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

UMA FERRAMENTA PARA AVALIAR ESTRATÉGIAS DE VOOS DE VANTS USANDO COSSIMULAÇÃO

JOSÉ DE SOUSA BARROS

JOÃO PESSOA – PARAÍBA – BRASIL
ABRIL / 2017

Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Informática

Uma Ferramenta para Avaliar Estratégias de Voos de VANTs usando Cossimulação

José de Sousa Barros

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.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sinais, Sistemas Digitais e Gráficos

Dr. Alisson Vasconcelos de Brito
(Orientador)

João Pessoa, Paraíba, Brasil

B277f Barros, José de Sousa.
Uma ferramenta para avaliar estratégias de voos de VANTs usando cossimulação / José de Sousa Barros. - João Pessoa, 2017.
76 f. : il. -

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

1. Informática. 2. Cossimulação. 3. Ptolemy. 4. VANT. 5. HLA. 6. SITL. 7. ArduPilot. I. Título.

UFPB/BC CDU: 004(043)

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

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de JOSÉ DE SOUSA BARROS, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 26 de abril de 2017.

Aos vinte e seis dias do mês de abril, do ano de dois mil e dezessete, às dez 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. José De Sousa Barros, vinculado a esta Universidade sob a matrícula nº 2015103394, 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 (UFPB), Examinador Interno, e Paulo Eduardo e Silva Barbosa (UEPB), 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 "Uma ferramenta para avaliar estratégias de voos de VANTs 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, 26 de abril de 2017.

Prof. Dr. Clauirton de Albuquerque Siebra

Clauirton de Albuquerque Siebra
Coordenador do Programa de
Pós-Graduação em Informática
SIAPE 1723491

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

Prof. Dr. Vivek Nigam
Examinador interno (PPGI-UFPB)

Prof. Dr. Paulo Eduardo e Silva Barbosa
Examinador Externo à Instituição (UEPB)

Resumo

Sistemas que utilizam Veículos Aéreos Não-Tripulados (VANT) são exemplos típicos de sistemas ciber-físicos. Projetar tais sistemas não é uma tarefa trivial porque traz consigo o desafio de lidar com a incerteza, que é algo inerente a este tipo de sistema. Por isso, é importante que o projeto seja feito com ferramentas apropriadas que possam viabilizar a execução desses sistemas com um certo nível de confiança para seus usuários. Deste modo, a proposta deste trabalho é unir dois simuladores, através do HLA, com o objetivo de simular e avaliar estratégias de voos mais próximas do ambiente de voo real. Para isso, foi construído um simulador onde é possível realizar diversos planos de voo com a finalidade de analisar diferentes estratégias em um ambiente provido de incertezas. O simulador foi desenvolvido na ferramenta Ptolemy e integrado, através do HLA, com o simulador SITL/ArduPilot. Os resultados mostram que com a utilização da abordagem defendida neste trabalho é possível obter resultados mais próximos da realidade, assim estratégias mais eficientes de voo podem ser desenvolvidas e avaliadas.

Palavras-chave: Cossimulação, VANT, Ptolemy, HLA, SITL, ArduPilot.

Abstract

Systems using Unmanned Aerial Vehicles (UAV) are typical examples of cyber-physical systems. Designing such systems is not a trivial task because it brings the challenge of dealing with the uncertainty that is inherent to this type of system. Therefore, it is important the usage of appropriate tools for design that can ensure implementation of these systems with a certain level of confiability. Thus, the purpose of this work is to integrate two simulators via HLA in order to simulate and evaluate different flights strategies. For this, it is presented a simulation environment that can execute flight plans in order to evaluate different strategies in uncertain scenarios. The simulator was developed in Ptolemy and integrated with SITL/ArduPilot via HLA. The results show that with the use of the approach presented in this paper it is possible to obtain results closer to reality, thus more efficient flight strategies can be developed and evaluate.

Keywords: Cossimulation, VANT, Ptolemy, HLA, SITL, ArduPilot.

Agradecimentos

A Deus por sua graça permitir minha existência e por permitir que mais uma etapa de minha vida seja completada.

A minha esposa Aganice pelo apoio, companheirismo, paciência e compreensão que sempre teve comigo e durante o desenvolvimento deste projeto.

Aos meus filhos, Filipe e Letícia, que suportaram minha ausência em brincadeiras e em outras situações em que estive ausente por estar ocupado com as atividades deste trabalho.

A minha irmã Rivânia e a minha Sogra Mara que sempre se disponibilizaram para nos ajudar com as crianças. Isso foi fundamental para que eu pudesse ter o tempo necessário para o desenvolvimento das minhas atividades no mestrado.

A minha mãe por ser meu exemplo de força e fé, por me ensinar a ser quem sou e a não desanimar mesmo quando as circunstâncias pareçam mostrar que esta é a opção que deva ser escolhida.

Ao meu orientador Professor Dr. Alisson Brito por ser um orientador responsável, compreensivo e conduzir com respeito e maestria a orientação deste trabalho.

Ao Professor Dr. Vivek Nigam pelo apoio e pelas contribuições dadas desde quando este projeto ainda estava no início.

Aos integrantes do LaSER pelo apoio e a todos que contribuíram de alguma maneira com a realização deste trabalho.

A sabedoria é a coisa principal; adquire pois a sabedoria, emprega tudo o que possuis na aquisição de entendimento.

Provérbios 4:7

Dedico este trabalho a Deus, aos meus familiares, em especial a minha esposa Aganice, aos meus filhos Filipe e Letícia, e a minha mãe Rozileide.

Conteúdo

Lista de Abreviaturas e Siglas

FED : *Federation Execution Details*

FOM : *Federation Object Model*

GPS : *Global Positioning System*

GCS : *Ground Control Station*

HLA : *High Level Architecture*

IEEE : *Institute of Electrical and Electronics Engineers*

OMT : *Object Model Template*

RTI : *Run-Time Infrastructure*

VANT : Veículo Aéreo Não Tripulado

XML : *Extensible Markup Language*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

Os Veículos Aéreos Não Tripulados (VANTs) são aeronaves capazes de realizar missões de voo sem a presença de uma tripulação a bordo para controlar o veículo. Eles são exemplos típicos de sistemas ciber-físicos porque contém elementos computacionais que interagem com o ambiente físico onde estão inseridos. A maioria destes veículos é controlada por rádio através de um piloto em terra. No contexto atual das tecnologias da aviação, a utilização deste tipo de veículo está em ascensão, despertando o interesse de empresas, instituições e indivíduos para um grande número de diferentes aplicações [Sineglazov e Godny 2015] [Yapp, Seker e Babiceanu 2016] [Xiang et al. 2016].

Os VANTs estão sendo cada vez mais utilizados em várias situações, tais como entrega de pacotes, filmagens e captura de imagens em eventos, monitoramento de áreas agrícolas, oceanografia e outras inúmeras possibilidades de utilização para várias finalidades [David e Ballado 2016] [Leira, Johansen e Fossen 2015]. Estes veículos estão presentes em várias áreas e possuem inúmeras utilidades de uso podendo ir da simples diversão pessoal até ao monitoramento e ataque a tropas inimigas quando em uso militar. Mais recentemente, alguns VANTs são controlados por um computador embarcado executando aplicações para realizar missões pré-programadas.

Nos últimos anos o poder computacional dos dispositivos embarcados tem aumentado consideravelmente à medida que o custo desses dispositivos tem diminuído, conforme foi previsto na Lei de Moore [Moore 1998]. Esse avanço permitiu o desenvolvimento de sistemas com um nível complexidade que não era possível anteriormente, e fez surgir a demanda por ambientes de projeto e simulação que acompanhassem essa evolução.

Sistemas embarcados complexos geralmente são heterogêneos, isso significa que um mesmo modelo pode ser composto por módulos distintos em relação às linguagens de programação utilizadas, aos níveis de abstração e à combinação de partes de software e de hardware. Por isso, o projeto desse tipo de sistema necessita de diversas ferramentas específicas para o modelo a ser desenvolvido. O Ptolemy [Ptolemaeus 2014] é uma ferramenta que, dentre outras funcionalidades, permite o projeto e a simulação de sistemas embarcados heterogêneos em um único ambiente.

Especificamente sobre os Veículos Aéreos Não-Tripulados (VANTs), a maioria das ferramentas existentes permite ao projetista planejar o voo através da configuração de pontos de passagem específicos, mas tem suporte limitado para avaliar diferentes estratégias em ambientes com a presença de incertezas. Mesmo assim, existem muitas ferramentas consolidadas especializadas em simular e avaliar características específicas. Assim, uma solução poderia ser a integração de diferentes simuladores em um único framework.

O foco deste trabalho é a integração entre simuladores heterogêneos para analisar diferentes estratégias de voo e viabilizar a simulação de voos de Veículos Aéreos Não-Tripulados (VANTs) em um ambiente com a representação de incertezas. Para a integração do Ptolemy com outros simuladores, foi utilizado um padrão chamado *High Level Architecture*, ou HLA [Rules 2010]. O HLA especifica uma infraestrutura para a gerência do tempo nos diversos simuladores, chamada *Run-Time Infrastructure*, ou RTI. Ela permite a integração de sistemas heterogêneos de forma síncrona e transparente, aproveitando todos sistemas legados.

Para representar o ambiente de voo da maneira mais real possível, foi utilizado o SITL/ArduPilot[SITL Simulator (Software in the Loop) 2016], que simula um piloto automático utilizado em muitos VANTs, chamado ArduPilot. Ele permite a simulação de voos de VANTs com o uso de mapas, GPS e outros recursos que o aproximam do ambiente real de voo. No ambiente proposto neste trabalho, de um lado estão as estratégias de voo desenvolvidas utilizando o Ptolemy, e do outro lado o ambiente de voo disponibilizado no SITL/ArduPilot. Os comandos da estratégia são enviados do Ptolemy para o SITL/ArduPilot através do HLA que é responsável pela integração dos dois simuladores.

A principal contribuição deste trabalho é: 1) o desenvolvimento de um framework de simulação (Ptolemy-SITL/ArduPilot) para avaliar diferentes estratégias de voo; 2) o uso do HLA para integração futura com outras ferramentas e até mesmo um VANT físico; 3) a

realização de experimentos como prova de conceito que demonstram a possibilidade de uso do framework para avaliar diferentes estratégias de voos de VANTs.

Este trabalho parte da hipótese de que o projetar sistemas ciber-físicos é uma tarefa complexa e, sendo assim, não há uma única ferramenta capaz de preencher todas demandas necessárias ao projeto desses sistemas. Dessa forma, faz-se necessária a cooperação entre diversas ferramentas de uma forma simples e sincronizada e que aproveite as ferramentas e conhecimentos prévios. Como principal contribuição e relevância científica deste trabalho devem ser destacados o simulador de aplicações para VANTs implementado no Ptolemy e a integração, através do HLA, do Ptolemy com o SITL/ArduPilot para simulação de incertezas.

1.1 Objetivos

Nesta seção são apresentados os objetivos deste trabalho.

1.1.1 Objetivo Geral

Desenvolver uma ferramenta onde seja possível analisar estratégias de voos de Veículos Aéreos Não Tripulados em um ambiente com a presença de incertezas, aproximando desse modo o ambiente simulado do ambiente real de voo.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Desenvolver uma ferramenta que seja capaz de oferecer dados suficientes para determinar que estratégia é mais adequada para cada cenário de aplicação de VANT;
- Desenvolver uma ferramenta que permita a integração futura com outros simuladores e até com VANTs reais;
- Validar o funcionamento da ferramenta através de um estudo de caso onde o comportamento de estratégias de voos possa ser analisado.

1.2 Metodologia

A cossimulação é uma abordagem de simulação que combina diferentes simuladores especializados em diferentes campos em um único ambiente de simulação [Sicklinger et al. 2014]. Este trabalho propõe o desenvolvimento de uma ferramenta para análise de estratégias de voos de VANTs usando cossimulação. Para isso, foi necessário seguir alguns procedimentos que viabilizaram alcançar os resultados almejados.

A metodologia empregada seguiu os passos descritos a seguir:

- Estudo sobre o funcionamento e uso dos componentes denominados atores no Ptolemy;
- Estudo do funcionamento da arquitetura de alto nível HLA;
- Implementação de um módulo que possibilite a substituição de estratégias de voo através do recurso visual para arrastar e soltar componentes no Ptolemy;
- Utilização da API DroneKit-Python para interação com um VANT no módulo de ambiente de voo representado pelo SITL/ArduPilot;
- Integração através do HLA do módulo de configuração das estratégias feito no Ptolemy com o módulo de ambiente de voo feito no SITL/ArduPilot;
- Definição de um estudo de caso com dois cenários para análise do comportamento das estratégias de voos;
- Para cada cenário, realizar várias simulações de voos com o uso da estratégia A, para cada possibilidade de ocorrência de vento;
- Para cada cenário, realizar várias simulações de voos com o uso da estratégia B, para cada possibilidade de ocorrência de vento;
- Análise dos resultados providos pela simulação do estudo de caso.

1.3 Trabalhos Relacionados

Existem várias ferramentas para simulação que auxiliam no projeto e validação de sistemas embarcados [Ptolemaeus 2014], [Forin, Neekzad e Lynch 2006], [Accellera 2015] e, por isso, o uso dessas ferramentas não deve ser menosprezado pelos projetistas por serem aliadas de grande relevância no desenvolvimento desses sistemas. Apesar disso, a simulação com o uso de apenas uma ferramenta nem sempre consegue atender a todas as características dos sistemas embarcados, porque esses sistemas geralmente são complexos e heterogêneos. Por este motivo, este assunto vem ganhando atenção dos pesquisadores.

No trabalho [Cheung, Hao e Xie 2007], é proposta uma nova abordagem para cossimulação que unifica hardware e software com o conceito de bridge (ponte). Nessa abordagem os simuladores Giano e ModelSim são especificamente integrados. Diferentemente do nosso trabalho que propõe a integração entre simuladores através de um padrão consolidado, os autores usam o conceito de um componente ponte para interface de hardware com partes de software, tudo em uma solução específica para um cenário, enquanto a nossa abordagem é baseada em HLA [Rules 2010], um padrão consolidado, que permite a integração de qualquer outro componente compatível com HLA.

Em [Ren et al. 2014], é proposto um método baseado em Matlab/Simulink, que consiste em modelagem, simulação, verificação e geração de código. Os códigos de software e protótipos de sistemas embarcados podem ser verificados passo-a-passo por software e hardware através da cossimulação em Matlab/Simulink. As ferramentas utilizadas no método proposto são proprietárias e comercializadas por meio de licenças, enquanto que as utilizadas no nosso trabalho são baseadas em padrões e open-source.

A cossimulação também é utilizada em [Hsu, Wen e Wang 2007], que representa uma plataforma de software que pode ser usada para auxiliar o projeto de sistemas embarcados com múltiplos processadores. A solução proposta baseia-se na interação entre o modo de usuário Linux, utilizado para abstrair o modelo de processador real programável onde o software embutido deve ser executado, e o dispositivo de hardware simulado por SystemC. Nessa plataforma é possível realizar a prototipagem virtual de novos dispositivos de hardware para ser adicionado a um sistema baseado em PC, de modo distinto dessa abordagem, na nossa proposta acontece a integração entre diferentes simuladores para permitir a simulação de

estratégias de voos de VANTs.

No trabalho [Fitzgerald, Pierce e Larsen 2014], é descrita uma abordagem colaborativa que permite aos engenheiros de diferentes áreas a construção de modelos individuais na notação mais adequada, e ainda permite a cossimulação destes diversos modelos em uma plataforma comum. A abordagem foi realizada com o uso da tecnologia Crescendo[1], que permite a definição e simulação de co-modelos compostos por modelos de Eventos Discretos expressos na notação VDM (Vienna Development Method) utilizando a ferramenta Overture[2] e modelos de Tempo Contínuo expressos usando o framework 20-sim[3]. Crescendo permite que os dois modelos constituintes executem em seus simuladores distintos, transferindo os dados e gerenciando o tempo de simulação entre eles, de maneira diferente dessa abordagem que permite a cossimulação entre as ferramentas Overture e o framework 20-sim, no nosso trabalho a transferência de dados e o gerenciamento do tempo é feito pelo HLA.

Uma plataforma de simulação distribuída usando HLA para projetos de sistemas embarcados é proposta por [Brito et al. 2015], onde são apresentados os resultados experimentais de cinco diferentes cenários, que integram cinco diferentes ferramentas de simulação: Ptolemy II, SystemC, Omnet ++, Veins, Stage e robôs físicos. Os experimentos foram realizados com sucesso na aplicação de redes de sensores sem fio, estimativa de energia no projeto de circuitos, simulação robótica e co-simulação de robôs reais. Apesar de utilizar o Ptolemy e o HLA que também são utilizados neste trabalho, o trabalho citado difere do desenvolvido nesta proposta porque aqui foi realizado a integração com o SITL/ArduPilot para representar o comportamento de VANTs, e o trabalho citado não apresenta este tipo de aplicação.

Em [Lasnier et al. 2013] é apresentado um framework para simulação distribuída de sistemas ciber-físicos (CPS). No framework citado são utilizados o Ptolemy e o HLA, também são apresentadas as extensões do Ptolemy para a interação com HLA e a abordagem é demonstrada em uma simulação de um sistema de controle de voo. Mesmo o trabalho citado tendo utilizado o Ptolemy e o HLA, ele difere do desenvolvido nesta proposta porque aqui foi realizado a integração com o SITL/ArduPilot para representar o comportamento de VANTs, enquanto que no trabalho citado o comportamento da aeronave é desenvolvida no próprio Ptolemy e não representa detalhes do ambiente físico.

[1] http://www.crescendotool.org/

[2] http://overturetool.org/

[3] http://www.20sim.com/

No trabalho [Kanduri et al. 2013], os autores tentam explorar plataformas de modelagem para o projeto de sistemas ciber-físicos. Um estudo de caso com Veículos Aéreos Não Tripulados é modelado e simulado utilizando o Ptolemy. Na conclusão os autores afirmam sentir que a adição de mais detalhes para os processos físicos traria credibilidade para o projeto. Em nosso trabalho seguimos essa sugestão e, mesmo as estratégias de voo tendo sido implementadas no Ptolemy, o ambiente de voo acrescenta detalhes do ambiente físico por termos utilizado o SITL/Ardupilot, uma ferramenta específica para simulação de voos de VANTs.

O projeto de sistemas para VANTs necessita se apoiar em ferramentas que sejam capazes de fornecer detalhes do estado do próprio VANT e da sua interação com o ambiente de voo. Além disso, existem as incertezas no ambiente onde o sistema irá atuar que devem ser levadas em consideração durante o projeto, porque elas influenciarão de alguma maneira no funcionamento do sistema.

Nesse cenário, a proposta deste trabalho é a construção de um ambiente de simulação onde seja possível realizar análise de estratégias de voos utilizando cossimulação. A ideia é aproveitar o que cada ferramenta ou simulador tem de melhor e usar isso com o objetivo de simular e analisar estratégias de voos de VANTs. Para isso, será realizada a integração entre o Ptolemy [Ptolemaeus 2014] e o SITL/ArduPilot [SITL Simulator (Software in the Loop) 2016] utilizando o HLA [Rules 2010] como middleware.

1.4 Estrutura da Dissertação

A organização deste trabalho de dissertação está estruturalmente dividida em cinco capítulos. O Capítulo 1 apresenta o contexto geral onde este trabalho está inserido e também os trabalhos relacionados. No Capítulo 2 são apresentados os conteúdos necessários para compreensão dos conceitos utilizados na proposta apresentada. O Capítulo 3 por sua vez discorre sobre o desenvolvimento do ambiente proposto. No Capítulo 4 é apresentado um estudo de caso e os resultados obtidos. Por fim, o Capítulo 5 apresenta a conclusão e os trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 High Level Architecture

O uso de simuladores heterogêneos permite a realização de simulações que abstraem diferentes situações do mundo real e, apesar disso, é difícil encontrar um único ambiente, mesmo sendo heterogêneo, que atenda a todas as necessidades de todos os tipos de aplicações. Deste modo, novos simuladores são criados para atender as novas demandas por novas formas de simulação. Neste cenário, existem questões genéricas recorrentes que devem ser levadas em consideração na integração desses novos simuladores e possuir uma arquitetura padrão ajuda nesse sentido.

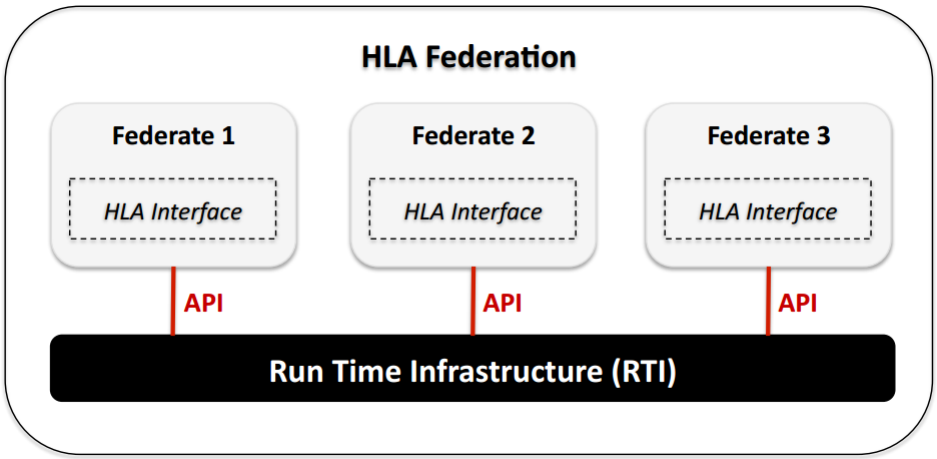
A *High Level Architecture* (HLA) é o nome dado para uma especificação de uma arquitetura padrão para integração de simuladores heterogêneos. A especificação é composta por três documentos diferentes, o primeiro documento [Rules 2010] especifica o framework e as regras para interação entre os federados em uma federação, o segundo documento [Interface 2010] especifica a interface para comunicação entre federados que permite que diferentes simuladores sejam conectados e coordenados, e o terceiro documento [OMT 2010] especifica um padrão de documentação para definição do modelo de dados que é transferido entre os federados.

Um componente essencial na arquitetura HLA é a *Run-Time Infrastructure* (RTI), ela é a especificação de um software que fornece os serviços de interface comuns para sincronização e troca de dados entre os federados em uma simulação [Rules 2010]. Por ser uma especificação de um software, existem várias implementações comerciais e não comerciais, entre

as comerciais podem ser citadas a *Pitch pRTI*[1] e a *SimWare RTI*[2], e entre as não comerciais podem ser citadas a *CERTI*[3] e a implementação feita no *Portico*[4].

A arquitetura geral de uma federação no HLA pode ser visualizada na Figura 2.1, onde podem ser visualizados os principais componentes de uma federação, os federados e a RTI. Uma federação é a combinação de um determinado *Federation Object Model* (FOM), um conjunto particular de federados e os serviços da interface HLA [Interface 2010]. Por sua vez, um federado é qualquer software que implementa os serviços da interface HLA e, por isso, é capaz se unir a uma federação para interagir com outros federados enviando ou recebendo informações.

Figura 2.1: Arquitetura geral de uma federação HLA.



Fonte: [Lasnier et al. 2013]

A interface HLA disponibiliza serviços que são implementados pela RTI e que devem ser chamados pelos federados para, por exemplo, iniciar uma nova federação, se tornar membro de uma federação ativa, enviar dados, receber dados, etc. Estes federados por sua vez devem implementar os serviços (métodos) da interface HLA para serem invocados pela RTI em *callbacks*, ou chamadas de retorno, eles são chamados no federado em resposta a alguma solicitação feita pelo próprio federado, ou para o recebimento dos dados de interesse enviados por outro federado, ou quando a RTI informa sobre a situação da federação.

[1] http://www.pitchtechnologies.com/products/prti/
[2] http://www.simware.es/
[3] http://savannah.nongnu.org/projects/certi/
[4] http://www.porticoproject.org/comingsoon/

A criação de uma federação acontece quando um dos federados invoca o serviço *createFederationExecution* na RTI. Uma vez que a federação tenha sido criada, é possível unir qualquer federado à federação através da chamada ao método *joinFederationExecution*. Depois disso, os federados devem informar qual é a classe e os atributos que o federado está interessado em receber ou publicar informações. O último federado a ser executado é quem deve registrar um rótulo ou uma etiqueta com o nome do ponto de sincronização.

Numa federação normalmente existem vários simuladores federados e essa federação que eles participam é responsável por sincronizar todos eles para então iniciar uma simulação. A sincronização acontece quando todos os federados indicam que alcançaram o ponto de sincronização registrado previamente por um dos federados e que depois disso foi anunciado pela RTI. Além disso, o HLA é responsável também pelo gerenciamento centralizado do tempo dos federados de uma federação.

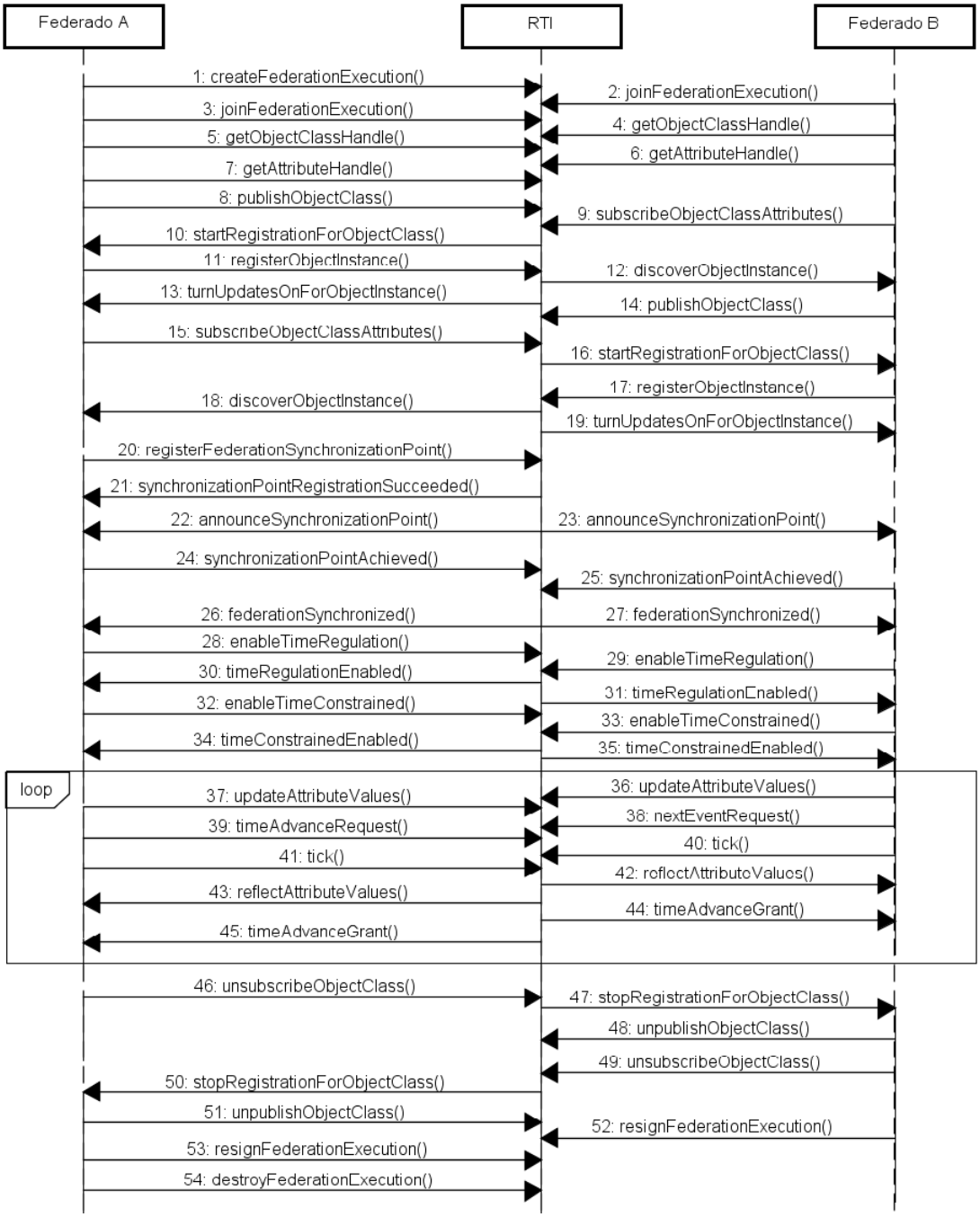
Esse gerenciamento está relacionado aos mecanismos disponibilizados para permitir o avanço do tempo lógico dos federados no tempo da federação. Quando um federado se une a uma federação ele pode se habilitar para ser regulador do tempo ao invocar o serviço *enableTimeRegulation* na RTI. Isso significa que o federado será consultado nas solicitações para avanço no tempo e pode enviar mensagens ordenadas no tempo com base na data e hora (*timestamp order* - TSO). Além disso, as mensagens não podem ser enviadas com o tempo menor que o tempo do federado mais um tempo mínimo não negativo configurado chamado de *lookahead*.

O federado unido à federação pode também indicar que se submete às restrições de tempo ao invocar o serviço *enableTimeConstrained* na RTI, isso significa que o federado está habilitado a receber mensagens ordenadas com base na data e hora. Em relação ao avanço do tempo lógico, o federado pode invocar dois serviços para solicitar o avanço no tempo, o *timeAdvanceRequest*, quando trabalha com um avanço de tempo de tamanho fixo, ou *nextEventRequest*, quando trabalha com um avanço de tempo de tamanho variável. Em ambos os casos a RTI invoca o método *reflectAttributeValues* no federado antes de permitir o avanço no tempo e o atualiza com todas as mensagens com tempo menor ou igual ao tempo especificado na solicitação, e só depois disso a RTI invoca o serviço *timeAdvanceGrant* concedendo a permissão para avançar no tempo.

Existem serviços que são implementados pela RTI e serviços que são implementados

pelos federados. Os principais serviços que são implementados pela RTI que um federado deve invocar são listados na Tabela 2.2, e do mesmo modo os principais serviços que são implementados pelos federados que a RTI deve chamar são listados na Tabela 2.1. Uma vez que os componentes possuam as respectivas implementações da interface HLA, é possível iniciar uma simulação com simuladores diferentes unidos em uma mesma federação.

Figura 2.2: Diagrama de sequência de uma simulação HLA.



A Figura 2.2 mostra um diagrama de sequência com um exemplo de uma simulação com dois simuladores, e neste exemplo é apresentada a sequência de chamadas aos serviços tanto nos federados quanto na RTI. Inicialmente, através da invocação do método *createFederationExecution* no passo 1, a federação é criada e dois federados se unem a ela ao invocar nos passos 2 e 3 o método *joinFederationExecution*, depois disso nos passos 4 a 7 os federados sinalizam a classe e os atributos deseja manipular. Isso é feito com a invocação dos métodos *getObjectClassHandle* e *getAttributeHandle* respectivamente.

Na sequência, cada federado publica qual é a classe para a qual irá fornecer informações e indica qual é a classe que assina para receber informações de outro federado. Isso é feito através da invocação dos serviços *publishObjectClass* e *subscribeObjectClassAttributes* que aparecem nos passos 8 a 19. Em seguida, no passo 20, um dos federados solicita o registro de um rótulo que será o ponto de sincronização através da invocação do método *registerFederationSynchronizationPoint*, e no passo 21 a RTI confirma que o registro foi realizado com sucesso ao invocar no federado o método *synchronizationPointRegistrationSucceeded*.

Após isso, nos passos 22 e 23 a RTI anuncia a todos os federados da federação qual é o ponto de sincronização registrado através da invocação do método *announceSynchronizationPoint* em todos os federados. Por sua vez, os federados informam nos passos 24 e 25 que alcançaram o ponto de sincronização através da chamada ao método *synchronizationPointAchieved* da RTI. Nos passos 26 e 27 a a RTI informa que a federação está sincronizada ao invocar o método *federationSynchronized* nos federados. Então, cada federado habilita suas restrições relacionadas ao tempo nos passos 28 a 35.

A simulação acontece especificamente nos passos 36 a 45 com a troca de dados entre os simuladores exibida no fragmento *loop*, onde o federado publica novos valores na RTI através da chamada ao método *updateAttributeValues* e em seguida solicita o avanço do seu tempo lógico com a chamada do serviço *timeAdvanceRequest* ou *nextEventRequest*. A RTI, por sua vez, invoca o método *reflectAttributeValues* no federado e o atualiza com as informações que ele tem interesse, e depois disso libera o avanço no tempo com a chamada ao serviço *timeAdvanceGrant*.

A finalização da federação acontece dos passos 46 ao 54, onde os federados cancelam a assinatura que possuíam para os objetos de uma determinada classe através da chamada ao método *unsubscribeObjectClass* e, depois disso, cancelam a publicação de informações para

a classe que forneciam informações através da invocação do método *unpublishObjectClass*. Em seguida, os federados se desassociam da federação ao invocar o método *resignFederationExecution* e, quando a federação não possui nenhum federado associado, então é destruída através da invocação do método *destroyFederationExecution* na RTI.

Tabela 2.1: Serviços da interface HLA chamados pela RTI nos federados.

Serviço	**Descrição**
synchronizationPointRegistration Succeeded	Indica que o registro do ponto de sincronização foi realizado com sucesso.
announceSynchronizationPoint	Anuncia aos federados de uma federação qual é o ponto de sincronização.
federationSynchronized	Informa aos federados que a federação está sincronizada porque que o ponto de sincronização foi alcançado por todos.
discoverObjectInstance	Informa ao federado associado a existência de uma instância de um objeto.
turnUpdatesOnForObjectInstance	Informa ao federado associado que os valores dos atributos do objeto são necessários para a federação.
reflectAttributeValues	Recebe uma atualização com os novos valores dos atributos de interesse do federado.
timeRegulationEnabled	Indica que um pedido prévio para habilitar a regulação do tempo foi aceito.
timeConstrainedEnabled	Indica que um pedido prévio para habilitar a restrição de tempo foi aceito.
timeAdvanceGrant	Informa que um pedido prévio para avançar o tempo lógico do federado foi aceito.

Tabela 2.2: Serviços da interface HLA chamados pelos federados na RTI.

Serviço	**Descrição**
createFederationExecution	Cria uma nova federação e para isto um FOM deve ser fornecido.
joinFederationExecution	Associa o federado a uma federação.
resignFederationExecution	Desassocia o federado de uma federação.
destroyFederationExecution	Destrói uma federação da RTI.
registerFederationSynchronizationPoint	Solicita o registro de um rótulo que será o ponto de sincronização.
synchronizationPoint Achieved	Informa a RTI que o federado alcançou ponto de sincronização registrado.
getObjectClassHandle	Retorna a classe do objeto que será manipulado.
getAttributeHandle	Retorna o atributo que será manipulado.
publishObjectClass	Publica a classe que contém as informações que são fornecidas pelo federado.
subscribeObjectClass Attributes	Assina o federado como interessado nas instâncias de uma classe.
unsubscribeObjectClass	Cancela a assinatura do federado como interessado nas instâncias de uma classe.
unpublishObjectClass	Cancela a publicação da classe que possui as informações fornecidas pelo federado.
registerObjectInstance	Registra uma instância da classe publicada com as informações do federado.
updateAttributeValues	Atualiza os valores dos atributos da instância registrada para a RTI.
enableTimeRegulation	Habilita o federado a regular o avanço do tempo em outros federados.
enableTimeConstrained	Habilita o federado a se submeter a restrição do tempo.
timeAdvanceRequest	Solicita o avanço do tempo lógico do federado.
nextEventRequest ou nextMessageRequest	Solicita o avanço do tempo lógico do federado para o tempo da próxima mensagem.

2.2 Ptolemy

O Ptolemy é uma ferramenta *open-source* para modelagem e simulação de sistemas heterogêneos destinada a experimentar técnicas de projeto, particularmente aquelas que envolvem combinações de diferentes tipos de modelos de computação [Ptolemaeus 2014]. Ele foi desenvolvido por pesquisadores no Departamento de Engenharia Elétrica e Ciências da Computação da Universidade da Califórnia em Berkeley, e vem sendo utilizado por pesquisadores de todo o mundo.

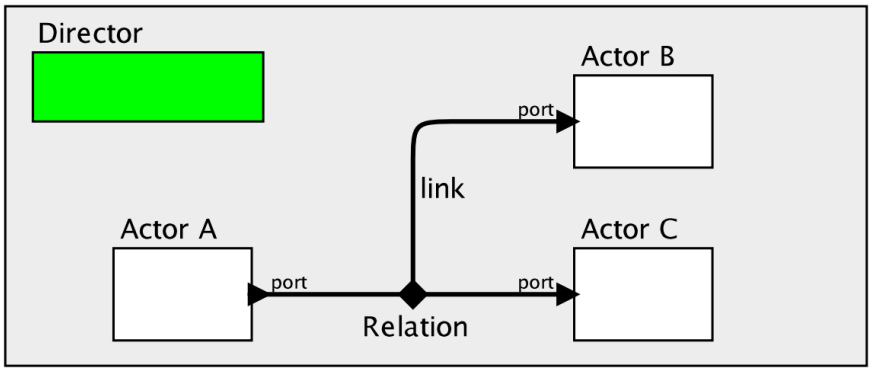
Um modelo de computação é o nome dado para um conjunto de comportamentos dos elementos existentes em um domínio específico, ele representa como acontece a interação desses elementos em um projeto. Existem vários tipos de modelos de computação e como exemplo desses modelos podem ser citados o modelo de eventos discretos (*Discrete Event* - DE) e o modelo de Máquinas de Estado Finito (*Finite State Machines* - FSM). Cada modelo possui uma coleção de regras que governam a execução dos componentes e a comunicação entre eles. No Ptolemy o componente denominado diretor é o responsável pela representação e implementação de um modelo de computação em particular.

O Ptolemy é baseado no modelo orientado a atores ou modelo de atores [Ptolemaeus 2014] e por isso os componentes de um projeto desenvolvido nesta ferramenta são chamados de atores. Eles são gerenciados por um diretor em um modelo simples, porém um modelo pode possuir submodelos representados pelos atores compostos. Neste caso, para cada submodelo pode possuir um diretor específico, que governa localmente os atores, ou não possuir nenhum diretor, quando os atores serão governados pelo diretor do modelo de nível superior. A Figura 2.3 exibe a representação visual de um modelo de ator com a presença de apenas um diretor. Do mesmo modo, a Figura 2.4 mostra a representação visual de um modelo de ator com a presença de mais de um diretor.

A criação de um modelo pode ser feito com o uso de uma organização hierárquica, onde atores simples e compostos são combinados. Um ator composto representa um submodelo e pode possuir um diretor local com seu próprio modelo de computação. Cada ator composto está em um nível na hierarquia, e cada nível possui um comportamento homogêneo localmente, porém níveis diferentes podem possuir diretores diferentes, o que significa que o modelo hierárquico normalmente é heterogêneo porque diretores diferentes representam

modelos de computação diferentes.

Figura 2.3: Representação visual de um modelo de atores.



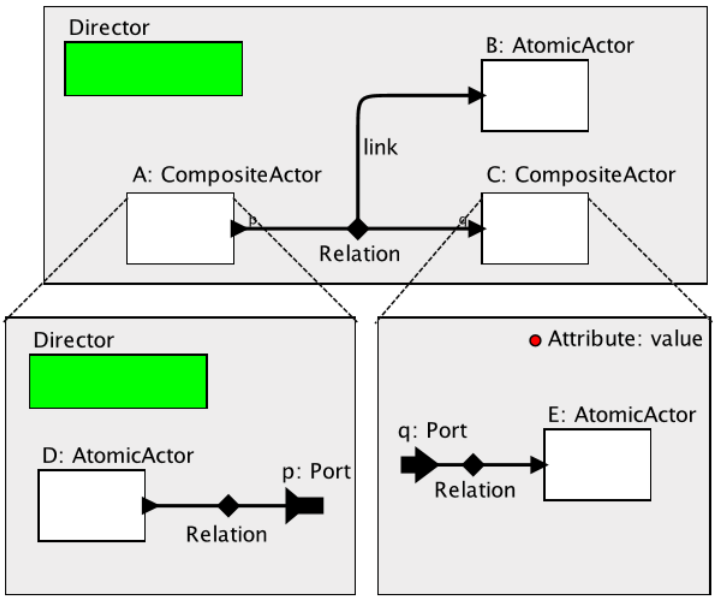
Fonte: [Ptolemaeus 2014]

O modelo de atores é uma teoria matemática onde as coisas podem ser vistas como sendo "atores", e estes são considerados os elementos básicos da computação digital [Hewitt 2010]. Seguindo este raciocínio, o ator é um elemento chave no modelo orientado a atores da mesma forma que o objeto é um elemento chave no modelo orientado a objetos. Os atores podem ser simples ou compostos, um ator simples é aquele que não possui outros atores em sua construção, um exemplo deste tipo de ator é o ator B que é apresentado na Figura 2.4, por sua vez um ator composto é aquele que possui outros atores em sua composição, na Figura 2.4 são apresentados os atores compostos A e C.

A comunicação entre os atores acontece através das portas. Essas portas podem ser simples quando existe apenas um canal de comunicação, múltiplas quando existe mais de um canal de comunicação, de entrada, de saída, ou de entrada e saída ao mesmo tempo. Para um ator enviar uma mensagem para outro ator é necessário fazer uma ligação entre a porta de saída do remetente para a porta de entrada do destinatário, conforme pode ser visualizado nas Figuras 2.3 e 2.4. Além disso, a relação representada nestas figuras pelo diamante indica que os dados que saíram do ator A serão enviados aos atores B e C.

Para [Lasnier et al. 2013] e [Ptolemaeus 2014], o ciclo de vida de um ator no Ptolemy compreende a execução das seguintes fases: *setup, iterate* e *wrapup*. Cada uma dessas fases pode executar uma ou mais ações, a fase de *setup* executa as ações *preinitialize* e *initialize*, por sua vez a fase *iterate* executa as ações *prefire, fire* e *postfire*, finalmente a fase *wrapup* executa apenas uma ação que carrega o próprio nome da fase: *wrapup*. Na Figura 2.5 são

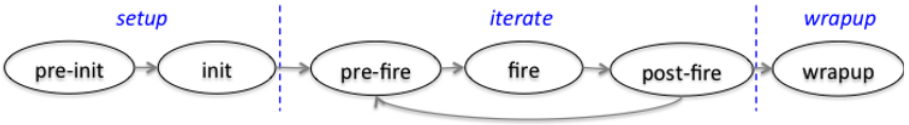
Figura 2.4: Representação visual de um modelo com atores simples e compostos.



Fonte: [Ptolemaeus 2014]

apresentadas as fases e as respectivas ações que acontecem em cada uma delas.

Figura 2.5: Ciclo de vida de um ator no Ptolemy.



Fonte: [LASNIER 2013]

A ação *preinitialize* é executada apenas uma vez em uma simulação. Ela pertence à fase *setup* e deve ser utilizada para fazer as tarefas relacionadas com a programação do ambiente, como a geração de código, a verificação de tipos, a criação de atores quando esta ação pertencer a atores compostos, etc. Outra ação que pertence à fase *setup* é a *initialize*. Ela pode ser executada mais de uma vez em uma simulação e deve ser utilizada para inicializar parâmetros, iniciar o estado local do ator e enviar mensagens iniciais. Uma vez que as ações da fase de *setup* tenham sido finalizadas é o momento de iniciar as ações da próxima fase.

A fase *iterate* é a principal fase em uma simulação e nela as ações *prefire*, *fire* e *postfire* são executadas várias vezes. A ação *prefire* é responsável por validar se existem informações suficientes para a execução da ação *fire*, um exemplo disso é verificar se existem dados para

serem lidos nas portas de entrada. Por sua vez, a ação *fire* é responsável pela leitura das informações nas portas de entrada, pelo processamento de um algoritmo que manipule os dados recebidos e pelo envio das novas informações pelas portas de saída. Uma vez que a ação *fire* tenha sido concluída, então é iniciada ação *postfire* que é responsável por atualizar o estado persistente do ator.

Finalmente, a fase *wrapup* executa a ação que carrega o mesmo nome *wrapup*, e é nesta ação que acontece a finalização de uma simulação. Existe a garantia de que esta ação será executada mesmo se a execução de outras fases falhar com o lançamento de alguma exceção. Esta ação deve ser utilizada para liberação dos recursos que foram alocados durante a inicialização para evitar que fiquem alocados e sem uso, como fechar arquivos, fechar conexões com banco de dados, fechar conexões com a rede, etc.

Além dos atores existentes na base disponibilizada pelo Ptolemy, é possível criar outros para atender as novas demandas que possam surgir. Para implementação de um novo ator é necessário que seja criada uma nova classe na linguagem Java que herde de algum ator já existente no Ptolemy, um exemplo desses atores é o *TypedAtomicActor*. Após isso, é necessário adicionar as portas de entrada e as de saída e, além disso, sobrescrever os métodos que fazem parte do ciclo de vida de um ator no Ptolemy.

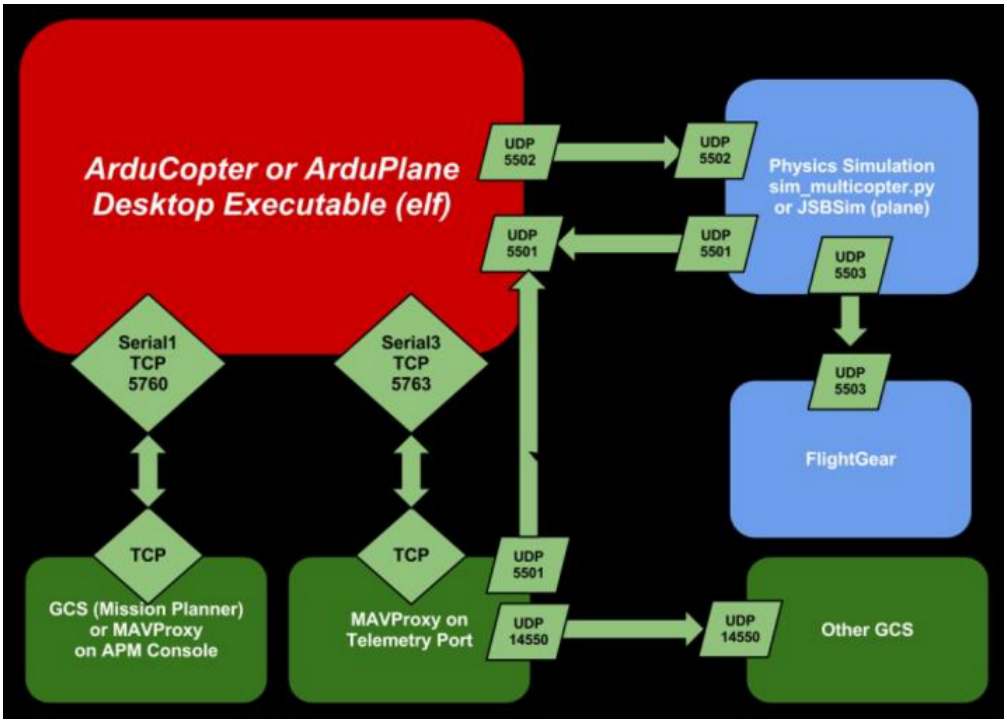
2.3 Simulador SITL/ArduPilot

O SITL/ArduPilot é um simulador *open-source* capaz de representar o comportamento de diferentes tipos de veículos, como multicópteros, helicópteros tradicionais, aeronaves de asa fixa e rovers sem o a necessidade de utilizar qualquer tipo de hardware [SITL Simulator (Software in the Loop) 2016]. Este simulador é uma compilação do código escrito em C++ de um sistema de piloto automático conhecido como ArduPilot. Apesar disso, as ferramentas de suporte são desenvolvidas com o uso de várias linguagens de programação, sendo mais comum o uso da linguagem python.

A arquitetura do simulador SITL/ArduPilot é apresentada na Figura 2.6 onde são apresentados os principais módulos, como eles estão organizados e mostra também que a comunicação entre eles é feita através de conexões de rede que utilizam o protocolo TCP ou o protocolo UDP. Isso promove a liberdade para a escolha da linguagem de programação

que será empregada na construção de novos módulos, porque não obriga o desenvolvedor a utilizar uma linguagem específica. Para visualizar uma simulação é necessário conectar uma *Ground Control Station* (GCS) ao SITL.

Figura 2.6: Arquitetura do SITL/ArduPilot.



Fonte: [SITL Simulator (Software in the Loop) 2016]

Segundo [Team 2016], uma *Ground Control Station* (GCS), ou estação de controle terrestre, é normalmente uma aplicação de software rodando em um computador localizado em solo, que se comunica com um VANT através da telemetria. Uma GCS é capaz de apresentar os dados em tempo real sobre a velocidade, carga da bateria, posição do veículo e, se equipado, até mesmo exibir ao vivo o vídeo capturado pela câmera. Além disso, também pode ser usada para planejar uma missão através da configuração de pontos que serão visitados, controlar o VANT em uma missão de voo do início até o fim, carregar novos comandos de missão e parâmetros de configuração.

Como exemplo de GCS a *MAVProxy* [Dade 2017] pode ser citada, ela foi escrita em python e por isso pode ser executada em vários sistemas operacionais equipados com um interpretador desta linguagem. Em sua execução existe a possibilidade de carregar vários módulos que oferecem suporte para consoles, mapas, joysticks, etc. Além disso, ela foi

desenvolvida para sistemas baseados no protocolo de comunicação *MAVLink* [MAVLink 2017] e permite a adição de outras GCS através da conexão com portas de saídas que podem ser configuradas durante a inicialização, como é mostrado no Código Fonte 2.2.

A API DroneKit foi criada tendo como objetivo permitir o desenvolvimento de aplicações que possam ser executadas em um computador embarcado e que se comunicam com o controlador de voo ArduPilot através de *MAVLink*. Ela foi desenvolvida em python e viabiliza o acesso programático a informações do estado e de parâmetros de um veículo conectado, além de permitir o controle direto sobre o movimento e as operações do veículo [Robotics 2016].

Código Fonte 2.1: Comando para iniciar o SITL/ArduPilot.

```
1 # Executar este comando em um terminal para iniciar o SITL na porta 5760
2 dronekit-sitl copter --home=-7.162675,-34.817705,36,259
```

Código Fonte 2.2: Comando para iniciar a *MAVproxy*.

```
1 # Executar este comando em um terminal para iniciar a GCS MAVproxy
2 mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 127.0.0.1:14550 --out
      127.0.0.1:14551
```

O simulador SITL/ArduPilot pode ser facilmente inicializado com o uso da API DroneKit-SITL[5] e o Código Fonte 2.1 demonstra como isso pode ser feito. Uma vez que o simulador esteja em execução, então a GCS deve ser inicializada, e isso é feito no Código Fonte 2.2 que mostra como inicializar a *MAVproxy*. Depois disso, a interação com o veículo deve ser feita através das funções disponibilizadas pela API DroneKit [Robotics 2016]. Inicialmente, a conexão com o veículo do SITL de ser feita com o uso da função *connect(ip:port, wait_ready=True)* que se conecta ao veículo através da porta configurada na inicialização da *MAVproxy*, conforme é apresentado na linha 4 do Código Fonte 2.3.

Depois disso, o veículo pode ter seus motores ligados e voar para uma determinada altura através da chamada a função *arm_and_takeoff(altitude)*, como é mostrado na linha 7, em seguida para voar em determinada direção a função *send_ned_velocity(x,y,z,duration)* deve ser utilizada, conforme é mostrado na linha 10. Esta função move o VANT com base nos valores de velocidade especificados para x, y e z, sendo que x representa os movimentos em direção ao norte e ao sul, y representa os movimentos em direção ao leste e ao oeste, e z

[5]http://python.dronekit.io/develop/sitl_setup.html

representa os movimentos relacionados a altitude.

Nas linhas 15 a 18 são recuperadas e exibidas as informações relacionadas a localização do VANT. Por sua vez, nas linhas 21 e 22 é recuperada e exibida a informação sobre o nível da bateria do veículo. O pouso do VANT acontece na linha 25 e, depois disso, na linha 27 o objeto *vehicle* é desconectado do veículo do SITL.

Código Fonte 2.3: *Script* para interagir com um veículo do SITL através da *MAVproxy*.

```
1  # Executar depois de iniciar a GCS MAVproxy
2  from dronekit import connect
3
4  vehicle = connect('127.0.0.1:14550', wait_ready=True)
5
6  #Liga e voa para a altitude de 5 metros
7  arm_and_takeoff(5)
8
9  #Move o VANT para o sul.
10 send_ned_velocity(-2,0,0,1)
11 #Move o VANT para o leste.
12 send_ned_velocity(0,2,0,1)
13
14 #Ler as coordenadas em que o VANT se encontra
15 longitude = vehicle.location.global_relative_frame.lon
16 latitude = vehicle.location.global_relative_frame.lat
17 print("Longitude: " + str(longitude))
18 print("Latitude: " + str(latitude))
19
20 #Ler a carga da bateria
21 battery = vehicle.battery.level
22 print("Carga da bateria: " + str(battery))
23
24 #Faz o pouso
25 vehicle.mode = VehicleMode("LAND")
26 #Desconecta do VANT
27 vehicle.close()
```

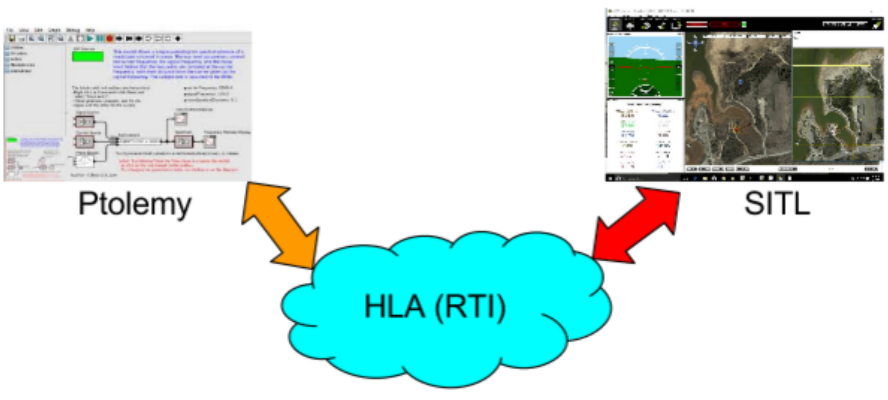
Capítulo 3

Ferramenta para simulação de Voos de VANTs

3.1 Visão Geral da Ferramenta

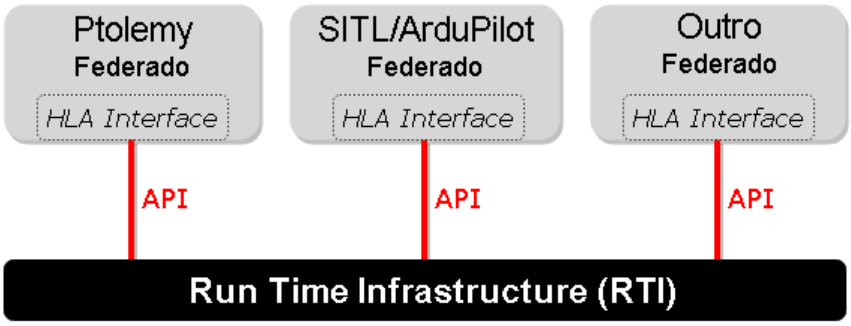
O ambiente proposto neste trabalho é um ferramenta que tem como objetivo simular e analisar diferentes estratégias voos de VANTs em um ambiente com a representação de incertezas. A ferramenta é composta por dois módulos oriundos da integração entre dois simuladores heterogêneos e tem uma visão geral apresentada na Figura 3.1. Um módulo foi desenvolvido no Ptolemy e é responsável pela configuração da estratégia que será utilizada em uma simulação de voo, o outro é responsável pela representação do ambiente de voo e para isso o SITL/ArduPilot foi utilizado. A comunicação entre os módulos é feita através da Arquitetura de Alto Nível - HLA.

Figura 3.1: Visão geral da ferramenta.



Essa comunicação entre os simuladores segue a ideia de que cada simulador deve possuir uma implementação da Interface HLA que é o código que o habilita a ser um federado numa federação HLA, isso pode ser visualizado na Figura 3.2 onde é apresentada a arquitetura geral da ferramenta. Assim, sendo um federado, ele é responsável pelo envio e recebimento de dados de acordo com o padrão HLA. A implementação que habilita o Ptolemy a ser um federado já vem disponibilizada junto com a instalação do próprio Ptolemy [LASNIER 2013], ela faz uso do JCerti[CERTI - Summary 2016], já a implementação para habilitar o SITL/Ardupilot a ser um federado foi desenvolvida com o uso do PyHLA [PyHLA — Python Bindings for M&S HLA 2016].

Figura 3.2: Arquitetura geral da ferramenta.

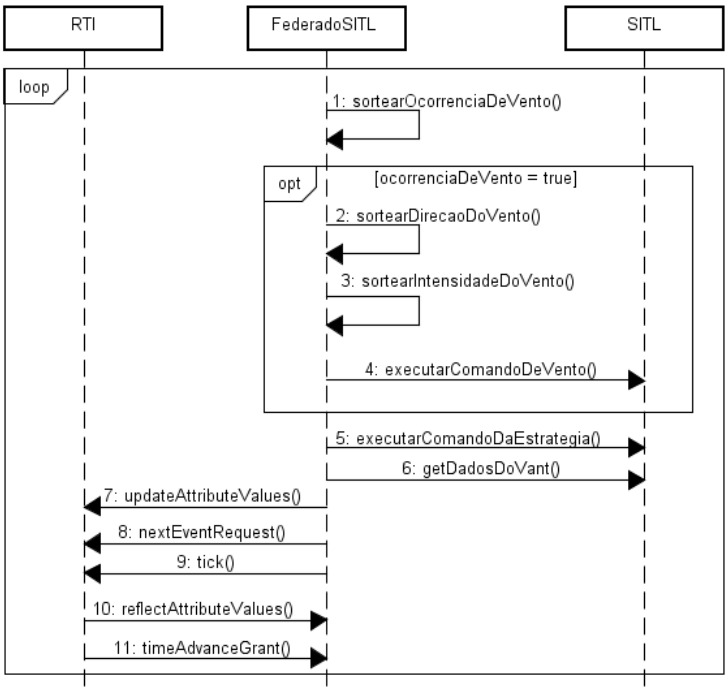


A Figura 3.2 apresenta a arquitetura geral da ferramenta, nela é possível visualizar três federados conectados à RTI e, apesar disso, nesta ferramenta foram utilizados apenas o Ptolemy e o SITL/ArduPilot como federados, o Outro federado apresentado na arquitetura empregada representa a viabilidade para adicionar futuramente outros simuladores federados ou até mesmo um veículo real federado, para isso é necessário que no simulador ou no veículo que será adicionado sejam implementados os serviços especificados na Interface HLA.

O fluxo da troca de dados entre o SITL e a RTI pode ser visualizado no diagrama de sequência apresentado na Figura 3.3, do mesmo modo o fluxo da troca de dados entre o Ptolemy e a RTI pode ser visualizado no diagrama de sequência apresentado na Figura 3.4. A Figura 3.3 mostra que em cada iteração da simulação o federado do SITL faz um sorteio para saber se o ambiente de voo será influenciado pela ocorrência de vento, quando este sorteio indica que haverá ocorrência de vento, são feitos mais dois sorteios, um para indicar qual será a direção e o outro para indicar a intensidade que o vento irá soprar. Em seguida,

é executado um comando no SITL que representa um vento soprando com as características sorteadas.

Figura 3.3: Diagrama de sequência com o fluxo das mensagens entre o SITL e a RTI.

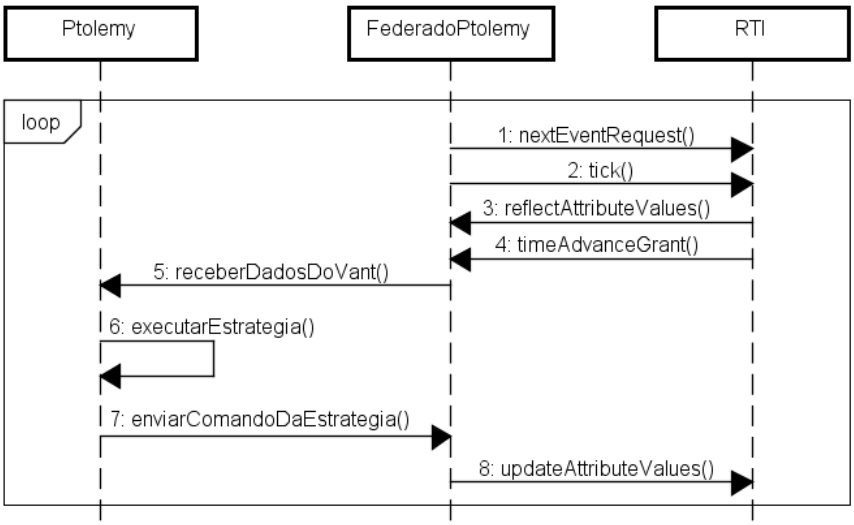


Após isso, o comando para direcionamento do VANT gerado pela estratégia utilizada no Ptolemy é executado no SITL. O próximo passo é recuperar os dados do VANT atualizados para a carga de bateria e a localização e depois enviar para o Ptolemy através da invocação do método *updateAttributeValues* na RTI. Na sequência, o federado do SITL solicita a RTI a permissão para avançar no tempo invocando o método *nextEventRequest* e sinaliza que está esperando por chamadas de retorno da RTI ao invocar o método *tick*. A RTI, por sua vez, invoca no federado o método *reflectAttributeValues* e o atualiza com o novo comando de direcionamento gerado pela estratégia e enviado pelo Ptolemy. Em seguida, a RTI permite o avanço do tempo ao invocar o método *timeAdvanceGrant* no federado.

Para o federado do Ptolemy, a Figura 3.4 mostra que a cada iteração o federado solicita a RTI a permissão para avançar no tempo invocando o método *nextEventRequest* e sinaliza que está esperando por chamadas de retorno da RTI ao invocar o método *tick*. A RTI, por sua vez, invoca no federado o método *reflectAttributeValues* e o atualiza com os dados do VANT relacionados a carga da bateria e a localização enviados pelo SITL. Em seguida, a RTI permite o avanço do tempo ao invocar o método *timeAdvanceGrant* no federado. O fe-

derado atualiza o Ptolemy com os dados do VANT recebidos do SITL e o Ptolemy executa a estratégia configurada. Após a execução da estratégia, o Ptolemy envia o comando de direcionamento gerado pela estratégia para o federado e este envia o comando de direcionamento para o SITL através da invocação do método *updateAttributeValues* na RTI.

Figura 3.4: Diagrama de sequência com o fluxo das mensagens entre o Ptolemy e a RTI.



3.2 Desenvolvimento de Estratégias

Uma estratégia pode ser considerada um plano com um conjunto de ações a serem realizadas para alcançar um objetivo, por exemplo, em um cenário onde se tenha a necessidade de fazer a vigilância de determinada área, o objetivo é tornar a área mais segura e o conjunto de ações que são realizadas de forma organizada para vigiar o local é uma estratégia. Considerando que um VANT é quem vai vigiar o local, é necessário que uma pessoa ou um software envie os comandos necessários, conforme a estratégia adotada, para realização da vigilância.

Neste trabalho o módulo de software com as estratégias de vigilância são desenvolvidas no Ptolemy que é baseado no modelo de atores, e, por isso, as novas funcionalidades normalmente são desenvolvidas com a implementação de novos atores. Isso significa que novas estratégias são implementadas com a construção de novos atores. Essas estratégias podem ser implementadas cada uma em um ator distinto, e é possível também empregar vários atores na construção de uma estratégia.

Basicamente, para implementação de uma nova estratégia com um único ator é necessário

que seja criada uma nova classe na linguagem Java que herde de algum ator já existente no Ptolemy, um exemplo desses atores é o *TypedAtomicActor*. Após isso, é necessário adicionar as portas de entrada e as de saída e, por fim, o método fire() da super classe herdada deve ser sobrescrito com a implementação da nova estratégia. O método fire() é responsável pela leitura das informações nas portas de entrada, pelo processamento da estratégia e pelo envio das novas informações pelas portas de saída, ele é invocado pelo ator diretor em cada iteração na simulação.

Para implementar uma estratégia utilizando vários atores, são realizadas as mesmas etapas empregadas na construção de uma estratégia que emprega um único ator. A diferença básica é que essas etapas serão repetidas na construção de cada um deles e um ator individualmente é responsável apenas pela implementação de um subconjunto das ações da estratégia. Uma vantagem dessa abordagem é que um mesmo ator pode ser aproveitado para compor diferentes estratégias que possuem o mesmo subconjunto de ações implementadas pelo ator, facilitando assim o reuso desses componentes.

Neste trabalho foram desenvolvidas duas estratégias, uma denominada de estratégia A e a outra denominada de estratégia B. Na estratégia A, o VANT visita os pontos alvos na ordem em que estão cadastrados, sem levar em consideração a distância entre a posição atual do VANT e do alvo em questão. Isso significa que mesmo que o primeiro ponto cadastrado esteja distante da posição atual do VANT, e que exista na lista de alvos outro ponto cadastrado mais próximo, o voo do VANT não levará esse fato em questão e irá visitar os pontos seguindo a ordem em que foram cadastrados.

Diferente da estratégia A, na estratégia B o VANT visita os pontos alvos sem considerar a ordem em que foram cadastrados. Nesta estratégia a distância entre a posição atual do VANT e o ponto alvo é levada em consideração. Isso significa que os pontos mais próximos do VANT serão visitados primeiro e só depois é que os mais distantes serão visitados. No desenvolvimento destas duas estratégias foi utilizada a abordagem que emprega várias atores para compor uma estratégia, cada um responsável por um subconjunto de ações que farão parte da estratégia.

A seguir é apresentado uma descrição da função desempenhada por cada um dos atores que foram desenvolvidos para serem utilizados na composição da estratégia A e na composição estratégia B. O tipo de dados *ptolemy.data.type.BaseType.STRING* foi utilizado em todas

as portas de entrada e de saída desses atores.

***Energy*:** possui as portas de entrada *battery* e *position* e a porta de saída *output*. Este ator tem a atribuição de verificar se o VANT pode executar uma ação de movimento na direção norte, sul, leste ou oeste, considerando sua posição atual e a energia disponível na bateria. Para cada ação que é possível de ser executada e ainda retornar para a estação base sem usar a carga reserva da bateria é adicionada a uma lista a respectiva ação e o valor 1.0 (um ponto zero).

No entanto, para aquelas ações que são possíveis de serem executadas e ainda voltar para a estação base apenas se for utilizada a carga reserva da bateria, são adicionadas as respectivas ações e o valor 0.5 (zero ponto cinco). Adicionalmente, para aquelas ações que não são possíveis de serem executadas e ainda voltar para a estação base mesmo utilizando a carga reserva da bateria, são adicionadas as respectivas ações e o valor 0.0 (zero ponto zero). A lista com as ações é ordenada de modo decrescente com base nos valores das ações e enviada pela porta de saída.

***GoPoints*:** possui a porta de entrada *position* e a porta de saída *output*. Este ator é responsável por enviar pela porta de saída os pontos alvos cadastrados para serem visitados ou fotografados na ordem em que foram cadastrados. Isso significa que o primeiro ponto cadastrado é enviado até que seja alcançado pelo VANT, depois disso o segundo ponto da sequência é enviado até que seja alcançado e assim sucessivamente até que todos sejam visitados.

***RankPoints*:** possui a porta de entrada *position* e a porta de saída *output*. Este ator faz um ranking com os pontos cadastrados e o envia pela porta de saída. Este ranking é feito com base na distância dos pontos em relação a posição atual do VANT e no tempo da última visita realizada. Um ponto mais próximo e com o tempo da visita mais antigo deve ser priorizado em relação aquele ponto mais distante e com o tempo de visita mais recente. A pontuação de cada ponto alvo considerada no ranking é feita com base na diferença entre dois somatórios.

O primeiro somatório é feito para os resultados do tempo atual da simulação menos o tempo da última foto de cada ponto. Em seguida, para cada ponto, é feito o segundo somatório para os resultados do tempo futuro do ponto menos o tempo da última foto dos demais pontos. Portanto, a pontuação de um ponto alvo é o resultado do primeiro somatório menos o segundo somatório. O tempo futuro de um ponto é o tempo atual da simulação

mais os passos necessários (cada passo consome uma unidade de tempo) para alcançar o determinado ponto.

***RankActions*:** possui as portas de entrada *inputRankPoints* e *position* e a porta de saída *output*. Este ator é responsável por fazer um ranking com as ações que podem ser realizadas pelo VANT e por enviá-lo pela porta de saída. As ações do ranking são adicionadas com base na posição recebidas pela porta *position* e na pontuação dos pontos recebidos pela porta *inputRankPoints*, elas são ordenadas de forma decrescente, ou seja, as ações para os pontos com maior pontuação aparecem nas primeiras posições da listagem.

***Combine*:** possui as portas de entrada *inputRankActions* e *inputActionsEnergy* e a porta de saída *output*. Este ator tem a atribuição de combinar as ações enviadas pelos atores *Energy* e *RankActions* e enviá-las pela porta de saída ordenadas de forma decrescente com base em dois critérios, primeiro pelo valor enviado pelo ator *Energy* e depois pelo valor enviado pelor ator *RankActions*.

***Selector*:** possui as portas de entrada *inputRankActions* e *position* e a porta de saída *output*. Este ator é responsável por selecionar a primeira ação enviada pelo ator *Combine* e a enviar pela porta de saída. Além disso, quando a ação escolhida é para realizar uma fotografia ele atualiza o tempo da última foto no ponto alvo visitado.

***Log*:** possui as portas de entrada *battery*, *position* e *action* e não possui nenhuma porta de saída. Este ator registra as informações do voo durante a simulação relacionadas ao nível da carga da bateria, à posição do VANT no formato de coordenadas geográficas longitude e latitude, à distância percorrida e à ação enviada pelo ator *Selector*.

Cada um destes atores foram desenvolvidos seguindo aquela abordagem inicial que emprega apenas um ator no desenvolvimento de uma estratégia. A diferença básica é que a estratégia implementada pelo ator é apenas um subconjunto das ações de uma estratégia maior que é composta por vários atores. Os atores desenvolvidos pelo usuário podem fazer parte de uma biblioteca e assim compor uma base que pode ficar disponível para reuso no desenvolvimento de novas estratégias, a Figura 3.5 apresenta a biblioteca do usuário contendo a lista de atores desenvolvida nesse trabalho.

A implementação de uma nova estratégia com vários atores deve ser realizada em um ator composto, e uma maneira simples de fazer isso é através da interface gráfica do Ptolemy, onde se deve arrastar o *CompositeActor* da biblioteca *Utilities*, que aparece selecionado na

Figura 3.5: Biblioteca de atores do usuário.

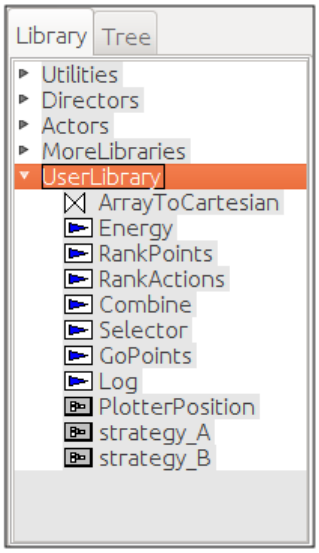
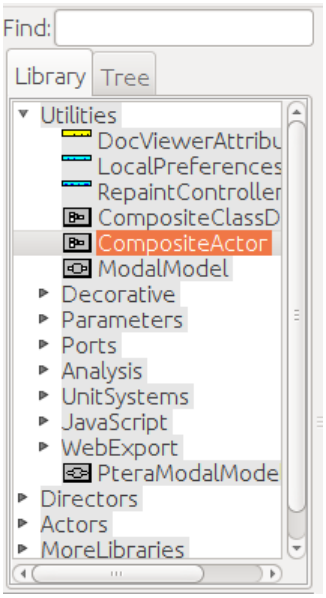


Figura 3.6, e depois clicar no ator com o botão direito, selecionar a opção *Customize* e depois *Rename* para renomeá-lo com o nome da nova estratégia. Após isso, clicar novamente com o botão direito sobre o ator e selecionar a opção *Open Actor* para abrir uma janela vazia para onde deverão ser arrastados e organizados os atores que irão compor a estratégia.

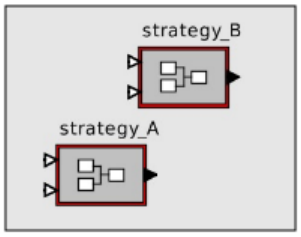
Figura 3.6: Biblioteca *Utilities* com o ator *CompositeActor*.



A implementação da estratégia A foi realizada no ator nomeado *strategy_A*, e a implementação da estratégia B foi realizada no ator nomeado *strategy_B*, ambos possuem as portas

de entrada *battery* e *position* e a porta de saída *output*, eles podem ser visualizados na Figura 3.7. Estes são atores compostos e o detalhamento da composição de cada um deles é apresentado nas Figuras 3.8 e 3.9. No detalhamento do ator *strategy_A* feito na Figura 3.8 são apresentados os atores *Energy, GoPoints, RankActions, Combine, Selector* e *Log*. Do mesmo modo, no detalhamento do ator *strategy_B* feito na Figura 3.9 são apresentados os atores *Energy, RankPoints, RankActions, Combine, Selector* e *Log*.

Figura 3.7: Estratégias A e B.



Percebe-se que os atores apresentados no detalhamento das duas estratégias são basicamente os mesmos, com exceção dos atores *GoPoints*, que aparece apenas para o ator *strategy_A*, e do ator *RankPoints*, que aparece apenas para o ator *strategy_B*. Isso demonstra que a ideia de dividir uma estratégia em partes menores traz a vantagem de promover o reuso dos atores, isso acontece porque ao invés de implementar toda a estratégia em um único ator, apenas uma parte dela é implementada sendo necessário a implementação de vários atores para compor toda a estratégia.

O funcionamento do ator *strategy_A* inicia quando este ator recebe como entrada o nível de bateria e a posição do VANT nas portas *battery* e *position* respectivamente. O ator *Energy* verifica se com o nível da bateria recebido e considerando posição atual do veículo é possível mover nas direções norte, sul, leste ou oeste, em caso afirmativo, adiciona a uma lista a ação e o valor 1.0, caso só consiga mover se utilizar a carga reserva, adiciona a ação e o valor 0.5, e no caso de não ser possível mover mesmo utilizando a carga reserva, adiciona a ação e o valor 0.0. A lista é ordenada de modo decrescente com base nos valores das ações e enviada pela porta de saída.

Figura 3.8: Ator composto *strategy_A*.

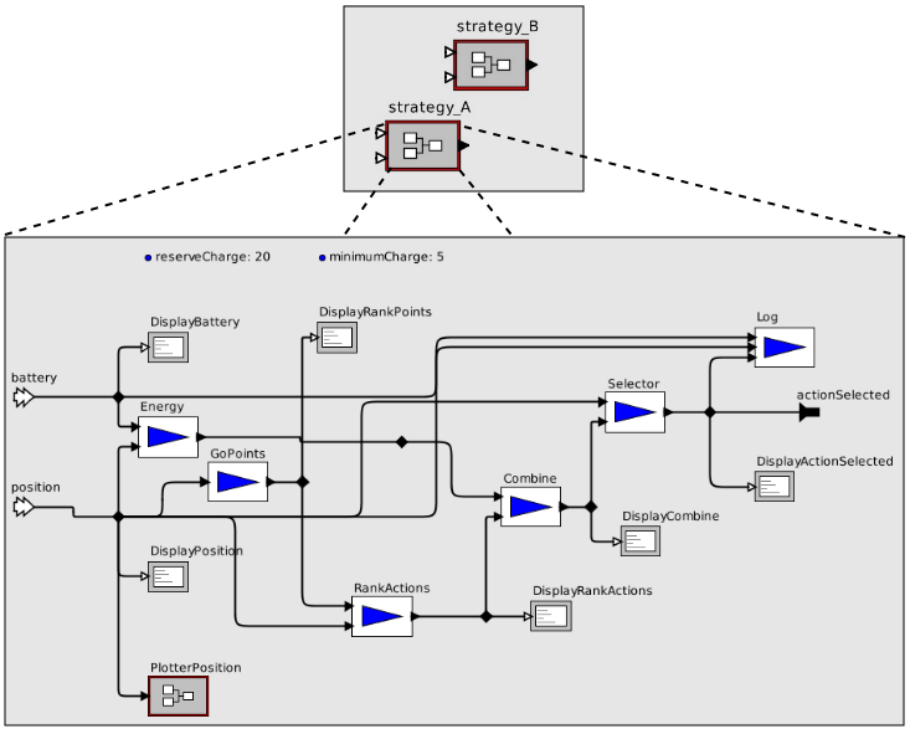
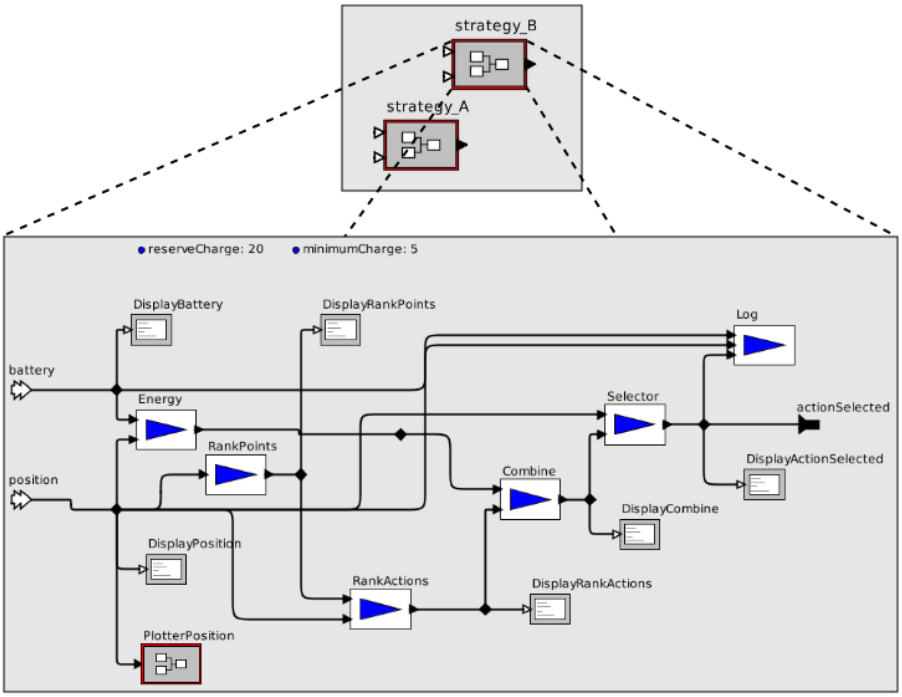


Figura 3.9: Ator composto *strategy_B*.



O ator *GoPoints* verifica qual é o próximo ponto da lista cadastrada a ser visitado e o envia pela porta de saída. Em seguida, o ator *RankActions* faz um ranking com as ações

que podem ser realizadas para os pontos recebidos do *GoPoints* ou do *RankPoints* levando em consideração a posição atual e o envia pela porta de saída. Por sua vez, o ator *Combine* combina as ações que foram recebidas dos atores *Energy* e *RankActions*, ele gera uma lista ordenada de modo decrescente com base em dois critérios, primeiro pela pontuação da lista recebida de *Energy* e depois pela pontuação da lista de *RankActions*.

Na sequência, o ator *Selector* seleciona a primeira ação da lista enviada pelo ator *Combine* e a envia pela porta de saída, a ação selecionada é recebida pelo ator *Log*, que registra as informações do voo, e é enviada para fora da estratégia através da porta *actionSelected*. O funcionamento do ator *strategy_B* possui o mesmos passos que foram apresentados para o ator *strategy_A* com apenas uma diferença, o ator *strategy_A* utiliza o *GoPoints* para indicar qual é o próximo ponto a ser visitado enquanto que *strategy_B* utiliza o *RankPoints* que faz um ranking com os pontos que serão visitados.

3.3 Módulo de Configuração de Estratégias

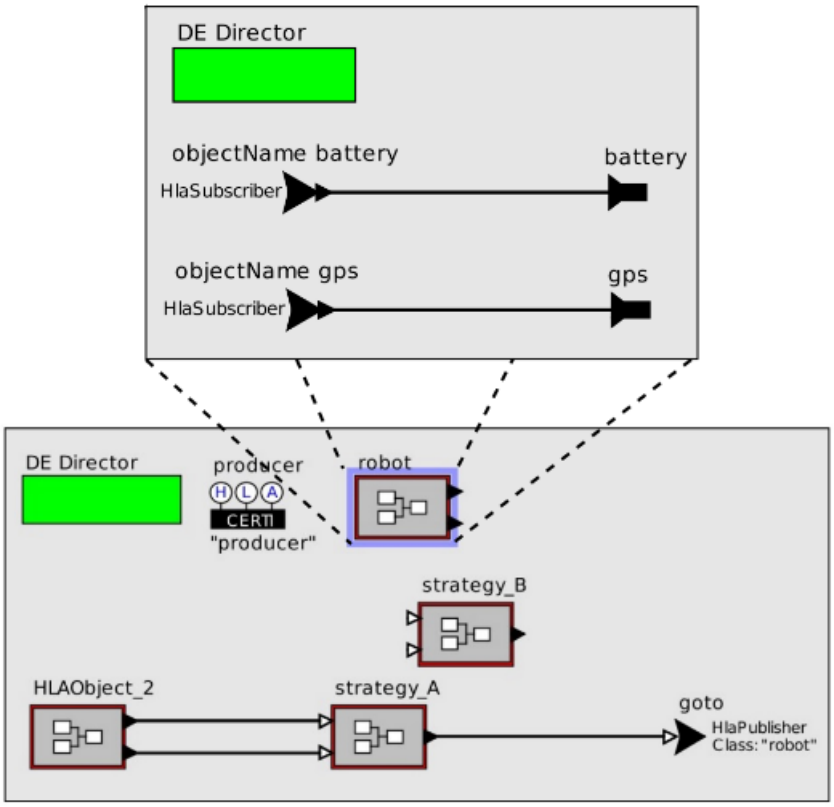
No módulo de configuração de estratégias desenvolvido no Ptolemy, a comunicação com o HLA é realizada pelos atores *HlaManager, HlaSubscriber* e *HlaPublisher*. O *HlaManager* é a implementação do federado do Ptolemy e pode ser visualizado na Figura 3.10 onde aparece com a sigla HLA e com o nome *producer*, ele é responsável por gerenciar de modo centralizado o avanço do tempo entre o Ptolemy e o HLA. O *HlaPublisher* é responsável por publicar na RTI qualquer informação que tenha origem no Ptolemy e seja direcionada a outros federados da mesma federação.

Este ator também é apresentado na Figura 3.10 onde aparece com o nome *goto*, é importante destacar que este nome deve ser igual ao de um atributo da classe que o federado registrou no FOM ao ingressar na federação. Neste módulo, a classe declarada pelo federado do Ptolemy foi a *robot*, ela é apresentada no Código Fonte 3.4 e um dos seus atributos é o *goto* que representa os comandos para direcionamento do VANT originários da estratégia utilizada no Ptolemy.

Por sua vez, o *HlaSubscriber* é responsável por assinar e receber informações publicadas na RTI. As informações de interesse das estratégias são aquelas relacionadas à carga da bateria e à localização do VANT que são publicadas na RTI pelo SITL. Por isso, neste módulo

foram necessários dois atores do tipo *HlaSubscriber*, um com o nome *battery* para receber as informações sobre a carga da bateria e o outro com o nome *gps* para receber as informações sobre a localização do VANT. Esta localização é representada por meio das coordenadas longitude e latitude do sistema de posicionamento global, em inglês *Global Positioning System* - GPS.

Figura 3.10: *HlaSubscriber* dentro do ator composto *robot*.



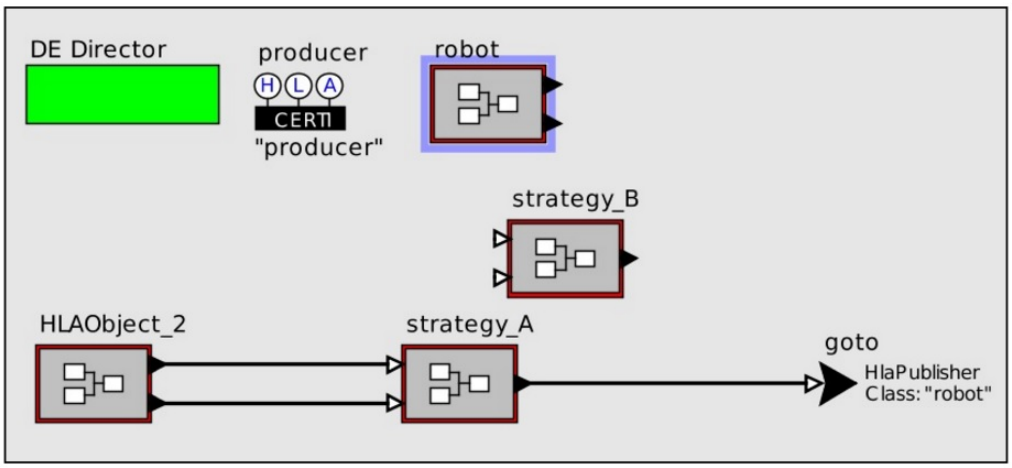
Os dois atores devem também possuir os nomes listados como atributos da classe *robot* declarada no FOM que é apresentado no Código Fonte 3.4, eles foram adicionados dentro de um ator composto da biblioteca *Utilities* chamado *CompositeClassDefinition*. Este, ao ser adicionado ao modelo, teve seu nome alterado para *robot*, por ser o mesmo nome da classe declarada no FOM. Depois disso, foi criada uma instância do ator *robot* através de um clique no ator com o botão direito, depois selecionada a opção *Class Actions* e em seguida a opção *Create Instance*. A instância de *robot* aparece na Figura 3.10 com o nome *HLAObject_2*. Nesta mesma figura é apresentado o detalhamento do ator composto *robot*, onde aparecem os atores *battery* e *gps* que são do tipo *HlaSubscriber*.

O Ptolemy pode ser utilizado de várias maneiras, nele é possível criar modelos com

os componentes preexistentes disponibilizados na própria ferramenta que são chamados de atores ou então criar novos atores para compor o modelo desejado. Uma vez que os atores estejam disponíveis, sejam os da própria ferramenta ou os recém-criados, o projetista pode fazer a seleção dos atores que farão composição de um modelo através da programação em arquivos no formato xml ou, ainda melhor, por meio do editor visual com o recurso de edição *drag and drop* onde um modelo pode ser construído ao arrastar e soltar os componentes.

A Figura 3.11 apresenta o módulo responsável pela configuração da estratégia de voo que será utilizada em uma simulação. Este módulo foi desenvolvido no Ptolemy. Pode-se perceber que o módulo apresentado faz uso de atores, portas simples de comunicação representadas pelas setas preenchidas com a cor preta, portas múltiplas de comunicação representadas pelas setas preenchidas com a cor branca e de canais de comunicação representados pelas linhas que ligam as portas de saída de um ator às portas de entrada de outro ator, todos representados através de componentes gráficos que fazem com que o módulo de configuração de estratégias seja de fácil utilização.

Figura 3.11: Módulo configurado com a estratégia A.



No módulo responsável pela configuração da estratégia de voo foram criados dois atores, um representando a estratégia A e o outro a B. Esse módulo pode ser visualizado na Figura 3.11, onde aparecem os atores *strategy_A* e *strategy_B*. Na figura apresentada, com a finalidade de ilustração, o módulo está configurado para utilizar a estratégia A durante a simulação de voo, e caso se tenha a necessidade de utilizar a estratégia B é necessário apenas que as linhas conectadas nas entradas de A sejam conectadas nas entradas de B e a linha

conectada na saída de A seja conectada na saída de B.

Embora a Figura 3.11 apresente apenas dois atores de estratégias, outras poderão ser adicionadas através da criação de novos atores que as implementem, e do mesmo modo que a estratégia A pode ser substituída pela B, as novas também poderão ser configuradas de maneira semelhante. A implementação de uma nova estratégia é feita com a criação de um novo ator no Ptolemy. Um novo ator pode ser considerado simples ou composto, o ator simples é aquele que não possui outros atores em sua composição, o ator composto é aquele formado a partir da união de outros atores simples ou compostos.

3.4 Módulo do Ambiente de Voo

A Figura 3.12 mostra o módulo responsável pela representação do ambiente de voo do SITL/ArduPilot. Pode-se perceber que a representação do ambiente de voo faz uso de imagens de satélite e mapas (do Google Maps), além disso possui um quadricóptero que consegue gerar informações com valores para localização (no formato das coordenadas geográficas longitude e latitude), velocidade e consumo de bateria que fazem com que o ambiente de voo representado possua detalhes que o aproximam do ambiente real de voo.

Figura 3.12: Ambiente de voo do SITL/ArduPilot.

A comunicação com a RTI foi viabilizada neste módulo através da criação de um federado que é responsável por fazer a implementação da interface HLA. Deste modo, o de-

senvolvimento do federado foi feito com a criação da classe *MyAmbassador* na linguagem Python, onde foi realizada a herança de *hla.rti.FederateAmbassador*, que pertencente a biblioteca PyHLA, e a sobrescrita dos métodos com o código necessário para assegurar a comunicação entre o SITL/ArduPilot e a RTI.

O Código Fonte 3.1 apresenta o método *sendData* que é responsável por ler as informações do VANT e publicá-las na RTI por meio da invocação do método *updateAttributeValues*. Na linha 2 é declarada uma variável que vai armazenar o tempo do federado na simulação, das linhas 3 a 6 os valores da posição e do nível de bateria são obtidos do objeto *vehicle* que pertence a classe *dronekit.Vehicle* da API DroneKit. Na linha 8 a variável que armazena o tempo do federado é atualizada com o tempo atual do federado mais o valor do *lookahead*, da linha 9 até a linha 12 é mostrado o método que publica na RTI com o tempo futuro do federado os novos valores atualizados que foram obtidos do veículo presente no SITL.

Código Fonte 3.1: Envio de informações do SITL/ArduPilot para a RTI.

```
1  def sendData(mya, rtia):
2          global federateTime
3          positionX = vehicle.location.global_relative_frame.lon
4          positionY = vehicle.location.global_relative_frame.lat
5          gps = "[" + str(positionX) + "," + str(positionY) + "]"
6          battery = vehicle.battery.level
7
8          federateTime = rtia.queryFederateTime() + lookahead;
9          rtia.updateAttributeValues(mya.myObject,
10                 {mya.batteryHandle: str(battery) + "0",
11                 mya.gpsHandle: gps},
12                 "update", federateTime)
13
14         ######## Time Management ########
15         rtia.nextEventRequest(federateTime)
16         while (mya.advanceTime == False):
17                 rtia.tick()
18         mya.advanceTime = False
```

Neste caso os valores publicados serão repassados ao Ptolemy para serem processados pela estratégia que estiver sendo utilizada. Em seguida, o federado solicita para avançar para o tempo do próximo evento na linha 15 e, depois disso, nas linhas 16 e 17 fica aguardando a permissão ser concedida com base o valor da variável *advanceTime*, que é atualizada para *True* quando o método *timeAdvanceGrant* é invocado no federado pela RTI e, posterior-

mente, na linha 18 é atualizada para *False*.

Por sua vez, o Código Fonte 3.2 a partir da linha 9 apresenta o método *reflectAttributeValues* que é responsável por refletir no federado as informações originárias da RTI, neste caso as informações que chegam são aquelas que foram geradas pela estratégia em uso e publicadas na RTI pelo Ptolemy. Uma visualização de como acontece a troca de informações na simulação é apresentada nos diagramas de sequência 3.3 e 3.4.

Neste módulo é possível configurar a chance de ocorrência de vento nas simulações, assim os experimentos podem utilizar cada estratégia em duas situações, uma em que o voo acontece normalmente sem interferências do ambiente, quando a chance de ocorrência de vento é configurada para zero, e outra quando a chance de ocorrência de vento configurada é maior que zero, em que o voo pode acontecer sob a influência de vento que empurra o VANT para uma direção não esperada.

Código Fonte 3.2: Recebimento de informações da RTI para o SITL/ArduPilot.

```
1  # Requires PyHLA, see http://www.nongnu.org/certi/PyHLA
2  import hla.rti
3  import hla.omt as fom
4  class MyAmbassador(hla.rti.FederateAmbassador):
5      def initialize(self):
6          self.attributes = None
7      ...
8      # RTI callbacks
9      def reflectAttributeValues(self, object, attributes, tag, order, transport, time=None,
           retraction=None):
10         self.attributes = attributes
11     ...
```

O veículo utilizado nas simulações foi um quadricóptero e para este tipo de veículo o SITL não oferece suporte para adicionar a ocorrência de vento no ambiente de voo. Por isso, foi necessário implementar um código, que é apresentado no Código Fonte 3.3, para emular a ocorrência de vento, e deste modo as incertezas do ambiente de voo foram simuladas com o uso desta implementação. Na linha 1 do código apresentado é feito um sorteio uniforme entre 1 e 100, isso significa que cada número do intervalo considerado tem a mesma possibilidade de ser sorteado. Quando o número sorteado é menor ou igual ao número configurado como sendo a chance de ocorrência de vento, significa que um vento deve acontecer no ambiente de voo e o próximo passo é saber em qual direção ele deve soprar.

Assim, tendo em vista que a direção que o vento sopra também é incerta, na linha 2 é feito um novo sorteio entre os números um e quatro, desta forma quando o resultado é igual a um significa que o vento deve soprar na direção norte, a dois que o vento deve soprar na direção sul, a três que o vento deve soprar na direção leste e a quatro que o vento deve soprar na direção oeste. Além disso, outra incerteza considerada é a intensidade que o vento deve soprar, por isso na linha 4 um novo sorteio é realizado e o seu resultado é interpretado da seguinte maneira: no caso do resultado ser igual a um significa um vento com intensidade fraca, igual a dois um vento com intensidade média e igual a três um vento com intensidade forte.

Código Fonte 3.3: Implementação da ocorrência de vento.

```
1  if random.uniform(1,100) <= chance:
2          direction = random.randint(1,4)
3          # intensidade do vento: 1-fraco, 2-medio,3-Forte
4          intensity = random.randint(1,3)
5          if direction == 1:
6                  send_ned_velocity(NORTH,0,0,DURATION * intensity)
7          elif direction == 2:
8                  send_ned_velocity(SOUTH,0,0,DURATION * intensity)
9          elif direction == 3:
10                 send_ned_velocity(0,EAST,0,DURATION * intensity)
11         elif direction == 4:
12                 send_ned_velocity(0,WEST,0,DURATION * intensity)
```

Este processo de sorteio é realizado em cada iteração da simulação e uma vez que ele tenha sido finalizado, e considerando que um vento deve acontecer, o comando com a interferência do vento simulado é enviado para o ambiente de voo do SITL através da função disponível na API DroneKit *send_ned_velocity()*, que envia um comando para o VANT e o desloca para a uma posição decorrente do resultado do sorteio da direção e da intensidade, simulando assim a ocorrência de um vento que possui como características as informações sorteadas. Depois disso, o comando de direcionamento que foi publicado na RTI pela estratégia utilizada no Ptolemy é enviado para o VANT também através da chamada à função *send_ned_velocity()*, conforme é apresentado no diagrama de sequência 3.3.

3.5 Modelagem de dados

Em uma simulação com o uso do HLA cada federado deve possuir informações para iniciar a execução de uma federação ou se juntar a uma preexistente, além disso deve trocar dados com os demais federados no formato de objetos que necessitam ser definidos seguindo o *Object Model Template* (OMT) [OMT 2010].

As informações para iniciar e gerenciar a execução de uma federação são descritas no arquivo *Federation Execution Details* (FED), onde estão também as informações do *Federation Object Model* (FOM) que tratam da declaração de classes, objetos compartilhados e seus atributos. O Código Fonte 3.4 apresenta o conteúdo do arquivo utilizado, onde é exibida a configuração da classe *robot* com os atributos que poderão ser enviados ou recebidos entre os federados na simulação. Deve existir uma cópia deste arquivo em cada máquina que será integrada à federação.

Código Fonte 3.4: *Federation Execution Details* (FED)

```
1  (FED
2    (Federation TestFed)
3    (FEDversion v1.3)
4    (spaces)
5    (objects
6     (class ObjectRoot
7       (attribute privilegeToDelete reliable timestamp)
8       (class RTIprivate)
9       (class robot
10        (attribute id reliable timestamp)
11        (attribute battery reliable timestamp)
12        (attribute gps reliable timestamp)
13        (attribute goto reliable timestamp)
14      )
15    )
16 )
17 (interactions)
18 )
```

Nos atributos da classe *robot* foi definido que o mecanismo de entrega deve ser confiável e o mecanismo de ordem temporal deve ser baseado na data e hora. Os atributos são: *id* que representa a identificação do federado, *battery* que representa a carga da bateria, *gps* que representa a posição do VANT, e *goto* que representa os comandos de direcionamento que serão enviados para o VANT.

Capítulo 4

Avaliação Experimental

4.1 Estudo de Caso

As simulações foram realizadas em um cenário onde um vigilante deveria visitar alguns pontos e fotografá-los. A partir dessa necessidade, foi definido que o VANT deveria fazer esse trabalho e ficar vigiando uma determinada área, fazendo fotos de determinados pontos. A vigilância poderia ter sido realizada apenas na situação ideal, naquela em que o voo acontece sem interferências externas, no entanto, sabe-se que no ambiente real de voo a situação ideal não é a regra e interferências podem acontecer a qualquer momento. Isso geralmente prejudica o desempenho do voo.

Desse modo, o objetivo deste estudo de caso foi adicionar obstáculos ao ambiente das simulações para aproximá-lo do ambiente real de voo, e assim, para simular o voo sob essas interferências, na área de vigilância da maioria das simulações foram introduzidas intencionalmente possibilidades de ocorrência de ventos que, quando aconteceram, empurraram o VANT e assim interferiram no voo ao provocar o desvio da rota que vinha sendo percorrida. O desafio para as estratégias implementadas foi fazer com que os locais previamente configurados fossem observados e não deixassem de ser fotografados mesmo nos casos sob interferência de vento, porque um ponto não observado é um ponto fora do alcance da vigilância e, portanto, sem cobertura.

Considera-se que quanto maior o número de fotografias de um determinado ponto forem feitas melhor será a vigilância desse local. Desse modo, neste estudo de caso foram definidos dois cenários de vigilância: o Cenário 01 e o Cenário 02. No Cenário 01 foi definido que

Figura 4.1: Pontos alvos do Cenário 01.

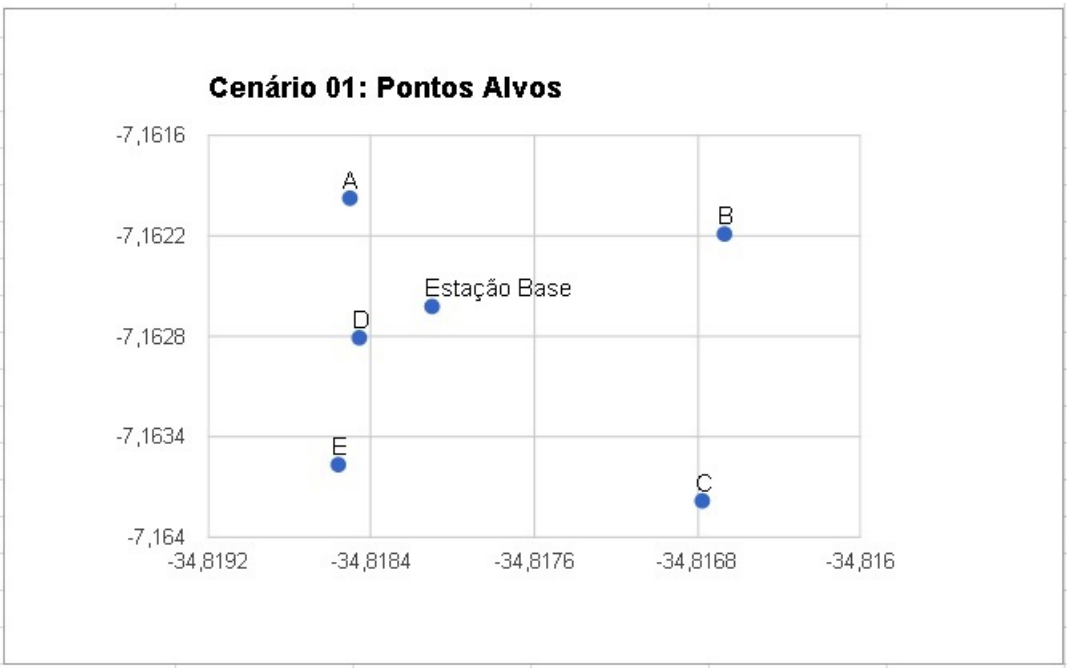
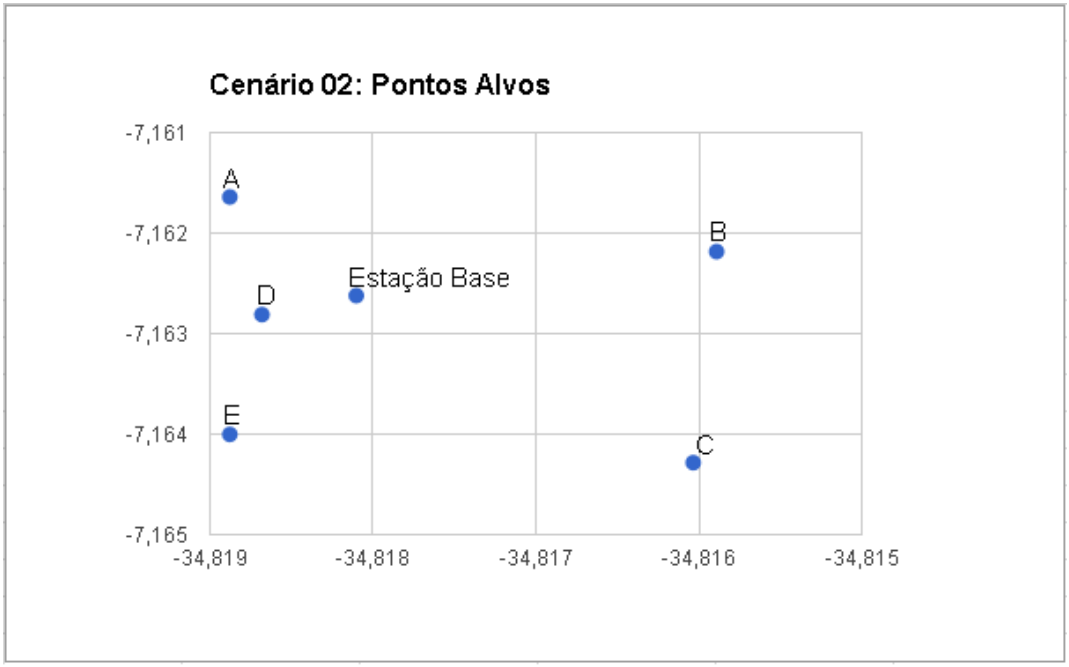


Figura 4.2: Pontos alvos do Cenário 02.



o VANT deveria vigiar os pontos alvos que podem ser visualizados na Figura 4.1. Para o Cenário 02 foi definido que seriam vigiados os pontos alvos que podem ser visualizados na Figura 4.2.

Os dois cenários possuem a mesma quantidade de pontos alvos e são bem parecidos, a

diferença básica entre eles é a extensão da área que será vigiada, uma vez que o Cenário 01 possui um comprimento de trajeto de 884,14 metros e uma área total de 28.577,12 metros quadrados, e o Cenário 02 possui um comprimento de trajeto de 1337,48 metros e uma área total de 62.035,31 metros quadrados. Em todas as simulações realizadas o VANT iniciou a vigilância dos pontos alvos tendo como ponto de partida a estação base, que foi estabelecida em uma mesma localização para ambos os cenários. Na tabela 4.1 é possível visualizar a distância entre os pontos alvos em metros.

Tabela 4.1: Distância entre os pontos alvos em metros.

Cenário 01						
De/Para	**Estação Base**	**A**	**B**	**C**	**D**	**E**
Estação Base	0,00	87,70	166,33	199,62	45,06	121,42
A	87,70	0,00	204,57	284,74	97,49	185,94
B	166,33	204,57	0,00	186,45	210,70	263,89
C	199,62	284,74	186,45	0,00	217,98	198,90
D	45,06	97,49	210,70	217,98	0,00	89,20
E	121,42	185,94	263,89	198,90	89,20	0,00

Cenário 02						
De/Para	**Estação Base**	**A**	**B**	**C**	**D**	**E**
Estação Base	0,00	143,09	249,48	299,80	67,64	182,52
A	143,09	0,00	336,01	440,47	138,31	275,72
B	249,48	336,01	0,00	245,64	316,69	392,22
C	299,80	440,47	245,64	0,00	339,30	315,87
D	67,64	138,31	316,69	339,30	0,00	140,82
E	182,52	275,72	392,22	315,87	140,82	0,00

A configuração dos pontos alvos é realizada no ator de estratégia, porque é ele quem tem a lógica de como os pontos serão visitados. No ambiente proposto foram apresentados os atores *strategy_A*, que possui a implementação da estratégia A, e *strategy_B*, que possui a implementação da estratégia B, ambos podem ser visualizados na Figura 3.8. Na estratégia A o

VANT visita os pontos alvos na ordem em que estão cadastrados, sem levar em consideração a distância entre a posição atual do VANT e do alvo em questão. Isso significa que mesmo que o primeiro ponto cadastrado esteja distante da posição atual do VANT, e que exista na lista de alvos outro ponto cadastrado mais próximo, o voo do VANT não levará esse fato em questão e irá visitar os pontos seguindo a ordem em que foram cadastrados.

Diferente da estratégia A, na estratégia B o VANT visita os pontos alvos sem considerar a ordem em que foram cadastrados. Nesta estratégia a distância entre a posição atual do VANT e o ponto alvo é levada em consideração. Isso significa que os pontos mais próximos do VANT serão visitados primeiro e só depois é que os mais distantes serão visitados. Ambos os atores foram configurados para iniciar o voo na estação base e após isso visitar os pontos alvos. Após a simulação do Cenário 01 ter sido concluída, iniciou-se a preparação para simular o Cenário 02 e os atores de estratégia foram configurados, do mesmo modo que no Cenário 01, para iniciar o voo na estação base e após isso visitar os pontos alvos do cenário em questão.

Tanto para o Cenário 01 quanto para o Cenário 02, as simulações foram realizadas nas seguintes situações, cada uma sendo uma possibilidade de ocorrência de vento: 0%, 2.5%, 5%, 8.3%, 12.5% e 25%. Onde 0% foi o valor mínimo para ocorrência de vento considerado, ou seja o cenário ideal quando não acontecem ventos, e 25% foi o valor máximo, sendo o pior cenário simulado. Cada simulação iniciou com o nível da carga da bateria em 100%, realizou a decolagem e prosseguiu com a execução de várias iterações até atingir o final da simulação. O processo de finalização de uma simulação foi iniciado sempre que o nível da carga da bateria atingiu 20%, quando então o VANT tentou voltar para a estação base e pousar, finalizando assim a simulação, porém quando nesse processo o nível da carga da bateria atingiu 5% o pouso foi feito imediatamente independente de ter ou não conseguido chegar à estação base, encerrando também a simulação.

Para cada possibilidade de ocorrência de vento foram executadas 153 simulações para a estratégia A e, da mesma maneira, 153 simulações para a estratégia B. Cada situação de ocorrência de vento consumiu cerca de 8 horas para executar as 153 simulações para a estratégia A, e o mesmo tempo foi necessário para realizar as 153 simulações para a estratégia B. Isso significa que para o Cenário 01 com a possibilidade de 0% de ocorrência de vento foram consumidas cerca de 16 horas de simulação, 8 horas para a estratégia A e 8 horas para a

estratégia B. O mesmo tempo foi necessário para simular cada uma das outras possibilidades de ocorrência de vento, tanto as do Cenário 01 quanto as do Cenário 02. O nível de confiança calculado para o tamanho da amostra (153 voos simulados) foi de 95% e a margem de erro considerada foi de 8%, este cálculo foi realizado através do uso da calculadora amostral disponibilizada por [SANTOS 2017].

4.2 Configuração do Ambiente

Os experimentos foram realizados em um ambiente equipado com uma máquina física tendo uma máquina virtual instalada. A configuração geral da máquina física é apresentada na Tabela 4.2, do mesmo modo a configuração geral da máquina virtual é apresentada na Tabela 4.3. Os softwares utilizados foram instalados na máquina virtual mencionada, eles são listados com suas respectivas versões na Tabela 4.4.

Tabela 4.2: Configuração da máquina física.

Tipo	Notebook
Marca	Dell
Modelo	Inspiron 14 Série 3000 - I14-3443-A30
Processador	Intel® Core™ i5-5200U 2.20 GHz
Memória RAM	4,00 GB
Disco Rígido	1 TB
Sistema Operacional	Windows10 Home Single Language
Tipo de Sistema	Sistema Operacional de 64 bits

Tabela 4.3: Configuração da máquina virtual.

Software de Virtualização	Oracle VM VirtualBox - 5.1.8
Processador	Intel® Core™ i5-5200U 2.20 GHz
Memória RAM	1,7 GB
Disco Rígido	72,3 GB
Sistema Operacional	Ubuntu 14.04 LTS
Tipo de Sistema	Sistema Operacional de 64 bits

Tabela 4.4: Softwares utilizados.

Software	Versão
Java Development Kit	1.8.0_101
Eclipse IDE for Java Developers	Mars Release (4.5.0)
Python	2.7.6
Ptolemy II	11.0.devel
CERTI/RTIG	3.5.1
PyHLA	1.1.1
MAVProxy	1.5.2
DroneKit-SITL	3.2.0

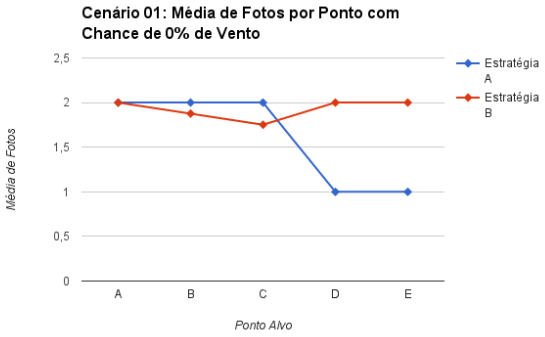
4.3 Resultados

Os dados das simulações foram coletados para viabilizar a comparação do comportamento das estratégias A e B, que foram utilizadas nas simulações, tanto no Cenário 01 quanto no Cenário 02, com as seguintes configurações de possibilidade de ocorrência de vento: 0%, 2.5%, 5%, 8.3%, 12.5% e 25%. Desse modo, as estratégias foram avaliadas com base nos dados, de cada cenário e possibilidade de ocorrência de vento, utilizando os seguintes critérios de comparação: média de fotos feitas em cada ponto alvo, média geral de fotos feitas, carga da bateria no momento do pouso, capacidade para retornar à estação base e a distância média percorrida.

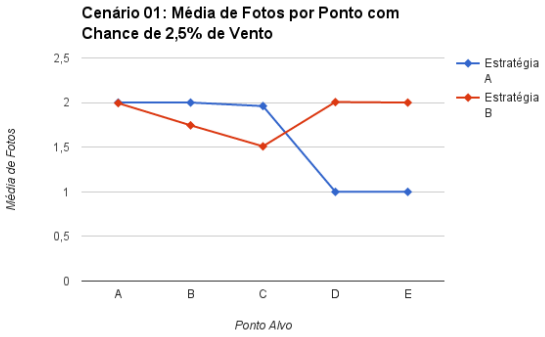
Inicialmente, são apresentados os resultados relacionados à média de fotos feitas em cada ponto alvo, essas médias foram calculadas, para cada ponto individualmente, através da divisão da soma dos números de fotos resultantes de cada simulação individual pelo número total de simulações realizadas (153). Os pontos alvos aparecem no eixo X e as médias de fotos capturadas no eixo Y da Figura 4.3, onde é possível visualizar o desempenho das estratégias A e B para o Cenário 01.

Figura 4.3: Média de fotos por ponto do Cenário 01.

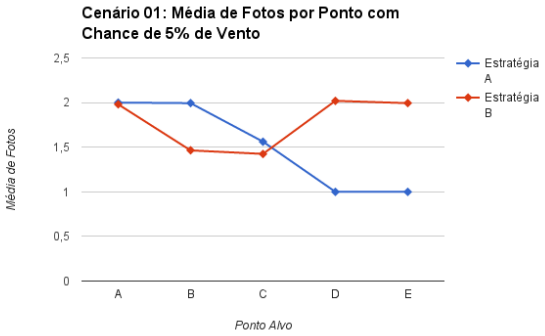
(a) 0% de chance de ocorrência de vento.



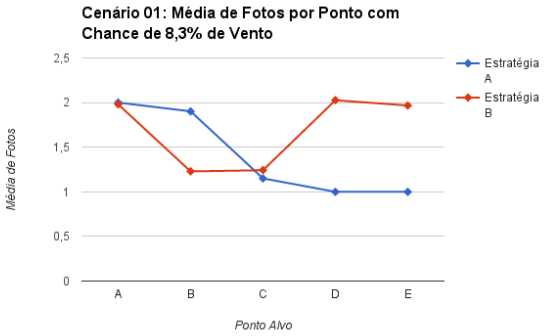
(b) 2.5% de chance de ocorrência de vento.



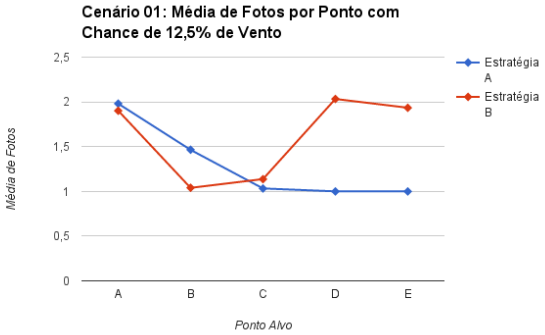
(c) 5% de chance de ocorrência de vento.



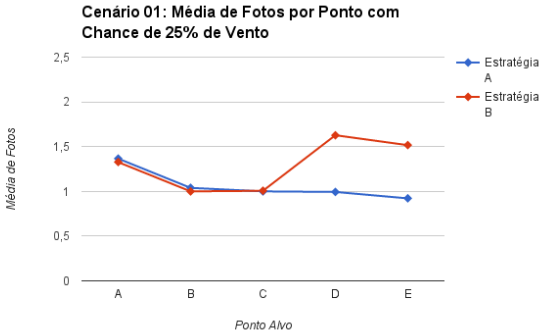
(d) 8.3% de chance de ocorrência de vento.



(e) 12.5% de chance de ocorrência de vento.



(f) 25% de chance de ocorrência de vento.



Pode-se perceber no cenário apresentado que a média de fotos do ponto A foi equivalente para as duas estratégias simuladas em todos os casos de possibilidade de ocorrência de vento configurados, isso pode ser visualizado nas Figuras 4.3a, 4.3b, 4.3c, 4.3d, 4.3e e 4.3f, já para o ponto B, a média de fotos da estratégia A foi melhor que a média da estratégia B nos casos com 0%, 2.5%, 5%, 8.3% e 12.5% de possibilidade de ocorrência de vento, isso pode ser observado nas Figuras 4.3a, 4.3b, 4.3c, 4.3d e 4.3e, porém as duas estratégias foram equivalentes no caso configurado com a possibilidade de 25%, isso pode ser observado na Figura 4.3f.

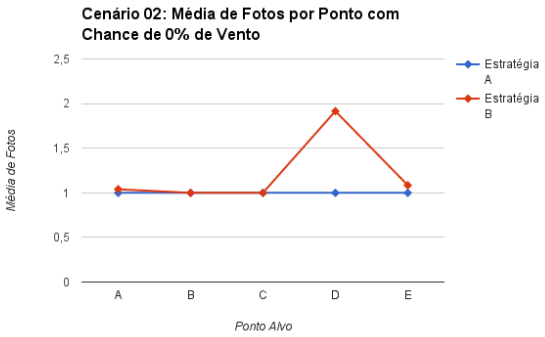
Para o ponto C a média de fotos da estratégia A é melhor que a média da estratégia B nos casos com 0%, 2.5% e 5% de possibilidade de ocorrência de vento, isso pode ser observado nas Figuras 4.3a, 4.3b e 4.3c, porém nos casos com 8,3% e 12.5% de possibilidade de ocorrência de vento a média de fotos da estratégia B é melhor que a média da estratégia A, isso pode ser observado nas Figuras 4.3d e 4.3e, no entanto o desempenho das duas estratégias é equivalente no caso configurado com a possibilidade de 25%, isso pode ser observado na Figura 4.3f. Para os pontos D e E a média de fotos da estratégia B foi melhor que a média da estratégia A em todos os casos de ocorrência de vento simulados, isso pode ser observado nas Figuras 4.3a, 4.3b, 4.3c, 4.3d, 4.3e e 4.3f.

De modo semelhante, os pontos alvos aparecem no eixo X e as médias de fotos capturadas no eixo Y da Figura 4.4, onde é possível visualizar o desempenho das estratégias A e B para o Cenário 02. Pode-se perceber no cenário apresentado que as estratégias A e B possuíram médias de fotos capturadas equivalentes para os pontos A, B e C nos casos simulados com 0%, 2.5%, 5%, 8.3% e 12.5% de possibilidade de ocorrência de vento, isso pode ser observado nas Figuras 4.4a, 4.4b, 4.4c, 4.4d e 4.4e, porém quando foi simulado com a possibilidade de 25% de ocorrência de vento, as estratégias foram equivalentes apenas para o ponto A, e a média de fotos da estratégia A foi melhor que a média da estratégia B nos pontos B e C, isso pode ser observado na Figura 4.4f.

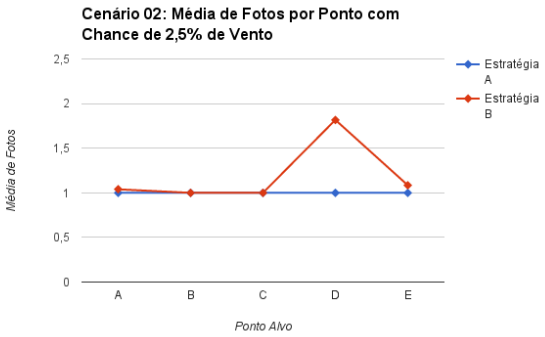
Para o ponto D, a média de fotos da estratégia B foi melhor que a média da estratégia A em todos os casos de ocorrência de vento simulados, conforme pode ser observado nas Figuras 4.4a, 4.4b, 4.4c, 4.4d, 4.4e e 4.4f. Já para o ponto E, as duas estratégias foram equivalentes para os casos com 0% e 2,5% de possibilidade de ocorrência de vento, isso é mostrado nas Figuras 4.4a, 4.4b, depois disso, considerando as possibilidades de 5%, 8.3%,

Figura 4.4: Média de fotos por ponto do Cenário 02.

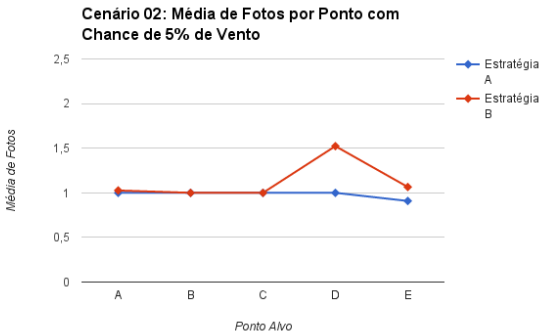
(a) 0% de chance de ocorrência de vento.



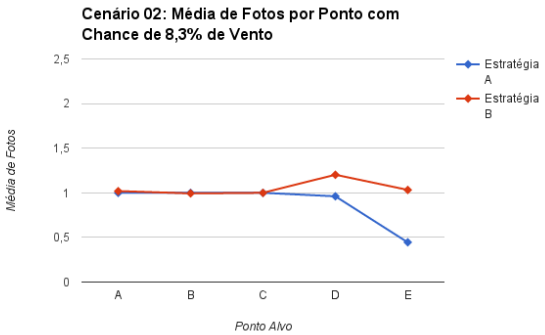
(b) 2.5% de chance de ocorrência de vento.



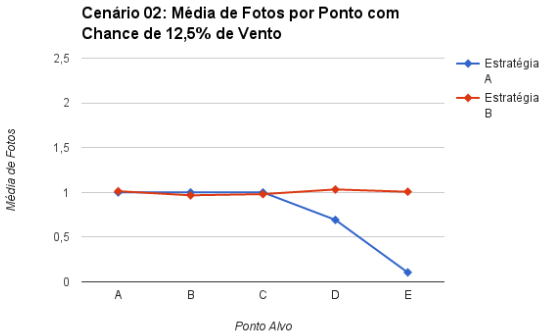
(c) 5% de chance de ocorrência de vento.



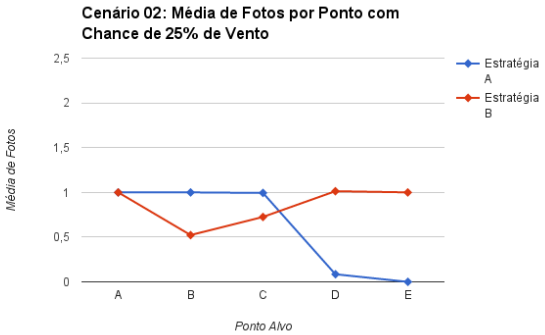
(d) 8.3% de chance de ocorrência de vento.



(e) 12.5% de chance de ocorrência de vento.



(f) 25% de chance de ocorrência de vento.



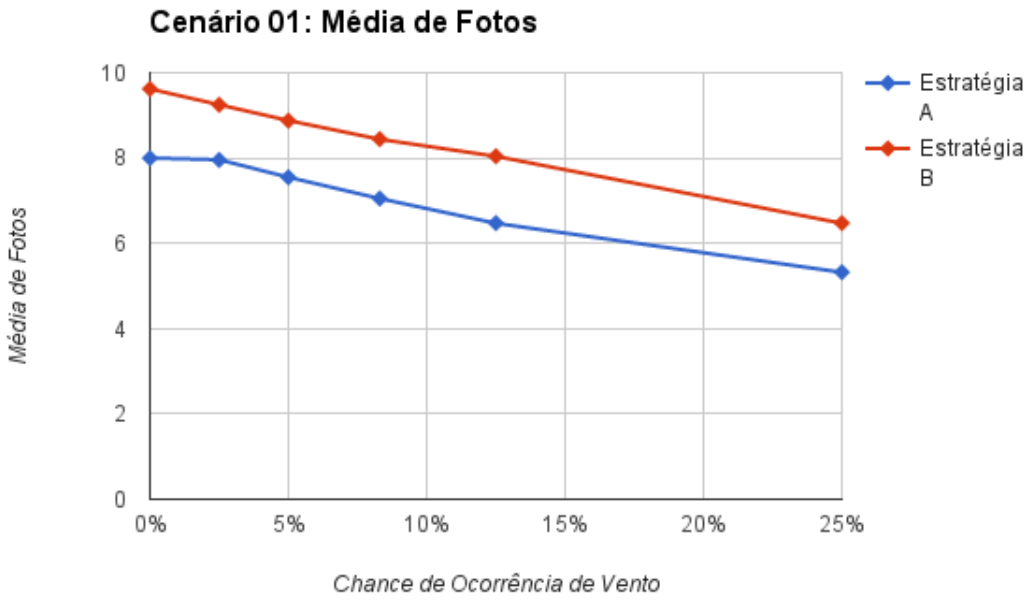
12.5% e 25%, a estratégia B conseguiu um desempenho melhor que o desempenho registrado pela estratégia A, conforme pode ser visualizado nas Figuras 4.4c, 4.4d, 4.4e e 4.4f.

Além disso, é importante destacar que na simulação utilizando a estratégia A e com 25% de possibilidade de ocorrência de vento o ponto E ficou com a média de 0 (zero) fotos capturadas e, portanto, sem observação. Desse modo, a influência dos ventos fez com que a média de fotos de alguns pontos diminuísse nos dois cenários analisados, em alguns casos

sendo uma estratégia melhor para determinados pontos que a outra, mas em geral a média de fotos de alguns pontos nas duas estratégias tenderam a diminuir ou de fato diminuíram nas simulações ao aumentar a influência dos ventos.

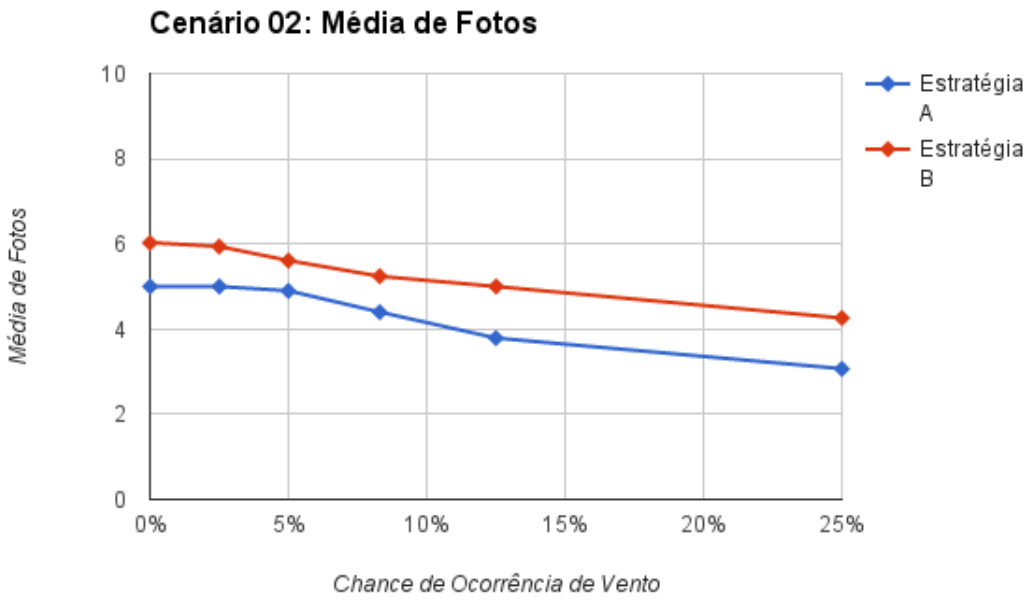
Esse comportamento pode ser observado nas Figuras 4.5 e 4.6 onde aparecem as médias de fotos acumuladas em cada situação de ocorrência de vento tanto para o Cenário 01 quanto para o Cenário 02. A estratégia B conseguiu um desempenho melhor que a estratégia A ao possuir médias de fotos superiores nos dois cenários analisados. Pode-se observar ainda que a média de fotos do Cenário 01 foi maior que a média do Cenário 02, isso se deve ao fato de que o Cenário 02 possui um comprimento de trajeto maior em cerca de uma vez e meia o comprimento de trajeto do Cenário 01.

Figura 4.5: Média de fotos capturadas no Cenário 01.



A carga da bateria no momento do pouso do VANT para o Cenário 01 pode ser observada na Figura 4.7, onde o eixo X representa o percentual da carga da bateria no momento do pouso e o eixo Y representa o percentual do número de pousos, assim é possível perceber, para cada estratégia de voo e para cada situação de ocorrência de vento, com qual percentual de carga de bateria aconteceram os pousos. Na situação com a possibilidade de 0% de ocorrência de vento a estratégia A foi melhor que a estratégia B porque os pousos aconteceram com a estratégia A possuindo um percentual de carga de bateria maior que o percentual de

Figura 4.6: Média de fotos capturadas no Cenário 02.



carga de bateria da estratégia B, isso pode ser observado na Figura 4.7a.

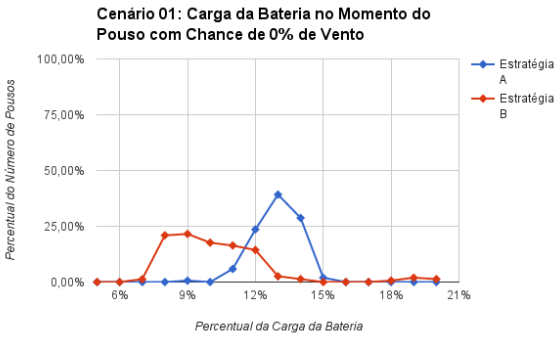
Quando a situação de possibilidade de ocorrência de vento passa a ser de 2,5% então o percentual de carga de bateria da estratégia A diminui e o percentual da estratégia B tem uma melhora passando a se aproximar da estratégia A nessa situação, conforme pode ser observado na Figura 4.7b. A partir daí o percentual de carga de bateria no momento do pouso passa a ter um comportamento semelhante para as duas estratégias, isso pode ser visualizado nas Figuras 4.7c, 4.7d, 4.7e e 4.7f.

Do mesmo modo, no Cenário 02 a carga da bateria no momento do pouso do VANT pode ser observada na Figura 4.8, onde o eixo X representa o percentual da carga da bateria no momento do pouso e o eixo Y representa o percentual do número de pousos, assim é possível perceber, para cada estratégia de voo e para cada situação de ocorrência de vento, com qual percentual de carga de bateria aconteceram os pousos. Nas situações com a possibilidade de 0%, 2.5% e 12.5% de ocorrência de vento as estratégias possuíram comportamento semelhante uma da outra em cada situação, isso pode ser observado nas Figuras 4.8a, 4.8b e 4.8e.

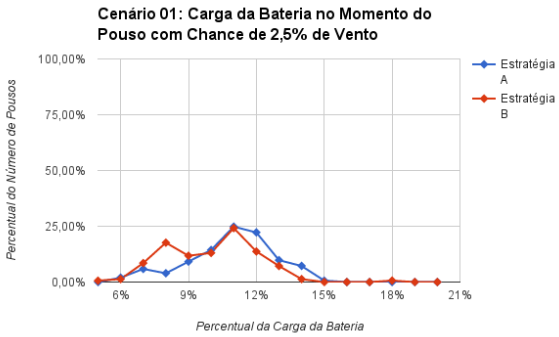
Nas situações com 5% e 8.3% de possibilidade de ocorrência de vento a estratégia B possuiu um desempenho melhor que a estratégia A porque o percentual de carga de bateria

Figura 4.7: Carga da bateria no momento do pouso no Cenário 01.

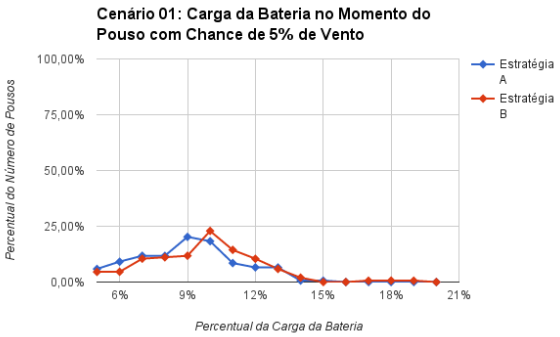
(a) 0% de chance de ocorrência de vento.



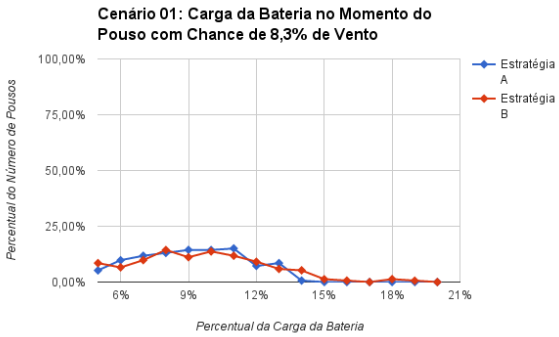
(b) 2.5% de chance de ocorrência de vento.



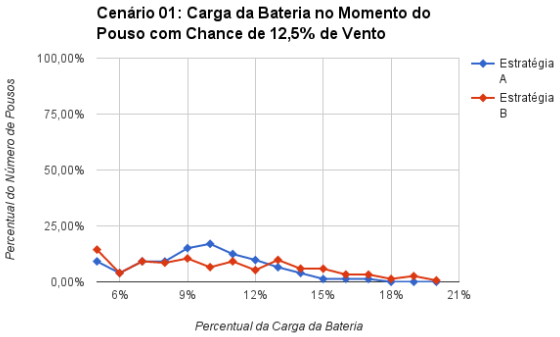
(c) 5% de chance de ocorrência de vento.



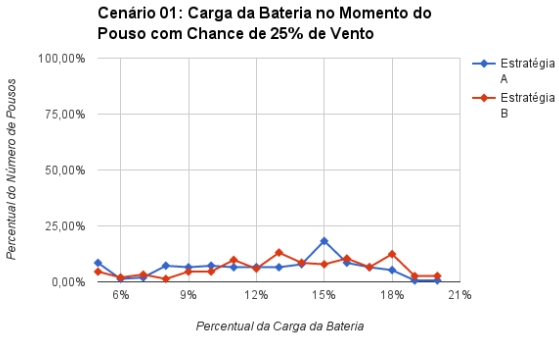
(d) 8.3% de chance de ocorrência de vento.



(e) 12.5% de chance de ocorrência de vento.



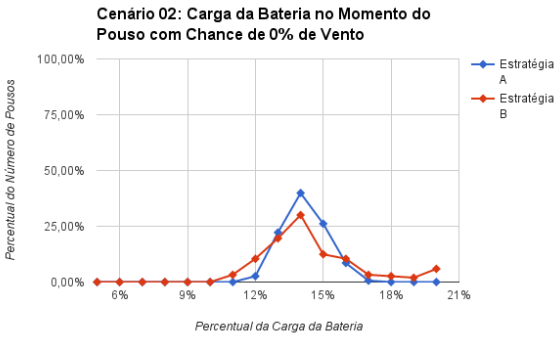
(f) 25% de chance de ocorrência de vento.



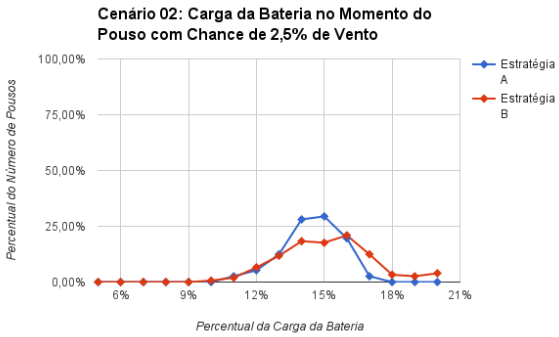
no momento do pouso foi maior para a estratégia B quando comparado com a estratégia A, esse desempenho pode ser observado nas Figuras 4.8c e 4.8d. Considerando a situação com 25% de possibilidade de ocorrência de vento a Estratégia A foi melhor que a estratégia B porque no caso da estratégia B mais de 75% dos pousos aconteceram com a energia mínima, enquanto que com a estratégia A foram cerca de 50%, isso pode ser visualizado na Figura 4.8f.

Figura 4.8: Carga da bateria no momento do pouso no Cenário 02.

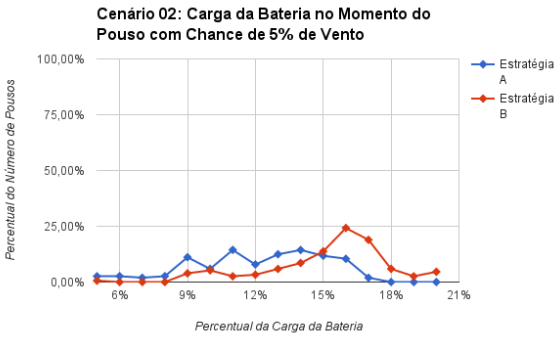
(a) 0% de chance de ocorrência de vento.



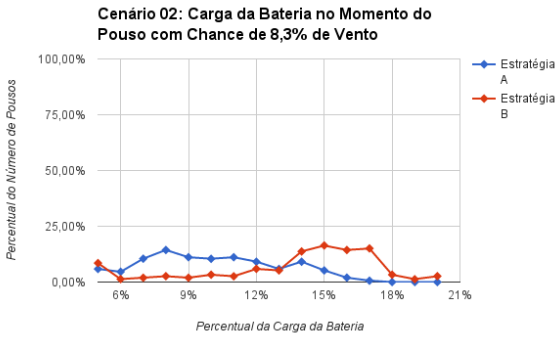
(b) 2.5% de chance de ocorrência de vento.



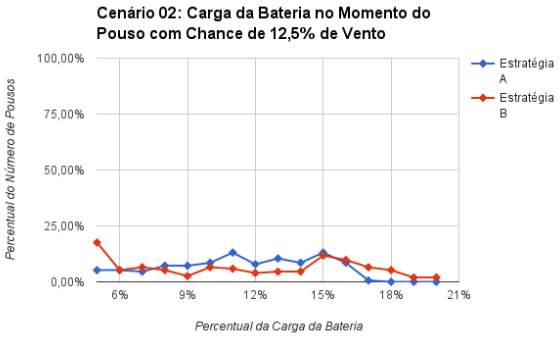
(c) 5% de chance de ocorrência de vento.



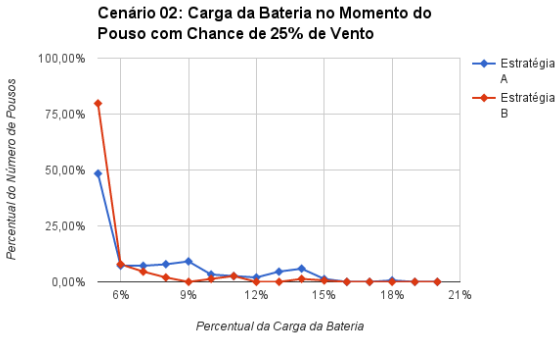
(d) 8.3% de chance de ocorrência de vento.



(e) 12.5% de chance de ocorrência de vento.



(f) 25% de chance de ocorrência de vento.



A capacidade para retornar à estação base é um valor percentual calculado através da divisão do número de pousos que foram realizados na estação base pelo número total de simulações realizadas, isso resulta no percentual de voos que terminaram com sucesso. Os demais pousos aconteceram fora da estação base quando o VANT ao tentar alcançá-la, e antes que isso acontecesse de fato, o nível da carga da bateria atingiu o valor de 5% que é o valor mínimo considerado. Nesses casos os voos terminaram com falha, quando o pouso foi

Figura 4.9: Capacidade para retornar à estação base no Cenário 01.

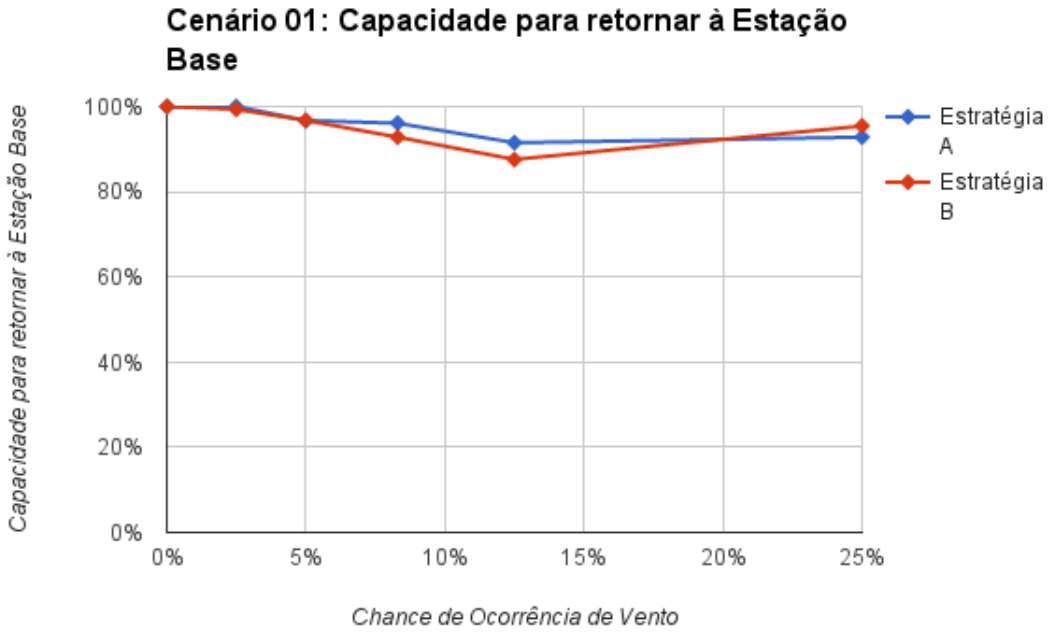
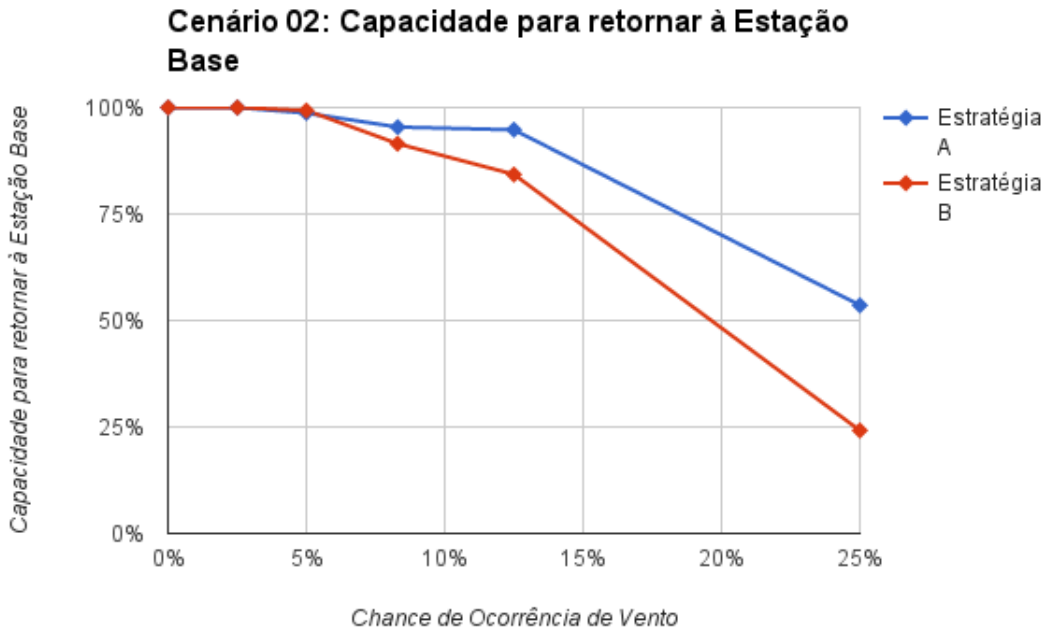


Figura 4.10: Capacidade para retornar à estação base no Cenário 02.



feito imediatamente encerrando a simulação sem ter alcançado o local de pouso pretendido.

A Figura 4.9 exibe a capacidade para retornar à estação base das estratégias A e B no Cenário 01, onde é possível perceber que nas simulações configuradas com 0%, 2.5% e

5% de possibilidade de ocorrência de vento as estratégias A e B possuíram desempenhos equivalentes, já para as situações com 8.3% e 12.5% de possibilidade de ocorrência de vento a estratégia A foi melhor que a B, porém nas simulações com 25% de possibilidade de ocorrência de vento a estratégia B foi melhor que a estratégia A.

A capacidade para retornar à estação base das estratégias A e B no Cenário 02 é apresentada na Figura 4.10, onde é possível perceber que nas simulações configuradas com 0%, 2.5% e 5% de possibilidade de ocorrência de vento as duas estratégias possuem desempenhos equivalentes, porém nas situações com 8.3%, 12.5% e 25% de possibilidade de ocorrência de vento a estratégia A foi melhor que a estratégia B, e é importante destacar que nesta última situação a estratégia A possuiu 53.59% de capacidade para voltar à estação base enquanto que a estratégia B possuiu apenas 24.18%.

A distância média percorrida é um valor calculado através da divisão da soma das distâncias percorridas pelo VANT, em cada simulação individual, pelo número total de simulações realizadas. O resultado das distâncias percorridas para o Cenário 01 pode ser visualizado na Figura 4.11, onde é apresentada a distância média percorrida em metros e onde é possível observar também que na situação com 0% de possibilidade de ocorrência de vento as distâncias médias percorridas das duas estratégias foram equivalentes, porém considerando os casos com 2.5%, 8.3%, 12.5% e 25% de possibilidade de ocorrência de vento as simulações que fizeram uso da estratégia A possuíram uma distância média percorrida levemente superior a distância média percorrida nas simulações que fizeram uso da estratégia B.

Para o Cenário 02, o resultado das distâncias médias percorridas em metros pode ser visualizado na Figura 4.12, onde é possível observar que nas situações com 0%, 2.5%, 5% e 8.3% de possibilidade de ocorrência de vento as simulações que fizeram uso da estratégia A percorreram uma distância média maior que a distância média percorrida nas simulações que fizeram uso da estratégia B, porém nos casos com 12.5% e 25% as distâncias médias percorridas nas duas estratégias foram equivalentes.

Para complementar as informações apresentadas, um resumo com a análise estatística dos resultados pode ser encontrado na Tabela 4.5 onde é possível visualizar informações para as várias situações de ocorrência de vento, simuladas tanto no Cenário 01 quanto no Cenário 02, e adicionalmente permite a comparação do desempenho das estratégias em cada situação de ocorrência de vento e em cada cenário analisado. Os dados apresentados mostram a

Figura 4.11: Distância média percorrida (em metros) no Cenário 01.

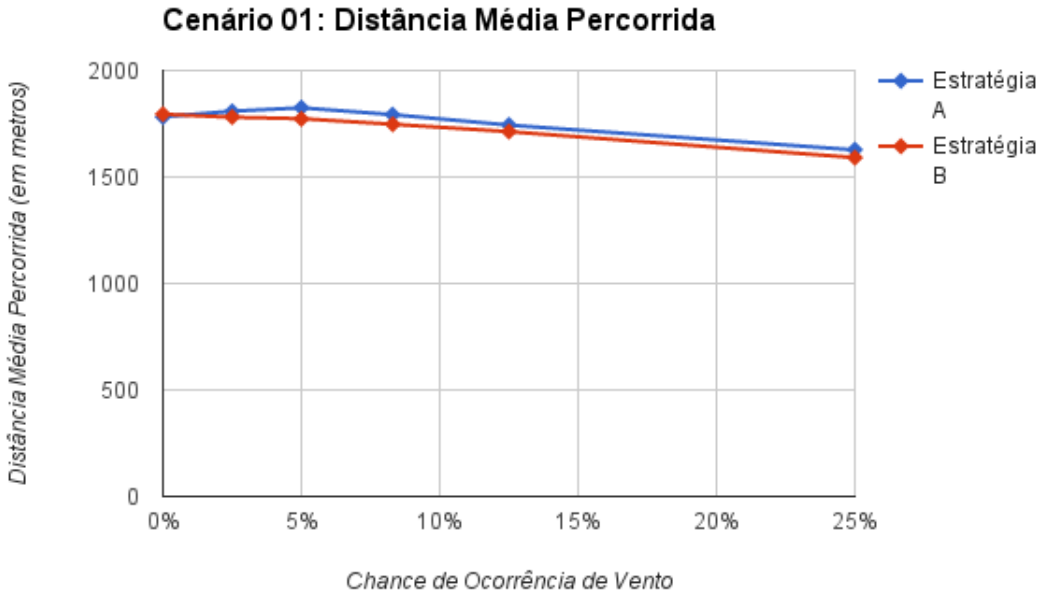
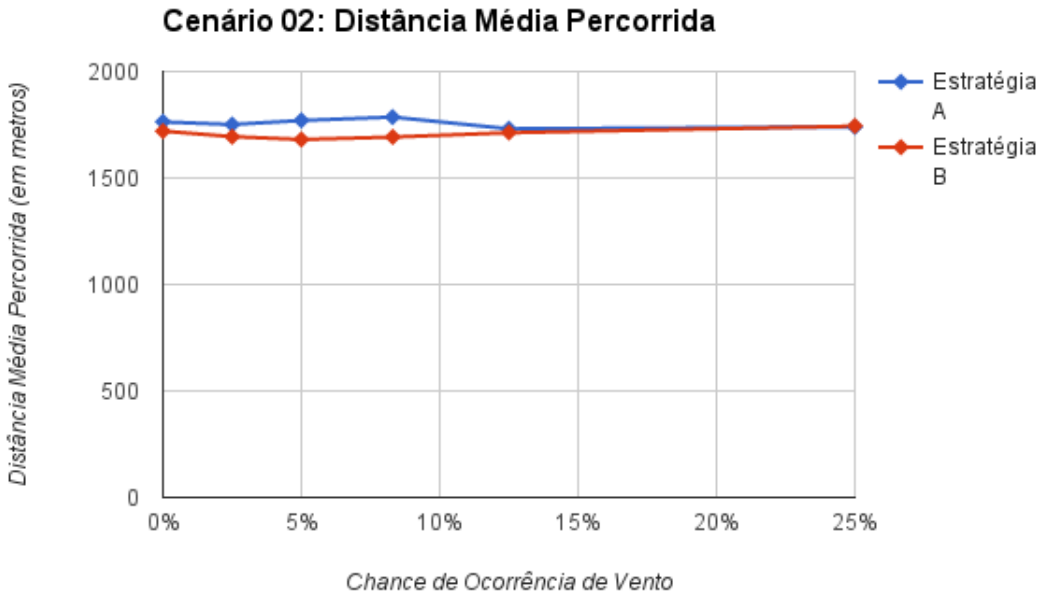


Figura 4.12: Distância média percorrida (em metros) no Cenário 02.



capacidade para retornar à estação base, a energia mínima observada, a quantidade média de fotos capturadas e a distância média percorrida nas simulações.

A lista [*eb,e,pts, d*] encontrada na Tabela 4.5 significa que o VANT é capaz de voltar

para a estação base em um percentual *eb*, onde 100 é sempre e 0.0 nunca, com um mínimo de energia *e*, visitando uma média de *pts* pontos, e percorrendo uma distância média de *d* metros. Lembrando que o Cenário 01 possui 5 pontos a serem visitados e o Cenário 02 também possui 5 pontos a serem visitados. Além disso, a energia aqui apresentada é o nível da carga da bateria considerado, esse nível pode assumir o valor máximo de 100 quando a bateria está com a carga completa e o mínimo de 5.0 quando a bateria está com a carga quase vazia e o VANT é forçado a pousar.

Tabela 4.5: Análise estatística dos resultados.

Cenário 01		
Chance de Vento	**Estratégia A**	**Estratégia B**
0%	[100, 9.0, 8.0, 1782.72]	[100, 7.0, 9.62, 1794.88]
2.5%	[100, 6.0, 7.96, 1809.17]	[99.34, 5.0, 9.25, 1780.78]
5%	[96.73, 5.0, 7.55, 1825.03]	[96.73, 5.0, 8.88, 1773.96]
8.3%	[96.07, 5.0, 7.05, 1792.96]	[92.81, 5.0, 8.44, 1747.75]
12.5%	[91.50, 5.0, 6.47, 1743.81]	[87.58, 5.0, 8.04, 1713.09]
25%	[92.81, 5.0, 5.32, 1627.29]	[95.42, 5.0, 6.47, 1591.18]

Cenário 02		
Chance de Vento	**Estratégia A**	**Estratégia B**
0%	[100, 12.0, 5.0, 1763.45]	[100, 11.0, 6.03, 1720.26]
2.5%	[100, 11.0, 5.0, 1751,27]	[100, 10.0, 5.94, 1694.37]
5%	[98.69, 5.0, 4.90, 1770.14]	[99.34, 5.0, 5.61, 1680.89]
8.3%	[95.42, 5.0, 4.40, 1786.15]	[91.50, 5.0, 5.24, 1692.29]
12.5%	[94.77, 5.0, 3.79, 1731.09]	[84.31, 5.0, 5.0, 1712.77]
25%	[53.59, 5.0, 3.07, 1740.15]	[24.18, 5.0, 4.26, 1743.46]

Na Tabela 4.5 é possível observar que nas simulações realizadas a estratégia A conseguiu uma distância média percorrida maior que a distância média percorrida pela estratégia B na maioria dos casos, apenas no Cenário 01 com 0% de ocorrência de vento e no Cenário 02 com 25% de ocorrência de vento foi registrado para a estratégia A uma média menor que

a média registrada para a estratégia B. Quando se trata da média de fotos capturadas nas simulações a estratégia B foi melhor que a estratégia A em todos as situações de ocorrência de vento tanto no Cenário 01 quanto no Cenário 02, mesmo tendo percorrido uma distância menor que a estratégia A, o que significa que a estratégia B conseguiu ser melhor nesse critério de avaliação.

Em relação ao consumo da bateria, a carga restante da bateria no momento do pouso da estratégia A foi melhor nos casos com 0% e 2.5% de possibilidade de ocorrência de vento tanto no Cenário 01 quanto no Cenário 02, para os demais casos a carga restante da bateria no momento do pouso foi igual para as duas estratégias.

Quanto à capacidade para retornar à estação base nas situações analisadas é possível perceber que no Cenário 01 nas situações com 0% e 2.5% de possibilidade de ocorrência de vento a estratégia A conseguiu voltar em 100% dos casos. No mesmo Cenário a estratégia B conseguiu voltar em 100% dos casos na situação com 0% de possibilidade de ocorrência de vento, porém quando a possibilidade de ocorrência de vento passa para 2.5% a estratégia B diminui um pouco o desempenho e consegue voltar em 99.34% dos casos. Na situação com 5% de ocorrência de vento as duas estratégias conseguem a mesma capacidade para retornar à estação base. Nos casos com 8.3% e 12.5% a estratégia A teve um melhor desempenho que a estratégia B nesse critério, porém com 25% a estratégia B conseguiu um desempenho melhor.

Para o Cenário 02, a mesma capacidade de retornar à estação base é observada para os casos com 0% e 2.5% de possibilidade de ocorrência de vento, onde tanto a estratégia A quanto a B tiveram a capacidade de retornar à estação base em 100% dos casos. Com 5% de possibilidade de ocorrência de vento a estratégia B foi melhor que a estratégia A, para os demais casos, 8.3%, 12.5% e 25%, a estratégia A conseguiu um desempenho melhor que a estratégia B, tendo alcançado neste último caso uma diferença significativa.

Após a realização dos experimentos, pode-se perceber que o ambiente aqui proposto permite a simulação de voos de VANTs com a utilização, em cada voo, de uma estratégia específica, isso permite ao projetista fazer a análise de várias estratégias e escolher aquela em que os dados apontem ser a mais adequada para a sua necessidade em questão.

Capítulo 5

Conclusão

5.1 Considerações Finais

Este trabalho apresentou uma proposta de um ambiente de simulação para auxiliar no projeto de Sistemas Ciber-Físicos, especificamente para avaliar e validar estratégias de voos para serem utilizadas em VANTs. O simulador desenvolvido é composto por dois módulos, um módulo é responsável pela configuração das estratégias de voo e foi desenvolvido no Ptolemy, o outro é responsável representação do ambiente de voo e para isso o SITL/ArduPilot foi utilizado. A comunicação entre os módulos é feita através da arquitetura de alto nível - HLA.

No simulador desenvolvido foi possível realizar simulações de modelos com a representação da incerteza, onde foi possível simular voos sem interferências externas e voos com a presença de interferências de eventos externos. Os eventos externos foram representados por várias possibilidade de ocorrência de vento que quando aconteceram interferiram no voo desviando o VANT para uma posição não determinada. As possibilidades de ocorrência de vento simuladas foram 0%, 2.5%, 5%, 8.3%, 12.5% e 25%.

Para cada possibilidade de ocorrência de vento foram realizados 153 simulações de voos no Cenário 01 com o uso da Estratégia A e, do mesmo modo, 153 simulações com o uso da Estratégia B. Após a finalização das simulações no Cenário 01, foi dado início as simulações no Cenário 02, onde foram utilizadas as mesmas possibilidades de ocorrência de vento, a mesma quantidade de simulações e o uso das mesmas estratégias. As informações do voo sobre o nível da bateria no momento do pouso, a distância percorrida, a capacidade para

retornar a estação base e a quantidade de fotos capturadas foram armazenadas para viabilizar a comparação das estratégias utilizadas nos voos. O nível de confiança calculado para o tamanho da amostra (153 voos simulados) foi de 95% e a margem de erro considerada foi de 8%.

A contribuição oferecida por este trabalho foi a construção de um ambiente através da integração de simuladores heterogêneos para viabilizar a análise de estratégias de voos de VANTs. Com a utilização da abordagem defendida neste trabalho é possível obter resultados mais próximos da realidade, dessa forma, estratégias mais eficientes poderão ser desenvolvidas e testadas. Esta abordagem segue a ideia de que a simulação não deve considerar apenas o cenário ideal de voo quando não acontecem interferências, mas deve se aproximar do processo real a ser simulado e considerar situações que acontecem no ambiente real.

Como trabalho futuro, novas estratégias de voo com outros algoritmos devem ser implementadas no módulo de configuração de estratégias, para compor uma biblioteca pronta para ser usada em futuras simulações. Espera-se que algumas dessas novas estratégias levem em consideração a realização de voos com múltiplos VANTs trabalhando juntos. Além disso, desenvolver um novo federado para um veículo real e substituir o federado do ambiente de voo pelo novo federado no ambiente de simulação, após isso, realizar voos e fazer a comparação dos resultados obtidos na simulação com os novos resultados obtidos nos voos com o uso do veículo real.

Bibliografia

[Accellera 2015]ACCELLERA. *The Language for System-Level Modeling, Design and Verification*. [S.l.], October 2015. Disponível em: <http://accellera.org/community/systemc/about-systemc>.

[Brito et al. 2015]BRITO, A. V. et al. A distributed simulation platform using hla for complex embedded systems design. In: *2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. [S.l.: s.n.], 2015. p. 195–202. ISSN 1550-6525.

[CERTI - Summary 2016]CERTI - Summary. 2016. Disponível em: <http://savannah.nongnu.org/projects/certi>.

[Cheung, Hao e Xie 2007]CHEUNG, P. H.; HAO, K.; XIE, F. Component-based hardware/-software co-simulation. In: *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*. [S.l.: s.n.], 2007. p. 265–270.

[Dade 2017]DADE, S. *MAVProxy*. 2017. Disponível em: <http://ardupilot.github.io/MAVProxy/html/index.html>.

[David e Ballado 2016]DAVID, L. C. G.; BALLADO, A. H. Vegetation indices and textures in object-based weed detection from uav imagery. In: *2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*. [S.l.: s.n.], 2016. p. 273–278.

[Fitzgerald, Pierce e Larsen 2014]FITZGERALD, J.; PIERCE, K.; LARSEN, P. Co-modelling and co-simulation in the engineering of systems of cyber-physical systems. In: *System of Systems Engineering (SOSE), 2014 9th International Conference on*. [S.l.: s.n.], 2014. p. 67–72.

[Forin, Neekzad e Lynch 2006]FORIN, A.; NEEKZAD, B.; LYNCH, N. L. *Giano: The Two-Headed System Simulator*. [S.l.], September 2006. 16 p. Disponível em: <http://research.microsoft.com/apps/pubs/default.aspx?id=70343>.

[Hewitt 2010]HEWITT, C. Actor model of computation: scalable robust information systems. *arXiv preprint arXiv:1008.1459*, 2010.

[Hsu, Wen e Wang 2007]HSU, Y.-T.; WEN, Y.-J.; WANG, S.-D. Embedded hardware/-software design and cosimulation using user mode linux and systemc. In: *Parallel Processing Workshops, 2007. ICPPW 2007. International Conference on*. [S.l.: s.n.], 2007. p. 17–17. ISSN 1530-2016.

[Interface 2010]INTERFACE, I. 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.

[Kanduri et al. 2013]KANDURI, A. et al. A multicore approach to model-based analysis and design of cyber-physical systems. In: *SoC Design Conference (ISOCC), 2013 International*. [S.l.: s.n.], 2013. p. 278–281.

[LASNIER 2013]LASNIER, G. *Toward a Distributed and Deterministic Framework to Design Cyber-PhysicalSystems*. [S.l.]: ISAE-Supaero, 2013.

[Lasnier et al. 2013]LASNIER, G. et al. Distributed simulation of heterogeneous and real-time systems. In: *2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications*. [S.l.: s.n.], 2013. p. 55–62. ISSN 1550-6525.

[Leira, Johansen e Fossen 2015]LEIRA, F. S.; JOHANSEN, T. A.; FOSSEN, T. I. Automatic detection, classification and tracking of objects in the ocean surface from uavs using a thermal camera. In: *2015 IEEE Aerospace Conference*. [S.l.: s.n.], 2015. p. 1–10. ISSN 1095-323X.

[MAVLink 2017]MAVLINK. *MAVLink Micro Air Vehicle Communication Protocol*. 2017. Disponível em: <http://www.mavlink.org/mavlink/start>.

[Moore 1998]MOORE, G. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, v. 86, n. 1, p. 82–85, Jan 1998. ISSN 0018-9219.

[OMT 2010]OMT, I. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Object Model Template (OMT) Specification. *IEEE Std 1516.2-2010 (Revision of IEEE Std 1516.2-2000)*, p. 1–110, Aug 2010.

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

[PyHLA — Python Bindings for M&S HLA 2016]PYHLA — Python Bindings for M&S HLA. 2016. Disponível em: <http://www.nongnu.org/certi/PyHLA>.

[Ren et al. 2014]REN, C. et al. Control software development of drive motor for electric vehicles. In: *Transportation Electrification Asia-Pacific (ITEC Asia-Pacific), 2014 IEEE Conference and Expo*. [S.l.: s.n.], 2014. p. 1–6.

[Robotics 2016]ROBOTICS, D. *About DroneKit*. 2016. Disponível em: <http://python.dronekit.io/about/overview.html>.

[Rules 2010]RULES, I. Ieee standard for modeling and simulation (m amp;s) high level architecture (hla)– framework and rules. *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, p. 1–38, Aug 2010.

[SANTOS 2017]SANTOS, G. E. de O. *Cálculo amostral: calculadora on-line*. 2017. Disponível em: <http://www.calculoamostral.vai.la>.

[Sicklinger et al. 2014]SICKLINGER, S. et al. Interface jacobian-based co-simulation. *International Journal for Numerical Methods in Engineering*, v. 98, n. 6, p. 418–444, 2014. ISSN 1097-0207. Disponível em: <http://dx.doi.org/10.1002/nme.4637>.

[Sineglazov e Godny 2015]SINEGLAZOV, V. M.; GODNY, A. P. Model of computer-aided design environment for uav projects. In: *2015 IEEE International Conference Actual Problems of Unmanned Aerial Vehicles Developments (APUAVD)*. [S.l.: s.n.], 2015. p. 41–44.

[SITL Simulator (Software in the Loop) 2016]SITL Simulator (Software in the Loop). 2016. Disponível em: <http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html>.

[Team 2016]TEAM, A. D. *Choosing a Ground Station*. 2016. Disponível em: <http://ardupilot.org/copter/docs/common-choosing-a-ground-station.html>.

[Xiang et al. 2016]XIANG, T. et al. Uav based target tracking and recognition. In: *2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. [S.l.: s.n.], 2016. p. 400–405.

[Yapp, Seker e Babiceanu 2016]YAPP, J.; SEKER, R.; BABICEANU, R. Uav as a service: Enabling on-demand access and on-the-fly re-tasking of multi-tenant uavs using cloud services. In: *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. [S.l.: s.n.], 2016. p. 1–8.