

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

Um ambiente para teste e diagnóstico de drones usando
cossimulacao

Renato Ricardo de Abreu

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

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

Alisson Vasconcelos de Brito
(Orientador)

João Pessoa, Paraíba, Brasil

©Renato Ricardo de Abreu, 16deJaneirode2019

Catálogo na publicação
Seção de Catalogação e Classificação

A162a Abreu, Renato Ricardo de.

Um ambiente para teste e diagnóstico de drone usando
Cossimulacao / Renato Ricardo de Abreu. - João Pessoa,
2019.

88 f. : il.

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

1. Vant, Drone, teste, Ptolemy, HLA, HIL. I. Brito,
Alisson Vasconcelos de. II. Título.

UFPB/BC

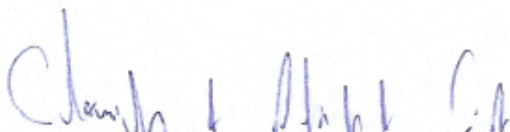


UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA



Ata da Sessão Pública de Defesa de Dissertação de Mestrado de Renato Ricardo de Abreu, candidato ao título de Mestre em Informática na Área de Sistemas de Computação, realizada em 16 de janeiro de 2019.

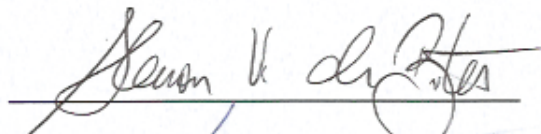
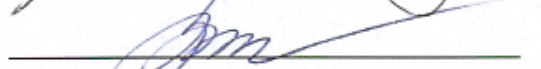
1 Aos dezesseis dias do mês de janeiro, do ano de dois mil e dezenove, às dez horas, no
2 Centro de Informática da Universidade Federal da Paraíba, em Mangabeira, reuniram-se os
3 membros da Banca Examinadora constituída para julgar o Trabalho Final do Sr. Renato
4 Ricardo de Abreu, vinculado a esta Universidade sob a matrícula nº 20161020695, candidato
5 ao grau de Mestre em Informática, na área de "Sistemas de Computação", na linha de
6 pesquisa "Sinais, Sistemas Digitais e Gráficos", do Programa de Pós-Graduação em
7 Informática, da Universidade Federal da Paraíba. A comissão examinadora foi composta
8 pelos professores: Alisson Vasconcelos de Brito (PPGI-UFPB) Orientador e Presidente da
9 Banca, Lucídio dos Anjos Formiga Cabral (PPGI-UFPB), Examinador Interno, Ramon Leonn
10 Victor Medeiros (IFPB), Examinador Externo à Instituição. Dando início aos trabalhos, o
11 Presidente da Banca, cumprimentou os presentes, comunicou aos mesmos a finalidade da
12 reunião e passou a palavra ao candidato para que o mesmo fizesse a exposição oral do
13 trabalho de dissertação intitulado: "Um ambiente para teste e diagnóstico de drone usando
14 Cossimulacao". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que
15 emitiu o seguinte parecer: "**aprovado**". Do ocorrido, eu, Claurton de Albuquerque Siebra,
16 Coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai
17 assinada por mim e pelos membros da banca examinadora. João Pessoa, 16 de janeiro de
18 2019.


Prof. Dr. Claurton de Albuquerque Siebra

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

Prof. Dr. Lucídio dos Anjos Formiga Cabral
Examinador Interno (PPGI-UFPB)

Prof. Dr. Ramon Leonn Victor Medeiros
Examinador Externo ao Programa (IFPB)


Resumo

Veículos aéreos não tripulados (VANT), também conhecidos como drones, são muito importantes para realizar voos sem a necessidade de um piloto no veículo, por isso são programados para executar missões de voos. Entretanto, eles necessitam ser confiáveis para executá-las. Então, com a realização de diagnósticos do estado do VANT, é possível prever falhas durante ou antes da execução do voo. O objetivo deste trabalho é apresentar um ambiente de teste, para analisar e avaliar drones durante o voo em ambiente fechado. Para este propósito, o framework Ptolemy II foi estendido para comunicação com drones reais usando o High Level Architecture (HLA). O ambiente de teste apresentado é extensível para outras rotinas, e pronto pra integração com outras ferramentas de simulação e análise. Para testar o ambiente foram realizados dois experimentos de detecção de falhas, com um total de 20 voos realizados para cada um deles. Destes 20 voos, 80% foram utilizados para treinar um algoritmo de árvore de decisão, e os outros 20% voos para testar o algoritmo em que uma das hélices possuía anomalia. A taxa de acerto ou detecção de falha foi de 70% para o primeiro experimento e de 90% para o segundo experimento.

Palavras-chaves: Veículo aéreo não tripulado, Teste, Simulação Hardware-in-the-loop, High Level Architecture

Abstract

The unmanned aerial vehicles (UAVs), also known as drones, are very important to execute flights with no necessary pilot in the vehicle, thus it is programmed to run flight missions. However, they require reliability to execute missions, then with diagnostic it is possible to predict vehicle failure during or before the flight. The objective of this work is to present a testing tool, which analyzes and evaluates drones during the flight in indoor environments. For this purpose, the framework Ptolemy II was extended for communication with real drones using the High Level Architecture (HLA). The presented testing environment is extendable for other testing routines, and is ready for integration with other simulation and analysis tools. For testing, a total of 40 flights were performed. From that, 20 were used to train a Decision Tree algorithm, and the other 20 to test the algorithm, where some of them had an anomaly added to one of the propellers. The accurate rate of fault detection was 70%.

Key-words: Unmanned Aerial Vehicle, Testing, Hardware-in-the-loop Simulation, High Level Architecture

Agradecimentos

Olhe ! Como não sou mal agradecido,
nem tão pouco abestado,
vou logo agradecendo ao Professor Doutor Alisson Brito,
que é um orientador arretado.

Agradeço também aos meus pais e as minhas filhas
agradeço aos meus irmãos,
Agradeço a toda a família,
que nunca me deixaram na mão.

Não posso esquecer uma pessoa,
que tem nome de princesa e cara de menina,
tenho que agradecer a todo o imenso apoio,
da minha namorada Ana Leopoldina.

Agradeço aos amigos do Laser:
Halamo, Joanacele, os Thiagos, Daniel...
e tem muito mais,
pra citar todo mundo faltaria papel

Por fim, agradeço a um cabra bom,
que quando precisei me deu uma luz,
agradeço o apoio na correção do trabalho,
que será feita pelo meu amigo Professor Antonio Jesus.

Sumário

1	Introdução	1
1.1	Objetivos	3
1.1.1	Objetivo Geral	3
1.1.2	Objetivos Específicos	3
1.2	Metodologia	3
1.3	Publicações Relacionadas	4
1.4	Estrutura da Dissertação	7
2	Fundamentação Teórica	8
2.1	Ptolemy	8
2.2	High Level Architecture	11
2.3	Ptolemy-HLA	17
2.4	Ar.Drone 2.0	19
2.5	PS-Drone	20
2.6	Py-HLA	21
3	Um ambiente para teste de drones	22
3.1	Visão Geral do ambiente	22
3.2	Módulo de criação da Sequência de teste (Federado do Ptolemy)	25
3.3	Módulo de controle do drone (Federado do drone)	28
3.4	Módulo de detecção de falhas (Federado do Ptolemy)	37
4	Avaliação Experimental	41
4.1	Descrição do experimento	41
4.2	Configuração do Ambiente	43

<i>SUMÁRIO</i>	vii
4.3 Resultados	45
5 Conclusão	52
5.1 Considerações Finais	52
Referências Bibliográficas	58
A Apêndice A	59
B Apêndice B	71
C Apêndice C	80

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

Arrastar e soltar : Arrastar e soltar (em Inglês: Drag-and-drop) é a ação de clicar em um objeto virtual e "arrastá-lo" a uma posição diferente ou sobre um outro objeto virtual.

API : Termo em inglês "Application Programming Interface" que significa "Interface de Programação de Aplicativos".

PWM : "Pulse Width Modulation" ou Modulação de Largura de Pulso, ou seja, através da largura do pulso de uma onda quadrada é possível o controle de potência ou velocidade.

Lista de Figuras

2.1	Representação visual de um modelo de atores.	9
2.2	Representação visual de um modelo com atores simples e compostos. . . .	9
2.3	Ciclo de vida de um ator no Ptolemy.	10
2.4	Arquitetura geral de uma federação HLA.	13
2.5	Diagrama de sequência de uma simulação HLA	16
2.6	Ícone do ator HLAManager.	18
2.7	Ícone do ator HLAPublish.	18
2.8	Ícone do ator HLASubscriber.	18
2.9	Modelo Centralizado Ptolemy.	19
2.10	Modelo descentralizado Ptolemy (Federado 1).	19
2.11	Modelo descentralizado Ptolemy (Federado 2).	19
3.1	Visão Geral do Ambiente	23
3.2	exemplo de sequência de teste criada no módulo de controle	25
3.3	Biblioteca de atores movimentação do Drone Ptolemy II	26
3.4	Janela de configuração dos parâmetros do ator Forward	27
3.5	Algoritmo executado no Federado do Drone pelo programa DroneFederate.py	28
3.6	Detalhe do ator composto drone.	38
3.7	Valores do PWM para cada motor, capturado durante o voo.	39
4.1	Ar.Drone com os índices dos motores	42
4.2	Valores do Yaw durante o voo	43
4.3	valores do PWM do motor 3 durante o voo	45
4.4	Gráfico nível de bateria durante o voo	48
4.5	Gráfico nível de bateria durante o voo	49

4.6	Gráfico valores YAW durante o voo	50
4.7	Gráfico valores PWM para os 4 motores durante o voo	51

Lista de Tabelas

2.1	Serviços da interface HLA chamados pela RTI nos federados.	14
2.2	Serviços da interface HLA chamados pelos Federados na RTI.	15
3.1	Atores desenvolvidos para testar o drone	24
3.2	Variáveis Globais do Federado do Drone.	33
4.1	Comparação das técnicas de aprendizagem da máquina para detecção de anomalias	43
4.2	Parâmetros do Drone	44
4.3	Configuração da máquina física.	45
4.4	Configuração da Raspberry.	46
4.5	Softwares utilizados.	46
4.6	Experimento 1 - Taxa de acerto do ator Fault Detection (Yaw)	47
4.7	Experimento 2 - Taxa de acerto do ator Fault Detection (PWMMotor3) . . .	48

Lista de Códigos Fonte

3.1	Exemplo cadeia caracteres com a missão drone	27
3.2	Função droneControlWrapper	29
3.3	Função ARDRONE_INIT.	30
3.4	Função ARDRONE_FLY	31
3.5	Função updateHLAAttributesWrapper.	32
3.6	Função ARDRONE_HLAUPDATE (Sincronizando as Threads).	33
3.7	Função ARDRONE_HLAUPDATE (Selecionado dados de telemetria).	34
3.8	Função ARDRONE_HLAUPDATE (Sinalização de início de publicação dos dados de telemetria).	35
3.9	Função ARDRONE_HLAUPDATE (Publicação dos dados de telemetria).	35
3.10	Função ARDRONE_HLAUPDATE (Sincronizando as Threads).	36
3.11	Exemplo arquivo de dados do voo (Voo_010320181212.csv).	39
3.12	Exemplo arquivo de Log do voo (Voo_010320181212.log).	40
3.13	Class Drone defined to be used by HLA for message exchanging	40
A.1	Modulo de Controle do Drone	59
B.1	Ator Move forward	71
C.1	Ator Drone Telemetry	80

Capítulo 1

Introdução

Veículos aéreos não tripulados (VANTs), popularmente conhecidos como drones, vêm adquirindo mais importância nos últimos anos. Drones estão sendo utilizados em vigilância e missões de resgates [Michini et al. 2011]. Eles podem ser pilotados remotamente ou executar missões pré-programadas, e como não são tripulados, seu tamanho e peso são reduzidos em comparação com aeronaves convencionais. Hoje em dia, nos grandes centros, não é incomum observar a presença de drones civis, não registrados, voando em lugares públicos, onde geralmente existem grandes aglomerações de pessoas. Drones também vêm sendo utilizados em operações policiais e missões de resgates, ou apenas como uma forma de entretenimento. Por exemplo, no Brasil, operações policiais utilizam drones para combate ao tráfico de drogas nas favelas do Rio de Janeiro, como também para mapeamento de eventos e manifestações populares, e até para perseguição de suspeitos [R7 2016] [UOL 2016] [sindepol 2016] [Globo 2016].

Todo drone é suscetível a falhas e acidentes. Estas falhas e acidentes são geralmente causados pelo mau funcionamento do drone, o que não é fácil de prever [Patelli e Mottola 2016; American 2015]. Esta imprevisibilidade traz riscos aos seres humanos e pode causar sérios danos materiais e ambientais.

Uma alternativa é verificar e controlar o comportamento do drone em um ambiente fechado e controlado. Isso pode aumentar a segurança antes de testes em espaços abertos. Este trabalho propõe um ambiente para a elaboração e execução de testes usando drones, com o objetivo de diagnosticar falhas e comportamento anormais. O ambiente é composto por módulos de hardware e software e permite a configuração de vários tipos de testes e análises

do comportamento do drone usando uma abordagem arrastar e soltar (drag-and-drop) no seu modulo de elaboração configuração de testes.

Para isso, o ambiente de simulação Ptolemy II foi integrado ao drone usando o padrão de integração de simuladores, IEEE 1516 (HLA) [IEEE 2000]. Isto permite não apenas a integração do Ptolemy com o drone, mas abre a possibilidade da integração com outros simuladores, como simulador de rede [Brito e Oliveira 2016], ROS [Junior et al. 2016], Sistemas embarcados [VS, Brito e Nascimento 2015], SystemC [Silva et al. 2018], ou até mesmo um grupo de diversos simuladores [Brito et al. 2015]. A integração desses dois componentes permitiu a criação, execução e coleta de dados a partir de uma definição da movimentação do drone.

A movimentação executada pelo drone é definida pela concatenação de atores. Atores específicos que representam os movimentos de um drone, foram implementados e integrados ao ambiente para realizar diagnósticos através da análise de dados de telemetria coletados do Drone durante os voos.

As análises podem ser utilizadas para diagnosticar falhas e para avaliar o controle aerodinâmico do drone. Neste trabalho foi analisado o comportamento físico de um drone real em um ambiente controlado. Para isso, dois experimentos foram configurados: no primeiro experimento, o drone decola, permanece estável e logo depois pousa. No segundo experimento, o drone decola, move-se para frente, depois para trás e então pousa. Nos dois experimentos, em 50% dos voos, foi inserido um peso de 0.02 gramas em uma das hélices, acrescentando uma instabilidade, também conhecido como falha de ecentricidade dinâmica. O objetivo dessa inserção foi verificar se é possível diferenciar os voos livres, considerados estáveis, dos voos com a anomalia inserida, considerados instáveis. Então, uma abordagem de diagnóstico foi implementada para detectar o problema de equilíbrio da hélice. Nos experimentos, o algoritmo foi bem-sucedido em 70% dos casos para o primeiro experimento e 90% dos casos para o segundo experimento.

Diversos grupos de pesquisadores ao redor do mundo buscam novas estratégias para testar ou avaliar drones. Nesse contexto, vários trabalhos podem ser citados: [Loh, Bian e Roe 2009], [Stojcsics e Molnar 2011], [YanJun, Yang e Shenglin 2013], [Jaw et al. 2008], [Wang et al. 2015], [Ahsan, Rafique e Ahmed 2013] and [Yoo, Kang e Park 2008]. Contudo, a integração do drone com o Ptolemy via HLA ainda não está presente na literatura. Isto

resulta em duas principais contribuições deste trabalho: : 1) o desenvolvimento de uma ferramenta de teste de drones utilizando o Ptolemy; e 2) a integração de drone via HLA, que permite o desenvolvimento de outras ferramentas no futuro.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo deste trabalho é desenvolver um ambiente com módulos de hardware e software onde seja possível analisar o comportamento de drones reais, durante o voo, a fim de diagnosticar falhas, como também analisar sua aerodinâmica.

1.1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Desenvolver um ambiente em que seja possível analisar e diagnosticar drones durante o voo.
- Desenvolver uma ferramenta que permita a integração futura com outros simuladores e vários tipos VANTs reais ou simulados;
- Validar o funcionamento da ferramenta através de um estudo de caso, composto por dois experimentos, em que o comportamento do drone é analisado, e um diagnóstico é apresentado.

1.2 Metodologia

Para alcançar os objetivos propostos 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;
- Estudo do funcionamento e da arquitetura do drone tipo AR.Drone da empresa Parrot;
- Integração através do HLA do módulo de configuração dos testes desenvolvidos no Ptolemy com o módulo controle do Drone Real, desenvolvido em Python, utilizando a API PS-Drone;
- Definição de um estudo de caso composto por dois experimentos para diagnóstico e análise do comportamento do drone real;
- Para cada experimento, realização de várias simulações de voos;
- Análise dos resultados providos pela simulação do estudo de caso.

1.3 Publicações Relacionadas

Diversos grupos de pesquisadores ao redor do mundo buscam novas estratégias para testar ou avaliar drones. Nesse contexto, vários trabalhos podem ser citados: [Loh, Bian e Roe 2009], [Stojcsics e Molnar 2011], [YanJun, Yang e Shenglin 2013], [Jaw et al. 2008], [Wang et al. 2015], [Ahsan, Rafique e Ahmed 2013] e [Yoo, Kang e Park 2008]. Na mesma linha de pesquisa, é possível encontrar trabalhos, como o de [Ducard, Kulling e Geering 2007], que estuda as influências nas falhas dos componentes físicos dos drones e avalia a mudança de desempenho após a ocorrência de alguma avaria. Como estudo de caso, os autores analisaram o comportamento do veículo após a ocorrência de falha no Aileron, que é o componente responsável por controlar a rotação do veículo no plano sobre o seu eixo longitudinal. Para isso, esse trabalho desenvolveu um sistema que detecta e isola a falha, e decide se o voo pode continuar ou abortar a missão. O trabalho de [Ducard, Kulling e Geering 2007], assim como a presente pesquisa, analisa o comportamento dos drones durante o voo, sendo que os autores estão focados na análise do comportamento do veículo após a ocorrência de falhas no Aileron ¹ e verificam o desempenho de voo, que é degradado em decorrência dessa falha, pode ser mensurado e usado para reconfigurar o sistema de orientação do veículo, enquanto

¹Os Ailerons são as partes móveis das bordas das asas das aeronaves de asa fixa e servem para controlar o movimento de roll da aeronave

o presente trabalho focaliza a implementação de um ambiente para testes analisando o comportamento de um drone físico em um ambiente simulado para análise e monitoramento de voo.

Os trabalhos de [Park, Lee e Yu 2015] e [Lee et al. 2013] também buscam monitorar e avaliar o desempenho de voo de drones. Nos dois trabalhos são estudados veículos alimentados por energia solar buscando realizar voos de longa duração. No primeiro são realizados testes monitorando o veículo a partir de uma estação em solo (*Ground Station*) usando simulação para replicar os movimentos do drone e através disso avalia o desempenho das principais partes do veículo. Como se trata de um veículo movido a energia solar, também é avaliada a eficiência energética desde a geração, o armazenamento e o fluxo de energia para otimizar o consumo em voos de longa duração. Nesse trabalho desenvolve-se um sistema de voo virtual para avaliação de drones movidos a energia solar. Esse sistema fornece dados de voo para avaliar a possibilidade de realização de voos contínuos. No estudo, o sistema é dividido em partes: Controle e Monitoramento, Gerenciamento de Energia e Fuselagem; e a parte de Propulsão. Como forma de comunicação é utilizado Zigbee [2016 2016] para monitorar os dados medidos no drone por meio de comunicação sem fio. Essa troca de dados acontece sem fio e em tempo real com o veículo, permitindo que a potência energética necessária para o voo seja calculada com base nos dados da trajetória de voo. Enquanto a altitude simulada é gerada pelo movimento real do drone. O mesmo acontece com as condições meteorológicas que são dados de condições reais.

Esses dois trabalhos diferem do proposto quanto ao objetivo, pois os dois fazem análise e avaliação do desempenho de drones em voo, enquanto este está focado no desenvolvimento de um ambiente, que permite avaliar os parâmetros de voo para desempenho.

Boa parte dos trabalhos que monitoram o desempenho dos drones em voo buscam alcançar resultados que ajudem a melhorar a segurança de voo, como o trabalho de [Deng e Yuan 2015], que propõe uma avaliação para melhorar essa segurança através de método estatístico ou por meio de testes simulados em solo usando HIL (Hardware-in-the-loop).

Neste trabalho é proposto um método de simulação em que se busca testar as respostas de um drone diante de estímulos de entrada que simulam condições adversas extremas, e, a partir disso avaliar o comportamento do veículo tendo em vista as reações aos estímulos recebidos. Assim como a presente pesquisa, o trabalho de [Deng e Yuan 2015] realiza ava-

liação do desempenho de drones, porém voltado ao comportamento do veículo diante de estímulos extremos, enquanto o presente trabalho está centrado na construção do ambiente de testes em que avaliações de segurança são realizadas.

O trabalho de [Vázquez et al. 2012] traz uma abordagem diferente das propostas dos trabalhos citados nos parágrafos anteriores. Neste busca-se construir uma estrutura para drones do tipo quadróptero, capaz de se comunicar com uma estação de solo, e, através de comunicação sem fio, possibilitar o monitoramento do posicionamento do veículo durante o voo. Nesse contexto, sensores como sonar e acelerômetro de três eixos são acoplados ao veículo e os dados coletados desses sensores são enviados para a estação de solo, para que mostrem em uma interface visual a orientação e o posicionamento do veículo durante o voo. Por outro lado, a estação de solo usa a mesma comunicação sem fio para enviar comandos ao drone.

Esse se assemelha ao nosso trabalho por objetivarem a construção de estruturas para monitorar o estado de drones. Porém, o trabalho de [Vázquez et al. 2012] objetiva apresentar os dados em uma interface visual, enquanto este trabalho objetiva usar essa estrutura para coletar os dados e sobre esses dados realizar análises utilizando HIL para fins diversos.

O trabalho de [Ducard, Kulling e Geering 2007] estuda a influência em falhas dos componentes físicos dos drones e avalia uma mudança de desempenho após a ocorrência de algum dano.

Diversos grupos de pesquisa estão buscando soluções para o monitoramento de drones em voo e avaliação de componentes, ou do comportamento do veículo para diversos fins: segurança do veículo, eficiência energética ou controle de rota. Todos os tipos de avaliações de drones necessitam de dados que expressem o estado desses veículos. Para tal, é desenvolvida uma solução por meio da qual vários tipos de análises, avaliações e diagnósticos de drones podem ser realizados. Nessa solução, um conjunto de ferramentas como o Ptolemy II fornece maneiras de monitorar os dados de um drone, bem como controlar seu comportamento, sendo essa solução um ambiente para avaliação de drone para diversas finalidades.

1.4 Estrutura da Dissertação

A organização deste trabalho de dissertação está estruturalmente dividida em cinco capítulos. O Capítulo 1 trata do contexto geral em que este trabalho está inserido e também os trabalhos relacionados. O Capítulo 2 apresenta os conteúdos necessários para a compreensão dos conceitos utilizados na proposta apresentada. O Capítulo 3, por sua vez, discorre sobre o desenvolvimento do ambiente proposto. O Capítulo 4 expõe 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 Ptolemy

Neste trabalho foi utilizado o Ptolemy para a implementação do módulo de configuração, análise e diagnóstico dos voos. Embora este módulo pudesse ter sido implementado em outra linguagem, a experiência que a equipe já tinha com a ferramenta foi fator decisivo para a sua escolha.

A ferramenta Ptolemy II é uma estrutura baseada nos princípios de design orientada por atores, cujo objetivo é projetar e simular sistemas heterogêneos complexos. Os Atores são criados na linguagem Java, implementam funcionalidades e se comunicam mediante portas e relações e de maneira concorrente. Os atores do Ptolemy II podem ser descritos como atômicos ou compostos. Atores compostos são aqueles que contêm outros atores internos, mas por fora suas funcionalidades se assemelham aos atores atômicos. Por exemplo, o ator display é utilizado para representar dados numéricos em gráficos, e sua representação é definida por um ator atômico. Os dados que chegam nas portas de entrada são processados a fim de criar novos dados e, então, enviá-los para outros atores por suas portas de saída. Ptolemy II gerencia a execução e a interação entre os atores através de duas entidades chamadas diretor e receptor, que juntos compõem o chamado domínio [Brito et al. 2013] [Cardoso 2018].

O Ptolemy baseia-se no modelo computacional de atores, onde cada componente de um projeto implementado nesta ferramenta é chamado de ator. Em um modelo simples, os atores são gerenciados por um diretor; em modelos mais complexos, atores compostos podem conter submodelos. Neste caso cada submodelo pode conter um diretor próprio, ou ser ge-

reenciado pelo diretor do modelo de nível superior. As Figuras 2.1 e 2.2 mostram respectivamente: a) representação visual de um modelo de ator com a presença de apenas um diretor; b) representação visual de um modelo de ator com a presença de mais de um diretor.

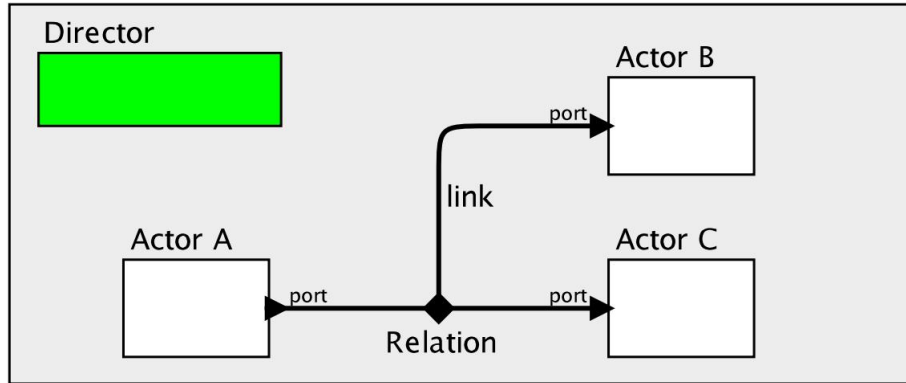


Figura 2.1: Representação visual de um modelo de atores.

Fonte: [Ptolemaeus 2014]

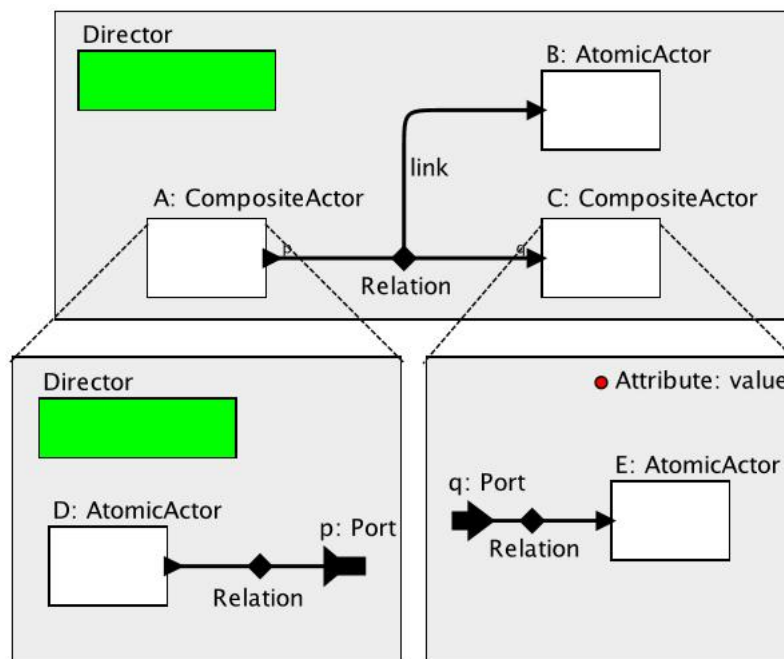


Figura 2.2: Representação visual de um modelo com atores simples e compostos.

Fonte: [Ptolemaeus 2014]

A criação de um modelo no Ptolemy se dá através da combinação de atores simples e compostos, seguindo uma organização hierárquica. Através de atores compostos, o modelo poderá ter subníveis hierárquicos, uma vez que cada ator composto implementa seu

próprio modelo computacional. Cada subnível apresenta um comportamento homogêneo localmente, porém, como diretores diferentes podem representar modelos diferentes, significa que o modelo hierárquico normalmente é heterogêneo.

No modelo, os atores se comunicam através de portas de entrada, saída ou de entrada e saída ao mesmo tempo. As portas podem ser simples, quando existem apenas um canal de comunicação, ou múltiplas quando existem mais de um canal de comunicação. Para um ator se comunicar com outro, dever haver uma ligação entre eles. Por exemplo, na figura 2.1, as setas mostram a comunicação entre os atores A,B e C. Nessa figura, o ator A envia dados para os atores B e C, simultaneamente.

Para [Barros 2017], [Lasnier et al. 2013] e [Ptolemaeus 2014], o ciclo de vida de um ator no Ptolemy compreende a execução de três fases, a saber: *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 os procedimentos de liberação de recursos alocados e o encerramento da federação. Na Figura 2.3, são apresentadas as fases com as suas respectivas ações.

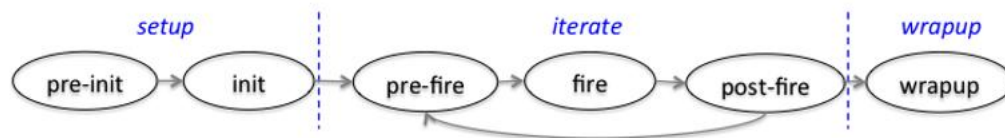


Figura 2.3: Ciclo de vida de um ator no Ptolemy.

Fonte: [Lasnier et al. 2013]

A fase *setup* possui duas ações: a)preinitialize (pre-init), que é executada apenas uma vez, ela realiza tarefas como: programação do ambiente, geração de código, verificação de tipos, criação de atores em atores compostos, etc. b)initialize (init), que pode ser executada mais de uma vez e realiza tarefas como inicializar parâmetros, iniciar o estado local do ator e enviar mensagens iniciais.

A fase *iterate* é a principal e possui três ações que podem ser executadas várias vezes. A ação prefire (pre-fire) é responsável por validar se existem informações suficientes para a execução da ação fire, por exemplo, verificar se há dados para serem lidos nas portas de entrada; a ação fire, é responsável pela leitura nas portas de entrada, processamento das informações e envio das informações para as portas de saída; a ação postfire (post-fire) é

responsável por atualizar o estado persistente do ator.

A fase wrapup executa a ação com o mesmo nome da fase: wrapup. Nesta ação acontece a finalização de uma simulação. Esta ação deve ser utilizada para liberação dos recursos que foram alocados durante a inicialização - fechar arquivos, fechar conexões com banco de dados, fechar conexões com a rede, etc.

É possível adicionar novos atores ao Ptolemy. Para a criação de um novo ator, é necessário implementar uma nova classe na linguagem Java que herde os métodos de alguma classe já existente no Ptolemy. Por exemplo, pode-se utilizar a classe TypedAtomicActor para criação de um novo ator. Após isso, é necessário adicionar as portas de entrada e as de saída e sobrescrever os métodos que fazem parte do ciclo de vida de um ator no Ptolemy [Barros 2017].

2.2 High Level Architecture

Neste trabalho, a HLA foi utilizada com objetivo de interligar o módulo de controle do drone real com o módulo de configuração e análise dos voos implementados no Ptolemy. Para este trabalho, a HLA fornece uma arquitetura aberta de troca de informações, a possibilidade de inclusão de novos componentes, o gerenciamento do tempo e sincronização entre os participantes. Embora outras arquiteturas forneçam as mesmas funcionalidades, o fator da escolha da HLA, se deu por causa da experiência que a equipe já tinha com a ferramenta.

O HLA é uma arquitetura de propósito geral definido pelo Defence Modelling and Simulation Office (DMSO). Suas principais características são a reutilização e interoperabilidade. Essa arquitetura surgiu para intercomunicar diferentes simuladores, assim como separar as funcionalidades especificidades de cada simulador usando uma infraestrutura de propósito geral. Na arquitetura HLA, o conjunto de vários sistemas interoperando dentro de um mesmo domínio é chamado Federação. Cada integrante de uma Federação é chamado Federado. Cada simulador precisa usar a Run-time Infrastructure (RTI) para se comunicar com HLA e outros simuladores. Esta conexão ao mesmo RTI forma uma federação. O padrão HLA faz a comunicação e é definido no IEEE 1516-2000 [IEEE 2000] [Brito et al. 2013].

A especificação da arquitetura do HLA é formado por três documentos: a) o primeiro [RULES 2010] especifica o framework e as regras para interação entre os federados; b) o

segundo [INTERFACE 2010] especifica a interface de comunicação entre federados que permite a conexão e coordenação entre diferentes simuladores; c) o terceiro [OMT 2010] especifica a documentação padrão para definição do modelo dos dados que são trocados entre os federados [Barros 2017] [Brito et al. 2013].

O Simulated Object Model (SOM) descreve e trata dos tipos de informações que os federados podem receber e enviar para as federações. O Federation Object Model (FOM) especifica as classes, atributos de objetos de classes, conteúdo MOM, entre outros. Por último, o Management Object Model (MOM) especifica os construtores que proporcionam o suporte para monitorar e controlar a execução das federações [OMT 2010] [Junior 2015].

O Federation Object Model (FOM) especifica o compartilhamento de informações em tempo de execução. Nestas informações estão incluídas: classes, atributos de objetos de classes, conteúdo MOM, entre outros. Por fim, o Management Object Model (MOM) especifica os construtores que proporcionam o suporte para monitorar e controlar a execução das federações.

A Run-Time Infrastructure (RTI) é um componente essencial do HLA. Ela é a especificação de um software que fornece os serviços de interface para sincronização e troca de dados entre os federados em uma simulação [RULES 2010] [Barros 2017] [Brito et al. 2013].

Na figura 2.4, é apresentada a arquitetura geral de uma federação. Nela podem ser visualizados os componentes principais de uma federação, que são: a) os federados e b) a RTI. Uma federação é composta pela combinação de um FOM, um conjunto de federados e os serviços da interface do HLA [INTERFACE 2010] [Barros 2017] [Brito et al. 2013]. Define-se por federado qualquer software que implemente os serviços da interface HLA e seja capaz de se unir a uma federação para enviar e receber informações de outros federados.

Os serviços chamados pelos federados como: iniciar uma nova federação, conectar-se a uma federação ativa e enviar e/ou receber informações, são especificados na Interface HLA e implementados na RTI. Por sua vez, os federados devem implementar os métodos da interface HLA para serem invocados pela RTI através de mensagens de chamadas de retorno (callbacks). Eles são invocados no federado em resposta a alguma solicitação realizada pelo próprio federado, ou para recebimento de informações de interesse do federado, ou ainda quando a RTI informa o federado sobre a situação da federação [Barros 2017] [INTERFACE 2010].

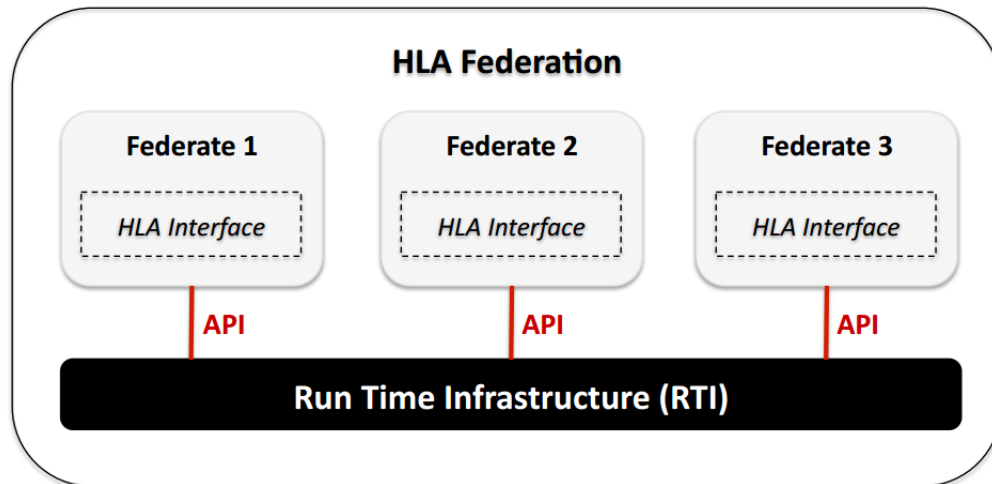


Figura 2.4: Arquitetura geral de uma federação HLA.

Fonte: [Lasnier et al. 2013]

Os serviços invocados pelo federado e implementados na RTI são listados na Tabela 2.1, já os serviços que invocados pela RTI e implementados no federado são listados na Tabela 2.2. Estes serviços serão explicados mais detalhadamente à frete quando for apresentado um exemplo de criação de uma federação envolvendo dois federados.

De acordo com [Barros 2017], na Figura 2.5, é apresentando um exemplo de criação de uma federação envolvendo dois federados: Federado A e o Federado B. Neste exemplo, é mostrada a sequência de chamados para a criação de uma federação envolvendo os Federados A e B; e a RTI.

No passo 1, o Federado A cria a federação através da invocação do método `createFederationExecution`. Esse procedimento é realizado pelo primeiro federado ao se conectar com a RTI. Nos passos 2 e 3, os dois federados se unem à federação invocando o método `JoinFederationExecution`. Então, nos passos de 4 a 7, os federados informam a classe e os atributos que desejam manipular. Isto é feito invocando os métodos `getObjectClassHandle` e `getAttributeHandle` respectivamente.

Nos passos de 8 a 19, através da invocação dos serviços `publishObjectClass` e `subscribeObjectClassAttributes`, cada federado publica na federação qual a classe para a qual irá publicar informações e qual a classe assinará para receber as publicações de outros federados.

Em seguida, no passo 20, através da invocação do método `registerFederationSynchroni-`

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

Serviço	Descrição
synchronizationPointRegistratioSucceeded	Indica que o registro do ponto de sincronização foi registrado 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.

Fonte: [Barros 2017]

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.
synchronizationPointAchieved	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	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.

Fonte: [Barros 2017]

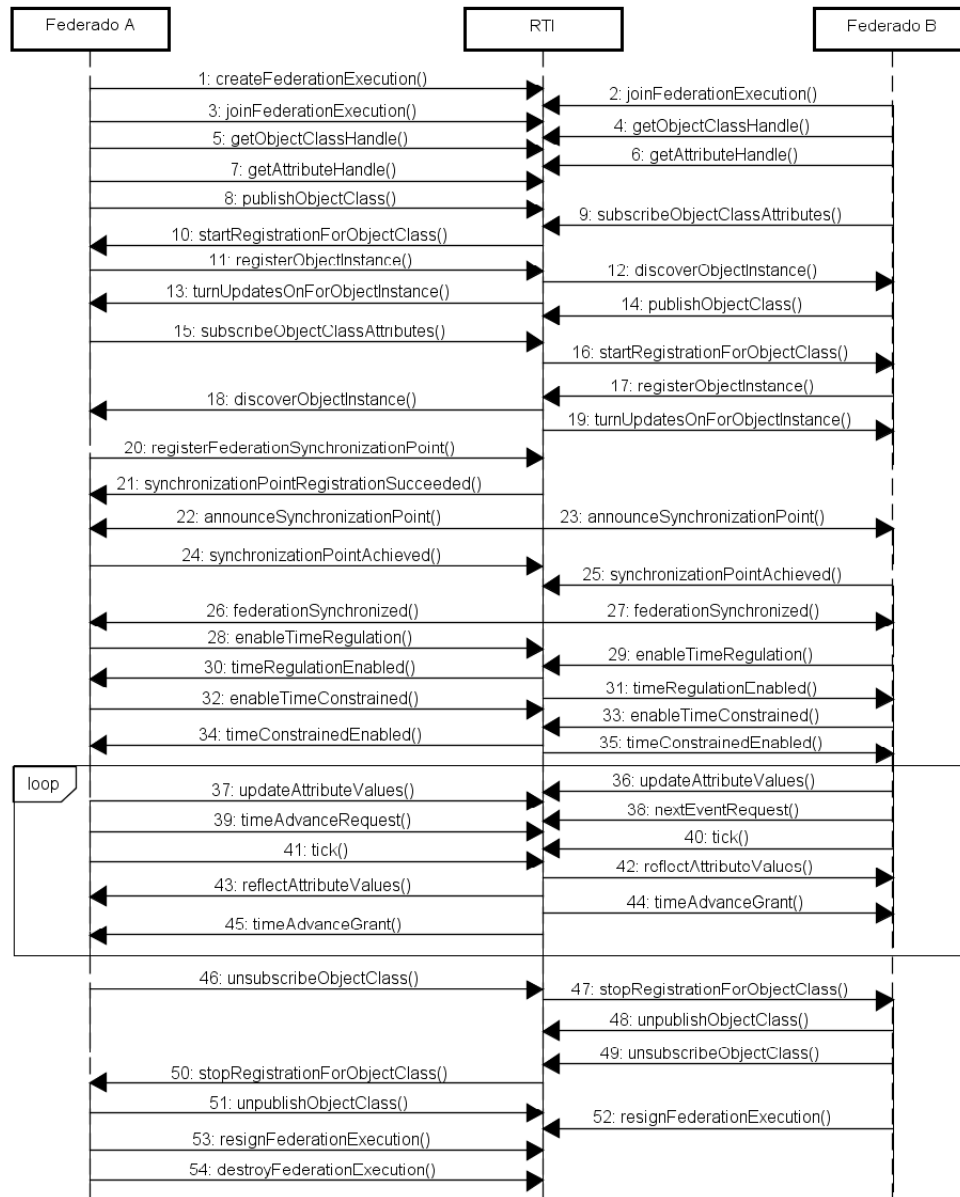


Figura 2.5: Diagrama de sequência de uma simulação HLA

Fonte: [Barros 2017]

zationPoint, o Federado A solicita o registro de um rótulo que será o ponto de sincronização, e no passo 21 a RTI invoca o método `synchronizationPointRegistrationSucceeded` no Federado A, confirmando que o registro foi realizado com sucesso.

Nos passos 22 e 23, a RTI, através da invocação do método `announceSynchronizationPoint` em todos federados, anuncia-se qual é o ponto de sincronização registrado. Então, nos passos 24 e 25, os federados informam que chegaram ao ponto de sincronização através da invocação do método `synchronizationPointAchieved` da RTI. Nos passos 26 e 27, a RTI

informa que todos os federados estão sincronizados, invocando o método `federationSynchronized` nos federados. As restrições de tempo são habilitadas em cada federado, nos passos de 28 a 35.

A troca de informações entre os federados acontece nos passos de 36 a 45 (fragmento loop). Aqui os Federados A e B publicam novos valores na RTI através do método `updateAttributeValues` e em seguida solicitam o avanço do tempo com a chamada do método `timeAdvanceRequest` ou `nextEventRequest`. A RTI invoca o método `reflectAttributeValues` nos federados, para atualizá-los das informações as quais eles têm interesse, e libera o avanço no tempo com a chamada ao serviço `timeAdvanceGrant`.

Nos passos de 46 a 54, os federados cancelam suas assinaturas para os objetos de uma determinada classe através da invocação ao método `unsubscribeObjectClass`. Logo após, os federados se desassociam da federação invocando o método `resignFederationExecution`. Quando a federação não possui mais nenhum federado, a RTI invoca o método `destroyFederationExecution` para destruir a federação.

2.3 Ptolemy-HLA

O framework de cossimulação Ptolemy-HLA [Cardoso 2018] utiliza duas ferramentas de código aberto: Ptolemy II e o HLA/CERTI. Ele permite distribuir a execução de um modelo do Ptolemy em uma federação utilizando o padrão HLA. Cada modelo do Ptolemy pode implementar o seu federado através de atores disponíveis nas bases de atores do Ptolemy. O Ptolemy e também o Ptolemy-HLA estão disponíveis para os sistemas operacionais Linux, Windows e MacOS.

Três novos atores foram adicionados ao Ptolemy: `HlaManager` (Figura: 2.6), `HlaPublisher` (Figura: 2.7) e o `HlaSubscriber` (Figura: 2.8). O primeiro, o `HlaManager`, implementa a coordenação de tempo entre o Ptolemy e o tempo lógico do HLA, 2) a troca de dados entre os federados (com o `HlaSubscriber` e o `HlaPublisher`). O `HlaPublisher`, implementa um editor em uma federação, ele relaciona um token do Ptolemy a um atributo de classe de um objeto descrito no FOM. Este ator é responsável pela publicação de informações na Federação. Por fim, o `HlaSubscriber` implementa um assinante em uma federação; ele relaciona um atributo de classe de um objeto FOM. Este ator é responsável pela leitura

de informações publicadas na Federação.

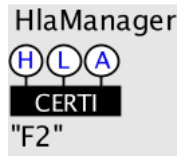


Figura 2.6: Ícone do ator HLAManager.

Fonte: [Cardoso 2018]

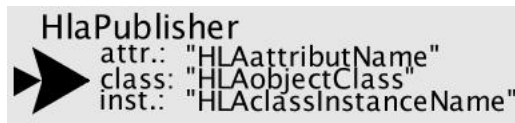


Figura 2.7: Ícone do ator HLAPublish.

Fonte: [Cardoso 2018]

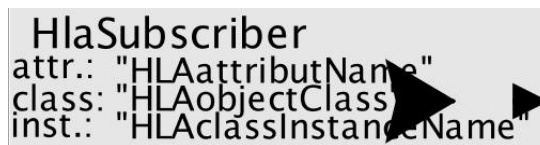


Figura 2.8: Ícone do ator HLASubscriber.

Fonte: [Cardoso 2018]

A Figura 2.9 apresenta um modelo centralizado do Ptolemy com dois federados A e B trocando informações localmente. As Figuras 2.10 e 2.11 apresentam o mesmo modelo implementado de forma descentralizada utilizando atores do HLA. No Federado 1 (Figura 2.10), são adicionados um ator HLAManager e um ator HLAPublish, tornando o federado em um editor que publicará informações na federação. Já no Federado 2, são adicionados um ator HLAManager e um ator HLASubscriber, tornando o federado em uma assinante para ler informações publicadas na federação. A troca de informações entre os federados ocorrerá através da RTI de forma descentralizada. Os atores do HLA, no framework, podem ser encontrados em: MoreLibrairies-> Co-Simulation-> HLA 4.

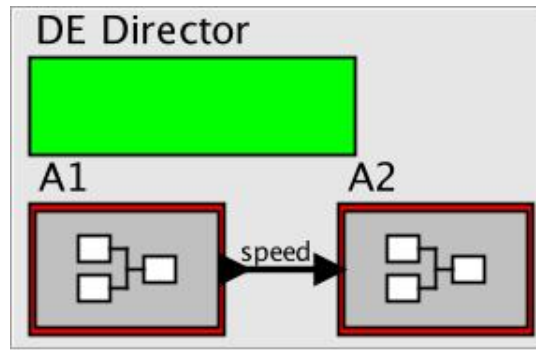


Figura 2.9: Modelo Centralizado Ptolemy.

Fonte: [Cardoso 2018]

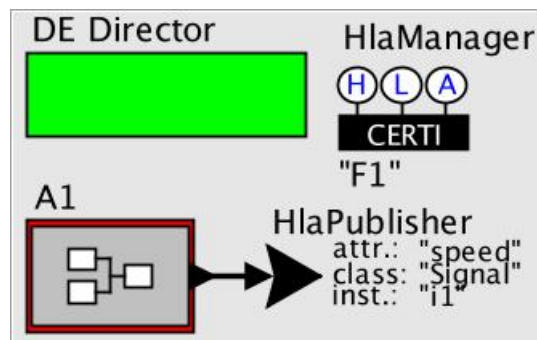


Figura 2.10: Modelo descentralizado Ptolemy (Federado 1).

Fonte: [Cardoso 2018]

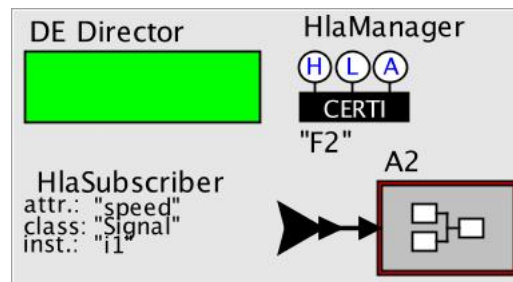


Figura 2.11: Modelo descentralizado Ptolemy (Federado 2).

Fonte: [Cardoso 2018]

2.4 Ar.Drone 2.0

Existe uma variedade de modelos de drones disponíveis no mercado. Para este trabalho escolhemos o Ar.Drone 2.0. A seleção se deu pela disponibilidade do aparelho e pela existência de uma API completa de controle do drone. Lembrando que o ambiente é aberto e um

módulo de controle pode ser implementando para qualquer tipo de drone.

O AR.Drone 2.0 é um quadricóptero desenvolvido e fabricado pela empresa francesa Parrot [Bristeau François Callou 2011]. Por fornecer a capacidade de ser controlado remotamente por computador, através de uma conexão sem fio, o AR.Drone pode ser utilizado em uma série de aplicações como jogos de realidade aumentada e sistemas de monitoramento de áreas de desastre [Saito e Mase 2013].

Até a publicação deste trabalho, existem duas versões do AR.Drone, a versão 1.0 e 2.0. Ambas vêm equipadas com sensores para medição e estabilização automática de pitch, roll e yaw como também um telémetro ultrassônico para aferição da altitude. A versão 2.0 inclui um magnetômetro de três eixos e um sensor de pressão para permitir a medição da altitude, independente da altura em que o drone se encontra [Hvizdoš e Sincák 2015].

O pitch é referente às variações no ângulo, θ , que são os ângulos formados entre os eixos longitudinal e horizontal do drone. O roll são as variações no ângulo φ , cujos os valores são gerados pelas inclinações laterais do drone, em relação à sua orientação frontal ou às variações no eixo longitudinal do mesmo. Por fim, o yaw diz respeito às variações no ângulo ψ , formados entre o norte e a projeção do eixo longitudinal do drone para com plano horizontal.

O controle do AR.Drone é exercido por três canais: o primeiro controla e configura o drone através do envio de comandos AT na porta UDP 5556; o segundo canal fornece informações sobre o drone (como posição, velocidade dos motores, altitude, entre outros), que são chamadas de navdata e são enviadas do drone para o cliente na porta UDP 5554; por último, o terceiro canal é responsável por enviar uma stream de vídeo para o cliente na porta 5555 (UDP para o AR.Drone 1.0 e TCP para o AR.Drone 2.0) [Hvizdoš e Sincák 2015].

2.5 PS-Drone

PS-Drone [psdrone 2016] é uma API escrita em Python para AR.Drone 2.0 da Parrot. É parte de uma dissertação de mestrado em Licenciatura em Ciências da Computação e foi projetada para ser de fácil aprendizado e utilização. Somando a isso, oferece um conjunto completo de funções de configuração e utilização para o AR.Drone 2.0.

2.6 Py-HLA

O PyHLA [PyHLA 2016] é um módulo escrito em Python para modelagem e simulação com HLA em simulação distribuída em sistemas de propósito geral. O módulo PyHLA pode ser utilizado em uma variedade de combinações de plataforma/compilador, incluindo Windows 95 e posteriores, Linux e Sun Solaris. O módulo utiliza Python (versão 2.4 ou superior) e necessita de uma implementação do HLA compatível com a versão do padrão 1.3.

Capítulo 3

Um ambiente para teste de drones

Este Capítulo descreve em detalhes o ambiente para testes de drones proposto.

3.1 Visão Geral do ambiente

O ambiente proposto neste trabalho é um ferramenta que tem como objetivo realizar o diagnóstico em drones reais através da captura e análise dos dados dos voos, em um ambiente controlado, identificando possíveis anomalias nos drones. A ferramenta é composta por dois módulos e tem uma visão geral apresentada na Figura 3.1. Um módulo foi desenvolvido no Ptolemy e é responsável pela criação da sequência de teste e a análise e diagnóstico dos dados; o outro é responsável pelo controle e coleta dos dados do drone durante o voo. Para isso foi implementado um modulo em Python. A comunicação entre os módulos é feita através do HLA.

Em simulações seguindo a High-Level Architecture (HLA) [IEEE 2000], cada participante é um federado que envia e recebe mensagens, e o conjuntos de todos os federados que trocam informações entre si criam uma federação. O *run-time infrastructure gateway* (RTIG) é o gateway que centraliza a comunicação é responsável por garantir a entrega das mensagens na ordem correta, de acordo um tempo global simulado, chamado aqui de tempo do HLA.

O ambiente proposto (Figura 3.1) divide-se em três partes: Drone Federate, Ptolemy Federate e RTIG/HLA. O Drone Federate é o software e o hardware embarcado no próprio drone, e é responsável por enviar comandos para a movimentação do drone e, também, por

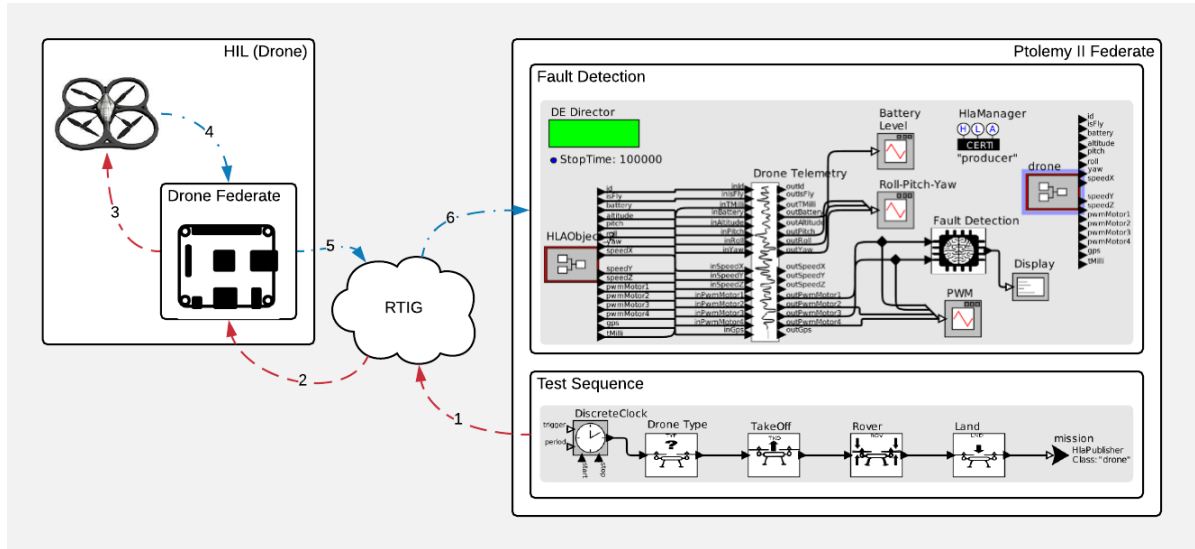


Figura 3.1: Visão Geral do Ambiente

coletar e publicar os dados da telemetria do drone. O Ptolemy Federate contém dois componentes, um responsável pela criação da sequência de teste e outro pela análise dos dados de telemetria publicados pelo Drone Federate. O RTIG é responsável pela comunicação e a sincronização entre o Drone Federate e o Ptolemy Federate.

O Drone Federate utiliza o PyHLA [PyHLA 2016], e o Ptolemy Federate utiliza o JCerti [Certi - summary 2016] para se juntarem à federação. No Ptolemy, a comunicação com o HLA é realizada pelos atores: HLAManager, HLA-Publisher e HLASubscriber (Figura 3.1).

No Ptolemy Federate, para integração com o HLA, foram utilizados atores da própria distribuição do Ptolemy [Cardoso 2018]. O ator HLAManager (Figura 3.1) é um ator especial que não processa mensagens, mas garante que o tempo do modelo seja consistente com o tempo global HLA. Este ator não tem portas de entradas ou saídas. A função do ator HLASubscriber é trazer para o modelo dados publicados na federação através das suas portas de saída. O ator HLAPublisher tem a tarefa de publicar atualizações do modelo para a federação. O federado só pode enviar atualizações para determinado objeto se tiver adquirido propriedade sobre ele. Isso exige que o federado declare anteriormente que publicará atualizações para esses objetos [Come 2015].

A implementação de um teste inicia-se pela definição da missão. A missão é o trajeto que será realizado pelo drone (em um ambiente indoor). Uma missão é criada através da con-

catenação de atores específicos, que foram implementados neste trabalho e acrescentados à biblioteca do Ptolemy II, (Tabela 3.1). Por exemplo, para que o drone realize um trajeto próximo de um quadrado, pode-se definir que o drone deverá: decolar, mover-se para frente durante 2 segundos a uma aceleração de 0,25; mover-se para esquerda durante 2 segundos a 0,25% da aceleração máxima do drone; mover-se para trás durante 2 segundos a 0,25% da aceleração máxima do drone; mover-se para esquerda durante 2 segundos a 0,25% da aceleração máxima do drone; e então pousar. Então, os atores específicos para estas movimentações são incluídos no modelo e concatenados e configurados adequadamente para a realização da missão.

Tabela 3.1: Atores desenvolvidos para testar o drone

Ator	Comando	Descrição	Parâmetros
DroneType	TYP	Especifica o tipo do drone	DroneType, NumberOfRepetitions
TakeOff	TKO	Executa o procedimento de decolagem	Height
Land	LND	executa o procedimento de pouco	none
Rover	ROV	Mantém o drone em uma posição fixa	Time of rover
Forward	FWD	Move o drone para frente	Speed, TimeInSeconds
Backward	BWD	Move o drone para trás	Speed, TimeInSeconds
Left	TLF	Move o drone para esquerda	Speed, TimeInSeconds
Rigth	TRH	Move o drone para direita	Speed, TimeInSeconds
TurnLeft	GLF	Gira o drone 90 graus para esquerda	Speed, TimeInSeconds
TurnRight	GRH	Gira o drone 90 graus para direita	Speed to right, TimeInSeconds
MoveUp	MUP	Aumenta a altitude do drone	Speed, TimeInSeconds
MoveDown	MDW	Diminui a altitude do drone	Speed, TimeInSeconds

Após a definição da missão do drone, executa-se o modelo. Então, uma cadeia de caracteres, contendo a missão do drone é publicada na federação para ser lida pelo Drone Federate. O Drone Federate, ao obter a missão, comanda o drone real para que ele realize a missão definida e, ao mesmo tempo, inicia a coleta e publicação dos dados de telemetria do drone. Os dados publicados pelo Drone Federate na federação, são lidos pelo Ptolemy Federate e

encaminhados ao ator DroneTelemetry, que tem como função a persistência e a preparação dos dados para geração de gráficos e/ou para serem analisados por atores específicos implementados para realização de diagnósticos.

Neste trabalho, foi utilizado o AR.Drone 2.0. O ambiente proposto pode ser adaptado para qualquer tipo de drone, bastando que seja implementado um módulo de controle específico para o drone a ser testado e, então, adicionado ao Drone Federate. Em trabalho anterior, foi desenvolvido um sistema embarcado que pode ser acoplado a qualquer drone para comandá-lo e enviar dados da telemetria para um computador central. [Medeiros et al. 2016]. Em trabalhos anteriores, um federado semelhante ao proposto neste trabalho foi desenvolvido, mas para o monitoramento de robôs terrestres [Brito et al. 2015][Junior et al. 2016].

3.2 Módulo de criação da Sequência de teste (Federado do Ptolemy)

O módulo de criação da sequência de testes (Figura 3.2), é uma parte do federado do Ptolemy e é composto por atores que definem a movimentação que será executada pelo drone.

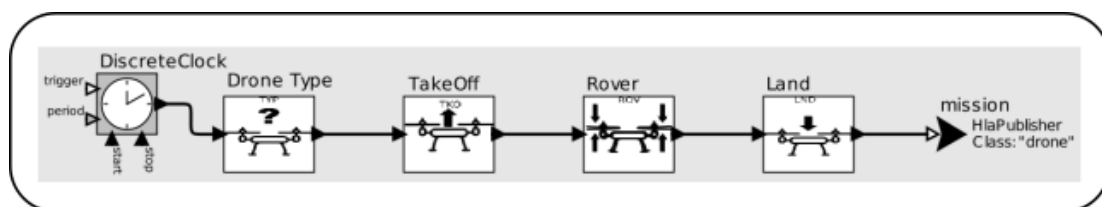


Figura 3.2: exemplo de sequência de teste criada no módulo de controle

Esse módulo é composto inicialmente por dois atores: o ator Discrete Clock e o ator HLA Publish. O ator Discrete Clock é um ator nativo do Ptolemy II e produz tokens com o valor 1 (um inteiro) espaçado de uma unidade de tempo, começando no tempo de início da execução (normalmente, o tempo zero)[Ptolemaeus 2014]. No ambiente, ele é utilizado para iniciar a execução do modelo. O ator HLA Publish (nomeado para Mission no modelo proposto) também é um ator nativo do Ptolemy II e é responsável por publicar a missão na federação

para ser lida pelo Federado do drone. Entre esses dois atores, são concatenados atores de movimentação do drone. A porta de saída do ator DiscretClock deve ser concatenada à porta de entrada do primeiro ator de movimentação (Drone type), e a porta de saída do último ator (Land) deve ser conectada à porta de entrada do ator Mission.

Os atores de movimentação (Tabela 3.1) implementam os movimentos básicos de um drone, como: decolar, mover-se para frente, para os lados e pousar. Vários tipos de trajetórias podem ser configuradas através da concatenação dos atores. Os atores se encontram na biblioteca do usuário do Ptolemy na Subseção "dronesactors", como pode ser visto na Figura 3.3. Para inserir um ator no teste, basta clicar nele e arrastá-lo para o modelo.

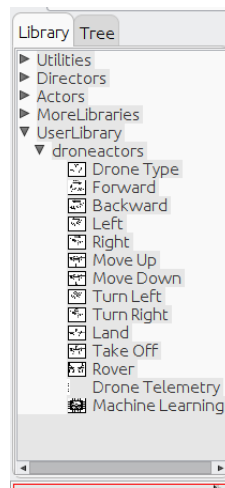


Figura 3.3: Biblioteca de atores movimentação do Drone Ptolemy II

Cada ator de movimentação é composto basicamente por uma porta de entrada e uma porta de saída. Pela porta de entrada, o ator recebe uma cadeia de caracteres contendo um comando de movimentação e parâmetros oriundos do ator anterior. Então, adiciona os seus próprios comandos e parâmetros ao final da cadeia e a envia pela porta de saída para o próximo ator. Essa cadeia de caracteres contém a sequência de movimentação e parâmetros de cada movimento que será executado pelo drone. Esta cadeia de caracteres será encaminhada pelo ator LAND para o ator Mission, que publicará a missão completa na federação. Essa cadeia de caracteres contendo a missão será lida e interpretada pelo Federado do drone. O Código fonte 3.1 apresenta um exemplo da cadeia de caracteres para executar a missão: a)Decolar; b)ficar em estado de rover; c)mover-ser para frente; d)ficar em estado de rover novamente; e)mover-ser para trás e f)então pousar.

Código Fonte 3.1: Exemplo cadeia caracteres com a missão drone

```
1 [ 'TST:DRONE_TEST:NONE' , 'NRP:1:NONE' , 'TYP:ARDRONE:NONE' , 'TKO:2.0:NONE' , 'ROV:NONE:10.0' , '
  FWD:0.25:2.0' , 'ROV:NONE:10.0' , 'BWD:0.25:2.0' , 'LND:NONE:NONE' ]
```

Na cadeia de caracteres, cada comando com seus parâmetros estão entre aspas simples (""") e separados por uma vírgula (","). Dentro da cadeia de cada comando, os parâmetros são separados por dois pontos (":").

A sequência apresentada no Código Fonte 3.1, inicia-se com três comandos do ator DroneType: 1) TST, que define o tipo do teste (a ser utilizado futuramente); 2) NRP, que define o número de repetições da sequência de movimentos (missão); 3) TYP, que define o tipo de drone que será testado. Na sequência, o comando TKO, do ator TakeOff, comanda o drone a decolar e atingir a altura 2.0 metros. Em seguida, o comando ROV, do Ator Rover, comanda o drone a ficar parado por 10 segundos. Logo após, o comando FWD, do ator Forward, comanda o drone a mover-se para frente a velocidade de 0.25% da aceleração máxima do drone, durante 2 segundos. Em seguida, o comando ROV, do Ator Rover, novamente, comanda o drone a ficar parado por mais 10 segundos. Na sequência, o comando BCW, do ator Backward, comanda o drone a mover-se para trás a velocidade de 0.25% da aceleração máxima do drone, durante 2 segundos. Por fim, o comando LND, do ator LAND, comanda o drone a pousar. A janela de configuração dos parâmetros do ator Forward (comum a todos os atores de movimento) pode ser vista na figura 3.4. Esta tela é acessível com um clique duplo do mouse sobre o ator.

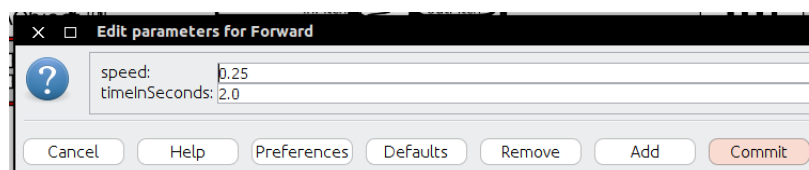


Figura 3.4: Janela de configuração dos parâmetros do ator Forward

O código fonte dos atores DroneType, TakeOff, MoveForward e Land está disponível no anexo I.

3.3 Módulo de controle do drone (Federado do drone)

O Federado do drone (Figure 3.1) está implementado em uma placa Raspberry Pi, executando o sistema operacional Raspbian [2018], e uma interface wifi externa acoplada a uma porta USB . A Raspberry pode ser substituída por um computador ou outro modelo de placa, que suporte a linguagem Python 2.0 e o hardware para controle do drone (se necessário) . A placa pode ser embarcada no drone, ou não. Isso dependerá do tipo do drone. Por exemplo, o ARDrone pode ser controlado remotamente via wifi e estão disponíveis várias API para controle e aquisição de dados, não sendo necessário embarcar uma placa no drone. O Federado do Drone é implementado na aplicação DroneFederate.py (Figure 3.5) escrito em Python 2.0. A aplicação DroneFederate.py é responsável por receber a sequência de teste publicado pelo Federado do Ptolemy, controlar o drone para execução da missão definida e simultaneamente coletar e publicar na federação dados da telemetria do drone.

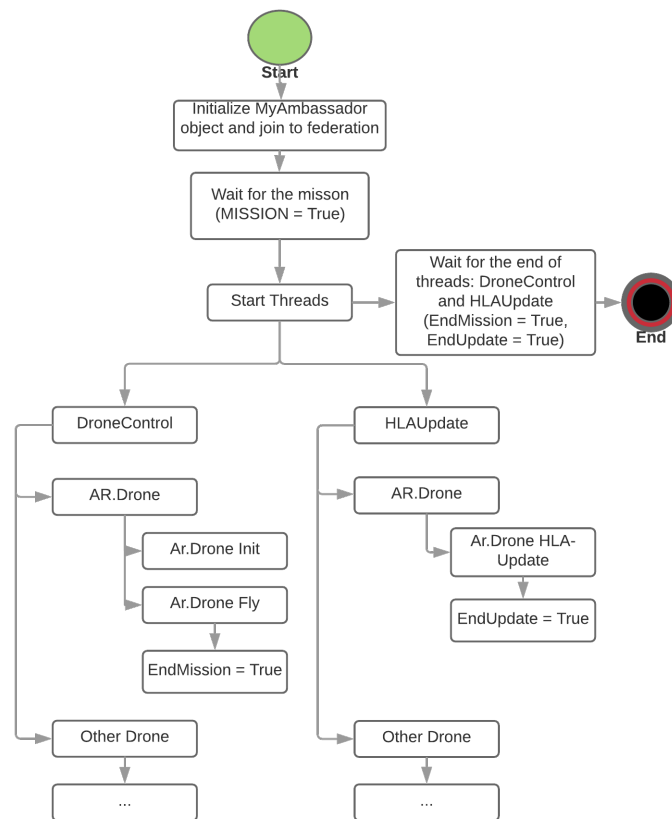


Figura 3.5: Algoritmo executado no Federado do Drone pelo programa DroneFederate.py

A aplicação utiliza duas threads para executar essas funções simultaneamente. Ao execu-

tar o DroneFedereate.py (Figura 3.5), são inicialmente declaradas três variáveis globais com o valor falso: Mission, para sinalizar o recebimento da missão; EndControl para sinalizar o final da thread DroneControl e EndUpdate, para sinalizar o final da thread HLAUpdate. Uma lista das variáveis globais suas descrições pode ser observada na Tabela 3.2. Na sequência, é criado o objeto MyAmbassador (classe que contém os métodos de conexão com o RTIG/HLA) e então é realizada a inclusão do Drone Federate na federação. O programa, então, aguarda a publicação (pelo Ptolemy Federate) da missão a ser executada pelo drone. Ao receber a missão, a variável Mission altera o seu valor para verdadeiro, e a execução do programa continua. Após o recebimento da missão são inicializadas duas threads: Drone Control e HLAUpdate. A thread Drone Control executa a movimentação do drone, e a thread HLAUpdate coleta os dados de telemetria durante o voo do drone e publica na federação (para serem lidos pelo Ptolemy Federate).

A thread de controle drone é iniciada pela função droneControlWrapper (Código fonte 3.2). Esta função é responsável por: a) separar os comandos da cadeia de caracteres que contém a missão, b) iniciar o drone específico de acordo com o parâmetro DroneType, c) executar os comandos da missão no drone específico. A função droneControlWrapper não recebe parâmetros de entrada, entretanto, da linha 2 à linha 7, são iniciadas variáveis globais que são utilizadas por toda a aplicação. A saber, AllCmdParStr é uma cadeia de caracteres que contém a missão, NumberOfRepetitions receberá a quantidade de repetições da missão (definido no ator DroneType), DroneFLY quando setada com o valor 1 indica que o drone está executando os movimentos da missão (está voando) e a variável EndMission quando setada com o valor "OK" indica que o drone finalizou a missão. Na linha 9, é declarada a variável CmdList que é uma variável do tipo lista para receber os comandos de movimentação do drone. Da linha 11 até à linha 22, os comandos são extraídos da variável AllCmdParStr. Na linha 19, ao ser extraído o tipo do drone, a variável global DroneType é setada e a função de iniciação do drone é executada (para o drone especificado no valor da variável DroneType). Por exemplo, no caso do valor da variável DroneType ser igual a "ARDRONE", é executada a função ARDRONE_INIT.

Código Fonte 3.2: Função droneControlWrapper

```
1 def droneControlWrapper():
2     global AllCmdParStr
3     global TestType
```

```

4      global NumberOfRepetitions
5      global DroneFLY
6      global IdxRep
7      global EndMission
8      timeBetweenRepetition = 3.0
9      cmdList = []
10     DroneType = ''
11     for cmdParStr in AllCmdParStr:
12         cmdStr, parStr1, parStr2 = cmdParStr.split(":")
13         if (cmdStr == "TST"):
14             TestType = parStr1
15         elif (cmdStr == "NRP"):
16             NumberOfRepetitions = int(parStr1);
17         elif (cmdStr == "TYP"):
18             DroneType = parStr1
19             if (DroneType == "ARDRONE"):
20                 ARDRONE_INIT()
21             else:
22                 cmdList.append([cmdStr, parStr1, parStr2])
23     for IdxRep in range(0, NumberOfRepetitions):
24         DroneFLY = "1"
25         for i in range(len(cmdList)):
26             if (DroneType == "ARDRONE"):
27                 ARDRONE_FLY(cmdList[i][0], cmdList[i][1], cmdList[i][2])
28         DroneFLY = "-1"
29         time.sleep(timeBetweenRepetition)
30     EndMission = True

```

A função ARDRONE_INIT (código Fonte 3.3) utiliza a biblioteca PS-Drone para iniciar e preparar o drone para executar a missão. Na linha 5, um objeto global "drone" do tipo `ps_drone` é instanciado. Logo após, na linha 6, o drone é iniciado com o método `startup`, que verifica se o drone está conectado e acessível, na linha seguinte, o método `reset` reinicia procedimentos internos do drone e altera os leds externos do drone de vermelho para verde, indicando que o drone está pronto para voar. As linhas 8,9 e 10 alteram o valor da variável `ControlOk` para "OK", indicando que a thread de controle esta pronta para executar a missão, e, então, o programa entra em laço aguardando que a variável global `UpdateOk` também tenha seu o valor alterado para "OK", indicando que a thread de publicação dos dados de telemetria do drone também está pronta pra iniciar suas publicações. Isto é necessário para sincronização entre as threads de controle e atualização, ou seja, para que o drone não comece a executar a missão antes que a thread de atualização tenha sido iniciada ou vice-versa.

Código Fonte 3.3: Função ARDRONE_INIT.

```
1 def ARDRONE_INIT() :
2     global drone
3     global UpdateOK
4     global ControlOK
5     drone = ps_drone.Drone() # Start using drone
6     drone.startup()
7     drone.reset()
8     ControlOK = 'OK'
9     while (UpdateOK != 'OK') or (ControlOK != 'OK') :
10         pass
```

Logo após a inicialização do drone, a função `droneControlWrapper` inicia a execução dos comandos de movimentação do drone. Então, na linha 24, é iniciado um laço para repetir a missão de acordo com o valor da variável `NumberOfRepetitions`. Logo após, a variável `DroneFly` é setada com o valor 1, indicando que o drone iniciará os movimentos da missão. Então, para cada entrada na lista `cmdList`, é executada a movimentação equivalente no drone específico. Por exemplo, para o `AR.Drone`, é executada a função `ARDRONEFLY`, passando como parâmetros o comando de movimentação e suas configurações.

A função `ARDRONEFLY` (Código Fonte 3.4) foi implementada especificamente para controlar o `AR.Drone`. A função recebe como parâmetros de entrada: o comando a ser executado, a velocidade que o comando deve ser executado e o tempo que o comando será executado. Para executar os comandos da missão no drone conectado, precisamos utilizar o objeto global `drone`, instanciado na função `ARDRONE_INIT`. Os comandos de movimentação são baseados nos métodos disponíveis na biblioteca `PS_Drone`. Ao executar um método, o comando será executado no drone até que seja executado o método `Stop`.

Podemos fazer uma analogia para a necessidade da utilização do método `Stop` a cada comando de movimentação, tomando por base o funcionamento de um automóvel: ao acelerar um automóvel (método `acelerar`) e se acionar o pedal da embreagem, o mesmo continua a se mover mesmo que a aceleração seja interrompida (inércia), e só irá parar quando o pedal do freio for acionado (método `Stop`). Então, após se executado um método de movimentação no objeto do drone, o drone físico continuará se movendo por causa da inércia até que o método `Stop`, que faz o drone parar, seja executado.

Código Fonte 3.4: Função ARDRONE_FLY

```
1 def ARDRONE_FLY(cmdStr, speed, stopTime):
```

```

2      global drone
3      if (cmdStr == "TKO"):
4          drone.takeoff()
5      elif (cmdStr == "LND"):
6          drone.land()
7      elif (cmdStr == "ROV"):
8          time.sleep(float(stopTime))
9      elif
10         if (cmdStr == "FWD"):
11             drone.moveForward(speed)
12         elif (cmdStr == "BCW"):
13             drone.moveBackward(speed)
14         elif (cmdStr == "TRH"):
15             drone.moveRight(speed)
16         elif (cmdStr == "TLF"):
17             drone.moveLeft(speed)
18         elif (cmdStr == "MUP"):
19             drone.moveUp(speed)
20         elif (cmdStr == "MDW"):
21             drone.moveDown(speed)
22         elif (cmdStr == "GLF"):
23             drone.turnLeft(speed)
24         elif (cmdStr == "GRH"):
25             drone.turnRight(speed)
26         time.sleep(float(stopTime))
27         drone.stop()
28         time.sleep(float(stopTime))

```

Por fim, na linha 30 da função droneControlWrapper, a variável global endMission tem seu valor alterado pra Verdadeiro (True), informando para a thread de publicação a finalização da execução da missão. Ao perceber a alteração do valor da variável, a thread publicação de dados também finaliza suas publicações.

A thread que realiza a publicação dos dados de telemetria do drone na federação é iniciada pela função updateHLAAttributesWrapper (Código Fonte 3.5). Esta função é responsável por executar a função de coleta e publicação dos dados de telemetria para o drone especificado na variável Global DroneType. No caso, para o AR.Drone (DroneType = "AR-DRONE") é executada a função ARDRONE_HLAUPDATE.

Código Fonte 3.5: Função updateHLAAttributesWrapper.

```

1 def updateHLAAttributesWrapper():
2     global AllCmdParStr
3     for cmdParStr in AllCmdParStr:
4         cmdStr, parStr1, parStr2 = cmdParStr.split(":")

```

```

5         if (cmdStr == "TYP"):
6             DroneType = parStr1;
7             if (DroneType == "ARDRONE"):
8                 ARDRONE_HLAUPDATE()
9             break

```

A função ARDRONE_HLAUPDATE inicia com a definição das variáveis globais: ControlOK, UpdateOK, drone, TestType, DroneFLY, EndMission, EndUpdate. Lembrando que as variáveis globais utilizadas no Federado do Drone podem ser observadas na Tabela 3.2.

Tabela 3.2: Variáveis Globais do Federado do Drone.

Variável	Descrição
ControlOK	Quando seu valor é OK, indica que o modulo de controle do drone está pronto para iniciar a missão.
UpdateOK	Quando seu valor é OK, indica que o modulo de atualização dos dados do drone está pronto para iniciar a transmissão.%
drone	Objeto global para controle do drone.%
DroneFLY	Quando seu valor é igual a 1, indica que o drone está executando missão.%
EndMission	Quando seu valor é OK, indica que o modulo de controle do drone encerrou a missão.%
EndUpdate	Quando seu valor é OK, indica que o modulo de atualização dos dados do drone encerrou a transmissão dos dados.%
TestType	Variável global que contém o tipo do teste %

Logo após a declaração das variáveis globais, a variável UpdateOk tem seu valor alterado para OK, indicando para a thread de controle que a thread de publicação está pronta pra iniciar a publicação dos dados de telemetria. Então, o sistema entra em um laço aguardando que a variável global ControlOK tenha o seu valor alterado para OK, indicando que a Thread de controle também está pronta para iniciar a execução da missão(Código Fonte 3.6). Este laço tem o objetivo de prover sincronismo entre o início da execução da missão pelo drone e o início da publicação dos dados de telemetria na federação. Um laço semelhante também é encontrado na Thread de controle do drone.

Código Fonte 3.6: Função ARDRONE_HLAUPDATE (Sincronizando as Threads).

```

1 UpdateOK = 'OK'
2 while (UpdateOK != 'OK') or (ControlOK != 'OK'):
3     pass

```

Logo após a sincronização da threads, o drone é preparado para a coleta dos dados de telemetria (Código Fonte 3.7). Na linha 1, o modo demo do Ar.drone foi desativado. Isso foi realizado para se obter acesso completo a todos os conjuntos de dados de telemetria disponível no Ar.Drone, como também para aumentar a quantidade de amostras de dados de telemetria por segundo, aumentando-se as amostras de 15 por segundo (no modo Demo) para 200 por segundo (com como modo demo desativado). Logo após, na linha 2, foram especificados os conjuntos de dados para a coletada das informações de telemetria. Neste caso, utilizaremos os conjuntos de dados demo, pwm e altitude. Do conjunto "demo", foram coletadas informações de Pitch, Roll, yaw em graus e a velocidade estimada em X/Y/Z em mm/s. Do conjunto altitude, foram coletadas informações da "altitude" do drone (durante o voo) em milímetros. Do conjunto "pwm", foram coletados os valores de pwm (Pulse-width mod - modulação por largura de pulso) dos motores, para uma lista completa dos conjuntos de dados de telemetria fornecidos pela biblioteca PS_DRONE veja [psdrone 2016].

Código Fonte 3.7: Função ARDRONE_HLAUPDATE (Selecionado dados de telemetria).

```

1 drone.useDemoMode(False)
2 drone.getNDpackage(["demo", "pwm", "altitude"])

```

Após a definição dos dados de telemetria que serão coletados, é publicada, na federação, uma atualização com o parâmetro idHandle setado com o valor -111111". Esta publicação tem como objetivo de informar ao ator DroneTelemetry (do Federado do Ptolemy) o início da transmissão dos dados de telemetria (Código Fonte 3.8). Ao receber uma atualização com esse valor, o ator droneTelemetry cria o arquivo com os dados de telemetria publicados. Na linha 1, é declarada uma variável que vai armazenar o tempo do federado na simulação. Na linha 2, a variável que armazena o tempo do federado é atualizada com o tempo atual do federado somado mais um. Na linha 3, é mostrado o método que publica na RTI com o tempo futuro do federado e o novo valor para a variável idHandle. Em seguida, o federado solicita para avançar para o tempo do próximo evento na linha 4 e, depois disso, nas linhas 5 e 6, 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, posteriormente, na linha 8, é atualizada para False.

Código Fonte 3.8: Função `ARDRONE_HLAUPDATE` (Sinalização de início de publicação dos dados de telemetria).

```
1 federateTime = 0
2 federateTime = rtia.queryFederateTime() + 1
3 rtia.updateAttributeValues(mya.myObject, {mya.idHandle: "-111111" + " "}, "update",
    federateTime)
4 federateTime = rtia.queryFederateTime() + 1;
5 rtia.nextEventRequest(federateTime)
6 while (mya.advanceTime == False):
7     rtia.tick()
8 mya.advanceTime = False
```

Então, após a publicação da sinalização do início da publicação dos dados de telemetria, a publicação começa efetivamente (Código Fonte 3.9). Os dados de telemetria devem ser publicados enquanto o drone executa a missão definida. Para isso, a publicação dos dados está contida dentro de um laço que será executado enquanto a variável global `EndMission` tiver o valor falso (indicando que a missão ainda não foi finalizada). Esta variável terá seu valor alterado para verdadeiro ao final da função `droneControlWrapper`. Na linha 2, a variável que armazena o tempo do federado é atualizada com o tempo atual do federado somado mais um. Da linha 3 até à linha 20, é apresentado o método que publica na RTI - com o tempo futuro do federado - os dados de telemetria coletados do drone. Cada valor a ser publicado na federação tem seu valor alterado pelo retorno de um método do drone executado e que retorna valores de telemetria. Por exemplo, para a atualização do atributo `mya.batteryHandle` do HLA, seu valor é alterado pela chamada do método `drone.getBattery()[0]`. Os valores retornados pelos métodos do drone têm seu tipo convertido em para o tipo `String`. Em seguida, o federado solicita para avançar para o tempo do próximo evento na linha 4 e, depois disso, nas linhas 5 e 6, 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, posteriormente, na linha 8 é atualizada para False. Como mencionado, essas atualizações estão em laço e são realizadas enquanto a variável `ControlOk` for igual a falso.

Código Fonte 3.9: Função `ARDRONE_HLAUPDATE` (Publicação dos dados de telemetria).

```

1  while not EndMission:
2      federateTime = rtia.queryFederateTime() + 1
3      rtia.updateAttributeValues(mya.myObject,
4          {mya.idHandle: str(IdxRep) + " ",
5           mya.isFly: DroneFLY + " ",
6           mya.tMilli: str(tMilli) + " ",
7           mya.batteryHandle: str(drone.getBattery()[0]) + " ",
8           mya.altitudeHandle: str(drone.NavData["altitude"][3]) + " ",
9           mya.pitchHandle: str(drone.NavData["demo"][2][0]) + " ",
10          mya.rollHandle: str(drone.NavData["demo"][2][1]) + " ",
11          mya.yawHandle: str(drone.NavData["demo"][2][2]) + " ",
12          mya.speedXHandle: str(drone.NavData["demo"][4][0]) + " ",
13          mya.speedYHandle: str(drone.NavData["demo"][4][1]) + " ",
14          mya.speedZHandle: str(drone.NavData["demo"][4][2]) + " ",
15          mya.pwmMotor1Handle: str(drone.NavData["pwm"][0][0]) + " ",
16          mya.pwmMotor2Handle: str(drone.NavData["pwm"][0][1]) + " ",
17          mya.pwmMotor3Handle: str(drone.NavData["pwm"][0][2]) + " ",
18          mya.pwmMotor4Handle: str(drone.NavData["pwm"][0][3]) + " ",
19          mya.gpsHandle: FileName + " "},
20          "update", federateTime)
21
22  ##### Time Management #####
23  federateTime = rtia.queryFederateTime() + 1;
24  rtia.nextEventRequest(federateTime)
25  while (mya.advanceTime == False):
26      rtia.tick()
27  mya.advanceTime = False

```

Quando a variável ControlOk tem o seu valor alterado para verdadeiro (ao final da função droneControlWrapper), o laço se encerra e então é enviada uma sinalização ao ator Drone-Telemetry informando o final da transmissão dos dados de telemetria. O procedimento é semelhante à sinalização de início da transmissão (Código fonte 3.10), apenas o valor do atributo idHandle tem seu valor alterado para "999999", o procedimento pode ser observado da linha 1 até à linha 8. Após a sinalização, na linha 10, a variável global EndUpdate tem seu valor alterado para verdadeiro (True) indicando o final das publicações.

Código Fonte 3.10: Função ARDRONE_HLAUPDATE (Sincronizando as Threads).

```

1  federateTime = rtia.queryFederateTime() + 1
2  rtia.updateAttributeValues(mya.myObject, {mya.idHandle: "-999999" + " "}, "update",
3      federateTime)
4  ##### Time Management #####
5  federateTime = rtia.queryFederateTime() + 1
6  rtia.nextEventRequest(federateTime)
7  while (mya.advanceTime == False):

```

```
7   rtia.tick()
8   mya.advanceTime = False
9
10  EndUpdate = True
```

Por fim, o federado do drone aguarda, em um laço, as variáveis globais EndControl e EndUptadte terem seus valores alterados para verdadeiro, e, então, encerra o programa.

3.4 Módulo de detecção de falhas (Federado do Ptolemy)

O módulo de detecção de falhas (Figura 3.1) é parte integrante do Federado do ptolemy. Ele é responsável por: a) recuperar informações da telemetria do drone publicadas na federação pelo Federado do drone; b) realizar a persistência e encaminhar os dados para geração de gráficos (durante o voo) e c) realizar análises e diagnósticos (utilizando atores implementados especificamente para esse fim). Inicialmente o ator HLAObject Subscribe, que é um ator nativo do Ptolemy II, recupera as informações de telemetria publicados pelo Drone Federate e os encaminha pelas suas portas de saídas para as portas de entrada do ator Drone Telemetry.

O ator HlaSubscriber é responsável por assinar e receber informações, relativas ao atributos assinados, publicadas na federação através do RTI. Neste módulo foram necessários dezesseis atores do tipo HlaSubscriber para receber as informações publicadas pelo Drone Federate. Os atores devem possuir os nomes listados como atributos da classe Drone declarada no arquivo do FED, que é apresentado no Código Fonte 3.13. Esses atores foram adicionados dentro de um ator composto chamado CompositeClassDefinition. Este ator ao ser adicionado ao modelo teve seu nome alterado para drone, por ser o mesmo nome da classe declarada no arquivo FED. O detalhamento do ator composto drone, pode ser observado na Figura 3.6. Cada ator do tipo HLASubscribe esta concatenado a uma porta de saída com o mesmo nome. Ao receber uma atualização publicada na federação, o ator HLASubscribe repassa o valor para uma porta de saída que tem o mesmo nome do atributo assinado. Por exemplo, para o HLASubscribe Battery, existe uma porta de mesmo nome concatenada ao ator, ao receber uma atualização referente ao nível da bateria, o ator HLASubscribe Battery repassa o valor para a porta de saída que tem o mesmo nome.

O ator Drone Telemetry tem a função de tratamento e persistência dos dados recebidos. O ator é composto por portas de entrada e saída correspondentes aos atributos da classe Drone

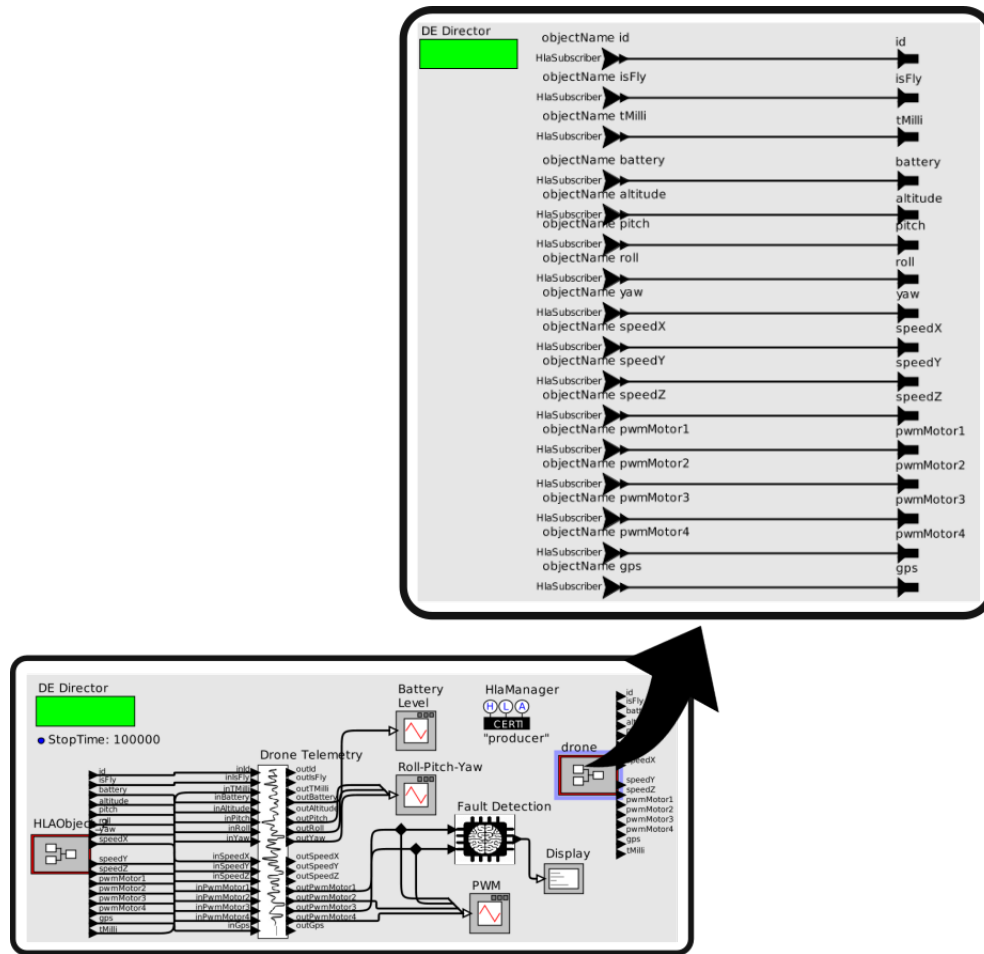


Figura 3.6: Detalhe do ator composto drone.

utilizada pelo HLA (Código Fonte 3.13). O ator recebe nas suas portas de entrada dados oriundos das portas de saída conectadas aos atores HLASubscribe específicos. Para sinalizar ao ator DroneTelemetry a inicialização da publicação de dados de telemetria do drone, o Federado do drone envia uma atualização do parâmetro idHandle com o valor "111111"setado. Ao receber essa atualização o ator DroneTelemetry cria um aquivo do tipo *Comma-separated values* (CSV) para salvar os dados recebidos e os encaminha pelas suas respectivas portas de saída. O ator também é responsável pela conversão dos dados recebidos nas portas de entrada do tipo Texto para o tipo Double. Os dados encaminhado nas portas de saída são utilizados para geração de gráficos, e para a detecção de falha. Por exemplo, um ator, que implementa um algoritmo baseado em aprendizado de máquina para detectar padrões de falhas. A Figura 3.7 apresenta um gráfico com os valores de PWM capturados de um drone pelo ambiente de teste e plotado durante o voo do drone.

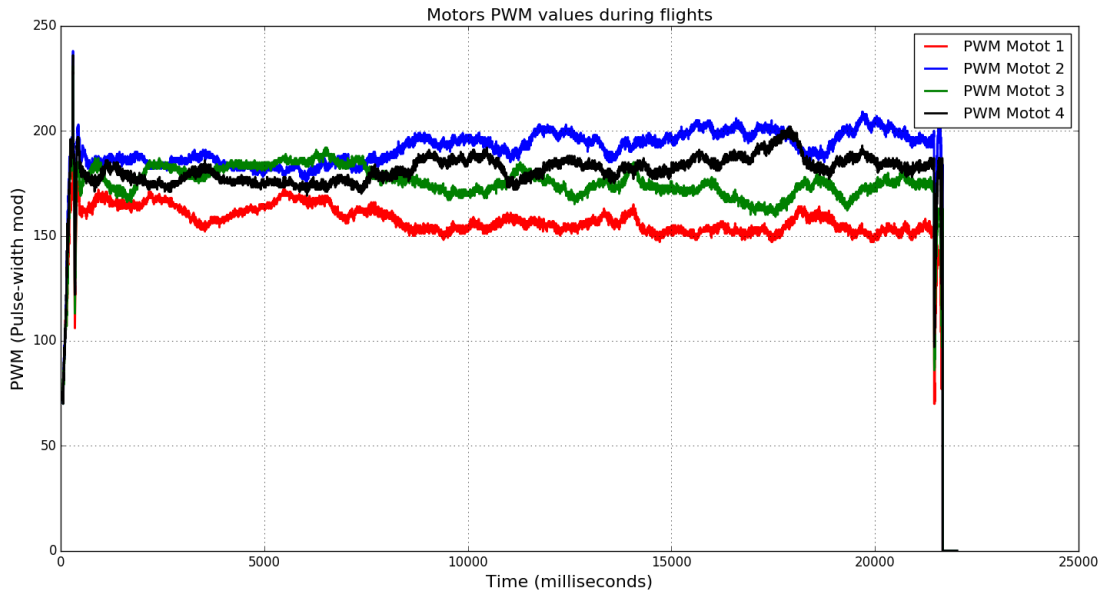


Figura 3.7: Valores do PWM para cada motor, capturado durante o voo.

Para sinalizar ao ator DroneTelemetry a finalização da publicação de dados de telemetria, o Federado do drone envia uma atualização com o parâmetro `idHandle` com o valor `-999999`". Ao receber essa atualização o ator DroneTelemetry salva o arquivo com os dados do voo. O nome do arquivo segue o seguinte padrão: `Voo_<dia><mês><ano><hora><minuto>.csv`. Após o arquivo com os dados do voo ser salvo é apresentado ao usuário uma caixa de diálogo para que o usuário forneça observações e impressões visuais sobre o voo, estes dados são salvos em um arquivo de texto com o nome semelhante ao do arquivo de dados, apenas alterando sua extensão para `".log"` (`Voo_<dia><mes><ano><hora><minuto>.log`). Um exemplo do conteúdo de um arquivo de dados é apresentado no Código Fonte 3.11. Um exemplo do conteúdo de um arquivo de log é apresentado no Código Fonte 3.12.

Código Fonte 3.11: Exemplo arquivo de dados do voo (Voo_010320181212.csv).

```

1 id;isFly;tMilli;battery;altitude;pitch;roll;yaw;speedX;speedY;speedZ;pwmMotor1;pwmMotor2;
  pwmMotor3;pwmMotor4;gps;
2 0;1;0.0;100;239;-0.296;-1.767;0.215;3.96575689316;-14.8758401871;0.0;72;72;72;72;010320181212;

3 0;1;0.0372500419617;100;239;-0.287;-1.734;0.211;3.93264222145;-14.0703144073;0.0;72;72;72;72;010320181212;

4 0;1;0.086345911026;100;239;-0.274;-1.708;0.223;3.91295623779;-13.5841064453;0.0;72;72;72;72;010320181212;

5 0;1;0.0937390327454;100;239;-0.25;-1.693;0.245;3.8669834137;-13.8551645279;0.0;72;72;72;72;010320181212;

6 0;1;0.100692033768;100;239;-0.264;-1.704;0.217;4.03366804123;-13.8844108582;0.0;72;72;72;72;010320181212;

```

Código Fonte 3.12: Exemplo arquivo de Log do voo (Voo_010320181212.log).

```
1 voo instavel – Anomalia inserida (fita isolante) – arrastou para tras e para direita
```

O ator Drone Telemetry foi implementação em uma nova classe na linguagem Java que herda de uma classe de um ator já existente no Ptolemy, o ator TypedAtomicActor. Foram adicionado portas de entrada e de saída, referentes aos atributos listados no arquivo FED. Também foram sobrescritos os métodos que fazem parte do ciclo de vida de um ator no Ptolemy, no caso, o método fire foi sobrescrito.

Código Fonte 3.13: Class Drone defined to be used by HLA for message exchanging

```
1 (FED
2 (class RTIprivate)
3     (class drone
4         (attribute id reliable timestamp)
5         (attribute isFly reliable timestamp)
6         (attribute tMilli reliable timestamp)
7         (attribute battery reliable timestamp)
8         (attribute altitude reliable timestamp)
9         (attribute pitch reliable timestamp)
10        (attribute roll reliable timestamp)
11        (attribute yaw reliable timestamp)
12        (attribute speedX reliable timestamp)
13        (attribute speedY reliable timestamp)
14        (attribute speedZ reliable timestamp)
15        (attribute pwmMotor1 reliable timestamp)
16        (attribute pwmMotor2 reliable timestamp)
17        (attribute pwmMotor3 reliable timestamp)
18        (attribute pwmMotor4 reliable timestamp)
19        (attribute gps reliable timestamp)
20        (attribute mission reliable timestamp)
21    )
22 )
```

Os códigos fontes dos atores DroneTelemetry e DroneForward estão disponíveis nos anexos B e C respectivamente.

Capítulo 4

Avaliação Experimental

4.1 Descrição do experimento

Para verificar a relevância do ambiente proposto, foram realizados dois procedimentos completos de teste e diagnóstico de um drone Ar.Drone 2.0. Estes procedimentos envolveram: 1) a definição da missão a ser executada pelo drone, 2) coleta e publicação de dados de telemetria e a análise, e 3) análise e diagnóstico de possíveis falhas. No primeiro experimento, foi implementada uma missão simples para ser realizada pelo drone. A missão consiste em: fazer o drone decolar, permanecer em estado de rover por 2 minutos e, então, pousar. Já no segundo experimento, a missão consistia em fazer o drone decolar, permanecer em estado de rover por 10 segundos, mover-se para frente, permanecer em estado de rover por 5 segundos, mover-se para trás, permanecer em estado de rover por 5 segundos e então pousar.

Durante a execução das missões, dados de telemetria do drone foram coletados e publicados na Federação. Esses dados foram lidos pelo ator Drone Telemetry no Federado do Ptolemy. O ator Drone Telemetry faz a persistência dos dados e os encaminha para serem analisados por um ator chamado Fault Detection que utiliza um algoritmo de inteligência artificial que foi implementado especificamente para este experimento. Ao final, Este ator apresentará um diagnóstico dos dados analisados.

O principal objetivo dos experimentos foi confirmar se é possível, utilizando o ambiente proposto, diagnosticar um comportamento anômalo do drone testado durante o voo. Para isso foram realizados, para cada experimento, vinte voos de testes sem nenhuma anomalia

inserida, e mais vinte voos de testes com uma anomalia inserida na hélice do motor 4 (Figura 4.1). A anomalia inserida foi uma fita isolante de 5 centímetros de comprimento, enrolada a 2 centímetros da borda da hélice. Essa fita tem um peso aproximado de 0,1 gramas, e introduz um desbalanceamento sutil na hélice. Os voos foram realizados em um ambiente fechado, sem interferências de fatores externos, como vento. Esses dados foram utilizados para treinamento do ator Fault Detection.



Figura 4.1: Ar.Drone com os índices dos motores

O ator Fault Detection implementa um conhecido algoritmo de aprendizado de máquina chamado árvore de decisão. Para a implementação foi utilizando a biblioteca scikit-learn [scikit-learn Machine Learning in Python 2018]. O principal objetivo do ator Fault Detection é detectar a anomalia inserida no drone durante os primeiros minutos de um voo. O ator classifica o voo do drone como estável (sem a anomalia) ou instável (com a anomalia). O algoritmo foi treinado com os dados históricos dos voos salvos. Para cada experimento, foram realizados 20 voos, sendo 10 voos sem anomalia e 10 com a anomalia inserida na hélice. Os voos foram classificados em bons e ruins, então, embaralhados de forma aleatória e dividido em dados de treinamento e teste.

Os dados foram divididos da seguinte maneira: 80% dos voos bons (sem a anomalia) e 80% dos voos ruins (com a anomalia) para treinamento e 20% dos voos bons e 20% dos voos ruins para teste. Para escolha do algoritmo foi implementado um programa de teste para testar a taxa de acerto dos seguintes algoritmos: Decision tree, Naive Bayes e Suporte Vector Machine. Esse programa foi executado 1000 vezes. O algoritmo que apresentou o melhor desempenho nos dois experimentos foi a Decision tree, com 85.3% de taxa de acerto. A comparação entre os algoritmos pode ser observada na Tabela 4.1.

Após a coleta de dados e treinamento do ator Fault Detection, foram realizados vinte voos, para o primeiro experimento e mais vinte voos para segunda dois experimentos, sendo 10 de cada experimento com a anomalia inserida. Os dados coletados durante esses voos

Tabela 4.1: Comparação das técnicas de aprendizagem da máquina para detecção de anomalias

Algoritmo	Número de testes	Taxa de acerto
Árvores de decisão	1000	85.3%
Naive Bayes	1000	60.4%
Máquinas de vetor de suporte	1000	70.7%

foram encaminhados para o ator Fault Detection para apresentação do diagnóstico. No primeiro experimento, dentro os parâmetros disponíveis (ver Tabela 4.2), foi selecionado o parâmetro YAW para ser analisado. Já no segundo experimento, optamos por escolher um parâmetro diferente, então, foi escolhido o parâmetro PWMMotor3 (PWM do motor 3). Estes parâmetros apresentam uma clara diferenciação entre um voo com e sem a anomalia inserida, para os dois experimentos. Isto pode ser observado na Figura 4.2, para o primeiro experimento (Yaw), e na Figura 4.3 para o segundo experimento (PWMMotor3). O aumento da porcentagem da taxa de acertos do algoritmo não faz parte do escopo deste trabalho, mas pretendendo realizar esses ajustes em um trabalho específico com este objetivo.

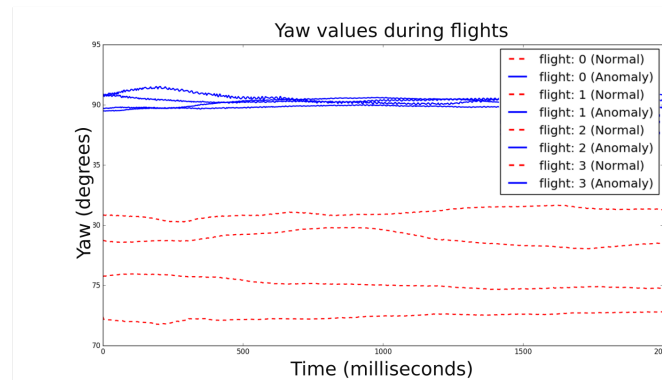


Figura 4.2: Valores do Yaw durante o voo

4.2 Configuração do Ambiente

O experimento foi realizado em um ambiente equipado com uma máquina física e uma placa Raspberry Pi B. A configuração geral da máquina física é apresentada na Tabela 4.3, do mesmo modo a configuração geral placa Raspberry é apresentada na Tabela 4.4. Os softwares

Tabela 4.2: Parâmetros do Drone

#	Parametro	Métodp no AR.Drone	Descrição
1	Battery	getBattery()[0]	Retorna a porcentagem do nível da bateria
2	Pitch	NavData["demo"][2][0]	Retorna o ângulo de pitch θ em mili-graus
3	Roll	NavData["demo"][2][1]	Retorna o ângulo de roll φ em mili-graus
4	Yaw	NavData["demo"][2][2]	Retorna o ângulo de yaw ψ em mili-graus
5	Altitude	NavData["altitude"][3]	Returns the Drone Altitude
6	Velocidade Linear Estimada no Eixo X	NavData["demo"][4][0]	Retorna a velocidade linear no eixo X in m/s
7	Velocidade Linear Estimada no Eixo Y	NavData["demo"][4][1]	Retorna a velocidade linear estimada no eixo Y em m/s
8	Velocidade Linear Estimada no Eixo Z	NavData["demo"][4][2]	Retorna a velocidade linear estimada no eixo Z em m/s

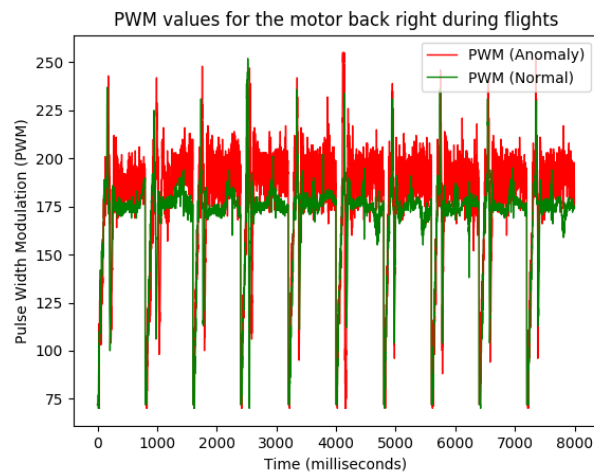


Figura 4.3: valores do PWM do motor 3 durante o voo

utilizados são listados com suas respectivas versões na Tabela 4.5.

Tabela 4.3: Configuração da máquina física.

Tipo	Notebook
Marca	Dell
Modelo	Inspiron 14-3442 - P53G001
Processador	Intel® Core™ i5-5200U CPU @ 2.20GHz × 4
Memória RAM	4,0 GB
Disco Rígido	1 TB
Sistema Operacional	Ubuntu Desktop 16.04
Tipo de Sistema	Sistema Operacional de 64 bits

4.3 Resultados

Como esperando, tanto para o primeiro como para o segundo experimento, o drone executou a sequência de movimentos definidos no ambiente de teste e os dados de telemetria foram coletados no drone e enviados ao ator Drone Telemetry pelo RTIG (HLA).

Para o primeiro experimento, foram realizados vinte voos de 60 segundos cada. Durante cada voo, foram coletados em média, 11,000 amostras contendo todos parâmetros listados

Tabela 4.4: Configuração da Raspberry.

Tipo	SoC
Marca	Raspberry
Modelo	Raspberry Pi model B
Processador	Broadcom BCM2835 - ARM1176JZF-S -700 MHz
Memória RAM	512 MB
Disco Rígido	MicroSD 32 GB
Sistema Operacional	Raspbian - wheezy - 7.0
Tipo de Sistema	Sistema Operacional de 34 bits
Adaptador Wireless USB	IEEE 802.11n - 150mbps Ralink RT5370

Tabela 4.5: Softwares utilizados.

Software	Versão
Java Development Kit	1.8.0_171-b11
Python	2.7.12
Ptolemy II	10.0.1_0
CERTI/RTIG	3.5.1
PyHLA	1.1.1
PsDrone	2.0.0 beta

na Tabela 4.2, ou seja, uma média de 183 amostras por segundo. Metade desses voos não tem nenhuma anomalia inserida, a outra metade tem uma anomalia inserida.

Para o primeiro experimento, o resultado dos diagnósticos realizados pelo ator de detecção de falhas são mostrados na Tabela 4.6. Para voos sem a anomalia inserida a taxa de acerto foi de 60%, ou seja, para 10 voos sem a anomalia, 6 voos foram identificados corretamente como estáveis e 4 foram classificados incorretamente com instáveis. Para voos com a anomalia inserida a taxa de acerto foi de 80%, ou seja, para 10 voos com a anomalia, 8 voos foram identificados corretamente como instáveis e 2 foram classificados incorretamente com estáveis. Os dados mostram que a taxa de acerto para os voos que contém a anomalia inserida é maior que os voos que não contém a falha. O resultado total da taxa de acerto foi de 70%, ou seja, para os 20 voos executados, 14 foram corretamente classificados e 6 foram incorretamente classificados.

Tabela 4.6: Experimento 1 - Taxa de acerto do ator Fault Detection (Yaw)

Tipo	#voos	Taxa de acertos	Taxa de erros
Estável	10	60%	40%
Instável	10	80%	20%
Total	20	70%	30%

Para o segundo experimento, foram realizados 80 voos de 28 segundos cada. Durante cada voo, foram coletados em média 3500 amostras, ou seja, uma média de 58 amostras por segundo, para os voos sem a anomalia. Com a anomalia inserida, foram coletados em média 1800 amostras, ou seja, uma média de 30 amostras por segundos. Houve uma diferença de amostras enviadas pelo drone nos experimentos com e sem a anomalia inserida, outro fato observado é que, mesmo com o menor número de amostras, no segundo experimento, o voo com anomalia demorou mais tempo para executar a missão, como pode ser observado na Figura 4.4.

Para o segundo experimento, o resultado dos diagnósticos realizados pelo ator de detecção de falhas são mostrados na Tabela 4.7. Para voos sem a anomalia inserida a taxa de acerto foi de 90%, ou seja, para 10 voos sem a anomalia, 9 voos foram identificados corretamente como estáveis e 1 foi classificado incorretamente com instável. Para voos com a anomalia

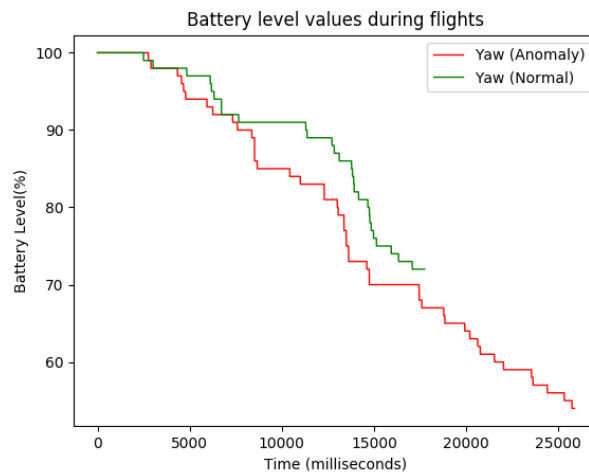


Figura 4.4: Gráfico nível de bateria durante o voo

inserida a taxa de acerto também foi de 90%, ou seja, para 10 voos com a anomalia, 9 voos foram identificados corretamente como instáveis e 1 foi classificado incorretamente com estável. Os dados mostram que a taxa de acerto tanto para os voos que contém a anomalia inserida quanto para os voos que não contém a falha são iguais. O resultado total da taxa de acerto foi de 90%, ou seja, para os 20 voos executados, 18 foram corretamente classificados e 2 foram incorretamente classificados. Vale a pena ressaltar que neste experimento, ao realizar o teste pela primeira vez obtivemos uma taxa de acerto de 100% para os voos com a anomalia inserida e 90% para os voos sem a anomalia. Para verificar o resultado obtido foi criando um programa baseado no ator Fault Detection para realizar novamente o diagnóstico (utilizando os mesmos dados). Este programa realiza o diagnostico com os dados dos voos repetidamente (1000 vezes) e ao final apresenta a média dos acertos, após a execução do programa chegamos a taxa de acerto para os voos com ou sem anomalia mostrado na Tabela 4.7.

Tabela 4.7: Experimento 2 - Taxa de acerto do ator Fault Detection (PWMMotor3)

Tipo	#voos	Taxa de acertos	Taxa de erros
Estável	20	90%	10%
Instável	20	90%	10%
Total	80	40%	10%

Para uma melhor taxa de acerto, é necessário a realização de mais voos para treinamento do algoritmo de inteligência artificial, como também ajustes nos parâmetros de configuração do algoritmo (Árvore de Decisão). Lembramos que a taxa de acerto do ator Fault Detection não é um dos objetivos deste trabalho.

A seguir apresentaremos alguns gráficos que demonstram as diferenças entre os voos com e sem anomalia. Na Figura 4.5, é apresentado o gráfico da porcentagem do nível da bateria durante o voo, nele é possível observar que o nível da bateria decai mais rapidamente no voo com a anomalia inserida. Na Figura 4.6, é apresentado o gráfico com valores de YAW durante o voo, nele é possível observar que os valores do yaw durante o voo são mais estáveis e tem valores maiores durante o voo sem a anomalia inserida, já no voo com a anomalia os valores de yaw tendem a ser menores e apresentar uma variação maior. Na Figura 4.7, é apresentado o gráfico com valores de PWM para os quatros motores durante o voo, nele é possível observar que os valores do PWM durante o voo variam mais durante os voos com a anomalia inserida, já no voo com a anomalia os valores de pwm tendem a variar menos.

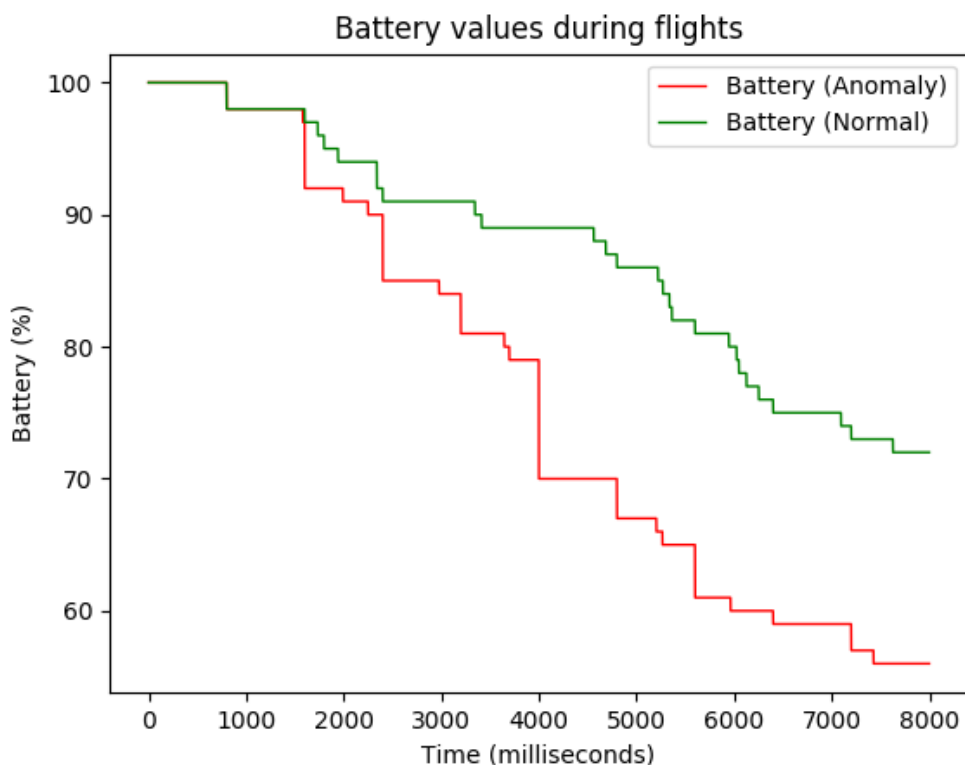


Figura 4.5: Gráfico nível de bateria durante o voo

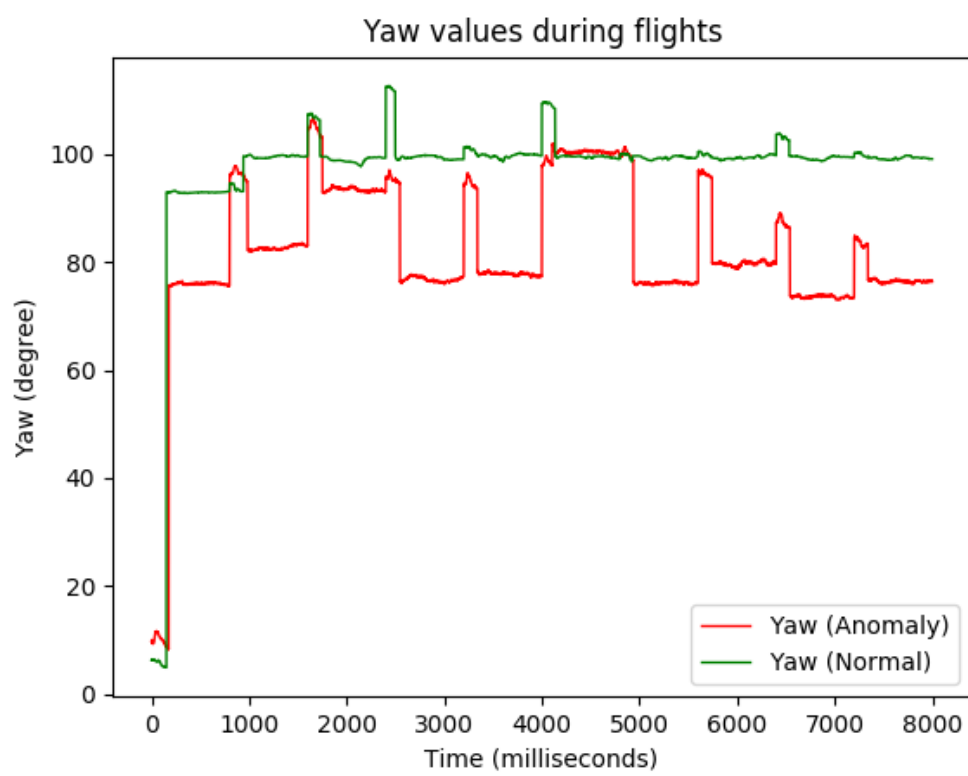


Figura 4.6: Gráfico valores YAW durante o voo

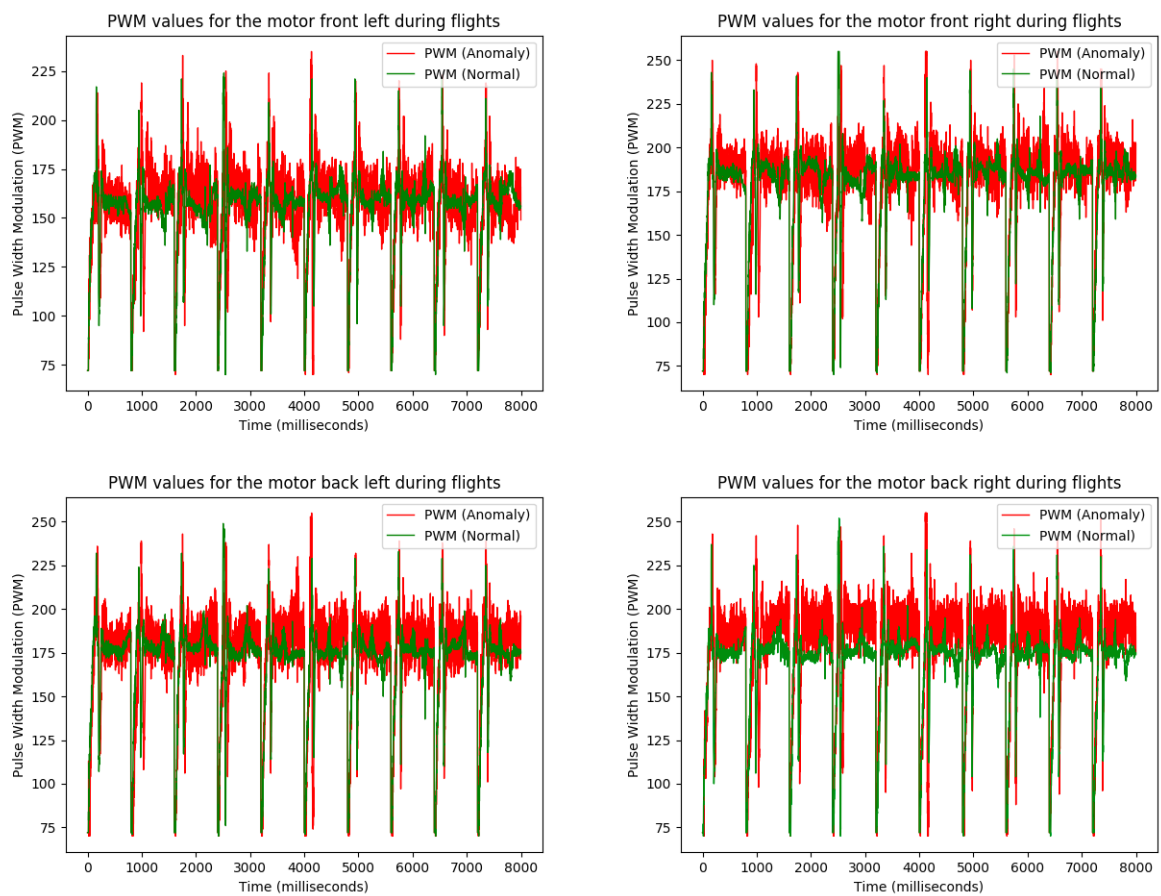


Figura 4.7: Gráfico valores PWM para os 4 motores durante o voo

Capítulo 5

Conclusão

5.1 Considerações Finais

Este trabalho apresentou um ambiente para teste de drones em tempo real. Com o ambiente, é possível implementar vários tipos de testes com o propósito de diagnosticar anomalias presentes nos drones. Neste trabalho, um drone do tipo Ar.Drone2 foi utilizado e um ator chamado Fault Detection foi implementado, utilizando o algoritmo Árvore de decisão, para detectar uma anomalia inserida em alguns voos. O ambiente é aberto para inclusão de novos tipos de drones, como também para a implementação de novos atores de detecção de falhas.

Devido às características do HLA, o ambiente pode ser integrado com outros drones ou outras ferramentas de simulação e análise. Os experimentos demonstraram satisfatoriamente a operação e funcionalidade do ambiente. Foi obtida uma taxa de acerto de 70% com um pequeno número de voos de treinamento e sem ajustes finos no algoritmo Árvore de Decisão, nem qualquer tipo de filtragem de sinal.

Como trabalhos futuros, pretendemos focar na utilização para a implementação de testes que realizem diagnósticos precisos de drones. Para isto, pretendemos: aumentar a base de dados históricos de drones, para treinamento de atores para detecção de falhas; criar novos atores de detecção de falhas baseados em outros algoritmos e outras técnicas; testar novos parâmetros, como também combiná-los para aumentar a taxa de acertos; inserir modelos de controles para novos drones, por exemplo, aqueles controlados pelo ArduPilot; e aumentar o número de ações e incluir sensores, como ultrassom e captura de imagem.

Referências Bibliográficas

[2018]2018. Disponível em: <<https://www.raspberrypi.org/>>.

[2016]2016, Z. *What is ZigBee?* 2016. [Http://www.zigbee.org/what-is-zigbee/](http://www.zigbee.org/what-is-zigbee/). Access: 2016-07-20.

[Ahsan, Rafique e Ahmed 2013]AHSAN, M.; RAFIQUE, H.; AHMED, W. Verification of equilibrium point stability for linearization of an aircraft model. In: *Multi Topic Conference (INMIC), 2013 16th International*. [S.l.: s.n.], 2013. p. 1–6.

[American 2015]AMERICAN, S. 5 epic drone flying failures—and what the faa is doing to prevent future mishaps. 2015. <http://goo.gl/tIXfHH> - Accessed June 2016.

[Barros 2017]BARROS, J. de S. *Uma ferramenta para avaliar estratégias de voos de VANTs usando cossimulação*. Dissertação (Mestrado) — Universidade Federal da Paraíba, Paraíba, Brasil, 2017.

[Bristeau François Callou 2011]BRISTEAU FRANÇOIS CALLOU, D. V. N. P. P.-J. The navigation and control technology inside the ar.drone micro uav. In: *18th IFAC World Congress*. Milano, Italy: [s.n.], 2011. p. 1477–1484.

[Brito e Oliveira 2016]BRITO, A.; OLIVEIRA, T. Simulation and test of communication in multi-robot systems using co-simulation. In: *New Advances in Information Systems and Technologies*. [S.l.]: Springer, 2016. p. 911–917.

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

- [Brito et al. 2013]BRITO, A. V. et al. Development and evaluation of distributed simulation of embedded systems using ptolemy and hla. In: *Distributed Simulation and Real Time Applications (DS-RT), 2013 IEEE/ACM 17th International Symposium on*. [S.l.: s.n.], 2013. p. 189–196. ISSN 1550-6525.
- [Cardoso 2018]CARDOSO, J. *Ptolemy-HLA Framework User Manual V0.1*. 2018. <https://github.com/icyphy/ptII/blob/master/org/hlacerti/manual-ptii-hla.pdf>. Access: 2018-10-25.
- [Certi - summary 2016]CERTI - summary. 2016. <http://savannah.nongnu.org/projects/certi>. Access: 2018-03-27.
- [Come 2015]COME, D. *Improving the HLA-CERTI framework*. [S.l.], Sep 2015. Disponível em: <<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-202.html>>.
- [Deng e Yuan 2015]DENG, D.; YUAN, H. Uav flight safety ground test and evaluation. In: *IEEE AUTOTESTCON, 2015*. [S.l.: s.n.], 2015. p. 422–427.
- [Ducard, Kulling e Geering 2007]DUCARD, G.; KULLING, K. C.; GEERING, H. P. Evaluation of reduction in the performance of a small uav after an aileron failure for an adaptive guidance system. In: *2007 American Control Conference*. [S.l.: s.n.], 2007. p. 1793–1798. ISSN 0743-1619.
- [Globo 2016]GLOBO. *Imagens mostram suspeitos apedrejando drone da polícia no RJ*. 2016. <http://g1.globo.com/rio-de-janeiro/noticia/2016/04/imagens-mostram-suspeitos-apedrejando-drone-da-policia-no-rj.html>. Access: 2016-09-12.
- [Hvizdoš e Sincák 2015]HVIZDOŠ, J.; SINCÁK, P. Control library for ar.drone 2.0. In: *Applied Machine Intelligence and Informatics (SAMI), 2015 IEEE 13th International Symposium on*. [S.l.: s.n.], 2015. p. 77–82.
- [IEEE 2000]IEEE. Std. 1516.1-2000. IEEE Standard for Modeling and Simulation (M & S) High Level Architecture (HLA) – Federate Interface Specification. 2000.
- [IEEE 2000]IEEE. Std. 1516.2-2000. IEEE Standard for Modeling and Simulation (M & S) High Level Architecture (HLA) – Framework and Rules. 2000.

- [INTERFACE 2010]INTERFACE. Ieee standard for modeling and simulation (m and s) high level architecture (hla)– federate interface specification - redline. *IEEE Std 1516.1-2010 (Revision of IEEE Std 1516.1-2000) - Redline*, p. 1–378, Aug 2010.
- [Jaw et al. 2008]JAW, L. et al. Model-based approach to validation and verification of flight critical software. In: *Aerospace Conference, 2008 IEEE*. [S.l.: s.n.], 2008. p. 1–8. ISSN 1095-323X.
- [Junior et al. 2016]JUNIOR, J. C. V. et al. Testing real-time embedded systems using high level architecture. *Design Automation for Embedded Systems*, Springer, v. 20, n. 4, p. 289–309, 2016.
- [Junior 2015]JUNIOR, J. C. V. e S. *Verificação de Projetos de Sistemas Embarcados através de Cossimulação Hardware/Software*. Dissertação (Mestrado) — Universidade Federal da Paraíba, Paraíba, Brasil, 2015.
- [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.
- [Lee et al. 2013]LEE, J.-S. et al. Design of virtual flight system for evaluation of solar powered uav. In: *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*. [S.l.: s.n.], 2013. p. 3463–3467. ISSN 1553-572X.
- [Loh, Bian e Roe 2009]LOH, R.; BIAN, Y.; ROE, T. Uavs in civil airspace: Safety requirements. *Aerospace and Electronic Systems Magazine, IEEE*, v. 24, n. 1, p. 5–17, Jan 2009. ISSN 0885-8985.
- [Medeiros et al. 2016]MEDEIROS, V. S. et al. An independent control system for testing and analysis of uavs in indoor environments. In: *IEEE. Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR), 2016 XIII Latin American*. [S.l.], 2016. p. 55–60.
- [Michini et al. 2011]MICHINI, B. et al. Design and flight testing of an autonomous variable-pitch quadrotor. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. [S.l.: s.n.], 2011. p. 2978–2979. ISSN 1050-4729.

- [OMT 2010]OMT. Ieee standard for modeling and simulation (m amp;amp;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.
- [Park, Lee e Yu 2015]PARK, H. B.; LEE, J. S.; YU, K. H. Flight evaluation of solar powered unmanned flying vehicle using ground testbed. In: *Control, Automation and Systems (IC-CAS), 2015 15th International Conference on*. [S.l.: s.n.], 2015. p. 871–874. ISSN 2093-7121.
- [Patelli e Mottola 2016]PATELLI, A.; MOTTOLA, L. Model-based real-time testing of drone autopilots. In: *Proceedings of the 2Nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*. New York, NY, USA: ACM, 2016. (DroNet '16), p. 11–16. ISBN 978-1-4503-4405-0. Disponível em: <<http://doi.acm.org/10.1145/2935620.2935630>>.
- [psdrone 2016]PSDRONE. *PS-Drone - Programming a Parrot AR.Drone 2.0 with Python*. 2016. H<http://www.playsheep.de/drone/>. Access: 2016-07-24.
- [Ptolemaeus 2014]PTOLEMAEUS, C. (Ed.). *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014. Disponível em: <<http://ptolemy.org/books/Systems>>.
- [Ptolemaeus 2014]PTOLEMAEUS, C. (Ed.). *System Design, Modeling, and Simulation using Ptolemy II*. Ptolemy.org, 2014. Disponível em: <<http://ptolemy.org/books/Systems>>.
- [PyHLA 2016]PYHLA. *Using M&S HLA in Python*. 2016. [Http://www.nongnu.org/certi/PyHLA/manual/node3.html](http://www.nongnu.org/certi/PyHLA/manual/node3.html). Access: 2018-03-27.
- [R7 2016]R7. *Drone auxilia Polícia Militar em operações especiais*. 2016. [Http://noticias.r7.com/minas-gerais/mg-no-ar/videos/drone-auxilia-policia-militar-em-operacoes-especiais-16102015](http://noticias.r7.com/minas-gerais/mg-no-ar/videos/drone-auxilia-policia-militar-em-operacoes-especiais-16102015). Access: 2016-09-12.
- [RULES 2010]RULES. Ieee standard for modeling and simulation (m amp;amp;s) high level architecture (hla)– framework and rules - redline. *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000) - Redline*, p. 1–38, Aug 2010.

- [Saito e Mase 2013]SAITO, T.; MASE, K. Dronepilot.net development: Ar.drone sdk supporting native and managed code. In: *Advanced Computer Science Applications and Technologies (ACSAT), 2013 International Conference on*. [S.l.: s.n.], 2013. p. 60–64.
- [scikit-learn Machine Learning in Python 2018]SCIKIT-LEARN Machine Learning in Python. 2018. Disponível em: <<http://scikit-learn.org/stable/>>.
- [Silva et al. 2018]SILVA, T. W. et al. Environment for integration of distributed heterogeneous computing systems. *Journal of Internet Services and Applications*, Springer, v. 9, n. 1, p. 4, 2018.
- [sindepol 2016]SINDEPOL. *IDrone será usado pela Polícia Militar de Minas para monitoramento*. 2016. [Http://sindepol.com.br/site/noticias/drone-sera-usado-pela-policia-militar-de-minas-para-monitoramento.html](http://sindepol.com.br/site/noticias/drone-sera-usado-pela-policia-militar-de-minas-para-monitoramento.html). Access: 2016-09-12.
- [Stojcsics e Molnar 2011]STOJCSICS, D.; MOLNAR, A. Fixed-wing small-size uav navigation methods with hil simulation for aerobot autopilot. In: *Intelligent Systems and Informatics (SISY), 2011 IEEE 9th International Symposium on*. [S.l.: s.n.], 2011. p. 241–245.
- [UOL 2016]UOL. *Contra o tráfico, drone da PM sobrevoa favela do Rio*. 2016. [Http://noticias.uol.com.br/cotidiano/ultimas-noticias/2013/08/27/contra-o-traffic-drone-da-pm-sobrevoa-favela-do-rio-veja-imagens.htm](http://noticias.uol.com.br/cotidiano/ultimas-noticias/2013/08/27/contra-o-traffic-drone-da-pm-sobrevoa-favela-do-rio-veja-imagens.htm). Access: 2016-09-12.
- [VS, Brito e Nascimento 2015]VS, J. C.; BRITO, A. V.; NASCIMENTO, T. P. Verification of embedded system designs through hardware-software co-simulation. *International Journal of Information and Electronics Engineering*, IACSIT Press, v. 5, n. 1, p. 68, 2015.
- [Vázquez et al. 2012]VÁZQUEZ, R. M. et al. Experimental platform of a physical model for a quadrotor helicopter. In: *Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2012 IEEE Ninth*. [S.l.: s.n.], 2012. p. 311–314.
- [Wang et al. 2015]WANG, Z. et al. Experimental verification of the model predictive control with disturbance rejection for quadrotors. In: *Society of Instrument and Control Engineers of Japan (SICE), 2015 54th Annual Conference of the*. [S.l.: s.n.], 2015. p. 778–783.

- [YanJun, Yang e Shenglin 2013]YANJUN, L.; YANG, L.; SHENGLIN, Y. Research on the algorithm of information fusion for height of uav. In: *Intelligent Systems Design and Engineering Applications, 2013 Fourth International Conference on*. [S.l.: s.n.], 2013. p. 523–526.
- [Yoo, Kang e Park 2008]YOO, C.; KANG, Y.; PARK, B. Hardware-in-the-loop test for fault diagnosis system of tilt rotor uav. In: *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*. [S.l.: s.n.], 2008. p. 320–323.

Apêndice A

Apêndice A

Código Fonte A.1: Modulo de Controle do Drone

```
1  #!/usr/bin/env python
2  # Requires PyHLA, see http://www.nongnu.org/certi/PyHLA
3  import hla.rti
4  import hla.omt as fom
5
6  import struct
7  import pdb
8
9  import time
10 import datetime
11 import math
12 import random
13
14 import os
15 import sys
16 from subprocess import PIPE, Popen, call, check_call
17
18 import argparse
19
20 import ps_drone
21 from threading import Thread
22
23 AllCmdParStr = ""
24 ControlOK = 'NONE'
25 UpdateOK = 'NONE'
26 MissaoOK = 'NONE'
27 drone = ''
28 TestType = 'NONE'
29 NumberOfRepetitions = 'NONE'
30 DroneFLY = "-1"
```

```

31 IdxRep = 0
32
33 global EndMission
34 global EndUpdate
35
36 EndMission = False
37 EndUpdate = False
38
39 date = datetime.datetime.now().strftime("%d%m%Y%H%M")
40
41 FileName = date
42
43 class MyAmbassador(hla.rti.FederateAmbassador):
44     def initialize(self):
45         # Variables
46         self.vehicle = None
47         self.advanceTime = False
48         self.isRegistered = False
49         self.isAnnounced = False
50         self.isReady = False
51         self.isConstrained = False
52         self.isRegulating = False
53         self.attributes = None
54
55         self.classHandleDrone = rtia.getObjectClassHandle("ObjectRoot.drone")
56
57         # -----
58         self.idHandle = rtia.getAttributeHandle("id", self.classHandleDrone)
59         self.isFly = rtia.getAttributeHandle("isFly", self.classHandleDrone)
60         self.tMilli = rtia.getAttributeHandle("tMilli", self.classHandleDrone)
61         self.batteryHandle = rtia.getAttributeHandle("battery", self.
62             classHandleDrone)
63         self.altitudeHandle = rtia.getAttributeHandle("altitude", self.
64             classHandleDrone)
65         self.pitchHandle = rtia.getAttributeHandle("pitch", self.classHandleDrone)
66         self.rollHandle = rtia.getAttributeHandle("roll", self.classHandleDrone)
67         self.yawHandle = rtia.getAttributeHandle("yaw", self.classHandleDrone)
68         self.speedXHandle = rtia.getAttributeHandle("speedX", self.classHandleDrone
69             )
70         self.speedYHandle = rtia.getAttributeHandle("speedY", self.classHandleDrone
71             )
72         self.speedZHandle = rtia.getAttributeHandle("speedZ", self.classHandleDrone
73             )
74         self.pwmMotor1Handle = rtia.getAttributeHandle("pwmMotor1", self.
75             classHandleDrone)
76         self.pwmMotor2Handle = rtia.getAttributeHandle("pwmMotor2", self.
77             classHandleDrone)

```

```

71         self.pwmMotor3Handle = rtia.getAttributeHandle("pwmMotor3", self.
              classHandleDrone)
72         self.pwmMotor4Handle = rtia.getAttributeHandle("pwmMotor4", self.
              classHandleDrone)
73         self.gpsHandle = rtia.getAttributeHandle("gps", self.classHandleDrone)
74         self.missionHandle = rtia.getAttributeHandle("mission", self.
              classHandleDrone)
75
76         rtia.subscribeObjectClassAttributes(self.classHandleDrone, [
77             self.idHandle, self.missionHandle
78         ])
79
80         rtia.publishObjectClass(self.classHandleDrone, [
81             self.idHandle, self.isFly, self.tMilli, self.batteryHandle, self.
              altitudeHandle, self.pitchHandle,
82             self.rollHandle, self.yawHandle, self.speedXHandle, self.speedYHandle, self.
              speedZHandle,
83             self.pwmMotor1Handle, self.pwmMotor2Handle, self.pwmMotor3Handle, self.
              pwmMotor4Handle,
84             self.gpsHandle, self.missionHandle
85         ])
86
87         self.myObject = rtia.registerObjectInstance(self.classHandleDrone) # , "
              ROBO_2")
88
89         # RTI callbacks
90         def terminate(self):
91             rtia.deleteObjectInstance(self.myObject, "ROBO_1")
92
93         def startRegistrationForObjectClass(*params):
94             print("START", params)
95
96         def provideAttributeValueUpdate(*params):
97             print("PROVIDE UAV", params)
98
99         def discoverObjectInstance(self, object, objectclass, name):
100             print("DISCOVER", name)
101             print("Object", object)
102             print("objectclass", objectclass)
103
104         def reflectAttributeValues(self, object, attributes, tag, order, transport, time=
              None, retraction=None):
105             self.attributes = attributes
106             global AllCmdParStr
107             global MissaoOK
108
109             reflect = attributes[self.missionHandle]

```

```

110         AllCmdParStr = reflect.replace('1;', '').replace('\x00', '').split(";")
111
112         MissaoOK = 'OK'
113
114     def synchronizationPointRegistrationSucceeded(self, label):
115         self.isRegistered = True
116         print ("MyAmbassador: Registration Point Succeeded")
117
118     def announceSynchronizationPoint(self, label, tag):
119         self.isAnnounced = True
120         print ("MyAmbassador: Announce Synchronization Point")
121
122     def federationSynchronized(self, label):
123         self.isReady = True
124         print ("MyAmbassador: Ready to run ")
125
126     def timeConstrainedEnabled(self, time):
127         self.isConstrained = True
128
129     def timeRegulationEnabled(self, time):
130         self.isRegulating = True
131
132     def timeAdvanceGrant(self, time):
133         self.advanceTime = True
134
135
136 def droneControlWrapper():
137     global AllCmdParStr
138     global TestType
139     global NumberOfRepetitions
140     global DroneFLY
141     global IdxRep
142     global EndMission
143
144     timeBetweenRepetition = 3.0
145
146     cmdList = []
147
148     DroneType = ''
149     for cmdParStr in AllCmdParStr:
150         cmdStr, parStr1, parStr2 = cmdParStr.split(":")
151         print("Missao : " + cmdStr + " - " + parStr1 + " - " + parStr2)
152         if (cmdStr == "TST"):
153             TestType = parStr1
154         elif (cmdStr == "NRP"):
155             NumberOfRepetitions = int(parStr1);
156         elif (cmdStr == "TYP"):

```

```

157         DroneType = parStr1
158         if (DroneType == "ARDRONE"):
159             ARDRONE_INIT()
160         else:
161             cmdList.append([cmdStr, parStr1, parStr2])
162         print("Test Type: " + TestType + " Number of Repetitions: " + str(
            NumberOfRepetitions))
163
164         for IdxRep in range(0, NumberOfRepetitions):
165             DroneFLY = "1"
166             print("repeticao : " + str(IdxRep))
167             for i in range(len(cmdList)):
168                 if (DroneType == "ARDRONE"):
169                     ARDRONE_FLY(cmdList[i][0], cmdList[i][1], cmdList[i
                        ][2])
170             DroneFLY = "-1"
171             time.sleep(timeBetweenRepetition)
172
173         EndMission = True
174
175
176 def updateHLAAtributesWrapper():
177     global AllCmdParStr
178
179     print("Missao :", AllCmdParStr[0])
180     for cmdParStr in AllCmdParStr:
181         cmdStr, parStr1, parStr2 = cmdParStr.split(":")
182         if (cmdStr == "TYP"):
183             DroneType = parStr1;
184
185             if (DroneType == "ARDRONE"):
186                 ARDRONE_HLAUPDATE()
187             break
188
189
190 def ARDRONE_INIT():
191     global drone
192     global UpdateOK
193     global ControlOK
194
195     print("Iniciando DroneControl")
196     print("Iniciando Drone - " + time.strftime("%H:%M:%S"))
197
198     print("Conectando/Inicializando Drone")
199     drone = ps_drone.Drone() # Start using drone
200     drone.startup()
201

```

```

202     print("Resetando as configuracoes do Drone")
203
204     # Sets drone's status to good (LEDs turn green when red)
205     drone.reset()
206
207     # sincroniza Threads
208     ControlOK = 'OK'
209     print("Esperando o thread Update ficar OK")
210     while (UpdateOK != 'OK') or (ControlOK != 'OK'):
211         pass
212
213
214 def ARDRONE_FLY(cmdStr, speed, stopTime):
215     global drone
216
217     print("Iniciando o ARDRONE_FLY")
218     if (cmdStr == "TKO"):
219         drone.takeoff() # Drone starts
220     elif (cmdStr == "LND"):
221         drone.land()
222     elif (cmdStr == "ROV"):
223         time.sleep(float(stopTime))
224         print("Rover The Drone " + stopTime + " sec")
225     elif (cmdStr == "FWD"):
226         print("MOVE FWD")
227         drone.moveForward(speed)
228         time.sleep(float(stopTime))
229         drone.stop()
230         time.sleep(float(stopTime))
231     elif (cmdStr == "BCW"):
232         print("MOVE BCW")
233         drone.moveBackward(speed)
234         time.sleep(float(stopTime))
235         drone.stop()
236         time.sleep(float(stopTime))
237         elif (cmdStr == "TRH"):
238             drone.moveRight(speed)
239             time.sleep(float(stopTime))
240             drone.stop()
241             time.sleep(float(stopTime))
242     elif (cmdStr == "TLF"):
243         drone.moveLeft(speed)
244         time.sleep(float(stopTime))
245         drone.stop()
246         time.sleep(float(stopTime))
247     elif (cmdStr == "MUP"):
248         drone.moveUp(speed)

```

```

249         time.sleep(float(stopTime))
250         drone.stop()
251         time.sleep(float(stopTime))
252     elif (cmdStr == "MDW"):
253         drone.moveDown(speed)
254     elif (cmdStr == "GLF"):
255         drone.turnLeft(speed)
256         time.sleep(float(stopTime))
257         drone.stop()
258         time.sleep(float(stopTime))
259     elif (cmdStr == "GRH"):
260         drone.turnRight(speed)
261         time.sleep(float(stopTime))
262         drone.stop()
263         time.sleep(float(stopTime))
264
265
266 def ARDRONE_HLAUPDATE():
267     try:
268         global ControlOK
269         global UpdateOK
270         global drone
271         global DroneFLY
272         global IdxRep
273         global EndMission
274         global TestType
275         global FileName
276         global EndUpdate
277
278
279         federateTime = 0
280
281         # Sincroniza as Thread
282         UpdateOK = 'OK'
283         print("Esperando o thread Control ficar OK")
284         while (UpdateOK != 'OK') or (ControlOK != 'OK'):
285             pass
286
287         # Wait until the drone has done its reset
288         while (drone.getBattery()[0] == -1):
289             time.sleep(0.1)
290
291         drone.useDemoMode(False)
292
293         print("Estabelecendo acesso as informacoes dos sensores do drone")
294         drone.getNDpackage(["demo", "pwm", "pressure_raw", "altitude", "magneto", "wifi"])
295

```

```

296         time.sleep(1.0)
297     NDC = drone.NavDataCount
298     end = False
299
300     # Inicia contagem do tempo
301     tBase = 0.0
302     tMilli = 0.0
303     tIdxRep = IdxRep
304
305     # Envia ao Ptolemy uma sinalizacao de inicio da missao (abrir o arquivo)
306     federateTime = rtia.queryFederateTime() + 1
307     rtia.updateAttributeValues(mya.myObject,
308     {mya.idHandle: "-111111" + " ",
309     mya.isFly: "-999" + " ",
310     mya.tMilli: "-999" + " ",
311     mya.batteryHandle: "-999" + " ",
312     mya.altitudeHandle: "-999" + " ",
313     mya.pitchHandle: "-999" + " ",
314     mya.rollHandle: "-999" + " ",
315     mya.yawHandle: "-999" + " ",
316     mya.speedXHandle: "-999" + " ",
317     mya.speedYHandle: "-999" + " ",
318     mya.speedZHandle: "-999" + " ",
319     mya.pwmMotor1Handle: "-999" + " ",
320     mya.pwmMotor2Handle: "-999" + " ",
321     mya.pwmMotor3Handle: "-999" + " ",
322     mya.pwmMotor4Handle: "-999" + " ",
323     mya.gpsHandle: FileName + " "},
324     "update", federateTime)
325     ##### Time Management #####
326     federateTime = rtia.queryFederateTime() + 1;
327     # rtia.timeAdvanceRequest(federateTime)
328     rtia.nextEventRequest(federateTime)
329     while (mya.advanceTime == False):
330         rtia.tick()
331     mya.advanceTime = False
332
333     print("Final da Tread Atualizacao 0")
334
335     while not EndMission:
336         while drone.NavDataCount == NDC:
337             time.sleep(0.001)
338             # Stop if any key is pressed
339             if drone.getKey():
340                 end = True
341             NDC = drone.NavDataCount
342

```

```

343         # Inicializa o contador de tempo para a contagem começar em zero
344         if (tBase == 0.0):
345             tBase = float(time.time())
346
347         # print("Atualizando dados para o HLA")
348
349         str_idHandle = str(IdxFRep)
350         str_isFly = DroneFLY
351         str_tMilli = str(tMilli)
352         str_batteryHandle = str(drone.getBattery()[0])
353         str_altitudeHandle = str(drone.NavData["altitude"][3]) # Altitude in cm (
354             only when flying)
355         str_pitchHandle = str(drone.NavData["demo"][2][0]) # Pitch in degree (
356             front/back)
357         str_rollHandle = str(drone.NavData["demo"][2][1]) # Roll in degree (left/
358             right)
359         str_yawHandle = str(drone.NavData["demo"][2][2]) # Yaw in degree (turn)
360         str_speedXHandle = str(
361             drone.NavData["demo"][4][0]) # Estimated speed in X in mm/s (forward/
362             backward, only when flyi
363         str_speedYHandle = str(
364             drone.NavData["demo"][4][1]) # Estimated speed in Y in mm/s (left/right,
365             only when flying)
366         str_speedZHandle = str(drone.NavData["demo"][4][2]) # Estimated speed in Z
367             in mm/s (up/down, only when flying)
368         str_pwmMotor1Handle = str(drone.NavData["pwm"][0][0]) # actual pwm for the
369             motor on the front left
370         str_pwmMotor2Handle = str(drone.NavData["pwm"][0][1]) # actual pwm for the
371             motor on the front left
372         str_pwmMotor3Handle = str(drone.NavData["pwm"][0][2]) # actual pwm for the
373             motor on the front left
374         str_pwmMotor4Handle = str(drone.NavData["pwm"][0][3]) # actual pwm for the
375             motor on the front left
376         str_gpsHandle = FileName
377
378         federateTime = rtia.queryFederateTime() + 1
379
380         rtia.updateAttributeValues(mya.myObject,
381             {mya.idHandle: str_idHandle + " ",
382             mya.isFly: str_isFly + " ",
383             mya.tMilli: str_tMilli + " ",
384             mya.batteryHandle: str_batteryHandle + " ",
385             mya.altitudeHandle: str_altitudeHandle + " ",
386             mya.pitchHandle: str_pitchHandle + " ",
387             mya.rollHandle: str_rollHandle + " ",
388             mya.yawHandle: str_yawHandle + " ",
389             mya.speedXHandle: str_speedXHandle + " ",

```

```

380         mya.speedYHandle: str_speedYHandle + " ",
381         mya.speedZHandle: str_speedZHandle + " ",
382         mya.pwmMotor1Handle: str_pwmMotor1Handle + " ",
383         mya.pwmMotor2Handle: str_pwmMotor2Handle + " ",
384         mya.pwmMotor3Handle: str_pwmMotor3Handle + " ",
385         mya.pwmMotor4Handle: str_pwmMotor4Handle + " ",
386         mya.gpsHandle: FileName + " "},
387     "update", federateTime)
388     ##### Time Management #####
389     federateTime = rtia.queryFederateTime() + 1;
390     # rtia.timeAdvanceRequest(federateTime)
391     rtia.nextEventRequest(federateTime)
392     while (mya.advanceTime == False):
393         rtia.tick()
394     mya.advanceTime = False
395
396     # Mede o tempo ( segundos no formato float)
397     tTime = float(time.time())
398     tMilli = tTime - tBase
399
400     print("Final da Tread Atualizacao 1")
401     # Envia ao Ptolemy uma sinalizacao de final da missao (Fechar o arquivo)
402     federateTime = rtia.queryFederateTime() + 1
403     rtia.updateAttributeValues(mya.myObject,
404     {mya.idHandle: "-999999" + " ",
405     mya.isFly: "-999" + " ",
406     mya.tMilli: "-999" + " ",
407     mya.batteryHandle: "-999" + " ",
408     mya.altitudeHandle: "-999" + " ",
409     mya.pitchHandle: "-999" + " ",
410     mya.rollHandle: "-999" + " ",
411     mya.yawHandle: "-999" + " ",
412     mya.speedXHandle: "-999" + " ",
413     mya.speedYHandle: "-999" + " ",
414     mya.speedZHandle: "-999" + " ",
415     mya.pwmMotor1Handle: "-999" + " ",
416     mya.pwmMotor2Handle: "-999" + " ",
417     mya.pwmMotor3Handle: "-999" + " ",
418     mya.pwmMotor4Handle: "-999" + " ",
419     mya.gpsHandle: FileName + " "},
420     "update", federateTime)
421     ##### Time Management #####
422     federateTime = rtia.queryFederateTime() + 1
423     rtia.nextEventRequest(federateTime)
424     while (mya.advanceTime == False):
425         rtia.tick()
426     mya.advanceTime = False

```

```

427         print("Final da Tread Atualizacao 2")
428
429         EndUpdate = True
430     except:
431         print "Mizera !!!!!"
432
433     ##### Time Management #####
434
435     count_string = "1"
436     stopTime = 1
437     print("Create ambassador")
438     rtia = hla.rti.RTIAmbassador()
439
440     federation = "federation"
441     try:
442         rtia.createFederationExecution(federation, "/home/renatoabreu/programas/pthla/
            PyhlaToPtolemy.fed")
443         print("Federation created.")
444     except hla.rti.FederationExecutionAlreadyExists:
445         print("Federation already exists.")
446
447     mya = MyAmbassador()
448     rtia.joinFederationExecution("uav", federation, mya)
449
450     mya.initialize()
451
452     ##### Announce Synchronization Point #####
453
454     label = "ReadyToRun"
455     # so deve descomentar este codigo no ultimo federado a ser executado
456     tag = bytes("ReadyToRun")
457     rtia.registerFederationSynchronizationPoint(label, tag)
458     print("Synchronization Point Register")
459     while (mya.isRegistered == False):
460         rtia.tick()
461     print "tick"
462
463     while (mya.isAnnounced == False):
464         rtia.tick()
465     print "tick"
466
467     ##### Achieve Synchronized Point #####
468     rtia.synchronizationPointAchieved("ReadyToRun")
469     while (mya.isReady == False):
470         rtia.tick()
471     print ("MyAmbassador : Is Ready to run ")
472

```

```

473 ##### Enable Time Policy #####
474 currentTime = rtia.queryFederateTime()
475 lookahead = 1 # rtia.queryLookahead()
476 rtia.enableTimeRegulation(currentTime, lookahead)
477
478 while (mya.isRegulating == False):
479     rtia.tick()
480
481 rtia.enableTimeConstrained()
482 while (mya.isConstrained == False):
483     rtia.tick()
484 #####
485
486 try:
487     print("Esperando o MISSAO ficar OK")
488     while (MissaoOK != "OK"):
489         ##### Time Management #####
490         federateTime = rtia.queryFederateTime() + 1;
491         rtia.nextEventRequest(federateTime)
492         while (mya.advanceTime == False):
493             rtia.tick()
494         mya.advanceTime = False
495
496         t1 = Thread(target=updateHLAAttributesWrapper, args=())
497         t2 = Thread(target=droneControlWrapper, args=())
498         t1.start()
499         t2.start()
500
501         # Preciso disso para o CTRL+C?
502         while (not EndMission) or (not EndUpdate):
503             pass
504
505         print "CHEGUEI AQUI (Final das Threads ?) !!!!!"
506
507 except KeyboardInterrupt:
508     print('preso aqui o !')
509     pass
510
511 rtia.resignFederationExecution(hla.rti.ResignAction.ReleaseAttributes)
512 print("Completed")
513 print("Done.")

```

Apêndice B

Apêndice B

Código Fonte B.1: Ator Move forward

```
1 package ptolamy.actor.lib;;
2
3 import java.awt.BorderLayout;
4 import java.awt.Container;
5 import java.awt.Font;
6 import java.awt.event.ActionEvent;
7 import java.awt.event.ActionListener;
8
9 import javax.swing.JButton;
10 import javax.swing.JFrame;
11 import javax.swing.JPanel;
12 import javax.swing.JScrollPane;
13 import javax.swing.JTextArea;
14
15 import java.util.Map;
16 import java.util.SortedSet;
17 import java.util.TreeSet;
18 import java.util.Map.Entry;
19
20 import java.io.BufferedReader;
21 import java.io.BufferedWriter;
22 import java.io.FileReader;
23 import java.io.FileWriter;
24 import java.io.PrintWriter;
25 import java.io.IOException;
26 import java.text.SimpleDateFormat;
27 import java.util.Date;
28
29 import java.sql.*;
30
```

```

31 import ptolemy.actor.TypedAtomicActor;
32 import ptolemy.actor.TypedIOPort;
33 import ptolemy.data.StringToken;
34 import ptolemy.data.DoubleToken;
35 import ptolemy.data.IntToken;
36 import ptolemy.data.expr.Parameter;
37 import ptolemy.data.type.BaseType;
38 import ptolemy.kernel.CompositeEntity;
39 import ptolemy.kernel.util.IllegalActionException;
40 import ptolemy.kernel.util.NameDuplicationException;
41
42 public class DroneTelemetryManager extends TypedAtomicActor {
43
44     private static final long serialVersionUID = 1L;
45
46     public TypedIOPort inId;
47     public TypedIOPort outId;
48
49     public TypedIOPort inIsFly;
50     public TypedIOPort outIsFly;
51
52     public TypedIOPort inTMilli;
53     public TypedIOPort outTMilli;
54
55     public TypedIOPort inBattery;
56     public TypedIOPort outBattery;
57
58     public TypedIOPort inAltitude;
59     public TypedIOPort outAltitude;
60
61     public TypedIOPort inPitch;
62     public TypedIOPort outPitch;
63
64     public TypedIOPort inRoll;
65     public TypedIOPort outRoll;
66
67     public TypedIOPort inYaw;
68     public TypedIOPort outYaw;
69
70     public TypedIOPort inSpeedX;
71     public TypedIOPort outSpeedX;
72
73     public TypedIOPort inSpeedY;
74     public TypedIOPort outSpeedY;
75
76     public TypedIOPort inSpeedZ;
77     public TypedIOPort outSpeedZ;

```

```
78
79     public TypedIOPort inPwmMotor1;
80     public TypedIOPort outPwmMotor1;
81
82     public TypedIOPort inPwmMotor2;
83     public TypedIOPort outPwmMotor2;
84
85     public TypedIOPort inPwmMotor3;
86     public TypedIOPort outPwmMotor3;
87
88     public TypedIOPort inPwmMotor4;
89     public TypedIOPort outPwmMotor4;
90
91     public TypedIOPort inGps;
92     public TypedIOPort outGps;
93
94     //public Parameter actions;
95     //public Parameter knowledgeBase;
96     //public Parameter TimeToRoverP;
97     //public Parameter timeInSecondsP;
98     /*
99     private String id;
100         private String battery;
101         private String temperature;
102         private String sensor1;
103         private String sensor2;
104         private String sensor3;
105         private String gps;
106         private String compass;
107         private String goto;
108         private String rotate;
109         private String activate;
110     */
111     //private int indice;
112     //private float altitude;
113     //private float timeInSeconds;
114     private String id;
115     private String isFly;
116     private String tMilli;
117     private String battery;
118     private String altitude;
119     private String pitch;
120     private String roll;
121     private String yaw;
122     private String speedX;
123     private String speedY;
124     private String speedZ;
```

```

125     private String pwmMotor1;
126     private String pwmMotor2;
127     private String pwmMotor3;
128     private String pwmMotor4;
129     private String gps;
130
131
132     //private DroneKnowledgeBase droneKnowledgeBase;
133
134     public DroneTelemetryManager(CompositeEntity container, String name) throws
        IllegalArgumentException, NameDuplicationException {
135         super(container, name);
136
137         // set values parameter
138         /*
139         actions = new Parameter(this, "actions");
140         actions.setExpression("");
141         actions.setStringMode(true);
142
143         knowledgeBase = new Parameter(this, "knowledgeBase");
144         knowledgeBase.setExpression("");
145         knowledgeBase.setStringMode(true);
146         */
147     //=====
148
149         inId = new TypedIOPort(this, "inId", true, false);
150         inId.setTypeEquals(BaseType.STRING);
151         outId = new TypedIOPort(this, "outId", false, true);
152         outId.setTypeEquals(BaseType.DOUBLE);
153
154         inIsFly = new TypedIOPort(this, "inIsFly", true, false);
155         inIsFly.setTypeEquals(BaseType.STRING);
156         outIsFly = new TypedIOPort(this, "outIsFly", false, true);
157         outIsFly.setTypeEquals(BaseType.DOUBLE);
158
159         inTMilli = new TypedIOPort(this, "inTMilli", true, false);
160         inTMilli.setTypeEquals(BaseType.STRING);
161         outTMilli = new TypedIOPort(this, "outTMilli", false, true);
162         outTMilli.setTypeEquals(BaseType.DOUBLE);
163
164         inBattery = new TypedIOPort(this, "inBattery", true, false);
165         inBattery.setTypeEquals(BaseType.STRING);
166         outBattery = new TypedIOPort(this, "outBattery", false, true);
167         outBattery.setTypeEquals(BaseType.DOUBLE);
168
169         inAltitude = new TypedIOPort(this, "inAltitude", true, false);
170         inAltitude.setTypeEquals(BaseType.STRING);

```

```
171         outAltitude = new TypedIOPort(this , "outAltitude", false , true);
172         outAltitude.setTypeEquals(BaseType.DOUBLE);
173
174         inPitch = new TypedIOPort(this , "inPitch", true , false);
175         inPitch.setTypeEquals(BaseType.STRING);
176         outPitch = new TypedIOPort(this , "outPitch", false , true);
177         outPitch.setTypeEquals(BaseType.DOUBLE);
178
179         inRoll = new TypedIOPort(this , "inRoll", true , false);
180         inRoll.setTypeEquals(BaseType.STRING);
181         outRoll = new TypedIOPort(this , "outRoll", false , true);
182         outRoll.setTypeEquals(BaseType.DOUBLE);
183
184         inYaw = new TypedIOPort(this , "inYaw", true , false);
185         inYaw.setTypeEquals(BaseType.STRING);
186         outYaw = new TypedIOPort(this , "outYaw", false , true);
187         outYaw.setTypeEquals(BaseType.DOUBLE);
188
189         inSpeedX = new TypedIOPort(this , "inSpeedX", true , false);
190         inSpeedX.setTypeEquals(BaseType.STRING);
191         outSpeedX = new TypedIOPort(this , "outSpeedX", false , true);
192         outSpeedX.setTypeEquals(BaseType.DOUBLE);
193
194         inSpeedY = new TypedIOPort(this , "inSpeedY", true , false);
195         inSpeedY.setTypeEquals(BaseType.STRING);
196         outSpeedY = new TypedIOPort(this , "outSpeedY", false , true);
197         outSpeedY.setTypeEquals(BaseType.DOUBLE);
198
199         inSpeedZ = new TypedIOPort(this , "inSpeedZ", true , false);
200         inSpeedZ.setTypeEquals(BaseType.STRING);
201         outSpeedZ = new TypedIOPort(this , "outSpeedZ", false , true);
202         outSpeedZ.setTypeEquals(BaseType.DOUBLE);
203
204         inPwmMotor1 = new TypedIOPort(this , "inPwmMotor1", true , false);
205         inPwmMotor1.setTypeEquals(BaseType.STRING);
206         outPwmMotor1 = new TypedIOPort(this , "outPwmMotor1", false , true);
207         outPwmMotor1.setTypeEquals(BaseType.DOUBLE);
208
209         inPwmMotor2 = new TypedIOPort(this , "inPwmMotor2", true , false);
210         inPwmMotor2.setTypeEquals(BaseType.STRING);
211         outPwmMotor2 = new TypedIOPort(this , "outPwmMotor2", false , true);
212         outPwmMotor2.setTypeEquals(BaseType.DOUBLE);
213
214         inPwmMotor3 = new TypedIOPort(this , "inPwmMotor3", true , false);
215         inPwmMotor3.setTypeEquals(BaseType.STRING);
216         outPwmMotor3 = new TypedIOPort(this , "outPwmMotor3", false , true);
217         outPwmMotor3.setTypeEquals(BaseType.DOUBLE);
```

```

218
219         inPwmMotor4 = new TypedIOPort(this, "inPwmMotor4", true, false);
220         inPwmMotor4.setTypeEquals(BaseType.STRING);
221         outPwmMotor4 = new TypedIOPort(this, "outPwmMotor4", false, true);
222         outPwmMotor4.setTypeEquals(BaseType.DOUBLE);
223
224         inGps = new TypedIOPort(this, "inGps", true, false);
225         inGps.setTypeEquals(BaseType.STRING);
226         outGps = new TypedIOPort(this, "outGps", false, true);
227         outGps.setTypeEquals(BaseType.DOUBLE);
228
229         // =====
230         /*_attachText("_iconDescription", "<svg width=\"30\" height=\"100\">\n"
231                                     + "<rect x=\"0\" y=\"0\" width=\"30\"
232                                           height=\"150\" "
233                                           + "style=\"fill:white\"/>\n"
234                                           + "Telemetry"
235                                           + "</svg>\n");*/
236         _attachText("_iconDescription", "<svg width=\"30\" height=\"175\">\n"
237                                     + "<rect x=\"0\" y=\"0\" width=\"30\"
238                                           height=\"175\" "
239                                           + "style=\"fill:white\"/>\n"
240                                     + "<image x=\"0\" y=\"0\" width=\"30\"
241                                           height=\"175\" xlink:href=\"telemetry.
242                                           png\" />"
243                                     + "</svg>\n");
244     }
245
246     @Override
247     public void fire() throws IllegalArgumentException {
248         super.fire();
249
250         Connection conn = null;
251         Statement stmt = null;
252
253         id = inId.get(0).toString().replace("\\", "");
254         isFly = inIsFly.get(0).toString().replace("\\", "");
255         tMilli = inTMilli.get(0).toString().replace("\\", "");
256         battery = inBattery.get(0).toString().replace("\\", "");
257         altitude = inAltitude.get(0).toString().replace("\\", "");
258         pitch = inPitch.get(0).toString().replace("\\", "");
259         roll = inRoll.get(0).toString().replace("\\", "");
260         yaw = inYaw.get(0).toString().replace("\\", "");
261         speedX = inSpeedX.get(0).toString().replace("\\", "");
262         speedY = inSpeedY.get(0).toString().replace("\\", "");
263         speedZ = inSpeedZ.get(0).toString().replace("\\", "");
264         pwmMotor1 = inPwmMotor1.get(0).toString().replace("\\", "");

```

```

261         pwmMotor2 = inPwmMotor2.get(0).toString().replace("\\", "");
262         pwmMotor3 = inPwmMotor3.get(0).toString().replace("\\", "");
263         pwmMotor4 = inPwmMotor4.get(0).toString().replace("\\", "");
264         gps = inGps.get(0).toString().replace("\\", "");
265
266         System.out.println(id+";"+isFly+";"+tMilli+";"+battery+";"+altitude+";"+
                pitch+";"+
267         roll+";"+yaw+";"+speedX+";"+speedY+";"+speedZ+";"+pwmMotor1+";"+
268         pwmMotor2+";"+pwmMotor3+";"+pwmMotor4+";"+gps);
269
270         String fname = "Voo_"+gps+".csv";
271         System.out.println("ID = "+id);
272
273         if(id.equals("-111111")){
274             System.out.println("INICIO !"+id);
275             try(PrintWriter output = new PrintWriter(new FileWriter(fname,true)
                )){
276                 String Line = "id;isFly;tMilli;battery;altitude;pitch;"+
277                 "roll;yaw;speedX;speedY;speedZ;pwmMotor1;pwmMotor2;
                pwmMotor3;pwmMotor4;"+
278                 "gps;";
279                 output.println(Line);
280                 output.close();
281             } catch (Exception e) {
282                 System.err.println("Exception Error: " + e.getMessage());
283             }
284         } else if(id.equals("-999999")){
285             System.out.println("FIM !"+id);
286             String fnameLog = "Voo_"+gps+".log";
287             new TextEditor(fnameLog);
288         } else {
289             System.out.println("MEIO !"+id);
290             outId.send(0, new DoubleToken(Double.valueOf(id)));
291             outIsFly.send(0, new DoubleToken(Double.valueOf(isFly)));
292             outBattery.send(0, new DoubleToken(Double.valueOf(battery)));
293             outTMilli.send(0, new DoubleToken(Double.valueOf(tMilli)));
294             outAltitude.send(0, new DoubleToken(Double.valueOf(altitude)));
295             outPitch.send(0, new DoubleToken(Double.valueOf(pitch)));
296             outRoll.send(0, new DoubleToken(Double.valueOf(roll)));
297             outYaw.send(0, new DoubleToken(Double.valueOf(yaw)));
298             outSpeedX.send(0, new DoubleToken(Double.valueOf(speedX)));
299             outSpeedY.send(0, new DoubleToken(Double.valueOf(speedY)));
300             outSpeedZ.send(0, new DoubleToken(Double.valueOf(speedZ)));
301             outPwmMotor1.send(0, new DoubleToken(Double.valueOf(pwmMotor1)));
302             outPwmMotor2.send(0, new DoubleToken(Double.valueOf(pwmMotor2)));
303             outPwmMotor3.send(0, new DoubleToken(Double.valueOf(pwmMotor3)));
304             outPwmMotor4.send(0, new DoubleToken(Double.valueOf(pwmMotor4)));

```

```

305         outGps.send(0, new DoubleToken(Double.valueOf(gps)));
306         try(PrintWriter output = new PrintWriter(new FileWriter(fname,true)
307             )){
308             String Line = id+";"+isFly+";"+tMilli+";"+battery+";"+
309                 altitude+";"+pitch+";"+
310                 roll+";"+yaw+";"+speedX+";"+speedY+";"+speedZ+";"+pwmMotor1
311                 +";"+
312                 pwmMotor2+";"+pwmMotor3+";"+pwmMotor4+";"+gps+";";
313             output.println(Line);
314             output.close();
315         }catch (Exception e) {
316             System.err.println("Exception Error: " + e.getMessage());
317         }
318     }
319     @Override
320     public void initialize() throws IllegalArgumentException {
321         super.initialize();
322     }
323 }
324
325 class TextEditor implements ActionListener {
326     // Size of editing text area.
327     private static final int NUM_ROWS = 5;
328     private static final int NUM_COLS = 20;
329
330     // Buttons to save and load files.
331     private JButton saveButton, loadButton;
332
333     // Area where the user does the editing
334     private JTextArea text;
335
336     private String FileName;
337
338     private JFrame frame;
339
340     // Creates the GUI
341     public TextEditor(String FName) {
342         FileName = FName;
343         frame = new JFrame("Drone flight observations");
344         JPanel buttonPanel = new JPanel();
345         saveButton = new JButton("Save Log File");
346         buttonPanel.add(saveButton);
347
348         text = new JTextArea(NUM_ROWS, NUM_COLS);

```

```
349         text.setFont(new Font("System", Font.PLAIN, 24));
350         JScrollPane textScroller = new JScrollPane(text);
351         Container contentPane = frame.getContentPane();
352         contentPane.add(textScroller, BorderLayout.CENTER);
353         contentPane.add(buttonPanel, BorderLayout.NORTH);
354
355         saveButton.addActionListener(this);
356         frame.pack();
357         frame.setVisible(true);
358     }
359
360     // Listener for button clicks that loads the
361     // specified files and puts it in the
362     // editor.
363     public void actionPerformed(ActionEvent event) {
364         if (event.getSource() == saveButton) {
365
366             try(PrintWriter output = new PrintWriter(new FileWriter(fileName,
367                 true)))
368             {
369                 output.println(text.getText());
370             }
371             catch (Exception e) {}
372             frame.dispose();
373         }
374     }
```

Apêndice C

Apêndice C

Código Fonte C.1: Ator Drone Telemetry

```
1 package ptolamy.actor.lib;;
2
3 import java.awt.BorderLayout;
4 import java.awt.Container;
5 import java.awt.Font;
6 import java.awt.event.ActionEvent;
7 import java.awt.event.ActionListener;
8
9 import javax.swing.JButton;
10 import javax.swing.JFrame;
11 import javax.swing.JPanel;
12 import javax.swing.JScrollPane;
13 import javax.swing.JTextArea;
14
15 import java.util.Map;
16 import java.util.SortedSet;
17 import java.util.TreeSet;
18 import java.util.Map.Entry;
19
20 import java.io.BufferedReader;
21 import java.io.BufferedWriter;
22 import java.io.FileReader;
23 import java.io.FileWriter;
24 import java.io.PrintWriter;
25 import java.io.IOException;
26 import java.text.SimpleDateFormat;
27 import java.util.Date;
28
29 import java.sql.*;
30
```

```

31 import ptolemy.actor.TypedAtomicActor;
32 import ptolemy.actor.TypedIOPort;
33 import ptolemy.data.StringToken;
34 import ptolemy.data.DoubleToken;
35 import ptolemy.data.IntToken;
36 import ptolemy.data.expr.Parameter;
37 import ptolemy.data.type.BaseType;
38 import ptolemy.kernel.CompositeEntity;
39 import ptolemy.kernel.util.IllegalActionException;
40 import ptolemy.kernel.util.NameDuplicationException;
41
42 public class DroneTelemetryManager extends TypedAtomicActor {
43
44     private static final long serialVersionUID = 1L;
45
46     public TypedIOPort inId;
47     public TypedIOPort outId;
48
49     public TypedIOPort inIsFly;
50     public TypedIOPort outIsFly;
51
52     public TypedIOPort inTMilli;
53     public TypedIOPort outTMilli;
54
55     public TypedIOPort inBattery;
56     public TypedIOPort outBattery;
57
58     public TypedIOPort inAltitude;
59     public TypedIOPort outAltitude;
60
61     public TypedIOPort inPitch;
62     public TypedIOPort outPitch;
63
64     public TypedIOPort inRoll;
65     public TypedIOPort outRoll;
66
67     public TypedIOPort inYaw;
68     public TypedIOPort outYaw;
69
70     public TypedIOPort inSpeedX;
71     public TypedIOPort outSpeedX;
72
73     public TypedIOPort inSpeedY;
74     public TypedIOPort outSpeedY;
75
76     public TypedIOPort inSpeedZ;
77     public TypedIOPort outSpeedZ;

```

```
78
79     public TypedIOPort inPwmMotor1;
80     public TypedIOPort outPwmMotor1;
81
82     public TypedIOPort inPwmMotor2;
83     public TypedIOPort outPwmMotor2;
84
85     public TypedIOPort inPwmMotor3;
86     public TypedIOPort outPwmMotor3;
87
88     public TypedIOPort inPwmMotor4;
89     public TypedIOPort outPwmMotor4;
90
91     public TypedIOPort inGps;
92     public TypedIOPort outGps;
93
94     //public Parameter actions;
95     //public Parameter knowledgeBase;
96     //public Parameter TimeToRoverP;
97     //public Parameter timeInSecondsP;
98     /*
99     private String id;
100         private String battery;
101         private String temperature;
102         private String sensor1;
103         private String sensor2;
104         private String sensor3;
105         private String gps;
106         private String compass;
107         private String goto;
108         private String rotate;
109         private String activate;
110     */
111     //private int indice;
112     //private float altitude;
113     //private float timeInSeconds;
114     private String id;
115     private String isFly;
116     private String tMilli;
117     private String battery;
118     private String altitude;
119     private String pitch;
120     private String roll;
121     private String yaw;
122     private String speedX;
123     private String speedY;
124     private String speedZ;
```

```

125     private String pwmMotor1;
126     private String pwmMotor2;
127     private String pwmMotor3;
128     private String pwmMotor4;
129     private String gps;
130
131
132     //private DroneKnowledgeBase droneKnowledgeBase;
133
134     public DroneTelemetryManager(CompositeEntity container, String name) throws
        IllegalArgumentException, NameDuplicationException {
135         super(container, name);
136
137         // set values parameter
138         /*
139         actions = new Parameter(this, "actions");
140         actions.setExpression("");
141         actions.setStringMode(true);
142
143         knowledgeBase = new Parameter(this, "knowledgeBase");
144         knowledgeBase.setExpression("");
145         knowledgeBase.setStringMode(true);
146         */
147     //=====
148
149         inId = new TypedIOPort(this, "inId", true, false);
150         inId.setTypeEquals(BaseType.STRING);
151         outId = new TypedIOPort(this, "outId", false, true);
152         outId.setTypeEquals(BaseType.DOUBLE);
153
154         inIsFly = new TypedIOPort(this, "inIsFly", true, false);
155         inIsFly.setTypeEquals(BaseType.STRING);
156         outIsFly = new TypedIOPort(this, "outIsFly", false, true);
157         outIsFly.setTypeEquals(BaseType.DOUBLE);
158
159         inTMilli = new TypedIOPort(this, "inTMilli", true, false);
160         inTMilli.setTypeEquals(BaseType.STRING);
161         outTMilli = new TypedIOPort(this, "outTMilli", false, true);
162         outTMilli.setTypeEquals(BaseType.DOUBLE);
163
164         inBattery = new TypedIOPort(this, "inBattery", true, false);
165         inBattery.setTypeEquals(BaseType.STRING);
166         outBattery = new TypedIOPort(this, "outBattery", false, true);
167         outBattery.setTypeEquals(BaseType.DOUBLE);
168
169         inAltitude = new TypedIOPort(this, "inAltitude", true, false);
170         inAltitude.setTypeEquals(BaseType.STRING);

```

```
171         outAltitude = new TypedIOPort(this , "outAltitude", false , true);
172         outAltitude.setTypeEquals(BaseType.DOUBLE);
173
174         inPitch = new TypedIOPort(this , "inPitch", true , false);
175         inPitch.setTypeEquals(BaseType.STRING);
176         outPitch = new TypedIOPort(this , "outPitch", false , true);
177         outPitch.setTypeEquals(BaseType.DOUBLE);
178
179         inRoll = new TypedIOPort(this , "inRoll", true , false);
180         inRoll.setTypeEquals(BaseType.STRING);
181         outRoll = new TypedIOPort(this , "outRoll", false , true);
182         outRoll.setTypeEquals(BaseType.DOUBLE);
183
184         inYaw = new TypedIOPort(this , "inYaw", true , false);
185         inYaw.setTypeEquals(BaseType.STRING);
186         outYaw = new TypedIOPort(this , "outYaw", false , true);
187         outYaw.setTypeEquals(BaseType.DOUBLE);
188
189         inSpeedX = new TypedIOPort(this , "inSpeedX", true , false);
190         inSpeedX.setTypeEquals(BaseType.STRING);
191         outSpeedX = new TypedIOPort(this , "outSpeedX", false , true);
192         outSpeedX.setTypeEquals(BaseType.DOUBLE);
193
194         inSpeedY = new TypedIOPort(this , "inSpeedY", true , false);
195         inSpeedY.setTypeEquals(BaseType.STRING);
196         outSpeedY = new TypedIOPort(this , "outSpeedY", false , true);
197         outSpeedY.setTypeEquals(BaseType.DOUBLE);
198
199         inSpeedZ = new TypedIOPort(this , "inSpeedZ", true , false);
200         inSpeedZ.setTypeEquals(BaseType.STRING);
201         outSpeedZ = new TypedIOPort(this , "outSpeedZ", false , true);
202         outSpeedZ.setTypeEquals(BaseType.DOUBLE);
203
204         inPwmMotor1 = new TypedIOPort(this , "inPwmMotor1", true , false);
205         inPwmMotor1.setTypeEquals(BaseType.STRING);
206         outPwmMotor1 = new TypedIOPort(this , "outPwmMotor1", false , true);
207         outPwmMotor1.setTypeEquals(BaseType.DOUBLE);
208
209         inPwmMotor2 = new TypedIOPort(this , "inPwmMotor2", true , false);
210         inPwmMotor2.setTypeEquals(BaseType.STRING);
211         outPwmMotor2 = new TypedIOPort(this , "outPwmMotor2", false , true);
212         outPwmMotor2.setTypeEquals(BaseType.DOUBLE);
213
214         inPwmMotor3 = new TypedIOPort(this , "inPwmMotor3", true , false);
215         inPwmMotor3.setTypeEquals(BaseType.STRING);
216         outPwmMotor3 = new TypedIOPort(this , "outPwmMotor3", false , true);
217         outPwmMotor3.setTypeEquals(BaseType.DOUBLE);
```

```

218
219         inPwmMotor4 = new TypedIOPort(this, "inPwmMotor4", true, false);
220         inPwmMotor4.setTypeEquals(BaseType.STRING);
221         outPwmMotor4 = new TypedIOPort(this, "outPwmMotor4", false, true);
222         outPwmMotor4.setTypeEquals(BaseType.DOUBLE);
223
224         inGps = new TypedIOPort(this, "inGps", true, false);
225         inGps.setTypeEquals(BaseType.STRING);
226         outGps = new TypedIOPort(this, "outGps", false, true);
227         outGps.setTypeEquals(BaseType.DOUBLE);
228
229         // =====
230         /*_attachText("_iconDescription", "<svg width=\"30\" height=\"100\">\n"
231             + "<rect x=\"0\" y=\"0\" width=\"30\"
232               height=\"150\" "
233             + "style=\"fill:white\"/>\n"
234             + "Telemetry"
235             + "</svg>\n");*/
236         _attachText("_iconDescription", "<svg width=\"30\" height=\"175\">\n"
237             + "<rect x=\"0\" y=\"0\" width=\"30\"
238               height=\"175\" "
239             + "style=\"fill:white\"/>\n"
240             + "<image x=\"0\" y=\"0\" width=\"30\"
241               height=\"175\" xlink:href=\"telemetry.
242               png\" />"
243             + "</svg>\n");
244     }
245
246     @Override
247     public void fire() throws IllegalArgumentException {
248         super.fire();
249
250         Connection conn = null;
251         Statement stmt = null;
252
253         id = inId.get(0).toString().replace("\\", "");
254         isFly = inIsFly.get(0).toString().replace("\\", "");
255         tMilli = inTMilli.get(0).toString().replace("\\", "");
256         battery = inBattery.get(0).toString().replace("\\", "");
257         altitude = inAltitude.get(0).toString().replace("\\", "");
258         pitch = inPitch.get(0).toString().replace("\\", "");
259         roll = inRoll.get(0).toString().replace("\\", "");
260         yaw = inYaw.get(0).toString().replace("\\", "");
261         speedX = inSpeedX.get(0).toString().replace("\\", "");
262         speedY = inSpeedY.get(0).toString().replace("\\", "");
263         speedZ = inSpeedZ.get(0).toString().replace("\\", "");
264         pwmMotor1 = inPwmMotor1.get(0).toString().replace("\\", "");

```

```

261         pwmMotor2 = inPwmMotor2.get(0).toString().replace("\\", "");
262         pwmMotor3 = inPwmMotor3.get(0).toString().replace("\\", "");
263         pwmMotor4 = inPwmMotor4.get(0).toString().replace("\\", "");
264         gps = inGps.get(0).toString().replace("\\", "");
265
266         System.out.println(id+";"+isFly+";"+tMilli+";"+battery+";"+altitude+";"+
                pitch+";"+
267         roll+";"+yaw+";"+speedX+";"+speedY+";"+speedZ+";"+pwmMotor1+";"+
268         pwmMotor2+";"+pwmMotor3+";"+pwmMotor4+";"+gps);
269
270         String fname = "Voo_"+gps+".csv";
271         System.out.println("ID = "+id);
272
273         if(id.equals("-111111")){
274             System.out.println("INICIO !"+id);
275             try(PrintWriter output = new PrintWriter(new FileWriter(fname,true)
                )){
276                 String Line = "id;isFly;tMilli;battery;altitude;pitch;"+
277                 "roll;yaw;speedX;speedY;speedZ;pwmMotor1;pwmMotor2;
                pwmMotor3;pwmMotor4;"+
278                 "gps;";
279                 output.println(Line);
280                 output.close();
281             } catch (Exception e) {
282                 System.err.println("Exception Error: " + e.getMessage());
283             }
284         } else if(id.equals("-999999")){
285             System.out.println("FIM !"+id);
286             String fnameLog = "Voo_"+gps+".log";
287             new TextEditor(fnameLog);
288         } else {
289             System.out.println("MEIO !"+id);
290             outId.send(0, new DoubleToken(Double.valueOf(id)));
291             outIsFly.send(0, new DoubleToken(Double.valueOf(isFly)));
292             outBattery.send(0, new DoubleToken(Double.valueOf(battery)));
293             outTMilli.send(0, new DoubleToken(Double.valueOf(tMilli)));
294             outAltitude.send(0, new DoubleToken(Double.valueOf(altitude)));
295             outPitch.send(0, new DoubleToken(Double.valueOf(pitch)));
296             outRoll.send(0, new DoubleToken(Double.valueOf(roll)));
297             outYaw.send(0, new DoubleToken(Double.valueOf(yaw)));
298             outSpeedX.send(0, new DoubleToken(Double.valueOf(speedX)));
299             outSpeedY.send(0, new DoubleToken(Double.valueOf(speedY)));
300             outSpeedZ.send(0, new DoubleToken(Double.valueOf(speedZ)));
301             outPwmMotor1.send(0, new DoubleToken(Double.valueOf(pwmMotor1)));
302             outPwmMotor2.send(0, new DoubleToken(Double.valueOf(pwmMotor2)));
303             outPwmMotor3.send(0, new DoubleToken(Double.valueOf(pwmMotor3)));
304             outPwmMotor4.send(0, new DoubleToken(Double.valueOf(pwmMotor4)));

```

```

305         outGps.send(0, new DoubleToken(Double.valueOf(gps)));
306         try(PrintWriter output = new PrintWriter(new FileWriter(fname,true)
307             )){
308             String Line = id+";"+isFly+";"+tMilli+";"+battery+";"+
309                 altitude+";"+pitch+";"+
310                 roll+";"+yaw+";"+speedX+";"+speedY+";"+speedZ+";"+pwmMotor1
311                 +";"+
312                 pwmMotor2+";"+pwmMotor3+";"+pwmMotor4+";"+gps+";";
313             output.println(Line);
314             output.close();
315         }catch (Exception e) {
316             System.err.println("Exception Error: " + e.getMessage());
317         }
318     }
319     @Override
320     public void initialize() throws IllegalArgumentException {
321         super.initialize();
322     }
323 }
324
325 class TextEditor implements ActionListener {
326     // Size of editing text area.
327     private static final int NUM_ROWS = 5;
328     private static final int NUM_COLS = 20;
329
330     // Buttons to save and load files.
331     private JButton saveButton, loadButton;
332
333     // Area where the user does the editing
334     private JTextArea text;
335
336     private String FileName;
337
338     private JFrame frame;
339
340     // Creates the GUI
341     public TextEditor(String FName) {
342         FileName = FName;
343         frame = new JFrame("Drone flight observations");
344         JPanel buttonPanel = new JPanel();
345         saveButton = new JButton("Save Log File");
346         buttonPanel.add(saveButton);
347
348         text = new JTextArea(NUM_ROWS, NUM_COLS);

```

```
349         text.setFont(new Font("System", Font.PLAIN, 24));
350         JScrollPane textScroller = new JScrollPane(text);
351         Container contentPane = frame.getContentPane();
352         contentPane.add(textScroller, BorderLayout.CENTER);
353         contentPane.add(buttonPanel, BorderLayout.NORTH);
354
355         saveButton.addActionListener(this);
356         frame.pack();
357         frame.setVisible(true);
358     }
359
360     // Listener for button clicks that loads the
361     // specified files and puts it in the
362     // editor.
363     public void actionPerformed(ActionEvent event) {
364         if (event.getSource() == saveButton) {
365
366             try(PrintWriter output = new PrintWriter(new FileWriter(fileName,
367                 true)))
368             {
369                 output.println(text.getText());
370             }
371             catch (Exception e) {}
372             frame.dispose();
373         }
374     }
```
