. O próximo capítulo apresenta uma revisão bibliográfica dos principais trabalhos relacionados à presente tese e que darão suporte a criação da arquitetura proposta.

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 essa tese.

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 do processamento em tempo real dos dados. Ele é o responsável por controlar o percurso e os sensores do VANT durante toda a realização da missão. MOSA é utilizado em Figueira *et al.* (2015) numa aplicação envolvendo a geração automática de mapas temáticos, através de monitoramento ambiental. O 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 de câmera térmica localizada no VANT e de um 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 relacionados à segurança da aeronave conforme proposto em Mattei (2015). Esse monitoramento é feito através de um conjunto de sensores que verificam o funcionamento dos principais componentes da aeronave, visando mitigar acidentes ou, simplesmente, atualizar o plano de voo. Uma proposta de implementação do sistema IFA é descrita em Mattei *et al.* (2015),

chamada *In-Flight Awareness Augmentation Systems* (IFA^{2}S), 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, enquanto em Mattei (2015) outra simulação bastante simples, que integra 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 integrando 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. Esse é um dos pontos a ser coberto neste doutorado.

Um modelo de avaliação de arquitetura para VANTs é proposto em Renault (2015), cujo propósito é auxiliar o processo de geração de arquiteturas, reduzindo ambiguidades nos estágios iniciais do desenvolvimento de projetos de hardware/software. Esse modelo avalia, entre outros aspectos, o nível de autonomia da aeronave para tomada de decisões em voo. 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 é proposto em Prodan *et al.* (2013), baseado em controle preditivo e considerando pertubações externas ao voo. Uma arquitetura avançada para controle de trajetória, combinando hardware e software, é desenvolvida em Ramasamy *et al.* (2016), permitindo ao VANT executar manobras para evitar obstáculos e outras aeronaves em tempo real. Os autores em Morozov e Janschek (2016) introduzem uma implementação otimizada, baseada em software para detectar falhas e evitar degradação de desempenho em sistemas embarcados, incluindo seu uso em VANTs. Os trabalhos mencionados apresentam arquiteturas voltadas à uma aplicação específica e não utilizam um modelo como proposto em Renault (2015) no desenvolvimento ou avaliação das arquiteturas. Este projeto pretende contribuir com a proposição e desenvolvimento de uma arquitetura mais versátil, adotando ou adaptando metodologias, como a proposta em Renault (2015), para o seu desenvolvimento e posterior avaliação.

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. Até onde foi pesquisado pelo autor deste projeto, há poucos trabalhos enfatizando o pouso de aeronaves em situação crítica como reportado em Meuleau *et al.* (2011), Meuleau *et al.* (2009), Li (2013). Nos dois primeiros trabalhos, considera-se o pouso emergencial de aeronaves tripuladas de grande porte. Trabalhos tratando o pouso emergencial de VANTs são ainda mais escassos. Os trabalhos Li, Chen e Li (2014), Li (2013) tratam o pouso emergencial, no entanto, não são bem definidas as

falhas e não são apresentados testes rigorosos considerando tais falhas.

O trabalho de Kim *et al.* (2013) se aproxima da presente proposta ao tratar o pouso totalmente autônomo de um VANT de asa fixa, em que ele possui sensores como câmera e GPS. Esse trabalho difere do proposto neste projeto ao efetuar o pouso sobre uma rede de recuperação e não sobre regiões no solo adequadas ao pouso. Os autores em Kim *et al.* (2013) também efetuam o pouso usando visão computacional, mas não utilizam dados de GPS, como se pretende utilizar neste projeto de doutorado.

Resultados obtidos pelo autor deste doutorado, durante o mestrado, são reportados em Arantes *et al.* (2015), Arantes (2016), em que o pouso de VANTs considerando situações críticas foi tratado. Sete métodos para o planejamento da rota de pouso são comparados: uma Heurística Gulosa (HG), dois métodos baseados em Algoritmos Genéticos (AG), dois métodos baseados em Algoritmos Genéticos Multi-Populacionais (AGMP) 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. Alguns experimentos usando o simulador de voo FlightGear (FG) também foram executados. Uma das contribuições no desenvolvimento desta pesquisa é a integração de sistemas que permitam o replanejamento da 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 em voo.

Além do replanejamento da trajetória, procedimentos que permitam mitigar falhas na aeronave devem 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, mas 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, mas são capazes de detectar antecipadamente as falhas no sistema. A desvantagem está em um elevado tempo de execução do sistema que pode demandar um maior consumo de memória e, consequentemente, levar a degradações de desempenho dos algoritmos de controle. Por isso, uma implementação baseada em software com otimizações para detectar falhas, que busca evitar degradação de desempenho foi 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 método HG e AG executando em paralelo foi desenvolvido durante esse doutorado em Arantes *et al.* (2017).

O presente projeto de doutorado introduz uma arquitetura para VANTs visando preencher diversas lacunas encontradas na literatura em projetos de desenvolvimento de plataformas em software e hardware para VANTS. A ideia é desenvolver uma plataforma que permita a integração

de sistemas como MOSA (FIGUEIRA *et al.*, 2015), IFA (MATTEI, 2015), algoritmos de planejamento e replanejamento de rotas (ARANTES *et al.*, 2015; ARANTES, 2016; ARANTES *et al.*, 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).

Os conceitos de MOSA e IFA foram definidos para serem tratados em conjunto, entretanto nenhum trabalho nesse sentido foi desenvolvido até o momento. O projeto de doutorado pretende preencher essa lacuna, implementando um sistema integrado visando acompanhamento de missão e 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.

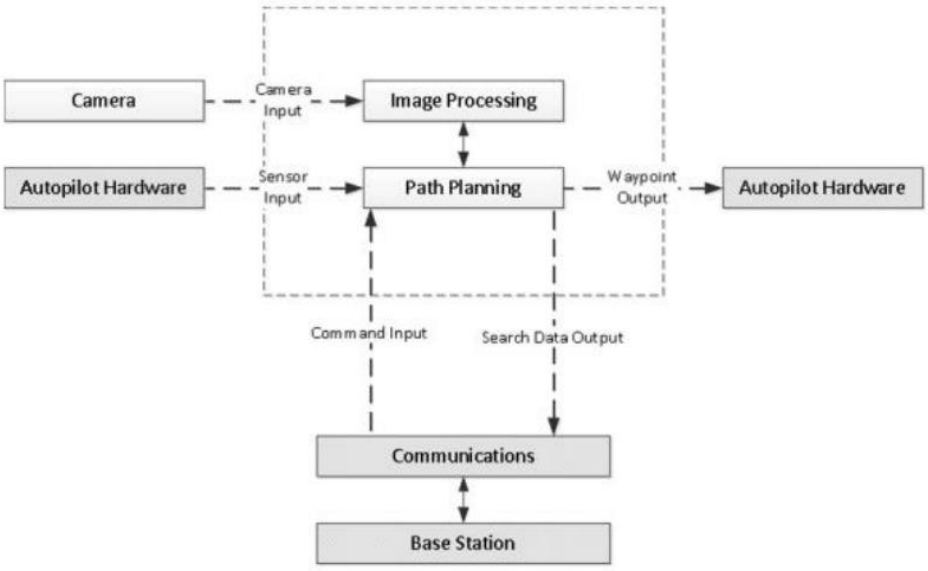
A tarefa de projetar e testar algoritmos de controle de VANTs é uma tarefa difícil, bastante onerosa e que pode gerar riscos a propriedades e pessoas. Simulações nem sempre são suficientes para testar os algoritmos, pois não levam em consideração todos os aspectos como funcionamento do controlador, ruídos 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 é apresentado em linhas gerais os elementos que compõem tal ambiente de simulação, como hardware *on-board*, 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 são apresentados. No trabalho em Gans *et al.* (2009) é apresentado um ambiente para executar simulações HITL para VANT usando um sistema de controle de visão que incorpora câmera, aviônicos e túnel de vento. A plataforma desenvolvida possui um VANT com piloto automático *on-board* e uma câmera interligados ao software de realidade virtual.

3.3 Arquiteturas da Literatura

Como mencionado anteriormente, os autores em (BROWN; ESTABROOK; FRANKLIN, 2011) descrevem uma arquitetura para VANTs que poderiam ser utilizados em missões de resgate de caminhantes perdidos no deserto. O projeto foi chamado WiND e desenvolvido no *Worcester Polytechnic Institute*. Um microprocessador é responsável pelo controle de três partes do VANT: piloto automático, sistema de navegação e sistema de processamento de imagem. A Figura 10

apresenta um diagrama da arquitetura utilizada.

Figura 10 – Arquitetura utilizada no sistema de resgate do projeto Wind.



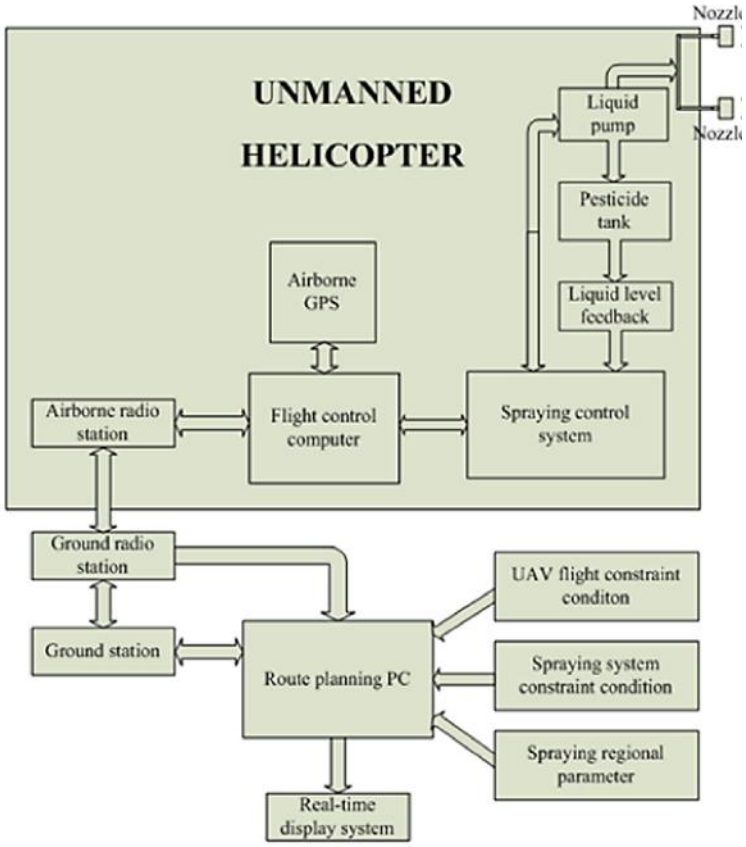
Fonte: Brown, Estabrook e Franklin (2011).

Os autores em Xue *et al.* (2016) apresentam uma arquitetura para VANTs, mostrada na Figura 11, que executa uma pulverização contra pragas agrícolas, composta por um controlador de voo, planejamento de rota e controle de aspersão. A aeronave é controlada de acordo com coordenadas determinadas previamente pelo GPS, ou seja, a rota foi previamente calculada. Um sinal distinto é enviado para controlar o sistema de aspersão que realimenta em tempo real o sistema a bordo, responsável por repassar as informações para a estação de solo.

Na Figura 12 um sistema mais elaborado para controle da trajetória, proposto em Prodan *et al.* (2013), utiliza controle preditivo para lidar com pertubações externas durante o voo. Esse sistema possui dois laços de controle: um laço interno mais rápido a bordo do VANT e um laço externo mais lento, implementado nos computadores de controle da estação de solo. Assim, o laço externo fornece a rota a ser seguida, enquanto o laço interno controla a dinâmica do VANT, minimizando os erros durante a execução da trajetória.

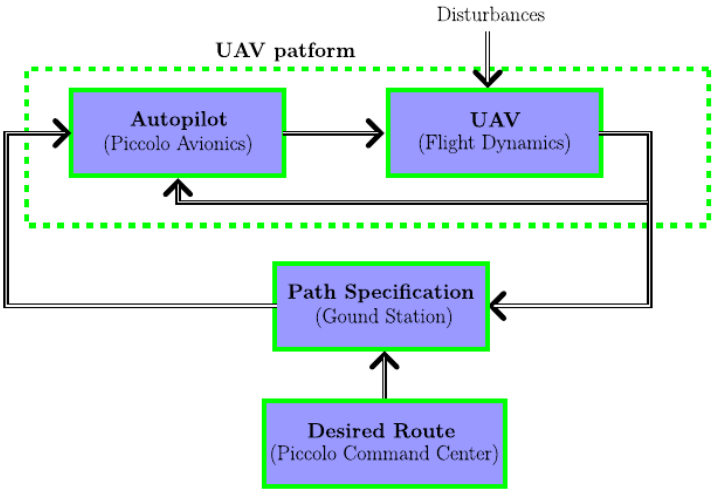
Uma arquitetura mais avançada para hardware e software, chamada *LIDAR Obstacle Warning and Avoidance System* (LOWAS), é descrita em Ramasamy *et al.* (2016) para aplicações com VANTs. LOWAS utiliza tecnologia *Light Detection and Ranging* (LIDAR), que emprega sensores laser e componentes eletro-óptico de direcionamento da aeronave que proporcionam alta resolução e precisão angular em diversas condições operacionais. O sistema 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

Figura 11 – Arquitetura utilizada no sistema de pulverização de pragas agrícolas.



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

Figura 12 – Arquitetura utilizada no sistema de controle da trajetória.

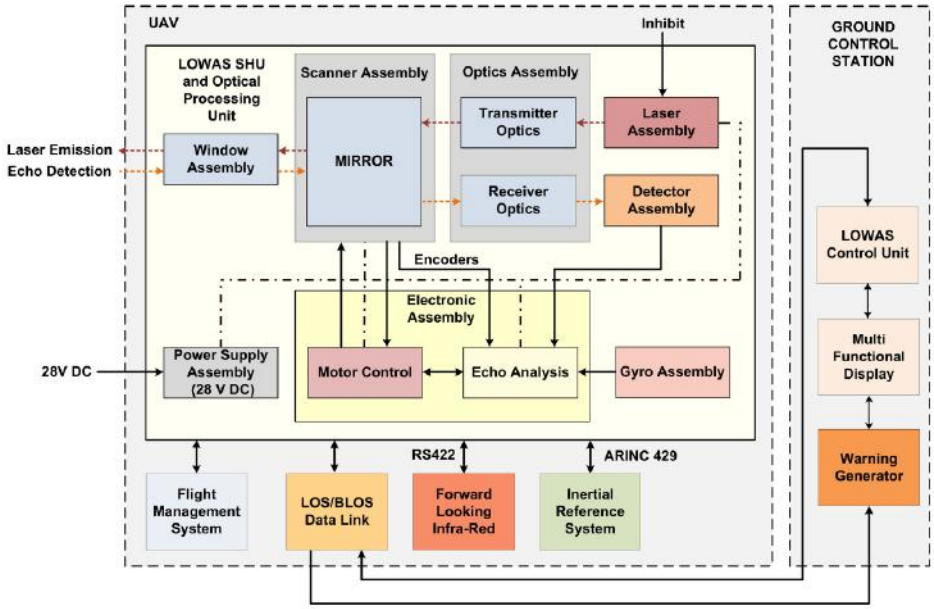


Fonte: Prodan *et al.* (2013).

voo, fornecendo os avisos necessários à execução de manobras de evasão. A Figura 13 ilustra a arquitetura LOWAS.

As interações com o piloto na estação de solo envolvem links de comunicação que

Figura 13 – Arquitetura utilizada no sistema para evitar colisões.



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

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.

3.4 Considerações Finais

Este capítulo apresentou os principais trabalhos relacionados a presente tese que servirão de base para o desenvolvimento da arquitetura proposta. O próximo capítulo apresenta o problema abordado.

CAPÍTULO

4

CENÁRIO DO PROBLEMA ABORDADO

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

Confúcio

4.1 Considerações Iniciais

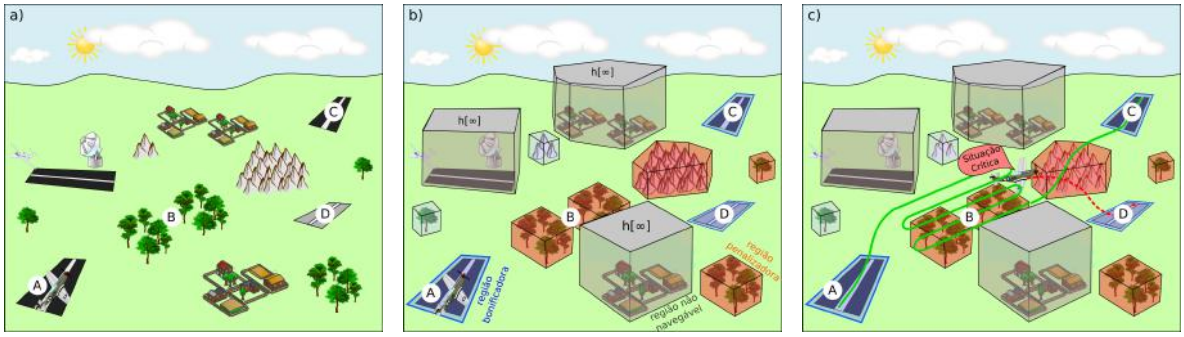
O cenário do problema abordado será descrito inicialmente em linhas gerais e ilustrado através de um exemplo. Em seguida, o problema de planejamento/replanejamento de rotas será estabelecido através de um modelo estocástico. Esse cenário do problema é a base para a elaboração da arquitetura proposta nesta tese.

4.2 Descrição Geral do Cenário do Problema Abordado

O sistema proposto será aplicado no contexto do problema de planejamento e execução de missão ilustrado na Figura 14, em que o VANT não pode sobrevoar áreas povoadas, aeroportos comerciais e regiões de tempestade, denominadas de Zonas de Exclusão Aérea ou *No-Fly Zones* (NFZ). Assume-se que o VANT pode sobrevoar o restante do mapa como florestas, montanhas e pistas para aeromodelos (Figura 14 (a)). Incertezas relacionadas à aeronave e ao ambiente aumentam o risco da aeronave sobrevoar uma NFZ ou colidir com uma montanha, por exemplo. A abordagem utilizada para modelar tal problema foi tratar NFZs como obstáculos com altura ilimitada (h[∞]), desse modo, o VANT não sobrevoaria tais regiões. Montanhas e florestas também são consideradas obstáculos, porém como sua altura é limitada, o VANT poderá sobrevoá-las (Figura 14 (b)). Dado esse tratamento para o problema, o planejamento de trajetória passa a ser executado em uma região não-convexa (devido a presença de NFZs).

Considera-se que todo ou pelo menos parte do mapa seja conhecido inicialmente, mas atualizações seriam realizadas durante o voo via enlace de comunicação com um operador em

Figura 14 – Cenário geral do problema abordado.



(a) Cenário inicial. (b) Cenário mapeado. (c) Cenário com rota de voo.

Fonte: Elaborada pelo autor.

solo ou por meio dos sensores da própria aeronave. As regiões descritas na Figura 14 (b) são classificadas em um dos conjuntos abaixo como estabelecido em Arantes (2016).

1. **Conjunto Não-Navegável (Φ^n):** A aeronave não pode sobrevoar e pousar nas regiões deste conjunto, também conhecido como NFZ (por exemplo, aeroportos, cidades, bases militares).

2. **Conjunto Navegável com Penalização (Φ^p):** O VANT pode sobrevoar regiões deste conjunto, mas não é desejável que ele pouse sobre elas (por exemplo, florestas, montanhas, lagos).

3. **Conjunto Navegável e Bonificador (Φ^b):** O VANT pode sobrevoar e é desejado que ele pouse numa das regiões deste conjunto (por exemplo, grandes áreas gramadas ou campos com plantações rasteiras).

4. **Conjunto 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.

A partir do cenário definido na Figura 14 (c), a situação descrita a seguir ilustra o tipo de problema a ser tratado nesta tese em que tem-se o plano da missão e as tarefas que os sistemas MOSA e IFA devem realizar.

Plano da Missão: *O VANT deve iniciar o seu voo na pista **A** e seguir em direção à pista **C**, devendo sobrevoar a região **B** para retirada de fotos. O plano de voo original considera sobrevoar as florestas, desviando das casas e do aeroporto.*

Sistema MOSA: *O sistema MOSA é o responsável pelo completo cumprimento da missão definida acima. Dessa forma, ele planeja toda a rota que minimiza o combustível da aeronave*

*entre os pontos decolagem (**A**) e pouso final (**C**). Em seguida, ele controla a rota a ser seguida pela aeronave, seu voo sobre a região **B** e o momento certo para retirar fotografias nessa região.*

Sistema IFA: *Enquanto o VANT realiza a missão, a aeronave apresenta uma falha como, por exemplo, superaquecimento da bateria. O sistema supervisor (IFA) detecta a falha e decide abortar a rota atual, encerrando o controle da missão executado pelo MOSA e acionando um algoritmo para planejar uma rota de pouso emergencial. Isso leva o VANT a pousar na região **D**.*

Algoritmos de planejamento e replanejamento de rotas serão integrados ao sistema MOSA considerando os conjuntos de regiões mencionados. No caso do sistema IFA, algumas falhas que podem levar ao pouso de emergência naquelas regiões foram modeladas durante o mestrado como descrito em Arantes (2016): motor para de funcionar (ψ^m), superaquecimento da bateria (ψ^b), aeronave passa a executar curvas apenas para esquerda (ψ^{s1}) e a aeronave executa curvas apenas para direita (ψ^{s2}).

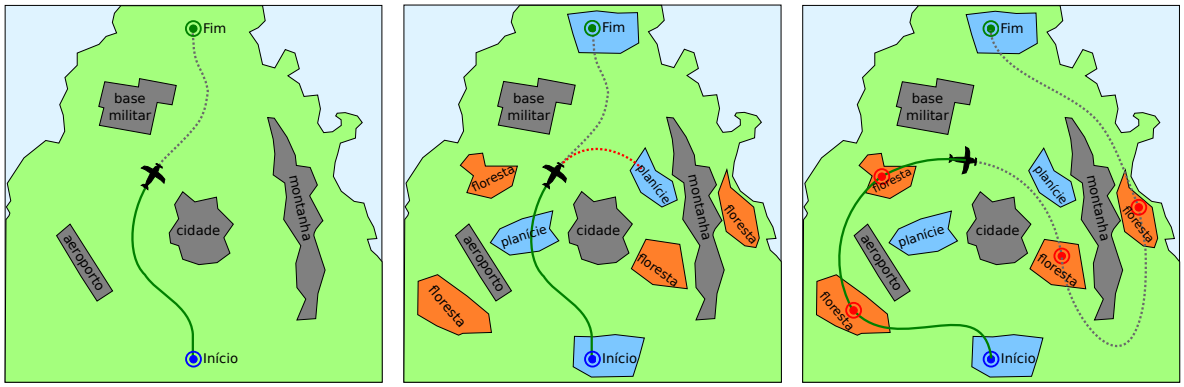
4.3 Problema de Planejamento de Missão e Segurança

A Figura 15 ilustra o cenário geral de três trabalhos publicados anteriormente, ilustrando a forma como esses trabalhos se encaixam dentro do contexto abordado nesta tese. A Figura 15 (a) mostra um cenário no qual o problema de planejamento de caminho foi abordado em Blackmore, Ono e Williams (2011), Arantes *et al.* (2016).

Os autores em Blackmore, Ono e Williams (2011) introduziram o Problema de Planejamento de caminho Não-convexo com restrições de probabilidade (*Chance-constraint*), nomeado em inglês *Chance-constraint Non-convex Path-planning Problem* (CNPP). Nesse problema, um VANT deve alcançar a posição objetivo partindo da posição inicial, evitando regiões não-navegáveis, como aeroporto regular, regiões povoadas, bases militares ou áreas de tempestade. É possível sobrevoar florestas e montanhas, mas 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* e o problema torna-se 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 este problema usando um modelo de Programação Não-Linear Inteira-Mista (PNLIM), mas 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 o mesmo problema em Arantes *et al.* (2016).

A Figura 15 (b) mostra um cenário em que o problema de replanejamento de rota ocorre a partir da detecção de uma situação crítica. Esse problema foi abordado por Arantes *et al.* (2015), onde o replanejamento de rota é definido com a mesma *chance-constraint* e dentro

Figura 15 – Cenários ilustrativos do problema de planejamento de missão e segurança.



(a) Exemplo de planejamento de rota abordados em Arantes *et al.* (2016).

(b) Exemplo de replanejamento de rota emergencial abordado em Arantes *et al.* (2015).

(c) Exemplo de missão com sistema de segurança abordado em Arantes *et al.* (2017).

Fonte: Elaborada pelo autor.

de um ambiente não-convexo de Blackmore, Ono e Williams (2011), Arantes *et al.* (2016). O tipo de situação crítica é considerado em Arantes *et al.* (2015), uma vez que pode impactar na capacidade de voo da aeronave (dinâmica do VANT). Assim, as regiões são previamente mapeadas e rotuladas como regiões não-navegáveis, em que o VANT não pode pousar; regiões bonificadora, na qual o VANT pode pousar; e regiões penalizadoras, em que o VANT pode pousar, mas sofrendo alguns danos. Um método chamado *Multi-Population Genetic Algorithm for security* (MPGA4s) foi aplicado para encontrar uma trajetória de pouso emergencial.

A Figura 15 (c) ilustra um cenário para planejamento de missão, que considera todos esses aspectos juntos. Nesse caso, o planejamento de rota é exigido para diferentes posições iniciais e objetivos, e isso pode ser feito durante o voo. Além disso, durante o voo, uma situação crítica pode ocorrer, levando a aeronave a realizar um pouso de emergência. Esse é o cenário abordado em Arantes *et al.* (2017), em que os métodos HGA4m e MPGA4s são combinados para a execução da missão com segurança.

Os aspectos mencionados acima serão devidamente descritos por duas formulações matemáticas estocásticas. A primeira formulação descreve o CNPP, como introduzido por Blackmore, Ono e Williams (2011) e utilizado em Arantes *et al.* (2016).

$$Minimizar \quad \sum_{t} \mathbf{u}_t \cdot \mathbf{u}_t \tag{4.1}$$

sujeito a:

$$\mathbf{x}_T = \mathbf{x}_{goal} \tag{4.2}$$

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t + \omega_t \qquad \forall(t) \tag{4.3}$$

$$\mathbf{x}_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_{x_0}), \quad \omega_t \sim \mathcal{N}(0, \Sigma_{\omega_t}) \qquad \forall(t) \tag{4.4}$$

$$Pr\left[\bigwedge_j \bigwedge_t \mathbf{x}_t \notin \mathbb{O}_j^n\right] \geq 1 - \Delta \tag{4.5}$$

No modelo acima, o vetor de estado $\mathbf{x}_t$ tem as posições do veículo p e velocidades v, enquanto que o vetor de controle $\mathbf{u}_t$ possui aceleração definida pelas expressões (4.6).

$$\mathbf{x}_t := [p_x\ p_y\ v_x\ v_y]^T, \quad \mathbf{u}_t := [a_x\ a_y]^T \tag{4.6}$$

As entradas incluem a posição inicial ($\hat{x}_0$), a posição objetivo ($\hat{x}_{goal}$), as matrizes de controle da dinâmica A e B, o horizonte de planejamento ($t = 0, \cdots T$), o intervalo de tempo (Δt), as incertezas externas (ω_t) e o nível de risco assumido (Δ). 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 (4.1) 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 (4.2). A transição de estados do veículo está descrita na restrição (4.3), 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}). Existe um ruído Gaussiano branco aditivo (ω_t) com matriz de covariância (Σ_{ω_t}) aplicado a cada transição de estado. A restrição (4.5) define que os estados do veículo ($\mathbf{x}_t$) devem estar fora dos obstáculos ($\mathbb{O}_j^n$) em todo tempo t. A expressão $Pr[\cdot] \geq 1 - \Delta$ reporta que a probabilidade do veículo estar fora de todos os obstáculos deve ser maior ou igual a $1 - \Delta$. Assim, a probabilidade de falha deve ser menor que Δ.

A segunda formulação, introduzida em Arantes *et al.* (2015), descreve todas as restrições relacionadas ao problema de replanejamento de rotas sobre situações críticas.

$$Minimizar\ Pr\Big(\bigvee_j \mathbf{x}_T \in \mathbb{O}_j^p\Big) - Pr\Big(\bigvee_i \mathbf{x}_T \in \mathbb{O}_j^b\Big) \tag{4.7}$$

sujeito a:

$$\mathbf{x}_{t+1} = F_\Psi(\mathbf{x}_t, \mathbf{u}_t) + \omega_t \qquad \forall(t) \tag{4.8}$$

$$\mathbf{x}_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_{x_0}), \quad \omega_t \sim \mathcal{N}(0, \Sigma_{\omega_t}) \qquad \forall(t) \tag{4.9}$$

$$Pr\left[\bigwedge_j \bigwedge_t \mathbf{x}_t \notin \mathbb{O}_j^n\right] \geq 1 - \Delta \tag{4.10}$$

No modelo acima, o vetor de estado $\mathbf{x}_t$ tem as posições do veículo p, velocidades v e ângulo α, enquanto que o vetor de controle $\mathbf{u}_t$ possui aceleração a e variação angular ε, como definido pelas expressões (4.11).

$$\mathbf{x}_t := [p_x \; p_y \; v \; \alpha]^T, \quad \mathbf{u}_t := [a \; \varepsilon]^T \tag{4.11}$$

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 ($t = 0, \cdots K$), o intervalo de tempo (Δt), as incertezas externas (ω_t) e o nível de risco (Δ). 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 (4.7) 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 (4.8), 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}). Um ruído Gaussiano branco (ω_t) com matriz de covariância (Σ_{ω_t}) é aplicado a cada transição. A restrição (4.10) define que os estados do veículo devem estar fora dos obstáculos ($\mathbb{O}_j^n$) em todo tempo t.

4.4 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 mapeados e os tipos de situações críticas a serem tratados. Assim, a partir do problema de planejamento, replanejamento e execução da missão com segurança, descrito neste capítulo, uma arquitetura de hardware será elaborada visando fornecer à aeronave maior autonomia para tomadas de decisão dentro do problema proposto.

CAPÍTULO

5

METODOLOGIA

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

Albert Einstein

5.1 Considerações Iniciais

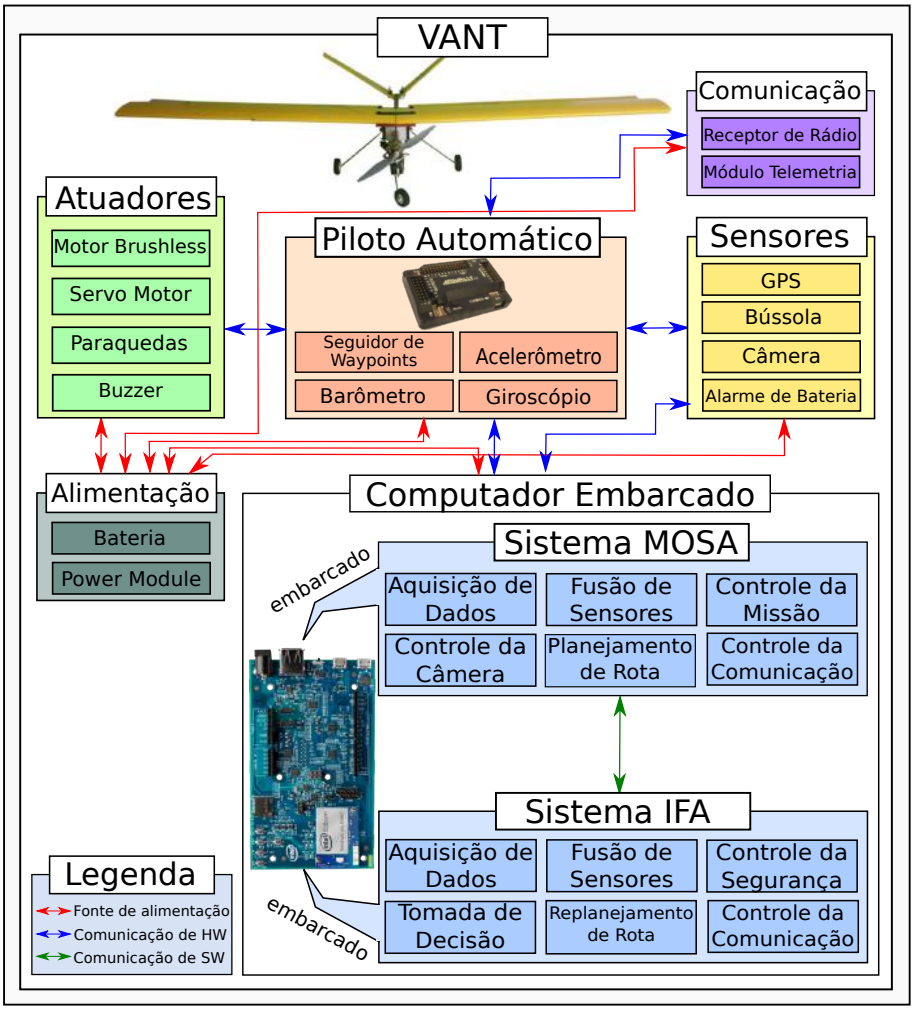
O presente capítulo apresenta a metodologia empregada. Inicialmente é mostrada a arquitetura proposta e o como as arquiteturas da literatura poderiam ser incorporadas ao presente trabalho. Em seguida, o estado atual das plataformas, Ararinha e iDroneAlpha, é descrito. Por fim, será mostrado um protocolo que está sendo definido para especificação do mapa e da missão a serem utilizados em voos reais.

5.2 Arquitetura Proposta

A metodologia proposta para execução deste doutorado consiste na construção passo a passo de uma arquitetura que combina o desenvolvimento de sistemas em hardware e software para VANTs. A arquitetura introduzida por este projeto inova ao permitir a execução de diferentes tipos de missões, garantir requisitos de segurança de voo em tempo real e fornecer o maior nível de autonomia possível para a aeronave. Nesta seção, algumas arquiteturas para VANTs existentes na literatura serão adaptadas a arquitetura proposta com o objetivo de ilustrar claramente quais serão as contribuições trazidas pela nossa arquitetura. Também será mencionada a forma como se pretende avaliar e comparar o sistema desenvolvido utilizando trabalhos da literatura.

A arquitetura do sistema embarcado proposto é mostrada na Figura 16, em que são integradas uma camada de monitoramento do cumprimento da missão (MOSA) e uma outra que garante a segurança da aeronave (IFA).

Figura 16 – Arquitetura proposta do sistema embarcado na aeronave.



Fonte: Elaborada pelo autor.

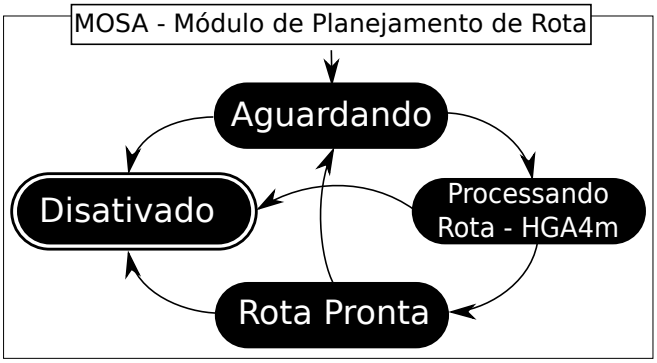
A arquitetura possui seis principais componentes de hardware: atuadores, piloto automático, sensores, comunicação, alimentação e computador embarcado. Os atuadores são os mecanismos que permitem o deslocamento e a execução das manobras do VANT; os sensores são responsáveis pela captura de informações do ambiente e auxílio a navegação, missão e segurança; o piloto automático é responsável pela navegação, mantendo a aeronave na rota planejada; 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 e, finalmente, tem-se o computador embarcado *dual-core* que fica dedicado a execução dos sistemas MOSA e IFA, que garantem a autonomia de voo da aeronave.

Dentro dos sistemas MOSA e IFA, um conjunto de módulos foram implementados a fim de garantir o correto cumprimento da missão levando em conta aspectos relacionados à segurança. Os principais módulos do MOSA são aquisição de dados, fusão de sensores, controle da missão, controle da câmera, controle da comunicação e planejamento de rota. No módulo de planejamento de rota foi integrado o algoritmo HGA4m, descrito em Arantes *et al.* (2016).

Os principais componentes do IFA são aquisição de dados, fusão de sensores, controle da segurança, tomada de decisão, controle da comunicação e replanejamento de caminho, onde o MPGA4s, descrito em Arantes *et al.* (2015), foi integrado. 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.

A Figura 17 ilustra a máquina de estados do módulo de planejamento de rota no sistema MOSA. Inicialmente, o subsistema encontra-se no estado **Aguardando**, o qual pode trocar para o estado **Processando Rota**, em que o método HGA4m é acionado, mudando então seu estado para **Rota Pronta**, quando a nova rota é enviada para o piloto automático. O estado **Desativado** pode ser alcançado sempre que uma falha crítica ocorrer, levando o módulo MOSA a abortar o processamento atual.

Figura 17 – Máquina de estados do módulo de planejamento de rota no MOSA.

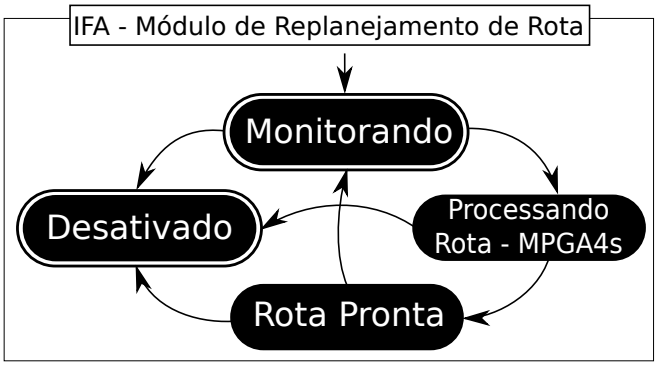


Fonte: Elaborada pelo autor.

A Figura 18 mostra a máquina de estados do módulo de replanejamento de rota no sistema IFA. Inicialmente, o módulo de replanejamento de rota encontra-se no estado **Monitorando** e troca para **Processando Rota** no caso de alguma falha no sistema, em que o método MPGA4s é acionado. Assim que a rota for retornada pelo MPGA4s, o estado atual torna-se para **Rota Pronta** e então a rota é enviada para o piloto automático. Em seguida, o sistema retorna ao estado **Monitorando**. Esse módulo pode entrar no estado **Desativado**, caso alguma falha severa ocorra no sistema IFA. Sendo assim, o sistema IFA é projetado para tomar o controle da aeronave em caso de situações adversas, tomando decisões como abortar a missão, retornar a base, realizando um pouso de emergência, ou ainda efetuar o disparo do paraquedas.

A seguir será apresentado como é possível fazer uma integração das arquiteturas existentes na literatura, descritas no Capítulo 3, com a arquitetura aqui proposta. O projeto Wind, descrito em Brown, Estabrook e Franklin (2011), está focado no processamento das imagens obtidas pela câmera visando a localização e resgate dos caminhantes. A arquitetura descrita em Xue *et al.* (2016) prioriza um controle automático da pulverização durante o voo. Nos trabalhos Brown, Estabrook e Franklin (2011), Xue *et al.* (2016), a arquitetura está fortemente relacionada

Figura 18 – Máquina de estados do módulo de replanejamento de rota no IFA.



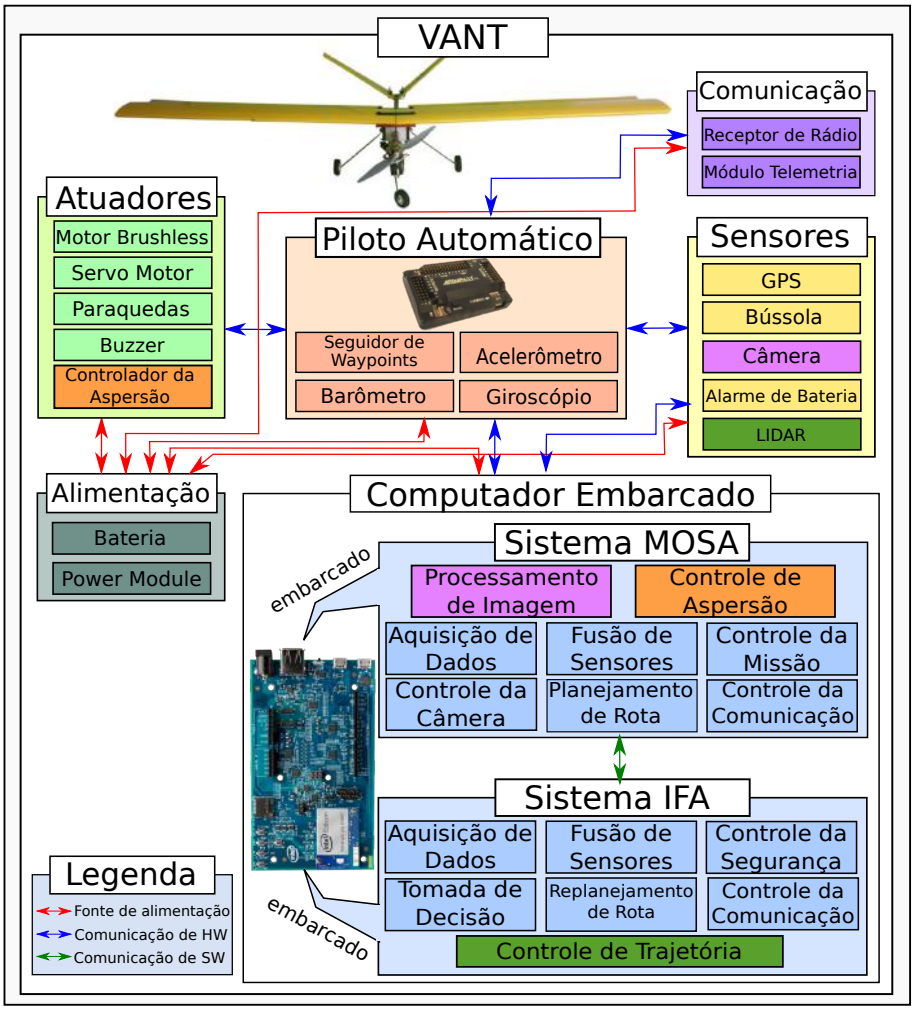
Fonte: Elaborada pelo autor.

à aplicação ou tipo de uso da aeronave. Os autores também consideram uma constante troca de informações entre a estação de solo e a aeronave, o que reduz o nível de autonomia da mesma para tomada de decisões. Observa-se também que esses trabalhos apresentam sistemas dedicados ao controle da trajetória ou voo, em que Brown, Estabrook e Franklin (2011) é o único que considera um sistema para planejamento da rota embarcado na aeronave. A Figura 19 apresenta como ficariam dispostas as funcionalidades descritas em Brown, Estabrook e Franklin (2011), Xue *et al.* (2016), Ramasamy *et al.* (2016) adaptadas ao contexto desta tese de doutorado.

Pode-se observar que o sistema controlador da missão (MOSA) será implementado de forma genérica o suficiente para se comunicar com os dados obtidos por um sistema de processamento de imagem ou um sistema de pulverização. Tais sistemas podem ser integrados ao MOSA conjuntamente ou separadamente. O sistema controlador da missão supervisiona se, por exemplo, no caso da pulverização da plantação, o sistema de aspersão é acionado como esperado quando o VANT atingir a região alvo. No caso da câmera, o sistema supervisiona, por exemplo, se a obtenção das imagens está ocorrendo e, se for o caso, se a retransmissão está sendo feita para a estação de solo. Diferente do que foi definido 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.

A arquitetura definida em Ramasamy *et al.* (2016) não foca em uma aplicação específica, mas propõem o desenvolvimento de sistemas mais elaborados para 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 como ilustrado na Figura 19. Nesse caso, se um obstáculo ou outra aeronave aparecesse durante a execução da missão, detectado pelo sensor LIDAR, o sistema IFA tomaria o controle da trajetória da aeronave e executaria as manobras evasivas necessárias para evitar

Figura 19 – Arquitetura proposta adaptada ao contexto de Brown, Estabrook e Franklin (2011), Xue *et al.* (2016), Ramasamy *et al.* (2016).



Fonte: Elaborada pelo autor.

colisão, retornando o controle ao MOSA em seguida. Além disso, 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), podem ser integrados à arquitetura proposta na Figura 19.

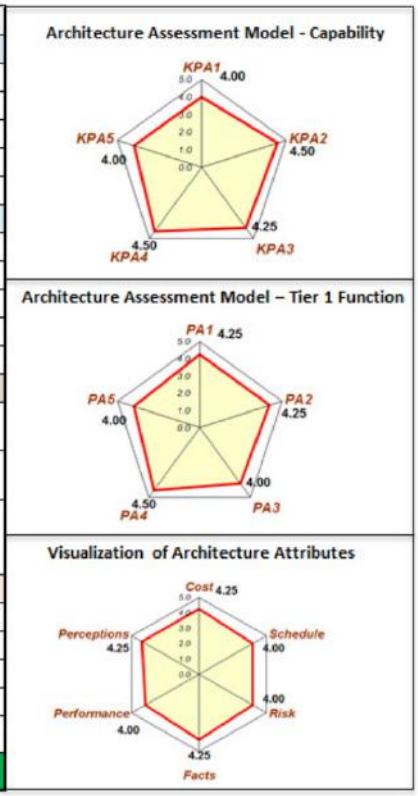
A metodologia proposta para viabilizar as funcionalidades descritas nesta seção passa pelo tratamento modular dos sistemas e estabelecimento de protocolos de comunicação bem definidos. O desenvolvimento de cada módulo passará por implementações em hardware e software, de forma similar as metodologias adotadas em Brown, Estabrook e Franklin (2011), Xue *et al.* (2016), Ramasamy *et al.* (2016), Morozov e Janschek (2016). Acredita-se que no decorrer do desenvolvimento de cada sistema, contribuições específicas serão alcançadas cujos resultados poderão ser contrastados com as abordagens da literatura. Além disso, vantagens e desvantagens de se estabelecer uma arquitetura mais ampla serão objetos de discussão deste projeto de doutorado, em que os ganhos em termos de robustez ou funcionalidade serão comparados às

possíveis perdas de desempenho do sistema embarcado e seu impacto na carga útil da aeronave. Para isso, comparações com arquiteturas como aquelas propostas em Brown, Estabrook e Franklin (2011), Xue *et al.* (2016) serão conduzidas.

Os autores em Renault (2015) apresentam um modelo de avaliação específico para sistemas de VANTs. O modelo de avaliação proposto combina heurísticas de projeto com representações quantitativas, qualitativas e visuais que permitem avaliar a probabilidade de que a arquitetura gerada atenda aos requisitos de desempenho e capacidade. O modelo de avaliação da arquitetura retorna uma pontuação combinada, a partir de vários aspectos que indicam se a arquitetura projetada é aceitável ou não. Os aspectos principais considerados na avaliação são: desempenho dos seus atributos chaves, conceitos e funcionalidades da arquitetura, níveis de autonomia e conceito de modelo de dados. O modelo de dados é usado para determinar as relações entre funções e suas entradas e saídas. O modelo de níveis de autonomia pode ser usado para determinar um nível adequado de autonomia do sistema Husar e Stracener (2013) para que o VANT execute as missões. Cenários de missões são usados para avaliar a probabilidade de que o VANT possa executar missões regulares e identificar deficiências no sistema. Os atributos são avaliados com uma pontuação numérica como mostrado na Figura 20.

Figura 20 – Utilizando o modelo de avaliação.

UAV Architectecture Assessment Model					
Model Type	**Capability**	**Description**	**Acceptable**	**Max**	**Actual**
Key Performance Attributes	KP1	Modularity	3.5	5	4
	KP2	Reliability	4	5	4.5
	KP3	Robustness	4	5	4.25
	KP4	Flexibility	4	5	4.5
	KP5	Survivability	3.5	5	4
Model Type	**Function**	**Description**	**Acceptable**	**Max**	**Actual**
Quantitative Fuzzy Qualitative	PA1	Transport	4	5	4.25
	PA2	Communicate	4	5	4.25
	PA3	Power	4	5	4
	PA4	Surveillance	4.25	5	4.5
	PA5	Control	4.25	5	4
Model Type	**Category**	**Description**	**Acceptable**	**Max**	**Actual**
Quantitative	LOA	System Autonomy	4	5	4.25
Quantitative	Modified DM2	DoDAF Meta-Model (DM2)	4	5	4.25
Fuzzy	VAA	Visualization of Architecture Attributes	4	5	4.13
Model Type	**Scenario Category**		**Acceptable**	**Max**	**Actual**
Scenario Based Assessment	Search and Rescue		80	100	85
	Border Patrol		80	100	90
	Tactical Law Enforcement		80	100	85
	Maritime Surveillance		80	100	90
Totals			371.5	465	404.88
Overall Score Percentage					**87.07**



Fonte: Renault (2015).

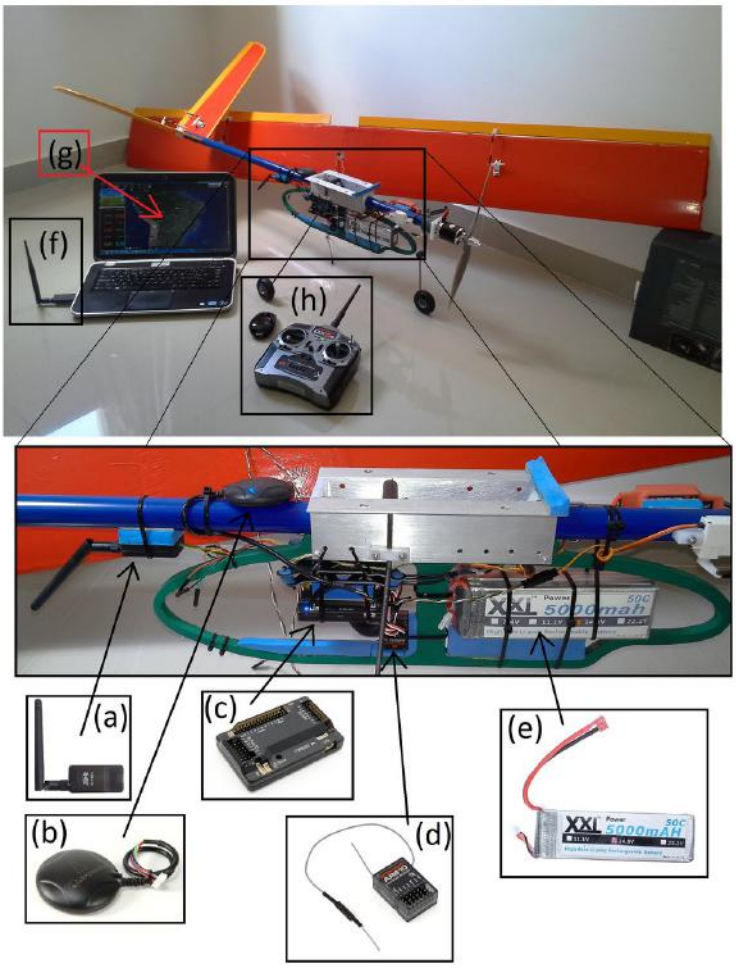
Assim, espera-se utilizar ou adaptar critérios como aqueles definidos em Renault (2015) para auxiliar numa correta definição da arquitetura proposta e avaliação dos sistemas desenvolvi-

dos.

5.3 Plataforma Ararinha

A Figura 21 apresenta o estado atual de uma das plataformas a serem utilizadas neste doutorado. Trata-se da aeronave Ararinha, modelo elétrico desenvolvida pelo grupo. A aeronave mostrada na figura é capaz de realizar voos rádio controlados enviando os *logs* de voo através da telemetria para a estação de controle de solo. A seguir é apresentada uma descrição detalhada dos equipamentos presentes na Figura 21, bem como suas funcionalidades.

Figura 21 – Plataforma Ararinha utilizada para validação da arquitetura proposta.



Fonte: Arantes (2017).

(a) Telemetria na aeronave: permite que a aeronave receba remotamente configurações para o Piloto Automático (AP) e estabeleça uma rota de voo definida por um conjunto de *waypoints*.

(b) GPS com IMU: módulo contendo um GPS e *Inertial Measurement Unit* (IMU) de alta precisão, que fornecem ao AP a posição e orientação da aeronave.

(c) Piloto automático: utilizado para navegação autônoma, capaz de seguir um conjunto de *waypoints* e executar algumas manobras básicas, como manter altitude, voar em círculos e estabilizar aeronave.

(d) Receptor do rádio controle: recebe comandos do controle manual, utilizado em situações que exigem a intervenção humana.

(e) Bateria recarregável: a bateria alimenta o motor e todo hardware da aeronave.

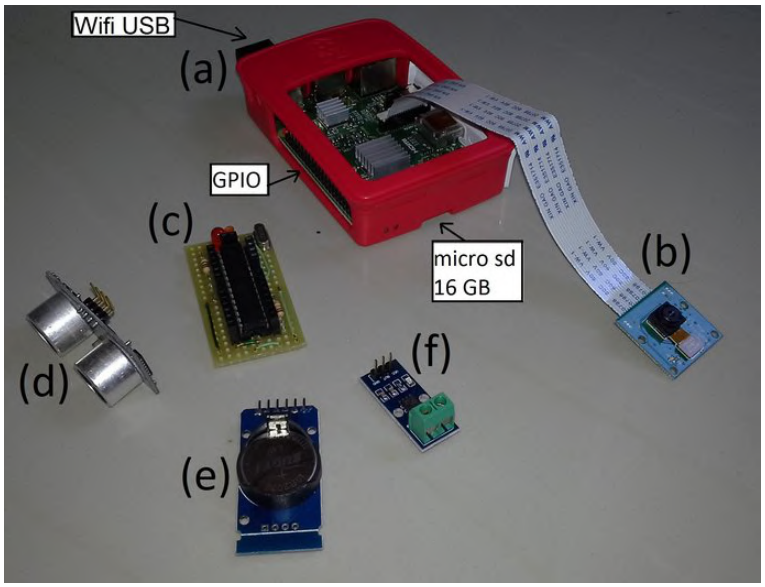
(f) Telemetria no computador: permite configurar o AP pelo computador utilizando a estação de controle de solo Mission Planner ou via protocolo MAVLink.

(g) Computador: utilizado para estabelecer uma rota de forma manual, utilizando o Mission Planner ou estabelecer uma rota de forma automática, através de algoritmo planejador de rota.

(h) Rádio controle: permite controlar o VANT diretamente ou passar a navegação para o AP.

A Figura 22 mostra alguns componestes que poderão ser adicionados a plataforma afim de permitir a realização de voos autônomos. Isso será possível através da implementação os sistemas MOSA e IFA dentro do *companion computer*, nesse exemplo, temos a Raspberry Pi:

Figura 22 – Componentes que serão integrados a plataforma visando voos autônomos.



Fonte: Arantes (2017).

(a) Raspberry Pi: trata-se de um mini computador, que estará na aeronave em que serão embarcados os sistemas MOSA e IFA.

(b) Câmera: utilizada para registrar o voo. As informações obtidas serão úteis para projetar algoritmos auxiliares que podem detectar situações adversas e auxiliar durante o pouso e decolagem autônomo.

(c) Arduíno: utilizado para fazer o processamento dos dados dos sensores. O uso de Arduíno é uma alternativa mais segura do que adicionar os sensores diretamente aos GPIOs na Raspberry Pi, pois uma sobrecarga nos circuitos dos sensores pode queimá-la, inutilizando o sistema IFA.

(d) Sensor Ultrassônico: será utilizado durante o pouso e decolagem para detectar a proximidade com o solo com mais precisão do que os atuais sensores GPS e barômetro da aeronave.

(e) Relógio de tempo real: visto que a Raspberry Pi não possui uma bateria interna capaz de manter a data (dia e hora) atualizadas, este módulo será utilizado para manter os sistemas MOSA e IFA a par dessa informação, o que pode ser útil, principalmente, para registrar quando ocorreram as mudanças no cenário e acesso correto a condições climáticas.

(f) Sensor de corrente: adicionado na bateria do VANT para monitorar o consumo, podendo ser utilizado para detectar falhas elétricas, sobrecarga e fazer previsão da carga restante.

micro sd: trata-se de um cartão de memória com 16 GB, que armazena o sistema operacional e demais softwares a serem embarcados na plataforma, além dos dados do voo.

GPIO: *General Purpose Input Output* (GPIO) é utilizado para entrada e saída de dados da Raspberry Pi, podendo ser conectados sensores e atuadores nos GPIOs. Como forma de exemplificar isso, pode-se pensar em um GPIO utilizado para disparar um paraquedas.

Wifi USB: permite conectar a Raspberry Pi a uma rede Wifi para acesso a internet em solo e atualizar as configurações do IFA e do MOSA facilmente com um computador.

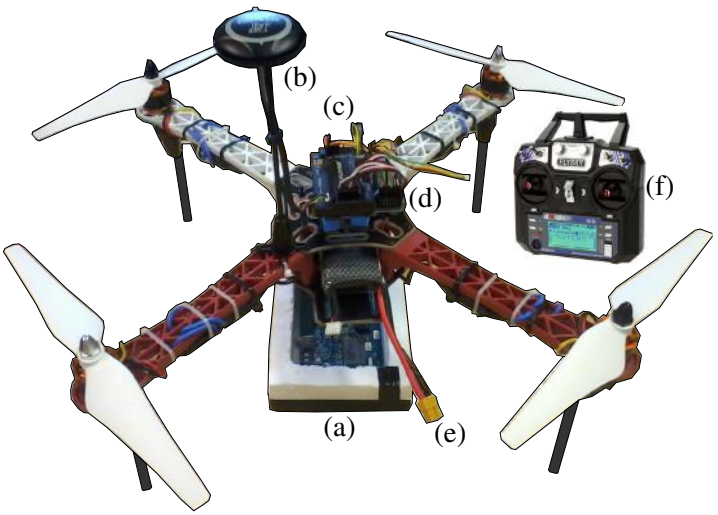
5.4 Plataforma iDroneAlpha

A Figura 23 mostra a plataforma iDroneAlpha montada durante a elaboração desta tese. O iDroneAlpha está sendo utilizado nas etapas iniciais, como uma alternativa ao Ararinha, por se tratar de uma aeronave menor e de asa rotativa. Isso permite que os experimentos se tornem mais fáceis de serem conduzidos. Os componentes que integram esse VANT e suas funcionalidades são descritos a seguir:

(a) Intel Edison: computador embarcado capaz de realizar o processamento *on-board* tornando o voo da aeronave com maior nível de autonomia.

(b) GPS: módulo capaz de obter a posição e orientação da aeronave.

Figura 23 – Plataforma iDroneAlpha montada durante este trabalho.



Fonte: Elaborada pelo autor.

(c) Piloto automático: utilizado para navegação autônoma e capaz de seguir vários *waypoints*.

(d) Receptor do rádio controle: recebe comandos do rádio controle.

(e) Bateria recarregável: a bateria alimenta os motores e todo hardware da aeronave.

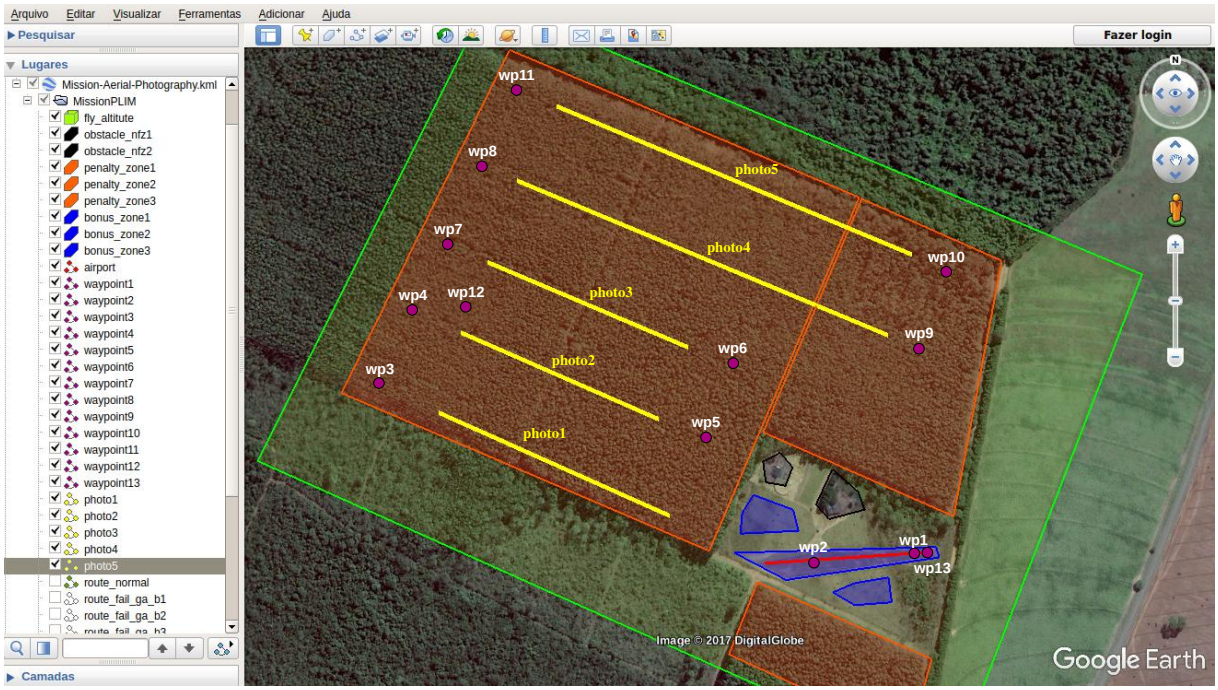
(f) Rádio controle: permite controlar o VANT diretamente ou passar a navegação para o AP.

5.5 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 será utilizada durante esta tese 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 (GOOGLE-EARTH, 2017). Dessa forma, um protocolo próprio está sendo definido para a criação do mapa e da missão usando esse ambiente. Na Figura 24, essa ferramenta foi utilizada para definir o cenário (mapa), os *waypoints*, e os locais de retirada de fotografias da missão. 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 forma a seguir os *waypoints* e efetuar a retirada de fotografias.

Esse mapeamento prévio das regiões é importante para que o VANT conheça o cenário de voo, uma vez que, não será feito processamento de imagens 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 foram analisados.

Figura 24 – Protocolo de mapeamento das regiões e definição da missão.



Fonte: Elaborada pelo autor.

As seguintes notações foram definidas afim de mapear as regiões em que ocorrerá a missão:

obstacle_nfz: esta região define as áreas que o VANT está estritamente proibido de sobrevoar ou pousar, podendo representar, por exemplo, edifícios, casas, bases militares, aeroportos, entre outras.

penalty_zone: esta região define as áreas que o VANT pode sobrevoar, mas não deve pousar. Caso pouse, uma penalização será aplicada. Isso porque nessa área encontram-se estruturas de tamanho mediano como as árvores, lagos, montanhas, etc.

bonus_zone: esta região define as áreas que o VANT pode sobrevoar e pousar. Essa região pode representar a pista de pouso do VANT, planícies, entre outras.

As seguintes notações foram definidas afim de registrar as atividades relacionadas a missão:

waypoint: define um ponto de passagem que a aeronave deve cumprir durante a sua missão. Os *waypoints* definidos serão utilizados pelo MOSA para estabelecer a rota a ser seguida pela aeronave.

photo: define um ponto ou região de pontos em que fotografia(s) deve(m) ser retirada(s). O sistema MOSA ao atingir esse ponto ou região de pontos efetua a retirada da(s) fotografia(s).

A justificativa da escolha dessa ferramenta para a definição do mapa e da missão se deu pelo ambiente de edição, que atende todos os requisitos que necessitávamos: simplicidade de utilização; capacidade de criação de polígonos, linhas e caminhos; capacidade de salvar/carregar as informações em arquivo e facilidade de interpretação do arquivo.

5.6 Considerações Finais

Este capítulo apresentou a arquitetura proposta e como as arquiteturas da literatura poderiam ser encaixadas neste trabalho. O estágio atual da plataforma Ararinha foi mostrado, bem como do iDroneAlpha. Por fim, foi mostrado um protocolo que está sendo definido para especificação do mapa e da missão. O capítulo seguinte apresenta os resultados preliminares obtidos até o momento.

CAPÍTULO

6

RESULTADOS PRELIMINARES

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

6.1 Considerações Iniciais

O presente capítulo apresenta alguns dos resultados preliminares já obtidos. Maiores detalhes podem ser encontrados nos artigos publicados anexados a este documento.

6.2 Resultados do Artigo do GECCO 2017

Esta seção discute os resultados computacionais obtidos em dois experimentos desenvolvidos e publicados no GECCO 2017, (ARANTES *et al.*, 2017), disponível no Apêndice A. 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 executado sobre a plataforma de hardware Intel Edison. O modelo dos métodos HGA4m e do MPGA4s foram brevemente apresentados no Capítulo 4. O segundo experimento é um estudo de caso mais específico em que simulações são conduzidas sobre a arquitetura do sistema proposto descrita no Capítulo 5. A Tabela 8 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).

Um total de 40 mapas foi aleatoriamente gerado, usando um gerador proposto em Blackmore, Ono e Williams (2011), para os experimentos com HGA4m, onde rotas para o planejamento da missão devem ser obtidas. 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), onde o replanejamento da rota para pouso em caso de situação crítica deve ser realizado. Os primeiros resultados foram obtido após se executar o HGA4m por 10 vezes sobre cada um dos 40 mapas. O HGA4m foi executado em duas plataformas de hardware. A

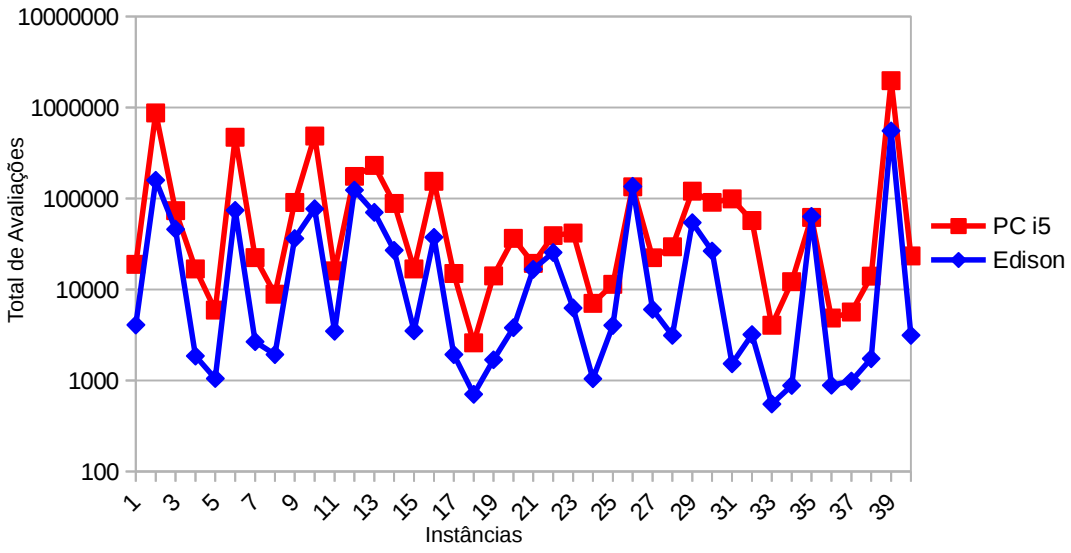
Tabela 8 – Configurações utilizadas nos métodos HGA4m e MPGA4s.

Parâmetro	Valor HGA4m	Valor MPGA4s
número de populações	3	3
tamanho da população	3x13	3x13
taxa de crossover	5	0,5
taxa de mutação	0,7	0,75
critério de parada	10 segundos	1 segundo

Fonte: Elaborada pelo autor.

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 25 mostra o número de avaliações de *fitness* médios sobre 10 execuções em cada instância.

Figura 25 – Número de avaliações por instância no método HGA4m.

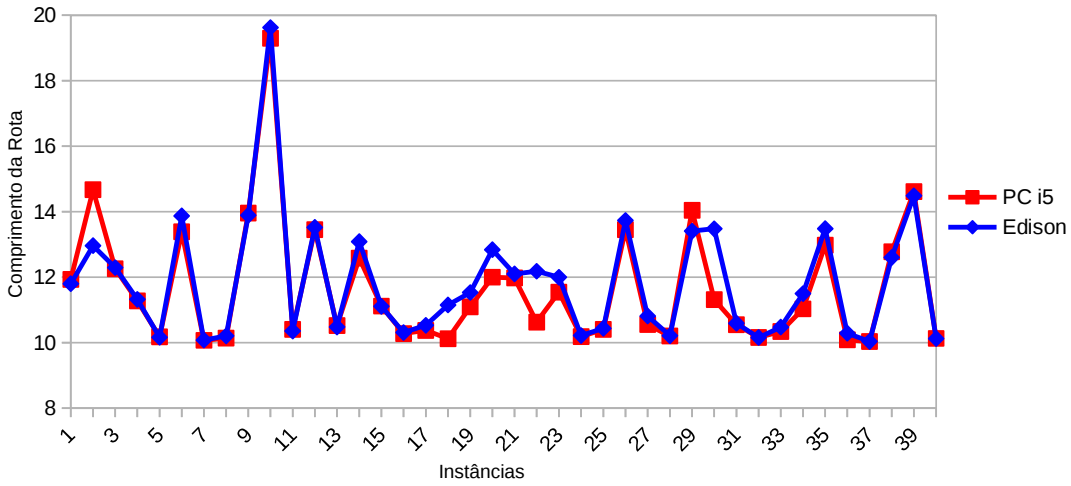


Fonte: Elaborada pelo autor.

O número de avaliações é reduzido quando executando o HGA4m sobre a Edison como poderia ser esperado. Isto significa que o HGA4m explora menos o espaço de busca 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 26. O tamanho da rota retornada com a Edison foi similar a alcançada sobre a plataforma PC. Logo, a segunda métrica indica que a qualidade das rotas retornadas é mantida quando o HGA4m é executado sobre uma plataforma de hardware inferior.

A seguir é avaliado o MPGA4s sobre ambas plataformas, PC i5 e Edison, para resolver um conjunto de 60 mapas. A Figura 27 mostra o número de avaliações (avaliações da função de

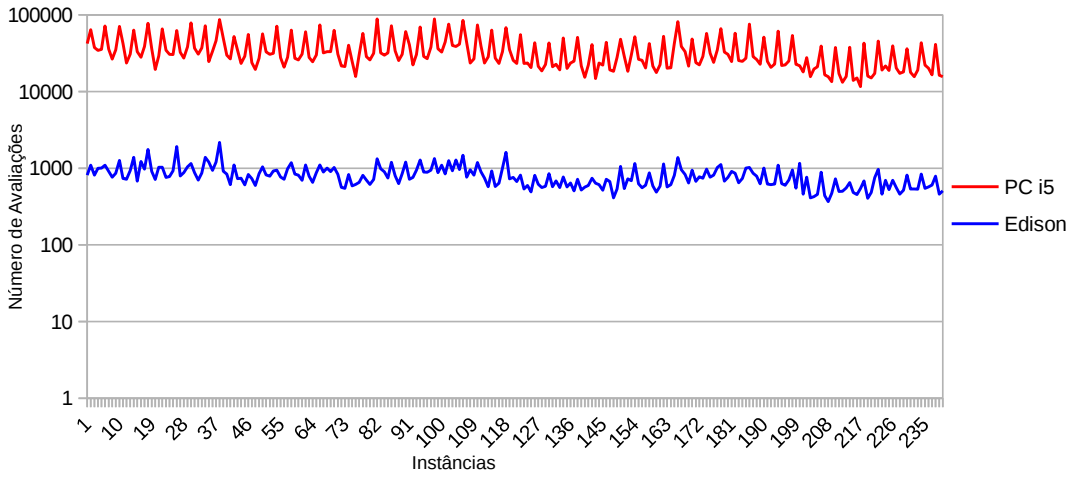
Figura 26 – Tamanho da rota por instância no método HGA4m.



Fonte: Elaborada pelo autor.

fitness).

Figura 27 – Número de avaliações por instância no método MPGA4s.

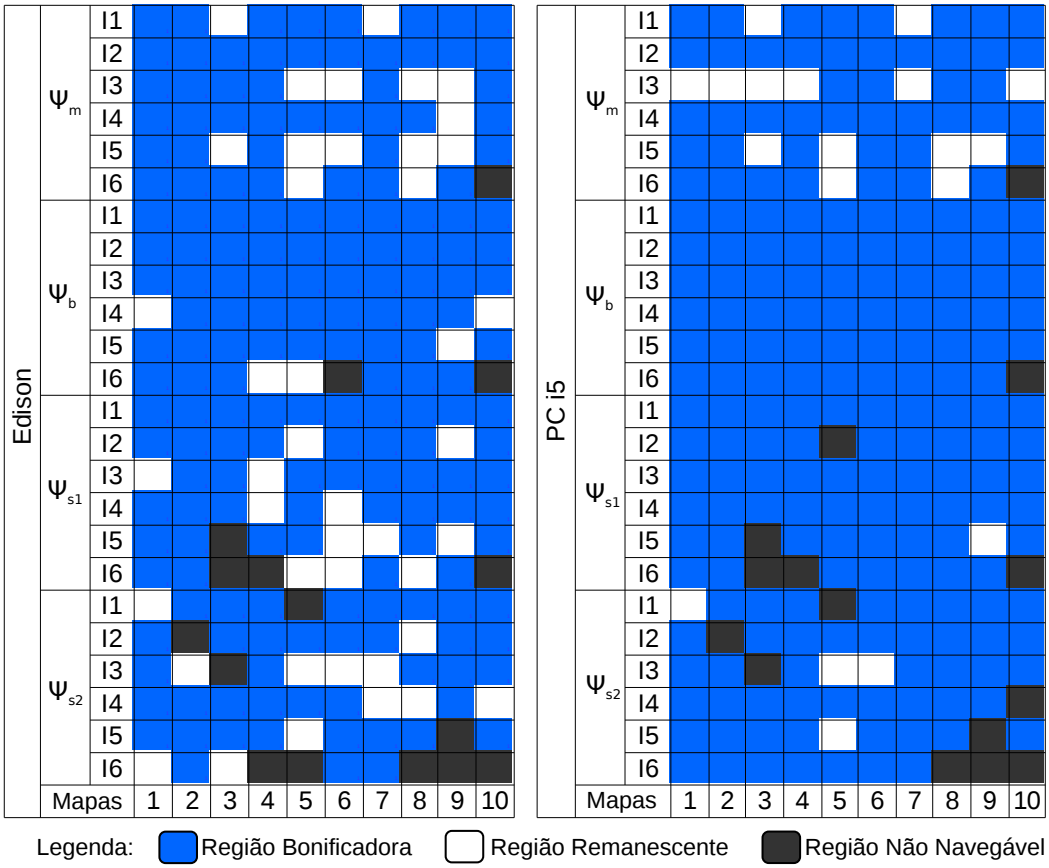


Fonte: Elaborada pelo autor.

Existem em média 805 avaliações da função de *fitness* executando MPGA4s com a plataforma Edison, contra 34.440 avaliações executando sobre PC i5. A plataforma PC executa aproximadamente 42 vezes mais avaliações do que a Edison, mas a qualidade do desempenho mantêm-se satisfatória como mostra a Figura 28.

O local de pouso pode ser pensando como uma maneira direta de medir a qualidade da solução. A Figura 28 ilustra os locais onde o pouso ocorreu. Esse local é obtido através das rotas do MPGA4s em 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}). A cor azul indica o local de pouso

Figura 28 – Local de pouso em ambas as arquiteturas de hardware para o MPGA4s.



Fonte: Elaborada pelo autor.

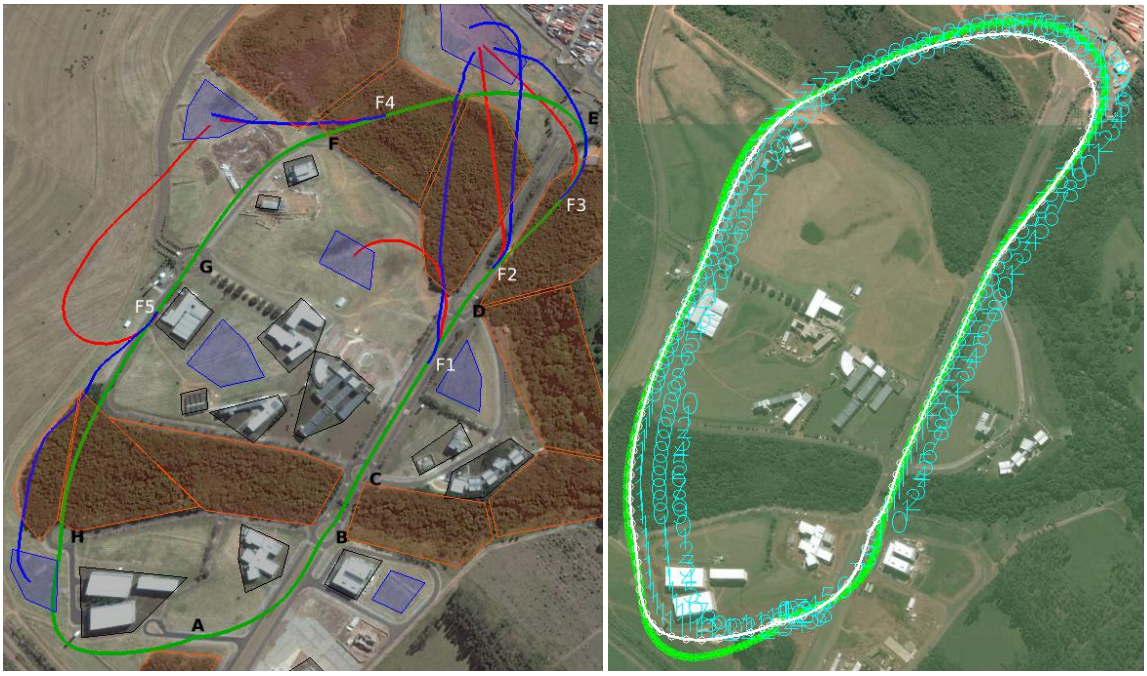
em região bonificadora, cor branca indica pouso em região remanescente e cor cinza escuro indica violação da restrição de não-navegabilidade. Essa imagem revela uma grande similaridade entre os locais de pouso para as rotas retornadas em ambas arquiteturas de hardware.

Um estudo de caso também foi avaliado em que uma aplicação do mundo real para planejamento de missão e segurança é conduzido. Nesse experimento, o desempenho do sistema é também comparado quando executado sobre as plataformas Edison e PC i5. A Figura 29 (a) mostra os resultados obtidos.

A rota completa planejada com o método HGA4m é representada por uma linha verde, em que o VANT deve executar todo o plano da missão. 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 29 (a) também mostra pontos rotulados por **F1**, **F2**, **F3**, **F4** e **F5**, que são lugares onde uma falha crítica foi detectada. Então, o sistema proposto é avaliado quando uma falha ocorre para cada uma 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

Figura 29 – 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 usando SITL.

Fonte: Elaborada pelo autor.

Tabela 9 – Diferentes simulações efetuadas usando 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.

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 29 (b) 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.

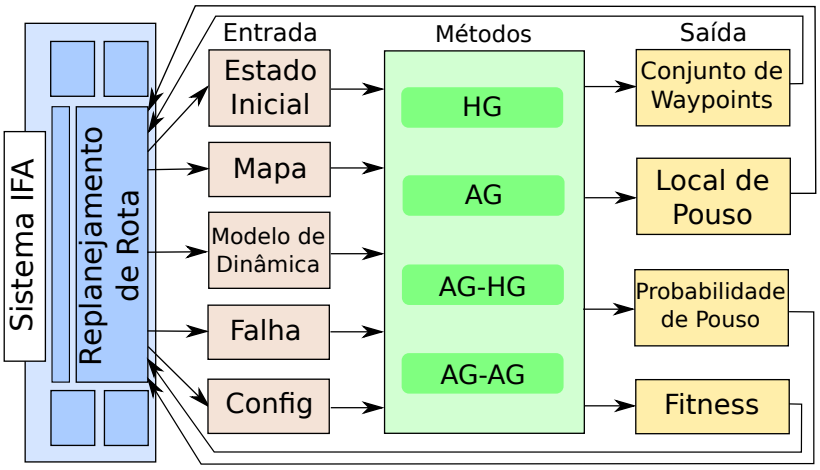
Os vídeos na Tabela 9 mostram quatro de 12 experimentos conduzidos usando SITL, incluindo quando falhas ocorrem, executando sobre PC i5 e Edison. É possível ver que a interação entre os módulos embarcados funcionam adequadamente e que o consumo de tempo com HGA4m e MPGA4s não compromete os experimentos.

6.3 Resultados do Artigo do ICTAI 2017

Esta seção discute brevemente os resultados computacionais obtidos nos experimentos desenvolvidos e publicados no ICTAI 2017, (ARANTES *et al.*, 2017), disponível no Apêndice B. Esses experimentos avaliam o desempenho de quatro métodos quando executado sobre a plataforma de hardware Intel Edison. Um estudo de caso foi avaliado, em que simulações são conduzidas sobre a arquitetura do sistema proposto descrita no Capítulo 5.

A Figura 30 detalha o módulo de replanejamento de rota do sistema IFA. Após detectar uma falha, o sistema IFA aciona o módulo de replanejamento que envia as entradas: Estado Inicial, Mapa, Modelo de Dinâmica, Falha e Configurações para o algoritmo de replanejamento. No módulo de replanejamento, uma das seguintes estratégias podem ser selecionadas: Heurística Gulosa (HG), Algoritmo Genético (AG), combinação do AG executando em paralelo (AG-AG) e combinação do AG com HG executando em paralelo (AG-HG). A estratégia selecionada retornará a seguinte informação: Conjunto de *Waypoints*, Local de Pouso, Probabilidade de Pouso e *Fitness*. Essas saídas retornam ao módulo de replanejamento de rota que comanda o AP para abortar a rota atual (missão) e seguir o novo caminho.

Figura 30 – Arquitetura do sistema embarcado no módulo de replanejamento de rotas.

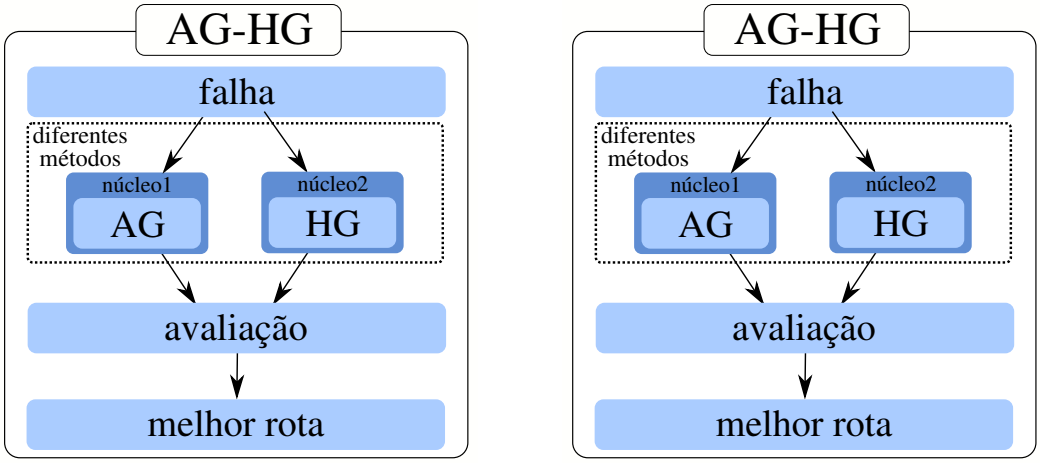


Fonte: Elaborada pelo autor.

Tratam-se de estratégias baseadas em comitê que serão comparadas a uma simples execução da HG e AG. A Figura 31 mostra o fluxo de execução das estratégias propostas. Depois que uma situação crítica foi detectada, dois métodos são executados em paralelo usando os dois núcleos disponíveis e a melhor solução é retornada no final. A Figura 31 (a) mostra a estratégia baseada em comitê que usa a combinação do AG e HG (AG-HG), enquanto o método da Figure 31 (b) executa dois AGs em paralelo (AG-AG).

As configurações do AG são as mesmas descritas na Tabela 8. Nesses experimentos duas situações críticas foram avaliadas: problema na bateria e no motor.

Figura 31 – Estratégias implementadas executando os métodos em paralelo.



(a) Combinando o AG e HG. (b) Duas execuções do AG.

Fonte: Elaborada pelo autor.

Um total de 30 mapas foram avaliados. A Tabela 10 apresenta os resultados alcançados para cada abordagem (HG, AG, AG-HG e AG-AG). Esses resultados são as taxas de sucesso, quando a aeronave executa um pouso seguro. Também é mostrado o Critério de Parada (CP) do algoritmo genético, em que a execução da HG 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).

Tabela 10 – Resultados obtidos depois de avaliar diferentes estratégias em mapas artificiais.

		Intel i5				Intel Edison			
Métodos	CP (ms)	Ψ_b	Ψ_m	Média	Tempo	Ψ_b	Ψ_m	Média	Tempo
HG	-	86,7%	60,0%	73,3%	41	86,7%	60,0%	73,3%	347
AG	250	96,3%	72,0%	84,2%	250	72,3%	62,3%	67,3%	250
AG	500	99,0%	73,0%	86,0%	500	84,7%	65,3%	75,0%	500
AG	1000	98,3%	75,7%	87,0%	1000	91,0%	68,0%	79,5%	1000
AG-HG	250	95,3%	71,0%	83,2%	250	89,3%	62,7%	76,0%	309
AG-HG	500	100,0%	72,3%	86,2%	500	90,3%	65,7%	78,0%	510
AG-HG	1000	99,3%	76,0%	87,7%	1000	92,3%	69,0%	80,7%	1000
AG-AG	250	99,3%	72,3%	85,8%	250	81,0%	65,0%	73,0%	250
AG-AG	500	99,7%	73,3%	86,5%	500	89,7%	68,3%	79,0%	500
AG-AG	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.

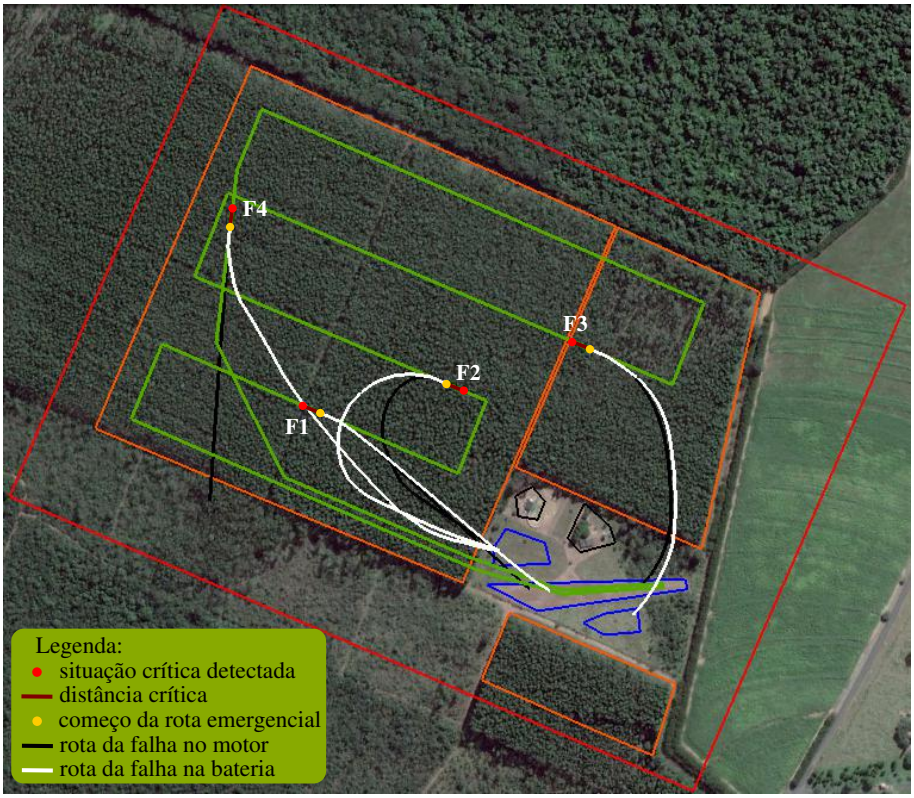
Os resultados dos métodos que executam no computador Intel i5 são competitivos com base nos valores relatados. O VANT conseguiu pousar com segurança, em média de 83,2% até

88,8% dos casos, exceto pela HG que pousa a aeronave 73,3% das vezes. Por outro lado, a HG precisa apenas de 41,2 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.

A HG 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%, o que significa uma diferença em torno de 15% com base na estratégia, bem como o limite de tempo associado. As estratégias baseadas em comitê superam a HG para a maioria dos casos, enquanto eles são melhores do que GA quando comparados dentro do mesmo limite de tempo de execução. A estratégia AG-HG retorna uma taxa de pouso de sucesso em 76,0% em pouco tempo (250 ms), enquanto a estratégia AG-AG tem melhores taxas de sucesso que variam de 500 ms a 1000 ms.

Um conjunto de simulações foram executadas a partir de um estudo de caso sobre um cenário para imageamento aéreo. A Figura 32 mostra a rota seguida pelo VANT durante a missão. Existem várias falhas rotuladas como F1, F2, F3 e F4 que serão simuladas em que o sistema IFA detecta essas falhas. Em seguida, o algoritmo de replanejamento é executado. A nova rota é enviado para o piloto automático e a missão atual é interrompida.

Figura 32 – Resultado do estudo de caso em um cenário do mundo real usando AG.



Fonte: Elaborada pelo autor.

As rotas brancas na Figura 32 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 AG-AG. 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).

A Tabela 11 apresenta algumas das simulações usando SITL para validar as rotas geradas pelo AG-AG, após uma falha crítica na bateria. Essa estratégia foi executada na Intel Edison. As simulações mostram que o piloto automático do VANT é capaz de seguir o caminho de pouso de emergência enviado pelo método.

Tabela 11 – Diferentes simulações efetuadas usando SITL para validar as rotas do AG.

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.

6.4 Considerações Finais

Este capítulo apresentou os resultados preliminares já obtidos durante o doutorado, em que dois artigos publicados em conferências internacionais foram brevemente descritos. O próximo capítulo apresenta o cronograma proposto para execução deste projeto de doutorado.

CAPÍTULO

7

CRONOGRAMA

" Toda e qualquer conquista de vida requer planejamento e estratégia. "

Thaianne Venâncio de Farias

As atividades definidas e o cronograma previsto para a tese são apresentados a seguir:

A) Disciplinas: Cumprimento do número mínimo de créditos em disciplinas exigido pelo programa de pós-graduação do ICMC/USP.

B) Proficiência: Exame de proficiência em língua inglesa.

C) Revisão Bibliográfica: Uma revisão abrangente da literatura será realizada e atualizada ao longo do desenvolvimento da pesquisa.

D) Desenvolvimento: Adaptar e desenvolver uma primeira versão conjunta de IFA e MOSA.

E) Desenvolvimento: Propor/desenvolver algoritmos para fazer a detecção automática de falhas que serão integrados no IFA, baseados nos dados dos sensores. Estratégias como fusão de sensores poderão ser utilizados nesta fase.

F) Qualificação: Ao final de um ano e meio, o candidato deverá redigir e apresentar seu exame de qualificação a uma banca examinadora.

G) Testes, Simulações e Análises dos Resultados: Os algoritmos elaborados serão avaliados procurando os melhores parâmetros e configurações. Simulações em experimentos serão realizadas. Análises e comparações dos resultados encontradas serão comparados entre si e com outras técnicas da literatura.

H) Embarcar: Primeiros testes em *Software-In-The-Loop* (SITL) com o sistema IFA e MOSA. Em seguida, efetuar testes em *Hardware-In-The-Loop*. Por fim, fazer testes em voos reais do sistema embarcado.

I) Elaboração de Artigos: O trabalho desenvolvido será documentado apropriadamente na forma de artigos científicos.

J) Elaboração da Tese: O trabalho desenvolvido será documentado na forma da tese de doutorado, com defesa prevista para Abril de 2019.

Tabela 12 – Cronograma de execução das atividades.

	Meses						
Atividades	01-06	07-12	13-18	19-24	25-30	31-36	37-42
A)	✓	✓					
B)	✓						
C)	✓	✓	✓	•	•	•	
D)			✓	•	•	•	
E)				•	•	•	
F)				•			
G)			✓	•	•	•	
H)			✓	•	•	•	•
I)		✓	✓	•	•	•	•
J)						•	•

O cronograma para execução das atividades previstas segue na Tabela 12. As atividades marcadas com o símbolo (✓) foram executadas e concluídas e as indicadas por (•) estão previstas para serem executadas nas próximas etapas deste trabalho.

CAPÍTULO

8

CONSIDERAÇÕES FINAIS

“ O futuro dependerá daquilo que fazemos no presente. ”

Mohandas Karamchand Gandhi

8.1 Resumo

A presente proposta de doutorado visa o desenvolvimento de um sistema autônomo para Veículos Aéreos Não-Tripulados (VANTs). Pretende-se aumentar a autonomia da aeronave incorporando nela um computador de bordo, em que os sistemas *Mission Oriented Sensor Array* (MOSA) e *In-Flight Awareness* (IFA) serão executados. Nessa primeira etapa, diversas arquiteturas foram estudadas de forma a desenvolver uma arquitetura capaz de integrar o MOSA e IFA com diversos requisitos existentes nas mais diversas aplicações reportadas na literatura.

Até o presente momento, avanços já foram feitos como: (i) construção do VANT iDroneAlpha, que incorpora um computador de bordo com piloto automático e seus periféricos; (ii) diversos experimentos avaliando os algoritmos planejadores de missão e rotas de pouso emergencial na Intel Edison; (iii) validação das rotas obtidas em experimentos *Software-In-The-Loop* (SITL); (iv) aprendizado sobre o funcionamento do ambiente de testes *Hardware-In-The-Loop* (HITL); (v) implementação básica dos sistemas MOSA e IFA.

8.2 Artigos Publicados

Os seguintes artigos foram escritos e publicados 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 A.

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 B.

Heuristic and Genetic Algorithm Approaches for UAV Path Planning under Critical Situation: Publicado na revista *International Journal on Artificial Intelligence Tools* (IJAIT) em 2017. Qualis B1. Resultado direto da minha dissertação de mestrado e indireto desta tese. Encontra-se no Apêndice C.

8.3 Próximas Etapas

Nas próximas etapas, pretende-se avançar em experimentos reais envolvendo as plataformas iDroneAlpha e Ararinha. Uma pequena parte da implementação dos sistemas MOSA e IFA já foi realizada e embarcada na placa Intel Edison. Essas implementações serão continuada de forma que os sistemas MOSA e IFA se tornem robustos o suficiente para a realização de voos autônomos, e capazes de realizarem missões reais em aplicações como imageamento aéreo e pulverização.

Espera-se atingir três objetivos principais: (i) uma contribuição efetiva para a área de VANTs, a partir da introdução de uma arquitetura de propósito geral, resiliente e autônoma; (ii) desenvolvimento dos sistemas relacionados com foco em sistemas autônomos que integrem efetivamente a execução da missão e a segurança de voo; (iii) avaliação da atuação conjunta dos sistemas MOSA e IFA, bem como, de algoritmos de planejamento de missão e pouso, através de experimentos simulados e de voos nas plataformas iDroneAlpha e Ararinha.

Ao final de toda a pesquisa deseja-se divulgar: os códigos-fonte como *open-source* na plataforma GitHub; e a montagem da arquitetura de hardware necessária a automatização do VANT. Espera-se obter uma plataforma de baixo custo e de baixo peso que poderá ser utilizada em VANTs de asa fixa e asa rotativa, além de poder ser expansível para outros veículos não-tripulados como veículos terrestres e marinhos.

REFERÊNCIAS

3DR. *3DR Site*. https://3dr.com/: [s.n.], 2017. Citado na página 36.

ANAC. **Instrução Suplementar - IS: IS n 21-002 Revisão A**. [S.l.], 2012. 21 p. Citado na página 28.

____. **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 28.

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 50 e 65.

ARANTES, J. S. **Planejamento de rota para VANTs em caso de situação crítica: Uma abordagem baseada em segurança**. Tese (Dissertação de Mestrado) — Universidade de Sao Paulo (USP), ago 2016. Sao Carlos, SP. Citado nas páginas 24, 41, 42, 48, 49 e 57.

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 41 e 70.

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 41, 42, 49, 50, 51, 55, 57, 65 e 67.

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 42, 49, 50, 54, 62 e 65.

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 24, 49, 59 e 60.

ARDUPILOT. **ArduPilot Open Source Autopilot**. http://ardupilot.org/: [s.n.], 2017. Citado na página 34.

____. **Choosing a Ground Station**. http://ardupilot.org/plane/docs/common-choosing-a-ground-station.html: [s.n.], 2017. Citado na página 29.

____. **Glossary**. http://ardupilot.org/planner/docs/common-glossary.html: [s.n.], 2017. Citado na página 33.

____. **HITL Simulators**. http://ardupilot.org/dev/docs/hitl-simulators.html: [s.n.], 2017. Citado na página 37.

____. **SITL Simulator**. http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html: [s.n.], 2017. Citado na página 36.

BANKS, J. Discrete event simulation. In: **Winter Simulation Conference**. [S.l.: s.n.], 2000. Citado na página 32.

BEAGLEBOARD. **beagleboard.org**. https://beagleboard.org/black: [s.n.], 2017. Citado na página 35.

BLACKMORE, L.; ONO, M.; WILLIAMS, B. C. **Chance-Constrained Optimal Path Planning with Obstacles**. 2011. IEEE Press. Citado nas páginas 49, 50, 62 e 65.

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 15, 23, 40, 42, 43, 55, 56, 57 e 58.

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 na página 42.

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 28.

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 37.

DJI. **DJI - NAZA-M V2**. http://www.dji.com/naza-m-v2: [s.n.], 2017. Citado na página 34.

FAA, F. A. A. **FAA Modernization and Reform Act of 2012**. Washington, DC, USA, 2012. 296 p. Citado na página 23.

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 39, 40 e 42.

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 23, 24 e 39.

FLIGHTGEAR. **FlightGear Wiki**. http://wiki.flightgear.org/: [s.n.], 2017. Citado na página 32.

____. **SimGear - FlightGear Wiki**. http://wiki.flightgear.org/SimGear: [s.n.], 2017. Citado na página 32.

____. **Table of models - FlightGear Wiki**. http://wiki.flightgear.org/Table_of_models: [s.n.], 2017. Citado na página 32.

FLYSKY. **FlySky Website)**. http://www.flysky-cn.com: [s.n.], 2017. Citado na página 36.

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 42.

GAMBOLD, K. A. Unmanned aircraft system access to national airspace. **Background Paper. Unmanned Experts LLC, Washington DC**, 2011. Citado na página 28.

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 42.

GISA. **Grupo de Interesse em SisVANTs e Aplicações**. gisa.icmc.usp.br/site: [s.n.], 2017. Citado na página 29.

GOOGLE-EARTH. **Google Earth**. https://www.google.com.br/intl/pt-PT/earth/: [s.n.], 2017. Citado na página 62.

HARDKERNEL. **ODROID - Hardkernel**. http://www.hardkernel.com/main/main.php: [s.n.], 2017. Citado na página 35.

HUSAR, R. M.; STRACENER, J. Autonomous systems modeling during early architecture development. **Procedia Computer Science**, v. 20, p. 242 – 247, 2013. ISSN 1877-0509. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050913010685>. Citado na página 58.

INDIAMART. **Unmanned Vehicles**. https://www.indiamart.com/rds/army-defence-quater-guards-stores-supply.html: [s.n.], 2017. Citado na página 28.

INTEL. **The Intel Edison Module - IOT**. https://software.intel.com/en-us/iot/hardware/edison: [s.n.], 2017. Citado na página 35.

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 na página 41.

LEITE, O. **Projeto de Lei Nº 5942**. [S.l.], 2013. 11 p. Citado na página 28.

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 40.

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 na página 40.

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 23, 24, 39, 40 e 42.

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 nas páginas 39 e 40.

MAVLINK. **MAVLink Micro Air Vehicle Communication Protocol**. http://mavlink.org: [s.n.], 2017. Citado na página 30.

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 40.

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 40.

MICROSOFT. **Microsoft Flight**. http://www.microsoft.com/games/flight/: [s.n.], 2017. Citado na página 33.

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 23, 40, 41 e 57.

MTBSAE. **MTB SAE - Serviços Aéreos Especiais**. www.mtbsae.com.br: [s.n.], 2017. Citado na página 29.

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 na página 62.

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 29.

PI, R. **Raspberry Pi - Teach, Learn, and Make with Raspberry Pi**. https://www.raspberrypi.org/: [s.n.], 2017. Citado na página 35.

PIXHAWK. **Hardware in the Loop Simulation Setup**. https://pixhawk.org/users/hil: [s.n.], 2017. Citado na página 33.

____. **Pixhawk Flight Controller Hardware Project**. https://pixhawk.org/: [s.n.], 2017. Citado na página 34.

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, 42, 43, 44 e 57.

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 15, 23, 40, 42, 43, 45, 56 e 57.

RENAULT, A. A model for assessing uav system architectures. **Procedia Computer Science**, v. 61, p. 160 – 167, 2015. ISSN 1877-0509. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050915030100>. Citado nas páginas 40 e 58.

SIGPLANES. **SIG - Rascal 110**. www.sigplanes.com/SIG-Rascal-110-EG-ARF_p_232.html: [s.n.], 2017. Citado na página 29.

WEATHERINGTON, D.; DEPUTY, U. Unmanned aircraft systems roadmap, 2005-2030. **Deputy, UAV Planning Task Force, OUSD (AT&L)**, 2005. Citado na página 28.

WIKIPéDIA. **Microsoft Flight - Wikipédia, a enciclopédia livre**. http://en.wikipedia.org/wiki/Microsoft_Flight: [s.n.], 2017. Citado na página 33.

X-PLANE. **FAA-Certified X-Plane**. http://www.x-plane.com/pro/certified/: [s.n.], 2017. Citado na página 32.

____. **X-Plane 11 is Here**. http://www.x-plane.com/: [s.n.], 2017. Citado nas páginas 32 e 33.

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 15, 23, 40, 42, 43, 44, 55, 56, 57 e 58.

GLOSSÁRIO

Aeronave de Asa Fixa: aeronave capaz de voar usando a asa fixada em seu chassi. Alguns exemplos dessas aeronaves são: Ararinha e Rascal.

Aeronave de Asa Rotativa: aeronave em que sua asa é representada por hélices que giram em torno de um eixo vertical. Alguns exemplos dessas aeronaves são: helicópteros, quadricópteros e multicópteros.

Aviônicos: diz respeito a toda a eletrônica a bordo do avião, como por exemplo, piloto automático, companion computer, receptor de rádio controle, módulo de telemetria e GPS.

Companion Computer: é um computador a bordo da aeronave, que auxilia nas tomadas de decisão durante o voo.

Estação de Controle de Solo: é um software executado em um computador no solo que recebe informações de telemetria do VANT, exibe seu progresso e status, e transmitir comandos a aeronave.

Firmware do Piloto Automático: é o programa que controla o hardware do piloto automático determinando como ele irá operar.

Logs de Voos: conjunto de informações, sobre o comportamento da aeronave, salva pelo piloto automático durante o voo.

MAVLink: protocolo de comunicação utilizado nos pilotos automáticos de VANTs.

Piloto Automático: equipamento acoplado ao VANT capaz de estabilizar o voo, manter a posição e seguir waypoints.

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 componentes são simulados e parte são físicos. A aeronave e o ambiente são simulados, já o AP, telemetria, receptor de rádio controle, controle de rádio são todos hardwares físicos.

Simulação SITL: permite executar missões de VANTs sem nenhum hardware físico, dessa forma, todo o ambiente, hardware da aeronave, hardware 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.

APÊNDICE

A

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 é feita uma análise de desempenho dos algoritmos de planejamento e replanejamento de rotas em VANTs embarcados no *companion computer* Intel Edison. Esse trabalho apresenta uma primeira versão dos sistemas MOSA e IFA e sua arquitetura funcionando em conjunto, sendo 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

GECCO '17, Berlin, Germany

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].

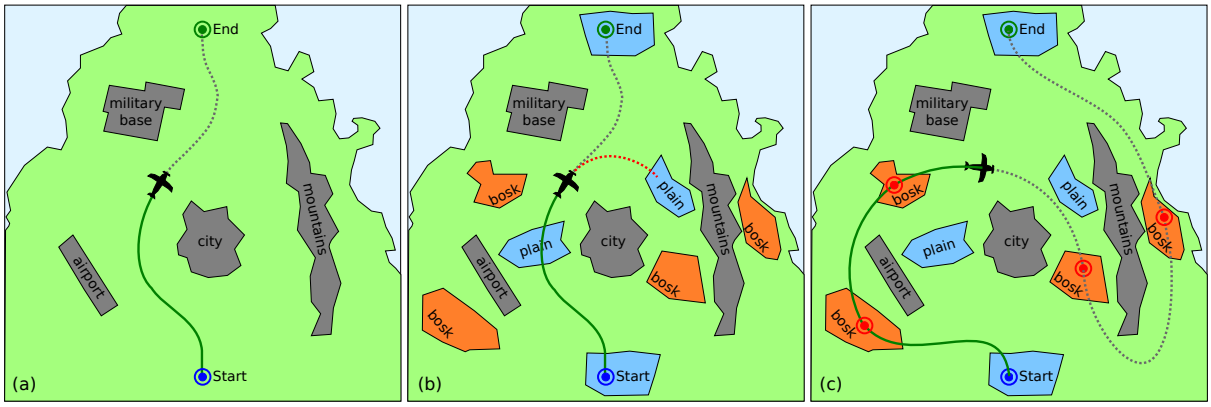


Figure 1: Illustrative scenarios of the mission and path planning problem. (a) Example of path planning addressed in [3] and [2]. (b) Example of emergency path re-planning addressed in [1]. (c) Example of mission addressed in this paper.

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.

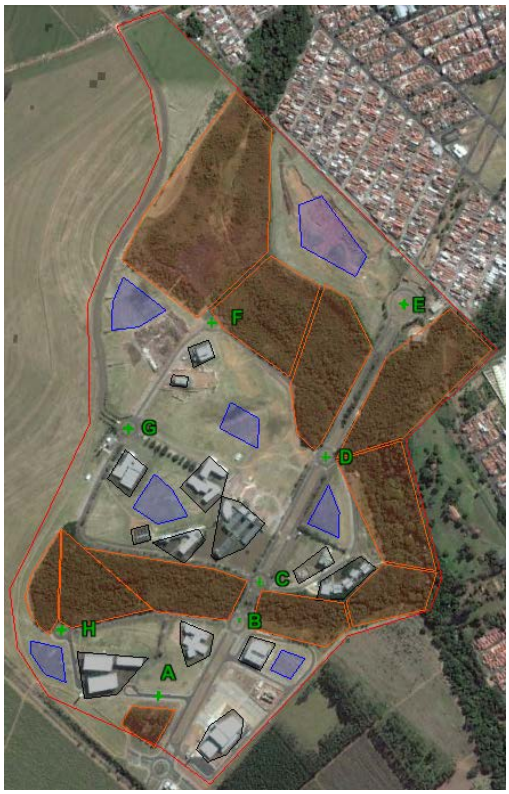


Figure 2: Real world case study for mission and safety.

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.

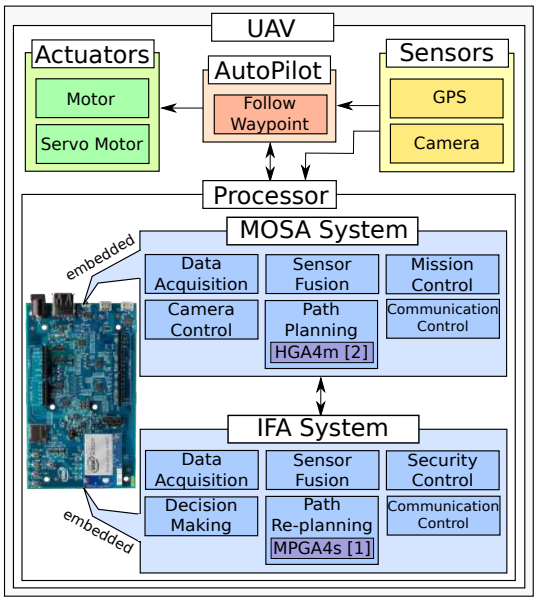


Figure 3: Embedded system architecture in the aircraft.

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.

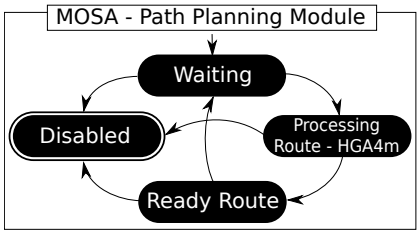


Figure 4: State machine of path planning module in MOSA.

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.

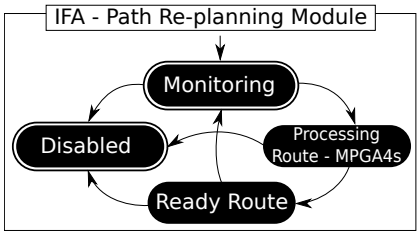


Figure 5: State machine of path re-planning module in IFA.

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.

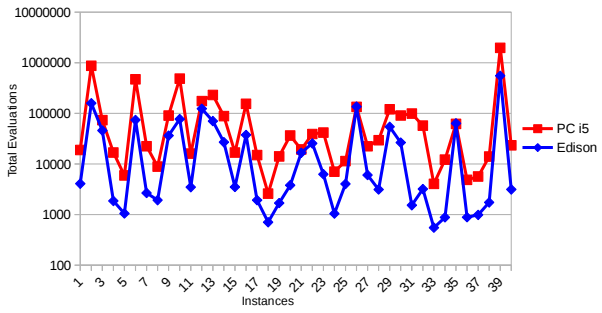


Figure 6: Number of evaluations by instance in the HGA4m.

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.

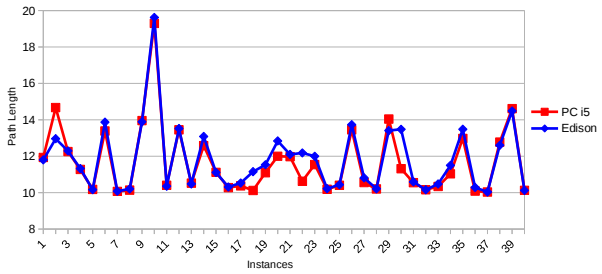


Figure 7: Path length by instance in the HGA4m method.

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).

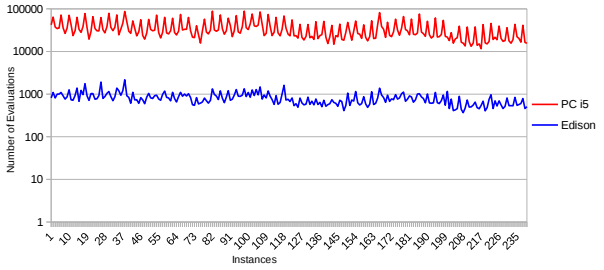


Figure 8: Number of evaluations by instance in the MPGA4s.

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.

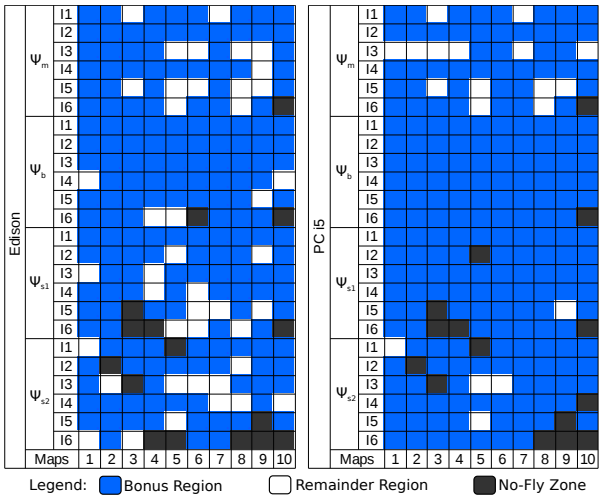


Figure 9: Landing sites in both architectures for MPGA4s.

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

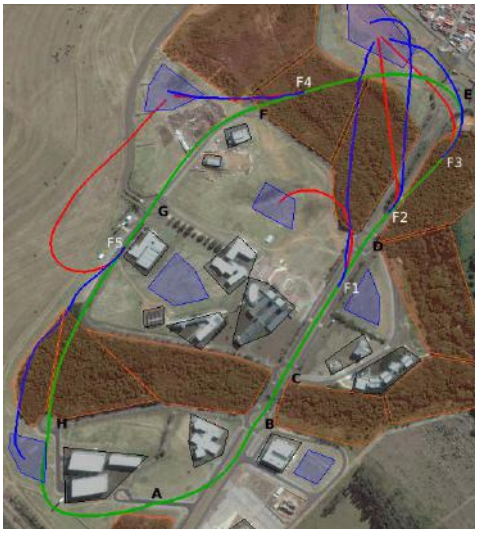


Figure 10: Results of case study in a real world scenario.

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

B

AVALIAÇÃO DE PLATAFORMAS DE HARDWARE EM REPLANEJAMENTO DE ROTA

Artigo completo a ser 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 computador embarcado Intel Edison. Nesse estudo foi destacada a arquitetura do sistema IFA, em que o replanejamento de rotas é efetuado. Esse artigo é resultado direto da presente proposta de doutorado.

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]

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.

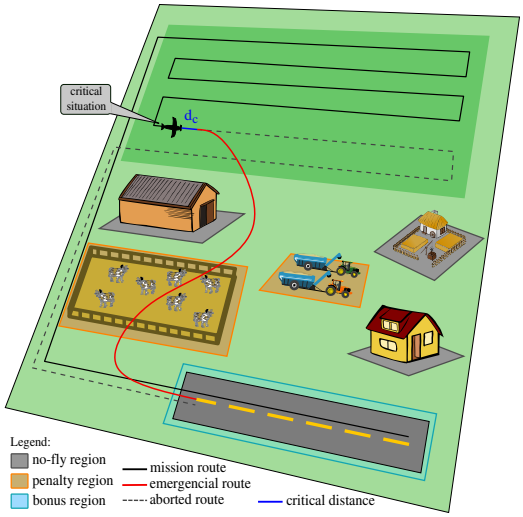


Fig. 1. Illustrative scenario of emergency path re-planning.

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/

$$\mathbf{x}_{t+1} = F_{\Psi}(\mathbf{x}_t, \mathbf{u}_t) + \omega_t \qquad \forall(t) \qquad (2)$$

$$\mathbf{x}_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_{x_0}), \quad \omega_t \sim \mathcal{N}(0, \Sigma_{\omega_t}) \qquad \forall(t) \qquad (3)$$

$$Pr\left[\bigwedge_j \bigwedge_t \mathbf{x}_t \notin \mathbb{O}_j^n\right] \geq 1 - \Delta \qquad (4)$$

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).

$$\mathbf{x}_t := [p_t^x \ p_t^y \ v_t \ \alpha_t]^T, \quad \mathbf{u}_t := [a_t \ \varepsilon_t]^T \qquad (5)$$

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.

$$F_{\Psi_b}(\mathbf{x}_t, \mathbf{u}_t) = \begin{bmatrix} p_t^x + v_t \cdot cos(\alpha_t) \cdot \Delta t + a_t \cdot cos(\alpha_t) \cdot (\Delta t)^2/2 \\ p_t^y + v_t \cdot sin(\alpha_t) \cdot \Delta t + a_t \cdot sin(\alpha_t) \cdot (\Delta t)^2/2 \\ v_t + a_t \cdot \Delta t - \frac{F_t^d}{m} \cdot \Delta t \\ \alpha_t + \varepsilon_t \cdot \Delta t \end{bmatrix} \qquad (6)$$

$$F_{\Psi_m}(\mathbf{x}_t, \mathbf{u}_t) = \begin{bmatrix} p_t^x + v_t \cdot cos(\alpha_t) \cdot \Delta t \\ p_t^y + v_t \cdot sin(\alpha_t) \cdot \Delta t \\ v_t - \frac{F_t^d}{m} \cdot \Delta t \\ \alpha_t + \varepsilon_t \cdot \Delta t \end{bmatrix} \qquad (7)$$

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

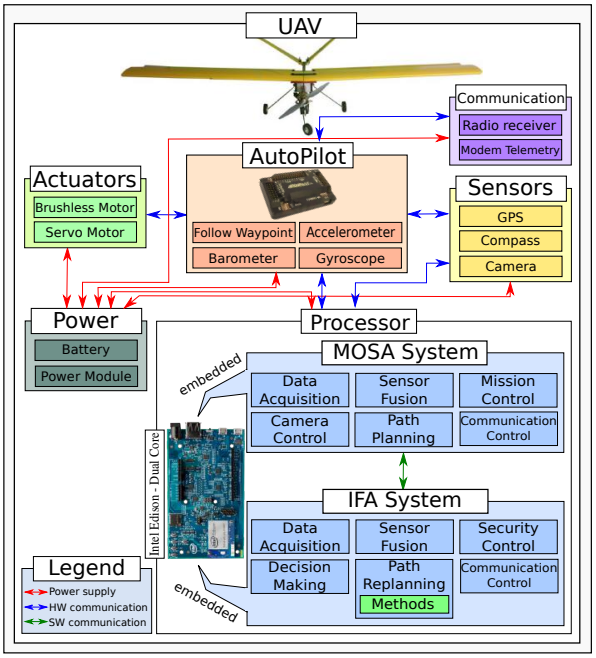


Fig. 3. Embedded system architecture in the aircraft.

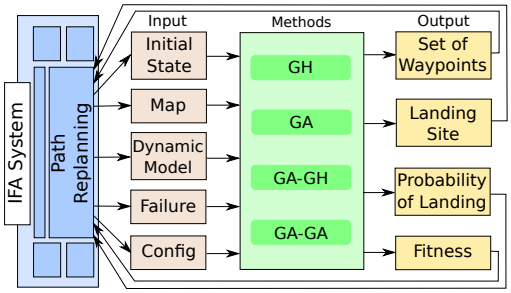


Fig. 4. Embedded system architecture in the path re-planning module.

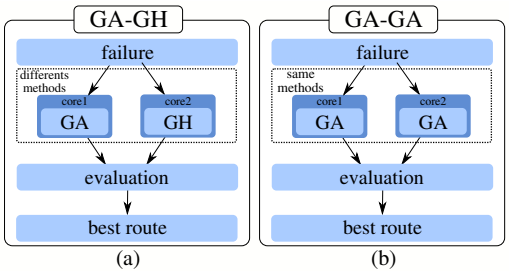


Fig. 5. Implemented strategies running methods in parallel. (a) Combining of GA e GH. (b) Two executions of GA.

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 (HG), Genetic Algorithm (GA), combination of GA executing in parallel (GA-GA) and combination of GA with HG executing in parallel (GA-HG). 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 HG 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.

```
1  begin
2      EmergencyRoute vectorRoutes[numIndividuals];
3      for i = 1 to numIndividuals do
4          EmergencyRoute route;
5          initialize(route, map);
6          evaluate(route, map);
7          vectorRoutes.add(route);
8      repeat
9          repeat
10             for i = 1 to crossRate × numIndividuals do
11                 EmergencyRoute ind1, ind2;
12                 select(ind1, ind2);
13                 EmergencyRoute child ← crossover(ind1, ind2);
14                 mutation(child);
15                 evaluate(child, map);
16                 vectorRoutes.add(child);
17         until converge(vectorRoutes);
18         restart(vectorRoutes);
19     until reach(stoppingCriterion);
20     EmergencyRoute bestRoute ← getBestRoute(vectorRoutes);
21     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.

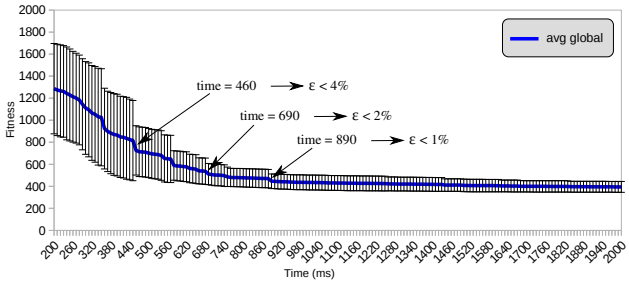


Fig. 6. Time analysis for convergence of the GA.

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	
	HG	-		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 HG, 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	HG	-	D			
	GA	250	D	C		
	GA-GA	250	D	C	B	
	GA-HG	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

C

AVALIAÇÃO DE ESTRATÉGIAS PARA POUSO EMERGENCIAL DE VANTS

Artigo completo publicado em revista *International Journal on Artificial Intelligence Tools* (IJAIT) em 2017, com qualis B1 em ciência da computação. Esse trabalho é uma extensão para revista de estudos iniciados em meu mestrado, em que três estratégias de pouso emergencial para VANT são apresentadas e comparadas usando testes estatísticos. Dessa forma, esse artigo é um resultado indireto do doutorado.

International Journal on Artificial Intelligence Tools
Vol. 26, No. 1 (2017) 1760008 (30 pages)

DOI: 10.1142/S0218213017600089

World Scientific
www.worldscientific.com

Heuristic and Genetic Algorithm Approaches for UAV Path Planning under Critical Situation

Jesimar da Silva Arantes*, Márcio da Silva Arantes†, Claudio Fabiano Motta Toledo‡ and Onofre Trindade Júnior§

University of São Paulo, USP, São Carlos, São Paulo, Brazil
**jesimar.arantes@usp.br*
†*marcio@icmc.usp.br*
‡*claudio@icmc.usp.br*
§*otj@icmc.usp.br*

Brian Charles Williams

Massachusetts Institute of Technology, MIT, Cambridge, USA
williams@mit.edu

Received 22 June 2016
Accepted 15 December 2016
Published 23 February 2017

The present paper applies a heuristic and genetic algorithms approaches to the path planning problem for Unmanned Aerial Vehicles (UAVs), during an emergency landing, without putting at risk people and properties. The path re-planning can be caused by critical situations such as equipment failures or extreme environmental events, which lead the current UAV mission to be aborted by executing an emergency landing. This path planning problem is introduced through a mathematical formulation, where all problem constraints are properly described. Planner algorithms must define a new path to land the UAV following problem constraints. Three path planning approaches are introduced: greedy heuristic, genetic algorithm and multi-population genetic algorithm. The greedy heuristic aims at quickly find feasible paths, while the genetic algorithms are able to return better quality solutions within a reasonable computational time. These methods are evaluated over a large set of scenarios with different levels of difficulty. Simulations are also conducted by using FlightGear simulator, where the UAV's behaviour is evaluated for different wind velocities and wind directions. Statistical analysis reveal that combining the greedy heuristic with the genetic algorithms is a good strategy for this problem.

Keywords: Genetic algorithms; unmanned aerial vehicles; path planning; risk allocation; uncertainty.

‡Corresponding author

1. Introduction

The present paper describes heuristics and metaheuristics applied to the path planning problem for Unmanned Aerial Vehicles (UAVs) under the occurrence of critical situations. The methods and results presented extend preliminary ones recently reported in Ref. 1.

The commercial application of UAVs and risks related to their operation lead to discussions about flight safety for this aircraft type. Safety is the most important factor to ensure the integration of UAVs to airspace and one relevant aspect is to bring back on board features lost by the absence of pilots.[14] A possible security measure to be considered is that the probability of failure for UAVs should be less than or equal to the currently accepted one in general aviation.[7]

The path planning problem under critical situation here approached aims to define autonomously a route that lands the UAV without risk to the safety of people, properties and aircraft itself. We assume an aircraft running a mission that needs to be aborted and re-planned by the proposed path planning methods. The mission is aborted in case of a critical situation due to failures in the equipments of the UAV or extreme environmental conditions.

Equipment failures can be caused by problems like battery overheating, motor crash, sensors failure, among others. Extreme environmental situations can be the approximation of a storm or another aircraft, the occurrence of strong turbulence or unforeseen storm, among others. In these cases, the UAV can return base or execute a hard landing.

This paper will consider critical situations caused only by equipment failures demanding a hard landing. In this context, a near-optimal solutions become more relevant than time-consuming optimal solutions which favors the use of heuristic and metaheuristic approaches. Evolutionary techniques have been successfully applied in real world problems.[17] Thus, a greedy heuristic and two evolutionary algorithms are applied to return emergency routes.

To summarize, the main contribution of this paper is to introduce the problem of path re-planning under an emergency landing for UAVs and to propose fast methods to solve it. A mathematical formulation is introduced aiming to describe such problem. First, a Greedy Heuristic (GH) is developed to build fast feasible paths. Next, a Genetic Algorithm (GA) and a Multi-Population Genetic Algorithm (MPGA) are also applied as path planner. The performance of GH is evaluated by itself and as an initialization operator for candidate solution used during the evolve process by GA and MPGA.

The paper differs from Ref. 1 since we increase the results by introducing comparisons of MPGA against GA. Also, a more elaborated statistical analysis is done to evaluate the methods performance. Finally, a series of new experiments involving FlightGear (FG) simulator is added, where different speeds and direction for winds are tested.

The paper is organized as follows: section 2 describes some related works and section 3 defines the problem approached. The proposed methods are presented in

section 4 and computational results achieved are reported in section 5. Finally, the last section 6 summarizes concluding remarks and interesting points.

2. Related Work

Path planning under critical situations for manned aircraft is approached by Refs. 16 and 15, where a system helps pilots to determine the best landing site for a damaged aircraft. After the critical failure of the aircraft, the pilot must first recover control of the aircraft and the goal is to find the best place for the emergency landing. The Emergency Landing Planner (ELP) system introduced in Ref. 16 evaluates possible landing sites for the aircraft using an A* graph search algorithm. The same algorithm is analyzed in Ref. 15 by applying it to real scenarios. The authors conducted tests with a flight simulator for large aircraft, where the ELP system is triggered to assist pilots for critical situations happening between 1 to 3 minutes of flight.

The studies conducted in our paper is closer to those described in Ref. 16, but we are planning emergency landing for autonomous UAVs, while the authors in Ref. 16 are dealing with manned aircraft. Also, we consider a different set of associated failures, which demand an emergency aircraft landing. The aircraft in Ref. 16 is still able to flight even during failure.

We are planning path for autonomous UAV taking into account the uncertainty from the environment and the risk of collision with obstacles. The authors in Refs. 13, 4 and 19 study the mission planning for autonomous vehicles. A sequence of discrete actions and continuous controls for air and underwater autonomous vehicles is presented in Ref. 13, where a system is able to plan those events is introduced. The risks and uncertainty during the path planning for UAV are evaluated in Ref. 4, where a certain margin of safety must be satisfy for the risk of collision with obstacles incurred during the path planning. A new planner system is proposed for mission planning in Ref. 19. This system introduces improvement in the treatment of risk in Ref. 4 and integrates scheduling for the tasks to be executed.

The inaccuracies of the equipment and the environment lead to risks when executing a route previously planned. The authors in Refs. 13, 4 and 19 present advances since they integrate risk allocation during the mission planning for autonomous vehicles. This allows to ensure that the planned mission is executed within a safety margin. However, they do not consider path re-planning under critical situations from the original path of an aborted mission. The present paper introduces a mathematical formulation for this path re-planning problem based on that described in Ref. 4 for path planning with risk allocation. Also, we are including risk allocation when planning a path under critical situation.

This paper proposes the use of GA and MPGA as path planners. GA with Voronoi diagram are applied by Ref. 20 for path planning of UAVs. A new mutation strategy is proposed that works with global and local random diversity. The initial phase of the method applies Voronoi diagram to create the population. GA is

applied in Ref. 23 to path planning for land mobile robots, where the environment to be traversed is represented by a grid. A new mutation operator is proposed that avoids premature convergence and helps to find optimal paths.

Different methods are compared in Ref. 3 as path planners using a metric proposed by the authors. This metric takes into account the complexity and peculiarities of the problem handled. The methods evaluated are GA, Particle Swarm Optimization (PSO) and Differential Evolution (DE). Based on the proposed metric, GA is the best method followed by PSO and DE. Path planning for UAV in three-dimensional environments is studied in Ref. 25, where the aircraft has to avoid no-fly zones, radars zones, missiles and anti-aircraft guns. The authors developed a DE method to define paths.

DE with constrained sampling is used in Ref. 24 for search and rescue in real time scenarios with UAVs. A set of autonomous UAVs is used for rescue task within a large area. The results are compared with a strategy based on Swarm Intelligence (SI). The evolutionary approach proved to be better than the SI, once SI decreased performance when the number of rescue targets increased.

MPGA evolves several populations trying to explore different regions from the search space. The individuals are hierarchically structured in trees in each population, where the best individual is the root. The method was introduced by Ref. 11 with individuals structured in ternary trees following a hierarchy based on their fitness value. Such approach has been applied to solve problems in different contexts like glass container production,[22] ordering microarray data[18] and asymmetric traveling salesman problem[6] with relevant results reported.

3. Path Planning Problem under Critical Situation

The path planning problem under critical situation is described in this section, following two steps. First, it is provided a problem overview by an example of UAV mission where a critical situation happens. This example will help to clarify all constraints assumed. Next, a general mathematical formulation is introduced with all constraints properly defined.

3.1. *Problem overview*

Figure 1 illustrates a UAV mission where a critical situation occurs. There are UAVs runway, storm area, airport, plain area, two populated regions (houses), woods and a scenic region.

Let us assume that the UAV mission starts from its runway, reaching the scenic region next, remaining on this region to take pictures for a while and flying back to the runway. During such mission, the UAV must avoid no-fly zones such as storms and airports. The aircraft can fly over populated and wood areas. However, throughout the mission accomplishment, the UAV embedded systems detect a problem, e.g., battery overheating, and the current mission is aborted and the UAV systems trigger the algorithm to re-plan the current path. The re-planning

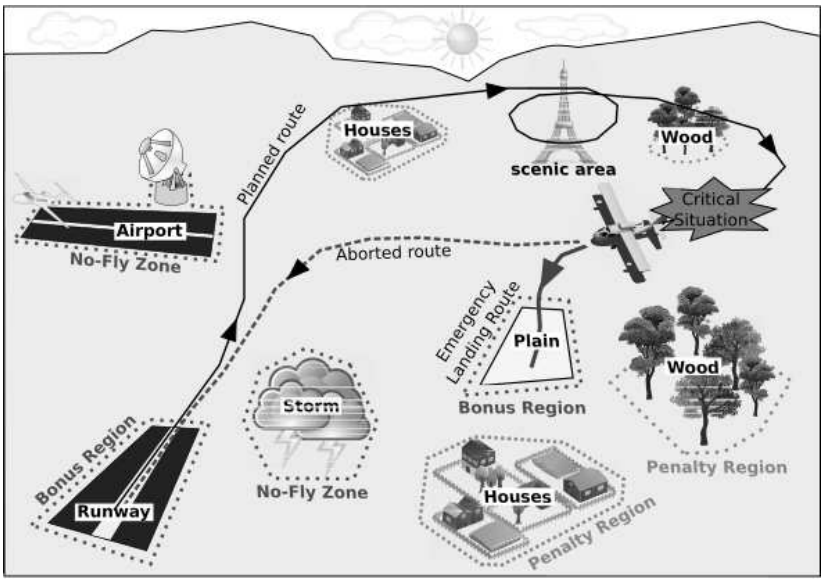


Fig. 1. Illustrative scenario for mission planning under critical situation.

will now minimize the possibility of damage during landing, considering information from the mentioned regions in Figure 1 and limitations caused by the aircraft's problem.

It is assumed that information from several regions can be mapped before the mission execution. It becomes possible to define disjoint sets of regions based on the probability of landing the aircraft in one of these regions. In the present paper, a total of four sets of regions are considered:

(1) **No-Fly Set (ϕ_n):** Regions on this set are represented by airports, military base and other areas with restrictions on the UAV flight. These regions are also called No-Fly Zone. Thus, it is forbidden for the UAV to fly over and land in regions of this set.
(2) **Navigable with Penalty Set (ϕ_p):** Regions on this set may represent populated regions, factories, forests, among others. UAV can fly over these regions, but it is not desired landing on them. Such regions are also called as penalty region.
(3) **Navigable and Bonus Set (ϕ_b):** Regions on this set contain flat and suitable regions for landing as grassy areas or fields with grounder plantations. UAV can fly over and it is possible to land on regions of this set. Thus, the regions on this set are called bonus regions.
(4) **Remainder Set (ϕ_r):** Regions on this set represent remain areas that are not classified for landing. There are no restrictions for flight or landing in these regions, but there is not enough information to classify them on set ϕ_b. The regions on this set are called remainder region.

The limitations caused by the aircraft's problem must be sent to the re-planning algorithm. This is done by the UAV system when the critical situation is detected.

The critical situations to aircraft operation considered by this paper are defined next:

(1) **Motor Failure (ψ_m):** The UAV engine m presents problems and stops functioning, and a region for landing has to be found while the aircraft is hovering.
(2) **Battery Failure (ψ_b):** The battery b presents an overheating, where all controls work, but the UAV should land as soon as possible.
(3) **Aerodynamic Surfaces Failure Type 1 (ψ_{s1}):** One of the wings $s1$ fails making the aircraft able only to turn left.
(4) **Aerodynamic Surfaces Failure Type 2 (ψ_{s2}):** One of the wings $s2$ fails making the aircraft able only to turn right.

3.2. *Problem modeling*

A mathematical formulation is introduced next to summarize all aspects approached for the path planning problem under critical situation. The proposed formulation extends a similar modelling described in Ref. 4 for path planning with risk allocation.

Parameters:

- $\phi_j = \{Z^1_{\phi_j}, Z^2_{\phi_j}, \ldots, Z^{|\phi_j|}_{\phi_j}\}$: Set of regions with $j \in \{n, p, b, r\}$;
- $Z^i_{\phi_j}$: ith region of the set ϕ_j;
- $|\phi_j|$: Number of regions in set ϕ_j;
- C_{ϕ_j}: Cost of landing in set ϕ_j;
- T: Number of time steps to land the UAV;
- Δ: Probability of UAV violate a region in the set ϕ_n;
- F_Ψ: State transition function for a given type of failure Ψ_k with $k \in \{m, b, s1, s2\}$;
- ΔT: Time discretization established in the simulation;
- ω_t: State-independent disturbance at time step t;
- Q: Covariance matrix 4×4 associated with environmental uncertainty;
- Σ_t: Covariance matrix 4×4 associated with the uncertainty of the UAV.

Decision variables:

- x_t: Set of states of the UAV ($x_t = [p^x_t, p^y_t, v_t, \alpha_t]^T$);
- p^x_t: Position in the x-axis of the UAV at time step t;
- p^y_t: Position in the y-axis of the UAV at time step t;
- v_t: Velocity at time step t of the aircraft;
- α_t: Angle of the UAV in time step t from x-axis;
- u_t: Set of UAV controls ($u_t = [a_t, \varepsilon_t]^T$);
- a_t: Acceleration of the UAV at time step t;
- ε_t: Angular variation of the UAV.

$$Minimize \sum_{i=1}^{|\phi_p|} (C_{\phi_p} \cdot P(x_T \in Z^i_{\phi_p})) - \sum_{i=1}^{|\phi_b|} (C_{\phi_b} \cdot P(x_T \in Z^i_{\phi_b})) \quad (1)$$

subject to:

$$x_{t+1} = F_\Psi(x_t, u_t) + \omega_t \qquad \forall\ t = 0, 1, \ldots T \tag{2}$$

$$\omega_t \sim \mathcal{N}(0, Q) \qquad \forall\ t = 0, 1, \ldots T \tag{3}$$

$$x_t \sim \mathcal{N}(\overline{x}_t, \Sigma_t) \qquad \forall\ t = 0, 1, \ldots T \tag{4}$$

$$P\left(\bigwedge_{t=0}^{T} \bigwedge_{i=1}^{|\phi_n|} x_t \notin Z^i_{\phi_n}\right) \geq 1 - \Delta \tag{5}$$

$$\Omega = \{(p^x_0, p^y_0), (p^x_1, p^y_1), (p^x_2, p^y_2), \ldots (p^x_T, p^y_T)\} \tag{6}$$

The decision variable x_t represents the state of UAV at time step $t = 1, 2, \ldots, T$. The UAV state is defined by its position (p^x_t, p^y_t), velocity (v_t) and angular direction (α_t) at Cartesian plane. Another decision variable, u_t, defines controls to be applied at each time step t given by acceleration (a_t) and angular variation (ε_t).

The fitness function (1) minimizes penalties and maximizes rewards. The penalties C_{ϕ_p} are given by a route landing the aircraft at the last time step $t = T$ on penalty set regions $x_T \in Z^i_{\phi_p}$. On the other hand, rewards C_{ϕ_b} are given for routes landing the aircraft on safety regions $x_T \in Z^i_{\phi_b}$. The role of function $P(\ldots)$ is to return the aircraft probability of being or not in a particular region ϕ_i for each state x_t.

Controls applied at t and the uncertainty ω_t from external factors define the dynamic of the UAV states at the next time step $t+1$ following the transition state constraint (2). Function F_Ψ is defined by this work based on the type of critical situation addressed (Ψ). We assume that the uncertainty ω_t follows a Gaussian white noise distribution constraints (3) with covariance matrix Q. Gaussian distribution for the initial position with mean $\hat{x}_0$ and covariance matrix Σ_{x_0}, $x_0 \sim \mathcal{N}(\hat{x}_0, \Sigma_{x_0})$ is also assumed. These assumptions are also made by Ref. 4, thus, future states follow a Gaussian distribution and x_t becomes a random variable by constraints (4). Moreover, UAV location is not precise with the risk of the UAV deviates from its route by reaching non-fly areas. A bound for such risk can be considered during the path planning, and constraints (5) describe the probability $(1 - \Delta)$ of the UAV being out of regions that belong to set ϕ_n. Equation (6) has the set of waypoints belonging to the final landing trajectory.

4. Methods

A heuristic and two methods, based on genetic algorithms, are described here, all of them share a same encoding (representation of solution), fitness function and genetic operators. Thus, this section begins by explaining such common aspects before describes the methods from their pseudocodes.

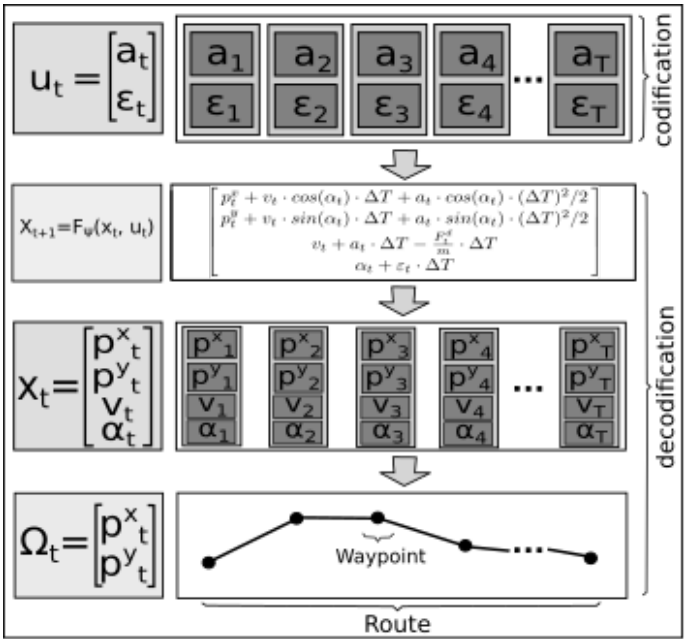


Fig. 2. Codification and decodification of the controls on the route of the aircraft.

4.1. *Representation of solutions and operators*

The real values for the controls $u_t = [a_t, \varepsilon_t]'$ applied to the aircraft at each instant t are encoded in the representation of solution. Figure 2 illustrates a solution that encodes controls from (a_1, ε_1) to (a_T, ε_T). The solutions are created applying four initialization operators:

- **random**: generates values with uniform distribution for $a_t \in U[a_{min}; a_{max}]$ and $\varepsilon_t \in U[\varepsilon_{min}; \varepsilon_{max}]$;
- **short curves**: defines values for $a_t \in U[a_{min}; a_{max}]$, but it draws reduced values for angular variation with $\varepsilon_t \in U[\frac{1}{2}\varepsilon_{min}; \frac{1}{2}\varepsilon_{max}]$;
- **short acceleration**:takes $\varepsilon_t \in U[\varepsilon_{min}; \varepsilon_{max}]$ and $a_t \in U[\frac{1}{2}a_{min}; \frac{1}{2}a_{max}]$;
- **greedy**: applies the greedy heuristic, described in the next section, generating controls to reach bonus regions.

The operators of initialization short curves and short acceleration allow to apply smoother controls when changing the direction and acceleration values of the aircraft. The crossover operators average, geometric, arithmetic, order crossover (OX) and blend crossover (BLX-α) were used as described in Refs. 10, 17 and 12 for real-code. Uniform, limit and creep were the real-code mutation operators applied as described in Refs. 17 and 9.

The operators were previously evaluated whose results defined as a best strategy to apply all of them. This means randomly select one among the five crossovers to be applied every time a new individual is created. A similar procedure occurs with mutation operators, where one among the three mutations is also randomly selected each time a mutation happens.

4.2. *Decoding and fitness function*

Figure 2 shows that the solution is encoded as controls $u_t = [a_t, \varepsilon_t]'$ which are decoded next as a route Ω_t by the transition function $F_\Psi(x_t, u_t)$ in Equation (7). The motion of a fixed-wing UAV acting over a 2-D space is approximated by $F_\Psi(x_t, u_t)$ with F_t^d and m being, respectively, resistance of the air and mass of the UAV.

$$F_\Psi(x_t, u_t) = \begin{bmatrix} p_t^x + v_t \cdot cos(\alpha_t) \cdot \Delta T + a_t \cdot cos(\alpha_t) \cdot (\Delta T)^2/2 \\ p_t^y + v_t \cdot sin(\alpha_t) \cdot \Delta T + a_t \cdot sin(\alpha_t) \cdot (\Delta T)^2/2 \\ v_t + a_t \cdot \Delta T - \dfrac{F_t^d}{m} \cdot \Delta T \\ \alpha_t + \varepsilon_t \cdot \Delta T \end{bmatrix} \tag{7}$$

All states x_t for $t = 1, \ldots, T$ will be determined by $F_\Psi(x_t, u_t)$, where the next expected state $\overline{x}_{t+1}$ depends from the current expected state $\overline{x}_t$ and nominal controls applied ($\overline{u}_t$). The state x_t is a random variable with $x_t \sim \mathcal{N}(\overline{x}_t, \Sigma_t)$.[4] Thus, the transition function can be used to calculate all the next expected states $\overline{x}_{t+1} = F_\Psi(\overline{x}_t, \overline{u}_t))$, but the uncertainty Σ_t in the expected state $\overline{x}_t$ grows at each time step. A close-loop control approach is proposed in Ref. 19 defining control inputs from a nominal control input $\overline{u}_t$, where a correction that slows down the growth of the uncertainty Σ_t is applied.

We assume a constant uncertainty Σ_t at any time t with $\Sigma_t = Q$. It is possible to estimate the probability function $P(x_t \in Z_\phi^i)$ by using a lookup table for a Gaussian distribution, if the uncertainty about the state x_t at any time step t, given by $x_t \sim \mathcal{N}(\overline{x}_t, \Sigma_t)$, is known. The covariance matrix used in this work is given by Equation (8) where $\sigma = 10$ meters is assumed.

$$\Sigma_t = Q = \begin{bmatrix} \sigma^2 & 0 & 0 & 0 \\ 0 & \sigma^2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8}$$

The critical situations are defined by changing Equation (7) as follows:

- Critical situation ψ_m: Acceleration is set null ($a_t = 0$) in F_{Ψ_m} since acceleration over the UAV is not possible during a motor failure.
- Critical situation ψ_{s1}: The angular variation becomes only positive ($\varepsilon_t \in [0, \varepsilon_{max}]$) for ψ_{s1}, when the UAV is able only to turn left.
- Critical situation ψ_{s2}: It is set $\varepsilon_t \in [\varepsilon_{min}; 0]$ when the UAV is able only to turn right.
- Critical situation ψ_b: There is no change in Equation (7), since the UAV with a battery failure has only to quickly land. In this case, Equation (16) is included into fitness function as explained latter.

Decoding procedure returns a set with all necessary waypoints to land the UAV by constraints (2) in section 3.2. The fitness function evaluates this set of waypoints as described by Equation (9).

$$fitness = f_{Land_{\phi_b}} + f_{Land_{\phi_p}} + f_{Flight_{\phi_n}} + f_{Curves} + f_{DistUAV_{\phi_b}} + f_{Violated_T} \quad (9)$$

The aircraft can land without spending all time steps available, so it can land at time step K such that $0 < K \leq T$. A trajectory like that is a possible solution.

Rewards and penalties for landing the UAV, respectively, in bonus and penalized regions are dealt with by Equations (10) and (11). However, fly over or landing on no-fly zones is a hard constraint violation, which is penalized following Equation (12).

$$f_{Land_{\phi_b}} = -C_{\phi_b} \cdot \sum_{i=1}^{|\phi_b|} (P(x_K \in Z^i_{\phi_b})) \quad (10)$$

$$f_{Land_{\phi_p}} = C_{\phi_p} \cdot \sum_{i=1}^{|\phi_p|} (P(x_K \in Z^i_{\phi_p})) \quad (11)$$

$$f_{Flight_{\phi_n}} = C_{\phi_n} \cdot max\left(0, 1 - \Delta - P\left(\bigwedge_{t=0}^{K} \bigwedge_{i=1}^{|\phi_n|} x_t \notin Z^i_{\phi_n}\right)\right) \quad (12)$$

It is also worth to avoid paths with unnecessary bend and taking long distances to reach bonus regions. These issues are considered, respectively, by Equations (13) and (14).

$$f_{Curves} = \frac{1}{|\varepsilon_{max}|} \cdot \sum_{t=0}^{K} |\varepsilon_t| \quad (13)$$

$$f_{DistUAV_{\phi_b}} = shortestDist(\overline{x}_K, \phi_b) \quad (14)$$

The aircraft can reach, e.g., a bonus region without actually landing on it. This happen when the aircraft has final velocity greater than its minimum value, so it is not landing in fact. Equation (15) will give preference for those paths where the UAV actually lands, besides it reaches the bonus region.

$$f_{Violated_T} = \begin{cases} C_{\phi_b}\,, & v_K - v_{min} > 0 \\ 0\,, & \text{otherwise} \end{cases} \quad (15)$$

Equation (16) is included to the fitness function when critical situation ψ_b happens. In this case, it becomes even more important to land the aircraft as soon as possible. Thus, the idea is to reduce the number of waypoints spent to land the aircraft.

$$f_{\psi} = \begin{cases} C_{\phi_b} \cdot 2^{\frac{(K-T)}{10}}\,, & \psi = \psi_b \\ 0\,, & \text{otherwise} \end{cases} \quad (16)$$

Algorithm 1: Greedy Heuristic

```
1 begin
2     RouteLanding route[] ← RouteLanding()[map.|φ_b|];
3     for i = 1 to map.|φ_b| do
4         initialize(route[i], map.Z^i_{φ_b});
5         evaluate(route[i]);
6     RouteLanding bestRoute ← getBestRoute(route);
7     return bestRoute;
8 end
```

4.3. *Proposed methods*

The first approach to solve the proposed problem was to define a heuristic fast enough to return feasible solutions. The main idea is to have a fast decision-making approach when a viable solution becomes more interesting than time-consuming better solutions. The greedy heuristic (GH) described by Algorithm 1 was then developed.

The method input is data from bonus regions $map.|\phi_b|$ in line 2 besides other problems inputs. Candidate solutions for each bonus regions are generated and evaluated from lines 3–5. A candidate solution is created by rotating the UAV until it reaches the same direction of a bonus region ($map.Z^i_{\phi_b}$ in line 4). Next, it is calculated one trajectory straight to this region. Figure 3 illustrates this rotation procedure by angles $\lambda_1 < \lambda'_1$ for region b_1, where the rotation occurs towards the angle λ_1. UAV is rotated always to the side with lower angle between the UAV and a straight line to a bonus region.

The best solution in Figure 3 lands the UAV in region b_3 that is the closest one. Another candidate solution that lands on b_1 is slightly worse and the solution that lands on b_2 violates the navigability constraint, so it presents low quality. All these possible solutions are evaluated by Equation (9).

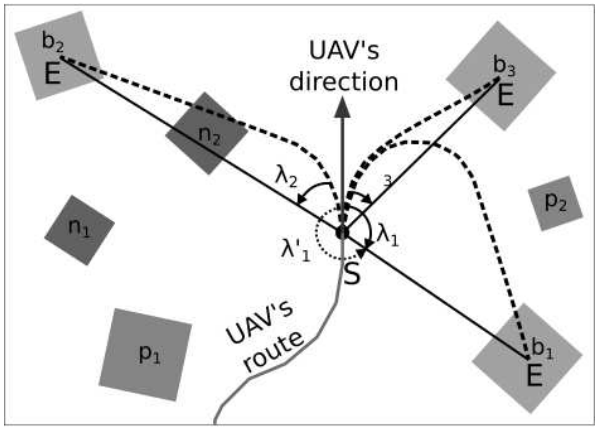


Fig. 3. Candidate routes using greedy heuristic.

Algorithm 2: Genetic Algorithm

```
1  begin
2      createPopulation(routes);
3      initialize(routes);
4      evaluate(routes, map);
5      repeat
6          repeat
7              for i = 1 to rateCross × numIndividuals do
8                  select(parents);
9                  child ← crossover(parents);
10                 mutation(child);
11                 evaluate(child, map);
12                 add(child);
13         until converge(routes) ;
14         restart(routes);
15     until reach(stoppingCriterion) ;
16     RouteLanding bestRoute ← getBestRoute(routes);
17     return bestRoute;
18 end
```

It was mentioned that GH can act as an initialization operator for GA and MPGA. In this case, a trajectory is built as descried, but for a bonus region randomly selected. Moreover, the controls defined for this solution will be an individual in the GA or MPGA.

The other two proposed methods are based on genetic algorithms. While GH is a determinist approach for this problem, the next methods are non-deterministic, i.e., they can return different solutions from different executions over the same problem instance. Algorithm 2 describes the first method named simply as Genetic Algorithm (GA).

This approach comes from standard genetic algorithm approaches. First, a population of routes is created and evaluated by Equation (9). Next, the evolutionary process occurs while there is no convergence. A total of $rateCross \times numIndividuals$ individuals are generated through crossover and mutation operators. The selection for reproduction applies a tournament operator. After crossover and mutation operators have been applied, the new individual, if better, replaces its parent with worst fitness. A convergence is assumed when no new individual is inserted into the population after $rateCross \times numIndividuals$ attempts. If there is convergence, the population is re-initialized keeping only the best individual found so far. This process is repeated by a previously defined number of fitness evaluations (stopping criterion). Algorithm 3 describes a multi-population genetic algorithm (MPGA) with individuals hierarchically structured in trees.

The population is first initialized, all individuals are evaluated and structured next in tree through lines 4–7. Figure 4 shows the individuals hierarchically structured in tree. The position of the individuals (nodes) reveals the hierarchy, where

Algorithm 3: Multi-Population Genetic Algorithm

```
1  begin
2      repeat
3          for i = 1 to numPop do
4              for j = 1 to numIndividuals do
5                  initialize(pop(i).ind(j));
6                  evaluate(pop(i).ind(j));
7              organize(pop(i));
8              repeat
9                  for j = 1 to rateCross × numIndividuals do
10                     select(parents);
11                     child ← crossover(parents);
12                     mutation(child);
13                     evaluate(child);
14                     add(child, pop(i));
15                 organize(pop(i));
16             until converge(pop(i)) ;
17         for i = 1 to numPop do
18             migrate(pop(i));
19     until reach(stoppingCriterion) ;
20     RouteLanding bestRoute ← getBestRoute(pop);
21     return bestRoute;
22 end
```

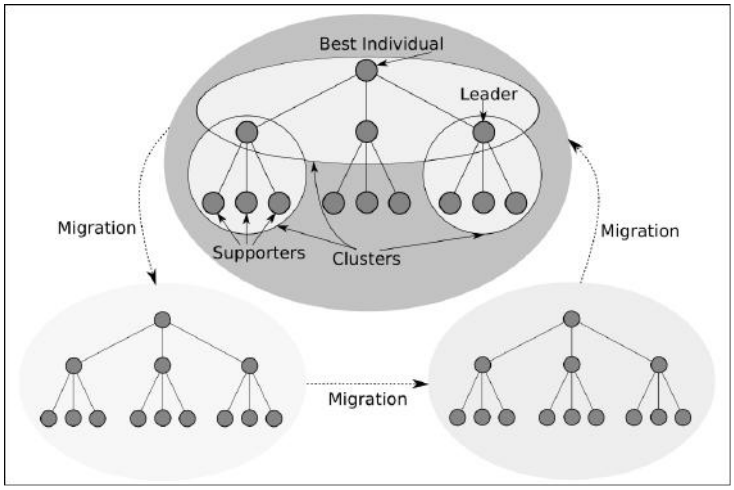


Fig. 4. MPGA organization using a hierarchical tree structure.

the leader has better fitness than its followers in each cluster. Moreover, the best individual is the root node and the worst individuals are the leaf nodes.

The evolutionary process happens on lines 10–14. First, the selection for reproduction always chooses randomly two individuals: a leader node and one of its followers. The new individual created after crossover and mutation replaces the

worst parent, if better.The population is hierarchically rearranged, after new individuals insertion, such that better individuals become leader in their clusters. If no individuals are inserted after $rateCross \times numIndividuals$ attempts, the population converges. After all populations have converged, a migration operator sends the best individual from population i to population $i+1$. Next, the populations are re-initialized, except by the best individual of each population and the migrated individual. MPGA is executed until the number of fitness evaluations has been reached (stopping criterion).

The main difference between GA and MPGA implementation is how solutions (individuals) are structured. MPGA has individuals belonging to multi-populations, where the individuals in each population are hierarchically structured as previously described. The multi-population approach presents the island effect that can improve the parallel search through the solution space.[9] The tree hierarchy impacts over how individuals are selected to reproduce as well as how the insertion of new individuals takes place, since there is an evolution pressure related to each cluster (Figure 4). The authors in Refs. 11, 18 and 21 reported better results applying MPGA approaches against classical GAs.

5. Computational Results

5.1. *Maps and problem parameters*

A map generator was developed taking into account the proportion of regions belonging to bonus set ϕ_b and map density. The aim is to create maps with different scenarios randomly generated to validate the landing path planners. The generator described is based on one introduced by Ref. 4 to create different maps.

The maps generated present dimensions 1000 m $\times$ 1000 m with critical situation happening always at position $(p_0^x, p_0^y) = (0; 0)$. The proportion of regions belonging to set ϕ_b defines three levels of difficulty:

- M_E — *Easy Maps*: There is many bonus regions, a median level of penalty (ϕ_p) and few no-fly (ϕ_n) regions.
- M_N — *Normal Map*: There is a balanced number of regions belonging sets ϕ_n, ϕ_p and ϕ_b.
- M_H — *Hard Map*: There is many regions in set ϕ_n, a median amount in set ϕ_p and few regions in ϕ_b.

The map density is used o define two levels of difficulty:

- $C_{25\%}$ — *Coverage 25%*: The regions spread out more sparsely across the map occupying around 25% of the total area of the map.
- $C_{50\%}$ — *Coverage 50%*: The regions will occupy around 50% of the total area.

Moreover, it is possible to define six group of instances (I1-I6) from these criteria with 100 maps generated for each group. Figure 5 illustrates one map for groups I1-I6.

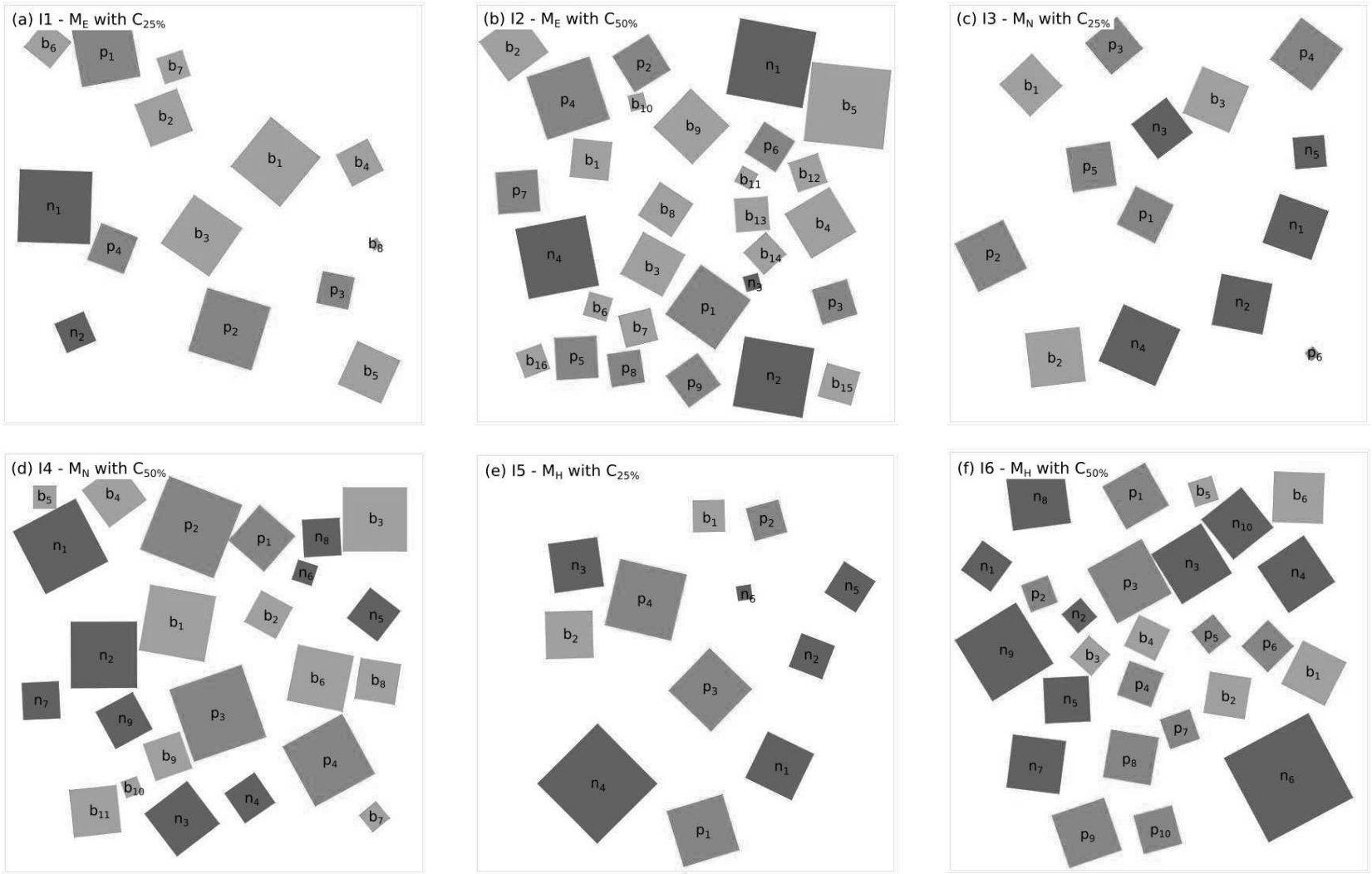


Fig. 5. Automatically generated maps. (a) Instance I1 easy map with coverage 25%. (b) Instance I2 easy map with coverage 50%. (c) Instance I3 normal map with coverage 25%. (d) Instance I4 normal map with coverage 50%. (e) Instance I5 hard map with coverage 25%. (f) Instance I6 hard map with coverage 50%.

Table 1. Settings of the UAV and weights of penalties used in the experiments.

Category	Parameter Description	Symbol	Value	Unit
Map	Dimension on the X axis	–	1000	meters (m)
	Dimension on the Y axis	–	1000	meters (m)
UAV	Starting position	(p_0^x, p_0^y)	$(0, 0)$	meters (m)
	Initial velocity	v_0	24	m/s
	Initial angle	α_0	90	degree ($^\circ$)
	Velocity minimum	v_{min}	11.1	m/s
	Velocity maximum	v_{max}	30.5	m/s
	Angular velocity minimum	ε_{min}	-3	$^\circ/s$
	Angular velocity maximum	ε_{max}	3	$^\circ/s$
	Acceleration minimum	a_{min}	0.0	m/s^2
	Acceleration maximum	a_{max}	2.0	m/s^2
Model	Number of time steps	T	60	seconds (s)
	Time discretization	ΔT	1	seconds (s)
	Acceptable risk of collide with region ϕ_n	Δ	0.001	–
Weights	Cost bonus region	C_{ϕ_b}	2000	$
	Cost penalty region	C_{ϕ_p}	8000	$
	Cost no-fly zone	C_{ϕ_n}	100 000	$
	Cost remainder region	C_{ϕ_r}	0	$

The maps have a set of regions defined by convex polygons, where non-convex regions are composed by two or more convex polygons. The map generator creates a set of polygons based on rotated squares using a similar approach adopted by Refs. 4, 13, 19 and 2 when dealing with non-convex regions.

Table 1 shows problem parameters. The UAV parameters are based on real data of the UAV model Tiriba.[5] It is provided the UAV velocity when the critical situation happens and values for minimum and maximum acceleration. There are other parameters related to problem and cost of land in different regions. The experiments were performed with machines under Linux-Ubuntu 13.10, Intel Core i5 processor, 64 bits, 1.80 GHz and 4 GB RAM.

A previous parameter tuning for the MPGA are available at web.[a] From these results, MPGA is set with 3 populations, where each one has 13 individuals disposed in ternary trees in a total of 39 individuals. The evolutionary process applies 0.5 and 0.75 of crossover and mutation rates, respectively. In despite of a 75% mutation rate, only a single control (acceleration and angular variation) from a total of $T = 60$ will change. The stop criterion is 10 000 fitness evaluations.

5.2. *Comparing GH, GAs and MPGAs*

Two versions of GA and MPGA are evaluated, one without applying GH as an initialize operator, named GA1 and MPGA1, and another applying it as initialize

[a]https://drive.google.com/file/d/0B1_zUKNiVRefRnZBZUNYMnFad2M/view

Table 2. Chance to save the UAV without putting risk on people, properties or itself.

Ψ	Inst.	Landing in Bonus Region (ϕ_b)				
		GH	GA1	MPGA1	GA2	MPGA2
Motor (ψ_m)	I1	79	81	81	**90**	**90**
	I2	92	88	92	**96**	**96**
	I3	58	59	60	**71**	**71**
	I4	86	81	84	95	**96**
	I5	30	34	36	**40**	**40**
	I6	62	59	60	81	**82**
	Avg	67.8	67.0	68.8	78.8	**79.2**
Battery (ψ_b)	I1	99	96	**100**	**100**	**100**
	I2	97	**99**	**99**	**99**	**99**
	I3	93	91	94	98	**99**
	I4	98	96	99	99	**100**
	I5	67	68	73	**95**	94
	I6	83	62	68	**96**	95
	Avg	89.5	85.3	88.8	**97.8**	**97.8**
s1 (ψ_{s1})	I1	81	91	90	**92**	91
	I2	88	88	89	**95**	93
	I3	68	77	76	**86**	**86**
	I4	82	81	84	**90**	89
	I5	41	49	49	73	67
	I6	56	39	46	**78**	**78**
	Avg	69.3	70.8	72.3	**85.7**	84.0
s2 (ψ_{s2})	I1	90	93	94	**99**	**99**
	I2	90	94	**95**	94	**95**
	I3	70	78	79	**92**	**92**
	I4	87	83	83	**94**	**94**
	I5	40	65	62	**78**	74
	I6	61	49	57	**77**	76
	Avg	73.0	77.0	78.3	**89.0**	88.3
Global Average		74.9	75.0	77.1	**87.8**	87.3

operator, named GA2 and MPGA2. Table 2 shows each type of critical situation associated with the UAV (Ψ), the level of difficulty given by each instance group (I1-I6) and the number of times the UAV landed in bonus regions (ϕ_b). The best results for each test case is highlighted.

Let us give an example from the results reported for motor problem (ψ_m) in instances I1 (*Easy Maps* with *Coverage 25%*). GH is able to land the UAV for 79 out of 100 maps in bonus regions. GA1, MPGA1, GA2 and MPGA2 return, respectively, 81, 81, 90 and 90 routes that land the UAV in a bonus area for I1 with motor problem.

The overall results show that GA2 and MPGA2 reached better and similar results. They landed the UAV in bonus area for 87.8% and 87.3% of the maps on average against 74.9%, 75.0% and 77.1% of GH, GA1 and MPGA1, respectively.

All methods present better performance dealing with battery overheating problem, which is expected since battery overheating problem is the only one that doesn't change the flight capacity or maneuvers of the UAV. For such critical situation, GA2 and MPGA2 landed the aircraft safely in 97.8% of maps, while GH returned 89.5%, GA1 85.3% and MPGA1 88.8%. GA2 and MPGA2 are the only approaches able to land the aircraft in bonus areas for more than 80% of maps when flaws in both wings occur. GH, GA1 and MPGA1 presents the worst performance for the majority of results with similar results.

The performance of GA1 and MPGA1 is not much better than GH for all critical situations evaluated. This means that evolving paths, without some previously routes to bonus regions, is not a better strategy. In the other hand, the combined use of GH with GA and MPGA is able to save the UAV without put in risk people or properties on more than 87% of the cases. However, hit people or damage properties must be avoided even in critical situations within the maximum possible accuracy. Table 3 shows the possibility of save people and properties without worrying about damages to the UAV.

GA2 and MPGA2 reach again the best results with 96.3% and 96.0% respectively of chance to avoid hit people and properties. Motor problem (ψ_m) and battery overheating problem (ψ_b) are even easier to work around with 99.2% of chance by GA2 and MPGA2.

5.3. *Statistical analysis*

Tables 2 and 3 reports a total of 2400 tests performed for each method regarding all failures and instances. Statistical tests are applied to check significant differences out based on solution quality and computational time.

The criteria of independence, homoscedasticity and normality must be satisfied to apply parametric tests. The samples are independent, if the occurrence of one does not affect the probability of the other and this criteria can be verified by Pearson correlation coefficient. The homoscedasticity occur if the samples have same variances and can be verified by equals variance test as Levene's test. The normality criteria is satisfied if the populations from which the samples were obtained are normally or approximately normally distributed, where a test like Anderson Darling test can be applied. If at least one criterion is not satisfied, we must use non-parametric tests.

All the three criteria fail from data in Tables 2 and 3. Moreover, the non-parametric Friedman test was performed to determine whether the results in Tables 2 and 3 are significantly different. Table 4 summarizes Friedman results where at least one method is significantly different for Tables 2 and 3. The P-Value $= 0.000 < 0.05$ reject the Null hypothesis, so the alternative hypothesis is that one median at least is significantly different.

The Nemenyi's test[8] is applied for pairwise comparison as shown on Table 5. The results indicate that GA2 and MPGA2 outperform GH, GA1 and MPGA1 for

Table 3. Chance of saving people and properties but without worrying with the UAV.

Ψ	Inst.	Landing in Bonus (ϕ_b) or Remainder (ϕ_r) Regions				
		GH	GA1	MPGA1	GA2	MPGA2
Motor (ψ_m)	I1	**100**	**100**	**100**	**100**	**100**
	I2	98	**99**	**99**	**99**	**99**
	I3	97	98	**99**	**99**	**99**
	I4	98	99	**100**	**100**	**100**
	I5	82	99	**100**	**100**	**100**
	I6	90	90	93	**97**	**97**
	Avg	94.2	97.5	98.5	**99.2**	**99.2**
Battery (ψ_b)	I1	99	99	**100**	**100**	**100**
	I2	97	**99**	**99**	**99**	**99**
	I3	94	98	**99**	**99**	**99**
	I4	98	97	99	99	**100**
	I5	68	99	**100**	**100**	**100**
	I6	83	79	85	**98**	97
	Avg	89.8	95.2	97.0	**99.2**	**99.2**
s1 (ψ_{s1})	I1	87	97	**98**	**98**	**98**
	I2	88	88	89	**95**	93
	I3	82	94	94	**95**	94
	I4	83	83	87	**90**	89
	I5	62	92	**95**	**95**	**95**
	I6	56	64	69	**83**	82
	Avg	76.3	86.3	88.7	**92.7**	91.8
s2 (ψ_{s2})	I1	93	98	98	**99**	**99**
	I2	90	95	**96**	**96**	**96**
	I3	87	94	95	**97**	**97**
	I4	88	91	91	**94**	**94**
	I5	57	96	97	**98**	**98**
	I6	64	66	70	**80**	**80**
	Avg	79.8	90.0	91.2	**94.0**	**94.0**
Global Average		85.0	92.3	93.8	**96.3**	96.0

landing on bonus regions. For landing on bonus or remainder regions the MPGA1, GA2 and MPGA2 outperform GH and GA1.

There is no significant difference between GA2 and MPGA2. Tables 2 and 3 show both methods performing equally well for the two land criteria evaluated. Therefore, it will be evaluated next which method reaches the best route first.

The time spent by GA2 and MPGA2 to reach the best route is also evaluated from 2400 executions. The hypothesis to apply parametric tests are not fulfilled, so the non-parametric Friedman and Nemeneyi's tests are performed in Table 6. The Friedman test returns significant differences (P-value < 0.0001), where MPGA2 median value is 0.761 sec. against 0.815 sec. from GA2, and the Nemeneyi's test shows MPGA2 outperforming GA2, GA1 and MPGA1.

Table 4. Friedman test evaluating the differences of methods for Tables 2 and 3.

		From Table 2		From Table 3	
Method	N	Median	Sum of Ranks	Median	Sum of Ranks
GH	24	80.850	42.0	92.350	30.0
GA1	24	81.750	43.5	96.250	50.0
MPGA1	24	82.250	64.5	97.450	81.5
GA2	24	91.250	106.0	97.950	101.5
MPGA2	24	91.150	104.0	97.750	97.0
Overall	120	85.450	–	96.350	–
P-value		0.000		0.000	
DF		4		4	
S		65.81		63.89	
S(ties)		70.51		79.25	

Table 5. Nemeneyi's test results for Tables 2 and 3 instances.

Instance	Group A	Group B
Table 2	GH, GA1, MPGA1	GA2, MPGA2
Table 3	GH, GA1	MPGA1, GA2, MPGA2

Table 6. Friedman's and Nemeneyi's test evaluating the time for each method to get the best solution.

		Friedman's Test		Nemeneyi's Test
Method	N	Median	Sum of Ranks	Groups
GH	2400	0.045	2424.5	A
GA1	2400	0.864	8421.5	C
MPGA1	2400	0.946	9748.5	D
GA2	2400	0.815	8288.0	C
MPGA2	2400	0.761	7117.5	B

Figure 6 shows the analysis of the expected time execution for each method. GH is the fastest one spending on average 0.07 sec. and no longer than 0.18 sec. for 99.9% of best samples. GA1, GA2, MPGA1 and MPGA2 spent on average around 1 sec. and no longer than 2.28, 1.97, 1.91 and 1.86 sec., respectively, considering 99.9% of best samples. These values are extract directly from the 2400 time samples.

5.4. *Examples of routes*

Figure 7 shows four routes determined by the planner MPGA2 in a map M_N with coverage $C_{25\%}$, where the aircraft is flying towards north direction when a failure occurs. It is shown the routes as well as the point S where the critical situation

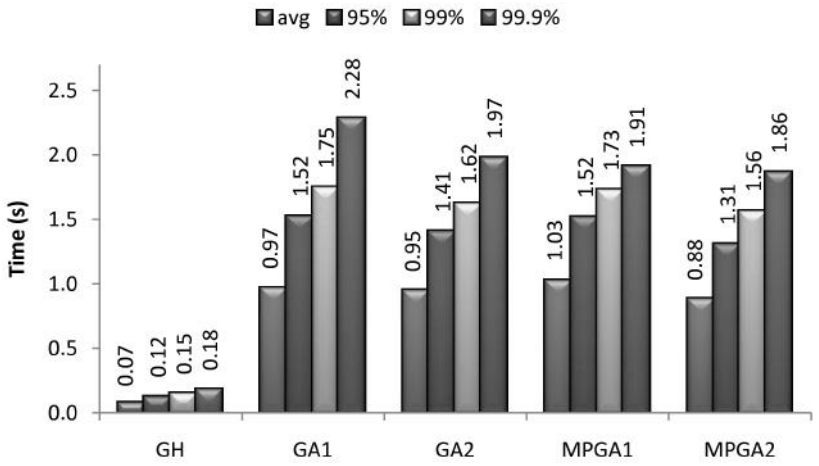


Fig. 6. Expected execution time for each method.

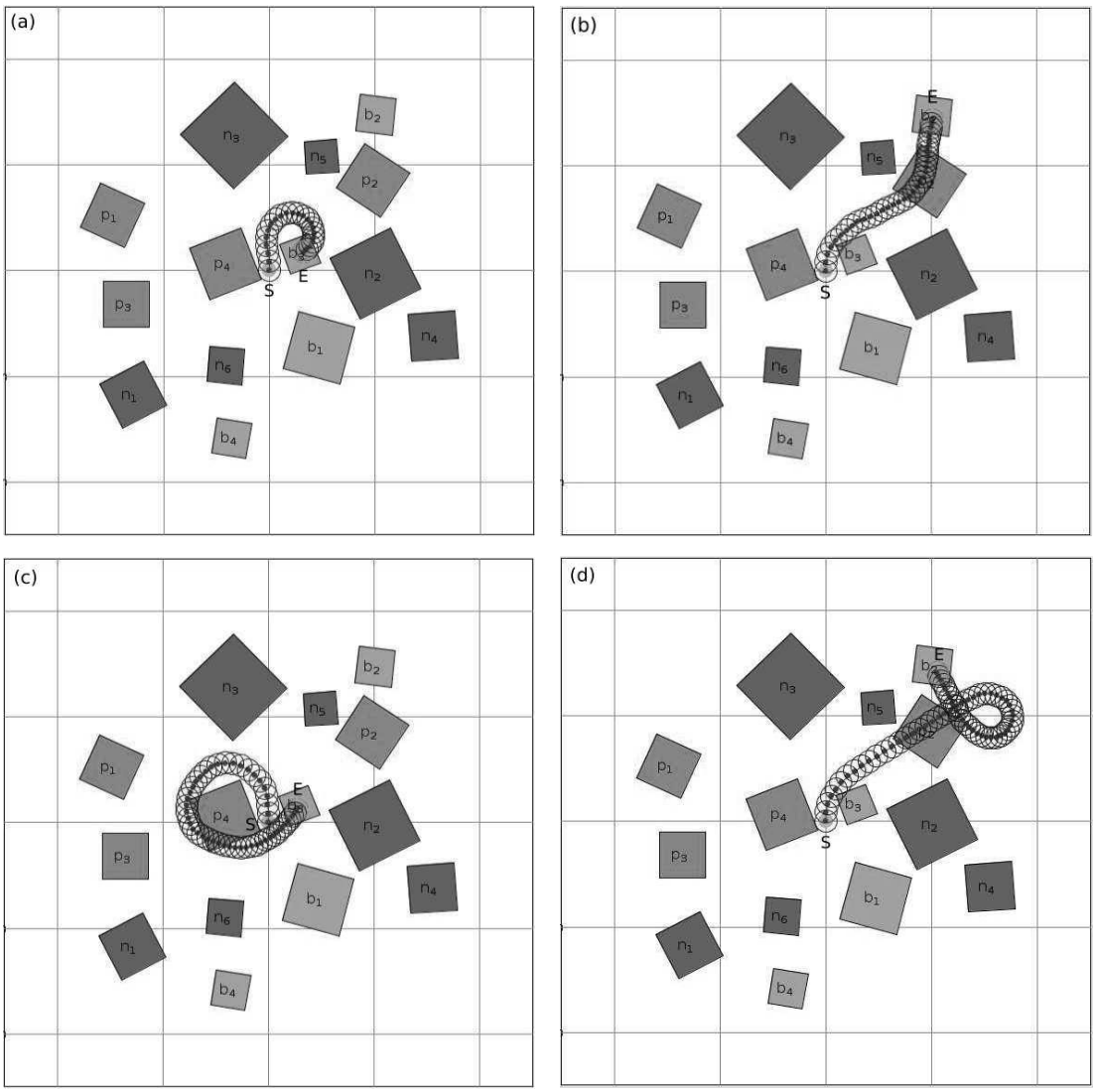


Fig. 7. Examples for each critical situation: (a) ψ_m, (b) ψ_b, (c) ψ_{s1} and (d) ψ_{s2}.

happens and the point E where the aircraft lands. The circles represent the uncertainty related with the UAV position as described in sections 3 and 4.

In Figure 7(a), the aircraft lands on bonus region b_3 faster due to motor failure (ψ_m). There is a landing on region b_2 in Figure 7(b). Figures 7(c) and (d) show landing routes after problems with wings $s1$ and $s2$.

Three routes generated by GH are shown by Figure 8, where the UAV reaches bonus region b_6 in Figure 8(a); the aircraft fails to reach ϕ_{b_2} due to its long distance in Figure 8(b); and there is a route over no-fly zone ϕ_{n_6}, indicating a bad solution, even if it allows to land in bonus region ϕ_{b_1} in Figure 8(c). The regions ϕ_{b_1} and ϕ_{b_3} are not chosen in Figure 8(b) because it would violate the constraint of non-navigability.

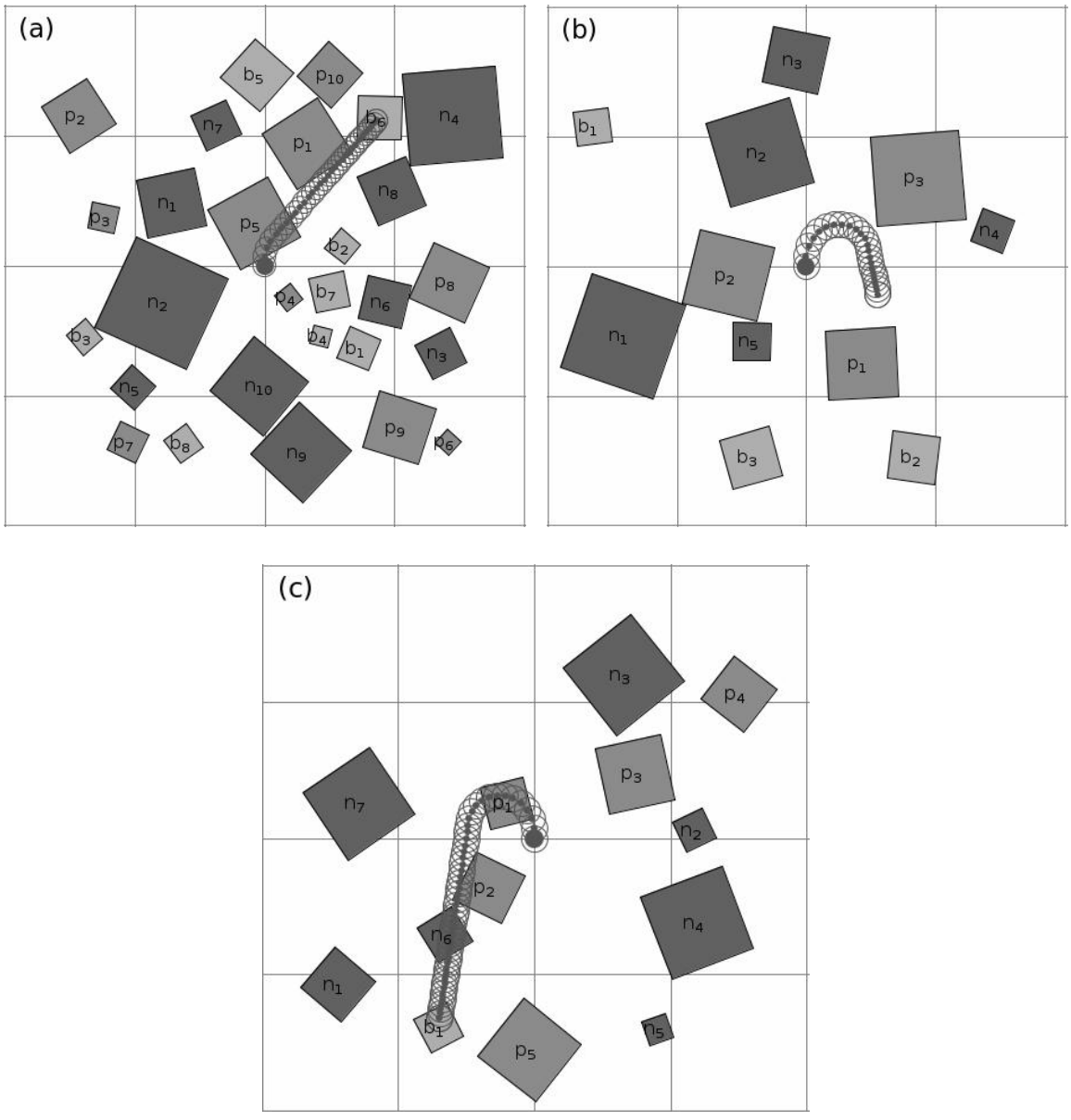


Fig. 8. Examples of routes obtained by the GH.

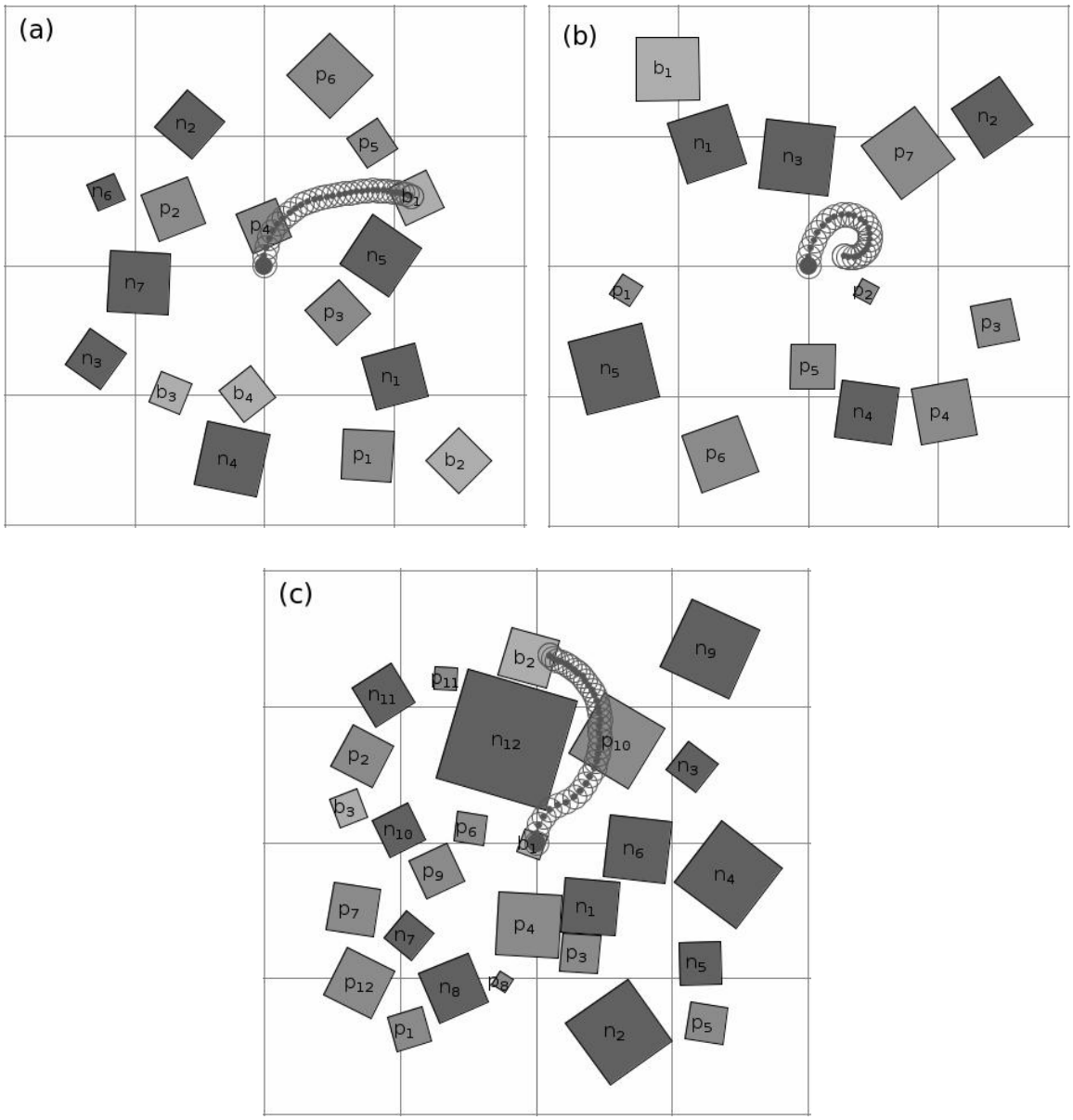


Fig. 9. Examples of routes obtained by the GA2.

Figure 9 shows routes generated by GA2. In Figure 9(a), the aircraft reaches a bonus region and land safely. In Figure 9(b), it fails to reach the ϕ_{b_1}, thus, landing in the remainder region. In Figure 9(c), the UAV has a small overlap when flying over $\phi_{n_{12}}$ and landing on ϕ_{b_2}.

A numerical example is presented next to illustrate how the fitness function is calculated with Equation (9) from solution (trajectory) in Figure 9(c). The first term of the fitness function assigns a bonus for solution that lands in a bonus region. This is done by estimating the probability of the aircraft to land in b_2 as $P(x_K \in Z^2_{\phi_b}) = 0.9356$. In this case, a bonus of -1871.2 was given following Equation (17).

$$f_{Land_{\phi_b}} = -C_{\phi_b} \cdot \sum_{i=1}^{|\phi_b|} (P(x_K \in Z^i_{\phi_b}))$$
$$= -C_{\phi_b} \cdot 0.9356 = -2000 \cdot 0.9356 = -1871.2 \quad (17)$$

The second term assigns a penalty by landing in penalty region. In this example, there is no landing over such region, i.e. $P(x_K \in Z^i_{\phi_p}) = 0$ for all region in ϕ_p and $f_{Land_{\phi_p}} = 0$. The chance of the aircraft violates any of the non-navigable regions is estimated as $P(\wedge_{t=0}^{K} \wedge_{i=1}^{|\phi_n|} x_t \notin Z^i_{\phi_n}) = 0.8522$. The region with high penalty was n_{12} (see Figure 9(c)). Moreover, the maximum between $(0, 1 - \Delta - 0.8522)$ is 0.1468 with $\Delta = 0.001$ following Equation (18).

$$f_{Flight_{\phi_n}} = C_{\phi_n} \cdot max\left(0, 1 - \Delta - P\left(\bigwedge_{t=0}^{K} \bigwedge_{i=1}^{|\phi_n|} x_t \notin Z^i_{\phi_n}\right)\right)$$
$$= C_{\phi_n} \cdot 0.1468 = 14680 \quad (18)$$

There are also penalties for the amount of curves that the aircraft performs (Equation (13)) and the distance from the center of the region where the aircraft lands (Equation (14)), respectively, given by $f_{Curves} = 20.33$ and $f_{DistUAV_{\phi_b}} = 50$. In this example the landing region was b_2 and the model gave a penalty of 50 that means a distance of 50 meters from the center of the region b_2 (see Figure 9(c)). Let us assume that there is no speeding of the UAV at the last time step (Equation (15)), so $f_{Violated_T} = 0$. Finally, let us suppose that the failure occurred in the battery ($\psi = \psi_b$), and the amount of waypoints allocated to the landing instant was $K = 35$ with $T = 60$ and $C_{\phi_b} = 2000$. Thus, the penalty associated is given by Equation (19).

$$f_\psi = \begin{cases} C_{\phi_b} \cdot 2^{\frac{(K-T)}{10}}, & \psi = \psi_b \\ 0, & \text{otherwise} \end{cases} \quad \Rightarrow \quad f_\psi = 2000 \cdot 2^{\frac{(35-60)}{10}} = 353.4 \quad (19)$$

The fitness value is returned by Equation (20), whose high value (13241.33) indicates an infeasible solution that is penalized.

$$\textit{fitness} = -1871.2 + 0 + 14688.8 + 20.33 + 50 + 0 + 353.4 = 13241.33 \quad (20)$$

The method will take a while to update the route and the UAV position will keep changing. The UAV speed ranges from 11.1 m/s to 33.5 m/s, but it was assumed 24 m/s as speed at the time a fault is detected. Let us assume that the method used for emergency route re-planning is MPGA2, so applying this speed to the worst case time of MPGA2 give us 24 m/s $\times$ 1.86 s = 44.64 m. The distance traveled between the failure detection and getting the final emergency route is about 45 meters. Figure 10 shows that such shift does not compromise the re-planning process given the dimensions of the map and UAV speed.

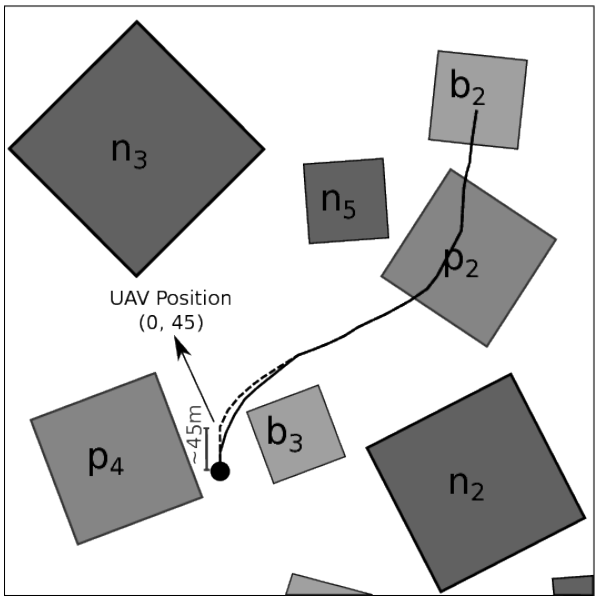


Fig. 10. Displacement of the UAV during the processing time of the emergency route.

5.5. *Simulations in FlightGear*

A simulation is executed aiming to illustrate the behaviour of the aircraft following one of the paths provided by MPGA2. The FlightGear (FG) is chosen since it is an open-source simulator with many resource available. FG is set to simulate the flight dynamic of Cessna 172, but an autopilot was coded to control automatically the Cessna 172 flight. Figure 11 shows the framework that integrates FG, autopilot and MPGA2 path. MGPA2 sends the path to the autopilot that aborts the current path and starts a new path. The autopilot executes the MPGA2 path based on the environment and aircraft simulations from FG. Due to the fact that the aircraft used in flight (Cessna 172) is larger than the UAV used in the previous experiments (Tiriba), the dimensions of the route and map used were increased.

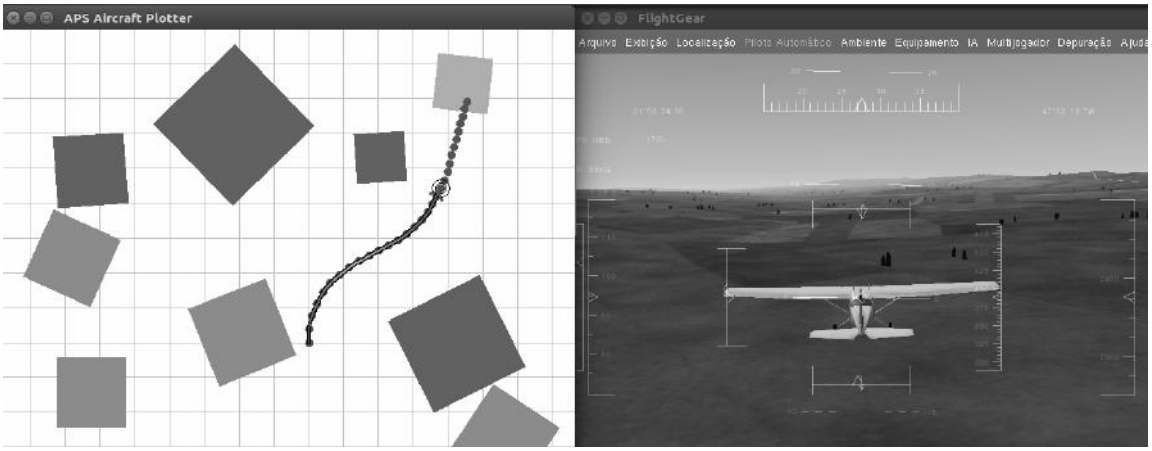


Fig. 11. Framework for FlightGear with MPGA2 paths.

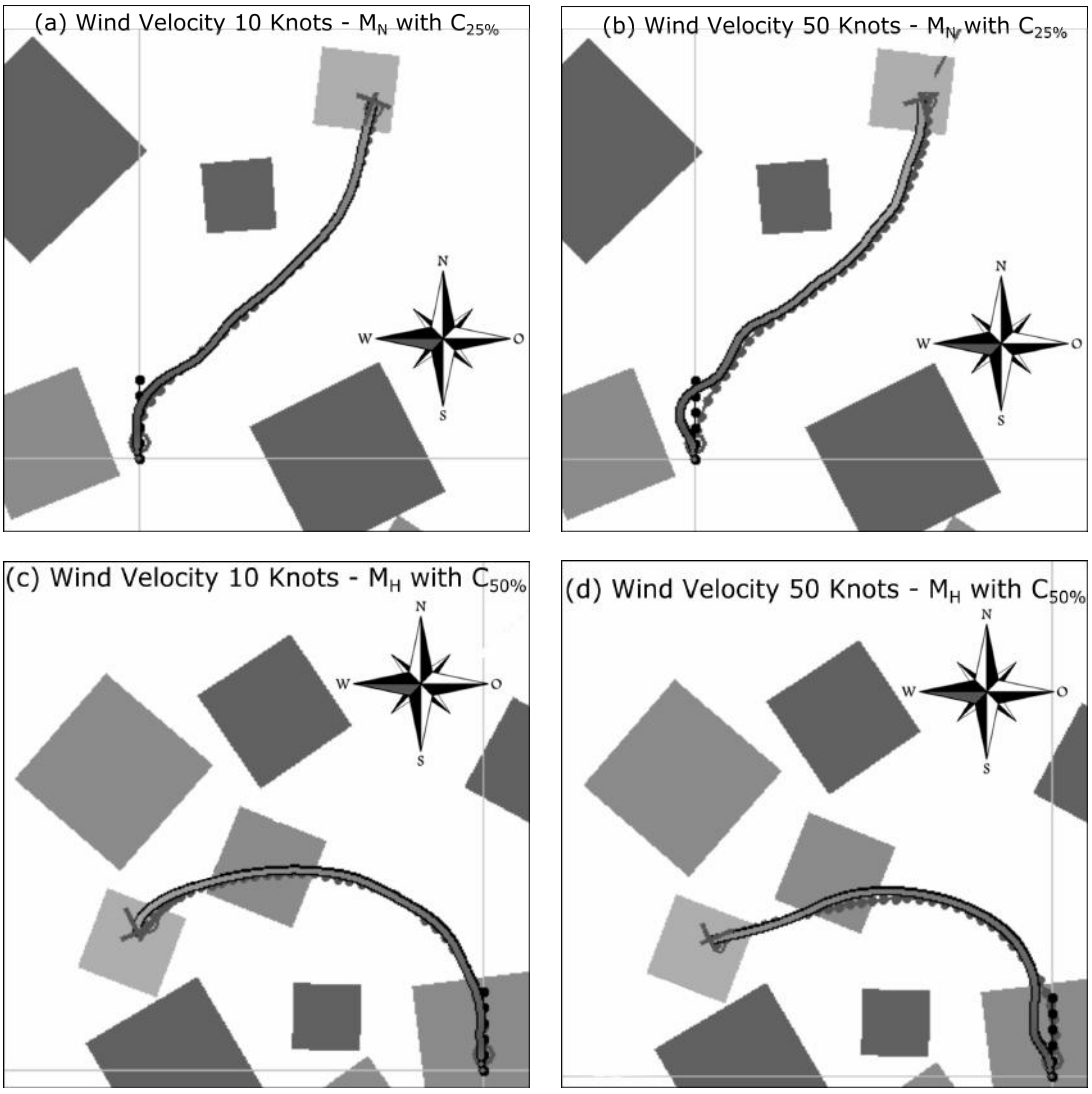


Fig. 12. (a) and (c) FG simulation with winds 10 knots, (b) and (d) with winds 50 knots. Wind direction: west.

Two maps from Instances I3 (M_N with $C_{25\%}$) and I6 (M_H with $C_{50\%}$) are selected to execute the simulations and a battery failure ψ_b is assumed. Figures 12(a) and (c) show the paths when there are winds with 10 knots during the flight simulation, while Figures 12(b) and (d) have the same path execution under winds with 50 knots. The winds bring disturbances that do not allow to follow exactly the MPGA2 route in both maps. As expected, the difference between MPGA2 path and the path executed by autopilot is larger under strong winds, but the aircraft stays away from non-fly zones and near the MPGA2 path in both cases.

Another amount of simulations is conducted using FG. These simulations aim to stress different wind speeds and directions on the UAV's flight. The same battery failure on a single map is used for all simulations. Table 7 summarizes the 17 experiments performed. The first row of this table shows a windless flight where simulation time spends 178 seconds, the average error was 8.65 meters with StDev

Table 7. Results obtained in FlightGear for different wind speeds and directions.

Wind Direction		Wind Speed [knots]	Simulation Time [seconds]	Average Error [meters]	St. Dev Error [meters]
No Direction	No Dir	0	178	8.65	7.79
North to South	NtoS	10	192	21.18	12.26
North to South	NtoS	20	208	36.21	19.16
North to South	NtoS	30	226	53.26	27.20
North to South	NtoS	40	241	70.47	37.93
South to North	StoN	10	164	12.04	7.85
South to North	StoN	20	157	24.99	15.77
South to North	StoN	30	154	39.93	24.78
South to North	StoN	40	139	57.38	33.66
East to West	EtoW	10	194	14.02	7.61
East to West	EtoW	20	197	32.24	11.41
East to West	EtoW	30	218	50.15	16.11
East to West	EtoW	40	219	69.25	23.05
West to East	WtoE	10	176	26.27	11.02
West to East	WtoE	20	172	43.72	19.37
West to East	WtoE	30	157	68.37	21.27
West to East	WtoE	40	157	92.33	29.51

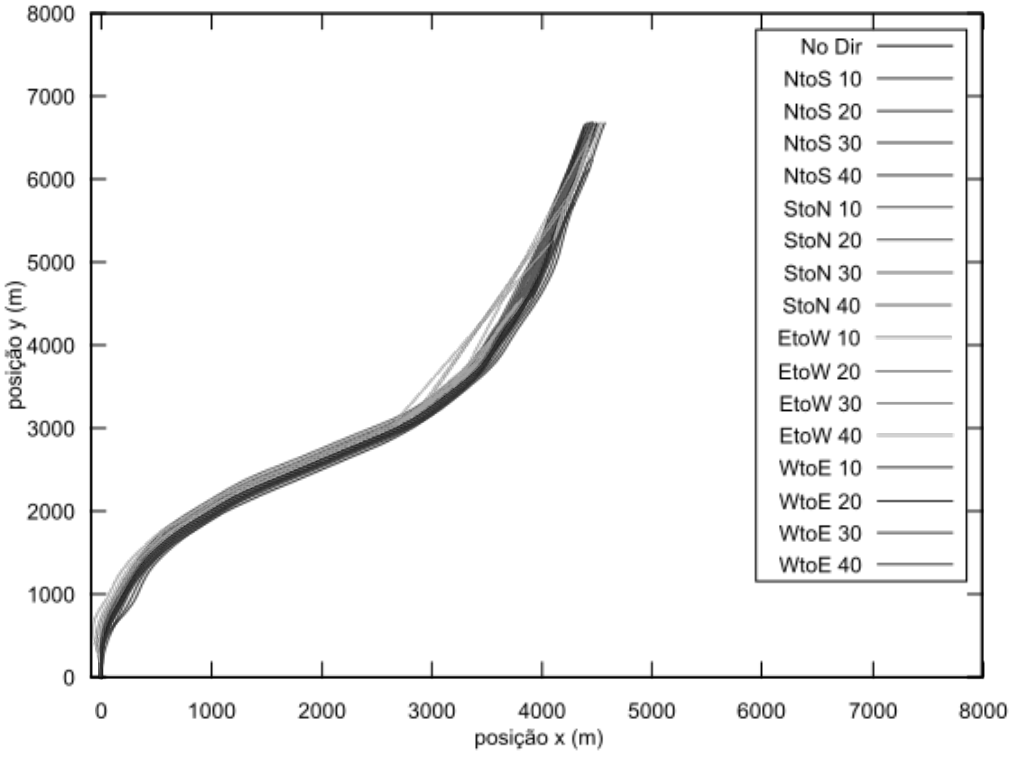


Fig. 13. Routes obtained in experiments using FlightGear simulator.

error 7.79 meters. The error of 8.65 m means the value on average that the aircraft is shifted from the original route taking into account the whole path. The results are shown for winds going from North to South and wind speeds ranging from 10 to 40 knots. It can be seen that the simulation time increases or decreases slightly depending on the wind speed and direction. The column Average error shows that wind speed increasing leads to error values that increase proportionally. Some videos of these simulations are available at web.[b]

The graph of Figure 13 shows all routes in Table 7. These routes show that the error of the aircraft in relation to original route is quite low, even under different wind conditions.

6. Conclusions

A path re-planning problem to land a UAV under four types of critical situation is approached by this paper. The aim was to minimize damages increasing the safety during the emergency landing. Three methods were proposed, GH, GA and MPGA, for path re-planning, where the GA and MPGA were evaluated using GH to initialize paths (GA1, MPGA1) and without such initialization (GA1, MPGA1). A set of 600 maps were generated to evaluate the proposed methods.

All methods were able to land the aircraft for more than 67% of maps on average under all critical situations. The type of landing executed by the UAV was evaluated under two situations. First, The UAV landing is evaluated taking into account the chance to save the UAV without putting risk on people, properties or itself. Next, the UAV landing is evaluated taking into account the emphasis on saving people and properties, without to care about UAV damages.

The methods are better dealing with battery problem and worse for landing the UAV under motor failure. Thus, this issue must be taken into account as future works to improve these methods. MPGA2 seems to be the most promising one since it takes more advantage to evolve individuals initialized by GH. The statistical analysis do not indicate significant difference among MPGA2 and GA2 based on landing percentages, but MPGA2 lands the UAV in safety regions for more than 79% of the maps. However, they show a better performance of MPGA against GA2 based on computer time to reach a better route. Of course, GH is the approach designed to return fast solution, but it can fail more than GA and MPGA to land in bonus regions.

The simulation with FlightGear showed that the aircraft ares able to follow the path even under different wind velocities and directions. During flights without wind, the UAV deviates from the original route on average 8.65 meters. This error can be considered low based on the aircraft and scenery dimensions. The average error reached around 92.33 meters under winds of 40 knots and the deviation can be considered average for the dimensions mentioned. However, even under

[b]https://youtu.be/sN6yfr_3VbU

such extreme conditions, the aircraft reaches adequate regions to land. Thus, the proposed methods find good quality solutions within a short computational time, which is a relevant aspect for this problem.

As future work, the mathematical model will be improved aiming to describe this problem using a mixed-integer linear programming model, allowing to solve it with exact methods. Finally, the functions to describe critical situations will also be improved and other failures will be considered.

Acknowledgments

This paper acknowledges the support of Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) projects 2013/26091-2, 2014/12297-0 and 2014/11331-0. We gratefully acknowledge the financial support of Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

References

1. 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, in *2015 IEEE 27th Int. Conf. on Tools with Artificial Intelligence* (*ICTAI*) (IEEE, 2015), pp. 486–493.
2. 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, in *Genetic and Evolutionary Computation Conference* (*GECCO*) (2016), pp. 853–860.
3. E. Besada-Portas, L. Torre, A. Moreno and J. L. Risco-Martín, On the performance comparison of multi-objective evolutionary UAV path planners, *Inf. Sci.* (July 2013) 111–125.
4. L. Blackmore, M. Ono and B. C. Williams, Chance constrained optimal path planning with obstacles (IEEE Press, 2011).
5. K. Branco, J. Pelizzoni, L. Oliveira Neris, O. Trindade, F. Osorio and D. Wolf, Tiriba — A new approach of UAV based on model driven development and multi-processors, in *ICRA* (IEEE, May 2011), pp. 1–4.
6. L. Buriol, P. M. França and P. Moscato, A new memetic algorithm for the asymmetric traveling salesman problem, *Journal of Heuristics* **10**(5) (September 2004) 483–506.
7. R. Clarke and L. B. Moses, The regulation of civilian drones' impacts on public safety, *Computer Law & Security Review* **30**(3) (2014) 263–285.
8. J. Demšar, Statistical comparisons of classifiers over multiple data sets, *J. Mach. Learn. Res.* **7** (December 2006) 1–30.
9. A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing* (Springer-Verlag, 2003).
10. L. J. Eshelman and J. D. Schaffer, Real-coded genetic algorithms and interval-schemata, in D. L. Whitley, ed. *Foundation of Genetic Algorithms 2* (Morgan Kaufmann, San Mateo, CA, 1993), pp. 187–202.
11. P. M. França, A. Mendes and P. Moscato, A memetic algorithm for the total tardiness single machine scheduling problem, *European Journal of Operational Research* **132**(1) (2001) 224–242.
12. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st edn. (Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989).

13. H. X. Li, *Kongming: A Generative Planner for Hybrid Systems with Temporally Extended Goals* (Massachusetts Institute of Technology, 2010), 237 pp.
14. A. L. P. Mattei, E. Fonseca, N. M. 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*) (Munich, 2013).
15. N. Meuleau, C. Neukom, C. Plaunt, D. E. Smith and T. Smith, The emergency landing planner experiment, in *21st Int. Conf. on Automated Planning and Scheduling* (2011).
16. N. Meuleau, C. Plaunt, D. E. Smith and T. B. Smith, An emergency landing planner for damaged aircraft, in K. Z. Haigh and N. Rychtyckyj (eds.) *IAAI* (AAAI, 2009).
17. Z. Michalewicz and M. Schoenauer, Evolutionary algorithms for constrained parameter optimization problems, *Evolutionary Computation* **4** (1996) 1–32.
18. P. Moscato, A. Mendes and R. Berretta, Benchmarking a memetic algorithm for ordering microarray data, *Biosystems* **88**(1-2) (2007) 56–75.
19. M. Ono, B. C. Williams and L. Blackmore. Probabilistic planning for continuous dynamic systems under bounded risk, *J. Artif. Int. Res.* **46**(1) (January 2013) 511–577.
20. Y. V. Pehlivanoglu, A new vibrational genetic algorithm enhanced with a Voronoi diagram for path planning of autonomous UAV, *Aerospace Science and Technology* **16**(1) (2012) 47–55.
21. C. Toledo, L. Oliveira and P. França, Global optimization using a genetic algorithm with hierarchically structured population, *Journal of Computational and Applied Mathematics* **261** (2014) 341–351.
22. C. F. M. Toledo, M. Arantes, R. R. R. de Oliveira and B. Almada-Lobo, Glass container production scheduling through hybrid multi-population based evolutionary algorithm, *Appl. Soft Comput.* **13**(3) (2013) 1352–1364.
23. A. Tuncer and M. Yildirim, Dynamic path planning of mobile robots with improved genetic algorithm, *Comput. Electr. Eng.* **38**(6) (November 2012) 1564–1572.
24. G. Varela, P. Caamaño, F. Orjales, A. Deibe, F. López-Peña and R. J. Duro, Autonomous UAV based search operations using constrained sampling evolutionary algorithms, *Neurocomputing* **132** (2014) 54–67.
25. X. Zhang and H. Duan, An improved constrained differential evolution algorithm for unmanned aerial vehicle global route planning, *Applied Soft Computing* (2015) 270–284.


ICMC
USP
SÃO CARLOS