Universidade Federal da Paraíba

Centro de Informática

Programa de Pós-Graduação em Informática

Um Ambiente de Simulação de Múltiplos Drones para Avaliação de Estratégias usando Cossimulação

Leydson Barbosa Silva

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Sinais, Sistemas Digitais e Gráficos

Alisson Vasconcelos de Brito (Orientador)

João Pessoa, Paraíba, Brasil

©**Leydson Barbosa Silva**, 15 de Agosto de 2019

Leydson Barbosa Silva

Um Ambiente de Simulação de Múltiplos Drones para Avaliação de Estratégias usando Cossimulação

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal da Paraíba como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Alisson Vasconcelos de Brito (Orientador)

João Pessoa, Paraíba, Brasil

©Leydson Barbosa Silva, 15 de Agosto de 2019

Catalogação na publicação
Seção de Catalogação e Classificação

S586a Silva, Leydson Barbosa.
Um Ambiente de Simulação de Múltiplos Drones para Avaliação de Estratégias usando Cossimulação / Leydson Barbosa Silva. - João Pessoa, 2019.
65 f. : il.

Orientação: Alisson Vasconcelos de Brito.
Dissertação (Mestrado) - UFPB/CI.

1. Simulador de Múltiplos Drones. 2. Cossimulação. 3. Avaliação de estratégias de voo. I. Brito, Alisson Vasconcelos de. II. Título.

UFPB/BC

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de Leydson Barbosa Silva, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 30 de agosto de 2019.

Aos trinta dias do mês de agosto, do ano de dois mil e dezenove, às quatorze horas, no Centro de Informática da Universidade Federal da Paraíba, em Mangabeira, reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final do Sr. Leydson Barbosa Silva, vinculado a esta Universidade sob a matrícula nº 20171003885, candidato ao grau de Mestre em Informática, na área de "Sistemas de Computação", na linha de pesquisa "Sinais, Sistemas Digitais e Gráficos", do Programa de Pós-Graduação em Informática, da Universidade Federal da Paraíba. A comissão examinadora foi composta pelos professores: Alisson Vasconcelos de Brito (PPGI-UFPB) Orientador e Presidente da Banca, Vivek Nigam (PPGI-UFPB), Examinador Interno, Ramon Leonn Victor Medeiros (IFPB), Examinador Externo à Instituição. Dando início aos trabalhos, o Presidente da Banca cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse a exposição oral do trabalho de dissertação intitulado: "Um Ambiente de Simulação de Múltiplos Drones para Avaliação de Estratégias usando Cossimulação". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: "***aprovado***". Do ocorrido, eu, Clauirton de Albuquerque Siebra, Coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai assinada por mim e pelos membros da banca examinadora. João Pessoa, 30 de agosto de 2019.

Prof. Dr. Clauirton de Albuquerque Siebra

Prof. Alisson Vasconcelos de Brito
Orientador (PPGI-UFPB)

Prof. Vivek Nigam
Examinador Interno (PPGI-UFPB)

Prof. Ramon Leonn Victor Medeiros
Examinador Externo à Instituição (IFPB)

Resumo

Aplicações com múltiplos *Drones* estão se tornando mais presente nas industrias, usadas em aplicações de defesa e nas comerciais, onde as RPAs executam suas tarefas de forma autônoma. Garantir, entretanto, que aplicações de múltiplos *Drones* autônomos sejam seguros, isto é, sejam capazes de executar suas tarefas e alcançar seus objetivos, é um desafio, já que decisões precisam ser tomadas com uma visão bastante limitada do mundo em um ambiente hostil. A simulação pode fornercer informações importantes sobre o desempenho das estratégias das missões. Por exemplo, simulações podem prover informações sobre quais estratégias poderiam fornecer garantias para a segurança do equipamento, tendo em vista o alto investimento necessário para criação de um sistema com múltiplos *Drones*. Nesse sentido, propomos um ambiente de cossimulação de múltiplos *Drones* para a análise de missões, a fim de avaliar estratégias de voo diferentes. Para tornar o ambiente o mais realista possível, introduzimos ameaças externas tais como, ventos, setores com restrições de voo e colisões de aeronaves. Para validar o ambiente proposto utilizamos um estudo de caso de um sistema de vigilância aérea, onde comparamos dois tipos de estratégias de voo diferentes. O sistema capacitou a análise entre as estratégias de voo, em termos de colisões entre *Drones*, consumo de energia e no desempenho das missões. Os resultados obtidos demonstram a eficácia da ferramenta na análise das estratégias, possibilitando a criação e avaliação de estratégias mais efetivas.

Palavras-chave: Simulador de Múltiplos *Drones*, Cossimulação, Avaliação de estratégias de voo..

Abstract

Multi-UAV applications are becoming more present in industry, used in defense and commercial applications, where UAVs perform tasks autonomously. Ensuring, however, that autonomous Multi-UAV applications are safe, that is, are able to perform their tasks and achieving their goals, is challenging, as decisions have to be taken with many time limited view of the world and in a hostile environment. Simulation can provide important insights on how well mission strategies perform. For instance, simulations can provide insights on which strategies could provide guarantees for the safety of equipments, in view of the high investments necessary for the creation of a multi-UAV system. We propose a co-simulation framework of multiple UAVs for the analysis of missions, in order to evaluate different flight strategies. To make the environment as realistic as possible, we introduced external threats such as winds, sectors with flight restrictions, and aircraft crashes. We validated our framework using a case study of an air surveillance system, where two types of flight strategies comparing two different flight strategies. We are able to analyse trade-offs between the flight strategies, in terms, of chances of UAV collision, running out of energy, and performing tasks. The results show the effectiveness of the tool in the strategy analysis, allowing the creation and evaluation of more effective strategies.

Keywords: Multi-UAV Simulator, Co-Simulation, Flight Strategy Valuation.

Agradecimentos

Gostaria de agradecer a Deus em primeiro lugar que sempre me guiou e esteve presente na minha vida.

Gostaria de agradecer à minha família, que sempre acreditou e me deu forças para que pudesse chegar cada vez mais longe.

Gostaria de agradecer a minha noiva Camilla Sales, por estar sempre ao meu lado, sofrendo nos momentos difíceis, comemorando nos mais alegres. Você tornou essa jornada mais prazerosa.

Gostaria de agradecer a todos os professores que, de alguma forma contribuiram para a minha formação, ao meu orientador, Professor Alisson Vasconcelos de Brito, por toda sinceridade, paciência e sabedoria. Você foi uma influência muito positiva para a realização de um sonho. Sou seu fã.

Gostaria também de agradecer a todos meus estimados colegas de estudos, que deixam o ambiente mais leve, que estimulam a busca pelo conhecimento e passam suas experiências positivas e negativas. Gostaria de agradecer especialmente a José de Souza Barros, que foi sempre muito solícito, tirando dúvidas e ajudando a concretizar esse sonho. Muito Obrigado.

Dedicatória

Dedico este trabalho a meu pai, que onde quer que esteja agora, sei que estaria feliz e orgulhoso.

Sumário

Lista de Siglas

ANAC : Agência Nacional de Aviação Civil.

API : *Application Programming Interface*.

CSV : *Comma-separated values*.

FED : *Federation Execution Details*.

FOM : *Federation Object Model*.

GPS : *Global Positioning System*.

GCS : *Ground Control Station*.

HLA : *High Level Architecture*.

IDE : *Integrated Development Environment*.

IEEE : *Institute of Electrical and Electronics Engineers*.

JOSM : *Java OpenStreetMap*.

JTS : *Java Topology Suite*.

MoC : *Model of Computation*.

NER : *Next Event Request*.

OMT : *Object Model Template*.

OSM : *OpenStreetMap*.

RO : *Receive Order*.

RPA : *Remotely-Piloted Aircraft*.

RPAS : *Remotely-Piloted Aircraft System*.

RTI : *Run-Time Infrastructure*.

RTIA : *Run-Time Infrastructure Ambassador*.

RTIG : *Run-Time Infrastructure Gateway*.

SITL : *Software in the Loop*.

TAG : *Time Advance Grant*.

TAR : *Time Advance Request*.

TCP : *Transmission Control Protocol*.

TSO : *Time Stamp Order*.

UAV : *Unmanned aerial vehicle*.

UDP : *User Datagram Protocol*.

VANT : Veículo Aéreo Não Tripulado.

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

Veículos Aéreos Não Tripulados (VANTs) são aeronaves que funcionam sem a necessidade de um piloto e tripulação a bordo, podendo voar de forma autômata, ou controlada remotamente por um indivíduo via rádio frequência. Algumas das alcunhas empregadas ao se referir sobre eles são: UAV, que vem do inglês *Unmanned Aerial Vehicle*, aeroplanos, ou simplesmente aeromodelos, e Drones, este pelo fato do zumbido que o motor faz ao girar as hélices em voo ser parecido com as dos zangões, *drones* em inglês. Recentemente, a Agência Nacional de Aviação Civil (ANAC) definiu algumas regras de utilização e operação de VANTs no âmbito civil, também chamados popularmente de *drones* [Civil 2017]. Nesse documento, com a finalidade de padronização, a ANAC define duas novas nomenclaturas para VANTs: Aeromodelos, definidas como toda e qualquer aeronave não tripulada e remotamente pilotada usada exclusivamente para diversão ou lazer, e as Aeronaves Remotamente Pilotadas (RPAs), que são as aeronaves pilotadas que servem a outro propósito quanto a seu uso, a exemplo, para fins comerciais, institucionais ou experimentais. Introduziu ainda o conceito de Sistemas de Aeronave Remotamente Pilotada (RPAS), que são o conjunto de equipamentos necessários para a utilização da RPA, que compreende o enlace de pilotagem e qualquer outro componente usado para a sua utilização.

Os RPAs podem ser usadas como ferramentas para fins diversos. Seu conceito surgiu ainda no século XIX, quando foram alçados balões com cargas de explosivos no céu de veneza. Já no século XX durante a primeira guerra mundial, os Estados Unidos começaram a pesquisa para a construção de aeronaves sem pilotos, mas a guerra terminou sem que houvesse alguma aplicação. Ainda na área militar, as RPAs foram aperfeiçoadas e usadas nas

décadas seguintes, nas missões de reconhecimento e aquisições de alvos, com a finalidade de diminuir as perdas humanas em missões de alto grau de risco. De lá pra cá seu uso tem se difundido em áreas de atuações diversas, a exemplo nas atividades comerciais [Ramesh et al. 2015] [Kersnovski, Gonzalez e Morton 2017] [Grippa 2016] [Zhou et al. 2016], nas áreas de cunho científico [Holness et al. 2016], em áreas com agricultura [Alsalam et al. 2017] [Ghazal, Khalil e Hajjdiab 2015], nas atividades de monitoramento, busca e salvamento [Erdos, Erdos e Watkins 2013] [Yuan, Liu e Zhang 2015], em áreas institucionais [Heintz, Rudol e Doherty 2007], em atividades pessoais lúdicas [Ross 2014], entre outros.

Por outro lado, o avanço da tecnologia tem tornado o RPAS cada vez mais complexo, a medida que sistemas heterogêneos passam a ser integrados. Na prática, isto significa que componentes de softwares de diversas linguagens de programação distintas precisam trocar informações. Com isso, faz-se necessário a utilização de ferramentas que simulem ambientes complexos de forma mais realista possível.

O uso de ferramentas de simulação é de suma importância para o planejamento e estudo do comportamento das RPAs em certas situações. Essas ferramentas permitem uma melhor análise e entendimento de diferentes aspectos do comportamento das RPAs, ajudando ao design de estratégias mais eficientes para o sucesso de suas missões. Além disso, o emprego de tais ferramentas é menos oneroso do que testes com modelos reais.

Apesar dessa importância, não é trivial encontrar ferramentas que suportem o projeto de estratégias de voos e permitam avaliar essas estratégias em um ambiente de simulação com detalhes que o aproximem de um ambiente real de voo.

Alguns esforços foram empreendidos no sentido de resolver este problema [Barros et al. 2016] [Barros et al. 2017] [Brito et al. 2013]. Nestes trabalhos, o Ptolemy foi utilizado para projetar as estratégias de voos e o simulador SITL/Ardupilot para representar o comportamento da RPA. Apesar disso, os trabalhos citados auxiliam no projeto e avaliação de estratégias de voos que utilizam apenas uma RPA.

O principal desafio do desenvolvimento de estruturas de cossimulação para cenários com múltiplas RPAs é o alto grau de complexidade envolvido. Enquanto para cenários com *Drone* único, a cossimulação envolve um único *Drone* e o modelo com a decisão, o uso de cenários com múltiplos *Drones* requer a sincronização de múltiplas instâncias de VANTs. Além disso, faz-se necessário reunir e consolidar os dados nas diferentes instâncias de simulação

de VANTs, a fim de determinar, por exemplo, se os VANTs colidiram.

Neste trabalho é proposto um ambiente de cossimulação que permite a simulação e a avaliação de estratégias de voos com a presença de mais de uma RPA. Cada RPA no cenário tem o seu comportamento visual descrito pela ferramenta de simulação SITL/Ardupilot. O ambiente usa o *High Level Architecture* (HLA) [IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Federate Interface Specification 2010] para reunir e trocar dados de telemetria entre as diferentes instâncias do SITL/Ardupilot. O modelo de decisão, juntamente com as estratégias foram desenvolvidas utilizando o Ptolemy II. Além disso, o ambiente proposto dispõe ainda de algumas características, tais como, influências externas de ventos, zonas com restrição de voo e colisões.

A principal contribuição desse trabalho é a melhoria e extensão do trabalho desenvolvido por [Barros et al. 2017], de modo a incrementar ao ambiente, a possibilidade de utilização de múltiplas RPAs. Devido a complexidade do ambiente base, que possui diversos componentes, qualquer alteração mínima em um desses componentes levam a alterações em todo os outros. O ambiente propicia a avaliação de estratégias de voos determinada pela simulação de múltiplos *Drones*. De modo a integrar diferentes simuladores o ambiente utiliza cossimulação. Foi utilizado RPAs denominadas quadricópteros, aeronaves compostas de quatro hélices sendo uma para cada motor, com decolagem e pouso vertical e capacidade de estabilização em voos. Um estudo de caso foi feito como prova de conceito para verificar a eficácia da ferramenta.

1.1 Objetivos

Nesta seção será descrito os objetivos que nortearam o desenvolvimento deste trabalho.

1.1.1 Objetivos Gerais

O objetivo geral deste trabalho é o desenvolvimento extensivo e melhoria de um ambiente com a integração de diferentes simuladores utilizando cossimulação. Este ambiente deve conter a capacidade de simular múltiplas RPAs e avaliar diferentes estratégias de voos.

1.1.2 Objetivos Específicos

Como objetivos específicos podemos citar:

- Extender um ambiente já existente de modo a possibilidade de utilização de múltiplas RPAs;
- Desenvolver a ferramenta capaz de prover dados suficientes para a análise das estratégias de voos utilizadas;
- implementar fenômenos de incertezas dentro do ambiente simulado;
- Desenvolver a ferramenta com a capacidade futura de interagir com RPAs físicos;
- Desenvolver a ferramenta com capacidade de ser executada em vários computadores de forma paralela.

1.2 Metodologia

"Método científico é o conjunto de processos ou operações mentais que devemos empregar na investigação"[Prodanov e Freitas 2013]. A Metodologia utilizada neste trabalho, do ponto de vista de sua natureza, foi uma pesquisa aplicada objetivando o desenvolvimento de uma ferramenta de cossimulação com a capacidade de fornecer dados para avaliar estratégias de voo. Em se tratando do ponto de vista de seus objetivos a pesquisa teve caráter exploratório com o estudo do tema e a aplicação de testes como prova de conceito, a fim de demonstrar, através de resultados, a eficácia da ferramenta. Quanto aos procedimentos técnicos adotados para essa pesquisa, foi utilizada uma pesquisa bibliográfica da literatura pertinente junto com uma pesquisa experimental, utilizando simulações através de um estudo de caso.

O trabalho é uma extensão da proposta de [Barros et al. 2017], que desenvolveu um ambiente de cossimulação para avaliação de estratégias utilizando apenas um *Drone*. A idéia central, bem como a utilização dos modelos de estratégias e o algorítimo de possibilidade de ocorrência de ventos foram mantidos, mas precisaram de mudanças para adaptção, quando se acrescenta várias RPAs.

Alguns procedimentos foram utilizados visando o desenvolvimento da ferramenta que este trabalho propõe:

- Estudo das ferramentas de simulação;
- Estudo da arquitetura de alto nível, e de sua integração entre os simuladores;
- Utilização de ferramentas de simulação para a representação da RPA;
- Estudo do funcionamento das estratégias legadas e seu adequamento quanto a utilização de múltiplos *Drones*;
- Utilização de mapas para a visualização das trajetórias das RPAs em tempo real;
- Integração das várias instâncias dos *Drones* com os modelos de estratégias;
- Implementação de zonas com restrição de voo no ambiente de simulação;
- Implementação de algorítimo de detecção de colisões, com a capacidade de determinar colisões entre RPAs e obstáculos;
- Definição de um estudo de caso com a finalidade de validar a ferramenta;
- Execução de várias simulações utilizando cenários e estratégias diferentes para analisar o comportamento das RPAs;
- Desenvovimento de um algoritimo para coleta de dados;
- análise das amostras coletadas e comparação entre os resultados;

1.2.1 Materiais e Métodos

Considerando o desenvolvimento extensivo do ambiente de simulação envolvendo múltiplos *Drones*, a utilização de algumas tecnologias se fez necessário e serão descritos nessa sessão.

O ambiente foi desenvolvido e executado em um computador contendo o *Dual Boot*, onde possui instalados o sistema operacional Windows 10, além do sistema linux, este e somente este, utilizado como o ambiente de desenvolvimento deste trabalho. A tabela 1.1

mostra a configuração da máquina utilizada para o desenvolvimento e execução do ambiente proposto, levando em consideração apenas o sistema linux.

Tabela 1.1: Configuração da máquina

Tipo	Notebook
Marca	Dell®
Modelo	Inspiron 14 i14-5458-B08P
Processador	Intel® Core™ i3-5005U CPU @ 2.00GHz x 4
Memória RAM	3,8 GB
Placa Gráfica	Intel® HD Graphics 5500 (Broadwell GT2)
Disco Rígido	453 GB
Sistema Operacional	Linux
Versão	Ubuntu 18.04.1 LTS
Arquitetura	X86-64
Kernel	Linux 4.15.0-38-generic

Alguns *softwares* foram utilizados para o desenvolvimento desse trabalho, dentre os quais, destacam-se o Ptolemy II, utilizado para o modelo de estratégias de voos, o SITL/Ardupilot, para a representação visual e fornecedor dos dados de telemetria das RPAs durante a simulaçao e o HLA como *middleware* de comunicação e sincronização. Para o Ptolemy II, foi utilizado a linguagem de programação JAVA. Já para o HLA e o SITL/Ardupilot utilizou-se o Python. A tabela 1.2 mostra os *softwares* utilizados junto às suas respectivas versões.

Tabela 1.2: Softwares utilizados

Software	**Versão**	**Descrição**
Java Development Kit	1.8.0_191	Linguagem de programação
Eclipse IDE for Java Developers	2018-09 (4.9.0)	Ambiente de Desenvolvimento Integrado
OpenStreetMap API	JMapViewer 2.6	Componente que fornece mapas e dados geográficos
GeoTools	JTS 1.14	Biblioteca que provê modelo de dados geométricos
Python	2.7.15rc1	Linguagem de programação
Pandas	0.20.3	Biblioteca para análise de dados para python
Ptolemy II	11.0.devel	Ferramenta de modelagem e simulação
CERTI/RTIG	3.5.1	Ponte de comunicação entre sistemas heterogêneos
PyHLA	1.1.1	Módulo da linguagem Python para utilização do HLA
MAVProxy	1.5.2	Estação de Controle no Solo
DroneKit-SITL	3.2.0	Simulador que provê dados de telemetria

1.3 Estrutura da Dissertação

A estrutura desse trabalho foi organizado como consta a seguir.

O capítulo 1 apresenta o problema que serviu como base para essa pesquisa, fundamentada através da introdução, bem como os objetivos que nortearam esse trabalho e a metodologia empregada.

O capítulo 2 demonstra o que há de relevante nas pesquisas focadas na área tema desse trabalho.

No capítulo 3 todo o embasamento teórico necessário para o entendimento e funcionamento desse trabalho junto com as principais tecnologias utilizadas são apresentados.

O capítulo 4 apresenta o ambiente de simulação proposto e mostra como foi seu desenvolvimento.

Já no capítulo 5 um estudo de caso é apresentado e os resultados obtidos são detalhados e descritos.

O capítulo 6 apresenta as considerações finais e os trabalhos futuros.

Capítulo 2

Trabalhos Relacionados

Devido ao crescente interesse da industria e da comunidade acadêmica em sistemas com múltiplas RPAs, já existem um número bastante elevado de publicações que demostram técnicas de simulações para tais sistemas. Estes trabalhos destacam a importância das simulações para o desenvolvimento de vários aspectos de sistemas com múltiplos RPAs, como por exemplo, trajetórias das RPAs, comunicação, consumo de energia, etc, bem como apresentam novas soluções que abordam as necessidades específicas para este tipo de simulação.

Já citados anteriormente os trabalhos de [Barros et al. 2016] e [Brito et al. 2013], juntamente com o trabalho de [Barros et al. 2017] serviram como base para o desenvolvimento desse trabalho. Apesar das significantes contribuições para a comunidade científica, estes trabalhos foram desenvolvidos visando-se apenas um *drone* em seus ambientes de simulação, enquanto que a proposta desenvolvida neste trabalho serve para múltiplas RPAs.

Um sistema de simulação de eventos discretos de computação paralela de alto rendimento é proposto por Corner e Lamond [Corner e Lamont 2004]. Este sistema foi implementado usando uma abordagem em camadas, onde o ambiente paralelo SPEEDES atua como a camada mais baixa, e o topo da camada consiste de um modelo de comportamento de enxames [Kadrovach 2003]. Embora a proposta seja significante para a área de simuladores de múltiplos RPAs, a implementação dos autores não conseguiu atingir a velocidade desejada da abordagem paralela que foi empregada.

Wei, Madey, and Blake apresentaram um *framework* de controle baseado em agentes [Wei, Madey e Blake 2013], que emprega vários agentes RPAs para agendamento de tarefas locais. Uma aplicação de simulação foi desenvolvida como prova de conceito para o

framework proposto. Embora a ferramenta de simulação tenha sido útil para a validação do *framework*, não é provável que ela possa ser facilmente estendida para simular outras abordagens que não a proposta pelos autores.

Zema *et al*. introduziu uma arquitetura para um simulador que é capaz de implementar sistemas de controle de redes distribuídas, chamado CUSCUS (*CommUnicationS-Control distribUted Simulator*) [Zema et al. 2017]. A arquitetura proposta aproveita duas soluções já validadas: o simulador FL-AIR (*Framework libre AIR*) e NS-3. A integração estre os simuladores foi feita via uma estrutura de dados de memória compartilhada. Uma desvantagem deste tipo de integração é o fato que esta é uma maneira não padronizada de integrar diferentes simuladores, o que pode dificultar a realização te integrações adicionais.

D-MUNS é um simulador de redes distribuídas para múltiplos UAVs [La, Park e Kim 2017]. O sistema utiliza dois simuladores: RotorS e NS-3. Um novo algorítimo de sincronização em tempo real é proposto para solucionar o problema de sincronização entre simuladores distintos. O trabalho citado difere-se do proposto pelo fato de que foi utilizado um componente bem conceituado como forma de sincronização entre simuladores diferentes, o HLA.

Mason *et al.* propos um *framework* formal para analisar veículos aéreos autômatos adaptativos [Mason et al. 2017]. O *framework* é construído em três componentes principais: Uma especificação formal executável do comportamento de um UAV, um simulador de UAV e um verificador de modelo estatístico. Apesar de algumas similaridades com o ambiente proposto, esse trabalho não possui um sistema de detecção de colisões.

Um simulador de alta fidelidade para um UAV quadricóptero usando ROS e Gazebo é proposto por Zhang *et al.* [Zhang et al. 2015]. O simulador proposto implementa modelos dinâmicos e navegação do sistema além de outros recursos. Por outro lado, esta proposta não inclui suporte para sistemas com múltiplos *drones*.

Siddiqui *et al.* propos o desenvolvimento de um simulador para enxame de *drones* que integra modelos realistas de controle de movimentos para operações em desastres [Siddiqui et al. 2017]. O simulador foi desenvolvido em uma conhecida ferramenta de desenvolvimento de jogos a Unity 3D. O objetivo deste trabalho é treinar e avaliar os operadores de *drones* em situações de desastres. Entretanto, o simulador não trabalha com rotas de voos automatizados e também não avalia as melhores estratégias utilizadas nos sucessos das mis-

sões.

Como o restante deste trabalho descreverá, a solução proposta deste trabalho é capaz de integrar diferentes simuladores bem validados para compor uma única simulação paralela capaz de fornecer resultados com um alto nível de realismo.

Capítulo 3

Fundamentação Teórica

3.1 Ptolemy II

Ptolemy II é uma ferramenta de código aberto de modelagem e simulação de sistemas heterogêneos que fornece um laboratório útil para experimentar tecnologias de design de sistemas Cyber-físicos [Ptolemaeus 2014]. Foi desenvolvido por pesquisadores do departamento de engenharia elétrica e ciência da computação da universidade da califórnia em *Berkeley*. Ele integra um número bastante razoável de domínios semânticos, que atuam como as leis físicas que interagem entre os componentes e provê regras de execução concorrente e de comunicação entre componentes. O conjunto desses domínios são chamados de Modelos de Computação (MoC) [Ptolemaeus 2014]. O Ptolemy II utiliza um componente chamado diretor para representar um MoC e implementar suas regras de comportamento.

O Ptolemy II é baseado em modelos de classes orientados a atores, que são componentes que executam de maneira concorrente e se comunicam, trocando informações através de portas[Ptolemaeus 2014]. Atores podem possuir portas de entradas e de saídas simples, quando há apenas um canal de comunicação ou múltiplas quando existirem vários canais de comunicação, e são regidas por um diretor que definirá as regras de comportamento. Eles se interligam através de suas portas com outros atores, formando uma relação, podendo trocar dados entre si. Além disso, os atores podem ser definidos como atômicos ou compostos, ou seja, um ator constituido de atores e, possivelmente, um diretor, formando um conjunto de submodelos. A figura 3.1 mostra um exemplo de modelo feito no ptolemy II. O diretor, componente na cor verde, representa o domínio e gerencia as regras de comportamento. O

ator A possui uma porta de saída e se conecta através de uma relação com outros dois atores B e C, que possuem portas de entradas. Já a figura 3.2 ilustra o funcionamento de atores compostos. O ator composto A possui em sua composição um diretor, que rege as regras locais, e um ator, denominado ator atômico D, que se interconecta com uma porta de saída. Esse ator se liga através de uma ligação com outros dois atores, um ator simples, o ator atômico B, e outro composto, o ator composto C, que por sua vez, possui uma porta de entrada ligada a um ator atômico E e um atributo com um valor qualquer.

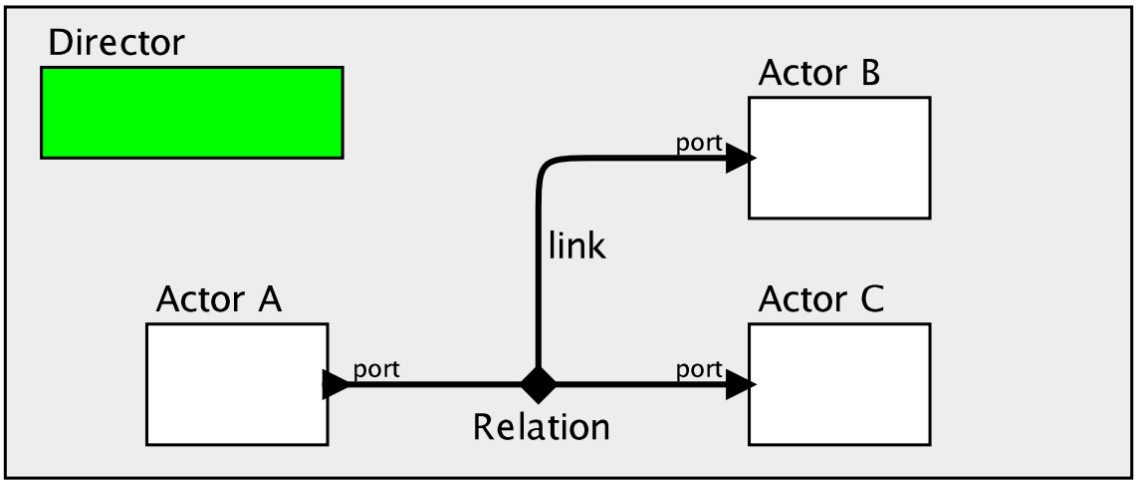

Figura 3.1: Exemplo de um modelo do Ptolemy II (Fonte: [Ptolemaeus 2014])

O conjunto de mensagem trocadas pelos atores é chamado de sinal. Um ator manda um sinal através de sua porta de saída, que é recebido por outro ator pela porta de entrada, desde que haja uma relação entre eles. Um sinal pode ser representado por vários tipos de dados, como caracteres, inteiros, pontos flutuantes, etc.

O processo de execução de um ator passa por três fases distintas: *setup*, *iterate* e *wrapup* [Lasnier et al. 2013]. A fase *setup* é dividida em duas subfases: *preinitialize* e *initialize*. A ação *preinitialize* é executado somente uma vez no início da simulação. Ela é responsável por qualquer operação que pode influenciar ou modificar a arquitetura ou impactar na análise estática. Faz parte das ações *preinitialize*, por exemplo, a checagem de tipos, geração de códigos, agendamentos, etc. Já a ação de *initialize* reinicializa estados locais, inicializa parâmetros e envia dados iniciais para portas de saídas.

A fase *iterate* é a principal fase de execução do modelo. Possui três subfases: *prefire*, *fire* e *postfire*. Na ação *prefire* as condições predefinidas, se houver, para a execução da ação

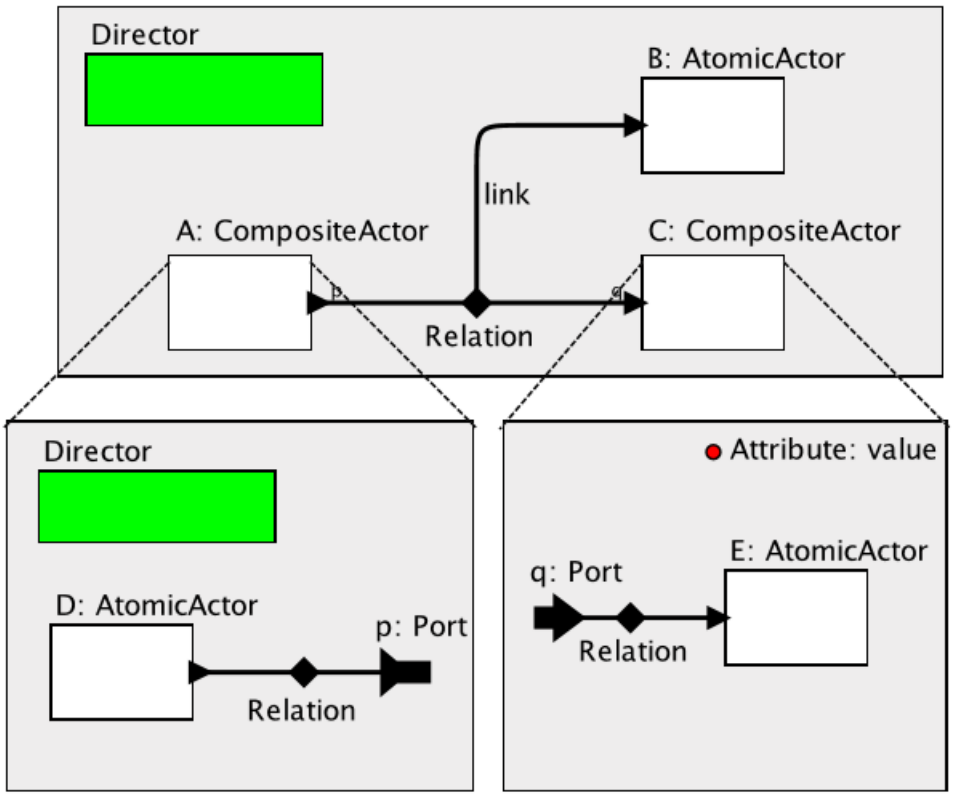

Figura 3.2: Exemplo de atores compostos no Ptolemy II (Fonte: [Ptolemaeus 2014])

fire são verificadas. Já a ação *fire* é a responsável pelas demandas computacionais como ler dados da porta de entrada, processa os dados e gerar dados para a porta de saída. Em contrapartida, a ação *postfire* serve para atualizar o estado do ator.

Por fim, a fase *wrapup* encerra o ciclo do ator. Mesmo que exista falhas de execução, esta fase tem garantia de executar.

3.2 Software in the Loop

O *Software in the Loop* (SITL) é um simulador de código livre desenvolvido de forma a ser usado em vários sistemas operacionais. O SITL permite a simulação de uma série de veículos, a saber, veículos aéreos com múltiplos rotores, veículos aéreos com asa fixa, veículos terrestres e veículos náuticos [SITL Simulator (Software in the Loop) 2018]. Foi desenvolvido a partir de um compilado de códigos escritos na linguagem de programação C++ de um sistema de piloto automático conhecido como Ardupilot, que permite representar o comportamento dos veículos sem o auxílio de qualquer equipamento especial.

O SITL/Ardupilot permite a conexão de vários *Grounds Controls Stations* (GCS). GCS é um *software* executado em um computador fisicamente localizado em solo que se comunica com a RPA através de telemetria via rede sem fio. Ele mostra em tempo real informações acerca da performance e posições da RPA [Choosing a Ground Station 2018].

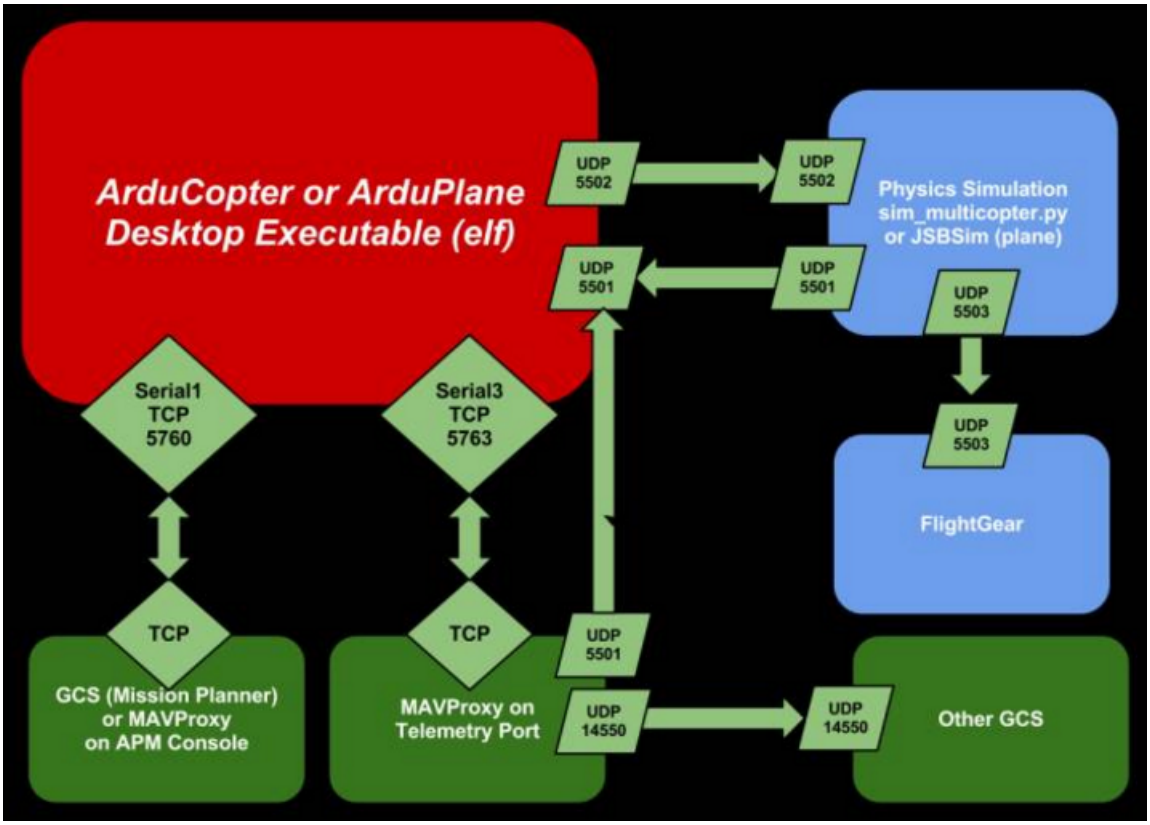

Figura 3.3: Arquitetura do SITL (Fonte: [SITL Simulator (Software in the Loop) 2018])

O GCS mas comumente usado junto ao SITL/Ardupilot é o MAVProxy. Segundo [MAVProxy 2018] MAVProxy é uma interface de linha de comando escrito na linguagem python, portável e extensível, que possui módulos gráficos e oferece suporte ao protocolo MAVLink [MAVLink Developer Guide 2018]. Pode-se notar, através da figura 3.3, que demostra a arquitetura do SITL/Ardupilot, que a comunicação entre o GCS e o SITL/Ardupilot ocorre pelo protocolo de rede TCP, através da porta 5760. Já na porta 5763 ocorre a comunicação via telemetria entre o MAVProxy e o SITL/Ardupilot. Ainda, existe a possibilidade de utilização do protocolo UDP, para a conexão de outros simuladores e de outros GCSs, utilizando o MAVProxy como ponte.

A API DroneKit-Python permite aos desenvolvedores criar aplicativos que rodem em

computadores embarcados e se comuniquem com o controlador de voo do Ardupilot utilizando um enlace de baixa latência [Introducing DroneKit-Python 2019]. A comunicação com a RPA se dá atravez do protocolo MAVLink. É uma ferramenta poderosa escrita na linguagem Python capaz de fornecer informações sobre o *drone* através de telemetria e, inclusive, controle direto sobre o veículo.

3.3 High Level Architecture

O *High Level Architecture* (HLA) foi desenvolvido pelo Departamento de Defesa dos Estados Unidos com a finalidade de fornecer uma arquitetura comum para prover uma estrutura que suportasse reuso e interoperabilidade entre diferentes simuladores [Dahmann, Fujimoto e Weatherly 1997]. Seu desenvolvimento se deu pela motivação de que sistemas complexos de simulação, por vezes, possuem componentes heterogêneos, *hardwares* ou *softwares*, sendo que reescrever ou mesmo desenvolver códigos em toda adição ou troca de componentes seria uma tarefa dispendiosa e difícil. A partir de 2010 o HLA virou um padrão mantido pelo *Institute of Electrical and Electronic Engineers* (IEEE). É definido por três documentos: o primeiro trata da estrutura geral e das regras principais [Committee et al. 2000], o segundo diz respeito às especificações da interface entre o simulador e o HLA [IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Federate Interface Specification 2010] e o terceiro é o modelo para especificações de dados (OMT) transferidos entre os simuladores [IEEE 2010].

Na terminologia do HLA, o conjunto de subsistemas heterogêneos que compõe o ambiente geral de simulação é chamado de Federados. Já o conjunto de federados que englobam e fazem parte da simulação trocando dados via *Run-Time Infrastructure* RTI e regidos por um *Object Model Template* OMT, compõem uma Federação. Ainda o *Federation Object Model* (FOM) é um documento que especifica as classes de objetos e classes de interação, juntamente com seus atributos e parâmetros, que são usados para a troca de dados entre os federados.

O HLA provê uma infraestrutura de serviços padronizada através de uma interface comum chamada *Run-Time Infrastructure* (RTI). Este é um *middleware*, que faz o intermédio da comunicação entre os federados, independente de qual linguagem de programação eles

tenham sido projetados. Ele prove diversos serviços dentre eles incluem a sincronização e trocas de dados entre os federados. A tabela 3.1 mostra o conjunto de interfaces juntamente com alguns dos serviços providos pelo RTI para os federados que compõem uma federação. Já a figura 3.4 mostra a arquitetura do HLA. Cada federado pode representar uma instância de um sistema heterogêneo e implementa uma interface HLA, que se conecta com o RTI, para trocar dados entre os federados de uma mesma federação.

Tabela 3.1: Serviços providos pelo RTI

Nome	**Serviços**
Federation Management	- Execução da Federação; - Salvamento da Federação; - Restauração da Federação;
Declaration Management	- Declaração de Objetos; - Declaração da Interação;
Object Management	- Registro de Objetos; - Atualização de Valores de Objetos; - Troca de Interações;
Data Distribution Management	- Criação de Regiões; - Registro de Objetos nas Regiões; - Troca de Interações entre as Regiões;
Ownership Management	- Adquire ou libera propriedades de atributos; - Notifica mudanças de propriedades;
Time Management	- Políticas de gerenciamento de Tempo; - Avanço do Passo no Tempo; - Avanço Baseado em Eventos; - Avanço Otimizado; - Gerencia de Filas; - Sincronização da Federação;

Uma das principais preocupações quando se fala em cossimulação é a questão de sincronização de mensagens. Em sistemas deste tipo não pode haver atrasos na entrega de mensagens, pois pode influenciar negativamente o andamento da simulação. Um exemplo

clássico desse problema é o do tiro que dobra a esquina: em uma simulação tem-se dois federados. O federado A (atirador), dispara um tiro, no tempo 1.0 da simulação, contra o federado B (alvo). Neste tempo o alvo se move, escondendo atrás de uma parede. Devido a atraso na entrega de mensagens, o alvo recebe a mensagem do tiro no tempo 2.0 da simulação, dando a impressão que a bala contornou a parede para atingir o alvo. O HLA possui mecanismos de gerencia de tempo que mitigam problemas como esses.

Existem dois tipos de ordenamento de mensagens: *Receive Order* (RO) e *Time Stamp Order* (TSO). Mensagens RO são enfileiradas assim que chegam, mesmo desordenadas e são elegíveis para entrega. Já em mensagens TSO, um carimbo de tempo marca as mensagens designadas aos federados. As mensagens também são enfileiradas, mas não são elegíveis para a entrega até o RTI garantir que todas as mensagens TSO destinadas ao federado não possuam registro de tempo menor, e nenhuma outra mensagem possa chegar em um momento posterior. Existe ainda dois mecanismos principais para avanço no tempo de simulação: *Time Advance Request* (TAR) e *Next Event Request* (NER). "TAR é bem adequado para federados que usam internamente um mecanismo escalonado no tempo, e o NER é o primitivo preferencial para federados baseados em eventos"[Fujimoto 1998]. Para indicar que o tempo de simulação avançou o HLA usa o *Time Advance Grant* (TAG).

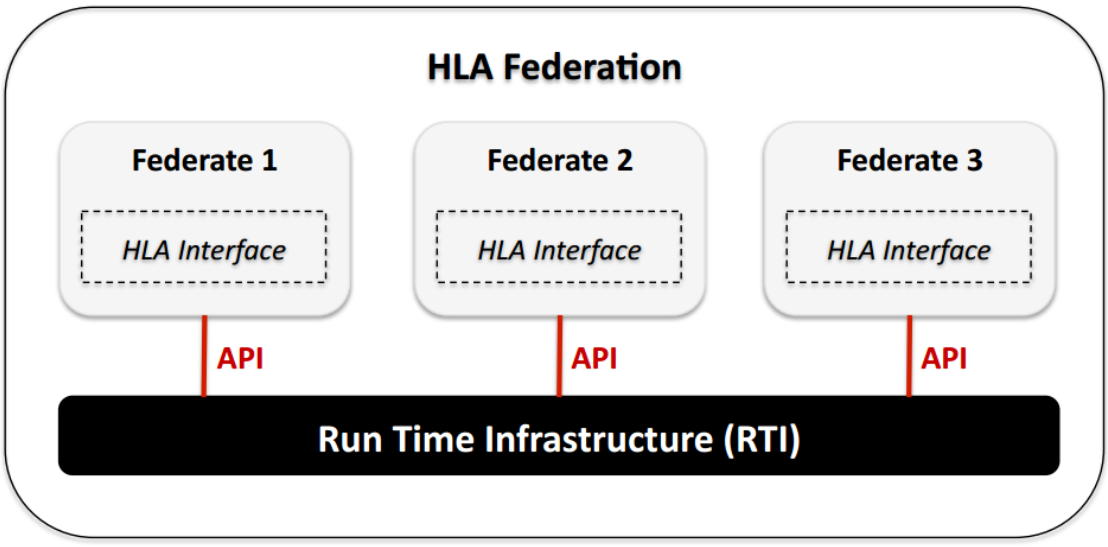

Figura 3.4: Arquitetura do HLA Fonte: [Lasnier et al. 2013]

Como se trata de um padrão especificado e consolidado existem várias implementações do HLA. Para esse trabalho foi escolhido o CERTI HLA. CERTI é um RTI de código aberto,

compatível com HLA, e foi desenvolvido pelo centro de pesquisa de Onera em Toulouse, nos laboratórios aeroespaciais franceses [CERTI DOC Introduction 2018].

Para o uso do CERTI, é necessário a configuração de variáveis de ambientes no sistema. Dentre algumas variáveis que devem ser configuradas estão o CERTI_HOME, local base de instalação do CERTI no sistema, CERTI_FOM_PATH, que indica os locais onde o algorítimo de busca irá procurar o FOM, CERTI_HOST, que indica em qual máquina o CERTI está sendo executado, entre outros.

O CERTI usa como componentes principais duas ferramentas de execução: o *Run-Time Infrastructure Ambassador* (RTIA) e o *Run-Time Infrastructure Gateway* (RTIG). O RTIG é o responsável pela interação dos federados com o RTI. Além disso, o RTIG fornece aos federados, através do RTIA, o arquivo FOM para se criar ou se juntar a uma federação [CERTI DOC Introduction 2018]. Para uma federação é necessário pelo menos um RTIG. Existe um RTIA para cada federado. O RTIA funciona como porta voz dos federados enviando suas demandas para o RTIG. A figura 3.5 mostra como o CERTI funciona.

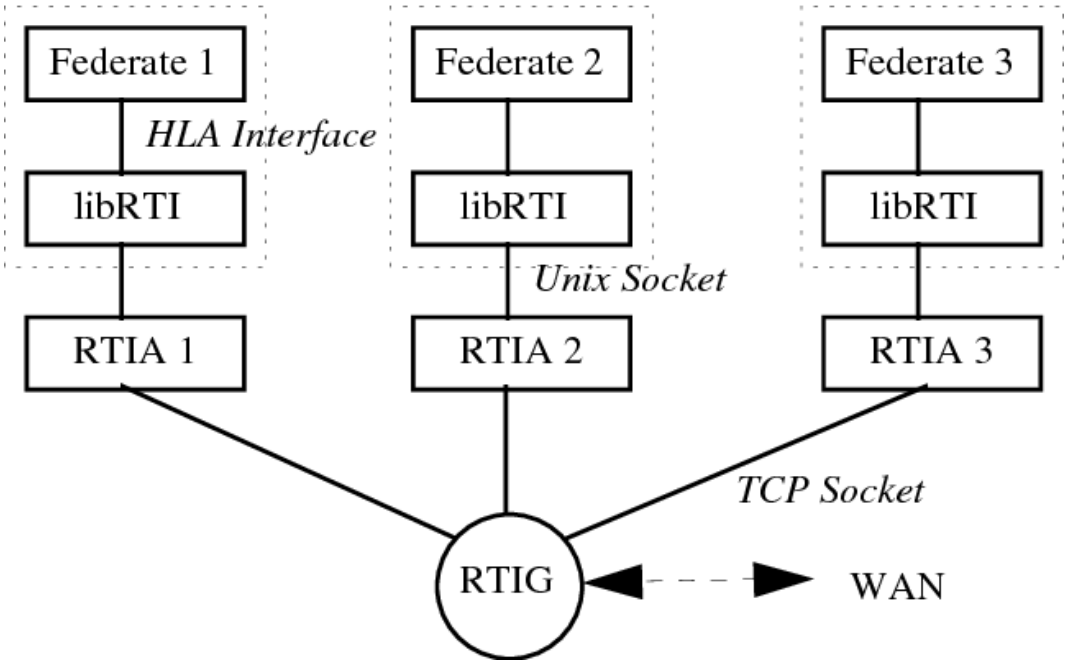

Figura 3.5: Arquitetura do CERTI. Fonte: [Noulard, Rousselot e Siron 2009]

O CERTI HLA provê diversos serviços através do *FederateAmbassador* e o *RTIAmbassador*, ou seja, alguns serviços são invocados pelo federado e outros pelo RTI. A tabela 3.2 mostra alguns dos serviços disponíveis pelo CERTI juntamente com uma descrição simples.

Tabela 3.2: Alguns serviços providos pelo CERTI HLA

Serviços	**Descrição**
createFederationExecution()	cria uma federação
joinFederationExecution()	junta-se a uma federação
resignFederationExecution()	sai de uma federação
destroyFederationExecution()	destrói uma federação
registerFederationSynchronizationPoint()	registra um ponto de sincronização
synchronizationPointRegistrationSucceeded()	êxito em criar registro de sincronização
announceSynchronizationPoint()	informa ponto de sincronização
synchronizationPointAchieved()	ponto de sincronização alcançado
federationSynchronized()	federação sincronizada
tick()	informa ao RTI que pode receber chamadas de retorno
publishObjectClass()	publica uma classe
subscribeObjectClassAttributes()	assina uma classe
unsubscribeObjectClass()	cancela a assinatura de uma classe
unpublishObjectClass()	cancela a publicação uma classe
registerObjectInstance()	registra a instância de objetos
updateAttributeValues()	envia e atualiza valores
reflectAttributeValues()	recebe atualização de valores
enableTimeRegulation()	torna o federado um regulador de tempo
timeRegulationEnabled()	êxito ao tornar o federado como regulador
enableTimeConstrained()	torna o federado restrito ao regulador
timeConstrainedEnabled()	êxito ao tornar federado restrito ao regulador de tempo
timeAdvanceRequest()	solicita avanço no tempo do federado
nextEventRequest()	solicita avanço no tempo do federado
timeAdvanceGrant()	informa que o avanço de tempo foi concedito

Capítulo 4

O Ambiente de Simulação

Com a utilização de softwares de simulações distintos, foi necessário implementar uma abordagem de cossimulação, afim de montar um ambiente que se aproxime ao máximo do mundo real. De acordo com Gomes *et al.* [Gomes et al. 2017], cossimulação consiste de teorias e técnicas para permitir simulação global de um sistema acoplado via a composição de simuladores .

Há basicamente dois sistemas de simulações, com funções diferentes, que foram usados para a criação da ferramenta proposta: O Ptolemy II e o SITL. Por se tratar de sistemas heterogêneos distintos, desenvolvidos com linguagens de programação diferentes, e a comunicação entre eles ser um fator de importante impacto, optou-se por usar um mecanismo que fosse mediador dessa comunicação chamado *High Level Architecture* (HLA).

O HLA foi utilizado como ponte de comunicação para troca de informações entre os sistemas heterogêneos. O SITL envia os dados da telemetria do *Drone* para o HLA, que por sua vez repassa para o Ptolemy II. Esses dados são processados e usados no modelo de decisão, que depende da estratégia que está sendo utilizada. O resultado do processamento informa qual será o próximo comando que deverá ser executado pela RPA e é transmitido através do HLA para o SITL.

A figura 4.1 mostra a arquitetura do ambiente de simulação proposto. Nela podemos verificar dois federados que fazem parte de uma federação. No Ptolemy II, que representa o federado da esquerda, foi usado a interface JCerti que implementa o HLA para a linguagem de programação Java. Em contrapartida, no SITL, representado na figura pelo federado a direita, foi usado a interface PyHLA, um módulo de interface que implementa o HLA para a

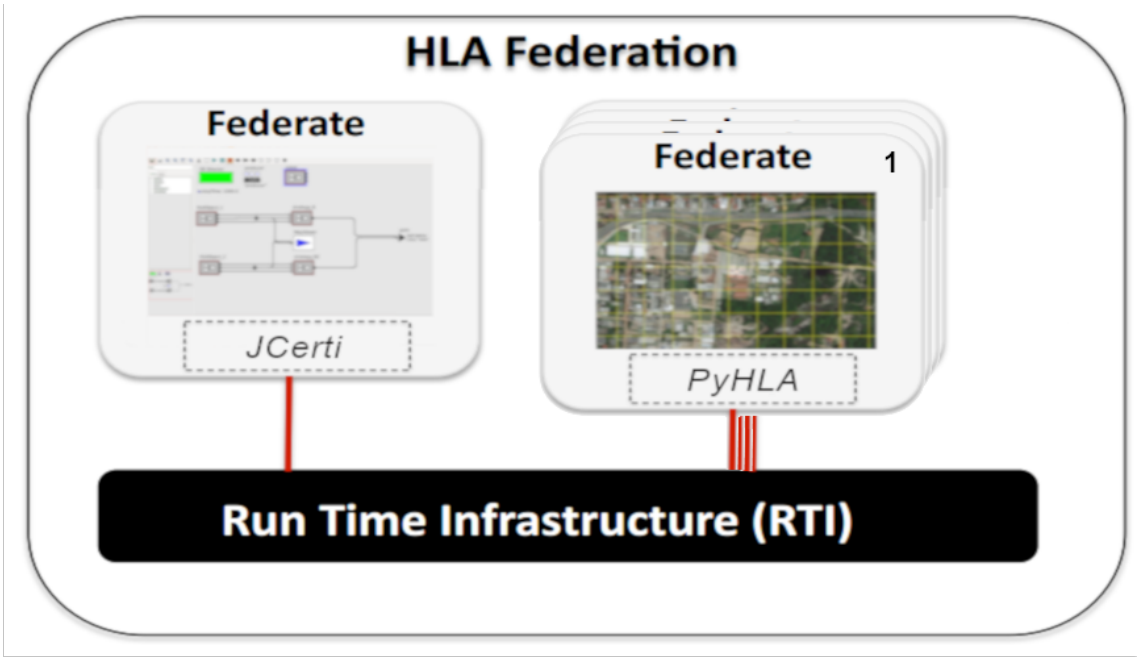

Figura 4.1: Arquitetura do ambiente de simulação

linguagem Python. Ambas as implementações fazem parte do projeto CERTI HLA. o RTI é o responsável pela sincronização e transmissão de dados entre os federados que compoem a federação.

Para o Ptolemy II foi utilizado um conjunto de atores para realizar a comunicação com o HLA: o *HlaManager*, 0 *HlaPublisher* e o *HlaSubscriber*. Esses atores podem ser encontrados na própria biblioteca do Ptolemy II.

O *HlaManager* é o responsável pela implementação do HLA no Ptolemy II. Ele foi renomeado como *producer* e pode ser visto na figura 4.2 identificado com o número 1. As configurações do *HLAManager* utilizadas no ambiente podem ser vista na figura na parte de cima identificada pelo número 4. Na configuração poe-se verificar o nome do federado *producer* e o da federação *federation1*. Em seguida tem-se o caminho do arquivo FOM, o PyhlaToPtolemy.fed, que indica as classes e atributos que deverão trocar dados entre os simuladores e o HLA. Depois vem o mecanismo para avanço no tempo. Foi utilizado o NER, pois o avanço no tempo do ambiente é baseado em eventos, onde cada comando de movimentação enviado para a RPA corresponde a uma unidade de tempo de simulação. Como o TAR não foi utilizado pulamos as próximas etapas. Em seguida vem a parte da sicronização. Foi definido como ponto de sincronização a palavra *ReadyToRun*. Isto significa o HLA deverá

esperar receber dos simuladores está palavra para sincronizar o tempo. Também foi definido a unidade de avanço de tempo do HLA em 1.0, ou seja, o tempo de simulação avança de um em um.

Na figura ainda, identificado pelo número 2, está o *HLAPublish*, nomeado como *goto*. Ele é o responsável por enviar e atualizar os dados para o HLA e funciona como uma porta de saída.

Ainda referenciado pela figura, identificado pelo número 3, pode-se ver um ator composto denomidado de *robot*, que é a representação da RPA. Internamente a esse ator, a direita da figura identificado pelo número 5, podemos ver três portas de entrada do *HLASubscribe*, responsável por receber os dados vindos do HLA. São três ao todo, o responsável por receber a identificação da RPA, a responsável por receber o posicionamento da RPA e a responsável por receber o nível de bateria da RPA. Os dados recebidos pelo *HLASubscribe* são repassados para a estratégia adotada.

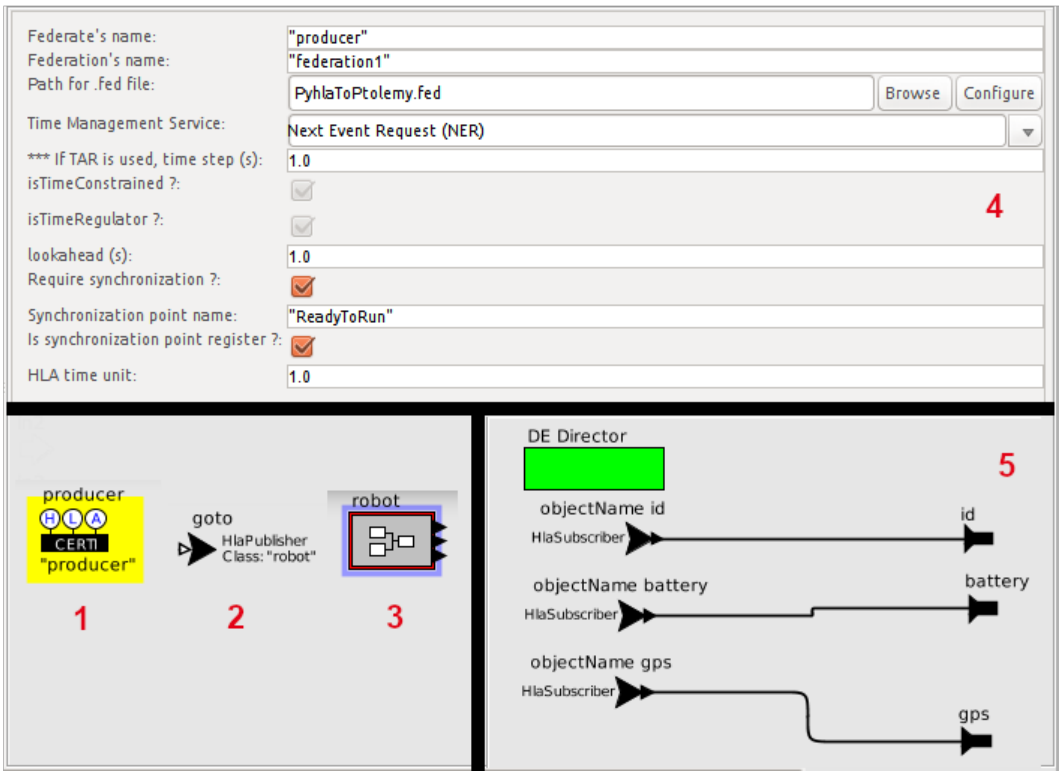

Figura 4.2: Atores HLA no Ptolemy II

Em contrapartida, foi utilizado um arquivo, codificado em Python, chamado *soft-constraints.py*. Esse arquivo é o responsável por iniciar o SITL e fazer a interligação dele

com o HLA. O *soft-constraints.py* funciona como um controlador, repassando as informações advindas do HLA para o SITL e enviando a telemetria recebida da RPA de volta para o HLA. Além de iniciar o SITL, juntar-se a federação que rege o ambiente e ser o intermediário entre a troca de dados do SITL com o HLA, o *soft-constraints.py* possui o módulo de decisão baseado na possibilidade de ocorrência de vento, do ambiente. Em outras palavras, quando o HLA repassa o próximo comando que a RPA deverá tomar, o módulo testa se naquele tempo o vento interagiu com a RPA, influenciando sua movimentação.

Dessa forma o código 4.1 mostra a forma como o SITL é instanciado no arquivo *soft-constraints.py* e também como é feita a conexão com o veículo. Na linha 1 é criado a variável *sitl*, que irá receber a instância do SITL. Em seguida são definidos alguns argumentos, necessários para a execução do SITL. São eles: -IO, parâmetro necessário para iniciar o SITL; *–model*, definido como *quad*, indica o modelo que a simulação usa; *–home*, indica o local onde a RPA decola e recebe os parâmetros de latitude e longitude; *–instance*, que indica a identificação da instância do SITL. Em termos gerais esse argumento que é usado como identificador da RPA; *–speedup* indica a velocidade que a simulação roda. Foi utilizado o valor 2.5 para agilizar o andamento das simulações.

Em seguida é instanciado as variáveis mavproxy e sitl. Então é indicado o uso do veículo quadricóptero, juntamente com a versão de e o SITL é iniciado. Logo após, a conexão com o Mavproxy é estabelecida, passando como parâmetros o tipo de veículo que será utilizado, o endereço e a porta de comunicação com o SITL, o mapa e o console de cada veículo e a instância, que serve como referencial para identificação da RPA.

Por fim ocorre a conexão do veículo que vai transmitir a telemetria das RPAs. A variável *connection_string*, na linha 9, indica qual o endereço do computador que será usado para realizar a conexão do protocolo mavproxy com o SITL. Neste caso a conexão se dá no próprio computador que está realizando a simulação. A forma de conexão se dá via *socket* que nada mais é que o endereço do computador + porta de destino. Em outras palavras o endereço identifica qual computador será usado para se fazer a conexão e a porta indica por onde trafegarão os dados daquela conexão. Dessa forma, cada RPA se conecta a uma porta diferente. De maneira a padronizar a qual porta o veículo se conectaria, foi definido o incremento de 10 ao número da porta por cada conexão. Exemplificando, a porta padrão para conexão é a 14550. No caso de utilização de dois veículos simultaneos na simulação, elas

usaram as portas 14560 e 14570, reespectivamente, para troca de informação. Em seguida, na linha 4 é onde a conexão do veículo é executada.

Código Fonte 4.1: código que executa o SITL e conecta a RPA

```
1     sitl = None
2     sitl_args = ['-I0', '--model', 'quad', '--home=' + args.home, '--instance=' + args.
          instance, '--speedup=2.5']
3     mavproxy = None
4         sitl = SITL()
5         sitl.download('copter', '3.3', verbose=True)
6         sitl.launch(sitl_args, await_ready=True, restart=True)
7         mavproxy = start_MAVProxy_SITL('ArduCopter', options='--sitl=127.0.0.1:5501 --out
              =127.0.0.1:19550 --map --console', instance=int(args.instance))
8
9     connection_string = '127.0.0.1:' + str(14550 + 10 * instance)
10    # Connect to the Vehicle
11    print 'Connecting to vehicle on: %s' % connection_string
12    vehicle = connect(connection_string, wait_ready=True)
```

O Ptolemy II foi utilizado como modelo de decisão das estratégias de voo. Houve necessidade de modificação e adaptação das estratégias desenvolvidas no trabalho mencionado [Barros et al. 2017]. As adaptações se deram de maneira lógica, outras de maneira estrutural e houve também mudança conceitual.

As adaptações consideradas lógicas, foram as alterações que o modelo de dados necessitou para se adequar a proposta deste trabalho. Estás alterações estão fortemente atreladas às mudanças estruturais. A figura 4.3 mostra a versão anterior do modelo, apresentado pelo trabalho que originou o desenvolvimento deste. O componente HLAObject_2 se relaciona com o componente strategy_A por intermédio de duas conexões. Já a saída do componente strategy_A se liga ao componente goto, responsável por transmitir os dados.

Em contrapartida a figura 4.4, mostra as mudanças lógicas do modelo de dados, necessárias para a adaptação deste trabalho. Nota-se que há um número maior de componentes em relação ao trabalho anterior, onde cada componente HLAObject representa uma RPA. Cada componente HLAObject se conecta conecta ao respectivo componente strategy_A por meio de três ligações. Existe ainda, o componente MapViewer que é ligado a uma das relações entre esses componentes. Por fim, as saídas dos componentes strategy_A se conecta ao componente goto.

Outra adaptação que precisou ser realizada foi nas estratégias. No modelo anterior, além

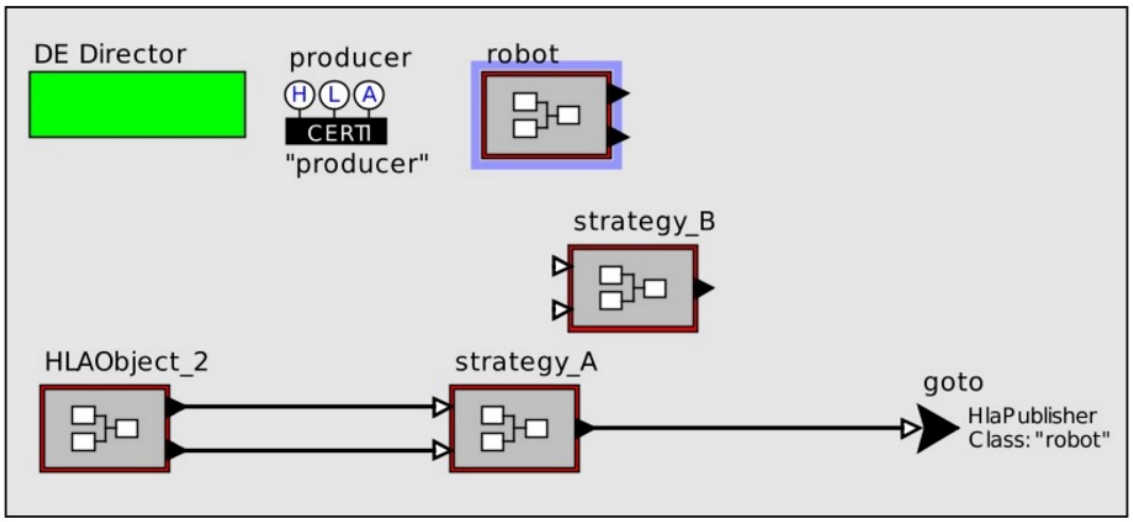

Figura 4.3: Exemplo de como era o modelo anterior (Fonte: [Barros et al. 2017])

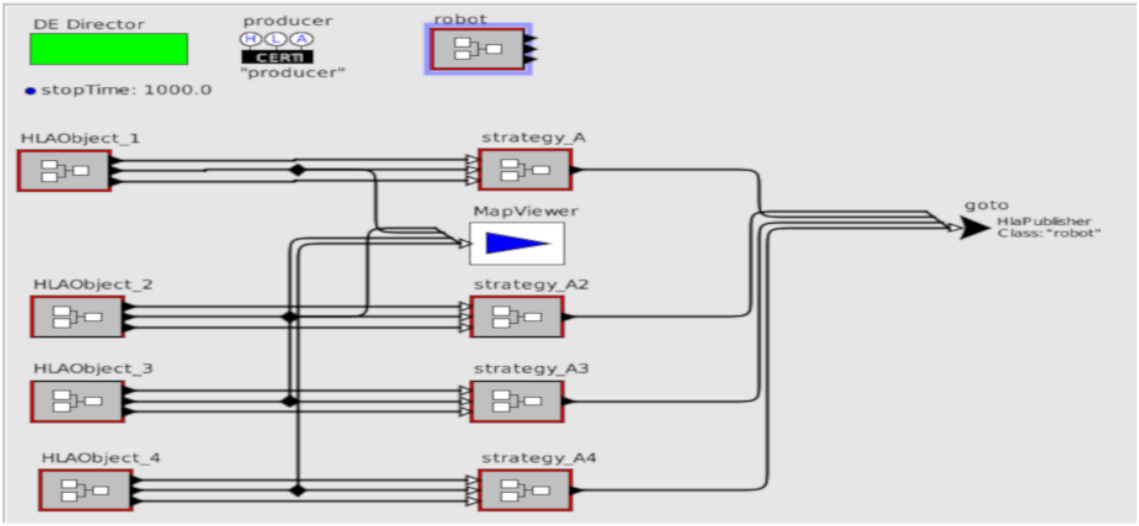

Figura 4.4: Exemplo de modelo modificado

de possuir menos portas de conexão e consequentemente ligações entre os componentes, possuia um componente, PlotterPosition, que registrava as posições do *Drone* em relação a sua trajetória como pode ser visto na figura 4.5.

Comparando com o modelo adaptado, existem mais conexões, no novo modelo. Isso se deu pela adição de uma porta de identificação da RPA, que trouxe mudanças estruturais em vários componentes. A adição desta porta de identificação não só modificou vários componentes do modelo, como também do simulador que fornece a telemetria e do interlocutor que gerencia a troca de mensagens entre os simuladores. O componente PlotterPosition, responsável pelo registro das posições também foi removido por questões conceituais. A figura 4.6, ilustra as modificações ocorridas em comparação com o modelo anterior.

Além dessas mudanças ocorreram também adaptações estruturais. Estas adaptações alteram o código fonte dos componentes que compoem o ambiente anterior para se encaixar na nova proposta. No componente robot e nas estratégias foram adicionados portas de identificação das RPAs, o que fez com que muitos componentes sofressem alterações em seus códigos. Além disso, o ambiente anterior utilizava funções de cálculos de distância de uma forma não consolidada. Em todos os componentes que utilizavam essas funções, foi utilizado uma biblioteca bem conceituada, que possui diversas funções de cálculos geométricos e de distâncias mais precisos. Houve necessidade de criação de novas classes para novos componentes inseridos neste trabalho. Também houve alteração de códigos nos arquivos da arquitetura de alto nível e no simulador responsável pelos comandos e telemetria dos dados. Ainda, foi desenvolvido um algorítimo de detecção de colisão e incluído características, como a inserção de áreas de restrição de voo. Por fim, foi desenvolvido um algorítimo de coleta dos dados da simulação, que separa as informações em tabelas e distribui para vários arquivos do tipo CSV, para melhor visualização.

Outra adaptação ocorrida foi a conceitual. No modelo anterior o *Drone* sobrevoava um ponto de visitação varias vezes, durante uma mesma simulaçao, até a bateria atingir o limite, onde retornaria pra base. No ambiente proposto ele visita o alvo apenas uma vez, e prossegue até todos os pontos de visitação tenham sido fotografados ou a carga de bateria chegue ao limite. Além disso o algorítmo que gera possibilidade de ocorrer vento, no trabalho base, lançava ventos de todos os lados no *Drone*, ou seja, os ventos às vezes vinham do norte e às vezes do sul, ou do leste. Isso foi modificado para um maior realismo, deixando o vento

atingir a RPA no sentido leste para o oeste, conforme a região escolhida para a simulação. A precisão da probabilidade de ocorrência de vento também foi melhorada.

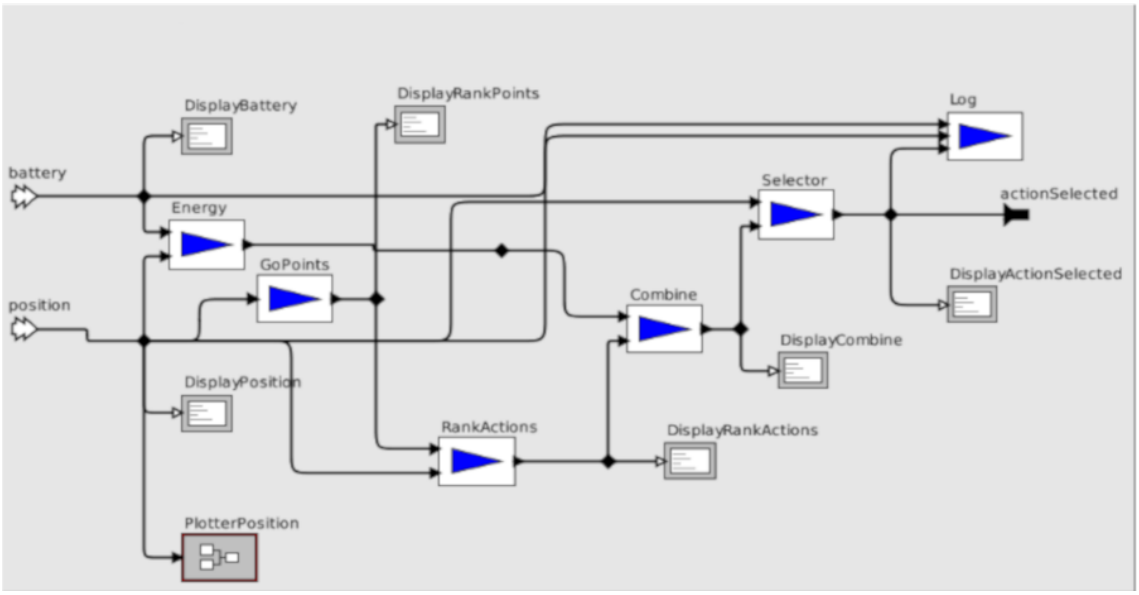

Figura 4.5: Exemplo de estrategia no modelo anterior

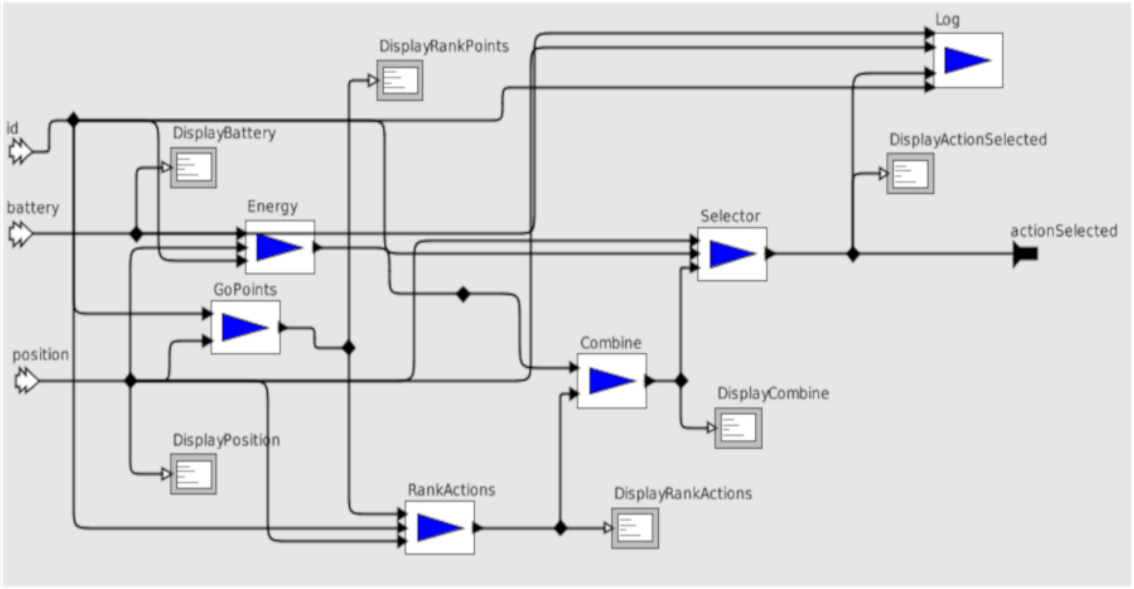

Figura 4.6: Exemplo de estratégia adaptada

Além disso, foi implementado um mapa, utilizando o componente JMapViewer, que precisou ser transformado em um ator, para visualização em tempo de simulação das RPAs, ilustrando suas trajetórias de voos, a fim de uma melhor análise de desempenho. O JMapViewer é uma API do *OpenStreetMap* (OSM), que é uma ferramenta aberta desenvolvida por

uma comunidade voluntária, que provê dados geográficos para centenas de sites e diversos dispositivos [OpenStreetMap 2019]. JMapViewer é desenvolvido em Java e é mantido como projeto independente do *Java OpenStreetMap* (JOSM) [JOSM 2019].

De forma a transformar essa API em um ator Ptolemy II foi preciso criar uma classe na linguagem Java, chamada MapViewer, e extender a subclasse TypedAtomicActor, usada na maioria dos atores atômicos do Ptolemy II. Adicionado a isso é necessário sobrescrever os métodos *preinitialize()* e *fire()*, podendo também sobrescrever os métodos *initialize()*, *prefire()*, *posfire()* e *wrapup()*. Esses são métodos padrões para qualquer ator do Ptolemy II.

O outro federado que fez parte do ambiente de simulação foi o SITL, devido sua habilidade de simular vários tipos de veículos e ainda prover dados de telemetria das RPAs. Para esse trabalho foi utilizado quadricópteros devido a facilidade com quem eles decolam e pousam, e por sua manobrabilidade. De forma simples, pode-se definir o simulador em três fases distintas: inicial, mediana e final. Para cada fase uma função: decolagem, voo e pouso.

Na primeira fase ou fase inicial, o SITL verifica todos os componentes fazendo a checagem de segurança. Então envia um sinal que informa que já está pronto para a decolagem. Então a RPA prossegue com a decolagem indo para cima a uma altura predeterminada e segue para a fase mediana. Na próxima fase o *Drone* move-se para frente com a finalidade de completar sua missão. Neste caso, o *Drone* recebe os comandos do Ptolemy II, via HLA, para definir o próximo movimento. Os comandos que o Ptolemy II envia para o SITL são: mvE, onde o movimento se dá para o leste, mvW, movimento para o oeste, mvN e mvS, respectivamente, move-se para o norte e para o sul. O comando completo que o Ptolemy II envia para o HLA para repassar para o SITL é UAVId + mvX, onde o UAVId é o identificador do *Drone* que deverá receber tal comando e o mvX especifica o movimento que o mesmo deverá seguir. Por fim, na fase final, a RPA recebe o comando para o pouso. Existe quatro momentos que a função pouso pode ser chamada. O primeiro momento é quando a RPA completa sua missão e retorna para a base. O segundo momento é quando a RPA chega no limite de 80% de consumo de bateria. Esse limite pode ser denominado como reserva de combustível. Neste caso, a RPA abandona sua missão e tenta retornar para a base. O terceiro momento ocorre quando chega no limiar de 95% de consumo de bateria. Nesse caso a RPA está em estado crítico de bateria. Para manter a segurança do equipamento, a RPA recebe o comando de pousar, independente do local que estiver. Seria uma espécie de pouso forçado.

Dessa forma, caso a RPA entre no limite de reserva de bateria a uma distância muito grande da base de retorno, e antes que consiga voltar, chegue ao nível crítico de bateria, o *Drone* pousa imediatamente. O último momento é quando há uma colisão. Neste caso o comando é usado para que a RPA saia da simulação.

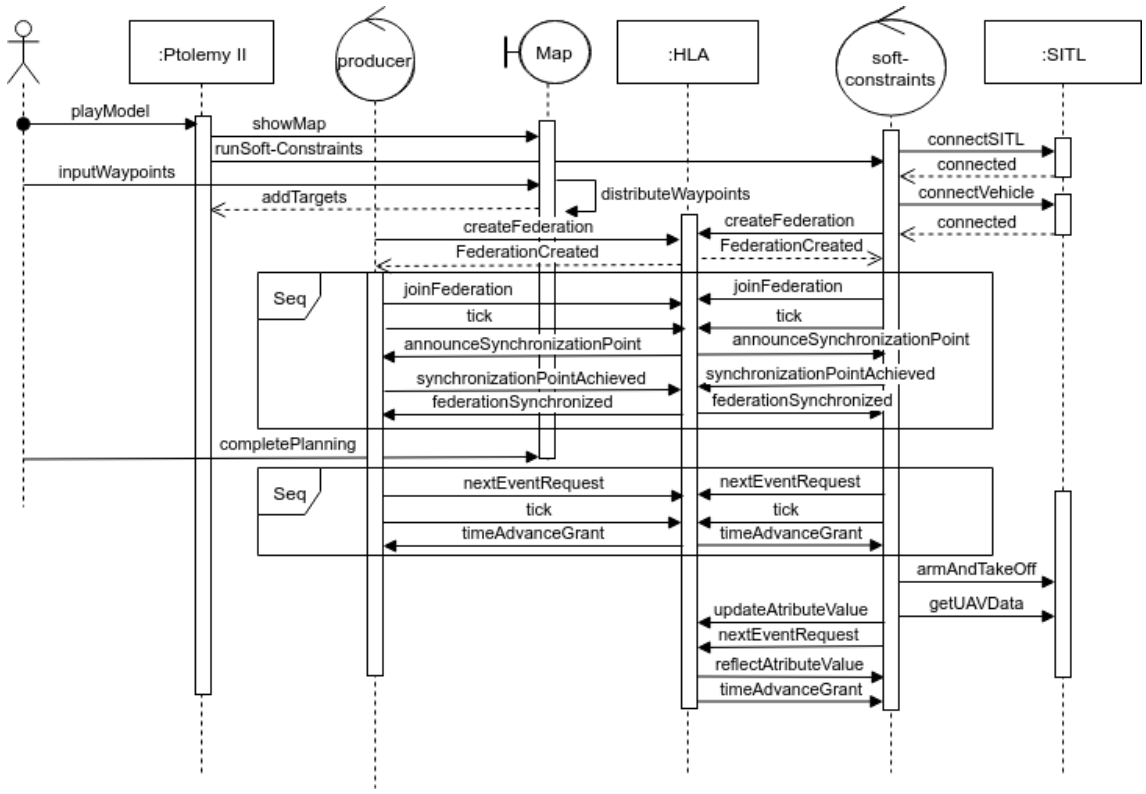

Figura 4.7: Diagrama de sequência de inicialização do ambiente Fonte: produção própria

O fluxo de mensagens sequenciadas do ambiente quando se inicia uma nova simulação está descrito no diagrama de sequência como mostra a figura 4.7. Pode-se observar, nesta primeira parte, o fluxo de interação desde o início até a fase onde a RPA está apta a decolar. A interação do usuário, isto é, da pessoa que inicia a simulação com o ambiente é mínima e se dá no início da simulação, na escolha dos pontos de visitação e nas áreas de restrição (se houver) e na conclusão do planejamento de voo. Se já houver pontos de visitação armazenados é ainda menor a interação, bastando iniciar o modelo e concluir o planejamento.

Depois de iniciado a simulação, o Ptolemy II exibe o mapa da API JMapViewer e envia o camando para executar os arquivos *soft-constraints.py*, responsáveis pela comunicação com o HLA no lado do SITL. Existe um componente no Ptolemy II, chamado *producer* responsável pela comunicação com o HLA também. Dessa forma são executados tantos

arquivos quanto RPAs utilizadas na simulação.

Por sua vez o *soft-constraints.py* faz a conexão com o SITL e conecta com o veículo (quadricóptero), sendo que cada arquivo se conecta a somente um veículo. Enquanto isso, o Ptolemy II cria a federação, se junta a federação criada e define um ponto de sincronização.

De maneira semelhante os arquivos *soft-constraints.py* em execução, tentam criar a federação, mas como ela já foi criada, apenas se juntam a ela e definem o mesmo ponto de sincronização já escolhido. Enquanto tudo isso é feito de forma transparente, o usuário pode escolher os pontos de visitação e, caso queira as áreas de restrição de voo. Depois de concluído o planejamento de voo,os federados enviam mensagens para o HLA solicitando o avanço do tempo de simulação. Essa troca de mensagens pode ser visto na segunda sequencia, onde os federados enviam o comando *nextEventRequest*, solicitando o avanço e logo em seguida *tick* avisando que aguarda resposta. Então o HLA envia o comando *timeAdvanceGrant* para os federados informando-os que podem avançar o tempo.

Dessa forma o *soft-constraints* envia o comando *armAndTakeOff* para o SITL solicitando que as RPAs decolem. Antes da decolagem, porém, o SITL faz uma série de checagem de verificação de segurança no veículo e então levanta voo na altitude informada previamente.

O *soft-constraints* recebe os dados da telemetria do SITL e repassa para o HLA a localização, no formato [latitude,longitude], o nível de bateria e a identificação do veículo, ao passo que solicita avanço no tempo e assim se tem início a simulação.

Já iniciada a simulação pode-se verificar através da figura 4.8 o diagrama de sequência de iterações entre os federados. Logo após decolar, o HLA já com os dados enviados via SITL, repassa para o federado do Ptolemy II. Então O Ptolemy II executa a estratégia definida no modelo para cada conjunto de dados recebidos. Dessa forma, como exemplo, os conjunto de dados com a identificação da RPA 1 são executadas separados e paralelamente aos do com identificação da RPA 2 e assim por diante.

O modelo com a estratégia recebe o conjunto de dados informando a localização, o nível de bateria e a identificação da RPA como entrada e transforma no próximo comando que a RPA executará, podendo ser comando de movimento, de captura de foto ou de pouso. Antes porém o Ptolemy II verifica se houve uma colisão com outra RPA ou com áreas de restrição. No caso de colisão detectada o comando é substituído pelo de pouso imediato e a RPA é retirada da simulação. Não havendo colisão o federado manda o próximo comando para o

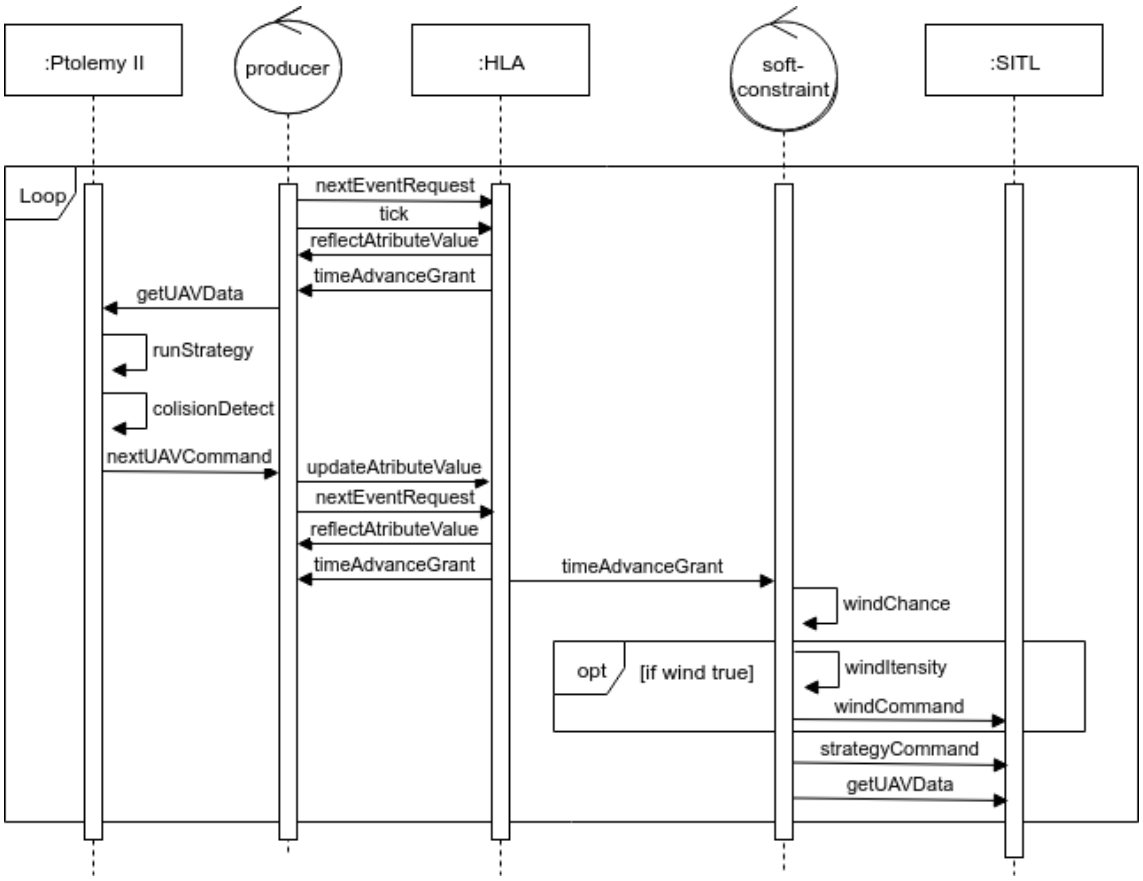

Figura 4.8: Diagrama de sequência de iterações. Fonte: produção própria

HLA através do *updateAtributeValue* e solicita avanço no tempo de simulação.

Uma vez que o avanço no tempo é executado o federado do SITL recebe o comando via HLA e executa o algoritmo de probabilidade de ocorrência de vento. Caso indique que ocorrerá vento nesta iteração é sorteado a intensidade do vento e sem seguida o comando com o vento é passado para o SITL. Exemplificando, se o próximo comando para a RPA fosse ir para o norte e ocorra vento forte, o comando que o SITL receberá é para deslocar a RPA para o oeste. Se não ocorra vento o comando recebido do Ptolemy II via HLA é executado. Em seguida um novo conjunto de dados é obitido e a iteração se inicia novamente, até a simulação terminar.

O SITL foi responsável também pela representação visual das RPAs. Entretando, a ferramenta tem uma limitação natural de simular somente um veículo por interface gráfica. Esse problema foi solucionado pela criação de diferentes instâncias para cada RPA utilizada. A figura 4.9 apresenta um exemplo de duas instâncias do SITL sendo executadas de forma simultânea, em modo gráfico, e representando, cada uma, quadricóteros distintos (direita). O Mapa ilustrado (esquerda) é o componente JMapViewer que mostra, em tempo de execução, as trajetórias que cada RPA segue.

Outro fator determinante para o ambiente ser mais realista, foi a inclusão de áreas de restrição de voos. Essas áreas são zonas delimitadas onde não é permitido o sobrevoo de RPA. Para o proposito deste trabalho essas áreas restritas funcionam como obstáculos físicos, isto é, áreas contendo edificações que bloqueiam a passagem das RPAs, e podem servir de ponto de impacto. Em contrapartida, o sistema permite o uso de áreas de restrição sem obstáculos físicos, ou seja, zonas proibidas para voos mas sem ponto de impacto.

Ainda no contexto de dar mais realidade ao sistema, foi introduzido um algorítmo de detecção de colisão. O algorítimo permite detectar colisão entre *Drones* ou entre um *Drone* com uma área restrita. Entre RPAs, o algorítimo funciona da seguinte forma: foi criado uma área circular configurável, em relação ao seu comprimento, ao redor das RPAs. Qualquer RPA que cruze essa área é detectado uma colisão. Para o caso de estudo proposto por esse trabalho, foi considerado uma área de 2,6 metros de comprimento ao redor das RPAs como limite de impacto. Essa área é considerado a distância máxima que uma RPA pode se aproximar de outra RPA sem que haja colisão. Na prática esse distância informa qual o limite máximo que uma RPA com capacidade de evitar colisão dispõe, para que não ocorra o

Figura 4.9: Exemplo de duas instâncias do SITL sendo executadas simultaneamente em modo gráfico

choque. Em contrapartida, o funcionamento do algorítimo para zonas com restrição de voo se dá de forma diferente. O algorítimo faz o cálculo da distância entre a RPA e as zonas. Qualquer RPA que intercepte a área de restrição, é considerado uma colisão. Quando ocorre uma colisão de qualquer tipo, o algorítimo força o pouso das RPAs envolvidas e as retira da simulação.

4.1 Configuração das Estratégias

Ptolemy II é uma ferramenta de modelagem que utiliza modelos orientado a atores. Atores são como classes de componentes designados para executar determinada função. A principal característica do uso de atores é a possibilidade do reuso de suas funções. Para esse trabalho, foi criado um ambiente capaz de simular múltiplos *Drones*. Foi utilizado duas estratégias distintas adapatadas e definimos qual estratégia usar em determinado momento.

Em relação aos componentes, foi utilizado um ator chamado robot para representar a RPA. Esse ator recebe três parâmetros: Id, utilizado para identificar qual RPA está transmitindo dados e para qual deve ser enviado so comandos; Battery, importante para registrar e medir o consumo de energia; e GPS, que informa a localização atual da RPA. Para cada RPA adicionado ao simulador é necessário adicionar instâncias do ator robot ao modelo. Essas intâncias são automaticamente nomeadas com *HLAObject_x*, onde o x é um valor numérico como pode ser visto na figura 4.10. Cada instância deve ser ligada a uma copia da estratégia usada. O ator *producer*, ao lado do diretor, é o responsável por implementar o HLA para o Ptolemy II. Pode-se notar ainda o parâmetro stopTime com o valor de 1000,0. Este parâmetro indica o tempo máximo que uma simulação pode atingir antes de finalizar. Na prática esse valor limite não é alcançado. Por fim, a saída das estratégias, são enviadas para o ator *goto*, que publica para o HLA repassar ao SITL.

Além disso, foram utilizados dois atores compostos, que são atores com submodelos implementados internamente, referente as estratégias que as RPAs deveriam usar durante as missões. O primeiro ator, chamado de Estratégia A, recebe os três parâmetros da instância da classe robot, e faz os cálculos de consumo de bateria e localização. Então, a Estratégia A encaminha a RPA para o próximo alvo, de acordo como foi escolhido no mapa, seguindo do primeiro ao último ponto, pela ordem predeterminada, retornando para a base em seguida.

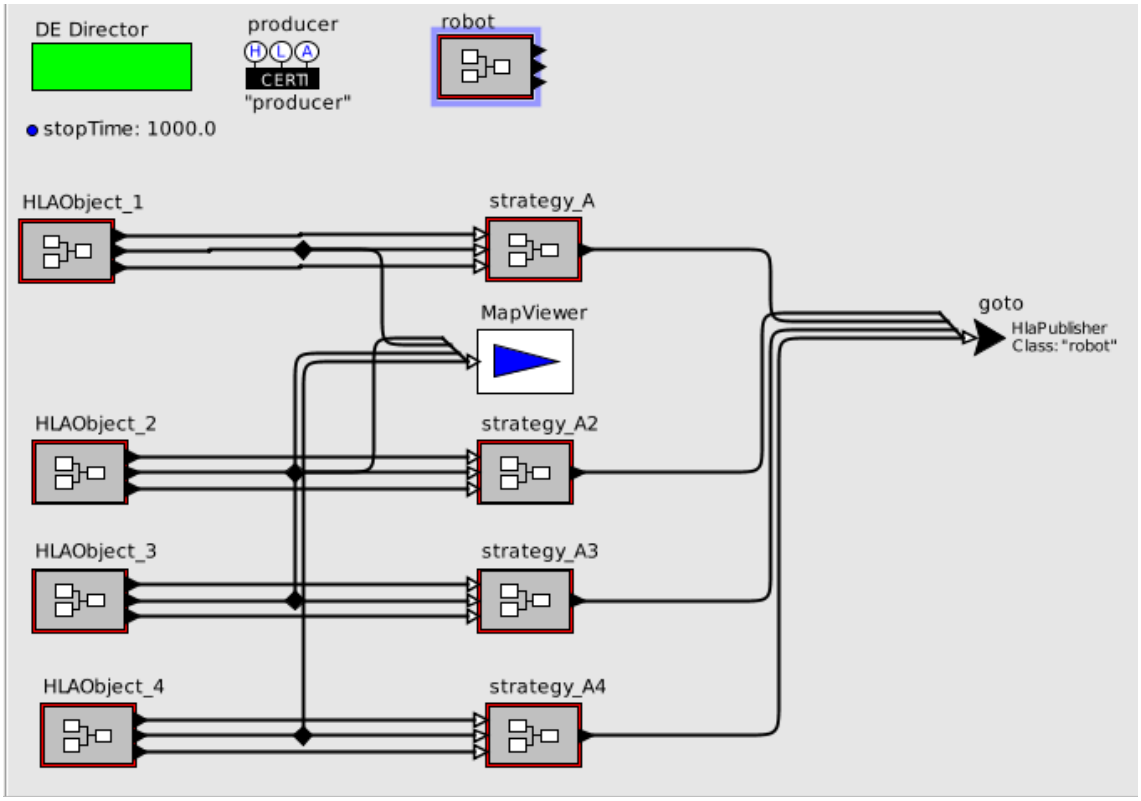

Figura 4.10: Exemplo de modelo criado para estratégia A

O ator chamado Estratégia B, possui atores adicionais em sua composição, que calculam a distância que a RPA percorrerá aos alvos e determina uma pontuação baseados na distância. Essa pontuação serve como base para a escolha do próximo alvo. A tabela 4.1 mostra os atores que compõem as estratégias, junto com as principais funções que cada ator executa.

Detalhando um pouco mais como cada estratégia funciona, com relação a seus atores interno, descrevemos as rotinas de processamento de cada estratégia. Tanto o atores composto que representa a estratégias A como o representante da estratégia B, possuem três portas de entradas e apenas uma porta de saída. Pelas portas de entrada eles recebem do ator robot, a identificação da RPA que envia os dados, a localização atual da RPA em forma de coordenadas geográficas latitudinais e longitudinais, e o nível de bateria restante da RPA.

Observando primeiramente a estratégia A, referenciada na figura 4.11, o ator *Energy* possui três portas de entrada, as quais recebem os dados repassados do ator *robot* para a estratégia A, e possui uma porta de saída. Esse ator indica se é possível executar determinada ação de movimento de acordo com a carga de bateria disponível e a localização atual da RPA. Para isso ele cria uma lista com as ações possíveis de realizar e dá uma pontuação de

Tabela 4.1: Atores utilizados nas estratégias e suas funções

Atores	**Estratégia**	**Funções**
Energy	A,B	- Verifica o estado da carga de bateria da RPA; - Verifica se a RPA pode executar determido comando de acordo com a carga disponível;
GoPoints	A	- Indica os pontos de visitação cadastrados em ordem predeterminada; - Verifica se alvo já foi visitado e passa nova localização para visitação;
RankPoints	B	- Cria uma pontuação dos pontos de visitação cadastrados; - Pontuação baseada na menor distância entre o ponto de visitação e a localização atual do *Drone*; - Resulta em uma lista ordenada pelos pontos de visitação mais próximos;
RankActions	A, B	- Cria uma lista ordenada com as ações que a RPA poderá tomar; - Lista baseada na localização atual e no ponto de visitação;
Combine	A, B	- Combina as ações enviadas pelos atores *Energy* e *RankActions*; - Ordena as ações dando prioridade àquelas advindas do ator *Energy*;
Selector	A, B	- Seleciona a primeira ação da lista enviada pelo ator *Combine* e envia para o HLA; - Verifica se houve colisões;
Log	A, B	- Registra as informações de voo das RPAs; - Informações como carga de bateria, localização, ações tomadas e distâncias percorrida;

1.0 ponto para aquelas que são possíveis de se executar e ainda retornar para a estação base. Já as ações que, excedem o limite de carga determinado para o retorno a base, recebem 0.5 pontos. Por fim, as ações que excederiam os 95% de consumo de bateria, ou seja, aquelas cujo retorno a base é inviável recebem 0.0 pontos. A lista com todas as ações de movimento com suas respectivas pontuações ordenadas de forma decrescente é a saída desse ator.

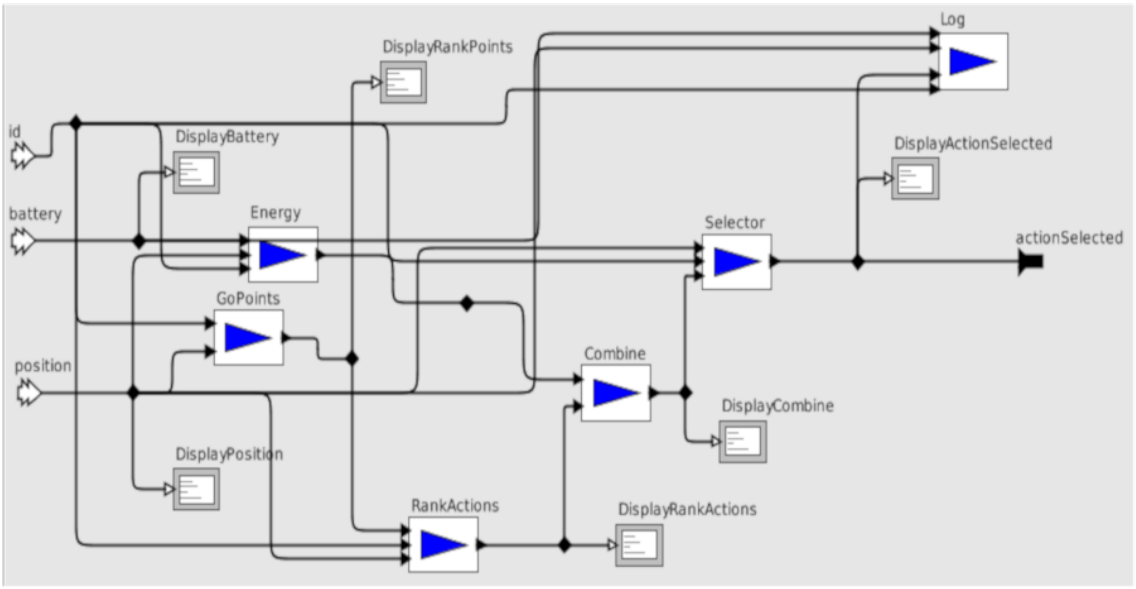

Figura 4.11: Visão interna detalhada da estratégia A

O ator *GoPoints* possui duas portas de entradas, as quais recebe a identificação da RPA e sua posição e uma porta de saída. Esse ator repassa o próximo ponto de visitação como para sua porta de saída. Esse ponto de visitação está ordenado de forma determinada, escolhido no início da simulação.

O ator *RankActions* dispõe de três portas de entrada, a identificação da RPA, a posição e a saída da porta do ator *GoPoints*. Esse ator é responsável por montar a lista com as ações de movimento de forma ranqueada, mas como para a estratégia A a RPA segue sempre para o próximo ponto determinado sem qualquer possibilidade de dinamismo com relação a trocas de pontos de visitação, esse ator repassa para sua porta de saída a próxima ação de movimento visando a conclusão da missão.

Já o ator *Combine* possui duas portas de entrada, que são as saídas dos atores *Energy* e *RankActions*, e uma porta de saída. Esse ator cria uma lista ordenada da combinação dos dados recebidos por suas portas de entradas, dando prioridade aos valores vindo do ator

Energy.

O ator *Selector* têm três portas de entrada, a saída do ator *Combine*, a identificação da RPA e sua posição e uma porta de saída. Ele seleciona a primeira ação de movimento enviada pelo ator *Combine* e envia para a porta de saída. Além disso, é neste ator que se encontra o algorítimo de detecção de colisão. Com base na sua posição, o ator mede a distância com todas as outras RPAs e com as áreas de restrição de voo. Se a distância medida estiver entre a zona de impacto de uma RPA ou fizer a intersecção com alguma área restrita, o ator detecta uma colisão e seleciona a opção de pouso imediato. A saída desse ator é um comando de movimento enviado para o HLA, para ser entregue ao SITL, que executa a ação recebida.

Por fim, o ator *Log* possui quatro portas de entrada, identificação da RPA, consumo de bateria, posição geográfica e a saída do ator *Selector*. Esse ator não possui porta de saída. Ele é o responsável por registrar todas as informações de voo de uma RPA, durante a simulação.

Diferente da estratégia A, a estratégia B não possui o ator *GoPoints*. Porém, em seu lugar existe um ator chamado *RankPoints*. Esse ator possui duas portas de entrada, a identificação da RPA e sua posição geográfica e uma porta de saída. Ele é responsável por calcular as distâncias de cada ponto de visitação, com base na localização atual da RPA e criar uma lista ranqueada e ordenada, de forma decrescente, pelos pontos de visitação mais próximos. O cálculo funciona dessa forma: é atribuido um valor igual a todos os pontos que ainda não foram visitados. Então calcula-se a distância para cada ponto de visitação independente, e também um tempo futuro. Esse tempo futuro é uma estimativa de tempo, baseada na soma do tempo atual da simulação com os passos que a RPA leva para chegar ao ponto de visitação. Passo é a medida de distância que cada RPA percorre em um tempo de execução.

Em seguida, para cada ponto de visitação de uma RPA soma-se o valor do tempo futuro do valor atribuído aos pontos não visitados. O resultado desse cálculo é ordenado e enviado para a porta de saída. Por sua vez, o ator *RankActions* cria a lista de ações ranqueadas, com base nas informações recebidas pela porta de saída do ator *RankPoints*. A figura 4.12 mostra a visão interna detalhada da estratégia B. Nota-se que além dos atores comentados existe também algum atores de *Display*. Esses atores mostram em tempo real de simulação toda a informação das RPAs, referentes ao voo, como consumo de bateria, ação de movimento realizada, etc.

A criação de uma estratégia distinta é feita de maneira simples, graças ao reuso dos atores

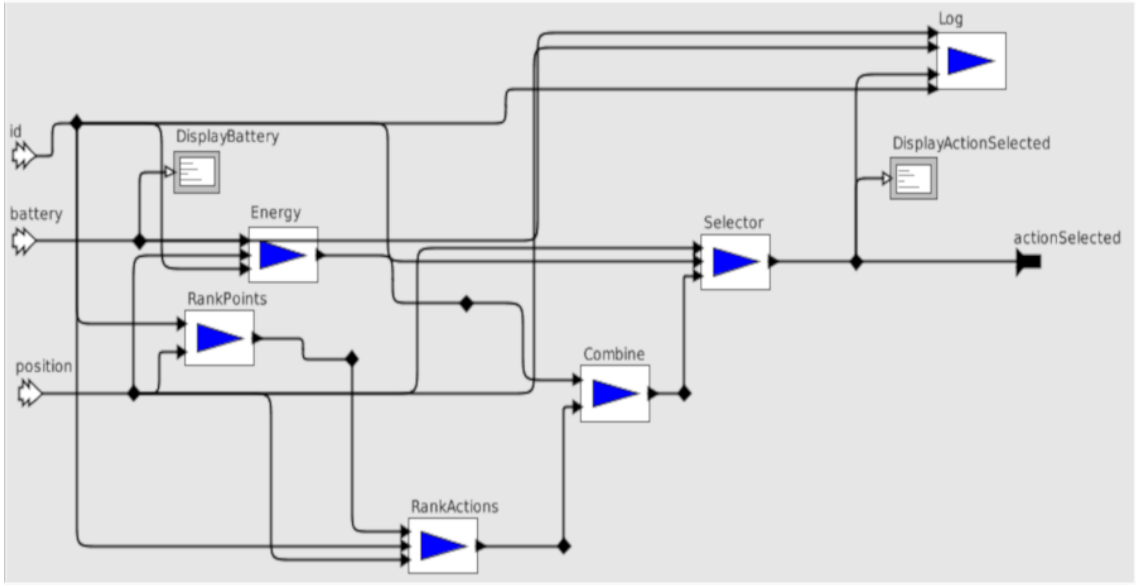

Figura 4.12: Visão interna detalhada da estratégia B

do Ptolemy II. Como visto, uma estratégia é desenvolvida em um ator composto, que contém diversos atores em sua estrutura interna, que realizam os tratamentos dos dados recebidos e transformam no próximo comando que será enviado para a RPA. Então, para o processo de criação de uma nova estratégia, deve-se arrastar um ator composto para a área gráfica do Ptolemy II. Essa ator se encontra na biblioteca *utilities*. Em seguida tem-se a opção de renomear o ator clicando com o botão direito, selecionando a opção *Customize* e em seguida *Rename*. Novamente com o botão direito, selecionar a opção *Open Actor* para poder editar o ator composto. A figura 4.13 mostra o ator composto aberto para edição.

Depois de aberto, o primeiro passo é colocar as portas de entrada e saída de dados. As portas de entrada servem para o recebimentos dos dados e a de saída é a porta por onde são enviados os dados logo após o processamento. Deve-se colocar três portas de entrada e uma de saída. Na figura 4.13 as portas *in*, *in2* e *in3* são as portas de entrada e a porta *out* a de saída. Aconselha-se renomear as portas de entrada de acordo com os dados que as portas receberão. Lembrando que as estratégias recebem três conjuntos de dados: a identificação da RPA, a localização da RPA no formato latitude, longitude e o nível de bateria da RPA. Todos os dados recebidos são do tipo *String*.

Em seguida, os atores responsáveis pelo processamento dos dados podem ser arrastados (*Drag-and-drop*) para a área de trabalho do Ptolemy II. Esses atores estão em *UserLibrary*,

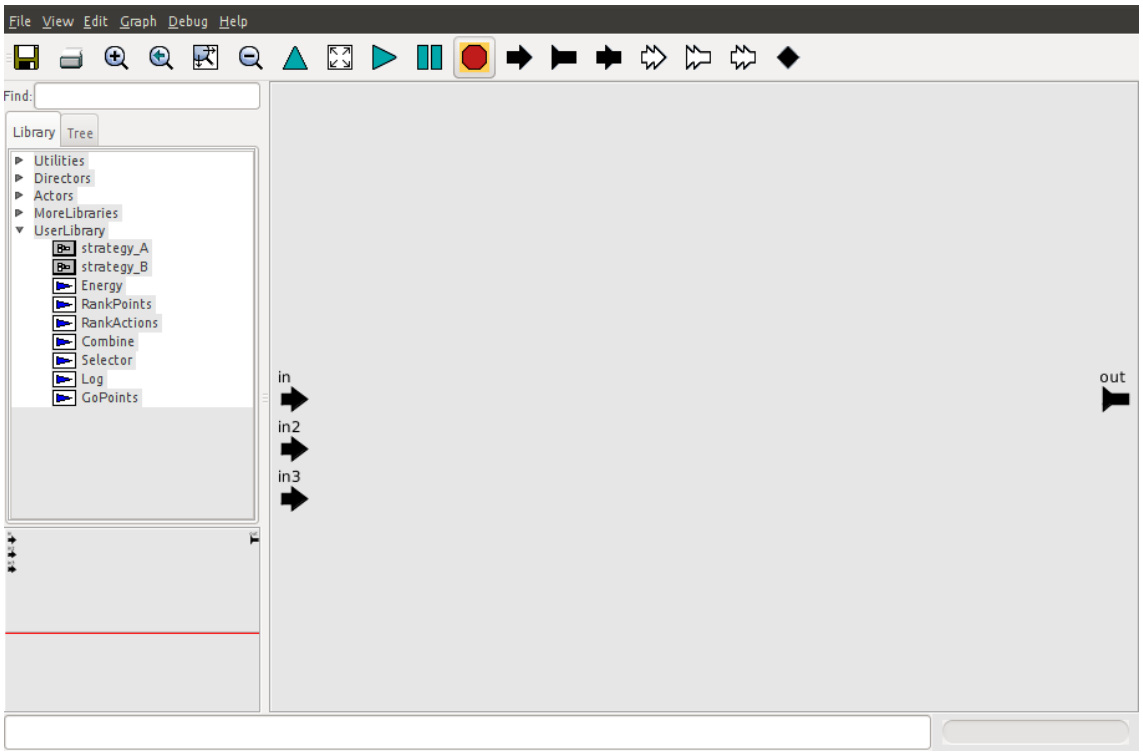

Figura 4.13: Exemplo de criação de uma estratégia

no lado esquerdo, visível na figura 4.13. Deve-se então fazer a interconexão dos atores através do relacionamento das portas de entradas com as de saída. Como um exemplo conecta-se a porta de entrada responsável pela recebimento do nível de bateria a porta de entrada da bateria do ator *Energy*, e continua ligando as portas de entradas restantes.

As estratégias utilizadas possuem todos os atores em comum, menos dois. Na estratégia A existe o ator *GoPoints* e na estratégia B o ator *RankPoints*. São esses atores que direcionam e definem o movimento da RPA. Em outras palavras, para se criar uma nova estratégia basta criar um ator com base em um desse atores (portas de entrada e saídas) e definir um algorítimo de direcionamento para a RPA. Desse modo, como exemplo, para criar uma estratégia onde a RPA deveria visitar os pontos mais distantes primeiro, bastaria modificar o ator *GoPoints* para calcular a distância dos pontos mais distantes, ordenar esse resultado e repassar para sua porta de saída.

4.2 Configuração do Ambiente

Antes de executar a simulação é necessário configurar o ambiente. Para isso se faz necessário editar o arquivo init.properties localizado na pasta raiz do projeto. Nele há diversos parâmetros para serem editados, a exemplo do local base de onde as RPAs começaram a simulação, cujo deve ser usados coordenadas de latitude e longitude. Por padrão cada RPA se posiciona a 5 metros de distância uns dos outros, tomando por base este parâmetro do local base. Esse valor também pode ser configurado. Isso se deve ao fato do algorítimo de detecção de colisão.

Existem também parâmetros referentes à altitude que as RPAs atingiram na decolagem e à quantidade de RPAs disponíveis para a simulação, este último, deve ter exatamente a mesma quantidade de instâncias contidas no modelo.

Existem algumas variáveis globais utilizadas pelas estratégias que podem ser configuradas: a variável *knowledgeBase*, que é a base conhecida pelas RPAs dos pontos de visitação utilizadas na simulação; a variável *reserveCharge* que indica a carga de bateria limite que quando atingida faz a RPA tentar voltar para a base. Para esse trabalho foi utilizado o valor 20, indicando que a RPA pode utilizar até 80% de carga de bateria antes de atingir o limite que indicará o retorno para a base; por fim a variável *minimumCharge* indica o valor de carga limite que a RPA dispõe para manter o bom funcionamento. Em outras palavras, independente de onde a RPA esteja, quando atingir esse limite é realizado um pouso forçado, para prevenção de acidentes e salvaguarda do equipamento. O valor atribuido a essa variável foi 5, ou seja, a RPA utiliza no máximo 95% da bateria.

O código fonte 4.2, mostra como é executado os arquivos *soft-constraints.py*, o federado responsável por repassar os dados do SITL para o HLA, e enviar os comandos de navegação para as RPAs.

Código Fonte 4.2: código que executa os federados

```
1    int copterAmount = Integer.parseInt(properties.getProperty("copter.amount"));
2    droneKnowledgeBase = DroneKnowledgeBase.getInstance();
3    droneKnowledgeBase.setCopterAmount(copterAmount);
4
5    ++countFederation;
6    String base = properties.getProperty("copter.home");
7
8    for (int i = 0; i < copterAmount; i++) {
```

```
9           softConstraints.add(Terminal.executarComando("python ./src/ptolemy/myactors/soft-
                constraints.py --count="
10              + (countFederation) + " --home=" + base + " --instance=" + (i) + " --altitude="
                     + properties.getProperty("copter.altitude")));
11              base = droneInitialDistance(Double.parseDouble(base.spli (",")[0]), Double.
                    parseDouble(base.split(",")[1]));
12      }
```

Explicando de forma simplista, o código fonte 4.2, na linha 1, é instanciada a variável *copterAmount*, e carregada com a configuração do arquivo init.properties. Essa variável é a responsável pela quantidade de RPAs usadas na simulação. Em seguida, nas linhas 2 e 3, essa quantidade é colocada na variável *droneKnowledgeBase*, que é a base conhecida pelo ambiente e pode ser usada pelas estratégias na simulação.

Na linha 5 é definido o identificador da federação e na linha 6 a variável *base* é carregada a partir do arquivo de configuração. Essa variável contém o endereço base de partida das RPAs, definido em coordenadas de latitude e longitude. Logo após, na linha 8, é iniciada a iteração responsável pela criação dos federados que se conectaram com o SITL e trocaram dados com o modelo de estratégias. Dessa forma o arquivo *soft-constraints.py* é executado um número de vezes igual a quantidade de RPAs definida na variável *copterAmount*.

Alguns parâmetros iniciais são passados nessa criação, como a base de lançamento da RPA, a altitude e a instância, que serve para identificar qual RPA é representado pelo federado. Em seguida na linha 11 é chamado o método *droneInitialDistance()* que define uma distância inicial de lançamento das RPAs. Como as RPAs levantam voo simultaneamente, esse método é essencial para que não haja colisões no início da simulação, pois a partir das coordenadas geográficas é determinado uma distância de separação entre as RPAs.

Ao executar o ambiente pela primeira vez, é necessário definir quais são os pontos de visitação que definiram a navegação das RPAs. O primeiro ponto marcado no mapa, sempre será o local onde as RPAs pousaram depois de completada a missão. Os outros pontos marcam os alvos que as RPAs visitaram. É necessário que a distribuição de pontos de visitação para cada RPA seja igualitária, ou seja, todas as RPAs devem ter a exatamente a mesma quantidade de pontos de visitação. Os pontos de visitação marcados são gravados em um arquivo chamado targets.txt, que serão lembrados nas próximas execuções subsequentes. Para a definição de novos pontos o arquivo target.txt deve ser apagado do disco rígido antes de se iniciar a simulação.

De modo semelhante, deve se desenhar no mapa, as áreas de restrições de voo, se necessário. Para isso deve-se escolher a opção *Restricted Areas* no componente localizado na parte superior do mapa. Depois de escolhido pode-se clicar no mapa, no lugar que se queria começar o desenho com o botão esquerdo do *mouse*. Em sequida, clicar em outros pontos a fim de desenhar qualquer polígono, sendo que cada clique corresponde a aresta do polígono. Por fim, quando o polígono já está formado, deve-se clicar com o botão direito do *mouse* para indicar que o desenho já está concluido. Pode-se colocar tantas áreas de restrição quanto se queira. Por padrão a escolha *Targets* é a que permite a escolha dos pontos de visitação.

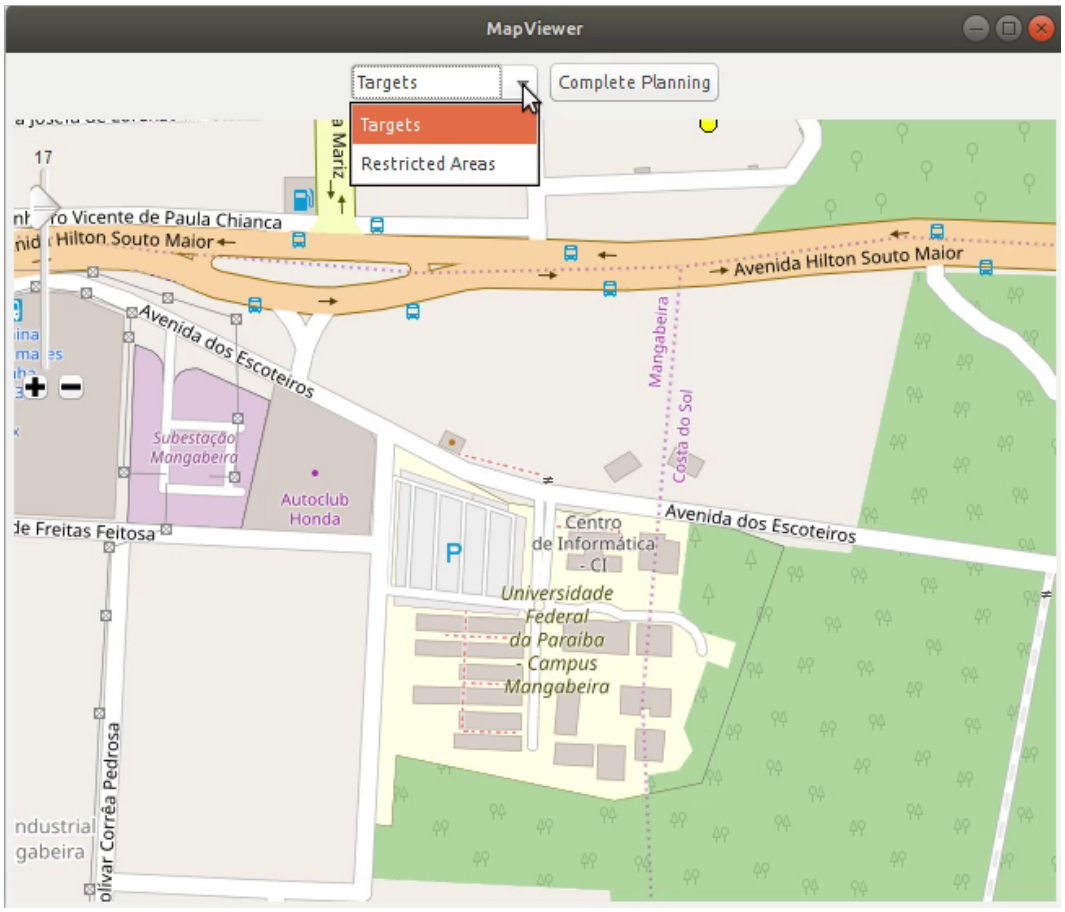

Figura 4.14: Configuração dos pontos de visitação e áreas restritas

A figura 4.14 mostra o componente localizado no topo do mapa que permite a escolha de inserção de pontos de visitação ou áreas de restrições de voo. Por fim, pode-se iniciar a simulação clicando no botão *Complete Planning* localizado ao lado do componente de escolha. Assim como os pontos de visitação, as áreas restritas são salvas em um arquivo chamado restrictedAreas.txt, em forma de pontos de coordenadas. Para apagar as áreas restritas já

definidas, deve-se excluir esse arquivo.

Ao final de cada simulação, são salvos as informações referentes às RPAs e suas missões, como distância percorrida, consumo de bateria, quantidade de fotos capturadas, juntamente com a localização de onde foi tirado a foto.

Capítulo 5

Experimentos

5.1 Estudo de Caso

Para validar o ambiente de simulação proposto, foi criado um estudo de caso. Consideramos um cenário de vigilância aérea, onde a missão de cada *Drone* seria verificar todos os pontos de visitação alvos espalhados de forma aleatória. Estes pontos são divididos de forma igualitária para cada RPA. Foi definido um total de 24 (vinte e quatro) pontos de visitações e usados até 4 (quatro) RPAs. Foram desenvolvidos dois cenários distintos: o primeiro com uma área livre de obstáculos e o segundo com áreas contendo zonas de restrições de voos, como mostra a figura 5.1. Estas áreas são definidas como áreas de impacto, onde a colisão deve ser detectada se qualquer *Drone* ultrapassar essa zona. Também foi adicionado possibilidade de ocorência de vento, de modo a aumentar o nível de realismo das simulações. Para as porcentagens de vento, foram usados os valores de 0%, que indica que não haveria vento na simulação, e ainda 2,5%, 5,0%, 7,5%, 10.0%, 12,5% e 15,0% de possibilidades de ocorrências de vento. Ainda, foram utilizadas três intensidades do vento, que indicam quão forte os ventos interagiriam com as RPAs: intensidade fraca, mediana e forte.

A figura 5.1 mostra como os pontos de visitação foram distribuídos ao longo do mapa. Os círculos em amarelo são os pontos de visitação. Os polígonos em azul são áreas restritas.

Adicionamente, foram utilizadas duas estratégias que nortearam a maneira como as RPAs agiriam para completar a missão. Na primeira estratégia, denominada estratégia A, as RPAs seguiriam os pontos de visitação sequencialmente de forma determinada, de acordo como foram distribuídos no mapa. Na segunda estratégia, chamada estratégia B, as RPAs

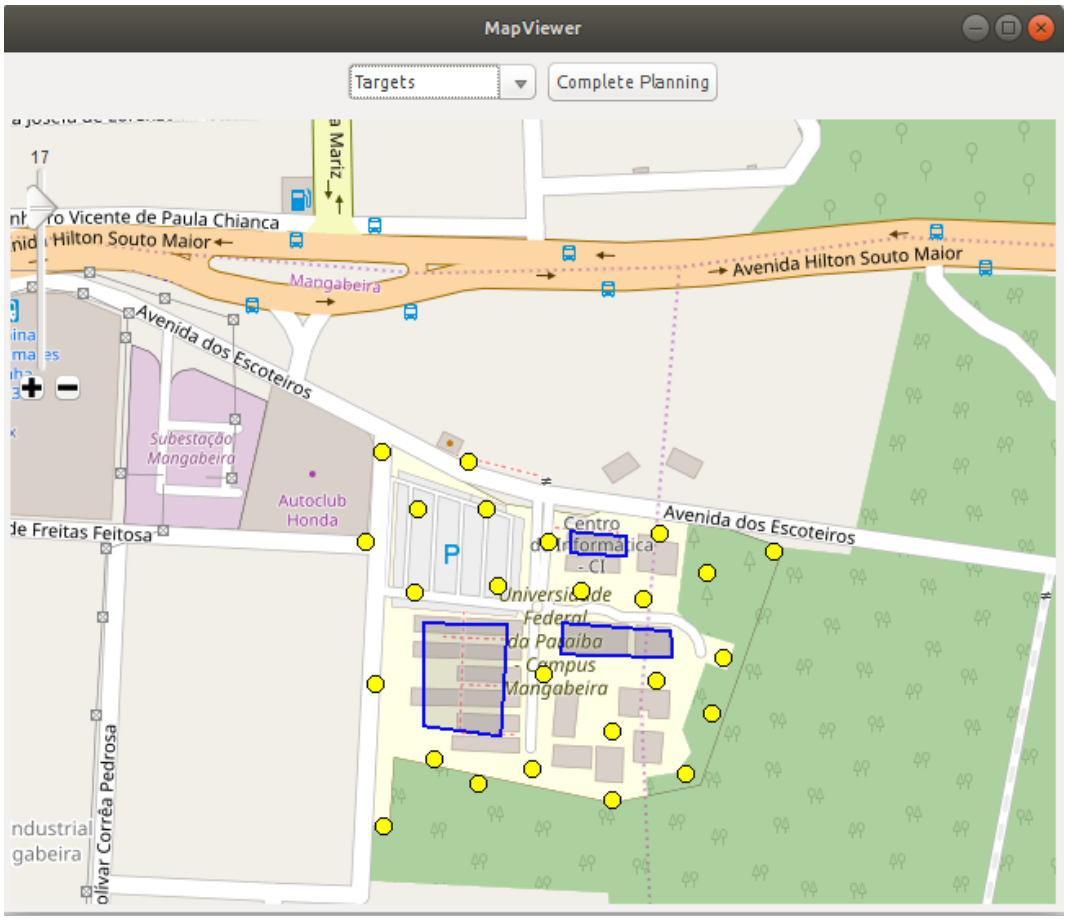

Figura 5.1: Exemplo de cenário utilizado com áreas restritas

devem visitar primeiro os pontos mais próximos de sua localização.

A missão das RPAs seria visitar todos os pontos alvos designados no mapa e capturar uma imagem do local de cada ponto, então retornar para a base. Se uma RPA alcançasse 80% de consumo de bateria, deveria abandonar a missão e retornar a base imediatamente. Além disso, caso uma RPA com dificuldade de retornar para a base chegasse a 95% de consumo de bateria, deveria pousar imediatamente, para a proteção do equipamento e seus componentes.

A figura 5.2 demonstra um exemplo de uma simulação sendo executada com múltiplas RPAs. Para essa simulação foram utilizadas 4 RPAS. As RPAs, neste exemplo, podem ser identificadas pelas cores azul, vermelho, verde e rosa. A medida que elas vão se movimentando deixam um rastro que indica o caminho percorrido por elas.

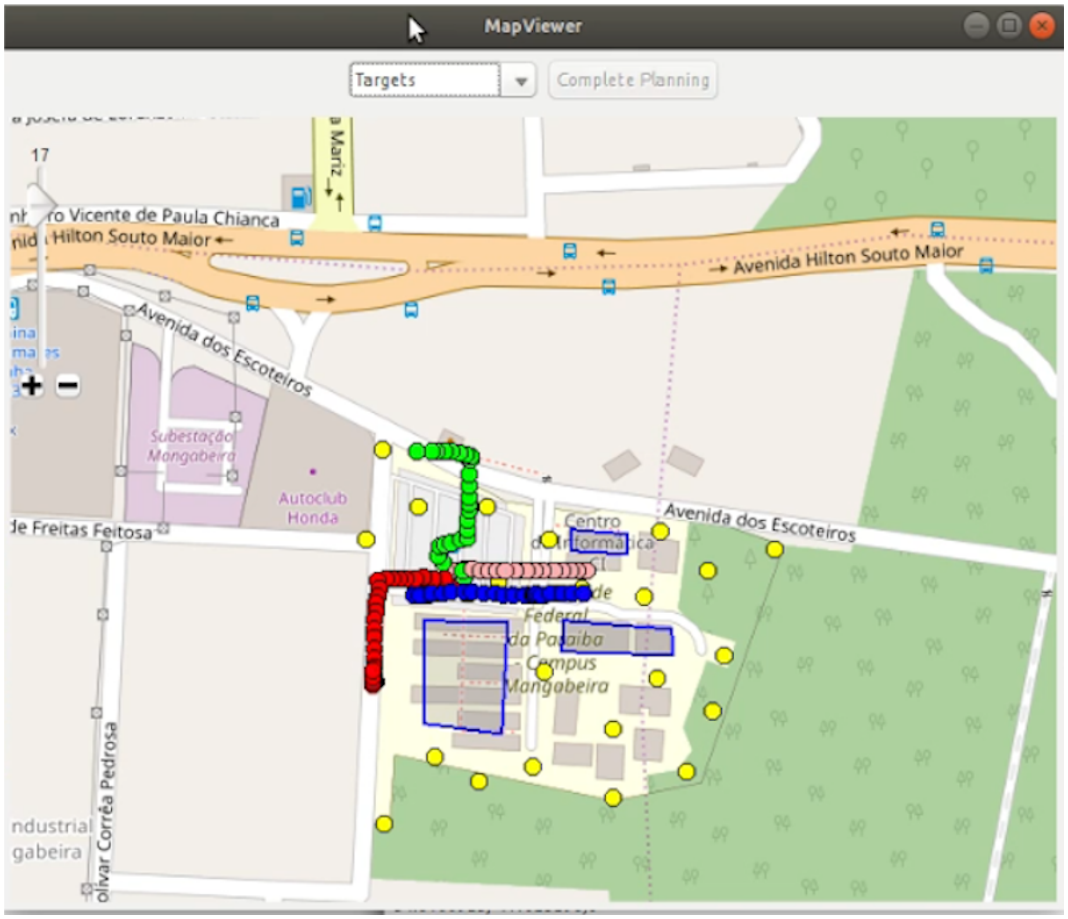

Figura 5.2: Demonstração de simulação sendo executada

Os dados de todas as simulações foram gravadas em arquivos e posteriormente transformados em uma tabela com informações de voo de todos os *Drones*. Para cada cenário, todas a estratégias foram simuladas de forma individual e um número de RPAs foram usadas, de

um até quatro, reespectivamente. Adicionamente, para cada possibilidade de ocorrência de vento, três simulações foram executadas, resultado em um total de 336 simulações. Cada simulação levava uma média de 4,20 minutos (quatro minutos e 20 segundos). Dessa forma, foram um total de 23,52 horas (vinte e três horas e 52 minutos) de tempo de simulação. Na verdade esse tempo ainda é maior, já que ao final de cada simulação era necessário mover os arquivos com as informações registradas da simulação, para uma nova pasta, finalizar os processos abertos pelo ambiente e iniciar outra simulação, tudo isso de forma manual.

Por fim foi criado um algorítimo de extração de dados usando a biblioteca pandas, que facilita a visualização dos dados coletados das simulações. É necessário colocar o caminho do diretório dos arquivos salvos pela simulação e executar o algorítimo. Ele salva todas as informações em uma tabela única, em um arquivo do tivo CSV. Além disso, salva tabelas individuais, em arquivos CSV separados, de acordo com as métricas escolhidas, como por exemplo uma tabela para a quantidade de colisões por área. A partir disso, fica fácil a visualização dos dados e a criação de gráficos.

5.2 Resultados

Para um melhor entendimento dos resultados obitidos, escolhemos como métricas de desempenho o quantitativo de imagens capturadas, a média das cargas das baterias restantes das RPAs, a distância total de percurso e a quantidade de colisões ocorridas durante as simulações. Ainda sobre as informações colhidas, definimos os conceitos de sucesso e falha nas simulações. Ocorre um sucesso quando uma RPA percorre todos os seus pontos de visitação atribuídos e retorne a base, desde que sua carga mínima de bateria fosse maior que 5%, pois, um nível de bateria no estado crítico seria um indício de pouso forçado, e uma falha ocorre em todas as outras situações. A Figura 5.3 mostra a quantidade de sucessos obtidos pela quantidade de RPAs utilizadas.

As barras em azul indicam a quantidade de sucessos obtidos pelos UAVs utilizando a Estratégia A. Já as em vermelho indicam a quantidade de sucesso na Estratégia B. A quantidade máxima de sucessos que poderiam ser obitidos para as simulações por estratégia com 1, 2, 3 e 4 RPAs, reespectivamente, eram 42, 84, 126, 168. Mesmo nenhuma das estratégias conseguindo obter 100% de aproveitamento, pode-se notar que a estratégia B, de fato, foi

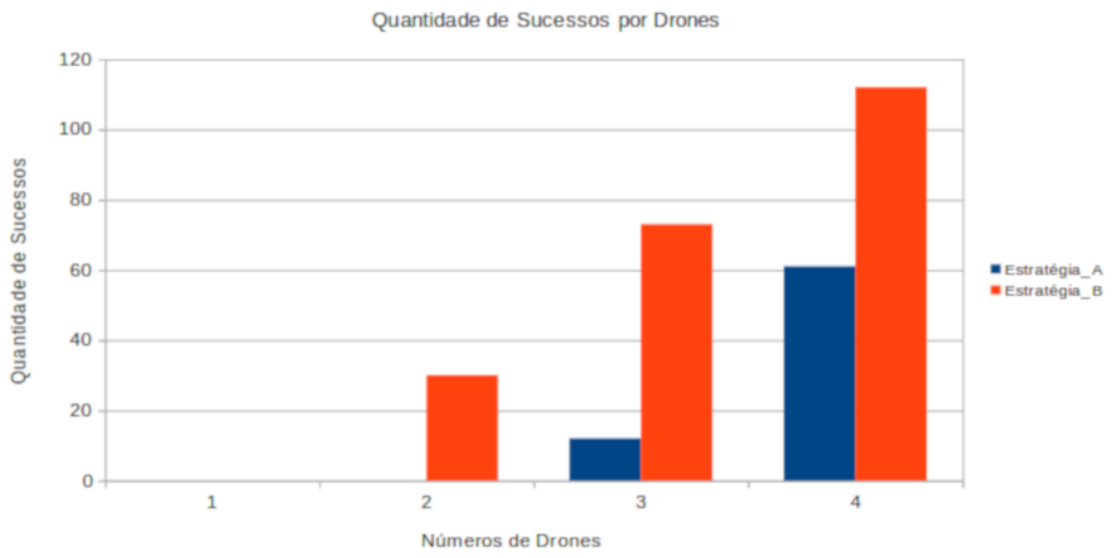

Figura 5.3: Quantidade de sucesso por RPA

superior a estratégia A, por obter um maior número de êxitos em suas missões.

Considerando apenas a quantidade total de fotos capturadas pelas RPAs em cada estratégia e por cenário, conclui-se que os que voaram pelas áreas livres de obstáculos conseguiram capturar mais imagens de seus alvos no geral. Entretanto, novamente, a estratégia B foi superior, inclusive se compararmos o resultado do cenário da área restrita contra o resultado do cenário de área livre da estratégia. A Figura 5.4 mostra essa diferença. No gráfico, observa-se a quantidade total de fotos obtidas, separadas pelos cenários com área livre e com área restrita. Em azul, está a estratégia A e, em vermelho, a estratégia B.

O ambiente permite ainda a análise do quantitativos de fotos capturadas por possibilidade de ocorrência de vento, observadas na figura 5.5. No gráfico, em azul escuro, tem-se as simulações que ocorreram sem vento. As demais cores indicam possibilidade de ocorrência de vento variando de 2,5% a 15%. Esperava-se um desequilíbrio maior na atuação das simulações com altas probabilidades de vento, tendo em vista que o vento atrapalharia os drones, empurrando-os para fora de suas trajetórias. De fato, observando o cenário de área livre da estratégia A nota-se que, quanto maior a probabilidade de vento, menos fotos foram capturadas. Em contrapartida, nos outros cenários, houve situações em que uma maior possibilidade de vento ajudou a aumentar o quantitativo de fotos obtidas. Isso pode ser explicado, em se tratando da estratégia B, pelo seu dinamismo, pois o vento acabava por empurrar os drones

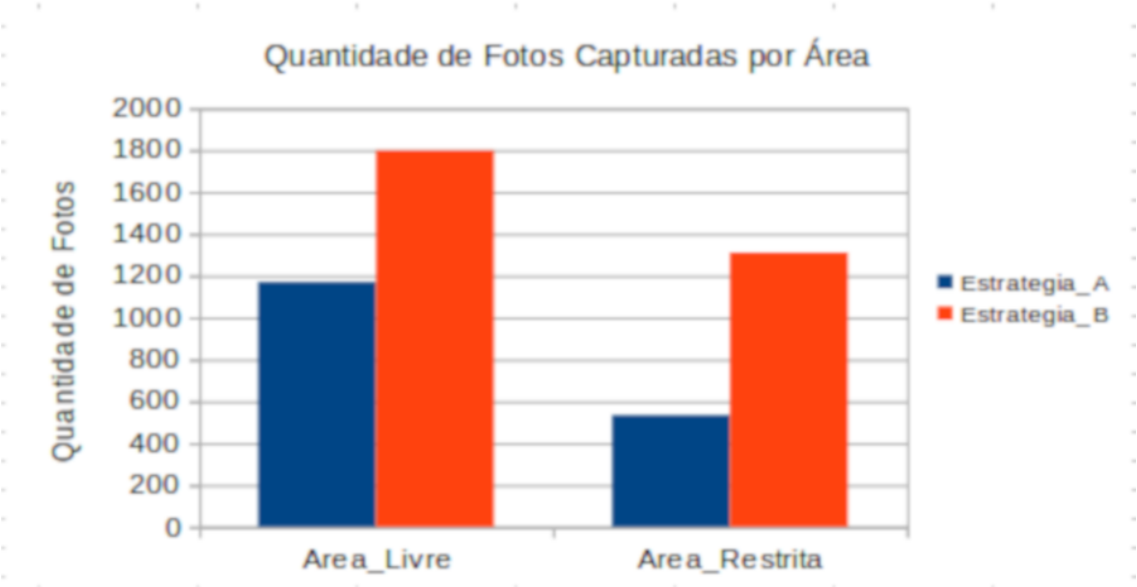

Figura 5.4: Quantidade de fotos capturadas por cenário

para pontos próximos, o que fazia com que eles mudassem suas trajetórias. Já pela estratégia A, o auto índice de colisões, ajudou a dar equilibrio ao gráfico.

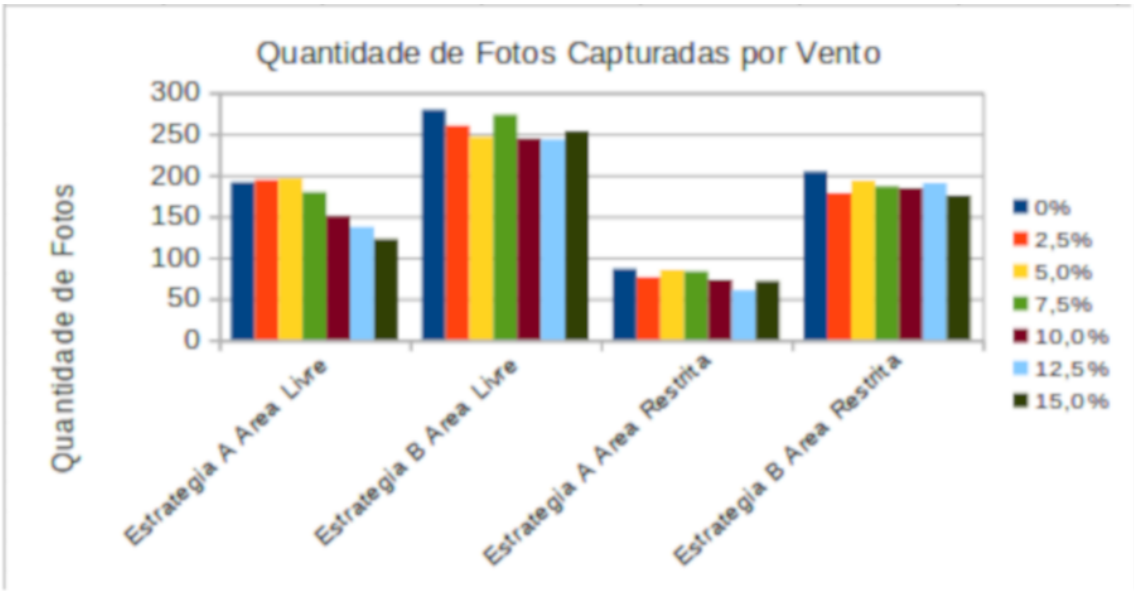

Figura 5.5: Quantidade de imagens capturadas por ocorrência de vento

Outro índice interessante de se analisar é o consumo de bateria. As missões em que a RPA fique muito tempo em vôo, ou que contenham pontos de visitação distantes uns dos outros, acarretam em maior consumo de bateria. Observado na figura 5.3, os UAVs que utilizaram a estratégia B tiveram mais sucessos na conclusão de suas missões que os da estratégia A.

Isto já dava indícios de que os drones da estratégia B teriam, no geral, mais cargas de bateria restantes. De fato, a média total de bateria restante dos drones que utilizaram a estratégia B foi de 41.83% enquanto a dos que utilizaram a estratégia A foi de 41.07%. Em contrapartida, ao se analisar a quantidade média restante de bateria, pelos cenários de áreas livre e restrita, mostradas na figura 5.6, pode-se observar que no cenário com área restrita a estratégia A, caracterizada pela cor azul, foi superior a estratégia B. O aumento da quantidade de bateria restante, quando em área restrita, se deve ao fato da quantidade de colisões no começo da simulação, fato que influenciou os resultados.

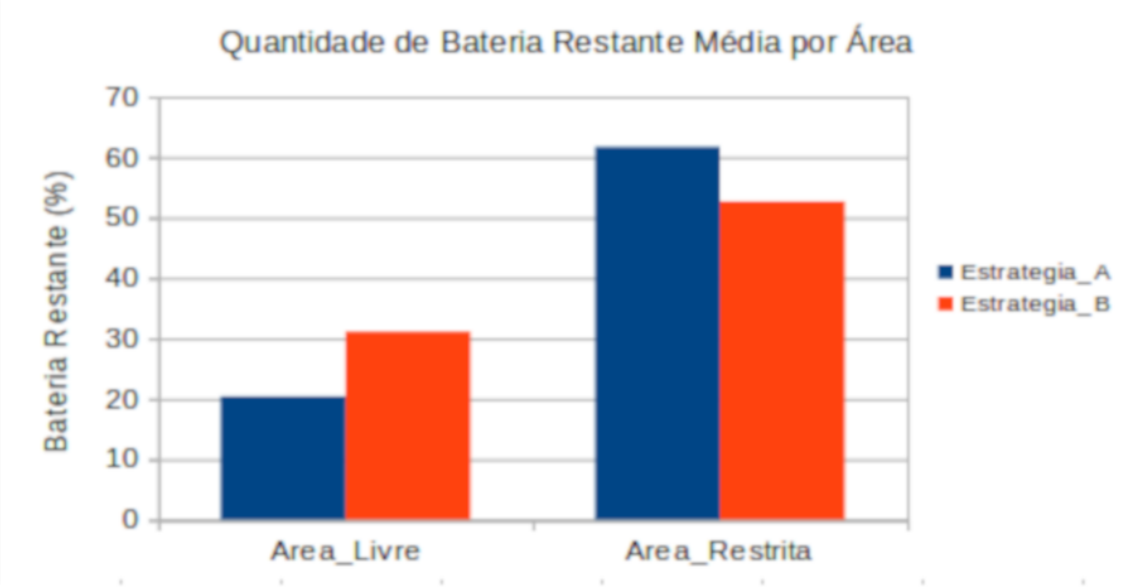

Figura 5.6: Quantidade média de bateria restante por cenário

Já em se tratando de carga de bateria restante em relação ao percentual de ocorrência de vento, notadamente pode-se observar na figura 5.7, que o consumo aumenta junto com a maior probabilidade de vento. Isso se dá pelo fato das RPAs corrigirem suas trajetórias na medida que os ventos os empurrem. O gráfico mostra a quantidade média de carga de bateria restante em relação ao percentual de probabilidade de ocorrência de vento, observados as estratégias e as áreas. As cores mostram o percentual de ocorrência de vento, sendo que o azul mais escuro significa sem vento, proseguindo até o verde escuro que siginifica 15% de probabilidade de que ocorra vento. O baixo consumo de bateria nas zonas de área restrita é justificado pelo alto índice de colisões nessa áreas, influenciado, de certo modo, pelo vento.

Outra métrica de desempenho utilizada foi a distância total percorrida, onde, para critério

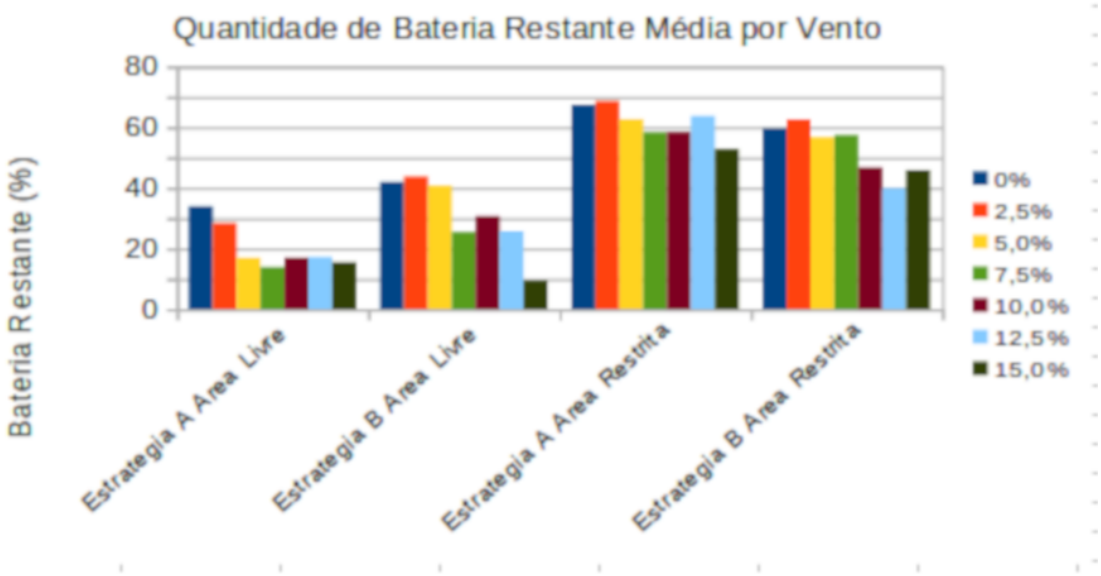

Figura 5.7: Quantidade média de bateria restante por ocorrência de vento

de avaliação, a estratégia mais eficiente seria a que possuísse menor distância total, entre suas missões. Novamente, a estratégia B foi a vencedora nesse quesito com uma distância total percorrida de um pouco mais que 400km, em comparação com a estratégia A que teve um pouco mais de 417km, pelo fato do dinamismo da escolha de seus pontos de visitação, já que, na estratégia A, mesmo tendo pontos de visitação mais perto, ela se guiava pela escolha predeterminada, independente da distância, enquanto que na estratégia B, a escolha do próximo ponto de visitação se dava pelo fator distância.

Quando analisado pelo zoneamento dos cenários, observa-se que, nas áreas restritas, as distâncias percorridas são menores. Isso pode ser explicado pela quantidade de colisões, que acabaram por diminuir a distância total, já que as RPAs não terminavam suas missões. Ainda, de acordo com a figura 5.8, o desempenho da estratégia A, em azul, foi superior a da estratégia B, em vermelho. Isso dá indícios que houve mais colisões no início das simulações na estratégia A que na estratégia B.

Ainda, em se tratando de distância total percorrida, pode-se verificar a influência do vento, nas simualções. A figura 5.9 mostra a distância total percorrida em relação ao percentual de probabilidade de ocorrência de vento, das duas estratégia, nos cenários de área livre e área restrita. Em azul mais escuro o percentual de ocorrência de vento é 0%, enquanto que em verde escuro se encontra o maior percentual que é de 15%. Pode-se analisar que,

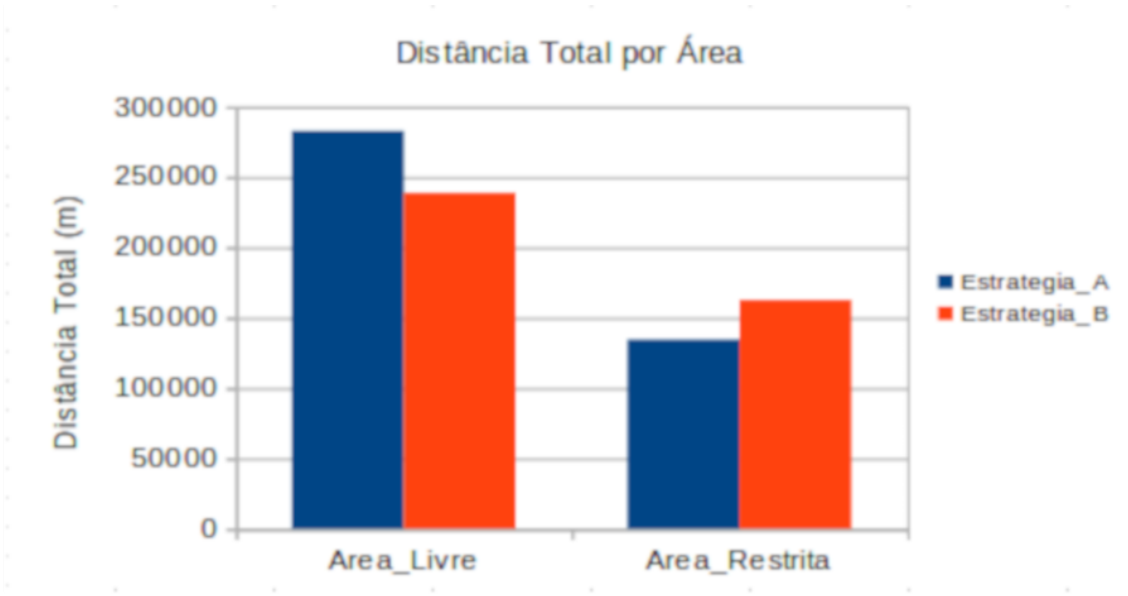

Figura 5.8: Distância total percorrida por cenário

salvo algumas exceções, quanto maior a ocorrência de vento, maior sua influência quanto a distância total de percurso.

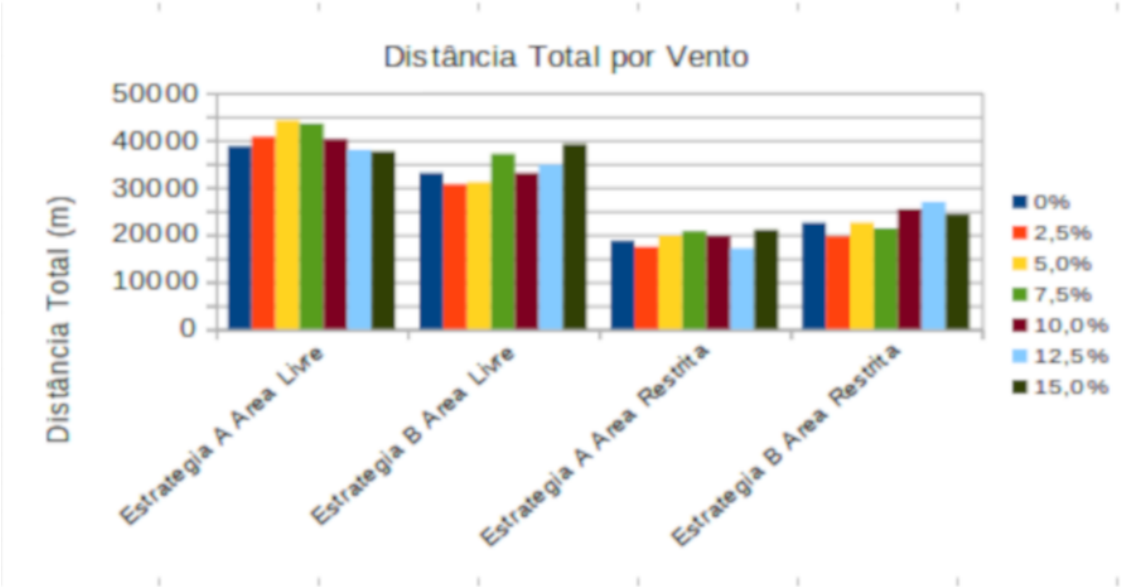

Figura 5.9: Distância total percorrida por ocorrência de vento

Uma das métricas mais importantes, que acabou por ter uma influência muito grande no desempenho dos resultados das simulações, foi a quantidade de colisões totais. Neste caso, não houve nenhum sistema de prevenção de colisões. As RPAs simplesmente seguim suas rotas e, caso deparassem com outra RPA, não usaria nenhuma técnica para evitar a colisão.

Já em relação as zonas com áreas restritas, oas RPAs tentariam contornar as áreas que estivessem em sua rota. Mas essa mudança de rota só se daria quando a RPA detectasse a área restrita a aproximadamente 3 metros de distância. Para critérios de avaliação, definimos o quantitativo total de colisões, independente do tipo de ponto de impacto, ou seja, indiferente de ser colisões do tipo RPA por RPA ou de RPA por área restrita, de forma que a estratégia com menor quantidade de colisões seria a mais eficiente. Para essa métrica a estratégia A teve um total de 215 colisões totais, enquanto que na estratégia B houve 181, como mostra a figura 5.10. No gráfico em azul, está a quantidade de colisões totais obtidas utilizando a estratégia A e em vermelho, aquelas utilizando a estratégia B.

Figura 5.10: Quantidade de colisões por estratégia

Por outro lado, pode-se analisar ainda as colisões ocorridas, separadas por pontos de impacto. Nesse caso, definimos ponto de impacto como as colisões ocorridas entre RPAs ou entre RPA e área restrita, que foi tratada como colisão em uma parede. A figura 5.11 mostra o gráfico de quantidade de colisões em relação ao ponto de impacto. Em azul está a quantidade de colisões ocorridas na parede, e em vermelho as ocorridas entre *Drones*. Observa-se que nas duas estratégias, a quantidade de colisões ocorridas na parede chega a ser o triplo em relação as obtidas entre RPAs. Esse fato pode ser caracterizado por diversos fatores, desde a incidência do vento agindo contra o RPA à detecção tardia da área de impacto, fazendo com que o *Drone* não conseguisse manobrar.

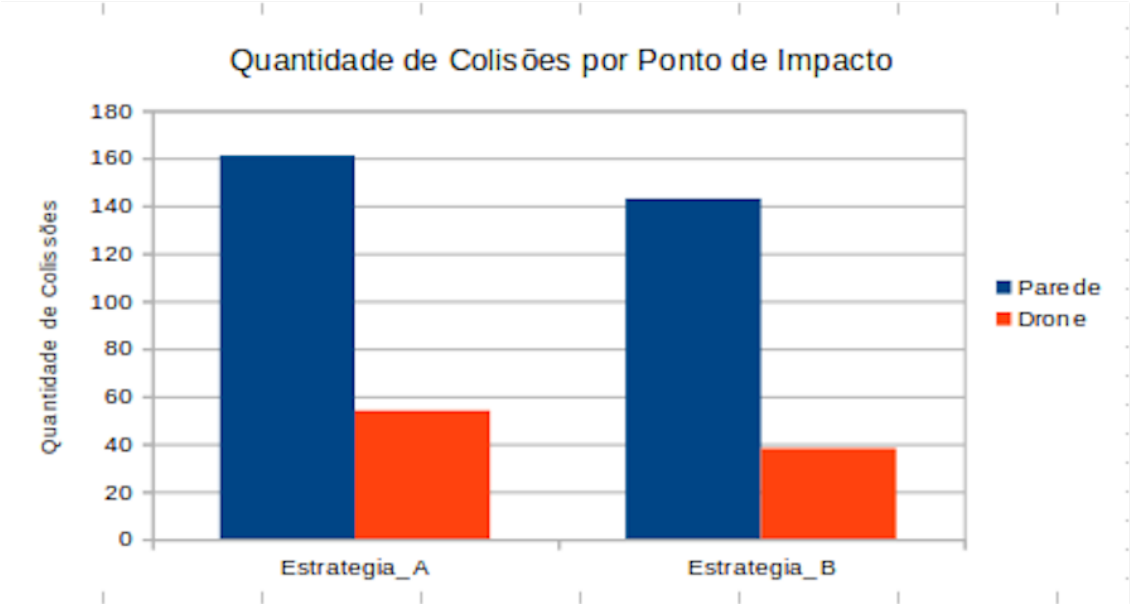

Figura 5.11: Quantidade de colisões por ponto de impacto

Ainda, para uma análise mais criteriosa, pode-se aferir o desempenho dos resultados das simulações pela quantidade de RPAs utilizadas. Dessa forma é possível ter uma visão mais profunda entre as diferentes quantidades de RPAs utilizadas. A figura 5.12 mostra a quantidade média de fotos capturadas dos pontos alvos pela quantidade de RPAs utilizadas. A barra em azul indica o uso da estratégia A, enquanto que a em vermelho indica o da estratégia B. O gráfico mostra, que na medida que acrescentamos RPAs à simulação, o desempenho aumenta de forma gradual nas duas estratégias.

Já em comparação ao gasto de carga de bateria, pode-se averiguar na figura 5.13, que mostra a média restante de bateria por quantidade de RPAs, que o consumo geral das RPAs tende a diminuir gradativamente ao se introduzir uma maior quantidade de RPAs nas simulações. Mais, nesse caso, houve uma exceção notadamente, identificada no uso da estratégia A, em azul no gráfico, quando usado apenas uma RPA. A explicação se da pela questão espacial, ou seja, o último ponto de visitação que o drone sobrevoava antes da bateria restante chegar em 20% era próximo a estação base de retorno e pouso.

Por fim, a utilização da estratégia B foi, a que atingiu melhor performance com esse conjunto de dados obitidos pelo ambiente de simulação proposto vide tabela 5.1, que mostra o desempenho das estratégias com relação as métricas utilizadas. O desempenho de cada estratégia foi medido utilizando a soma total de cada métrica, ou seja, tomando como base a

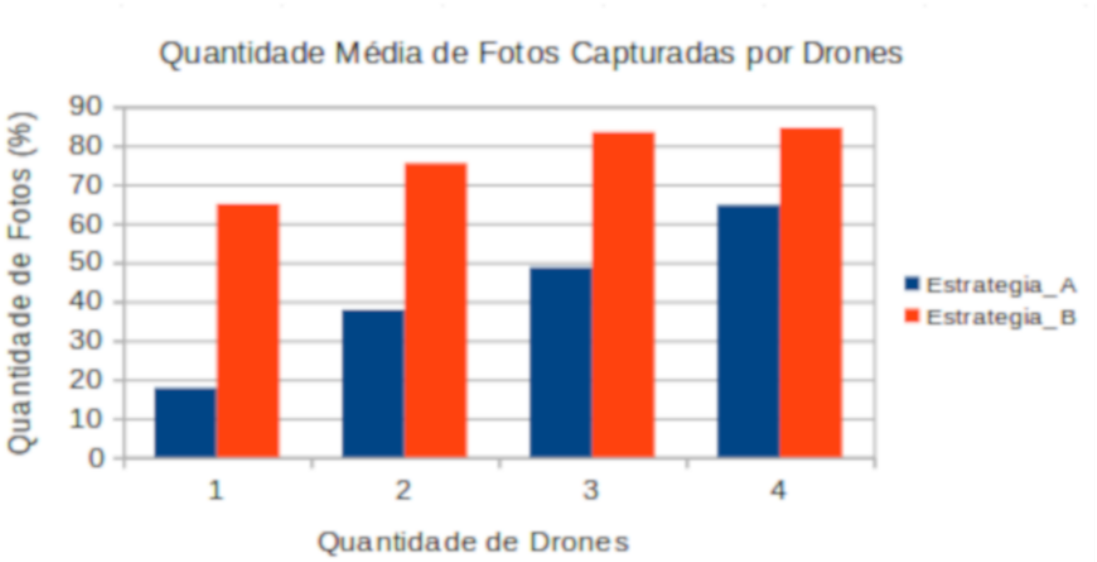

Figura 5.12: Quantidade média de imagens capturadas por quantidade de RPAs

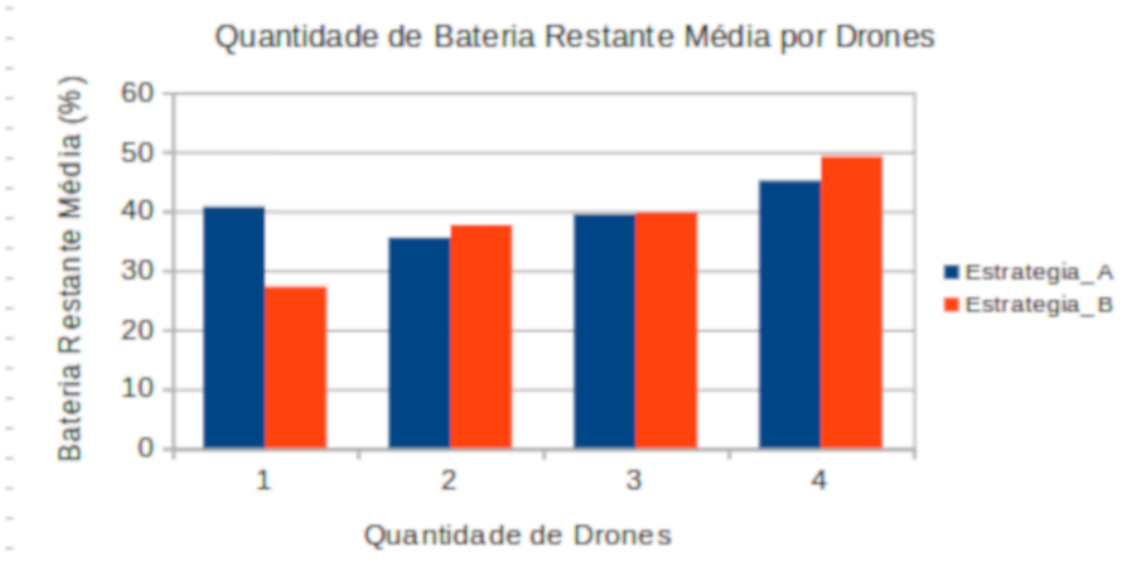

Figura 5.13: Quantidade média de bateria restante por quantidade de RPAs

bateria restante por área, foi somado o desempenho das estratégias em todos os cenários, e o resultado foi utilizado como o parâmetro para medir o desempenho final.

Pode-se concluir que o ambiente proposto cumpre o seu papel de simular e analisar várias estratégias diferentes, utilizando missões com múltiplas RPAs entre diversos cenários e gerando informações bastante conclusivas, que podem ser úteis nas tomadas de decisões ou em testes reais.

Tabela 5.1: Tabela de desempenho das Estratégias

Métricas (Totais)	**Estratégia A**	**Estratégia B**
Sucessos por quantidade de drones		✔
Quantidade de fotos por área		✔
Quantidade de fotos com vento		✔
Quantidade de bateria restante por área		✔
Quantidade de bateria restante com vento		✔
Distância percorrida		✔
Distância percorrida com vento		✔
Quantidade de Colisões		✔

Os dados obtidos através das simulações são ainda mais significativos. Pode-se estimar, por exemplo, a autonomia média das RPAs, baseado no consumo de bateria e nas distâncias percorridas, medir o desempenho individual de cada RPA, separadamente, etc. Em todo caso, a ferramenta proposta se mostrou bastante satisfatória, provendo informações suficientes para a análise das estratégias utilizadas.

Capítulo 6

Considerações Finais

O presente trabalho propos o desenvolvimento extensivo de uma ferramenta existente, com o objetivo de criar um ambiente de simulação para a avaliação de estratégias de múltiplos *Drones* utilizando a cossimulação, em um ambiente com incertezas, aproximando-se ao máximo da realidade. A ferramenta é composta por dois simuladores distintos, o Ptolemy II e o SITL/Ardupilot, que se trocam informações utilizando o HLA.

De modo a validar a ferramenta proposta, um experimento foi conduzido. Foi considerado um cenário de vigilância aérea, onde as missões das RPAs seriam a de visitar e fotografar os pontos de visitação cadastrados no mapa. Para isso, foram empregados duas estratégias de voos diferentes. Para tornar o ambiente o mais próximo da realidade possível, algumas dificuldades foram impostas, tal como áreas com restrições de voo, colisões e a influência dos ventos nas RPAs.

Como a proposta foi a extensão de uma ferramenta existente, houve bastante dificuldade na codificação e integração para tornar o ambiente adaptado a utilização de múltiplos *Drones*, de forma a precisar de centenas de testes e execuções até deixar o ambiente funcional. Alguns *bugs* foram retirados e a estética do ambiente também foi melhorada, mas a ferramenta ainda é um pouco pesada quanto a sua execução, embora este fato não tenha dificultado o andamento do trabalho.

Os resultados obtidos deste trabalho permite a avaliação e a discussão de melhores métodos de planejamento de missões e estratégias de voo a serem adotadas. Como demonstrado, o ambiente proposto foi capaz de simular múltiplos *Drones* entre diferentes cenários, coletando dados para análise posterior e ajudando a avaliação das melhores estratégias a serem

empregadas.

Como trabalho futuro pretende-se criar um canal de comunicação entre as RPAs, para que possam trocar mensagens entre si a curtas distâncias, informando por exemplo suas posições atuais ou pontos de visitação ainda não fotografados. Dessa forma, pode-se criar estratégias bem mais dinâmicas para os *Drones*. Além disso, pretende-se criar um mecanismo de controle para mitigar as colisões, utilizando o canal de comunicação para atingir esse objetivo.

Referências Bibliográficas

[Alsalam et al. 2017]ALSALAM, B. H. Y. et al. Autonomous uav with vision based on-board decision making for remote sensing and precision agriculture. In: IEEE. *Aerospace Conference, 2017 IEEE*. [S.l.], 2017. p. 1–12.

[Barros et al. 2016]BARROS, J. d. S. et al. A framework for the analysis of uav strategies using co-simulation. In: *VI Brazilian Symposium on Computing Systems Engineering (SBESC)*. [S.l.: s.n.], 2016. p. 9–15.

[Barros et al. 2017]BARROS, J. d. S. et al. Uma ferramenta para avaliar estratégias de voos de vants usando cossimulação. Universidade Federal da Paraíba, 2017.

[Barros et al. 2017]BARROS, J. de S. et al. Analysis of design strategies for unmanned aerial vehicles using co-simulation. *Design Automation for Embedded Systems*, Springer, v. 21, n. 3-4, p. 157–172, 2017.

[Brito et al. 2013]BRITO, A. V. et al. Development and evaluation of distributed simulation of embedded systems using ptolemy and hla. In: IEEE COMPUTER SOCIETY. *Proceedings of the 2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications*. [S.l.], 2013. p. 189–196.

[CERTI DOC Introduction 2018]CERTI DOC Introduction. 2018. Http://www.nongnu.org/certi/certi_doc/User/html/index.html. 2018-08-22.

[Choosing a Ground Station 2018]CHOOSING a Ground Station. 2018. Http://ardupilot.org/copter/docs/common-choosing-a-ground-station.html. 2018-08-22.

[Civil 2017]CIVIL, A. N. de A. *REQUISITOS GERAIS PARA AERONAVES NÃO TRIPULADAS DE USO CIVIL*. [S.l.], Maio 2017. 26 p. Disponível em:

<http://www.anac.gov.br/assuntos/legislacao/legislacao-1/rbha-e-rbac/rbac/rbac-e-94-emd-00/@@display-file/arquivo_norma/RBACE94EMD00.pdf>.

[Committee et al. 2000]COMMITTEE, S. I. S. et al. Ieee standard for modeling and simulation (m & s) high level architecture (hla)-framework and rules. *pp. i–22*, 2000.

[Corner e Lamont 2004]CORNER, J. J.; LAMONT, G. B. Parallel simulation of uav swarm scenarios. In: *Proceedings of the 36th Conference on Winter Simulation*. [S.l.]: Winter Simulation Conference, 2004. (WSC '04), p. 355–363. ISBN 0-7803-8786-4.

[Dahmann, Fujimoto e Weatherly 1997]DAHMANN, J. S.; FUJIMOTO, R. M.; WEATHERLY, R. M. The department of defense high level architecture. In: IEEE COMPUTER SOCIETY. *Proceedings of the 29th conference on Winter simulation*. [S.l.], 1997. p. 142–149.

[Erdos, Erdos e Watkins 2013]ERDOS, D.; ERDOS, A.; WATKINS, S. E. An experimental uav system for search and rescue challenge. *IEEE Aerospace and Electronic Systems Magazine*, IEEE, v. 28, n. 5, p. 32–37, 2013.

[Fujimoto 1998]FUJIMOTO, R. M. Time management in the high level architecture. *Simulation*, Sage Publications Sage CA: Thousand Oaks, CA, v. 71, n. 6, p. 388–400, 1998.

[Ghazal, Khalil e Hajjdiab 2015]GHAZAL, M.; KHALIL, Y. A.; HAJJDIAB, H. Uav-based remote sensing for vegetation cover estimation using ndvi imagery and level sets method. In: IEEE. *2015 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. [S.l.], 2015. p. 332–337.

[Gomes et al. 2017]GOMES, C. et al. Co-simulation: State of the art. *arXiv preprint arXiv:1702.00686*, 2017.

[Grippa 2016]GRIPPA, P. Decision making in a uav-based delivery system with impatient customers. In: IEEE. *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. [S.l.], 2016. p. 5034–5039.

[Heintz, Rudol e Doherty 2007]HEINTZ, F.; RUDOL, P.; DOHERTY, P. From images to traffic behavior-a uav tracking and monitoring application. In: IEEE. *2007 10th International Conference on Information Fusion*. [S.l.], 2007. p. 1–8.

[Holness et al. 2016]HOLNESS, C. et al. Remote sensing archeological sites through unmanned aerial vehicle (uav) imaging. In: *IGARSS*. [S.l.: s.n.], 2016. p. 6695–6698.

[IEEE 2010]IEEE. Ieee standard for modeling and simulation (m&s) high level architecture (hla)—object model template (omt) specification. *IEEE No. 1516.2-2010*, 2010.

[IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Federate Interface IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Federate Interface Specification. *IEEE Std 1516.1-2010 (Revision of IEEE Std 1516.1-2000)*, p. 1–378, Aug 2010.

[Introducing DroneKit-Python 2019]INTRODUCING DroneKit-Python. 2019. Http://python.dronekit.io/about/index.html. 2019-04-04.

[JOSM 2019]JOSM. 2019. Https://josm.openstreetmap.de. 2019-04-08.

[Kadrovach 2003]KADROVACH, B. A. *A communications modeling system for swarm-based sensors*. [S.l.], 2003.

[Kersnovski, Gonzalez e Morton 2017]KERSNOVSKI, T.; GONZALEZ, F.; MORTON, K. A uav system for autonomous target detection and gas sensing. In: IEEE. *Aerospace Conference, 2017 IEEE*. [S.l.], 2017. p. 1–12.

[La, Park e Kim 2017]LA, W. G.; PARK, S.; KIM, H. D-muns: Distributed multiple uavs' network simulator. In: IEEE. *ICUFN*. [S.l.], 2017. p. 15–17.

[Lasnier et al. 2013]LASNIER, G. et al. Distributed simulation of heterogeneous and real-time systems. In: IEEE. *Distributed Simulation and Real Time Applications (DS-RT), 2013 IEEE/ACM 17th International Symposium on*. [S.l.], 2013. p. 55–62.

[Mason et al. 2017]MASON, I. A. et al. A framework for analyzing adaptive autonomous aerial vehicles. In: SPRINGER. *International Conference on Software Engineering and Formal Methods*. [S.l.], 2017. p. 406–422.

[MAVLink Developer Guide 2018]MAVLINK Developer Guide. 2018. Https://mavlink.io/en/. 2018-08-22.

[MAVProxy 2018]MAVPROXY. 2018. Http://ardupilot.github.io/MAVProxy/html/index.html. 2018-08-22.

[Noulard, Rousselot e Siron 2009]NOULARD, E.; ROUSSELOT, J.-Y.; SIRON, P. Certi, an open source rti, why and how. In: *Spring Simulation Interoperability Workshop*. [S.l.: s.n.], 2009. p. 23–27.

[OpenStreetMap 2019]OPENSTREETMAP. 2019. Https://www.openstreetmap.org/about. 2019-04-08.

[Prodanov e Freitas 2013]PRODANOV, C. C.; FREITAS, E. C. de. *Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico-2ª Edição*. [S.l.]: Editora Feevale, 2013.

[Ptolemaeus 2014]PTOLEMAEUS, C. (Ed.). *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014. Disponível em: <http://ptolemy.org/books/Systems>.

[Ramesh et al. 2015]RAMESH, K. et al. Automatic detection of powerlines in uav remote sensed images. In: IEEE. *Condition Assessment Techniques in Electrical Systems (CATCON), 2015 International Conference on*. [S.l.], 2015. p. 17–21.

[Ross 2014]ROSS, P. E. Open-source drones for fun and profit. *IEEE Spectrum*, IEEE, v. 51, n. 3, p. 54–59, 2014.

[Siddiqui et al. 2017]SIDDIQUI, K. T. A. et al. Development of a swarm uav simulator integrating realistic motion control models for disaster operations. *arXiv preprint arXiv:1704.07335*, 2017.

[SITL Simulator (Software in the Loop) 2018]SITL Simulator (Software in the Loop). 2018. Http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html. Access: 2018-07-27.

[Wei, Madey e Blake 2013]WEI, Y.; MADEY, G. R.; BLAKE, M. B. Agent-based simulation for uav swarm mission planning and execution. In: *Proceedings of the Agent-Directed Simulation Symposium*. San Diego, CA, USA: Society for Computer Simulation International, 2013. (ADSS 13), p. 2:1–2:8. ISBN 978-1-62748-029-1.

[Yuan, Liu e Zhang 2015]YUAN, C.; LIU, Z.; ZHANG, Y. Uav-based forest fire detection and tracking using image processing techniques. In: IEEE. *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. [S.l.], 2015. p. 639–643.

[Zema et al. 2017]ZEMA, N. R. et al. Cuscus: Communications-control distributed simulator. In: *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*. [S.l.: s.n.], 2017. p. 601–602.

[Zhang et al. 2015]ZHANG, M. et al. A high fidelity simulator for a quadrotor uav using ros and gazebo. In: IEEE. *IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society*. [S.l.], 2015. p. 002846–002851.

[Zhou et al. 2016]ZHOU, G. et al. Robust real-time uav based power line detection and tracking. In: IEEE. *2016 IEEE International Conference on Image Processing (ICIP)*. [S.l.], 2016. p. 744–748.