AVALIAÇÃO DO USO DA LINGUAGEM PDDL NO PLANEJAMENTO DE MISSÕES PARA ROBÔS AÉREOS

LUIZ FERNANDO ABRAS CANTONI

AVALIAÇÃO DO USO DA LINGUAGEM PDDL NO PLANEJAMENTO DE MISSÕES PARA ROBÔS AÉREOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Luiz Chaimowicz

Belo Horizonte

Agosto de 2010

Abras Cantoni, Luiz Fernando
C232a Avaliação do uso da linguagem PDDL no planejamento de missões para robôs aéreos / Luiz Fernando Abras Cantoni. — Belo Horizonte, 2010
xxvi, 128 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Luiz Chaimowicz

1. Inteligência Artificial. 2. Planejamento. 3. Robótica. I. Título.

CDU 519.6*82(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Avaliação do uso da linguagem PDDL no planejamento de missões para robôs aéreos

LUIZ FERNANDO ABRAS CANTONI

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LUIZ CHAIMOWICZ - Orientador
Departamento de Ciência da Computação - UFMG

PROF. RICARDO POLEY MARTINS FERREIRA
Departamento de Engenharia Mecânica - DEMEC - UFMG

PROF. MARIO FERNANDO MONTENEGRO CAMPOS
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 30 de agosto de 2010.

Aos meus pais, Ângelo e Tida, e ao Rodrigo Silva, um grande amigo que permitiu e apoiou, incondicionalmente, a realização deste trabalho.

Agradecimentos

Chegar até aqui exigiu esforços não somente da minha parte, mas de todas as pessoas que me acompanharam nesse processo.

Aos meus pais, Ângelo Cantoni e Maria Aparecida (Tida). Foram eles que, em 1985, compraram um Apple IIe e, a partir daí, felizmente, não larguei mais a computação. Mas acima de tudo, agradeço a eles pelos exemplos de caráter e amor.

Ao meu professor e orientador Dr. Luiz Chaimowicz. O Chaimo não só me orientou, mas não me deixou desistir ou me abater perante às tamanhas dificuldades em realizar um trabalho dessa magnitude. Muito Obrigado Chaimo!

À banca examinadora, composta pelos professores Dr. Mário Campos e Dr. Ricardo Poley. Muito Obrigado pelos conselhos e correções!

Para finalizar e não correr o risco de esquecer de alguém, estendo meus agradecimentos a todos aqueles que, direta ou indiretamente, contribuíram para a conclusão deste trabalho.

Resumo

O planejamento de uma missão para robôs aéreos é um processo complexo e que exige, dentre outras coisas, determinar quais os veículos devem ser utilizados e quais as tarefas deverão ser executadas (por veículo) para o cumprimento da missão. Dependendo da instância do problema, esse processo pode se tornar muito complexo para um ser humano. Dessa maneira, o auxílio de ferramentas computacionais se torna muito desejável e, em alguns casos, essencial.

No campo da inteligência artificial (IA), existe uma área denominada de *Planejamento Automático* que desenvolve linguagens e algoritmos de planejamento que permitem a geração de sequências de ações de maneira automática. A *Planning Domain Definition Language* (PDDL) é fruto desse desenvolvimento e é a linguagem oficial das competições de planejamento automático da IA. Uma das suas vantagens é o fato dela ser independente de domínio, podendo, portanto, ser aplicada a uma variedade de problemas de naturezas distintas, desde o tradicional mundo dos blocos a problemas mais complexos como os de logística que envolvem as dimensões tempo e recursos.

O presente trabalho avalia o uso PDDL dentro do planejamento de missões para robôs aéreos. A PDDL é utilizada para gerar automaticamente as sequências de ações necessárias para o cumprimento de duas missões desenvolvidas neste trabalho.

A primeira missão trata especialmente do deslocamento, uma tarefa essencial para a maioria dos veículos aéreos. Nesta missão, são explorados alguns aspectos essenciais que devem ser considerados quando um veículo se desloca, entre eles estão a duração, a velocidade, a distância e o consumo de combustível. Esses aspectos são modelados utilizando a PDDL e, assim, é possível avaliar se a linguagem possui expressividade suficiente para lidar com este tipo de domínio. Além disso, os planos gerados são executados em um arcabouço experimental desenvolvido para este trabalho. Essa execução permite comparar o planejado (em PDDL) com o executado (no simulador de voo). A partir daí, refina-se o modelo, tentando

aproximá-lo da realidade. Já a segunda missão, trata de um cenário hipotético de combate a incêndios florestais. O objetivo é explorar a capacidade que a linguagem possui de gerar planos temporais para múltiplos veículos.

Palavras-chave: Inteligência Artificial, Planejamento automático, PDDL, VAANTs.

Abstract

Unmanned Aerial Vehicle (UAV) mission planning is a complex task that is comprised of, among other things, determining which vehicles should be used and which tasks each vehicle has to perform in order to accomplish the desired outcome. In some cases, this task can be too complex for human operators. Using computational tools for mission planning is desirable or even essential in some cases.

Automated Planning is the area of AI that develops planning methodologies and techniques to automatically generate the sequences of actions necessary to solve problems in different domains. One important tool in this area is the *Planning Domain Definition Language* (PDDL). Being domain independent, this language can be applied to problems of distinct nature, from simple blocks worlds to complex logistics problems where time and resources are fundamental dimensions.

The present work studies the use of PDDL for UAV mission planning. PDDL is used to automatically generate the sequences of actions necessary to perform two distinct missions developed in this work.

The first mission focuses on UAV mobility. We explore some essential aspects such as flight duration, speed, distance and fuel consumption. We then model these elements using PDDL to assess how powerful the language is. We develop an experimental framework based on a flight simulator in order to run the PDDL-generated missions and compare the plans to the simulated reality. This allows us to refine and improve the models and to further explore PDDL and its limitations. The second mission models a forest fire suppression scenario where we explore PDDL's ability to generate temporal plans for multiple UAVs.

Keywords: Artificial Intelligence, Automated Planning, PDDL, UAV.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

ADL *action description language*

AIPS *Artificial Intelligence Planning Systems*

ECP *European Conference on Planning*

FgFS *FlightGear Flight Simulator*

FPS *frames per second*

GPS geo-posicionamento por satélite

HWIL *Hardware-in-the-loop*

HTN *hierarchical task network*

IA inteligência artificial

IAS *indicated airspeed*

ICAPS *International Conference on Automated Planning and Scheduling*

IPC *International Planning Competition*

LPO lógica de primeira ordem

MsFS *Microsoft Flight Simulator®*

PDDL *Planning Domain Definition Language*

SI Sistema Internacional de Unidades

STRIPS *Stanford Research Institute Problem Solver*

TAS *true airspeed*

UAV *unmanned aerial vehicle*

VAANT veículo aéreo autônomo não tripulado

X-Pi *X-Plane Interface*

Sumário

Capítulo 1

Introdução

1.1 Motivação

Um robô aéreo, também conhecido por veículo aéreo autônomo não tripulado (VAANT), pode ser definido como uma aeronave não tripulada que voa de forma parcialmente ou totalmente autônoma e que é utilizada para executar as mais diversas tarefas, seja nos contexos militar ou civil[1].

Entre as tarefas militares amplamente executadas destacam-se as de reconhecimento e vigilância. Por outro lado, existem diversas aplicações civis com objetivos comerciais e de pesquisa. Entre estas, pode-se citar: monitoramento ambiental, de incêndios e da vida selvagem, inspeção de linhas de transmissão de energia, telecomunicações, busca e salvamento, agricultura de precisão e monitoramento do trânsito [Goktogan & Sukkarieh, 2007].

Nos últimos anos, observou-se um crescimento considerável na utilização desses veículos. Um dos fatores que estimulou esse crescimento foram os avanços tecnológicos ocorridos em diversas áreas, tais como: miniaturização de componentes, sensores mais precisos, câmeras de vídeo cada vez menores e com maior poder de resolução, sistemas de geo-posicionamento por satélite (GPS) de baixo custo e comunicação sem fio [Ryan et al., 2004; Valavanis, 2007].

O uso de VAANTs permite maior segurança em operações de risco, já que retira o ser humano de dentro da aeronave. Além disso, o interesse por esse tipo de veículo está também associado aos custos menores de desenvolvimento e operação se comparados aos de aeronaves maiores e tripuladas. Entretanto, pelo fato de não possuir um piloto a bordo, surgem diversos desafios para fazer com

[1]Esse tipo de veículo é conhecido mundialmente pela sigla *unmanned aerial vehicle* (UAV).

que o voo (ou pelo menos uma parte dele) seja realizado de forma autônoma. São necessários sistemas eletrônicos embarcados como piloto automático e sensores, além de algoritmos de controle e navegação utilizados para manter a estabilidade e a trajetória do voo. Nesse sentido, um requisito fundamental para a maioria dos projetos é a realização de simulações computacionais realistas com o intuito de testar e validar esses algoritmos antes dos voos reais.

Contudo, os desafios não residem apenas na fase de construção física da aeronave e no desenvolvimento do *hardware* e *software* que serão embarcados ou utilizados para controlar e monitorar o VAANT à distância. A operação em si é algo que exige pessoas habilitadas e preparadas para lidar com um veículo de dinâmica complexa imerso em um ambiente com várias incertezas. Tarefas como guiar o veículo utilizando um controle remoto, monitorar os parâmetros do voo, obter fotografias de pontos de interesse ou planejar a missão demandam atenção e treinamento. Portanto, a autonomia desses veículos é um fator importante a ser considerado, pois tende a minimizar a necessidade de intervenção humana em sua operação.

Um bom exemplo da complexidade operacional desses veículos é o sistema *Predator* desenvolvido para o exército americano. Utilizado principalmente para executar missões de reconhecimento e vigilância, um sistema totalmente operacional é composto por quatro aeronaves e 82 pessoas em terra executando tarefas que passam pela manutenção, monitoramento, interpretação dos dados e a guiagem das aeronaves [Valdes, 2004].

O planejamento de uma missão para robôs aéreos é um processo complexo e que exige, dentre outras coisas, determinar quais serão os veículos utilizados e qual a sequência de tarefas que deverá ser executada para o cumprimento da missão. Este trabalho fica ainda mais complexo para um ser humano caso múltiplos VAANTs sejam utilizados já que, dependendo da instância da missão, existem várias alternativas para completá-la. Neste contexto, uma das demandas é o desenvolvimento de *softwares* que facilitem esse processo [Nelson, 1995]. Esses sistemas vão desde interfaces gráficas amigáveis que ajudam na guiagem e monitoramento dos VAANTs até algoritmos que determinam, de forma automática, a sequência de tarefas a ser executada pelos veículos.

Em uma missão de combate a um incêncio florestal, por exemplo, as tarefas poderiam ser definidas como reabastecer, deslocar, obter água, lançar água, decolar e pousar. Sendo assim, uma possível sequência de tarefas para extinção de um incêndio seria:

1. Reabastecer;
2. Decolar;
3. Deslocar até o lago;
4. Obter água;
5. Deslocar até o incêndio;
6. Lançar água no foco do incêndio;
7. Deslocar até a base;
8. Pousar.

Note que essas tarefas são consideradas de alto nível, já que não possuem detalhes suficientes para a sua efetiva execução. Assim, uma questão é determinar que o veículo deve se deslocar até o lago e a outra é como ele efetivamente realizará esse trajeto.

Uma das maneiras de se determinar uma sequência de ações de alto nível é utilizar as linguagens e algoritmos de planejamento provenientes da IA, especificamente de uma área denominada de *Planejamento Automático*. Basicamente, o objetivo é produzir uma sequência de ações que deverá ser executada por um ou mais agentes. Sendo assim, dado o estado inicial, um conjunto de operadores (ações que transformam o estado do mundo) e um estado final, o objetivo de um algoritmo de planejamento (denonimado planejador) é encontrar a sequência de ações (plano) que conduz o agente do estado inicial para o estado final. Na sequência exemplificada acima, o estado inicial poderia ser definido como: avião pousado com tanque de combustível e água vazios e o estado final poderia ser definido como: incêndio extinto e avião pousado.

A comunidade de pesquisa da área de planejamento automático desenvolve há alguns anos uma linguagem formal para a descrição de problemas de planejamento denominada *Planning Domain Definition Language* (PDDL) [Ghallab et al., 1998; Fox & Long, 2003]. Com ela, é possível especificar o domínio, constituído pelos tipos de objetos, predicados, funções e ações, além da instância do problema a ser resolvido, constituída pela descrição dos estados inicial e final. Os dois arquivos que representam o domínio e o problema são passados como paramêtros de entrada para um planejador e este, por sua vez, encontra ou não uma sequência de ações a ser executada pelo agente.

Uma das características principais da PDDL é o fato dela ser independente de domínio. Sendo assim, ela pode ser utilizada para resolver os mais diversos tipos de problemas, desde o tradicional mundo dos blocos [Russell & Norvig, 2003] a desafios mais complexos como os de logística onde as dimensões tempo e recursos são essenciais.

Neste contexto, é interessante que sistemas de planejamento de missão para VAANTs, incluam, em seu núcleo, a possibilidade de utilizar as linguagens e algoritmos de planejamento da IA para a determinação automática de quais os veículos serão utilizados na missão e qual a sequência de ações de alto nível que deverá ser executada para que a missão seja cumprida.

1.2 Objetivos

O objetivo princial deste trabalho é avaliar o uso da PDDL dentro do planejamento de missões para robôs aéreos. Basicamente, deseja-se explorar a linguagem por dois aspectos. O primeiro está relacionado com o poder de expressividade da linguagem. Para esse caso, é investigado se a linguagem possui expressividade suficiente para modelar missões que envolvam um ou mais VAANTs. Já o segundo aspecto, possui relação com a capacidade que a linguagem possui de gerar planos temporais e concorrentes para múltiplos veículos.

Para realizar a avaliação proposta, foram desenvolvidas duas missões. A primeira trata especialmente do deslocamento, uma tarefa essencial para a maioria dos veículos aéreos. Nesta missão, são explorados alguns aspectos essenciais que devem ser considerados quando um veículo se desloca, entre eles estão a duração, a velocidade, a distância e o consumo de combustível. Esses aspectos são então modelados utilizando a PDDL e, assim, é possível avaliar se a linguagem possui expressividade suficiente para lidar com este tipo de domínio. Os planos gerados são executados em um arcabouço experimental desenvolvido para este trabalho. Essa execução permite comparar o planejado (em PDDL) com o executado (no simulador de voo). A partir daí, refina-se o modelo, tentando aproximá-lo da realidade.

Já a segunda missão, trata de um cenário hipotético de combate a incêndios florestais. Este cenário é composto por veículos heterogêneos, aeroportos, lagos e incêndios. Nos aeroportos é realizado o reabastecimento dos veículos, nos lagos é obtida a água que será, posteriomente, lançada em um incêndio. O objetivo é explorar a capacidade que a linguagem possui de gerar planos temporais e concorrentes de combate aos incêndios florestais, sendo estes para múltiplos veículos.

Sob a ótica desse trabalho, o algoritmo de planejamento é considerado como uma caixa preta. Dessa maneira, está fora do escopo desenvolver um planejador PDDL ou realizar qualquer modificação em um já existente. Nos experimentos será utilizado o planejador *LPG-td*.

1.3 Contribuições deste trabalho

Com esses objetivos em mente, as principais contribuições deste trabalho são sumarizadas abaixo:

- Avaliação da PDDL dentro do planejamento de missão para VAANTs. Até onde se tem conhecimento, essa é a primeira vez que a PDDL é aplicada dentro de um domínio desse tipo. Sendo assim, este trabalho pode servir como um ponto de partida para pesquisas mais específicas sobre o tema;

- Desenvolvimento de duas missões. Sendo a primeira uma missão que trata do deslocamento do veículo, uma tarefa essencial para a maioria dos robôs aéreos. Os aspectos estudados para o desenvolvimento dessa missão são independentes do sistema de planejamento que esteja sendo utilizado. Assim, podem ser modelados e aplicados utilizando um outro tipo de linguagem que não seja a PDDL;

- Desenvolvimento de um arcabouço experimental onde um dos produtos foi uma biblioteca para comunicação com o simulador de voo X-Plane®, denominada de *X-Plane Interface* (X-Pi). Esta biblioteca permite obter e alterar diversos parâmetros do simulador e pode ser inserida em uma aplicação que seja executada externamente. O código foi publicado no Google Code.

1.4 Organização do Trabalho

Neste capítulo foi apresentada a motivação e os objetivos deste trabalho. Já o Capítulo 2, aborda exclusivamente a área de planejamento automático da IA. Primeiramente é realizada uma introdução sobre a importância das linguagens e dos algoritmos de planejamento dentro da IA. Depois disso, é feita uma breve revisão sobre a Lógica de Primeira Ordem, por ser necessária para realizar a descrição formal de um problema de planejamento. Logo depois, detalham-se os tipos de planejamento (clássico, temporal e hierárquico). Vale ressaltar que o foco deste trabalho é o planejamento temporal e com recursos. As principais linguagens de planejamento, STRIPS e PDDL, são também descritas, como forma de mostrar a importância da representação para a área de planejamento automático. Por fim, realiza-se uma exposição sobre os sistemas de planejamento automático dependentes, independentes e configuráveis em relação ao domínio. Em resumo, esse capítulo aborda a parte da IA necessária para a compreensão deste trabalho.

O Capítulo 3 mostra como a PDDL pode ser aplicada dentro do planejamento de missão para robôs áreos. Para isso, primeiramente detalha-se a sintaxe e a semântica da versão 2.2 da PDDL. Basicamente, o objetivo é mostrar ao leitor como modelar um domínio utilizando a PDDL. A compreensão de algumas partes dos capítulos de metodologia e experimentos dependem das explicações contidas nesse detalhamento. Feito isso, é discutido o que é o planejamento de missão para robôs aéreos. Dentro desta questão, o planejamento é dividido em simbólico e especializado. Basicamente, o simbólico trata das tarefas em um nível mais alto e o especializado no nível da execução. Por fim, é dado um exemplo de como a PDDL pode ser aplicada dentro do planejamento de missão.

O Capítulo 4 detalha a metodologia utilizada para avaliar a PDDL dentro do planejamento de missão. As missões de navegação entre *waypoints* e de combate a incêndios florestais são desenvolvidas neste capítulo. Entretanto, antes é realizada a exposição de alguns conceitos importantes da aviação que são aplicados diretamente nas missões, entre eles estão as unidades de medida comumente utilizadas, as velocidades de uma aeronave e as fases de um voo. Logo depois é mostrada a metodologia proposta para medir a qualidade do plano gerado, comparando o planejado (em PDDL) com o realizado (no simulador de voo). Finalmente, detalha-se a missão de navegação entre *waypoints* e a sua modelagem em PDDL. O mesmo é feito com a missão de combate a incêndios florestais.

O Capítulo 5 apresenta o arcabouço experimental utilizado para realizar a execução (no simulador de voo X-Plane®) dos planos gerados e a comparação com o planejado (em PDDL). Primeiro é realizada uma descrição da importância dos simuladores de voo nas pesquisas que envolvam VAANTs. Logo depois, descreve-se o arcabouço, mostrando cada módulo que foi construído para este trabalho ou utilizado de terceiros. Por fim, é detalhada a estratégia utilizada para realizar o controle de navegação autônomo dos veículos.

Os experimentos realizados com as missões de navegação entre *waypoints* e combate a incêndios florestais são apresentados no Capítulo 6. Finalmente, as conclusões sobre o uso da PDDL dentro do planejamento de missão para robôs aéreos são realizadas no Capítulo 7.

Capítulo 2

Planejamento Automático

Esse capítulo aborda o referencial teórico necessário para compreensão dos conceitos de planejamento automático utilizados ao longo deste trabalho. Em primeiro lugar, na Seção 2.1, é feita uma introdução do surgimento e da importância da área de planejamento automático dentro da IA. Já na Seção 2.2, serão vistos alguns conceitos essenciais da lógica de primeira ordem aplicados na área de planejamento automático. O objetivo é preparar o leitor para uma descrição formal de um problema de planejamento realizada na Seção 2.3.

Na Seção 2.4, serão descritos os tipos de planejamento clássico, temporal e hierárquico. Já as linguagens mais conhecidas (PDDL e STRIPS) para a descrição de problemas de planejamento são abordadas na Seção 2.5. Os sistemas de planejamento automático (dependentes, independentes e configuráveis) são abordados na Seção 2.6.

2.1 Introdução e Histórico

Planejamento Automático é uma área de pesquisa da IA que lida com a geração automática da sequência de ações necessárias para resolver um problema. Sendo assim, dado uma situação inicial, um conjunto de ações e uma situação final desejada, a tarefa de planejamento consiste em determinar uma sequência de ações que soluciona o problema, ou seja, uma sequência que atinja a situação desejada a partir da situação inicial.

O grande desafio desta área é o desenvolvimento de um solucionador genérico, ou seja, um sistema independente de domínio que seja capaz de solucionar uma grande variedade de tipos de problemas. De fato, a construção deste sistema é uma tarefa ambiciosa e de difícil sucesso, já que existem os mais diversos tipos de

problemas no mundo real. Entretanto, é com esse objetivo que a área evoluiu de forma significativa até então.

De uma maneira geral, pode-se dizer que os estudos nessa área começaram em 1959, quando John McCarthy, um dos fundadores da IA, publicou o primeiro artigo que propõe o uso da lógica para a representação de conhecimento em um computador [McCarthy, 1968]. Neste artigo, McCarthy formulou o que é considerado o primeiro problema de planejamento da história [Lifschitz et al., 2000], enunciando-o da seguinte maneira:

> Suponha que eu esteja no escritório da minha casa e deseje ir ao aeroporto. O meu carro está na garagem da minha casa. A solução para esse problema é andar até o carro e dirigir até o aeroporto.

No enunciado acima, identifica-se de forma clara os três principais elementos que compõe *um problema de planejamento*. O primeiro elemento é o estado inicial, definido somente com a proposição *John dentro de casa*. Já o estado final (ou estado objetivo) é o segundo elemento e é descrito com a proposição *John no aeroporto*. O terceiro elemento é o conjunto das ações do problema, são elas: *Andar até o carro* e *Dirigir até o aeroporto*. Estas ações, se devidamente sequenciadas, conseguem conduzir John do estado inicial ao estado final, ou seja, solucionam o problema proposto.

As ações são descritas pelas suas pré-condições e efeitos. Nas pré-condições é indicado o que deve ser verdadeiro antes que a ação possa ser aplicada e nos efeitos é descrito como o estado atual do mundo se altera se a ação for aplicada. Uma possível representação, utilizando lógica proposicional, para o problema acima, pode ser descrita da seguinte maneira:

> **Estado inicial**: *John dentro de casa*.
> **Estado final** : *John no aeroporto*.
>
> **Ação**: *Andar até o carro*:
>
> - **Pré-condição**: *John dentro de casa*.
> - **Efeito**: $\neg$*John dentro de casa* $\wedge$ *John dentro do carro*.
>
> **Ação**: *Dirigir até o aeroporto*:
>
> - **Pré-condição**: *John dentro do carro*.
> - **Efeito**: $\neg$*John dentro do carro* $\wedge$ *John no aeroporto*.

A sequência de ações que soluciona um problema de planejamento é chamada de *plano* e o algoritmo que a produz é denominado *planejador*. O planejador recebe a descrição formal de um problema de planejamento especificado em uma determinada linguagem (PDDL, por exemplo) e usa essa representação explícita do problema para derivar as heurísticas e realizar as inferências e buscas necessárias para solucioná-lo. A Figura 2.1 fornece uma visão geral dos três componentes utilizados na solução de um problema de planejamento.

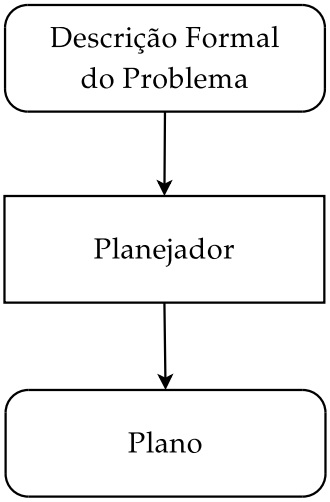

Figura 2.1. Visão geral da integração entre os componentes utilizados na solução de um problema de planejamento.

Existem várias técnicas para resolver um problema de planejamento. As mais básicas são as buscas para frente e para trás no espaço de estados. Um planejador poderia resolver o problema de chegar ao aeroporto utilizando uma busca para frente no espaço de estados. Começando a busca pelo estado inicial, o planejador seleciona de forma não determinística uma ação aplicável ao estado atual do mundo. Neste caso, a única ação aplicável é a *Andar até o carro*, já que apenas essa ação possui em sua pré-condição a proposição *John dentro de casa*. Ao aplicar essa ação, o planejador altera o estado do mundo e John deixa de estar dentro de casa e agora está dentro do carro. Progredindo na busca, o planejador novamente procura (de maneira não determinísitica) uma ação aplicável ao novo estado do mundo *John dentro do carro*. A ação *Dirigir até o aeroporto* é aplicável, pois a sua pré-condição é satisifeita pelo estado do mundo atual. Ao aplicar essa ação no mundo, o planejador verifica que o novo estado do mundo *John no aeroporto* satisfaz o estado objetivo e o problema é resolvido pela sequência: *Andar até o carro* e *Dirigir até o aeroporto*.

De maneira inversa, o problema poderia ser resolvido por uma busca para trás no espaço de estados (*backtracking*). Porém, ao invés de iniciar pelo estado inicial,

o algoritmo inicia a busca pelo estado final. Sendo assim, o algoritmo busca nos efeitos das ações por condições que satisfaçam ao estado do mundo atual. Caso encontre uma ação, as pré-condições são aplicadas como efeitos e são adicionadas como novos objetivos do problema naquele instante do planejamento. A busca termina quando o planejador encontra um estado do mundo que satisfaça o estado inicial do problema.

Segundo Russell & Norvig [2003], um dos principais motivos do interesse pela área de planejamento é o fato dela combinar conceitos de duas grandes áreas da IA: *busca* e *lógica*. Dessa maneira, um planejador pode ser visto tanto como um algoritmo que busca por uma solução quanto um algoritmo que, de maneira construtiva, prova a existência (ou não) de uma solução.

Nos conceitos descritos acima, a separação existente entre a descrição do problema de planejamento e o algoritmo utilizado para resolvê-lo, definem os principais desafios da área de planejamento automático, são eles: a *representação* e a *resolução*. Nesse sentido, o uso de linguagens formais é essencial, pois além de padronizar a forma de representar os problemas, permite que as estratégias utilizadas pelos planejadores para resolver o problema sejam independentes de domínio. Por isso, segundo Russell & Norvig [2003], uma linguagem para a descrição de problemas de planejamento deve ser ao mesmo tempo expressiva e restritiva. Expressiva, pois deve ser possível representar uma larga variedade de diferentes classes de problemas. Restritiva, pois deve permitir que algoritmos operem sobre ela.

O conceito de separação entre a representação de um problema e o método para resolvê-lo surgiu no fim da década de cinquenta com um programa de computador denominado *General Problem Solver*[1] [Newell et al., 1959]. A ideia dos autores era que esse programa fosse um solucionador de problemas gerais, utilizado para provar teoremas, jogar xadrez, entre outros. A sua estrutura geral de controle foi a base para o *Stanford Research Institute Problem Solver* (STRIPS) que se refere tanto à linguagem utilizada para representar um problema de planejamento quanto ao planejador [Fikes & Nilsson, 1971]. A forma de representação de um domínio em STRIPS teve grande influência na área de planejamento e, devido a isso, ele é a base para linguagens modernas como a *action description language* (ADL) e a PDDL [Russell & Norvig, 2003].

[1] Tradução livre: Solucionador geral de problemas.

2.2 Lógica de Primeira Ordem

As principais linguagens de planejamento utilizam como base a lógica de primeira ordem (LPO) para respresentar um domínio. A LPO é um formalismo que estende a lógica proposicional adicionando mais expressividade e permitindo a modelagem de domínios mais complexos. Na lógica proposicional, o mundo é modelado por meio de fatos que são válidos ou não, já na LPO, modela-se o mundo utilizando objetos, relações e funções [Russell & Norvig, 2003]. Devido a isso, essa seção dedica-se a revisar os principais conceitos da LPO aplicados na representação dos problemas de planejamento. A descrição aqui realizada é baseada em Russell & Norvig [2003].

A LPO possui quatro primitivas básicas que são definidas pelo usuário. As constantes, as funções, as variáveis e os predicados. As constantes denotam objetos determinados (ou conhecidos) do mundo. Já as variáveis denotam objetos indeterminados do mundo. As funções mapeiam tuplas de objetos para um outro objeto e os predicados formam as relações entre os objetos e, portanto, mapeiam tuplas de objetos para os valores verdadeiro ou falso. É comum chamar um predicado ou sua negação de *literal*. Convenciona-se aqui que nomes de objetos começam com letra maiúscula seguidos por outras letras ou números e nomes de variáveis começam com letra minúscula seguidos por outras letras ou números.

Um predicado *livre de função* é aquele que não possui como um dos elementos da tupla um símbolo de função. Já um predicado *básico* é aquele em que todas as variáveis (elementos da tupla) foram substituídas pelos símbolos de constantes, em outras palavras, é um predicado que não possui variáveis, apenas constantes[2].

Para exemplificar os conceitos descritos acima, considere que existam três objetos no mundo, representados pelos seguintes símbolos de constante: $Pessoa1$, $Lugar1$ e $Lugar2$. Além disso, considere as seguintes relações (predicados) entre esses objetos: $Em(x, y)$ e $PodeViajar(x, de, para)$. $Em(x, y)$ é um predicado que representa um valor verdadeiro caso a pessoa representada pela variável x esteja no lugar representado pela variável y. Já o predicado $PodeViajar(x, de, para)$ é um predicado que diz se a pessoa x pode viajar do lugar *de* para o lugar *para*. Considere também, o seguinte símbolo de função: $Pai(x)$ que representa o objeto que é o pai da pessoa x.

Um predicado básico pode ser descrito por $Em(Pessoa1, Lugar2)$ ou $PodeViajar(Pessoa1, Lugar1, Lugar2)$. Sendo assim, o predicado $Em(Pessoa1, y)$ não é básico já que a variável y não foi substituída por um símbolo de constante (objeto).

[2]Utiliza-se em inglês o termo *ground literal* ou *ground atom* para denotar um predicado básico.

Já o predicado $Em(Pai(x), y)$ não pode ser considerado livre de função, já que o primeiro parâmetro contém uma função. Um predicado livre de função seria $Em(x, y)$.

2.3 Descrição formal de um problema de planejamento

A partir das definições de LPO fornecidas na seção anterior, é possível descrever formalmente um problema de planejamento. A descrição aqui utilizada foi adaptada de Alford et al. [2009] que, por sua vez, foi baseada em Nau et al. [2004].

Seja L o conjunto de todos os literais livres de função e descritos pela LPO. Um *estado* do mundo é caracterizado por um conjunto de literais básicos extraídos de L. Um problema de planejamento pode ser descrito pela tupla $P = (I, O, F)$. Onde I é o *estado inicial*, F é o *estado final* e O é o conjunto de *operadores*. Cada operador é representado pela tupla $o = (nome(o), precond(o), efeitos(o))$. Onde o elemento $nome(o)$ é o nome do operador e sua lista de argumentos. O elemento $precond(o)$ é formado por um conjunto de literais que formam a condição lógica necessária para que a ação se torne aplicável em um determinado estado. Já o elemento $efeitos(o)$ pode ser descrito pela tupla $efeitos(o) = (adicionados(l), eliminados(l))$ onde cada elemento da tupla representa, respectivamente, o conjunto dos literais que devem ser adicionados e eliminados do estado atual mundo caso a ação seja aplicada.

Uma ação α é a instância de um operador o, ou seja, é um operador com todos os parâmetros de entrada substituídos por constantes. Considere a ação $\alpha = (nome(\alpha), precond(\alpha), efeitos(\alpha))$, se um estado s satisfaz a $precond(\alpha)$ então essa ação é dita *aplicável* no estado s. Caso seja aplicada, então os efeitos da ação produzem um novo estado s', da seguinte maneira:

$$s' = (s - literais\ eliminados\ nos\ efeitos(\alpha) \cup literais\ adicionados\ nos\ efeitos(\alpha))$$

A solução para um problema de planejamento é uma sequência de ações (ou plano) $\pi = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ que, se aplicada a partir do estado incial s_0, produz um estado s_n que satisfaz o estado objetivo.

2.4 Tipos de Planejamento

2.4.1 Planejamento Clássico

O formalismo descrito na Seção 2.3 prevê um ambiente completamente observável, determinístico, estático, finito e discreto. Além disso, o plano é uma sequência totalmente ordenada de ações e o planejamento é realizado de maneira *offline*[3]. O tipo de planejamento que considera essas restrições é denominado de *planejamento clássico*.

Entretanto, apesar dessas limitações, este tipo de planejamento já oferece enormes desafios pois, mesmo problemas de pequeno porte envolvem grandes quantidades de estados. Bylander [1994] mostra que o planejamento clássico é classificado como PSPACE-Completo e que são necessárias várias restrições na representação do domínio para que seja classificado como NP-Completo. Isso demonstra a dificuldade em lidar com esse tipo de problema. Devido a isso, existem pesquisas e publicações constantes que buscam e propõem novas heurísticas que consigam deixar tratável o processo de geração de um plano.

2.4.2 Planejamento Temporal e com Recursos

As dimensões *tempo* e *recursos* são essenciais para modelar problemas do mundo real. No planejamento temporal as ações possuem duração e, por isso, os efeitos não são aplicados de forma instantânea no mundo. Em algumas linguagens, como a PDDL, é possível representar que os efeitos de uma ação acontecem no início e no final de sua duração. Dessa maneira, planos temporais podem explorar a concorrência e o paralelismo na execução das ações. Assim, o plano deixa de ser uma sequência ordenada de ações e passa a ser uma sequência parcialmente ordenada.

Já os recursos são utilizados para expressar quantidades númericas de um determinado item modelado. Por meio dessa característica é possível representar, por exemplo, a quantidade de combustível de uma aeronave. Isso significa que as variáveis de estado armazenam também valores númericos ao invés de apenas valores booleanos, como é o caso do planejamento clássico. A utilização de recursos gera a possibilidade de definir métricas para o plano. Com isso, é possível indicar ao planejador que ele deve tentar encontrar um plano que minimize ou maximize o valor total de um determinado recurso. Como por exemplo, o total de combustível gasto.

[3]No planejamento *offline* todo o planejamento é realizado antes da execução.

No planejamento clássico, a medida da qualidade de um plano é o número de ações que foram necessárias para resolver o problema. Sendo assim, quanto menor for o número de ações melhor é o plano gerado. Já no planejamento temporal, o objetivo é a geração de planos com a menor duração possível ou planos que atendam uma determinada restrição de tempo. Dessa maneira, o número de ações do plano deixa de ser a métrica relevante e isso, consequentemente, pode levar a planos que possuam uma quantidade maior de ações.

2.4.3 Planejamento Hierárquico (HTN)

Planejamento hierárquico, também conhecido por *hierarchical task network* (HTN), é uma metodologia de planejamento que trata o problema de gerar o plano como uma decomposição hierárquica de tarefas. Este tipo de planejamento pode ser visto como uma generalização do planejamento clássico e inclui, além das ações regulares, chamadas nesse caso de *ações primitivas*, um conjunto de *tarefas* ou ações de alto nível. Como no planejamento clássico, o estado do mundo é alterado pela execução das ações primitivas, entretanto, para alcançar estas ações é necessário decompor, de maneira recursiva, tarefas em subtarefas até que o planejador encontre as ações primitivas. Assim, o planejamento HTN consegue lidar melhor com a complexidade inerente ao planejamento clássico evitando explorar partes desnecessárias do espaço de busca.

2.5 Linguagens para a descrição de problemas de planejamento

Como dito na Seção 2.1, as linguagens para a representação dos problemas de planejamento constituem uma parte fundamental no processo da geração dos planos. As seções seguintes detalham as mais conhecidas.

2.5.1 STRIPS

No início dos anos 70 foi desenvolvido o robô Shakey (figura 2.2), o primeiro robô móvel com capacidade de planejar as suas tarefas. Na camada de planejamento deste robô foi implementada uma linguagem e um planejador ambos denominados STRIPS [Fikes & Nilsson, 1971]. O planejamento era realizado para tarefas de navegação e deslocamento de caixas no ambiente. A linguagem será detalhada

nessa seção devido a influência da sua sintaxe e semântica na área de planejamento automático. Os conceitos aqui descritos complementam os já citados na Seção 2.3.

Figura 2.2. O robô Shakey, primeiro robô móvel com capacidade de planejar as suas tarefas.

Para ilustrar o uso de STRIPS na representação de um problema de planejamento, foi criado um domínio parecido com o problema de chegar ao aeroporto descrito na Seção 2.1. O objetivo do problema orginal era chegar ao aeroporto saindo de casa. O objetivo agora é chegar ao aeroporto com dinheiro (também partindo de casa). As ações originais foram substituídas pelas seguintes ações: *EntrarNoCarro*, *SairDoCarro*, *Dirigir* e *SacarDinheiro*. A Figura 2.3 exibe o domínio criado.

As Tabelas 2.1 e 2.2 descrevem cada objeto do mundo e os predicados respectivamente. Conforme pode ser observado na tabela 2.1 e na descrição do problema de planejamento, o domínio foi criado com a intenção de permitir múltiplos aeroportos, bancos, carros e pessoas. Sendo assim, nada impede, por exemplo, a definição da Pessoa 2 ($P2$) e do Carro 2 ($C2$). Na prática, um planejador deve inferir que as ações que envolvam as Pessoas $P1$ e $P2$ podem ser executadas de forma paralela, já que na descrição no domínio não existe nada que impeça isso. Nesse caso, o plano gerado não será uma sequência totalmente ordenada de ações, mas uma sequência parcialmente ordenada.

2.5.1.1 Representação dos Estados

Em STRIPS, o estado do mundo é uma condição lógica representada por uma conjunção de literais positivos, básicos e livres de função descritos por meio da lógica de primeira ordem. STRIPS faz uso da *hipótese do mundo fechado*, ou seja, todo predicado não mencionado em um estado é considerado falso. No estado inicial, por exemplo, não é preciso descrever que a pessoa não está dentro do carro

```
Init(Em(P1,R1) ∧ ForaDoCarro(P1) ∧ Pessoa(P1) ∧ Carro(C1) ∧
  Banco(B1) ∧ Local(A1) ∧ Local(B1) ∧ Local(R1))

Goal(Em(P1,A1) ∧ ComDinheiro(P1) ∧ ForaDoCarro(P1))

Action(EntrarNoCarro(p))
  Precond: ForaDoCarro(p) ∧ Pessoa(p)
  Effect: ¬ForaDoCarro(p) ∧ DentroDoCarro(p)

Action(SairDoCarro(p))
  Precond: DentroDoCarro(p) ∧ Pessoa(p)
  Effect: ¬DentroDoCarro(p) ∧ ForaDoCarro(p)

Action(Dirigir(p, c, de, para))
  Precond: DentroDoCarro(p) ∧ Em(p, de) ∧ Pessoa(p) ∧
           Carro(c) ∧ Local(de) ∧ Local(para)
  Effect: ¬Em(p,de) ∧ Em(p, para)

Action(SacarDinheiro(p, b))
  Precond: Em(p, b) ∧ ForaDoCarro(p) ∧ Pessoa(p) ∧
           Local(b) ∧ Banco(b)
  Effect:  ComDinheiro(p)
```

Figura 2.3. Descrição em STRIPS do problema de chegar ao aerporto com dinheiro. Utiliza-se aqui a mesma notação encontrada em Russell & Norvig [2003].

Objeto	*Descrição*
A1	Aeroporto1.
B1	Banco1.
C1	Carro1.
P1	Pessoa1.
R1	Residência1.

Tabela 2.1. Descrição dos objetos do problema de chegar ao aerporto com dinheiro.

($\neg DentroDoCarro(P1)$), pois como esse predicado não foi mencionado no estado, ele é considerado falso.

Os estados inicial e final e as pré-condições das ações são estados parcialmente especificados e por isso seguem as mesmas regras de representação descritas acima. Um estado s satisfaz a um estado g (parcialmente especificado) se s contém todas as condições em g e possivelmente outras [Russell & Norvig, 2003]. Considere o estado atual do mundo como $Em(P1, Casa) \wedge ForaDoCarro(P1) \wedge Pessoa(P1) \wedge Carro(C1)$.

Objeto	*Descrição*
Em(p,l)	Indica que a pessoa p está no local l.
ComDinheiro(p)	Indica que a pessoa p já sacou dinheiro.
ForaDoCarro(p)	Indica que a pessoa p está do lado de fora do carro.
DentroDoCarro(p)	Indica que a pessoa p está dentro do carro c.
Pessoa(p)	Indica que o parâmetro p é uma pessoa.
Carro(c)	Indica que o parâmetro c é um carro.
Banco(b)	Indica que o parâmetro b é um banco.

Tabela 2.2. Descrição dos predicados do problema de chegar ao aerporto com dinheiro.

Se o parâmetro p da ação $EntrarNoCarro(p)$ for substituído pelo objeto $P1$, então a pré-condição resultante, $ForaDoCarro(P1) \wedge Pessoa(P1)$, é satisfeita pelo estado atual e, portanto, a ação é considerada aplicável nesse estado.

2.5.1.2 Representação das Ações

As ações em STRIPS são especificadas por um nome, uma lista de paramêtros, as pré-condições e os efeitos. Qualquer variável que apareça nas pré-condições e efeitos deve aparecer também na lista de parâmetros. Diferentemente da representação da pré-condição, os efeitos permitem a utilização de predicados negativos na condição lógica (formada por uma conjunção de predicados). Entretanto, note que a possibilidade de inserir um predicado negativo é apenas uma forma mais simplificada de expressar a lista de predicados adicionados (*add*) e eliminados (*del*) da ação. Portanto, a açao $EntrarNoCarro(p)$ poderia ser reescrita da seguinte maneira:

```
Action(EntrarNoCarro(p))
  Precond: ForaDoCarro(p) ∧ Pessoa(p)
  Add: DentroDoCarro(p)
  Del: ForaDoCarro(p)
```

Uma das regras mais importantes de STRIPS é denominada de *Hipótese de STRIPS* e pode ser resumida da seguinte maneira: "*todo predicado não mencionado no efeito de uma ação permanece inalterado*" [Russell & Norvig, 2003]. Para esclarecer essa hipótese, considere s como o estado atual do mundo. Se uma ação for aplicada no estado s e na lista de predicados adicionados existir um predicado P que já exista no estado s, então P não será incluído novamente. Da mesma maneira, se um

predicado $\neg P$ que deva ser eliminado, já não existir em s, ele será ignorado pelo planejador.

Uma ação, como definida na Figura 2.3, é melhor denominada *esquema de ação* ou operador. Formalmente, a ação é uma instância de um operador. Um operador é instanciado quando todos os seus parâmetros (variáveis) são substituídos por objetos do mundo (os símbolos de constante na lógica de primeira ordem)[4]. No problema de chegar ao aeroporo com dinheiro, o esquema de ação $EntrarNoCarro(p)$ pode ser instanciado apenas de uma maneira, já que existe apenas um objeto pessoa no mundo, o objeto $P1$. Já o esquema de ação $Dirigir(p, c, de, para)$, pode ser instanciado de nove maneiras distintas, pois existe um objeto pessoa ($P1$), um objeto carro ($C1$) e três objetos locais ($R1, A1, B1$). Note que não existe nada no esquema de ação *Dirigir* que impeça a pessoa de dirigir para o próprio local de origem. Portanto, é perfeitamente possível a seguinte instância: $Dirigir(P1, C1, A1, A1)$. Uma maneira de evitar esse problema é criar um predicado $Diferente(de, para)$ que deve ser incluído no estado incial e na pré-condição do esquema de ação $Dirigir(p, c, de, para)$. Sendo assim, sempre que *de* for diferente de *para* gera-se um predicado $Diferente(de, para)$, como formalizado na fórmula abaixo:

$$\forall x \forall y \; Local(x) \wedge Local(y) \wedge \neg(x = y) \implies Diferente(x, y)$$

Dessa maneira, um planejador não gerará a ação $Dirigir(P1, C1, A1, A1)$, pois a pré-condição falhará ao testar $Diferente(A1, A1)$, já que esse predicado não existirá no estado atual do mundo. Note que não é possível utilizar o quantificador universal ($\forall$) em STRIPS, cada predicado deve ser escrito um a um.

A linguagem ADL relaxa algumas restrições de STRIPS permitindo a inclusão de efeitos condicionais, disjunção nas pré-condições, suporte a variáveis tipadas, uso de quantificadores existenciais ($\exists$) e universais, predicados negativos nas pré-condições, além de utilizar a *hipótese do mundo aberto* onde literais não mencionados em um estado são desconhecidos. Assim, bastaria adicionar o literal negativo $\neg Em(de, de)$ na pré-condição da ação *Dirigir* para resolver o problema de dirigir para o próprio local de origem. Da mesma maneira, o uso de literais negativos nas pré-condições evitaria o uso de dois predicados para representar o fato da pessoa estar ou não dentro do carro.

A utilização de variáveis tipadas é um avanço interessante da ADL, posteriormente adotada pela PDDL. Note que, na definição do domínio na Figura 2.3, não

[4]Formalmente, a denominação esquema de ação é a mais correta. Entretanto, por clareza, apenas nesse parágrafo o termo será utilizado. Portanto, ao longo do trabalho continuará sendo mencionado o termo ação tanto para o modelo quanto para a instância.

existe uma maneira formal de dizer que o objeto $P1$ é do tipo *Pessoa*. Utiliza-se o predicado $Pessoa(P1)$ para expressar esse fato. A ação *EntrarNoCarro* poderia ser reescrita em ADL da seguinte maneira:

```
Action(EntrarNoCarro(p: Pessoa))
  Precond: ForaDoCarro(p)
  Effect: ¬ForaDoCarro(p) ∧ DentroDoCarro(p)
```

2.5.2 PDDL

Criada em 1998, a PDDL é a linguagem oficial das competições internacionais de planejamento[5] que acontecem de dois em dois anos dentro da principal conferência de planejamento da área, denominada *International Conference on Automated Planning and Scheduling* (ICAPS)[6]. O objetivo principal da linguagem, no contexto das competições, é permitir comparações de desempenho entre os planejadores em diferentes domínios [Ghallab et al., 1998; Long et al., 2000]. A PDDL é o resultado da junção de diversos formalismos da área de planejamento, principalmente o STRIPS e ADL.

Devido ao sucesso obtido nas competições, a linguagem evoluiu de forma significativa e já é utilizada em aplicações reais. Atualmente, existem cinco versões da PDDL. A cada versão novas características e mais expressividade são adicionadas à linguagem, permitindo que uma diversidade maior de problemas sejam especificados. As evoluções são propostas por um grupo de pesquisadores com amplo conhecimento da área. Após a publicação das novas características e da gramática, planejadores são desenvolvidos ou modificados para operarem sobre a nova versão. A tabela 2.3 exibe as versões, o ano de lançamento, a competição, a conferência e qual a publicação que descreve as modificações na linguagem.

Por ser um dos temas centrais deste trabalho, a Seção 3.1 dedica-se a detalhar a versão 2.2 da PDDL.

2.5.2.1 PDDL1.2

A PDDL1.2 foi desenvolvida para a primeira competição de planejamento (IPC-1). Além de STRIPS e ADL, a linguagem é baseada em outros formalismos, como SIPE-2, Prodigy-4.0, UMCP, Unpop e UCPOP [Ghallab et al., 1998]. A primeira versão da PDDL introduziu vários conceitos e formas de expressar um problema de

[5] *International Planning Competition* (IPC).

[6] A ICAPS é o resultado da junção entre as conferências *Artificial Intelligence Planning Systems* (AIPS) e *European Conference on Planning* (ECP) - http://www.icaps-conference.org/

[7] http://ipc.informatik.uni-freiburg.de/PddlExtension

Versão	*Ano*	*Competição*	*Conferência*	*Publicação que descreve a versão*
1.2	1998	IPC-1	AIPS-98	Ghallab et al. [1998]
2.1	2002	IPC-3	AIPS-02	Fox & Long [2003]
2.2	2004	IPC-4	ICAPS-04	Edelkamp & Hoffmann [2004]
3.0	2006	IPC-5	ICAPS-06	Gerevini & Long [2005]
3.1	2008	IPC-6	ICAPS-08	Não existe publicação[7].

Tabela 2.3. As cinco versões da PDDL.

planejamento, alguns dos quais sobrevivem até hoje, outros só foram especificados para esta versão e nem mesmo foram utilizados na competição. Os próprios autores da linguagem, [Ghallab et al., 1998], argumentam que poucos planejadores poderiam lidar com a especificação completa da linguagem. Com esta versão, era possível expressar um domínio puramente STRIPS ou utilizar as características da ADL como efeitos condicionais, quantificadores universais, disjunção nas pré-condições, pré-condições com predicados negativos e entre outras.

2.5.2.2 PDDL2.1

Na PDDL2.1, foram introduzidas características que permitem especificar problemas de planejamento que exigem as dimensões tempo e recursos. A dimensão tempo é especificada pelas *ações durativas*. As ações durativas podem ser definidas como ações que possuem duração, ou seja, ações cujo os efeitos não são aplicados de forma instantânea ao longo da geração do plano. Já os recursos são variáveis de estado númericas. Na versão anterior, PDDL1.2, os efeitos das ações eram aplicados de forma instantânea e, apesar de haver uma maneira de especificar variáveis númericas, foi apenas na PDDL2.1 que isso foi efetivamente formalizado e utilizado nas competições. Além disso, foi a partir da versão 2.1 que foi introduzido o conceito de métricas na PDDL e, por meio delas, é possível minimizar ou maximizar o valor de uma variável númerica. Um dos principais objetivos dos autores ao lançar a versão 2.1 foi permitir a modelagem de domínios mais realistas [Fox & Long, 2003]. A versão foi dividida em cinco níveis, são eles:

- **Nivel 1**: Corresponde à versão anterior (1.2) da PDDL;
- **Nivel 2**: Uso das variáveis de estado númericas;
- **Nivel 3**: Uso de ações durativas discretas;
- **Nivel 4**: Uso de ações durativas contínuas;
- **Nivel 5**: Especificação completa da PDDL2.1, níveis 1,2,3,4.

2.5.2.3 PDDL2.2

Já na PDDL2.2, os autores [Edelkamp & Hoffmann, 2004], adicionaram os conceitos de *predicados derivados* e *eventos exógenos*. Informalmente, os predicados derivados podem ser definidos como regras que alteram o valor dos predicados independente de qualquer ação do domínio, em outras palavras, são predicados que não são alterados pelas ações. Já os eventos exógenos podem ser definidos como eventos incondicionais e determinísticos que alteram o valor de um predicado para verdadeiro ou falso em pontos de tempo conhecidos *a priori* (também independente de qualquer ação).

2.5.2.4 PDDL3.0

Gerevini & Long [2005] desenvolveram a PDDL3.0 para a quinta competição internacional de planejamento (IPC-5). Nessa competição, o foco principal foi a qualidade do plano gerado. Nas edições anteriores, a principal preocupação era em relação ao tempo de processamento gasto para produzir um plano. Para atender essa nova demanda, os autores introduziram as restrições na trajetória de um plano. As restrições são regras aplicadas a possíves ações do plano e estados intermediários alcançados pelo planejador. Para exemplificar, considere que no mundo dos blocos, um bloco classificado como frágil, em nenhum momento do plano, pode ter sob ele um bloco que não seja frágil. Essa regra é declarada como uma restrição e o planejador deve obedecê-la durante todo a trajetória de construção do plano, ou seja, durante todos os estados que serão gerados do estado inicial ao final. As restrições, assim como os objetivos, foram classificadas como fortes e fracas. As fortes devem ser obrigatoriamente obedecidas pelo planejador, já as restrições fracas não precisam ser necessariamente cumpridas. O mesmo vale para os objetivos. Os objetivos fortes devem ser obrigatoriamente atingidos pelo planejador já os fracos são desejáveis mas não obrigatórios.

2.5.2.5 PDDL3.1

Poucas modificações ocorreram da PDDL3.0 para a PDDL3.1 e não existe uma publicação que descreva a nova versão, apenas uma página na Web (ver tabela 2.3). Antes da versão 3.1, as variáveis de estado são declaradas como tuplas de objeto que representam valores booleanos ou númericos. A partir da versão 3.1, existe a possibilidade de declarar tuplas de objetos que representam um outro objeto.

2.6 Sistemas de Planejamento Automático

Nau et al. [2005] classificam os sistemas de planejamento automático em três categorias: *dependentes*, *independentes* e *configuráveis* em relação ao domínio. Para explicar cada categoria deve-se antes realizar a definição de *conhecimento necessário* e *conhecimento específico*.

Dentro da perspectiva da área de planejamento automático, entende-se como conhecimento necessário o conhecimento sem o qual um planejador não conseguiria resolver o problema. Portanto, é o conhecimento básico de um domínio. As ações, predicados e funções são exemplos de conhecimento necessário.

Por sua vez, o conhecimento específico é o tipo de conhecimento declarado de forma explícita e que ajuda o planejador a realizar a sua busca pela solução de maneira mais eficiente. Em outras palavras, são regras de controle baseadas no conhecimento específico do problema e que têm como principal objetivo diminuir o espaço de soluções, reduzindo o tempo de geração do plano e aumentando a qualidade.

O que difere cada categoria mencionada acima é a utilização ou não de regras de controle explícitas seja na descrição formal do problema, no código do planejador ou em ambos. Em PDDL, por exemplo, é permitido especificar apenas o conhecimento necessário para modelar um domínio. Não é permitido o uso de regras de controle que guiem o planejador. As regras devem ser inferidas de maneira automática baseado na descrição do domínio.

Um exemplo de conhecimento específico é a definição explícita de como as ações se conectam umas as outras. Em PDDL não é possível dizer de forma explícita que a ação *Dirigir* deve acontecer depois da ação *LigarCarro*, isso deve ser feito utilizando os predicados que são testados nas pré-condições e alterados nos efeitos das ações. Já com o planejador SHOP2 [Nau et al., 2005], descrever a forma como as ações se conectam é uma parte fundamental da descrição do domínio. O SHOP2 é um planejador baseado na metodologia de planejamento HTN.

2.6.1 Sistemas dependentes de domínio

Um sistema de planejamento dependente de domínio é construído para operar sobre um dominio específico e, portanto, tende a ser bem mais eficaz que os planejadores que são independentes de domínio. Devido a isso, muitas aplicações práticas de planejamento estão inseridas nessa classe. O sucesso desse tipo de abordagem se deve principalmente a utilização de conhecimento específico do problema (ver

Seção 2.6). O problema deste tipo de abordagem é que ela não pode ser aproveitada facilmente para a resolução de outros problemas.

O programa de computador *Bridge Baron*[7] é um exemplo de sistema de planejamento que utiliza conhecimento específico para jogar *Contact Bridge*, um famoso jogo de cartas. O sistema foi motivo de reportagens em jornais como *The New York Times* e *The Washington Post*, pois na época ganhou a competição internacional *Baron Barclay World Bridge Computer Challenge*. O sistema de planejamento é baseado em HTN [Smith et al., 1998].

2.6.2 Sistemas independentes de domínio

Um sistema de planejamento independente de domínio é construído para operar (a princípio) em qualquer tipo de domínio. Um sistema deste tipo deve ser capaz de resolver problemas para a indústria automobilística, determinando a ordem de produção dos veículos, bem como problemas para uma empresa de transporte de cargas, definindo, por exemplo, uma sequência de ações que minimze o tempo total gasto para a entrega das cargas. Deve ser capaz também de atuar na camada de tomada de decisão de um robô, provendo, para este, a capacidade de determinar, de maneira autônoma, a sequência de ações necessárias para cumprir determinada tarefa.

Por isso, a definição de uma linguagem formal para especificar os problemas se torna mais evidente e essencial do que na abordagem dependente de domínio. Sem o conceito de separação entre a representação e a resolução, não seria possível construir um sistema independente de domínio, já que os problemas do mundo real muitas vezes não possuem relações entre si. Resolver um problema de entrega de pacotes para uma transportadora é muito diferente de prover um robô com autonomia suficiente para realizar tarefas no solo de Marte.

Entretanto, soluções independentes de domínio tendem a ter um desempenho bem inferior se comparados a abordagem dependentes de domínio. O principal fator que contribui para a queda de desempenho é a não utilização de conhecimento específico. O exemplo mais bem sucedido de planejamento independente de domínio é a linguagem PDDL e os diversos planejadores desenvolvidos para operar sobre ela.

[7] http://www.bridgebaron.com/

2.6.3 Sistemas de domínio configurável

Sistemas de planejamento de domínio configurável combinam algumas das principais vantagens existentes nas abordagens dependente e independente de domínio. Neste tipo de sistema, a descrição formal do problema é realizada utilizando conhecimento específico do problema, entretanto, o planejador utilizado para resolvê-lo é independente do problema tratado, podendo, portanto, ser utilizado para resolver outros tipos de domínio.

Um dos sistemas de planejamento de domínio configurável mais conhecidos é o SHOP2. Baseado na metodologia HTN, o SHOP2 participou da IPC-3 e por utilizar conhecimento específico venceu vários planejadores PDDL em diferentes tipos de domínios, ganhando alguns prêmios por isso. Note que, como é possível utilizar conhecimento específico do problema, a linguagem deve ser muito mais expressiva do que PDDL, por exemplo. No caso do SHOP2, os domínios são codificados na linguagem LISP. É possível, até mesmo, fazer chamadas a funções externas dentro de uma ação. Uma das desvantagens deste tipo de abordagem é que a codificação do problema fica muito mais complexa se comparada à PDDL.

Outros exemplos de sistemas de planejamento desse tipo são o TALPlanner, TLPlan, SIPE-2, O-PLAN e SHOP [Alford et al., 2009].

2.6.4 Sistemas independentes versus configuráveis

Existe uma divisão na comunidade de planejamento entre aqueles que defendem a abordagem independente de domínio da PDDL e outros que preferem os sistemas de domínio configurável, como é o caso do SHOP2. O primeiro grupo acredita que utilizar conhecimento específico não deve ser permitido, pois é papel do planejador inferir tudo que ele precisa para resolver o problema. Além disso, domínios codificados em PDDL são muito mais simples de serem produzidos e compreendidos do que domínios produzidos em SHOP2. Já o segundo grupo defende que quanto mais conhecimento puder ser codificado melhor, pois diminui de forma significativa o tempo de geração do plano. Apesar dessa divisão, a PDDL é a linguagem padrão e somente planejadores PDDL podem participar das competições internacionais (com exceção da IPC-3).

Não há dúvida que essa maneira conservadora de pensamento dos que defendem a não utilização de regras de controle explícitas na representação do problema produziu muitos avanços no planejamento independente de domínio, já que as pesquisas precisam se concentrar em soluções mais genéricas que atendam uma maior quantidade de domínios. Entretanto, como dito anteriormente, os planejadores

com maior sucesso em aplicações práticas são os que utilizam regras de controle específica.

O fato é que a área como um todo se enriquece com essas abordagens, pois são realizadas diversas pesquisas que procuram implementar em HTN características da PDDL e vice-versa. Praticamente em todas as conferências da ICAPS existem trabalhos que fazem esse intercâmbio. Em Alford et al. [2009], por exemplo, os autores traduzem domínios HTN para PDDL. O objetivo é utilizar os planejadores PDDL para operar em domínios escritos inicialmente em HTN. Já em Armano et al. [2003], os autores propõem uma extensão em PDDL implementando os conceitos da metodologia HTN.

Capítulo 3

A PDDL no Planejamento de Missões

O principal objetivo deste capítulo é demonstrar como a PDDL pode ser utilizada dentro do planejamento de missão para robôs aéreos. Primeiramente, na Seção 3.1, é explicado como são modelados os domínios e as instâncias (estado inicial e final) utilizando a versão 2.2 da PDDL. Feito isso, é explicado na Seção 3.2, o que significa e representa o planejamento de missão para robôs aéreos. Neste ponto, é realizada a diferenciação entre planejadores simbólicos e especializados. Finalmente, a Seção 3.3 tem como objetivo mostrar, a partir de um exemplo, como a PDDL pode ser aplicada dentro do planejamento de missão.

3.1 PDDL2.2

Como explicado na Seção 2.5.2, a PDDL é a linguagem oficial das competições internacionais de planejamento e este foi um fato motivador para sua utilização no presente trabalho. Esta seção dedica-se a explicar de maneira detalhada os conceitos da PDDL2.2, a versão escolhida para o desenvolvimento desta pesquisa. Ao longo do trabalho, tentou-se utilizar versões mais recentes, como a 3.0, entretanto, os planejadores existentes para essa versão não conseguiram gerar planos para os domínios tratados em tempo satisfatório.

Um problema de planejamento em PDDL é descrito por dois arquivos denominados de *arquivo de domínio* e *arquivo de problema*. Ambos são passados como parâmetros para um planejador PDDL e este, por sua vez, gera (se possível) um plano. A Figura 3.1 mostra uma visão geral da resolução de um problema de planejamento em PDDL.

Em um arquivo de domínio deve ser descrito o conhecimento a respeito do problema a ser resolvido. Esse conhecimento deve ser traduzido em tipos de objeto, predicados, funções e ações. Já no arquivo de problema, é descrita uma instância de um problema a ser resolvido. A instância é composta pelos estados inicial e final e por uma métrica (opcional). Sendo assim, um mesmo arquivo de domínio pode ser utilizado por diferentes instâncias (arquivos de problema) e essa é a principal razão da separação de um problema de planejamento em dois arquivos. As Seções, 3.1.1 e 3.1.2, dedicam-se a detalhar os arquivos de domínio e problema. Como será visto, a sintaxe da PDDL é inspirada na linguagem LISP.

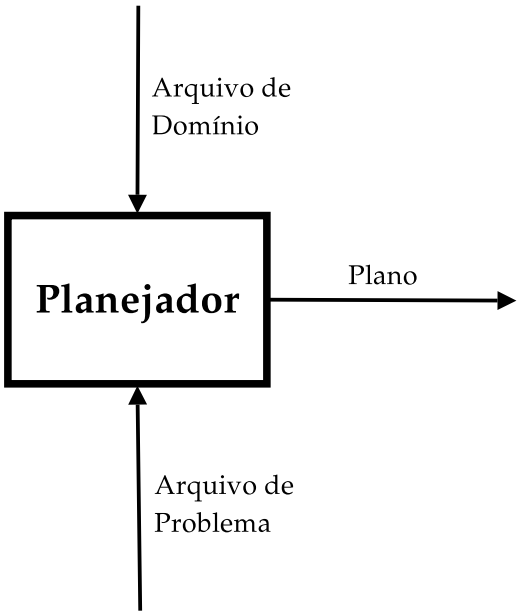

Figura 3.1. Visão geral da resolução de um problema de planejamento em PDDL.

3.1.1 Arquivo de Domínio

A descrição do arquivo de domínio é realizada por meio de um nome e das seguintes seções: *Requisitos*, *Tipos*, *Constantes*, *Predicados*, *Funções* e *Ações*. O nome é definido logo no início do arquivo e as seções são normalmente inseridas na ordem mencionada. Na Figura 3.2, pode ser observada a estrutura de um arquivo de domínio. Caso o domínio seja temporal, a palavra reservada *:action* deve ser substiuída por *:durative-action* indicando, assim, que a ação declarada possui duração.

```
(define (domain NomeDoDominio)
  (:requirements ...)
  (:types ...)
  (:constants ...)
  (:predicates ...)
  (:functions ...)
  (:action act1 ...)
  (:action act2 ...)
  (:action actn ...))
```

Figura 3.2. Estrutura de um arquivo de domínio em PDDL.

3.1.1.1 Requisitos

Na seção de requisitos (*:requirements*), são declaradas as características da PDDL que o domínio utiliza. Dessa maneira, ao interpretar o arquivo de domínio, o planejador poderá saber se ele é capaz ou não de tratar o domínio e realizar o planejamento. Caso um domínio seja temporal, por exemplo, ele deverá incluir como requisito a palavra chave *:durative-actions*. Assim, se um planejador não consegue tratar ações com duração, ele não poderá prosseguir com o planejamento. As palavras reservadas que representam as características devem vir após a palavra chave *:requirements*, conforme pode ser observado na Figura 3.3.

```
(:requirements
  :typing
  :durative-actions)
```

Figura 3.3. Exemplo de um conjunto de requisitos em um arquivo de domínio.

3.1.1.2 Tipos de Objetos

A palavra reservada *:typing*, na seção de requisitos da Figura 3.3, indica que os objetos do domínio devem ser declarados com o seu respectivo tipo. A declaração dos objetos é realizada dentro da seção *:types*, conforme exibido na Figura 3.4. A linguagem permite a criação de uma hierarquia de objetos (herança). No exemplo mostrado na Figura 3.4, os tipos de objetos *Aeroporto* e *Waypoint* herdam diretamente do tipo de objeto *Local*. A declaração de tipos é uma das características que diferencia STRIPS de PDDL. A definição de tipos em STRIPS é realizada de forma não natural utilizando predicados, já em PDDL existe uma maneira formal de fazer essa definição.

Já a declaração hierárquica de tipos é uma caracterísica importante da linguagem, pois permite a definição de predicados e funções genéricas que podem ser aplicados a diferentes tipos de objetos. Além disso, os tipos especificados nas ações podem servir como uma pré-condição, já que uma ação só pode ser instanciada (transformada em uma ação básica, ver Seção 2.2) para um determinado objeto, caso esse objeto herde ou seja do mesmo tipo do objeto definido na ação. Uma ação que receba como parâmetro um tipo de objeto *Aeroporto* não pode ser instanciada para o tipo *Waypoint*, já que os objetos não possuem relação. Por outro lado, se a ação receber como parametro um objeto do tipo *Local* então ela pode ser instanciada para objetos do tipo *Aeroporto* e *Waypoint*.

```
(:types
  UAV - object
  Local - object
  Waypoint - Local
  Aeroporto - Local)
```

Figura 3.4. Exemplo de uma declaração hierárquica de tipos em um arquivo de domínio.

3.1.1.3 Predicados

Os predicados devem ser declarados dentro da seção *:predicates*. Cada predicado é composto por um nome e por uma tupla de objetos que representa um valor booleano. Se o domínio for tipado, então as variáveis dos predicados deverão ser tipadas. A Figura 3.5 exibe dois predicados. Vale notar que o predicado (*noLocal ?uav - UAV ?l - Local*) é genérico, ou seja, pode ser aplicado para toda a hierarquia de objetos do tipo *Local* descrita na Figura 3.4.

```
(:predicates
  ;indica se o aeroporto possui local para reabastecimento
  (possuiReabastecimento ?arpt - Aeroporto)
  ;indica se o uav esta no local l
  (noLocal ?uav - UAV ?l - Local)
)
```

Figura 3.5. Exemplo de uma seção de predicados em um arquivo de domínio.

3.1.1.4 Funções

As funções são declaradas dentro da seção *:functions*. A sintaxe para a declaração das funções é mesma dos predicados. A diferença é na semântica. Enquanto um predicado é uma tupla de objetos que representa um valor booleano, uma função é uma tupla de objeto que representa um valor número. Dessa forma, funções são úteis para declarar variáveis que representam quantidades numéricas (combustível gasto, taxa de consumo de combustível, velocidade, etc). Na Figura 3.6 podem ser observadas cinco funções. Para utilizar essa funcionalidade é preciso declarar na seção de requisitos (ver Seção 3.1.1.1), a palavra chave *:fluents*, indicando ao planejador que esse domínio possui variáveis de estado numéricas.

```
(:functions
  ;velocidade do UAV
  (velocidade ?uav - UAV)
  ;nivel de combustivel atual do uav
  (nivelCombustivel ?uav - UAV)
  ;capacidade do tanque de combustivel do uav
  (capacidadeCombustivel ?uav - UAV)
  ;distancia entre o local1 e o local2
  (distancia ?l1 - Local ?l2 - Local)
  ;total de reabastecimentos realizados por todos os veiculos
  (totalReabastecimento)
)
```

Figura 3.6. Exemplo de uma seção de funções em um arquivo de domínio.

3.1.1.5 Ações

As ações são as entidades que realizam a transição entre os estados. Para isso, fazem uso das declarações de tipos, constantes, predicados e funções. Cada ação deve ser declarada de forma independente (não existe uma seção onde todas devem ser declaradas juntas). Em um domínio atemporal, a descrição de uma ação deve iniciar pela palavra reservada *:action*. Já em um domínio temporal utiliza-se a palavra *:durative-action*.

Uma ação é constituída por um nome, uma lista de parâmetros, as pré-condições e os efeitos. No caso de uma ação durativa, deve-se inserir também o campo relativo à duração da ação. Qualquer variável que apareça no corpo da ação deve ter sido mencionada na lista de parâmetros. A Figura 3.7 mostra uma mesma ação de reabastecimento implementada sem e com duração.

O conceito de domínio temporal foi introduzido a partir da PDDL2.1. A duração pode ser discreta ou contínua, entretanto, a duração contínua é implementada por poucos planejadores devido a sua complexidade. Por esse motivo, apenas as ações com duração discreta serão abordadas neste trabalho.

Dois aspectos diferenciam uma ação durativa de uma ação não durativa. O primeiro, já mencionado acima, é relativo à duração da ação. Para especificar a duração de uma ação deve-se utilizar o campo *:duration*. O segundo aspecto importante de uma ação durativa é que ela deve ser marcada temporalmente nas condições e nos efeitos. Nas condições é preciso informar o que deve ser verdadeiro no início, durante e no fim da aplicação da ação. Já nos efeitos é preciso informar o que será aplicado no estado atual do mundo no início e no fim da ação. As marcações são realizadas utilizando os seguintes termos: *at start*, *over all*, *at end*.

A ação com duração da Figura 3.7 mostra a utilização das anotações temporais. O fato mais curioso desta ação é o predicado *(noLocal ?uav ?arpt)*, pois foi marcado temporalmente no início, durante e no fim da ação. Na prática, isso significa que o objeto *UAV* para o qual a ação está sendo aplicada deverá permanecer no aeroporto até o final da ação. Portanto, o planejador não poderá aplicar outra ação (em paralelo) que mude o objeto *UAV* para outro local que não seja o aeroporto onde ele está sendo reabastecido. Em outras palavras, o predicado deverá permanecer verdadeiro até que a ação seja totalmente aplicada no mundo. Caso fosse aplicado apenas os marcadores *at start* e *over all*, então o planejador poderia aplicar uma outra ação que mudasse o veículo de local antes que a ação de reabastecimento terminasse. O uso incorreto das marcações temporais pode gerar planos insatisfatórios e incorretos.

Nos efeitos de uma ação, é possível utilizar algumas palavras chaves que para alterar o valor numérico representado pelas funções. A palavra *assign* por exemplo, atribui uma quantidade para esse valor. Já a palavra *increase*, como o próprio nome diz, realiza um incremento nesse valor. É possível utilizar aritmética básica ($+ - /*$) nas pré-condições e nos efeitos das ações. A Figura 3.8 mostra um trecho de código que verifica, na pré-condição de uma ação de deslocamento, se o veículo possui combustível suficiente para ir do local 1 para o local 2.

Não é possível realizar cálculos mais complexos como os de raiz quadrada, seno, cosseno e etc. A opção para estes casos é efetuar o cálculo fora da PDDL e entregar o valor já pronto, associando-o a uma função.

```
(:action reabastecer
  :parameters (?uav - UAV ?arpt - Aeroporto)
  :precondition
  (and
    (< (nivelCombustivel ?uav) (capacidade ?uav))
    (possuiReabastecimento ?arpt))
    (noLocal ?uav ?arpt))
  )
  :effect
  (and
    (assign (nivelCombustivel ?uav) (capacidade ?uav))
    (increase (totalReabastecimento) 1)
  )
)

(:durative-action reabastecer
  :parameters (?uav - UAV ?arpt - Aeroporto)
  :duration (= ?duration (tempoParaRebastecer ?uav))
  :condition
  (and
    (at start (< (nivelCombustivel ?uav) (capacidade ?uav)))
    (at start (possuiReabastecimento ?arpt))
    (at start (noLocal ?uav ?arpt))
    (over all (noLocal ?uav ?arpt))
    (at end (noLocal ?uav ?arpt))
  )
  :effect
  (and
    (at end (assign (nivelCombustivel ?uav) (capacidade ?uav)))
    (at end (increase (totalReabastecimento) 1))
  )
)
```

Figura 3.7. Exemplo de uma mesma ação implementada sem e com duração em um arquivo de domínio.

3.1.2 Arquivo de Problema

Como dito anteriormente, um arquivo de problema representa uma instância a ser resolvida. Nesse arquivo, são realizadas as declarações dos objetos e dos estados inicial e final. Os estados inicial e final são compostos por predicados e funções associados a valores booleanos e númericos respectivamente. Na PDDL2.2, a descrição do arquivo de problema é realizada por meio de um nome e pelas seguintes seções: *Objetos*, *Estado Inicial*, *Estado Final* e *Métrica*. Além disso, apesar de não ser obrigatório, é recomendando que seja especificado o domínio referente

```
(at start
  (>= (nivelCombustivel ?uav)
    (*
      (/
        (distancia ?l1 ?l2)
        (velocidade ?uav)
      )
      (taxaConsumoCombustivel ?uav)
    )
  )
)
```

Figura 3.8. Exemplo do uso de aritmética básica em PDDL. Esta condição verifica se o UAV possui combustível suficiente para fazer o trajeto do local l1 ao local l2.

a essa instância. Na Figura 3.9, pode ser observada a estrutura de um arquivo de problema.

```
(define (problem NomeDoProblema) (:domain NomeDoDominio)
  (:objects ...) ; objetos
  (:init ...) ; estado inicial
  (:goal ...) ; estado final
  (:metric ...) ; métrica
)
```

Figura 3.9. Estrutura de um arquivo de problema em PDDL.

3.1.2.1 Objetos

Nessa seção, são instanciados os objetos do problema. Se no domínio os objetos forem tipados então cada objeto declarado nessa seção deve estar associado a um tipo. A Figura 3.10 mostra a declaração de alguns objetos de diferentes tipos.

3.1.2.2 Estado Inicial

No estado inicial, os predicados e funções descritos no domínio são associados a seus valores iniciais (booleanos e númericos respectivamente). Entretanto, nem todos precisam ter essa associação. Predicados que devem começar com valor *falso* não precisam constar no estado inicial já que a PDDL faz uso da hipótese do mundo fechado (ver Seção 2.5.1.1). A Figura 3.11 mostra um exemplo de estado inicial.

```
(:objects
  UAV1 - UAV
  UAV2 - UAV
  WP0 - Waypoint
  WP1 - Waypoint
  SBBH - Aeroporto
)
```

Figura 3.10. Exemplo de uma seção de objetos em um arquivo de problema.

```
(:init
  (= (velocidade UAV1) 100.0)
  (= (taxaConsumoCombustivel UAV1) 35.0)
  (= (nivelCombustivel UAV1) 0.0)
  (= (capacidadeCombustivel UAV1) 315.0)

  (= (velocidade UAV2) 200.0)
  (= (taxaConsumoCombustivel UAV2) 60.0)
  (= (nivelCombustivel UAV2) 200.0)
  (= (capacidadeCombustivel UAV2) 600.0)

  (noLocal UAV1 SBBH)
  (noLocal UAV2 WP1)
  (possuiReabastecimento SBBH)

  (= (distancia WP0 WP1) 4.5)
  (= (distancia WP1 WP0) 4.5)
  (= (distancia SBBH WP0) 10.2)
  (= (distancia WP0 SBBH) 10.2)

  (= (totalReabastecimento) 0)
)
```

Figura 3.11. Exemplo de um estado inicial em um arquivo de problema.

Um recurso interessante da PDDL2.2 é a possibilidade de especificar que certos predicados terão o seu valor alterado por mecanismos que não sejam os efeitos das ações. Existem duas maneiras de fazer isso. Uma é utilizando os predicados derivados (não abordada neste trabalho) e a outra é com os eventos exógenos[1]. Com essa funcionalidade, é possível especificar que em um determinado momento do plano (conhecido *a priori*), um predicado específico se tornará verdadeiro ou falso. Esse predicado, então, poderá ser utilizado nas pré-condições das ações para saber

[1]Conhecidos por *Timed Initial Literals* na PDDL2.2.

se a mesma pode ou não ser aplicada.

Para exemplificar, considere que um determinado voo só pode ser realizado no máximo em 50 min. Para isso, poderia ser inserido na pré-condição de uma ação de deslocamento o predicado (*podeVoar*). Assim, a ação só poderia ser aplicada caso esse predicado estivesse verdadeiro (inserido no estado atual do mundo). Para alterar o valor do predicado, seria especificado no estado inicial do arquivo de problema as seguintes cláusulas:

```
(:init
   ...
   (at 0 (podeVoar))
   (at 50 (not (podeVoar)))
   ...
)
```

Note que é necessário utilizar uma palavra reserveda (*at*) para especificar um evento exógeno. Assim, no momento inicial do plano o predicado se torna verdadeiro e aos 50 min se torna falso.

3.1.2.3 Estado Final

O estado final é o estado objetivo, ou seja, é o estado que o planejador deve tentar atingir a partir do estado inicial. A forma de descrevê-lo é similar a maneira de descrever o estado inicial, ou seja, valores booleanos e númericos devem ser associados aos predicados e funções, compondo, assim, o estado desejado.

No estado final, faz sentido associar um predicado ao valor falso. Considere, por exemplo, que no domínio exista um predicado que indique que o veículo está voando *(voando ?uav - UAV)*. Considere também que ao longo do plano foi aplicada a ação de decolagem e esse predicado foi adicionado ao estado atual do mundo, ou seja, ele está verdadeiro *(voando UAV1)*. Suponha que no estado objetivo o veículo não deva estar voando. A única maneira de especificar esse fato é negar o respectivo predicado *(not (voando UAV1))*. Sendo assim, nos efeitos de uma possível ação de pouso esse estado seria atingido, pois essa ação retiraria o predicado do estado atual do mundo e isso ficaria compatível com o estado que deseja-se atingir.

A Figura 3.12 mostra um exemplo de um estado final. Note que ambos os veículos devem estar no aeroporto com tanque cheio, ou seja, ambos devem ser reabastecidos (pela ação descrita na Figura 3.7). Considerando o estado inicial descrito na Figura 3.11, o veículo *UAV2* deverá se deslocar até o aeroporto *SBBH*

para ser reabastecido. Sendo assim, alguma ação de deslocamento (que não está descrita aqui) deve existir para que o planejador encontre um plano que atenda o estado final.

```
(:goal
  (noLocal UAV1 SBBH)
  (noLocal UAV2 SBBH)
  (= (nivelCombustivel UAV1) (capacidadeCombustivel UAV1))
  (= (nivelCombustivel UAV2) (capacidadeCombustivel UAV2))
)
```

Figura 3.12. Exemplo de um estado final em um arquivo de problema.

3.1.2.4 Métrica

A seção de métrica (opcional) é utilizada no arquivo de problema para especificar alguma função do domínio que o planejador deva tentar minimizar ou maximizar. Com isso, é possível, por exemplo, indicar para o planejador que ele deve tentar minimizar o número de reabastecimentos realizados por todos os veículos. A Figura 3.13 mostra como declarar essa métrica em um arquivo de problema. Na mesma figura, é demonstrado como indicar ao planejador que ele deve tentar minimizar a duração total do plano. Note que, *total-time*, é uma palavra reservada da linguagem. Um arquivo de problema pode ter apenas um campo de métrica, entretanto, é possível combiná-las em uma expressão aritmética.

```
(:metric minimize (totalReabastecimento))

(:metric minimize (total-time))
```

Figura 3.13. Exemplo de métricas em um arquivo de problema.

3.1.3 Planejadores PDDL

Os planejadores são os programas que recebem como parâmetro os arquivos de domínio e problema e tentam encontrar um plano. Geralmente, são criados ou modificados para as competições de planejamento (ver Seção 2.5.2) e a grande maioria é gratuita e de código aberto. Entretanto, nem todos conseguem planejar utilizando todas as características da versão da PDDL para a qual foram contruídos. O planejador *TLP-GP*, por exemplo, consegue lidar com ações durativas, mas não

consegue lidar com variáveis de estado númericas (as funções) [Maris & Régnier, 2008]. Por outro lado, o planejador *LPRPG* consegue lidar com as funções, mas não aceita ações durativas [Coles et al., 2008].

Portanto, como será visto mais adiante, uma das etapas deste trabalho foi selecionar um planejador que conseguisse lidar com funções, ações durativas e tivesse um bom desempenho na geração do plano. Desempenho, neste caso, é a combinação de um plano com qualidade razoável (ver Seção 2.4.2) gerado em um tempo satisfatório.

3.2 Planejamento de missão

Não existe uma definição formal do que é um sistema de planejamento de missão para VAANTs. Alguns dos conceitos aqui descritos foram retirados de Nelson [1995]. Nesse relatório técnico, o autor define quais são os requisitos que um sistema de planejamento de missão deve possuir com o objetivo de ajudar especialistas a planejar e gerenciar missões para múltiplos VAANTs. Apesar do relatório ser direcionado para o uso militar, os conceitos também podem ser utilizados para a área civil.

Segundo Nelson [1995], o processo de planejamento de missão é uma atividade baseada em um mapa e que tem como ponto de partida a solicitação do cumprimento de uma determinada tarefa. Por sua vez, um especialista deve desenvolver um plano que satisfaça os requisitos e as restrições da tarefa a ser cumprida. Geralmente, a tarefa inclui a observação de uma determinada área por meio dos sensores de um ou mais VAANTs e a interpretação dos dados obtidos dentro de um intervalo de tempo determinado. Além dos requisitos específicos da tarefa, existem outros critérios da missão que devem ser observados, como evitar a colisão dos veículos com o chão, evitar que os veículos fiquem sem combustível e obedecer as regulamentações do espaço aéreo.

Dentro do processo de planejar uma missão, os especialistas devem se preocupar com diversas questões, muitas delas são atividades que exigem cálculos complexos e que, por isso, devem ser delegadas para computadores. Alguns exemplos são: planejar a trajetória a ser percorrida considerando as restrições não-holonômicas do veículo e evitando possíveis obstáculos, estimar a cobertura dos sensores para o correto mapeamento do solo, calcular as altitudes ao longo do voo evitando colisão com montanhas, calcular o tempo e a velocidade necessária para cumprir os prazos da missão, calcular a distância entre os veículos e as estações de

solo evitando interrupções nas comunicações [Nelson, 1995].

Além dessas questões, existem outras de mais alto nível como definir quais os tipos e a quantidade de veículos utilizar, verificar se VAANTs em voo podem ser reaproveitados para cumprir a missão sem a necessidade de pouso e reconfiguração e, por último, deve-se determinar a sequência de ações que os veículos devem executar para cumprir a missão. Estas últimas indagações, apesar de não envolverem cálculos matemáticos complexos, são também desafiadoras para um ser humano, principalmente quando a missão é complexa, ou seja, quando existem diferentes possibilidades de completá-la. Uma das maneiras de responder a essas últimas questões é utilizar o planejamento automático proveniente da IA.

3.2.1 Planejadores Simbólicos e Especializados

Conforme visto na seção anterior, planejar uma missão para robôs aéreos envolve questões de diferentes naturezas e que por isso são respondidas por diferentes tipos de planejadores. Para exemplificar, considere que um VAANT deve ir do *waypoint A* para o *waypoint B* e obter fotografias do solo durante cinco minutos. Uma questão é definir que o veículo deve ir do *waypoint A* para o *waypoint B* e a outra é como ele realizará esse trajeto considerando as suas restrições cinemáticas, dinâmicas e os possíveis obstáculos no caminho. Da mesma maneira, uma questão é definir que o veículo deve obter fotos do solo e a outra é definir a orientação do veículo e da camera de tal maneira que melhores imagens possam ser obtidas.

Sendo assim, costuma-se dividir a camada de planejamento de um robô em dois tipos de planejadores, os *planejadores simbólicos* e os *planjeadores especializados* (também conhecidos por planejadores geométricos) [Guitton et al., 2008; Gravot et al., 2003; Alili et al., 2010]. De maneira geral, pode-se dizer que um planejador simbólico responde a questão sobre *quais* ações executar e um planejador especializados responde a questão sobre *como* executar tais ações.

Obviamente, o ideal seria ter um único planejador que conseguisse resolver o problema como um todo, levando em consideração as diferentes necessidades para a sua solução. Porém, infelizmente, esse não é o caso e essa distinção se torna relevante, já que planejadores simbólicos como STRIPS e PDDL não conseguem expressar problemas numericamente complexos como o cálculo da trajetória de um veículo ou a melhor orientação da camera de vídeo. Essas questões são delegadas a planejadores especializados que geralmente são escritos em linguagens que permitem expressar cáclulos matemáticos complexos, como é o caso de Matlab, Java e C++.

Guitton et al. [2008] discutem esse problema e propõe uma arquitetura de planejamento para robôs móveis que combina os dois tipos de planejadores. Neste trabalho, os autores dizem que não é possível planejar em alto nível sem informações dos níveis inferiores e que por isso a integração entre um planejador simbólico e vários planejadores especializados, cada um exercendo um papel, é essencial. Eles utilizam o planejador SHOP2 que por permitir chamadas a funções externas facilita o processo de integração. Vale ressaltar que o SHOP2 permite essa flexibilidade pois tanto o código do planejador quanto o domínio são codificados em LISP, dessa maneira, o poder de expressão aumenta de forma significativa.

Por outro lado, Ilghami & Murdock [2005], propõe um extensão para a PDDL que permite especificar chamadas a funções externas a partir do código PDDL. Os autores argumentam que essa possibilidade é muito importante, já que, geralmente, ao longo do processo de planejamento, são necessárias informações externas para tomada de decisão. Entretanto, essa proposta (ou alguma similiar) ainda não foi incoporada na linguagem. Dessa maneira, não há como interferir no planejamento após o seu início. Deve-se aguardar o planejador retornar (ou não) um plano.

3.3 A PDDL no planejamento de uma missão

Para explicar melhor como a PDDL pode ser utilizada dentro do planejamento de missão, considere que um especialista receba o seguinte pedido: *obter fotografias de pontos de interesse dentro de um intervalo de tempo*. Os pontos de interesse poderiam ser cruzamentos e/ou avenidas de uma grande capital que diariamente enfrenta vários quilômetros de congestionamentos. Cada veículo envolvido na missão deve se deslocar até um dos locais e circulá-lo por algum tempo obtendo fotografias do trânsito. A missão deve ser cumprida dentro de uma janela de tempo específica já que congestionamentos são fenomênos que acontecem durante alguns períodos do dia apenas.

3.3.1 Especificação do Domínio

Um domínio com essa missão poderia ser codificado em PDDL com as ações temporais de decolagem, pouso, deslocamento em linha reta, deslocamento em círculo e obtenção de fotografias. O domínio teria o tipo de objeto VAANT e cada ação seria baseada nesse objeto, permitindo, assim, que o planejador pudesse gerar um plano para múltiplos veículos. O fato das ações serem temporais é um requisito da missão, já que a mesma deve ser completada dentro de uma janela de tempo

determinada. Dessa maneira, como cada ação possui uma duração, a soma das durações de todas as ações (sequenciais) contidas no plano não pode exceder o tempo total estabelecido para a missão. Além disso, essa característica permite que o planejador explore o paralelismo na execução das ações. Na prática isso significa que enquanto um VAANT está circulando e obtendo fotografias de um ponto de interesse outro poderia estar pousando para realizar o reabastecimento.

3.3.2 Especificação da Instância

Uma possível instância para este domínio poderia ser composta por vinte pontos de interesse que devem ser fotografados em (no máximo) 90 min, dois aeroportos onde os veículos poderiam reabastecer e quatro VAANTs, dois de asa fixa e dois helicópteros. Os VAANTs de asa fixa são heterogêneos entre si, ou seja, possuem velocidade e autonomia de voo distintas. Já os helicópteros são homogêneos. Um helicóptero e um avião já estão em voo, entretanto, o helicóptero possui pouca quantidade de combustível e, dependendo da distância que deve ser percorrida, pode ser necessário realizar um reabastecimento.

Para especificar a janela de tempo, poderia ser utilizado o recurso de eventos exógenos da PDDL2.2. Um predicado denominado *(missaoAberta)* poderia ser utilizado na pré-condição de cada ação, indicando que a ação pode ser executada apenas se esse predicado for verdadeiro. No arquivo de problema seria especificado o tempo no qual esse predicado deixaria de ser verdadeiro. Exemplo: *(at 90 (not (missaoAberta)))*. Sendo assim, essa missão poderia durar no máximo noventa unidades de tempo (no caso dessa missão, 90 min).

3.3.3 Geração do plano

O especialista tem a sua disposição uma estação de solo contendo um mapa em duas dimensões da cidade. Neste mapa é exibida a posição atual de cada VAANT, além dos parâmetros de voo de cada veículo (posição, atitude, velocidade, quantidade e consumo de combustível, etc). Depois de definir quais são os pontos e os veículos disponíveis para o cumprimento da tarefa, o especialista solicita ao sistema a geração de um plano. Nesse momento, o planejador PDDL é acionado recebendo como parâmetros de entrada os arquivos que representam o domínio e a instância do problema a ser resolvida.

O plano produzido é a sequência de ações que deve ser seguida pelos veículos para que a missão proposta seja cumprida. Nessa sequência estará especificado

o que cada VAANT deverá executar. Consequentemente, a informação de quais VAANTs foram selecionados para a missão pelo planejador pode ser derivada dessa lista. É interessante notar que o planejador pode decidir utilizar apenas os dois VAANTs já em voo. Isso vai depender diretamente da métrica especificada no problema de planejamento. Caso seja a minimização do total de combustível gasto pelos veículos, então o planejador tentará encontrar uma sequência que utilize a menor quantidade de veículos ou tentará utilizar apenas os que consomem menos combustível. Caso o problema seja de minimização do tempo, então talvez seja mais interessante utilizar uma maior quantidade de veículos e terminar a missão o quanto antes. Todas essas questões possuem relação com a qualidade do plano gerado. Como explicado na seção 2.4.1, gerar um plano é um problema computacionalmente difícil e produzir um plano ótimo é um desafio ainda maior e sem solução em tempo aceitável. Portanto, a utilização de heurísticas é fundamental e, consequentemente, isso também influi diretamente na qualidade do plano gerado.

3.3.4 Considerações

Note que no cenário acima não foi citado como o veículo realizaria o caminho entre os pontos de interesse ou como a camêra e a aeronave seriam posicionadas para obter melhores fotos de uma determinada avenida. Como visto nas seções anteriores, isso não é papel da PDDL, já que ela realiza o planejamento simbólico. As tarefas mencionadas seriam designadas a planejadores especializados.

Planejadores especializados são frutos de pesquisas constantes na área de robótica aérea. Um exemplo são os algoritmos para a geração de trajetória que levam em consideração as restrições não-holonômicas do VAANT e possíveis obstáculos. Estes algoritmos são utilizados para encontrar uma sequência de pontos no espaço que, se seguida pelo veículo, o conduz de uma configuração inicial P_i para uma configuração final P_f. Cada configuração é a pose do robô no espaço, sendo esta composta pela posição x, y, z e a orientação θ do veículo. Exemplos podem ser encontrados em Alves Neto & Campos [2009a]; Alves Neto et al. [2010].

Por outro lado, mesmo com todos os avanços ocorridos na área de planejamento automático, principalmente no planejamento independente de domínio e de domínio configurável, ainda existem poucos trabalhos que aplicam os planejadores simbólicos dentro do planejamento de missões para robôs aéreos (nos moldes descritos na seção 3.2). No caso específico da PDDL, não foram encontrados artigos que a tenham explorado para tal fim.

Um exemplo da utilização do planejamento simbólico para robôs aéreos pode

ser encontrada no projeto COMETS. O objetivo principal deste projeto é a integração de múltiplos VAANTs heterogêneos (helicópteros e dirigíveis) para cumprir missões de vigilância, monitoramento de incêndios, mapeamento e buscas [Ollero et al., 2004]. As pesquisas realizadas neste projeto geraram diversos artigos científicos e em um deles, [Gancet et al., 2005], foi explorado o planejamento simbólico. Nele é proposto uma arquitetura de planejamento e controle para múltiplos veículos aéreos heterogêneos. O tema central é a autonomia dos veículos. Os autores dividem os VAANTs em três categorias. Os que são *controlados diretamente* por uma pessoa. Os que possuem a *autonomia operacional*, ou seja, veículos que apenas seguem um plano recebido da estação de solo e não exibem capacidade de decisão autônoma e, por último, os veículos que possuem a *autonomia decisional*, ou seja, veículos que são dotados com a capacidade de gerar o próprio plano.

Existem cinco tipos de ações que podem ser executadas pelos veículos, são elas: decolar, pousar, deslocar, agir, esperar. A ação agir pode ser definida como a obtenção de fotos ou uma longa vigilância a ser realizada em um determinado local. Para a camada decisional (planejamento simbólico), foi selecionado o planejador SHOP2 e para o planejamento de trajetória utiliza-se um planejador especializado. A interface entre o planejador simbólico e o planejador especializado é realizado através de uma ação especial no SHOP2. Essa ação faz uma chamada externa a uma função que realiza os cálculos de trajetória e devolve para o planejador simbólico dados como a duração, custo e a lista de *waypoints* que conduz o veículo até o local desejado. A partir desses dados, o planejador simbólico continua o planejamento até encontrar o plano a ser seguido pelo veículo.

Capítulo 4

Metodologia

4.1 Introdução

Neste capítulo, será detalhada a metodologia que foi concebida para avaliar utilização da PDDL dentro do planejamento de missão para robôs aéreos.

A Seção 4.2, dedica-se a explicar alguns conceitos importantes da aviação que serão aplicados diretamente no desenvolvimento das missões e, consequentemente, na modelagem dos domínios PDDL. Entre esses conceitos estão as unidades de medida comumente utilizadas, as diferenças entre a velocidade indicada e a velocidade verdadeira de uma aeronave, as fases mais comuns em um voo e os cálculos utilizados para descobrir a distância e direção entre duas coordenadas geográficas.

A primeira missão desenvolvida é denominada de *Missão de Navegação entre waypoints*. Esta missão, descrita na Seção 4.4, modela a tarefa de deslocamento e explora alguns aspectos que são essenciais quando se deseja realizar um planejamento de missão. Os planos gerados em PDDL são executados em um simulador de voo. Com os dados coletados na simulação, utiliza-se a metodologia descrita na Seção 4.3 que permite medir a qualidade do plano gerado, comparando o planejado (em PDDL) com o realizado (no simulador). A partir dessa comparação, o modelo é refinado de maneira incremental até que se chegue a um domínio PDDL que, dentro das condições e limitações impostas, reflita, da melhor maneira possível, a realidade. Realizando esses passos, é possível avaliar o poder de expressividade da PDDL dentro do planejamento de missão, já que muito das características e dos cálculos envolvidos devem ser expressos na linguagem.

Já a segunda missão, detalhada na Seção 4.5, é denominade de *Missão de combate a incêndios florestais* e possui, além da ação de deslocamento desenvolvida na missão anterior, ações específicas para o combate a incêndios florestais. O objetivo é

avaliar a capacidade que a linguagem e os planejadores possuem de gerar planos temporais para múltiplos veículos. Neste caso, os planos não são executados no simulador e, por isso, a qualidade não é medida com a metodologia utilizada na primeira missão. Para esse caso, aplicam-se as métricas comumente utilizadas pela comunidade de planejamento da IA.

Para planos temporais, a qualidade do plano é geralmente medida utilizando a duração ou uma métrica específica. Se for a duração, então planos de menor duração são melhores, mesmo que, para isso, uma quantidade maior de ações devam ser executadas. Da mesma maneira, se uma métrica específica for utilizada, por exemplo, a quantidade de combustível consumida pelos veículos, então, planos que tenham uma duração maior são preferíveis se estes minimizam esta métrica.

4.2 Alguns conceitos de aviação

4.2.1 Unidades de medida

Este trabalho utiliza unidades de medida para expressar as grandezas distância, altitude, velocidade e massa que não são adotadas pelo Sistema Internacional de Unidades (SI). Entretanto, foram escolhidas por serem comumente utilizadas na aviação. A Tabela 4.1 exibe essas unidades.

Unidade	*Descrição*	*Grandeza*	*Conversões*
nm	milha náutica	Distância	1 nm = 1,852 km
ft	pé	Altitude	1 ft = 0,3048 m
knot	nó	Velocidade	1 knot = 1 nm/h
lb	libra	Massa	1 lb = 0,45359237 kg

Tabela 4.1. Unidades de medida utilizadas nesse trabalho.

4.2.2 Atmosfera padrão internacional

Todos os modelos contruídos e os experimentos realizados neste trabalho consideraram as condições especificadas na atmosfera padrão internacional (ISA)[1]. A ISA é uma atmosfera modelo utilizada largamente na aviação para a calibração de equipamentos como altímetros e velocímetros (baseados no tubo de *pitot*). Além

[1] *International Standard Atmosphere* (ISA)

disso, segundo Homa [1998], ela é utilizada para comparar os desempenho de diferentes aviões e padronizar os critérios para essas comparações.

Basicamente, o modelo dita como a pressão, temperatura e densidade do ar variam de acordo com a mudança da altitude. A especificação da ISA é complexa e extensa. Os principais parâmetros medidos no nível do mar podem ser observados abaixo (ventos e umidade são inexistentes) [Sonnemaker, 1999]:

Pressão: 1013.25 hPa
Densidade: 1.225 kg/m^3
Temperatura: 15°C

A temperatura decresce com o aumento da altitude numa taxa constante de $-6,5$°C para cada 1.000 m (ou $-1,98$°C para cada 1.000 ft). Esse comportamento é observado de 0 ft (nível do mar) até 36.098 ft (altitude que define o término da camada troposfera e o início da tropopausa) [Sonnemaker, 1999].

Sendo assim, a seguinte função pode ser utilizada para saber a temperatura em um determinado nível de voo dentro da troposfera [Shelquist, 2010]:

$$T(h) = T_0 - 1.98 \times \frac{h}{1.000}. \tag{4.1}$$

Onde T_0 é a temperatura no nível do mar (15°C) e h é a altitude (em ft) que se deseja saber a temperatura. Abaixo está um exemplo da aplicação da Equação 4.1 para a altitude de 10.000 ft:

$$T(10.000) = 15 - 1.98 \times \tfrac{10.000}{1.000} = \ -4.8°\text{C}.$$

4.2.3 Velocidades de uma aeronave

Uma aeronave possui dois tipos de velocidade, a *velocidade vertical* e a *velocidade horizontal*.

A velocidade vertical representa a taxa em que uma aeronave está ganhando ou perdendo altitude. Geralmente é expressa em ft/min. Assim, um avião que possui uma velocidade vertical de 500 ft/min está ganhando, a cada minuto, 500 ft de altitude. Se a velocidade vertical for negativa, então a aeronave está perdendo altitude. Uma taxa nula indica que o avião não está perdendo nem ganhando altitude. Esta velocidade é muito importante em um voo e o instrumento que a mede (chamado de variômetro ou indicador de velocidade vertical) é um dos mais importantes e básicos em uma aeronave.

Já a velocidade horizontal é geralmente expressa utilizando a unidade knot. Como visto anteriormente, 1 knot equivale a 1 nm/h. Se o fator vento não for considerado, pode-se dizer que, basicamente, existem dois tipos de velocidade horizontal em uma aeronave, são elas: *indicated airspeed* (IAS) e *true airspeed* (TAS)[2]. Será visto mais adiante que utilizar a velocidade correta no planejamento, aumenta de maneira significativa a qualidade do plano gerado.

A IAS é lida diretamente no velocímetro localizado no painel do avião e a TAS é a velocidade real que o veículo está voando em relação ao solo, caso não seja considerado o fator vento. A IAS decresce com o aumento da altitude enquanto a TAS aumenta. A única situação em que as duas velocidades são iguais acontece quando o avião está voando em condições ISA e no nível do mar (ver 4.2.2) [Homa, 1998]. Isso acontece, pois o sistema que mede a velocidade é calibrado nessas condições.

Este comportamento da IAS é reflexo da maneira pela qual essa velocidade é medida. Utiliza-se um equipamento baseado em pressão (tubo de *pitot*) para obtê-la. Como a pressão decresce com o aumento da altitude, então é natural que o sistema meça uma velocidade cada vez menor. Por outro lado, a TAS aumenta, pois quanto maior a altitude, menor é a densidade do ar, menor é o arrasto produzido e, consequentemente, maior é a velocidade real do veículo. Na prática, isso significa que uma aeronave voando a 5000 ft está voando com uma velocidade real maior do que a indicada pelo velocímetro.

A TAS não é indicada no painel do avião e para obtê-la, aplica-se uma fórmula cujo o objetivo é corrigir a IAS baseado na altitude do voo, na própria IAS e em um fator percentual constante. A seguinte equação representa essa fórmula:

$$TAS(h, IAS) = IAS + (IAS \times 2\% \times \frac{h}{1.000}). \tag{4.2}$$

Onde h é a altitude. Utilizar um fator de correção de 2% indica que a cada 1000 ft (de incremento na altitude), a TAS fica maior do que a IAS em 2%. Esse valor percentual foi calculado baseado na atmosfera ISA e é adotado como um padrão na aviação. Entretanto, será visto no capítulo de experimentos, que um fator de $1,5\%$ se ajustou melhor à realidade do presente trabalho.

Sendo assim, um avião voando a 5000 ft com uma IAS de 100 knots implica em uma TAS de 110 knots, conforme pode ser observado no cálculo abaixo (aplicação direta da Equação 4.2):

$$TAS(5.000, 100) = 100 + (100 \times 2\% \times \tfrac{5.000}{1.000}) = 110 \text{ knots.}$$

[2]A IAS é a velocidade indicada e a TAS é a velocidade verdadeira ou aerodinâmica.

Se o fator vento for considerado, então uma terceira velocidade denominada de *velocidade em relação ao solo* (GS)[3] deve ser considerada. Essa velocidade é obtida a partir da TAS aplicando a correção para o efeito dos ventos. Como neste trabalho a ação do vento não foi considerada, então utilizar essa velocidade é desnecessário.

A IAS é uma velocidade muito importante para os pilotos, pois é com ela que são especificadas diversas velocidades críticas de uma aeronave. Além disso, a configuração de um piloto automático para seguir determinada velocidade é realizada com base na IAS e, como será visto na Seção 5.3 do Capítulo 5, este trabalho faz uso do piloto automático para controlar os aviões de maneira autônoma.

4.2.4 Fases de um voo

Normalmente, um voo é dividido em fases, as principais são: voo de cruzeiro, voo de subida e voo de descida. Isso fica evidente em voos comerciais, já que os pilotos realizam o planejamento pensando principalmente nessas três fases, pois são as que consomem a maior parte do tempo, da distânca e, consequentemente, do combustível. Outros exemplos de fases são a decolagem e o pouso.

Geralmente, uma aernoave realiza cada fase com velocidades distintas. Na aeronave Cessna 172SP, por exemplo, o voo de subida é normalmente realizado a uma velocidade (IAS) de 75 knots, já o voo de cruzeiro e descida são ambos realizados a uma IAS de 100 knots. Obviamente, o consumo de combustível também muda em cada fase. No voo de subida, a tendência é que mais combustível seja gasto, já que o motor deve fornecer mais potência para a aeronave vencer a força da gravidade. Por outro lado, no voo de descida, gasta-se menos combustível, pois a ação da gravidade é utilizada como força motora. Nesse caso, o motor é configurado para trabalhar em um ritmo mais lento e econômico.

A Figura 4.1 mostra as três principais fases de um voo. Saber a distância, o tempo e o combustível necessários para realizar cada fase é essencial para um piloto realizar um planejamento de qualidade.

Este trabalho faz uso das fases de voo tanto no planejamento quanto na execução do plano. Devido a isso, os tópicos abaixo dedicam-se a explicar como são realizados os cálculos para determinar a distância, o tempo e o combustível necessários no voo de subida e descida. No voo de cruzeiro o interesse está apenas na duração e no combustível, já que a distância (conforme será visto adiante) é dada.

[3]GS é a sigla em inglês para *ground speed.*

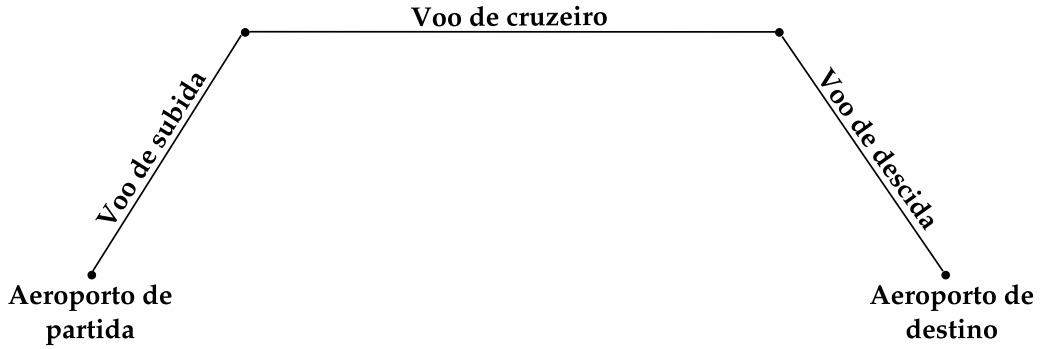

Figura 4.1. As três principais fases de um voo.

4.2.4.1 Cálculos para a fase de voo de cruzeiro

Um voo de cruzeiro é geralmente realizado com velocidade e altitude constantes. Considere que uma aeronave está voando a 8.000 ft com IAS constante de 120 knots e que a distância a ser percorrida é de 20 nm. Considere ainda que essa aeronave consome aproximadamente 30 lb/h de combustível em regime de voo de cruzeiro. A primeira etapa do cálculo consiste em aplicar a Equação 4.2 para descobrir qual é a TAS para o nível de voo dado. A partir desse valor, descobre-se o tempo e, por último, o combustível necessário. Os cálculos podem ser observados abaixo.

(1) Cálculo da TAS: $TAS(8.000 \text{ ft}, 120 \text{ knots}) = 120 + (120 \times 2\% \times \frac{8.000}{1.000}) = 139$ knots,

(2) Cálculo do tempo: $t = 20 \text{ nm} \div 139 \text{ knots} = 0,14 \text{ h}$ ou $8,63$ min,

(3) Cálculo do combustível: $c = 0,14 \text{ h} \times 30 \text{ lb/h} = 4,2$ lb.

Portanto, a aeronave gastaria $8,63$ min para percorrer 20 nm voando a 8.000 ft com velocidade constante (TAS) de 139 knots. Nesse cenário, $4,2$ lb de combustível seriam consumidos.

4.2.4.2 Cálculos para a fase de voo de subida

Caso a velocidade horizontal e vertical sejam mantidas constantes durante um voo de subida, então é possível saber qual será o tempo e o combustível necessários além da distância percorrida durante a subida. Considere, por exemplo, que um avião está voando a 2.000 ft e deva subir até a altitude de 10.000 ft. Durante a subida, será empregada uma velocidade horizontal constante de 75 knots (IAS) e uma velocidade vertical constante de 500 ft/min. Além disso, o veículo consome em média 45 lb/h de combustível em regime de voo de subida.

O primeiro passo é determinar qual o tempo será gasto na subida. Considere a altitude atual como *alt1* e a altitude desejada como *alt2*. A diferença entre a *alt1* e a *alt2* é chamada de ΔAlt. Sendo assim, o tempo necessário para subir da *alt1* para a *alt2* pode ser encontrado utilizando a seguinte fórmula:

$$t = \frac{\Delta Alt}{VV}. \tag{4.3}$$

Onde VV é a velocidade vertical que será empregada. Nesse caso, 500 ft/min. Instanciando a fórmula para os valores do problema exemplificado, descobre-se que o tempo gasto na subida é de 16 min, conforme pode ser observado abaixo:

$$t = \frac{(10.000 - 2.000)\ \text{ft}}{500\ \text{ft/min}} = 16\ \text{min ou}\ 0,27\ \text{h}.$$

A partir do cálculo do tempo, pode-se encontrar diretamente o combustível (c) necessário para realizar a subida.

$$c = 0,27\ \text{h} \times 45\ \text{lb/h} = 12\ \text{lb}.$$

Ainda com o tempo e a velocidade horizontal (já dada anteriormente), é possível calcular a distância percorrida. A aeronave empregará uma velocidade horizontal constante (IAS) de 75 knots, entretanto, como foi visto anteriormente, a velocidade que o avião está realmente voando (se desconsiderado o fator vento) é a TAS. Assim, para o cálculo ficar mais aderente a realidade é necessário converter a IAS para a TAS utilizando a Equação 4.2. Porém, essa função considera a conversão para uma dada altitude e, nesse caso, o avião não manterá uma altitude constante, já que, durante o trajeto, deverá subir até atingir a altitude desejada.

Para resolver essa questão, é comum utilizar uma média aritmética simples da altitude e considerá-la como a entrada para a correção da TAS. Assim, a TAS poderia ser calculada substituindo na equação a variável h pela média da altitude $\overline{alt} = (alt1 + alt2)/2$. Para esse exemplo, a média vale 6.000 ft. Utilizando esse valor, a TAS calculada é de 84 knots, como pode ser observado abaixo:

$$TAS(6.000, 75) = 75 + (75 \times 2\% \times \tfrac{6.000}{1.000}) = 84\ \text{knots}.$$

Finalmente, pode-se calcular a distância percorrida durante a subida. Note que o tempo está expresso em minutos e deve antes ser convertido para horas, para que assim, seja possível multiplicar pela velocidade calculada no passo anterior.

$$d = 84\ \text{knots} \times 0,27\ \text{h} = 22,68\ \text{nm}.$$

Portanto, o veículo percorrerá 22,68 nm durante 16 min para subir de 2.000 ft até 10.000 ft. Durante o trajeto, consumirá 12 lb de combustível.

4.2.4.3 Cálculos para a fase de voo de descida

Para saber o tempo e a distância durante a fase de voo de descida, o cálculo é exatamente o mesmo. Para exemplificar, considere que a mesma aeronave, agora com com altitude de 10.000 ft deseja chegar na altitude de 4.000 ft. A velocidade vertical e horizontal na descida são, respectivamente, 100 knots (IAS) e -700 ft/min e o consumo de combustível em regime de voo de descida é de 20 lb/h. Nesse cenário, a aeronave gastará 8,57 min para descer até os 4.000 ft, percorrendo uma distância de 15,96 nm e consumindo 2,8 lb de combustível. Os cálculos podem ser observados abaixo.

(1) Cálculo do tempo: $t = \frac{(4.000-10.000)\ \text{ft}}{-700\ \text{ft/min}} = 8,57$ min ou $0,14$ h,

(2) Cálculo do combustível: $c = 0,14\ \text{h} \times 20\ \text{lb/h} = 2,8\ \text{lb}$,

(3) Cálculo da TAS: $TAS(7.000, 100) = 100 + (100 \times 2\% \times \frac{7.000}{1.000}) = 114$ knots,

(4) Cálculo da distância: $d = 114\ \text{knots} \times 0,14\ \text{h} = 15,96$ nm.

4.2.4.4 Notas sobre o consumo de combustível

O consumo de combustível é uma variável muito sensível a diversos parâmetros. Além das condições atmosféricas, a altitude e o peso do avião também influem diretamente. Portanto, criar um modelo para o comportamento dessa característica não é algo trivial. Note que, no caso do voo de cruzeiro, como é um regime de voo mais controlado (sem variação da altitude), fica um pouco mais simples definir um valor que represente a taxa de consumo de combustível. Entretanto, nos voos de subida e descida, a altitude da aeronave varia de maneira constante e, consequentemente, a taxa de consumo também. Geralmente, os fabricantes e os pilotos utilizam um valor conservador para essa variável, considerando a atmosfera padrão. Essa será a alternativa adotada por esse trabalho, como será visto mais adiante no modelo e nos experimentos.

4.2.5 Cálculos da distância e direção entre dois *waypoints*

Aqui é detalhado como é calculada a distância e a direção entre dois *waypoints*. A distância é importante tanto no planejamento quanto na execução. O valor calculado é passado para o arquivo de problema por meio da função *(distance ?l1 - Location ?l2*

- *Location)*. Já a direção, como será explicado mais a frente, não é utilizada dentro da PDDL, somente na execução do plano.

4.2.5.1 Distância

A distância entre dois *waypoints* é calculada por uma fórmula chamada *Haversine* e pode ser encontrada em Sinnott [1984] *apud* Veness [2010]. Esta fórmula encontra a menor distância entre dois pontos na superfície da Terra. Os pontos são descritos por suas respectivas latitudes e longitudes. Além disso, a fórmula leva em consideração o raio médio da Terra.

Considere os *waypoints* A e B descritos por meio das seguintes tuplas respectivamente: $A = \langle lat_1, lon_1 \rangle$ e $B = \langle lat_2, lon_2 \rangle$. Para encontrar a distância (d) aplica-se a seguinte sequência de cáclulo:

(1): $\Delta Lat = lat_2 - lat_1$,
(2): $\Delta Lon = lon_2 - lon_1$,
(3): $a = \sin^2(\Delta Lat/2) + \cos(lat_1) \times \cos(lat_2) \times \sin^2(\Delta Lon/2)$,
(4): $c = 2 \times atan2(\sqrt{a}, \sqrt{1-a})$,
(5): $d = R \times c$.

Onde R é o raio médio da Terra, geralmente considerado como 6371 km (ou 3440.06 nm). As funções trigonométricas geralmente aceitam os valores apenas em radianos. Portanto, pode ser necessário realizar a conversão de graus para radianos antes de utilizá-las.

4.2.5.2 Direção

Para determinar a direção entre dois *waypoints*, utilizou-se o cálculo encontrado em Veness [2010]. Considerando os *waypoints* A e B especificados no item anterior, a direção (θ) de A para B pode ser calculada utilizando a seguinte sequência de cálculos:

(1): $y = \sin(\Delta Lon) \times \cos(lat_2)$,
(2): $x = \cos(lat_1) \times \sin(lat_2) - \sin(lat_1) \times cos(lat_2) \times \cos(\Delta Lon)$,
(3): $\theta = atan2(y, x)$.

A função $atan2$ retorna um valor no intervalo de $(-\pi$ a $\pi]$. Portanto, é necessário converter essa valor para graus e normalizá-lo no intervalo de 0° a 360°.

4.3 Qualidade do plano

A palavra planejamento é utilizada em diversos contextos. Entretanto, deve-se notar que, independentemente da situação onde é aplicada, geralmente ela se refere ao ato de tentar estimar, a partir de dados históricos, utilizando um modelo ou ambos, o que será realizado no futuro.

Uma empresa, ao realizar o planejamento financeiro para o ano seguinte, tenta prever, a partir do seu histórico, dos novos contratos e dos investimentos necessários para próximo ano, quais serão as suas receitas e os seus gastos. Essa previsão é utilizada para criar estratégias de atuação ao longo do período. Já um piloto, ao descrever o seu plano de voo, está formalizando o que ele espera que será realizado do aeroporto de origem ao aeroporto de destino. Isso inclui (mas não se limita) às aerovias, velocidades e altitudes utilizadas. O planejamento realizado por um planejador PDDL segue os mesmos princípios, pois, a partir do modelo descrito no arquivo de domínio e dos estados inicial e final, ele deve determinar a sequência de ações a ser executada para resolver o problema.

Em todas as situações descritas acima, é necessário verificar se o modelo descrito se aproxima da realidade. Em outras palavras, é fundamental comparar o planejado com o efetivamente realizado. Esse tipo de comparação permite refinar o modelo, calibrando-o com as características da realidade que precisam ser melhor detalhadas. Entretanto, deve-se ter em mente que nem todas as características do problema real podem ser modeladas em um computador devido a limitações de processamento e memória. Por isso, todo modelo é uma abstração da realidade e um dos grandes desafios é diminuir as diferenças entre ambos.

Como será visto adiante, a primeira etapa foi composta por diversas comparações entre o planejado (em PDDL) e o realizado (no simulador). Para realizar essa comparação utilizou-se a metodologia descrita a seguir.

Seja $\psi = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ o conjunto de todas as ações disponíveis no domínio. Cada ação desse conjunto, por sua vez, pode ser descrita pela seguinte tupla: $\alpha_i = \langle \phi_1, \phi_2, \ldots, \phi_n \rangle$, onde cada elemento desta tupla representa um parâmetro que precisa ser comparado.

Considere ainda $\pi_p = \langle \alpha_1, \alpha_2, \ldots, \alpha_m \rangle$ como sendo o plano gerado em PDDL e $\pi_r = \langle \alpha_1, \alpha_2, \ldots, \alpha_m \rangle$ o mesmo plano, porém, executado no simulador de voo.

Uma das maneiras de realizar essa comparação é utilizando a seguinte fórmula de diferença, baseada em Oliveira [2010]:

$$diferenca_{ij} = \left(\frac{\pi_r^i(j)}{\pi_p^i(j)} - 1\right) \times 100.$$

O índice i representa uma mesma ação nos conjuntos π_p e π_r, porém, instanciada de maneira distinta. O índice j representa um deteminado parâmetro da respectiva ação i. A ação i no conjunto π_p possui os parâmetros instanciados com os valores planejados em PDDL. Já ação i no conjunto π_r possui os parâmetros instanciados com os valores realizados no simulador de voo. Portanto, a $diferenca_{ij}$, representa a diferença percentual entre o realizado e o planejado para um determinado parâmetro em uma determinada ação.

Portanto, com esta fórmula, é possível comparar, de maneira direta, o que foi realizado com o que foi planejado. Um percentual negativo significa que o valor realizado foi menor que o valor planejado e um percentual positivo o contrário. Um exemplo do uso da fórmula pode ser observado na Tabela 4.2. Nessa tabela, está descrito um plano com quatro ações executadas na sequência especificada na primeira coluna. O parâmetro comparado é o consumo de combustível (medido em libras). As três primeiras ações tiveram o consumo de combustível realizado menor que o planejado. Já na quarta ação, gastou-se mais combustível do que o esperado.

Seq.	*Ação*	*Planejado (lb)*	*Realizado (lb)*	*Diferença (%)*
1	Subir	4,31	3,45	-19,95
2	Descer	1,12	0,90	-19,64
3	Subir	4,85	3,77	-22,27
4	Descer	1,26	1,59	26,19

Tabela 4.2. Exemplo da aplicação da fórmula de diferença entre o planejado e o realizado para um plano com quatro ações. O parâmetro comparado é o consumo de combustível medido em libras (lb).

Note que, para este trabalho, o interesse está em avaliar o modelo implementado em PDDL e não o planejador. Devido a isso, essa métrica se difere das normalmente utilizadas para avaliar os planejadores, como o número de ações e a duração do plano (ver Seção 2.4.2).

4.4 Missão de Navegação entre *waypoints*

Uma das tarefas fundamentais dos robôs aéreos é o deslocamento. Veículos de asa fixa devem estar em constante movimento, caso contrário não conseguem permanecer voando. Já helicópteros, apesar de conseguirem pairar no ar, têm na tarefa de deslocamento a mais fundamental. Sendo assim, torna-se necessário modelar esta tarefa da maneira mais fiel possível, pois, a partir dela, é possível

construir missões mais complexas que possuam ações específicas do problema a ser tratado.

Para isso, a primeira missão desenvolvida foi a de navegação entre *waypoints*. Nesta missão, o veículo deve visitar um conjunto de *waypoints*, sendo que, cada um, descrito por uma latitude, longitude e altitude, deve ser visitado uma única vez. O objetivo da missão é modelar a ação de deslocamento em PDDL o mais próximo possível da realidade, ou seja, uma ação onde a diferença entre o planejado e o realizado seja a menor possível. A Figura 4.2 mostra uma visão geral da missão. Nesta instância, o veículo iniciou a missão no WP0 e deve terminar no WP5, passando em sequência pelos *waypoints* 1,2,3 e 4.

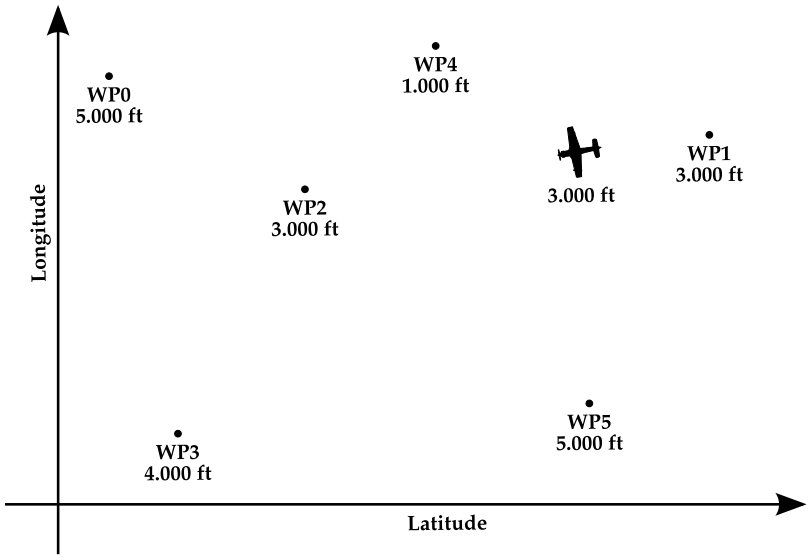

Figura 4.2. Visão geral da missão de navegação entre *waypoints*.

Como forma de guiar melhor o trabalho e os experimentos, foram definidas quatro caracteríticas a serem modeladas: *velocidade*, *distância*, *altitude* e *consumo de combustível*. Entende-se que modelar essas características de uma maneira fiel dimimui a diferença entre o planejado e o realizado. Além disso, elas podem ser consideradas características essenciais no deslocamento de um veículo aéreo. Cada domínio modelado procurava explorar uma dessas características. No caso da velocidade, por exemplo, foram desenvolvidos dois domínios. O primeiro considerou a IAS como a velocidade planejada, já o segundo considerou a TAS.

A abordagem utilizada foi iterativa e incremental. Um determinado domínio era desenvolvido e experimentado no simulador. Os dados coletados na simulação eram utilizados para comparar o planejado com o realizado. Caso houvesse muita divergência, um novo domínio era construído tentando modelar com mais detalhes

os parâmetros que obtiveram maior diferença. Os parâmetros utilizados para a comparação podem ser observados na Tabela 4.3, seguidos pela respectiva unidade de medida e uma descrição. Cada parâmetro foi medido por ação e a metodologia utilizada para compará-los foi descrita na Seção 4.3.

Parâmetro	*Unidade*	*Descrição*
Duração	s	Duração da ação.
Combustível	lb	Total de combustível gasto.
Distância	nm	Distância percorrida.
Velocidade	knots	Velocidade do veículo.
Taxa de Consumo de Combustível	lb/h	

Tabela 4.3. Parâmetros (e suas respectivas unidades de medida) utilizados na comparação entre o planejado e o realizado na missão de deslocamento entre *waypoints*. Cada parâmetro é medido por ação.

No total, foram desenvolvidos seis domínios PDDL para essa missão. Cada um foi incluído em um grupo de acordo com a característica que está sendo explorada. Na Tabela 4.4, podem ser visualizados todos os domínios criados. Por ter sido utilizada uma abordagem incremental, domínios novos eram baseados em domínios anteriores (com exceção do primeiro). Sendo assim, ao final, o domínio C2 contemplava todas as características desenvolvidas em todos os outros domínios, tornando-se, assim, o domínio que reflete de maneira mais fiel a tarefa de deslocamento entre *waypoints*.

4.4.1 Modelagem em PDDL

4.4.1.1 Tipos de Objeto

Foram modelados três tipos de objeto, como pode ser observado na Tabela 4.5. O objeto *Location* foi criado para permitir que, em uma missão mais específica, fosse possível definir outros tipos de locais como aeroportos, lagos e etc.

4.4.1.2 Predicados

Como visto na Seção 3.1.1.3, os predicados são as variáveis de estado booleanas em PDDL. Para esta missão foram definidos três predicados, como pode ser observado na Tabela 4.6.

Grupo	*Domínio*	*Característica Modelada*	*Descrição*
A	A1	Velocidade	Considera a IAS como a velocidade planejada.
A	A2	Velocidade	Considera a TAS como a velocidade planejada.
B	B1	Distância	Não considera um fator para ajustar a distância planejada.
B	B2	Distância	Considera um fator para ajustar a distância planejada.
C	C1	Altitude	Voo de subida, cruzeiro e descida planejados pela mesma ação.
C	C2	Altitude	Ações de subida e descida planejadas com mais detalhes.

Tabela 4.4. Lista de domínios PDDL modelados para a missão de navegação entre *waypoints*.

Objeto	*Tipo*	*Descrição*
UAV	object	Representa um veículo.
Location	object	Representa um local abstrato.
Waypoint	Location	Representa um waypoint que deve ser vistado pelo UAV (herda de Location).

Tabela 4.5. Tipos de objetos definidos para a missão de deslocamento entre *waypoints*.

Predicado	*Descrição*
(atLocation ?uav - UAV ?l - Location)	Indica que o uav está no local l.
(visited ?uav - UAV ?w - Waypoint)	Indica que o ?uav visitou o *waypoint* ?w.
(reachable ?l1 - Location ?l2 - Location)	Indica que o local ?l2 é alcançável a partir do local ?l1.

Tabela 4.6. Predicados definidos para a missão de navegação entre *waypoints*.

4.4.1.3 Funções

As funções são as variáveis de estado númericas em PDDL (ver Seção 3.1.1.4) e são essenciais para qualquer domínio que envolva robôs aéreos, pois é por meio delas que são modelados diversos parâmetros como a velocidade do veículo, a distância entre dois *waypoints* e etc.

A Tabela 4.7 exibe as funções que foram modeladas. Note que apenas três foram utilizadas em todos os domínios. Algumas como as funções *cruiseSpeed*, *climbSpeed* e *descentSpeed* só aparecem no domínio C2 (utilizado para modelar a característica altitude). Essas funções substituem a função (*speed ?uav*) usada nos outros domínios. O objetivo, nesse caso, foi refinar o modelo, considerando velocidades distintas para cada fase de voo (cruzeiro, subida e descida) que podem ocorrer em uma ação de deslocamento.

4.4.1.4 Ações

Os domínios A1,A2,B1,B2 e C1 utilizam apenas uma ação para representar o deslocamento do veículo. A Figura 4.3 exibe a ação de deslocamento encontrada no domínio A1, as partes mais importantes da ação estão comentadas na própria figura. O que muda para os outros domínios é a forma como é calculada a duração e o consumo de combustível. Já o domínio C2 utiliza três ações para representar o deslocamento, uma para cada fase de voo (cruzeiro, subida e descida), mais detalhes sobre este domínio serão vistos mais adiante.

```
(:durative-action go
  :parameters (?uav - UAV ?l1 - Waypoint ?l2 - Waypoint)
  ; t = (d/v)*60 (convertido de horas para minutos)
  :duration (= ?duration (* (/ (distance ?l1 ?l2) (speed ?uav)) 60))
  :condition
  (and
    ; veículo não pode ter visitado o local l2
    (at start (not (visited ?uav ?l2)))
    ; veículo deve estar no local l1
    (at start (atLocation ?uav ?l1))
    ; local l2 deve ser alcançável a partir do l1
    (at start (reachable ?l1 ?l2))
    ; o veículo deve possuir combustível suficiente
    ; para se deslocar do l1 para o l2
    (at start (>= (fuelLevel ?uav)
      (* (/ (distance ?l1 ?l2) (speed ?uav)) (fuelFlow ?uav))))
  )
  :effect
  (and
    ; no início da aplicação da ação considera-se que
    ; o veículo já não está no local l1
    (at start (not (atLocation ?uav ?l1)))
    ; todo o combustível que será utilizado
    ; é consumido no início da aplicação da ação
    (at start (decrease (fuelLevel ?uav)
      (* (/ (distance ?l1 ?l2) (speed ?uav)) (fuelFlow ?uav))))
    ; no final o veículo está no local l2
    (at end (atLocation ?uav ?l2))
    ; no final o veículo visitou o local l2
    (at end (visited ?uav ?l2))
  )
)
```

Figura 4.3. Ação de deslocamento encontrada no domínio A1.

Função	*Domínios*	*Descrição*
(distance ?l1 - Location ?l2 - Location)	Todos	Distância entre os locais l1 e l2.
(fuelLevel ?uav - UAV)	Todos	Nível de combustível atual do veículo.
(fuelCapacity ?uav - UAV)	Todos	Capacidade do tanque de combustível do veículo.
(speed ?uav - UAV)	A1,A2,B1,B2,C1	Representa a velocidade do veículo.
(fuelFlow ?uav - UAV)	A1,A2,B1,B2,C1	Taxa de consumo de combustível do veículo.
(altitude ?l - Location)	A2,B1,B2,C1,C2	Representa a altitude do local l.
(tasFactor)	A2,B1,B2,C1,C2	Fator de correção para a velocidade.
(distanceFactor ?uav - UAV)	B2,C1,C2	Fator de ajuste para a distância planejada.
(cruiseSpeed ?uav – UAV)	C2	Velocidade de cruzeiro do veículo.
(climbSpeed ?uav – UAV)	C2	Velocidade de subida do veículo.
(descentSpeed ?uav – UAV)	C2	Velocidade de descida do veículo.
(cruiseFF ?uav - UAV)	C2	Taxa de consumo de combustível do veículo em voo de cruzeiro.
(climbFF ?uav - UAV)	C2	Taxa de consumo de combustível do veículo em voo de subida.
(descentFF ?uav - UAV)	C2	Taxa de consumo de combustível do veículo em voo de descida.
(distanceAux ?uav - UAV)	C2	Distância percorrida durante as fases de voo de subida ou descida.
(climbRate ?uav - UAV)	C2	Velocidade vertical do veículo na subida.
(descentRate ?uav - UAV)	C2	Velocidade vertical do veículo na descida.

Tabela 4.7. Funções utilizadas em todos os domínios PDDL que representam a missão de navegação entre *waypoints*.

4.4.2 Característica Velocidade

Diversos parâmetros como a duração e consumo de combustível são diretamente dependentes da velocidade do veículo. Sendo assim, torna-se essencial modelar essa característica da maneira mais real possível. Para isso, os domínios A1 e A2 foram construídos. O primeiro domínio, A1, utilizou a IAS como a velocidade planejada e o segundo, A2, foi adaptado a partir do primeiro para planejar utilizando a TAS. A qualidade do plano gerado com a TAS foi muito superior a do plano gerado com a IAS. Isso poderá ser observado mais adiante no capítulo de experimentos.

O cálculo do combustível gasto na ação de deslocamento (em PDDL) considerando a IAS e a TAS podem ser observados respectivamente nas Figuras 4.4 e 4.5. Em ambos os casos, a IAS é a velocidade associada à função (*speed ?uav*). A TAS, portanto, é calculada pelo planejador dinamicamente (no momento do planejamento) levando em consideração a IAS, a altitude do *waypoint* e o fator de correção (*tasFactor*). Como pode ser observado, a fórmula para encontrar a TAS é um pouco diferente da explicada em 4.2.3, porém, o resultado produzido é o mesmo. Optou-se por essa formulação por facilitar a implementação em PDDL.

Note que, para calcular a TAS, é necessário saber a altitude que o veículo está voando. Devido a isso, no domínio A2, foi inserida uma função chamada de (*altitude ?l - Location*). Esta função representa a altitude de um determinado local, consequentemente, de um *waypoint*.

```
(*
  (/
    (distance ?l1 ?l2)
    (speed ?uav)
  )
  (fuelFlow ?uav)
)
```

Figura 4.4. Código em PDDL para o cálculo do combustível gasto. Este código utiliza a IAS para saber quanto combustível foi gasto.

4.4.3 Característica Distância

Após finalizar a característica velocidade, iniciou-se o processo de modelagem da distância planejada. Como visto anteriormente (4.2.5), a distância planejada leva em consideração o menor caminho entre dois *waypoints*. O problema é que as restrições não-holonômicas de um veículo (principalmente de asa fixa), não permitem que ele

```
(*
  (/
    (distance ?l1 ?l2)
    (* (speed ?uav) (+ 1 (* (altitude ?l1) (tasFactor))));TAS
  )
  (fuelFlow ?uav)
)
```

Figura 4.5. Código em PDDL para o cálculo do combustível gasto. Este código utiliza a TAS para saber quanto combustível foi gasto.

siga esse mesmo caminho no momento da execução da ação. Consequentemente, a duração da ação, a própria distância e o consumo de combustível planejados ficarão bem diferentes do realizado e isso pode comprometer de maneira significativa a qualidade do plano gerado.

A Figura 4.6 mostra a diferença entre as distâncias planejada e realizada por um veículo com restrições não-holonômicas. Nesse exemplo, o veículo atinge o *waypoint A* com direção de 225° e chega ao *waypoint B* com direção de 75°. Quanto mais próxima de 180° estiver a diferença entre a direção incial e final, pior será a qualidade do plano gerado. Se a navegação fosse iniciada com a direção de 100°, então essa diferença seria praticamente inexistente e, assim, a distância planejada seria praticamente igual a distância realizada.

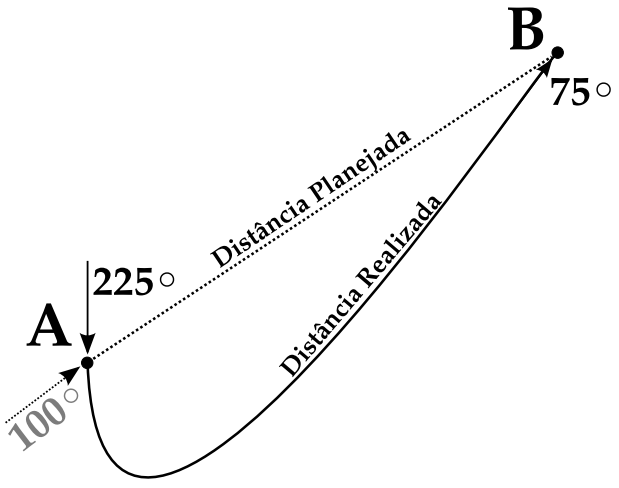

Figura 4.6. Diferença entre as distâncias planejada e realizada.

Vale ressaltar que a direção do veículo é uma variável que não é levada em consideração no momento planejamento. O planejador apenas decide que a ação de ir do *waypoint A* ao *B* será aplicada baseado na distância, velocidade do veículo e

no consumo de combustível. A direção a ser seguida é decidida pelo controlador no momento da execução do plano.

O motivo de não levar em consideração a direção no planejamento é que os cálculos envolvidos são complexos e a linguagem não possui expressividade para isso. Por isso, uma possível solução seria utilizar um algoritmo de planejamento especializado para a geração da trajetória. Exemplos podem ser encontrados em Alves Neto & Campos [2009a,b]. Este tipo de algoritmo recebe como entrada as poses[4] inicial e final do veículo e tenta encontrar uma trajetória que as conecte levando em consideração as restrições não-holônomicas do veículo. Nesse caso, a distância raramente será o menor caminho entre os *waypoints*. Entretanto, a PDDL dificulta o uso desse tipo de abordagem, pois não permite chamadas a funções externas e isso impossibilita o processo de integração com um planejador especializado.

Uma outra alternativa, adotada por esse trabalho, é utilizar um fator de ajuste que tem por objetivo aumentar a distância planejada. Por exemplo: considere que o fator de ajuste para um determinado avião foi calculado como sendo 2 nm. Se a distância entre o *waypoint A* e *B* for de 5 nm, então a aplicação do fator aumentará essa distância para 7 nm. O fator foi obtido empiricamente[5] medindo a distância que um determinado veículo percorre para ficar na direção oposta daquela em que ele partiu. Assim, este valor atende qualquer mudança de direção que o veículo deva realizar para chegar de um ponto a outro. A Figura 4.7 mostra como o fator foi calculado. O avião parte do *waypoint A* com direção de 180° e inclinação (ângulo de rolagem) constante. A distância percorrida (o comprimento do arco) é medida até que a sua direção fique oposta, ou seja, até que a bússola marque 0°. Por fim, o fator de ajuste é o comprimento do arco.

Dois domínios foram utilizados para a característica distância. O primeiro, B1, é uma cópia do domínio A2 e por isso não considera o fator de ajuste. Já o segundo, B2, é o domínio B1 levando em consideração o fator de ajuste. Um exemplo de código em PDDL onde o fator de ajuste da distância (*distanceFactor ?uav - UAV*) é aplicado pode ser observado na Figura 4.8. Nesse caso, o código é utlizado para o cálculo da duração da ação de deslocamento. Note que por ser baseado no domínio A2, a velocidade considerada é a TAS.

Essa abordagem sempre levará a um plano mais conservador, já que, em

[4]A pose descreve a posição e orientação do veículo no espaço. No caso de um veículo aéreo, a pose é descrita por seis variáveis, sendo três para a posição: latitude, longitude e altitude e três para a atitude: ângulos de rolagem, arfagem e guinada.

[5]Os dados foram obtidos realizando experimentos no simulador de voo X-Plane®.

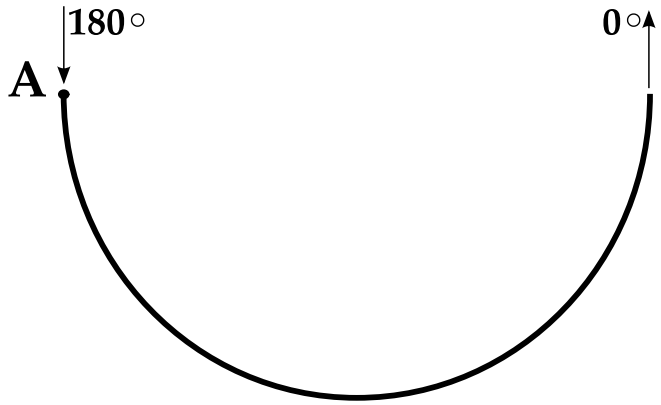

Figura 4.7. Cálculo do fator de ajuste para a distância planejada. O fator de ajuste é o comprimento do arco.

```
(/
  (+ (distance ?l1 ?l2) (distanceFactor ?uav))
  (* (speed ?uav) (+ 1 (* (altitude ?l1) (tasFactor))))
))
```

Figura 4.8. Aplicação do fator de ajuste para a distância planejada em PDDL. O código é utilizado para o cálculo da duração da ação de deslocamento.

algumas situações, o veículo percorrerá efetivamente o menor caminho (ou algum valor próximo) e isso faz com que a duração, a distância e o consumo de combustível realizados sejam menores do que o planejado.

4.4.4 Característica Altitude

A altitude é uma característica fundamental em qualquer modelo de planejamento para veículos aéreos. Ela influencia diretamente em muitas outras variáveis.

Foram utilizados os domínios C1 e C2 para modelar a altitude. O domínio C1 é equivalente ao B2 visto anteriormente. Já o domínio C2 é uma adaptação do C1 com a inclusão de três ações para o deslocamento. Cada ação representa uma fase de voo.

4.4.4.1 Domínio C1

O domínio C1 considera apenas uma ação para o deslocamento. Isso significa que apenas um valor para a velocidade e outro para a taxa de consumo de combustível podem ser utilizados. Entretanto, nem sempre o veículo executará a ação de

deslocamento utilizando apenas uma velocidade. Ele pode, por exemplo, voar uma parte do trajeto em voo de subida e a outra em voo de cruzeiro.

Para exemplificar, considere o cenário representado na Figura 4.9. Neste caso, o veículo deve voar do WP0 para o WP3 (passando pelo WP1 e WP2). Note que o planejamento foi realizado considerando que o veículo voaria durante toda a missão com velocidade de 100 knots. Entretanto, ao executar esse plano, o veículo voou com duas velocidades que não foram planejadas. Do WP0 ao WP1, por exemplo, ele voou primeiramente com 75 knots e depois com 100 knots. Do WP2 ao WP3 ele voou primeiramente com 110 knots e logo depois com 100 knots. Apenas do WP1 ao WP2 o realizado foi o planejado.

O veículo voa em fases diferentes durante uma mesma ação de deslocamento, pois ele pode atingir a altitude do *waypoint* de destino antes de atingir o *waypoint* propriamente dito, assim, ele realiza um trecho subindo ou descendo e o restante em regime de voo de cruzeiro.

Essa diferença entre a execução e o planejamento pode interferir na qualidade do plano e o domínio C1 é utilizado para verificar em quais condições planejar dessa maneira mais simplificada é interessante.

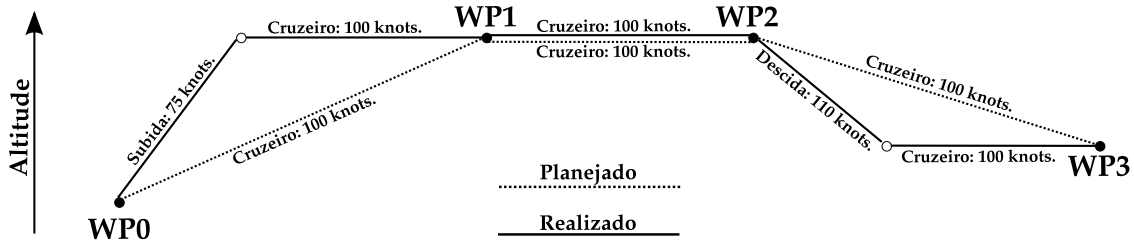

Figura 4.9. Domínio C1: Apenas uma velocidade e taxa de consumo de combustível considerados para todo o planejamento.

4.4.4.2 Domínio C2

O domínio C2 modela o deslocamento do veículo utilizando três ações. Cada ação representa uma fase de voo. Com essa maneira de abordar o problema é possível considerar velocidades e taxas de consumo de combustível distintas por ação. Na Tabela 4.7, podem ser observadas as funções que foram criadas para esse domínio. As ações de subida e descida levam ainda em consideração a velocidade vertical do veículo, modelada, respectivamente, pelas funções (*climbRate ?uav*) e (*descentRate ?uav*). Dessa maneira, é possível planejar a duração, velocidade, distância e o consumo de combustível nas fases de voo de subida ou descida.

O objetivo do domínio C2 é modelar o deslocamento vertical do veículo e, assim, melhorar a qualidade do plano gerado. A Figura 4.10 mostra o comportamento que é esperado ao criar as duas ações de deslocamento junto com suas respectivas velocidades verticais. Note que, nesse caso, espera-se que o planejado fique próximo do realizado.

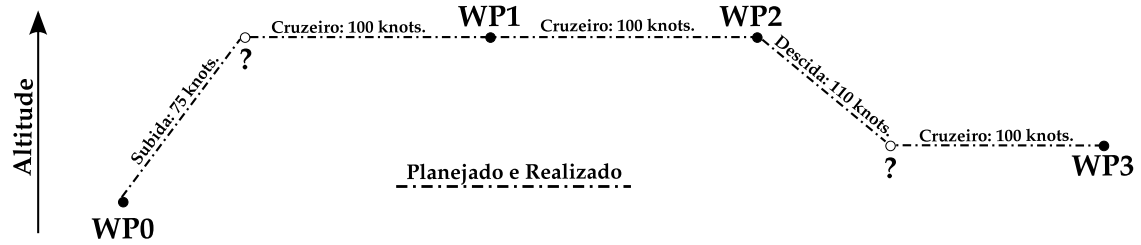

Figura 4.10. Domínio C2: Velocidades distintas por fase de voo. Além disso, as ações de subida e descida são modeladas levando em consideração o fato do veículo chegar na altitude de destino antes de chegar no destino propriamente dito.

Os cálculos da duração e do consumo de combustível de cada ação foram baseados na Seção 4.2.4. Um desafio ao modelar esse domínio surgiu, pois o ponto onde a aeronave passa do voo de subida ou descida para o voo de cruzeiro não é um ponto conhecido pelo planejador, ou seja, não é um *waypoint* inserido no arquivo de problema. Na Figura 4.10 este tipo de ponto está mostrado como um círculo branco e o símbolo de interrogação.

Para exemplificar, considere o trecho entre o WP0 e o WP1. O planejador conhece apenas esses dois *waypoints* e a distância entre eles. Entretanto, quando o avião atinge o nível de voo do WP1, essa distância já não pode mais ser considerada para saber a duração e consumo no voo de cruzeiro, pois ele já se deslocou horizontalmente.

A solução encontrada foi utilizar uma distância auxiliar, modelada utilizando a função (*distanceAux ?uav*). Essa função recebe a distância percorrida pelas ações de subida ou descida. Ao aplicar a ação de cruzeiro, uma nova distância é calculada pelo planejador como sendo a diferença entre a distância original e a auxiliar. Caso sucessivas ações de voo de cruzeiro sejam aplicadas em sequência então essa distância auxiliar sempre valerá zero. Note que a distância auxiliar é definida por aeronave, podendo assim, ser utilizada em um planejamento para múltiplos veículos.

O domínio C2 ficou extenso, são mais de 250 linhas de código PDDL. Devido a limitações da linguagem, boa parte do código são cálculos repetidos em vários trechos. Por isso, mostrar o domínio aqui se torna inviável. Entretanto, alguns

trechos são interessantes de serem visualizados. O cálculo da distância auxiliar (*distanceAux ?uav*) é realizado nos efeitos das ações de subida e descida conforme pode ser observado na Figura 4.11. Lembrando que essa função representa a distância que o veículo percorreu durante a fase de subida ou descida e será descontada na ação de voo de cruzeiro. O mesmo cálculo é utilizada na pré-condição das ações subir e descer com o objetivo de verificar se a distância que será percorrida nessa fase é maior do que a distância total entre os dois *waypoints*. Caso seja, então a ação não pode ser aplicada, pois significa que os *waypoints* estão muito próximos um do outro.

```
(at end (assign (distanceAux ?uav)
  (+ ; distancaAux = df + d2
    ; df = Fator de ajuste da distancia
    (distanceFactor ?uav)
    ; d2 = Distancia percorrida na subida
    (* ; d2=vt
      ; velocidade (TAS)
      (* (climbSpeed ?uav)
         (+ 1
           (* (/ (+ (altitude ?l1) (altitude ?l2)) 2)
           (tasFactor))
         )
      )
      ; tempo do voo de subida (em horas)
      (/ (/ (- (altitude ?l2) (altitude ?l1)) (climbRate ?uav)) 60)
    )
  )))
```

Figura 4.11. Domínio C2: Modelagem da distância auxiliar em PDDL.

Uma outra questão interessante foi como inserir o fator de ajuste da distância planejada mostrado no domínio B2 neste domínio. Ele foi inserido nas três ações (subida, descida e cruzeiro), sendo que se uma ação de cruzeiro for aplicada depois de uma ação de subida ou descida então ele é considerado como sendo zero (pois já foi aplicado na ação anterior). A Figura 4.12 mostra como foi realizado o cálculo da duração da ação de subida. Note que é uma soma do tempo necessário para percorrer a distância descrita pelo fator de ajuste e o tempo necessário para subir até a altitude de destino.

4.5 Missão de combate a incêndios florestais

Considere a seguinte situação: três incêndios florestais estão ocorrendo simultaneamente. Uma equipe de bombeiros está tentando combater os incêndios em terra e possui à sua disposição um conjunto de robôs aéreos autônomos e heterogêneos que conseguem obter água em dois lagos e lançá-la nos focos de incêndio. No total, são

```
(+   ; t=t1+t2
  (* ; t1 --> converte horas para minutos
    (/ ;t1=d/v
      ; distancia
      (distanceFactor ?uav)
      ;velocidade (tas)
      (* (climbSpeed ?uav)
        (+ 1 (* (/ (+ (altitude ?l1) (altitude ?l2)) 2) (tasFactor))))
     )
     60
   )
   ; t2 --> tempo para subir ate o local l2
   (/ (- (altitude ?l2) (altitude ?l1)) (climbRate ?uav))
)
```

Figura 4.12. Domínio C2: Cálculo da duração da ação de subir.

quatro aeronaves que se diferem pela velocidade e capacidades dos reservatórios de água e combustível. Além disso, existem dois aeroportos nas proximidades e são apenas nesses locais que os veículos podem reabastecer. A Figura 4.13 ilustra esse cenário.

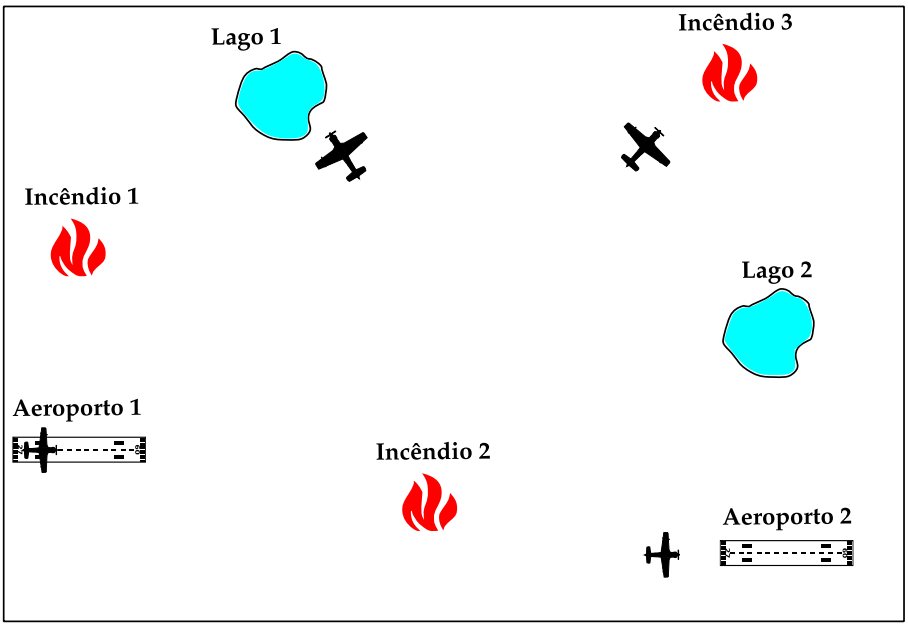

Figura 4.13. Visão geral da missão de combate a incêndio florestais.

Um bombeiro possui um dispositivo em mãos (*palm top, celular, etc.*) que contém um mapa em 2D do local. Com a experiência de combate a incêndios deste tipo, o bombeiro consegue saber com certa precisão, o local onde a água deve ser lançada para que os efeitos do fogo sejam minimizados e a equipe em terra consiga extingui-lo. Com apenas alguns cliques na tela, o bombeiro seleciona os locais onde ele deseja que a água seja lançada. Além disso, ele informa qual é o tempo máximo

para que esses lançamentos sejam realizados. Após confirmar, esses dados são enviados por satélite para um sistema que, a partir dos locais selecionados e da situação de cada veículo, gera um plano de combate a esses incêndios.

Como pode ser observado na figura, algumas aeronaves já estão em voo, outras estão pousadas e sem água no reservatório. As que estão em voo, podem ter que reabastecer para conseguir ajudar na missão. Além disso, dependendo da intensidade dos incêndios, pode ser necessário realizar mais de um lançamento de água no mesmo incêndio. Com isso, vários deslocamentos podem ocorrer o que, consequentemente, pode gerar vários reabastecimentos. Uma outra questão importante é que nem todas aeronaves precisam participar da missão, já que, se os incêndios forem de baixa intensidade, utilizando poucos veículos é possível eliminá-los.

A questão a ser respondida é: *Quais veículos devem ser utilizados e qual é a sequência de tarefas que cada um deve realizar para cumprir a missão*? Note que existem muitas alternativas para resolver esse problema e é por isso que ferramentas computacionais são essenciais.

Uma das maneiras de responder a essa questão é utilizar um planejador PDDL. Dentro deste contexo, o objetivo desta missão hipotética de combate a incêndios florestais é avaliar a capacidade que a PDDL possui de gerar planos temporais para múltiplos veículos. A próxima seção dedica-se a detalhar a maneira como essa missão foi implementada em PDDL.

4.5.1 Modelagem em PDDL

O domínio criado foi baseado no C2. Sendo assim, os objetos, predicados (com exceção do *visited*) e funções já exibidos, respectivamente, nas Tabelas 4.5, 4.6 e 4.7 estão presente neste novo domínio. As seções a seguir detalham o que foi inserido.

4.5.1.1 Tipos de Objetos

A Tabela 4.8 mostra os tipos de objetos que foram criados para a missão de combate a incêndios florestais. Note que todos eles herdam do tipo *Location*.

4.5.1.2 Predicados

Quatro predicados foram criados como pode ser observado na Tabela 4.9. O predicado (*haveWater ?uav - UAV*) é utilizado para indicar que um determinado veículo possui água, e se torna verdadeiro no efeito da ação usada para obter água

Objeto	*Tipo*	*Descrição*
Airport	Location	Representa um aeroporto.
Lake	Location	Representa um lago.
Fire	Location	Representa um incêndio.

Tabela 4.8. Tipos de objetos definidos para a missão de combate a incêndios florestais.

em um lago. Já o predicado (*missionOn*), é utilizado para indicar que o tempo máximo para a realização da missão ainda não foi atingido. As pré-condições de todas as ações do domínio devem incluí-lo. Assim, se ele estiver falso, as ações não poderão ser executadas, indicando, dessa maneira, que o tempo máximo para a realização da missão finalizou. Este predicado é alterado pelos eventos exógenos (ver Seção 3.1.2.2).

O predicado (*available ?arpt - Airport*), é utilizado na ação reabastecer para indicar que o aeroporto está disponível (ou não) para realizar um reabastecimento. Na prática, ele é utilizado para que apenas um veículo possa reabastecer por vez, evitando, assim, que mais de uma ação de reabastecimento sejam executadas ao mesmo tempo no mesmo aeroporto.

Predicado	*Descrição*
(fuelStation ?airport - Airport)	Indica que o aeroporto é também um posto de reabastecimento.
(haveWater ?uav - UAV)	Indica que o veículo possui água em seu reservatório.
(available ?arpt - Airport)	Indica que o aeroporto está disponível para realizar um reabastecimento.
(missionOn)	Se verdadeiro, indica que o tempo máximo para a realização da missão ainda não foi atingido.

Tabela 4.9. Predicados definidos para a missão de combate a incêndios floretais.

4.5.1.3 Funções

As funções criadas para esta missão podem ser observadas na Tabela 4.10. A função (*fireIntensity ?f - Fire*) representa a intensidade do incêndio. Quanto maior o número inteiro mapeado por essa função, maior é a intensidade. Essa função é decrementada em uma unidade a cada lançamento de água realizado no incêndio. Assim, quando a intensidade chegar a zero, considera-se que o incêndio está extinto.

Por outro lado, a capacidade de supressão de incêndios do veículo é representada pela função (*fireSupression ?uav - UAV*). Assim, ele só poderá lançar agúa em um determinado incêndio se ele possuir capacidade para tal, ou seja, se (*fireIntesity <= fireSupression*).

Por exemplo, considere que os incêndios *Fire1* e *Fire2* possuam intensidades 2 e 3 respectivamente e que existam dois veículos, *UAV1* e *UAV2*, o primeiro com capacidade 3 e outro com capacidade 1.

O *UAV2* só poderá lançar água no *Fire1* depois que sua intensidade passar de 2 para 1 e isso só será feito quando o *UAV1* lançar água neste incêndio. Isso cria um desafio para o planejador PDDL e representa (de certa forma) a heterogeneidade dos veículos.

Um trecho do arquivo de problema em PDDL poderia ser descrito da seguinte forma:

```
(= (fireIntensity Fire1) 2)
(= (fireIntensity Fire2) 3)
(= (fireSupression UAV1) 3)
(= (fireSupression UAV2) 1)
```

A função (*totalFuelUsed*) é incrementada com o valor do combustível gasto a cada ação de deslocamento (independentemente do veículo). Portanto, o valor que ela armazena é a soma do combustível gasto por todos os veículos. Já a função (*totalRefuel*) é incrementada em uma unidade a cada reabastecimento realizado (independentemente do veículo). Essas funções são úteis caso deseja-se minimizar o total de combustível gasto ou o número de reabastecimentos realizados.

4.5.1.4 Ações

Além da ação deslocamento aproveitada inteiramente do domínio C1, foram adicionadas três ações: reabastecer, obter água e lançar água. A ação de reabastecimento só pode ser realizada se o veículo estiver em um aeroporto que possua posto de reabastecimento. Sendo assim, antes de realizar o reabastecimento, o veículo precisa se deslocar até o respectivo aeroporto. A Tabela 4.11 mostra as ações que são utilizadas nesse domínio. Vale destacar que, por simplificações no modelo, as ações de obter e lançar água não consomem combustível. Por outro lado, a duração dessas ações é sempre muito pequena (1 min). Assim, o fato de não consumir combustível não é um fato tão relevante para o planejamento.

Função	*Descrição*
(fireIntensity ?f - Fire)	Número inteiro que representa a intensidade do incêndio.
(fireSupression ?uav - UAV)	Número inteiro que representa a capacidade de supressão de incêndios do veículo.
(timeToRefuel ?uav - UAV)	Tempo necessário para realizar o reabastecimento.
(timeToDropWater ?uav - UAV)	Tempo necessário para lançar água em um incêndio.
(timeToPickupWater ?uav - UAV)	Tempo necessário para obter água em um lago.
(totalFuelUsed)	Representa o combustível gasto por todos os veículos.
(totalRefuel)	Representa o número de reabastecimentos que todos os veículos realizaram.

Tabela 4.10. Funções utilizadas na missão de combate a incêndios florestais.

Ação	*Parâmetros*	*Descrição*
go	(?uav - UAV ?l1 - Location ?l2 - Location)	Representa o deslocamento do ?uav do local ?l1 para o local ?l2.
dropWater	(?uav - UAV ?f - Fire)	Representa o lançamento de água no incêndio ?f pelo ?uav.
pickupWater	(?uav - UAV ?l - Lake)	Representa a obtenção de água no lago ?l pelo ?uav.
refuel	(?uav - UAV ?a - Airport)	Representa o reabastecimento do ?uav no aeroporto ?a.

Tabela 4.11. Ações utilizadas na missão de combate a incêndios florestais.

Note que não foram criadas ações específicas para o pouso e a decolagem. A própria ação de deslocamento é utilizada para esses casos. A ação de deslocar recebe como parâmetro o veículo e dois objetos do tipo *Location*. O primeiro representa o local de partida e o segundo o local de destino. Assim, o primeiro pode representar um aeroporto e o segundo um lago ou o primeiro um incêndio e o segundo um lago e assim por diante. Dessa maneira, utilizou-se a funcionalidade de hierarquia de tipos da PDDL para construir um domínio mais simples de ser resolvido por um planejador.

A Figura 4.14 mostra a ação utilizada para lançar água em um incêndio. Note

```
(:durative-action dropWater
  :parameters (?uav - UAV ?f - Fire)
  :duration (= ?duration (timeToDropWater ?uav))
  :condition
  (and
    (at start (missionOn))
    (at start (atLocation ?uav ?f))
    (over all (atLocation ?uav ?f))
    (at end (atLocation ?uav ?f))
    (at start (<= (fireIntensity ?f) (fireSupression ?uav)))
    (at start (haveWater ?uav))
  )
  :effect
  (and
    (at start (decrease (fireIntensity ?f) 1))
    (at start (not (haveWater ?uav)))
  )
)
```

Figura 4.14. Ação *dropWater* utilizada para lançar água em um incêndio.

que, na pré-condição, é especificado, utilizando os marcadores temporais *at start*, *over all* e *at end*, que o veículo deve permanecer durante toda a duração da ação no local do incêndio (*atLocation*). Isso impede que a ação de deslocar seja aplicada antes que esta termine. Na prática, isso significa que o avião deve lançar toda a água para depois iniciar o seu deslocamento para outro ponto. Portanto, isso evita que o planejador PDDL explore o paralelismo na aplicação dessas duas ações. Deve-se ficar claro, entretanto, que isso vale apenas para o mesmo veículo. Assim, nada impede que duas ações de lançamento de água sejam executadas ao mesmo tempo e no mesmo incêndio, porém, para veículos distintos.

É interessante notar também, que o predicado (*haveWater*), é alterado no início da ação, pois se fosse alterado no fim, nada imepediria o planejador de gerar duas ações de lançamento de água ao mesmo tempo e para o mesmo veículo e local.

Capítulo 5

Arcabouço Experimental

Neste capítulo será descrito o arcabouço que foi desenvolvido para a realização dos experimentos. Entretanto, antes de detalhá-lo, será descrito na Seção 5.1 a importância e as vantagens encontradas na utilização de simuladores de voo para testes e validações em robôs aéreos. Após, será detalhado na Seção 5.2, cada módulo que compõe o arcabouço e, por último, na Seção 5.3, será demonstrada a estratégia utilizada para realizar o controle autônomo dos veículos no simulador.

5.1 Simuladores de Voo

Simuladores de voo são sistemas que tentam reproduzir da forma mais realista possível a experiência de voar uma aeronave. Existem vários tipos de simuladores. Alguns são específicos para uma determinado avião e acompanham não só o *software* e o *hardware*, como também a cabine do piloto com os mesmos controles existentes na aeronave real. Geralmente, este tipo de abordagem é utilizada quando um piloto profissional deve ser treinado com as características específicas do veículo em questão. Entretanto, possuem um custo muito alto e como o propósito deles é bem específico, não é possível utilizá-los, por exemplo, para a validação de algoritmos de controle de um robô aéreo.

Por outro lado, existem simuladores de voo para uso em computadores pessoais que são muito famosos e amplamente utilizados em todo o mundo. O principal atrativo deste tipo de *software* é a forma realista com que é tratata tanto a simulação do modelo da dinâmica de voo dos veículos, quanto os gráficos, cenários e aeroportos. Além disso, por utilizarem dados reais de navegação aérea, permitem que os pilotos virtuais consigam realizar um voo, por exemplo, de Belo Horizonte ao Rio de Janeiro com as mesmas cartas de navegação utilizadas em um voo real.

Os simuladores de voo de uso pessoal mais famosos são o *Microsoft Flight Simulator*® (MsFS) e o X-Plane®. Ambos são comercializados a preços compatíveis aos benefícios que oferecem[1]. Um outro simulador muito conhecido principalmente no mundo científico é o *FlightGear Flight Simulator* (FgFS), esse último é gratuito e de código aberto [Olson, 2007; Sorton & Hammaker, 2005]. O nível de realismo encontrado nesses simuladores é tão satisfatório que o X-Plane® possui uma versão para treinamento de pilotos que é certificada pelo órgão regulador da aviação civil nos EUA[2].

Um atrativo a mais que esses *softwares* oferecem é a possibilidade de customizar várias características, entre elas destacam-se: a inclusão ou modelagem de aeronaves e o desenvolvimento de novos cenários incluindo informações detalhadas de relevos e de aeroportos. Uma outra funcionalidade fundamental que todos eles compartilham é a possibilidade de extrair e enviar dados para o simulador por intermédio de programas que são executados internamente ou externamento ao simulador. No caso dos programas executados externamente, a comunicação é realizada utilizando protocolos de rede bem conhecidos como TCP e UDP. Já a comunicação interna é realizada com o uso de bibliotecas que são desenvolvidas seguindo uma interface padrão e executadas no mesmo espaço de memória do simulador. Este tipo de característica permite que terceiros desenvolvam programas para os simuladores estendendo as suas funcionalidades nativas. Já do ponto de vista científico, permite que esses simuladores sejam utilizados para a realização de diversos tipos de pesquisas.

Um dos principais usos dos simuladores de voo em pesquisas científicas são para testes denominados *Hardware-in-the-loop* (HWIL) [Alves Neto & Campos, 2008; Adiprawita et al., 2007; Astuti, 2007]. Neste tipo de abordagem, o simulador disponibiliza para um dispositivo de piloto automático (por meio de comunicação externa) dados como a posição e atitude da aeronave. O piloto automático, por sua vez, recebe esses dados, realiza algum tipo de processamento e envia para o simulador comandos que movimentam as superfícies de controle do veículo fazendo com que o voo seja realizado de maneira autônoma. Portanto, os algoritmos e estratégias de controle podem ser testados no *hardware* real em segurança, diminuindo os custos e agilizando o processo de validação. Após a realização dos testes, o mesmo piloto automático com os algortimos validados pelo processo HWIL, podem ser embarcados com mais confiança e segurança no avião real.

[1] O MsFS custa aproximadamente R$100,00, já o X-Plane®, em sua versão mais básica, custa aproximadamente R$60,00, entretanto, é muito difícil encontrá-lo no Brasil.

[2] *Federal Aviation Administration* (FAA)

Em resumo, os simuladores de voo citados acima oferecem diversas vantagens para a comunidade científica, entre elas, descatam-se [Cantoni et al., 2009]:

- modelagem dinâmica bastante próxima ao comportamento real das aeronaves;
- aspectos do motor (temperatura e pressão do óleo), bem como o consumo de combustível e peso da aeronave já modelados;
- ambiente de simulação e visualização tridimensional;
- possibilidade de criação de novos veículos e cenários;
- possibilidade de intercâmbio de dados entre o simulador e programas externos;
- realismo na simulação de efeitos atmosféricos como rajadas de vento, neblina e térmicas.

Com isso, os pesquisadores podem se concentrar no tema principal da pesquisa e deixar para o simulador resolver outros detalhes também complexos e necessários para sua realização. O presente trabalho faz uso do simulador de voo X-Plane® para a execução dos planos gerados em PDDL. Nesse caso, não foi utilizado um piloto automático externo para controlar o avião. Como o foco da pesquisa foi avaliar a PDDL dentro do planejamento de missões, utilizou-se o piloto automático do próprio X-Plane® para guiar o veículo de maneira autônoma. Ambos serão detalhados respectivamente nas Seções 5.2.1 e 5.3.

5.2 Descrição do Arcabouço

O arcabouço desenvolvido para a realização dos experimentos é composto por diversos módulos. Alguns foram desenvolvidos para este trabalho, outros são *softwares* desenvolvidos por terceiros. A linguagem Java foi utilizada no desenvolvimento e alguns *scripts* para coleta de dados e testes no X-Plane® foram escritos em Groovy[3]. Optou-se por Groovy por ser de sintaxe mais simples, não necessitar de compilação e ser totalmente integrada à linguagem Java.

A Figura 5.1 mostra uma visão integrada dos módulos. Cada módulo, por sua vez, é detalhado abaixo.

Desenvolvimento do Arquivo de Domínio: Esta é a primeira etapa, indicada na figura por (1). Aqui, o arquivo de domínio (descrito em PDDL) é desenvolvido. O arquivo de domínio representa um determinado tipo de missão. Esta tarefa é realizada apenas uma vez, já que o mesmo arquivo pode ser reutilizado

[3]http://groovy.codehaus.org/

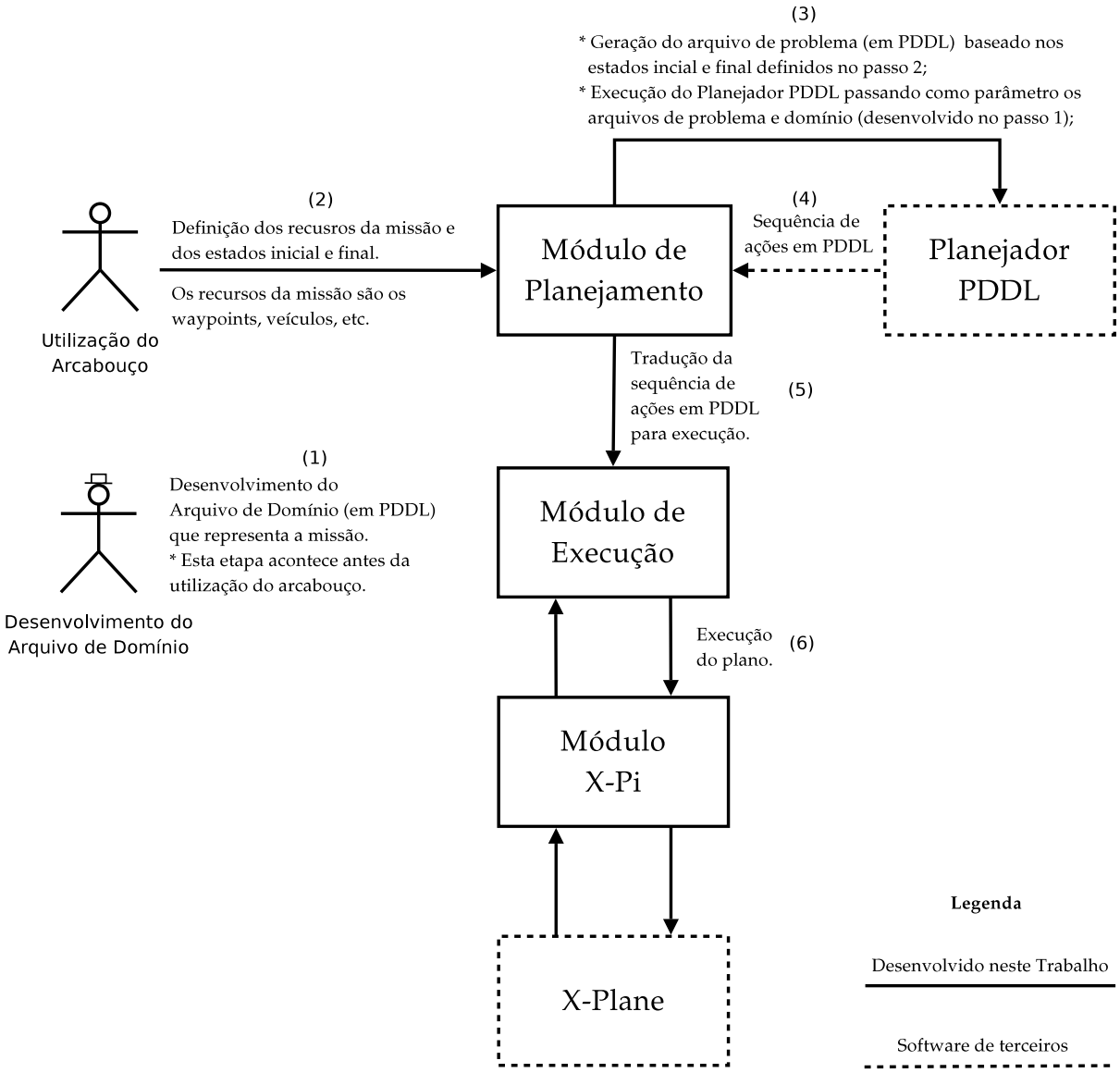

Figura 5.1. Visão geral do arcabouço experimental desenvolvido para a execução dos planos gerados em PDDL.

para diferentes instâncias. Por isso, essa etapa acontece antes da utilização do arcabouço.

Utilização do Arcabouço: Nesta estapa, indicada por (2) na figura, o arcabouço pode ser utilizado e a primeira tarefa é definir uma instância da missão. No caso da missão de navegação entre *waypoints*, uma instância é composta pelos veículos que estão disponíveis para cumprí-la e os *waypoints* que devem ser vistados. A pessoa que utiliza o arcabouço não precisa ser a mesma que desenvolveu o arquivo de domínio.

Módulo de Planejamento: Módulo responsável pela terceira etapa, indicada na figura como (3). As seguintes tarefas são realizadas:

- Geração do arquivo de problema (em PDDL). Nesta tarefa, são contruídos os estados inicial e final baseados nos recursos que foram definidos na etapa (2);
- Execução do Planejador PDDL, passando os arquivos de domínio e problema como parâmetros. A saída é uma sequência de ações (o plano) indicada pelo passo (4) na figura;
- Tradução da sequência de ações em PDDL gerada no passo (4) para as ações em Java que serão efetivamente executadas. Para cada ação PDDL contida no plano, uma classe Java (que representa essa ação) é instanciada para posterior execução.

Planejador PDDL: Responsável pela geração do plano. Recebe como parâmetro o arquivo de problema gerado pelo módulo de planejamento e o arquivo de domínio gerado na etapa (1). A saída é um plano (uma sequência de ações) que será traduzida para execução pelo módulo de planejamento, conforme descrito acima.

Módulo de Execução: Este módulo é responsável pela execução do plano. Recebe como parâmetro a lista de ações traduzidas para execução pelo módulo de planejamento. Para cada ação, diversos parâmetros são coletados, consolidados e exportados para posterior análise. Este módulo fica em execução até que todas as ações sejam executadas.

X-Pi: *X-Plane Interface* é o módulo responsável pela interface entre o módulo de execução e o X-Plane®. A interface consiste em enviar e receber dados do simulador X-Plane® por meio de comunicação de rede via protocolo UDP.

X-Plane®: Simulador de voo escolhido para realização dos experimentos.

5.2.1 X-Plane®

O X-Plane® é um simulador de voo que já está no mercado há mais de 10 anos. Um dos principais diferenciais entre os seus concorrentes é maneira como ele calcula o modelo da dinâmica de voo das aeronaves. Ao invés de ser baseado em uma abordagem paramétrica como são os casos do MsFS e o FgFS, no X-Plane®, os parâmetros do modelo são obtidos a partir da geometria do avião por um processo

de engenharia chamado de *blade element theory*. O processo basicamente consiste em dividir o avião em pequenos elementos e, a partir dessa divisão, encontrar as forças que agem em cada pequeno elemento. As forças são então convertidas em acelerações que são integradas para encontrar a velocidade e a posição do veículo [Gimenes et al., 2008]. Esse processo facilita a criação de aeronaves já que é necessário apenas desenhar[4] a aeronave e fornecer alguns parâmetros, o resto é derivado pelo próprio simulador. Uma possibilidade interessante no X-Plane® (também compartilhada pelo MsFS e o FgFS) é a possibilidade de desabilitar o modelo da dinâmica de voo nativo e utilizar um modelo externo.

O simulador foi escolhido para a realização dos experimentos como forma de enriquecer um outro trabalho feito em 2009 que comparou o MsFS com o FgFS, [Cantoni et al., 2009]. Dessa forma, ao entender como o X-Plane® funciona é possível fazer uma comparação completa entre os três principais simuladores de voo para pesquisas que envolvam veículos aéreos. A escolha se mostrou muito adequada e, como será visto mais adiante, todo o trabalho desenvolvido com o X-Plane® poderá ser reaproveitado para outras linhas de pesquisa.

5.2.1.1 Qualidade Gráfica e Taxa de Atualização do Modelo

A qualidade gráfica do simulador é muito satisfatória. Entretanto, para a execução em sua máxima capacidade, uma placa de vídeo com alto poder de processamento deve ser utilizada. A Figura 5.2 mostra uma imagem do simulador obtida com uma placa de vídeo GeForce 9800 GT da NVIDIA. Nesse exemplo, as propriedades gráficas foram todas habilitadas no máximo. Devido a isso, a taxa de atualização dos quadros, conhecida por *frames per second* (FPS), diminuiu consideravelmente e, consequentemente, a experiência de voar ficou muito comprometida.

A frequência de simulação do modelo da dinâmica de voo no X-Plane® está diretamente ligada à taxa de atualização gráfica da tela, como em jogos digitais onde a cada iteração do laço principal todas essas tarefas (e outras mais) são realizadas [Garcia & Barnes, 2010]. O problema com esse tipo de abordagem é que ela prejudica os algoritmos de controle que necessitam de uma frequência mínima para conseguir, por exemplo, manter a altitude e direção de uma aeronave[5]. Já no FgFS, essas duas tarefas acontecem separadamente, de modo que é possível aumentar consideravelmente a frequência de simulação aerodinâmica (para centenas de *hertz*) em detrimento da atualização gráfica [Cantoni et al., 2009].

[4] A construção de uma aeronave é realizada com a ferramenta *PlaneMaker*® que acompanha o X-Plane®.

[5] O MsFS também utiliza essa abordagem.

Felizmente, esse não é um problema tão sério para muitas pesquisas científicas, já que, geralmente, os pesquisadores estão interessados na parte interna do simulador, ou seja, nos modelos atmosférico, da dinâmica de voo, de consumo de combustível, de falhas e etc. Nesses casos, a parte gráfica não é tão importante e, sendo assim, basta reduzir as opções gráficas até que seja possível realizar o voo de maneira suave. Entretanto, isso não elimina a necessidade do computador possuir uma placa de vídeo razoável.

Figura 5.2. Avião Cessna 172SP 180HP no simulador de voo X-Plane®.

5.2.1.2 Entrada e Saída de Dados

Um dos principais mecanismos que um simulador de voo deve fornecer para ser utilizado em pesquisas científicas é o de entrada e saída de dados, ou seja, deve ser possível obter e alterar parâmetros do simulador por meio de programas que podem ser executados internamente ou externamente. O X-Plane® fornece dois mecanismos para isso. O mais básico permite que 132 mensagens sejam visualizadas no próprio simulador[6], gravadas em um arquivo texto ou enviadas pela rede utilizando pacotes UDP. Cada mensagem é composta por oito parâmetros no máximo. A Figura 5.3 exibe um pedaço do painel do avião Cessna 172SP com algumas mensagens sendo

[6]Versão 9.55.

exibidas (sustentação e arrasto das asas, consumo de combustível em *lb/h* e entre outros).

Uma outra possibilidade muito utilizada pela comunidade de usuários do X-Plane® é a utilização do X-Plane® SDK, um conjunto de bibliotecas escritas em C++ que permite o desenvolvimento de aplicações (*plugins*) que são executados dentro do simulador (no mesmo espaço de memória). Com isso, o número de alternativas para customizações aumenta muito, já que, ao ser executada dentro do simulador, a aplicação possui acesso a várias estruturas de dados que permitem o desenvolvimento de programas mais complexos. Nesse caso, o número de parâmetros que podem ser obtidos e alterados é muito maior do que os 132 utilizados na abordagem citada acima. O SDK é de código fechado, mas gratuito e desenvolvido por terceiros.

Nesse trabalho, como será visto na Seção 5.2.2, optou-se pelo desenvolvimento de uma biblioteca que utiliza a primeira abordagem.

Figura 5.3. Painel do Cessna 172SP e algumas mensagens sendo exibidas.

5.2.2 X-Pi

A *X-Plane Interface* (X-Pi) é uma biblioteca desenvolvida em Java para este trabalho. O objetivo é prover mecanismos que facilitem o envio e recebimento de dados para o X-Plane®. Uma das vantagens da X-Pi é o fato dela ser independente do restante dos módulos, podendo, portanto, ser utilizada em outros projetos. Devido a isso, ela foi disponibilizada no Google Code e pode ser acessada pelo seguinte endereço: `http://code.google.com/p/x-pi/`. Por ter sido desenvolvida em Java ela pode ser utilizada de forma natural em outras linguagens como Groovy, Jython[7] e Matlab®.

Por meio da X-Pi todas as 132 mensagens disponibilizados pelo menu *Data Input and Output* do simulador podem ser facilmente obtidas. O funcionamento da interface é baseado em uma *thread* que fica a todo tempo recebendo os dados (pacotes UDP) do simulador e disponibilizando-os em um formato de fácil acesso para o desenvolvedor. O envio de dados é independente dessa *thread* e acontece no mesmo momento em que é feita a chamada ao comando de envio da biblioteca.

Entretanto, a biblioteca abstrai toda essa complexidade e disponibiliza para o desenvolvedor uma interface bem definida para o acesso ao simulador. A Figura 5.4 mostra um exemplo de utilização da X-Pi. Nesse caso é obtido e impresso na tela a altitude em relação ao nível do mar (*position.altMsl*) e a velocidade indicada (*position.ias*). Além disso, altera-se a quantidade de combustível em uma das asas caso o valor atual seja menor do que 157 lb.

Abaixo segue a explicação para as principais chamadas realizadas no código exibido na Figura 5.4.

XPlaneInterface: Essa é a principal classe da biblioteca. É reponsável por disponibilizar todos os métodos necessários para troca de dados com o X-Plane®, em outras palavras, é a abstração de todo o processo de comunicação com o simulador. O usuário da biblioteca precisa ter conhecimento somente desta classe. O construtor recebe o endereço IP do computador onde o X-Plane® está sendo executado, as portas de recebimento e envio de dados, além de um arquivo XML contendo a configuração das mensagens. Todos os métodos descritos abaixo são membros dessa classe.

unregisterDATAMessage: Cada uma das 132 mensagens disponibilizadas pelo X-Plane® podem ser desabilitadas utilizando este método. Desabilitar uma mensagem significa que ela não será mais enviada pelo simulador via rede.

[7] http://www.jython.org/

```
1  xplaneIP = "192.168.1.126"
2  receivePort = 49002
3  sendPort = 49000
4  dataGroupXML = "DATAGroupConfig.xml"
5
6  xpi = new XPlaneInterface(xPlaneIP,recvPort,sendPort,dataGroupXML)
7  xpi.unregisterDATAMessages("*")
8  xpi.registerDATAMessages("0,1,3,4,6,20,21,25,29,34,35,36,37,47")
9  xpi.startReceiving()
10 Thread.sleep 1000
11
12 altitude = xpi.getValue ("position.altMsl")
13 speed = xpi.getValue ("position.ias")
14 println speed
15 println altitude
16
17 fuel = xpi.getValue ("fuel.fuel1")
18 if (fuel < 157.0f) {
19   xpi.setValue "fuel.fuel1", 157.0f
20 }
21
22 xpi.stopReceiving()
```

Figura 5.4. Código em Groovy mostrando os comandos necessários para o estabelecimento de uma conexão, além do envio e recebimento de dados entre a X-Pi e o X-Plane®.

Caso seja passado o caracter "*" como parâmetro, então todos as mensagens serão desabilitados.

registerDATAMessage: Similar ao método anterior, porém, o objetivo deste é habilitar as mensagens que se deseja receber do simulador. Nesse caso, não é permitido passar o "*" como parâmetro, sendo necessário informar a identificação de cada mensagem.

startReceiving: Este método habilita a *thread* que recebe os dados do X-Plane®. A *thread* executará até que o método *stopReceiving* seja chamado.

getValue: Este método obtém o valor de um determinado parâmetro do simulador. Recebe como parâmetro a *string* que o representa.

setValue: Este método altera um determinado parâmetro no simulador. Recebe como parâmetro uma *string* que representa o parâmetro a ser alterado e o

valor. A *string* recebida neste método e no anterior (*getValue*) deve ter sido antes configurada no arquivo XML que será descrito abaixo.

Portanto, apenas com os métodos *getValue* e *setValue* é possível obter e alterar vários parâmetros do X-Plane®. Isso torna a utilização do X-Pi rápida e intuitiva.

O arquivo XML passado como parâmetro no construtor da classe *XPlaneInterface* (*DATAGroupConfig.xml*) possui a configuração de todas as mensagens que serão recebidas do simulador. É neste arquivo que está configurado que o nível de combustível do primeiro tanque deve ser obtido e alterado utilizando a string *fuel.fuel1*. Vale ressaltar que, no X-Plane®, o dado não é alterado ou obtido dessa maneira. No simulador, a mensagem que corresponde ao nível de combustível dos tanques é a 62 e o primeiro tanque de combustível é representando pelo primeiro parâmetro.

É devido a isso, que a comunidade de usuários do X-Plane® não é muito adepta a essa forma nativa de comunicação com o X-Plane®. A maioria utiliza o SDK. A alegação principal é que não existem garantias que as mensagens e parâmetros serão mantidos nas respectivas posições e que, por isso, as aplicações necessitam de ser constantemente modificadas. Esse é um argumento válido, entretanto, utilizando a abordagem do arquivo XML desenvolvida no X-Pi, esse problema é minimizado, já que essa forma de identificação nativa é totalmente abstraída (substituída por nomes), sendo assim, qualquer alteração deve ser feita em apenas um lugar (no próprio XML). Além disso, o SDK possui uma curva de aprendizado muito acentuada e quando o problema é apenas obter e alterar os principais dados do voo, não justifica toda a complexidade contida no SDK, já que a interface nativa via UDP resolve bem.

A Figura 5.5 mostra como é realizada a configuração das mensagens e dos parâmetros do piloto automático no arquivo XML. Note que um grupo de dados (*DATAGroup*) pode conter parâmetros de várias mensagens distintas. Essa é uma forma de abstrair ainda mais a forma como o X-Plane® organiza as mensagens.

A biblioteca pode ou não ser executada na mesma máquina do simulador. De qualquer maneira, o X-Plane® precisa ser configurado para enviar os dados para o endereço IP e porta onde está sendo executada a biblioteca. Um outro parâmetro também importante é a porta que o X-Plane® receberá os dados enviados pelo X-Pi. Esses parâmetros são passados no construtor da classe *XPlaneInterface*, como pode ser observado na Figura 5.4.

```
<DATAGroup name="autopilot">
  <Entries>
    <DATA name="fdirMode" message="108" parameter="1"></DATA>
    <DATA name="navArm" message="116" parameter="0"></DATA>
    <DATA name="altArm" message="116" parameter="1"></DATA>
    <DATA name="autoThrottle" message="117" parameter="0"></DATA>
    <DATA name="modeHeading" message="117" parameter="1"></DATA>
    <DATA name="modeAlt" message="117" parameter="2"></DATA>
    <DATA name="speed" message="118" parameter="0"></DATA>
    <DATA name="heading" message="118" parameter="1"></DATA>
    <DATA name="vvi" message="118" parameter="2"></DATA>
    <DATA name="dialAlt" message="118" parameter="3"></DATA>
    <DATA name="useAlt" message="118" parameter="5"></DATA>
  </Entries>
</DATAGroup>
```

Figura 5.5. Configuração das mensagens e dos parâmetros do piloto automático do X-Plane® no arquivo de configurações do X-Pi.

5.3 O piloto automático do X-Plane®

O piloto automático do X-Plane® foi utilizado para realizar a navegação autônoma dos veículos. Essa foi uma alternativa interessante e se mostrou muito adequada para o contexto deste trabalho. Essa abordagem evita o desenvolvimento de algoritmos específicos para o controle da altitude, velocidade e direção da aeronave. Com isso, mais tempo foi concentrado no tema da pesquisa. Entretanto, foi necessário entender como o piloto automático do X-Plane® funciona e quais são os parâmetros que devem ser combinados para obter o comportamento desejado. A Figura 5.6 mostra os controles do piloto automático do avião *Cirrus Jet*.

Figura 5.6. Piloto automático do avião *Cirrus Jet* no X-Plane®.

Como pode ser observado na Figura 5.6, existem várias opções em um piloto automático. Algumas delas podem ser combinadas para que um determinado comportamento seja seguido. Para este trabalho, não foram utilizadas todas as opções, já que muitas delas não são úteis para as situações aqui abordadas. As que foram utilizadas neste trabalho são listadas abaixo.

Altitude: Neste modo, o piloto automático mantém a altitude atual do avião. Caso seja selecionada uma altitude diferente da atual, o piloto automático tentará atingir a nova altitude. Entretanto, só iniciará a subida ou descida após ser

selecionado o comportamento que a aeronave deverá assumir. Uma nova altitude pode ser atingida por meio de uma velocidade constante, de uma razão de subida ou descida constantes ou ambos.

Velocidade: Neste modo, o piloto automático mantém uma determinada velocidade. Isso pode ser feito por dois comportamentos. O primeiro é manter a velocidade controlando a potência do motor. O segundo é manter a velocidade controlando o ângulo de arfagem (*pitch*). Em um voo de cruzeiro, o piloto automático mantém a velocidade controlando a potência. Já em um voo de subida ou descida pode ser utilizada uma das duas alternativas.

Direção: Neste modo, o piloto automático mantém uma determinada direção.

Razão de Subida ou Descida: Neste modo, o piloto automático mantém uma determinada razão de descida ou subida até que a altitude desejada seja alcançada pela aeronave.

Velocidade e Razão de Subida ou Descida: Já neste modo, o piloto automático além de manter uma velocidade fixa, também mantém uma razão de subida ou descida. Esse comportamento é mantido até que a altitude desejada seja alcançada.

Figura 5.7. Piloto automático do avião *Cirrus Jet* mantendo velocidade de 200 knots, altitude de 18.000 ft e direção de 62 graus.

A Figura 5.7 mostra o piloto automático do avião *Cirrus Jet* mantendo velocidade constante de 200 knots, altitude de 18.000 ft e direção de 62°. A figura mostra ainda as três mensagens com todos os parâmetros do piloto automático. Como visto na Seção 5.2.2, é por meio dessas mensagens que a biblioteca X-Pi altera as informações no X-Plane®.

A Figura 5.8 mostra como alterar, via X-Pi, o piloto automático de tal maneira que ele se comporte como na Figura 5.7.

```
xpi.setValue "autopilot.speed", 200.0f
xpi.setValue "autopilot.heading", 62.0f
xpi.setValue "autopilot.vvi", 0.0f
xpi.setValue "autopilot.altArm", 0.0f
xpi.setValue "autopilot.modeAlt", 6.0f
xpi.setValue "autopilot.dialAlt", 18000.0f
xpi.setValue "autopilot.useAlt", 18000.0f
xpi.setValue "autopilot.autoThrottle", 1.0f
xpi.setValue "autopilot.modeHeading", 1.0f
xpi.setValue "autopilot.fdirMode", 2.0f
```

Figura 5.8. Alterando o piloto automático do X-Plane® com a biblioteca X-Pi. Código escrito na linguagem Groovy. Alguns detalhes foram omitidos para melhor clareza.

Uma característica interessante do piloto automático do X-Plane® é que mesmo não havendo possibilidades de configurar determinado comportamento pelo painel da aeronave, é possível fazê-lo utilizando a sua API. Um exemplo é o avião Cessna 172SP. Este avião não permite que seja configurado pelo painel uma determinada velocidade a ser mantida. Entretanto, por meio da X-Pi os parâmetros do piloto automático são alterados de tal maneira a permitir esse comportamento.

Capítulo 6

Resultados

Neste capítulo são apresentados os experimentos que foram realizados com as duas missões propostas. A primeira seção menciona o planejador PDDL que foi utilizado nos experimentos. Já a Seção 6.2, detalha o *hardware* utilizado nos experimentos. Depois disso, a Seção 6.3, apresenta os resultados obtidos com a missão de navegação entre *waypoints*. Cada uma das características (velocidade, distância e altitude) que geraram os domínios desenvolvidos no Capítulo 4 são experimentadas aqui. A última seção, 6.4, dedica-se a detalhar os resultados obtidos com a missão de combate a incêndios florestais.

6.1 Planejador LPG-td

O planejador PDDL escolhido para realização dos experimentos foi o *LPG-td* [Gerevini et al., 2004]. O LPG-td consegue lidar com a versão 2.2 da PDDL e é uma extensão do planejador LPG. O seu funcionamento é baseado em busca estocástica local em um grafo de ações. Esse grafo é derivado diretamente do problema de planejamento descrito pelos arquivos de domínio e problema. A versão utilizada não foi a encontrada no site, já que a mesma teve dificuldades para lidar com os domínios propostos neste trabalho. Os autores gentilmente cederam uma nova versão que obteve um desempenho muito superior e conseguiu resolver certas instâncias que não eram resolvidas pela versão antiga.

Ao longo do trabalho procurou-se utilizar outros planejadores como o *MIPS-XXL* [Edelkamp et al., 2006] e o *SGPLAN 6* [Hsu et al., 2006], entretanto, o desempenho obtido com esses não foi satisfatório para os domínios tratados. O MIPS muitas vezes não retornava um plano. Já o SGPLAN, retornava planos de baixa qualidade (duração muito alta).

6.2 *Hardware* utilizado

Dois computadores foram utilizados para realizar os experimentos. O computador com a placa de vídeo de maior capacidade, denominado Computador 1 na Tabela 6.1, foi responsável por executar o X-Plane® (versão 9.55). Já o Computador 2, detalhado na mesma tabela, foi responsável por executar o código Java desenvolvido no arcabouço e o planejador PDDL *LPG-td*. A comunicação entre o Computador 2 e o Computador 1 era realizada por meio do protocolo UDP (em uma rede sem fio) utilizando a biblioteca X-Pi (ver Seção 5.2.2).

	Computador 1	Computador 2
Processador	Athlon X2 Dual Core 5200+	Intel Core 2 Duo T5250
Placa de Vídeo	Nvidia GeForce 9800 GT	Intel 965GM
Memória RAM	4 GB	2 GB
Sistema Operacional	Ubuntu 9.04 64 bits	Ubuntu 9.04 32 bits

Tabela 6.1. *Hardware* utilizado nos experimentos.

Na missão de navegação entre *waypoints*, ambos os computadores foram utilizados, já que, para esse caso, o plano deveria ser executado no simulador de voo X-Plane®. Já na missão de combate a incêndios floretais, utilizou-se somente o Computador 2.

6.3 Missão de Navegação entre *waypoints*

Nesta seção, serão demonstrados os experimentos realizados com a missão de navegação entre *waypoints*. Primeiro, será descrita a metodologia experimental utilizada. Logo após, serão apresentados os resultados de cada característica (velocidade, distância e altitude) abordada no Capítulo 4.

6.3.1 Metodologia Experimental

Por ser uma missão de navegação entre *waypoints*, as ações de decolagem e pouso não foram consideradas. Todo experimento tinha início com o avião já em voo. A sequência de *waypoints* a ser visitada era retornada pelo planejador PDDL. Entretanto, para essa missão, a sequência tinha sempre a mesma característica. O veículo iniciava no WP0, passava pelos WP1, WP2, WP3 e assim por diante, até chegar no último *waypoint* a ser visitado. Para a PDDL sempre gerar a mesma sequência,

utilizou-se o predicado (*reachable ?l1 ?l2*)[1] no arquivo de problema como mostrado abaixo (considerando cinco *waypoints*):

```
(reachable WP0 WP1)
(reachable WP1 WP2)
(reachable WP2 WP3)
(reachable WP3 WP4)
```

Dessa maneira, poderia-se argumentar o motivo de estar utilizando PDDL para esta missão, já que a sequência de *waypoints* é sempre a mesma, ou seja, não existe uma tomada de decisão a respeito da ordem em que os *waypoints* devem ser visitados. Deve-se notar, entretanto, que o objetivo da missão de navegação entre *waypoints* é compreender quais são os aspectos essenciais em um planejamento de missão para veículos aéreos quando se considera características como velocidade, distância, altitude e consumo de combustível e, baseado nisso, tentar construir domínios em PDDL que consigam refletir integralmente ou parcialmente essas características. A partir daí, esse modelo pode ser utilizado em outros tipos de missão onde a PDDL será utilizada efetivamente para tomada de decisão.

Cada característica foi experimentada fixando a altitude, a direção ou ambos. Por exemplo: a característica velocidade não considerou mudança de altitude ou mudança de direção durante a execução do experimento. Isso significa que um experimento era iniciado em uma altitude e direção determinadas e nelas o veículo permanecia até o fim. A Tabela 6.2 mostra uma visão geral de cada domínio experimentado, incluindo a velocidade horizontal utilizada, se era ou não utilizado o fator de ajuste para a distância e a quantidade de ações PDDL que foram utilizadas para representar o deslocamento do veículo. Será possível observar mais adiante que vários experimentos foram realizados ou iniciavam na altitude de 10 ft. Essa altitude foi escolhida, pois não era possível fazer o avião voar exatamente no nível do mar (a 0 ft), já que ele poderia colidir com a água.

Nas tabelas que mostram a diferença percentual entre o planejado e o realizado, convencionou-se que, uma diferença maior do que 5% (negativa ou positiva), será destacada em negrito, indicando, assim, que houve uma divergência significativa entre o planejado e o realizado.

[1]Conforme mostrado na Tabela 4.6, o predicado *reachable* indica se o local l2 é alcançável a partir do local l1.

Domínio	*Característica Modelada*	*Ações PDDL*	*Velocidade Horizontal*	*Fator Ajuste Distância*	*Mudança Altitude*	*Mudança Direção*
A1	Velocidade	1	IAS	Não	Não	Não
A2	Velocidade	1	TAS	Não	Não	Não
B1	Distância	1	TAS	Não	Não	Sim
B2	Distância	1	TAS	Sim	Não	Sim
C1	Altitude	1	TAS	Sim	Sim	Não
C2	Altitude	3	TAS	Sim	Sim	Não

Tabela 6.2. Visão geral de cada domínio experimentado.

6.3.1.1 Aeronave selecionada

O avião selecionado para os experimentos foi o *Cesnna 172SP 180HP*, mostrado anteriormente na Figura 5.2. As velocidades horizontais e verticais definidas por fase de voo podem ser observadas na Tabela 6.3.

Fase de Voo	IAS knots	VV ft/min
Subida	75	500
Cruzeiro	100	0
Descida	100	-700

Tabela 6.3. Configuração das fases de voo para o avião *Cesnna 172SP* na missão de navegação entre *waypoints*.

6.3.1.2 Cálculo da taxa de consumo de combustível

Nos domínios A1, A2, B1, B2 e C1, apenas uma taxa de consumo de combustível pode ser considerada. Já no domínio C2, pode ser considerada uma taxa por fase de voo. O valor escolhido para o planejamento é sempre independente da altitude voada e das condições atmosféricas. Para descobrir qual taxa utilizar para o avião *Cessna 172SP*, foram realizados três experimentos, um para cada fase. Cada experimento foi conduzido da seguinte maneira: a aeronave era configurada em uma determinada altitude e em regime de voo de subida, descida ou cruzeiro. Após estabilizada nessa configuração, a taxa de consumo de combustível (em lb/h) era coletada. O Gráfico 6.1 mostra os valores que foram coletados por fase de voo e

altitude[2].

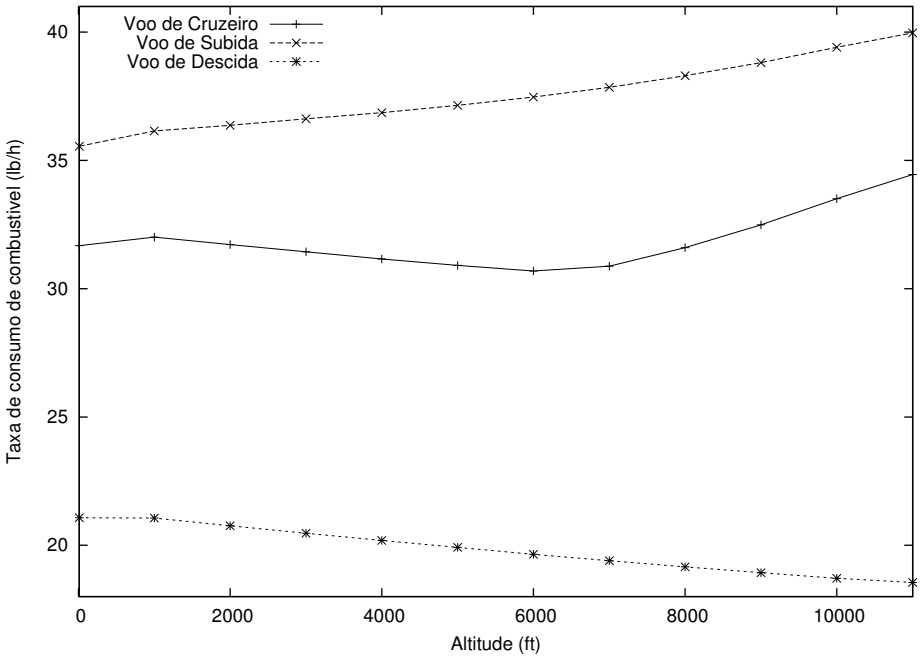

Figura 6.1. Taxas de consumo de combustível do avião *Cessan 172SP 180HP* por fase de voo e altitude.

Portanto, a questão é definir, baseado nestes dados, qual será o valor planejado para a taxa de consumo de combustível por fase de voo. Algumas estratégias são:

- Considerar o valor mais alto. No caso da fase de voo de cruzerio, seria o valor de $34,45$ lb/h;
- Considerar uma média aritmética. No caso do voo de subida, o valor seria de $37,54$ lb/h;

Se apenas um valor puder ser considerado, independente da fase de voo, então uma estratégia conservadora é considerar o consumo mais alto encontrado no gráfico, nesse caso, $39,97$ lb/h. Portanto, não importa se o avião voará metade do tempo em regime de voo de cruzeiro a 2.000 ft e a outra metade subindo e descendo entre as altitudes de 2.000 ft e 10.000 ft, a taxa planejada será sempre a mesma, $39,97$ lb/h.

Deve-se ter em mente, entretanto, que independente da alternativa adotada, o valor escolhido (planejado) é uma aproximação e que, dependendo das condições

[2]Para facilitar a obtenção desses valores, o controle da mistura ar-combustível do *Cessna 172SP* foi modificado para ser realizado de maneira automática. Para isso, modificou-se a aeronave utilizando a ferramenta *PlaneMaker* do X-Plane®.

atmosféricas e do peso da aeronave, pode divergir bastante do valor efetivamente realizado. Sendo assim, um valor conservador se torna uma opção mais interessante.

6.3.1.3 Cálculo do fator de correção para a TAS

Como explicado na Seção 4.2.3, o cálculo da TAS é baseado na IAS, na altitude de voo e em um fator de correção. Geralmente, esse fator é considerado como 2%, entretanto, para esses experimentos, um fator de 1,5% foi mais adequado. Para encontrar este valor, realizou-se um experimento onde o avião *Cessna 172SP* voava em regime de voo de cruzeiro por diversas altitudes. Para cada altitude foi coletada a IAS e a TAS e, a partir dos resultados, foi calculado o fator de correção para cada altitude voada. Para esse cálculo, utilizou-se a seguinte equação (baseada na Equação 4.2):

$$fator = 1.000 \times \frac{TAS - IAS}{IAS \times h}. \tag{6.1}$$

A Figura 6.2 mostra os valores coletados nos experimentos e os valores calculados a partir desses dados. As linhas *IAS* e *TAS Real* representam os valores coletados. Já as linhas TAS 2% e TAS 1,5% representam o cálculo da TAS considerando os fatores de correção de 2% e 1,5% respectivamente. Note que, ao considerar o fator de 1,5%, o resultado se aproximou melhor da TAS real.

Para encontrar o fator de correção de 1,5%, foi calculada a média aritmética do fator para cada altitude, desconsiderando o valor do fator para a altitude de 10 ft (este valor destoa significativamente dos demais). A média obtida foi de 1,51%, mas considerou-se 1,5%.

6.3.2 Característica Velocidade

O objetivo da característica velocidade foi verificar qual a diferença entre utilizar a IAS (domínio A1) ou a TAS (domínio A2) como velocidade planejada. A seguinte metodologia experimental foi seguida: primeiro era definida uma determinada altitude onde a missão seria realizada. Feito isso, dois experimentos eram conduzidos, um para cada domínio. As altitudes selecionadas foram 10 ft, 5.000 ft e 10.000 ft. Como dito anteriormente, após iniciado o experimento, não havia mudança de altitude ou direção. O avião voava durante todo o tempo em regime de voo de cruzeiro. A sequência era composta por onze *waypoints* cada um a uma distância de 4 nm do seu antecessor. O avião iniciava a missão no WP0 e terminava no WP10, sendo, portanto, executadas dez ações de deslocamento.

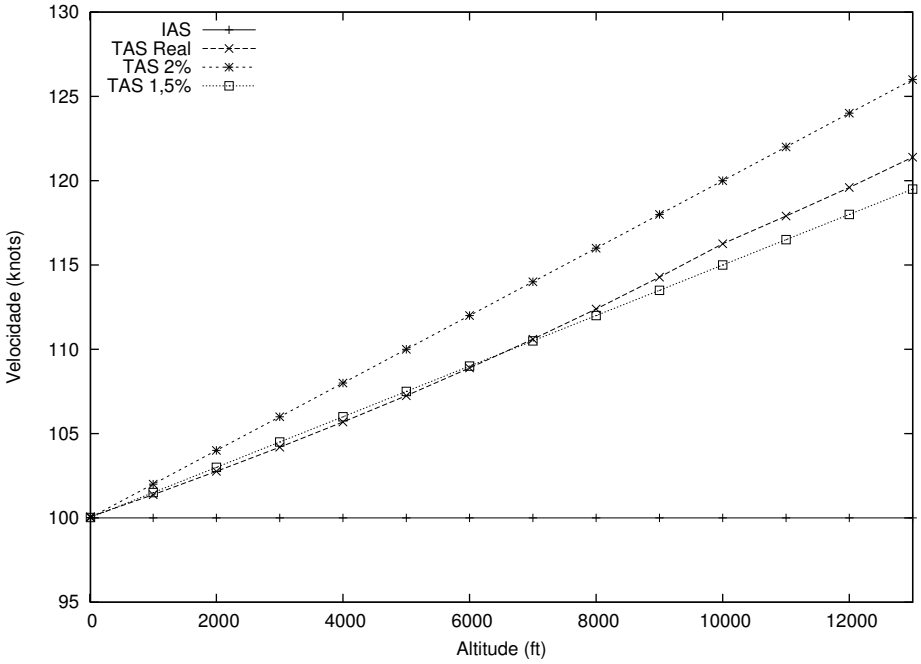

Figura 6.2. Comparativo entre utilizar um fator de correção de 1,5% e 2% para a TAS.

A Tabela 6.4 mostra os valores planejados (por parâmetro) para os domínios A1 e A2. Cada valor é a média aritmética obtida utilizando os valores das dez ações de deslocamento. Note que no domínio A1, os parâmetros duração, combustível e distância não se alteram com o incremento da altitude. Já no domínio A2, como a TAS é utilizada, os valores mudam devido a correção que é realizada na velocidade. O único parâmetro que permanece constante nos dois domínios é a distância planejada, já que essa independe da velocidade. Como a aeronave voava todo o tempo em voo de cruzeiro ficou mais simples definir a taxa de consumo de combustível. Para isso, verificou-se no Gráfico 6.1, o valor para a respectiva altitude. Por essa razão, a 10.000 ft, a taxa escolhida foi de 35,00 lb/h.

A Tabela 6.5 mostra os valores que foram realizados por domínio. Os valores são praticamente os mesmos, pois os experimentos foram realizados nas mesmas condições. Este é um fato interessante e mostra que a estratégia experimental adotada poderia ter sido diferente. O mais interessante seria realizar o experimento no simulador uma única vez por altitude e utilizar os dados coletados em várias instâncias de planos, comparando assim, o mesmo valor realizado com vários planejados. Por exemplo, na altitude de 5.000 ft a execução no simulador seria realizada apenas uma vez e os valores obtidos seriam comparados com o planejado do domínio A1 e A2. Entende-se, entretanto, que isso não prejudica a metodologia

Altitude	10 ft		5.000 ft		10.000 ft	
Domínio	*A1*	*A2*	*A1*	*A2*	*A1*	*A2*
Duração (s)	143,84	143,82	143,84	133,81	143,84	125,08
Combustível (lb)	1,28	1,28	1,28	1,19	1,40	1,22
Distância (nm)	4,00	4,00	4,00	4,00	4,00	4,00
Velocidade (knots)	100,00	100,02	100,00	107,50	100,00	115,00
Taxa de Consumo (lb/h)	32,00	32,00	31,00	31,00	35,00	35,00

Tabela 6.4. Valores planejados para os domínios A1 e A2 (característica velocidade).

experimental, já que os valores das execuções são praticamente iguais.

Altitude	10 ft		5.000 ft		10.000 ft	
Domínio	*A1*	*A2*	*A1*	*A2*	*A1*	*A2*
Duração (s)	143,93	143,92	134,32	134,29	123,93	123,95
Combustível (lb)	1,29	1,29	1,18	1,19	1,22	1,23
Distância (nm)	3,99	3,99	3,99	3,99	3,99	3,99
Velocidade (knots)	100,07	100,07	107,25	107,26	116,26	116,26
Taxa de Consumo (lb/h)	32,15	32,15	31,72	31,89	35,49	35,65

Tabela 6.5. Valores realizados para os domínios A1 e A2 (característica velocidade).

Note que, apesar dos valores expressos nas Tabelas 6.4 e 6.5 representarem a média aritmética, isso não prejudica a confiança nesses dados, já que as dez ações são planejadas e executadas com a mesma altitude, velocidade, distância e taxa de consumo de combustível.

Finalmente, a Tabela 6.6 mostra a diferença percentual entre o planejado e o realizado para os domínios A1 e A2. Note que, na altitude de 10 ft, praticamente não há diferença entre os domínios para nenhum dos parâmetros considerados. Isso aconteceu, pois o avião voou quase no nível do mar e em condições ISA. Nessa situação, a IAS se aproxima muito da TAS (ver Seção 4.2.3).

Já em altitudes maiores como 5.000 ft e 10.000 ft, a diferença começa a ficar significativa para os parâmetros duração, combustível e velocidade quando se usa a IAS para o planejamento. No caso do parâmetro duração a 10.000 ft, a diferença foi de $-13,84\%$ se utilizada a IAS. Isso significa que, em média, cada ação foi $13,84\%$ mais rápida que o planejado. Por outro lado, se a TAS for considerada, então as

diferenças ficam praticamente inexistentes. A diferença do combustível utilizado em média por ação foi de apenas 0,93% a 10.000 ft (quando utilizada a TAS).

Portanto, utilizar a TAS como velocidade horizontal planejada pode fazer uma diferença significativa na qualidade do plano. Devido a isso, todos os domínios desenvolvidos posteriormente utilizaram a TAS.

Altitude	10 ft		5.000 ft		10.000 ft	
Domínio	*A1*	*A2*	*A1*	*A2*	*A1*	*A2*
Duração	0,07	0,07	**-6,62**	0,36	**-13,84**	-0,90
Combustível	0,56	0,58	**-7,72**	-0,31	**-12,93**	0,93
Distância	-0,14	-0,16	-0,14	-0,17	-0,20	-0,14
Velocidade	0,07	0,06	**7,25**	-0,22	**16,26**	1,10
Taxa de Consumo	0,47	0,48	2,32	2,87	1,40	1,87

Tabela 6.6. Diferença percentual entre o planejado e o realizado para os domínios A1 e A2 (característica velocidade).

6.3.3 Característica Distância

O objetivo dos experimentos com a característica distância foi verificar qual a diferença encontrada na qualidade do plano se o fator de ajuste para a distância for ou não considerado (ver Seção 4.4.3).

Para isso, os onze *waypoints* da missão foram separados em dois grupos, os pares e os ímpares, como exibido na Figura 6.3. Os *waypoints* pares são posicionados todos no mesmo lugar e a 180° dos ímpares e vice-versa. Os experimentos foram realizados a uma altitude de 10.000 ft e a distância entre os grupos foi de 4 nm e 20 nm. A sequência a ser seguida pelo veículo tinha início no WP0 e terminava no WP10.

O objetivo de posicionar os *waypoints* dessa maneira foi considerar o pior caso para a característica distância que acontece quando a diferença angular entre a direção atual do veículo e a necessária para ele seguir para o próximo *waypoint* é próxima de 180°. Assim, como a distância planejada é o menor caminho entre os *waypoints*, espera-se que a diferença entre o planejado e o realizado seja significativa caso o fator de ajuste não for considerado.

Para o avião *Cessna 172SP*, o fator de ajuste foi calculado como sendo 2 nm (a metodologia para o cáclulo pode ser encontrada em 4.4.3). Lembrando que o fator de ajuste é somado à distância planejada no momento do planejamento.

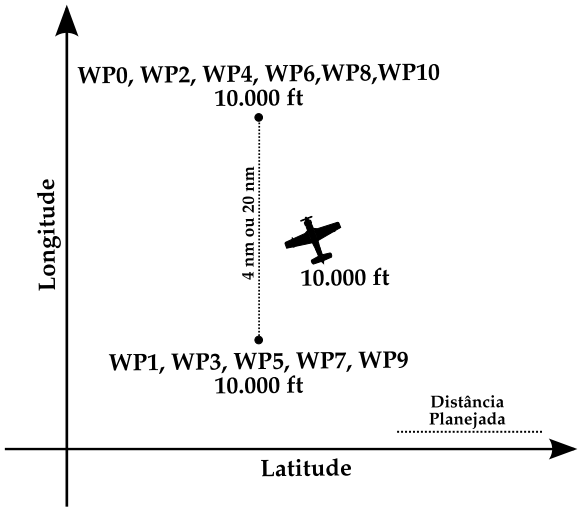

Figura 6.3. Representação visual da missão de navegação entre *waypoints* para os experimentos com a característica distância.

A Tabela 6.7 mostra a diferença percentual entre o planejado e o realizado para os domínios B1 e B2. O domínio B1 não considera o fator de ajuste para a distância, já o domínio B2 considera. Como pode ser observado, utilizar o fator de ajuste produz uma diferença considerável na qualidade do plano. A diferença para o parâmetro duração no domínio B1 foi de $49,07\%$, isso significa que, em média, cada ação gastou $49,07\%$ a mais de tempo para ser realizada do que o planejado. Porém, quando utilizado o fator de ajuste, essa diferença caiu para $-0,61\%$.

Um fato interessante é que à medida que a distância entre os *waypoints* aumenta, a diferença diminui, mesmo quando o fator não é utilizado. A explicação para isso é que a distância percorrida durante a curva fica diluída na distância percorrida durante o tempo em que o avião voa em linha reta. A diferença para o combustível gasto considerando uma distância de 4 nm foi de $51,15\%$, porém, ao aumentar a distância para 20 nm, essa diferença caiu para $10,48\%$. Entretanto, note que mesmo nessa situação o fator foi importante e a diferença caiu para $0,47\%$ quando ele foi considerado.

Enfim, utilizar o fator foi benéfico, mas deve-se levar em consideração dois aspectos. O primeiro é que essa configuração de posicionamento dos *waypoints* é muito específica e nem sempre eles estarão dispostos dessa maneira e assim, a diferença para a distância em linha reta pode não ser tão significativa. O segundo aspecto é que se a distância entre eles for muito maior do que o fator, então não faz tanta diferença utilizá-lo.

Distância	4 nm		20 nm	
Domínio	*B1*	*B2*	*B1*	*B2*
Duração	**49,07**	-0,61	**10,02**	0,06
Combustível	**51,15**	0,79	**10,48**	0,47
Distância	**50,40**	0,28	**11,23**	1,14
Velocidade	1,11	1,11	1,12	1,12
Taxa de Consumo	1,43	1,43	0,41	0,41

Tabela 6.7. Diferença percentual entre o planejado e o realizado para os domínios B1 e B2 (característica distância).

6.3.4 Característica Altitude

Diferentemente dos experimentos anteriores com as características velocidade e distância, neste caso, acontece mudança de altitude durante a execução do plano. Sendo assim, em cada ação de deslocamento de um *waypoint* para outro, poderão acontecer duas fases de voo (no máximo). Se os *waypoints* de origem e destino estiverem em altitudes distintas, então será executada uma fase de voo de subida (ou descida) e outra de cruzeiro. Se os *waypoints* estiverem na mesma altitude então será executada apenas uma fase de voo de cruzeiro.

Na Tabela 6.3, podem ser observadas as velocidades horizontais e verticais que serão planejadas e realizadas por fase de voo. Como visto anteriormente na Seção 4.4.4, o domínio C1 permite especificar apenas uma velocidade horizontal e uma taxa de consumo de combustível para todo o planejamento, independente das fases de voo que serão efetivamente executadas pelo veículo. Note também que para este domínio a velocidade vertical não é utilizada no planejamento, apenas na execução. Já o domínio C2 permite que sejam planejadas velocidades horizontais e verticais além de taxas de consumo de combustível distintas por fase de voo.

A seguinte metodologia experimental foi utilizada: o veículo iniciava a missão no WP0 com altitude de 10 ft e terminava no WP10 com altitude de 10.000 ft. Portanto, o veículo subia 1.000 ft a cada *waypoint* visitado. A missão era realizada sem mudança de direção e a distância entre cada *waypoint* foi fixada em 4 nm para um experimento e em 20 nm para outro.

Para esta característica, a escolha da taxa de consumo não é tão simples, pois o veículo voará em altitudes diferentes durante a missão. No caso do domínio C1, a taxa de consumo de combustível foi considerada como $40,00$ lb/h para toda a missão. Já para o domínio C2, foram consideradas as taxas de $40,00$ lb/h e $35,00$ lb/h para as fases de voo de subida e cruzeiro respectivamente. Note

que a escolha para ambos os domínios adota uma estratégia conservadora para esse parâmetro (ver Gráfico 6.1). O voo de descida não foi considerado nesses experimentos.

Para exemplificar a diferença entre utilizar os domínios C1 e C2, será exibido abaixo os dados do planejamento e execução do trecho de voo em que o veículo se deslocou do WP4 (altitude de 4.000 ft) para o WP5 (altitude de 5.000 ft).

A Tabela 6.8 mostra os dados do planejado e realizado para o domínio C2. Note que os dados são exibidos por fase de voo, pois neste domínio existe uma ação para cada fase. Além disso, na coluna total está a soma das duas fases para os parâmetros duração, combustível e distância e a média para os parâmetros velocidade e taxa de consumo de combustível. É interessante ressaltar que os valores realizados para a velocidade e taxa de consumo foram obtidos do simulador por amostragem e representam a média aritmética das amostras coletadas[3].

	Planejado			*Realizado*		
	Subida	*Cruzeiro*	*Total*	*Subida*	*Cruzeiro*	*Total*
Duração (s)	120,00	44,40	164,40	117,79	48,27	166,06
Combustível (lb)	1,20	0,42	1,62	1,13	0,55	1,68
Distância (nm)	2,67	1,33	4,00	2,67	1,34	4,01
Velocidade (knots)	80,06	107,50	93,78	80,96	100,34	90,65
Taxa de Consumo (lb/h)	40,00	35,00	37,50	34,73	41,37	38,05

Tabela 6.8. Domínio C2: Valores planejados e realizados para o deslocamento do WP4 para o WP5.

Já a Tabela 6.9 mostra a diferença percentual entre o planejado e o realizado por fase de voo e para a coluna total. Por fase de voo, a diferença em alguns parâmetros é significativa, principalmente para os parâmetros que tem relação com o combustível. No voo de cruzeiro, por exemplo, a taxa de consumo de combustível apresentou uma diferença de 18,20%, sendo o valor planejado 35,00 lb e o realizado 41,37 lb.

Essa diferença ocorreu, pois, ao terminar o voo de subida, o veículo deve passar da velocidade (IAS) de 75 knots para 100 knots e isso acontece de maneira contínua (diferente do planejado onde a mudança é discreta). Sendo assim, o veículo voa um trecho da ação de voo de cruzeiro com velocidades entre 75 knots e 100 knots. Por

[3]O Módulo de Execução do arcabouço experimental permite que amostras sejam coletadas em intervalos de tempo especificados. Para este experimento, os valores da velocidade e da taxa de consumo de combustível foram coletados de cinco em cinco segundos.

	Subida	*Cruzeiro*	*Total*
Duração	-1,84	**8,72**	1,01
Combustível	**-5,86**	**31,05**	3,71
Distância	-0,13	0,75	0,16
Velocidade	1,13	**-6,66**	-3,34
Taxa Consumo	**-13,18**	**18,20**	1,47

Tabela 6.9. Domínio C2: Diferença percentual por fase de voo e total para o deslocamento do WP4 para o WP5.

sua vez, o piloto automático do X-Plane® deve fornecer potência ao motor para que a IAS de 100 knots seja atingida o mais breve possível. Isso significa que, durante esse tempo, o consumo de combustível ficará naturalmente acima do planejado. O problema acontece porque a duração da ação de cruzeiro é de apenas 48, 27 s e uma parte significativa desse tempo é realizada com consumo e velocidade diferentes do planejado. Também por esse motivo, a diferença do combustível gasto foi de 31, 05%.

Entretanto, é interessante notar que, se for considerada a diferença percentual total, então nenhum dos parâmetros apresenta um valor significativo. Isso aconteceu, pois uma ação compensou a outra. Por exemplo, o planejado e o realizado para a duração no voo de subida foi respectivamente de 120, 00 s e 117, 79 s e para o voo de cruzeiro foi de 44, 40 s e 48, 27 s.

A Tabela 6.10 exibe os dados do planejado, realizado e a diferença percentual para o domínio C1. Note que, com exceção do parâmetro distância, os demais obtiveram uma diferença percentual significativa. O parâmetro velocidade teve uma diferença de $-18,74\%$. Isso aconteceu porque a maior parte do deslocamento foi realizada em fase de voo de subida com velocidade (IAS) de 75 knots. Entretanto, no planejamento foi considerada uma velocidade (IAS) de 100 knots. Este comportamento prejudicou diretamente a duração (22, 95%) que, por sua vez, prejudicou o consumo de combustível (mesmo sendo a taxa de consumo de combustível realizada menor do que a planejada).

Os resultados acima mostram que a qualidade do plano gerado pelo domínio C1 é significativamente pior do que a do domínio C2 e, assim, é razoável imaginar que considerar as fases de voo pode aumentar a qualidade do plano. Entretanto, a Tabela 6.11 mostra que dependendo da situação, não existe tanta diferença em utilizá-las no planejamento.

Esta tabela mostra o resultado para toda a missão. Para a distância de 4 nm

	Planejado	*Realizado*	*Diferença*
Duração (s)	135,00	165,98	**22,95**%
Combustível (lb)	1,48	1,69	**14,19**%
Distância (nm)	4,00	4,00	0,00%
Velocidade (knots)	106,75	86,74	**-18,74**%
Taxa de Consumo (lb/h)	40,00	36,59	**-8,53**%

Tabela 6.10. Domínio C1: Valores planejados, realizados e a diferença percentual para a ação de deslocamento do WP4 para o WP5.

era esperado que o domínio C1 fosse pior do que o C2, já que o exemplo acima foi retirado do conjunto das ações de deslocamento que geraram esses dados. Porém, para a distância de 20 nm a diferença já não foi tão significativa. Por exemplo, a diferença para o parâmetro duração no domínio C1 foi de $4,13\%$ e no domínio C2 de $-0,90\%$. A explicação para esse comportamento é que o veículo permaneceu na velocidade de 100 knots por mais tempo quando a distância a ser percorrida foi de 20 nm e essa foi a velocidade planejada escolhida. Em outras palavras, pode-se dizer que os valores para o tempo gasto e a distância percorrida no voo de subida (a 75 knots) foram diluídos no tempo e na distância em que o veículo permaneceu em voo de cruzeiro (a 100 knots).

Por outro lado, o combustível gasto e a taxa de consumo apresentaram diferenças negativas e significativas para a distância de 20 nm em ambos os domínios. Isso aconteceu, pois foi adotada uma estratégia conservadora na escolha das taxas.

Enfim, considerar as fases de voo no planejamento é algo que melhora a qualidade do plano, entretanto, se a distância entre os *waypoints* for significativa e o veículo voar a maior parte dessa distância com a velocidade planejada, essa abordagem não produzirá uma diferença tão considerável.

Distância	4 nm		20 nm	
Domínio	*C1*	*C2*	*C1*	*C2*
Duração	**23,25**	0,55	4,13	-0,90
Combustível	**13,15**	-4,27	**-14,69**	**-8,90**
Distância	0,25	0,24	0,16	0,16
Velocidade	**-18,91**	-3,05	-3,83	0,78
Taxa Consumo	**-7,87**	1,88	**-18,01**	**-10,68**

Tabela 6.11. Diferença percentual entre o planejado e o realizado para os domínios C1 e C2 (característica altitude).

6.3.5 Experimento Final

Nos experimentos realizados até aqui, as mudanças de direção e de altitude não eram consideradas juntas na mesma instância (uma ou outra eram fixadas), ver Tabela 6.2. Assim, o objetivo com o experimento final foi configurar e executar uma missão que considerasse ambas as mudanças. Para isso, foram gerados 15 *waypoints* distribuídos aleatoriamente utilizando as seguintes regras:

- a distância entre um par de *waypoints* poderia ser de 25 nm, 35 nm ou 45 nm;
- A altitude de um *waypoint* poderia ser de 1.000 ft, 3.000 ft, 5.000 ft, 7.000 ft ou 9.000 ft;
- A direção para o próximo *waypoint* era selecionada entre um valor que começa de 0° e termina em 330° (incrementado de 30° em 30°).

A Figura 6.4 mostra como ficaram posicionados cada *waypoint*. A informação dentro do parênteses é a altitude do *waypoint*, expressa em ft. O veículo iniciava no WP0 e terminava no WP15.

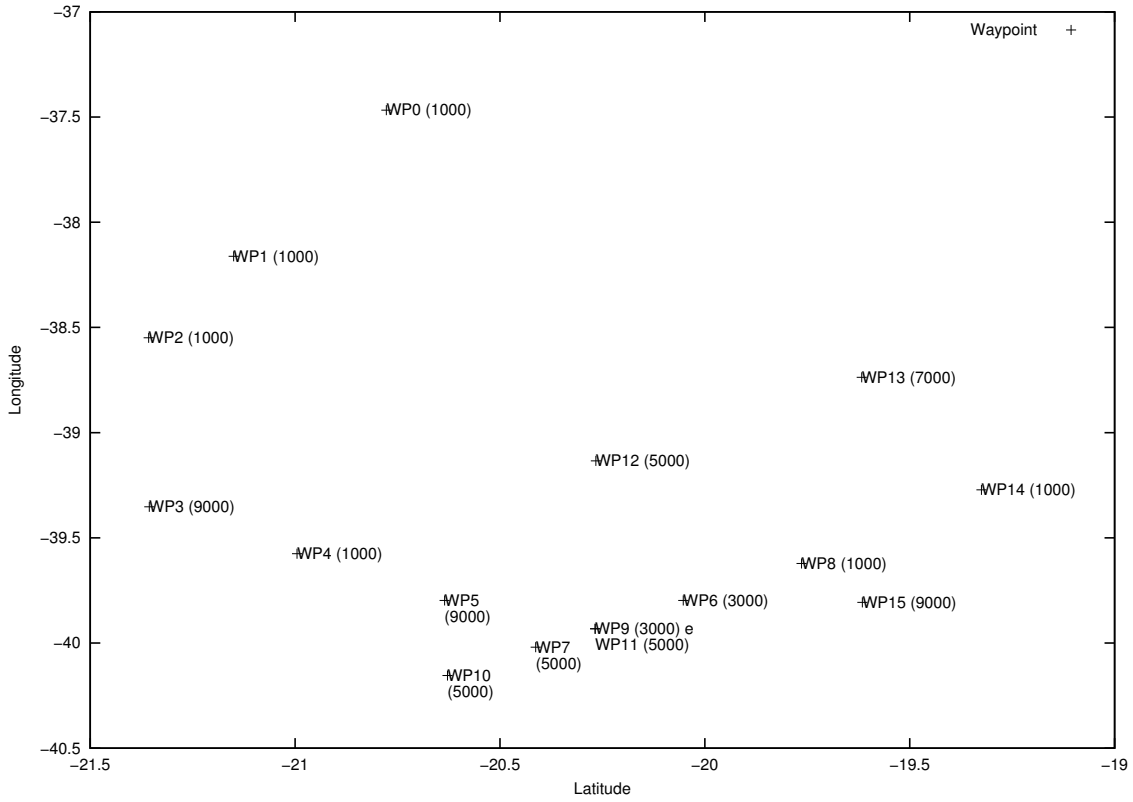

Figura 6.4. Posicionamento de cada *waypoint* para o experimento final.

Note que os WP4 e WP5 foram configurados nas altitudes de 1.000 ft e 9.000 ft respectivamente e a distância entre eles (apesar de não mostrada) é de 25 nm. Isso implica em uma situação onde o veículo percorrerá 21,5 nm em voo de subida e o restante, 3,5 nm, em voo de cruzeiro (cálculos realizados utilizando a metodologia descrita na Seção 4.2.4). Sendo assim, se o domínio C1 for utilizado e a velocidade de cruzeiro (100 knots) for escolhida para o planejamento, então esse trecho da missão terá uma qualidade ruim, já que a maior parte do tempo o veículo voará em voo de subida (75 knots). Para este caso, o domínio C2 geraria um plano de melhor qualidade. De fato, dos 1075 s voados nesse trecho, 959 s foram realizados em regime de voo de subida e o restante, 116 s, em regime de voo de cruzeiro.

Por outro lado, os *waypoints* WP8 e WP9 foram configurados nas altitudes de 1.000 ft e 3.000 ft respectivamente e a uma distância de 35 nm. Nesse caso, o veículo percorrerá apenas 6,87 nm em voo de subida e o restante, 28,13 nm, em voo de cruzeiro. Neste trecho, dos 1317 s voados, 236 s foram em voo de subida e 1081 s em voo de cruzeiro. Nessa situação, como visto anteriormente, tanto faz utilizar o domínio C1 ou C2.

Uma outra questão importante a ser considerada é o fator de ajuste da distância. Por exemplo, para voar em direção ao WP8 a partir do WP7, o veículo teve que fazer uma curva de 177° e isso, como visto anteriormente, produz uma diferença entre o planejado e o realizado. Das 44,95 nm planejadas foram percorridas 46,81 nm, uma diferença de 4,15%.

Sendo assim, para tentar descobrir qual o domínio se adequa melhor a essa missão foram realizados dois experimentos, um para cada domínio. Abaixo estão alguns dados (planejados e realizados) comuns aos dois experimentos. Lembrando que, como os experimentos eram realizados nas mesmas condições, os dados extraídos do simulador são praticamente os mesmos, não prejudicando, portanto, qualquer análise.

Distância percorrida (realizado): 522 nm;
Distância planejada (sem o fator): 514 nm;
Distância planejada (com o fator): 544 nm;
Tempo em voo de cruzeiro (realizado): 12.828 s (69%);
Tempo em voo de subida (realizado): 3.828 s (20%);
Tempo em voo de descida (realizado): 2.034 s (11%).

A Tabela 6.12 mostra os resultados para os parâmetros duração e consumo de combustível. Conforme pode ser observado, a instância que se adequou melhor a essa missão foi a $C2_{s/fator}$, pois gerou apenas 1,56% de diferença para o parâmetro

duração e $-7,95\%$ para o combustível gasto. Vale ressaltar que as diferenças significativas e negativas observadas no parâmetro combustível são frutos de uma escolha conservadora para a taxa de combustível.

Um fato interessante nesse dados é que a instância $C1_{c/fator}$ ficou muito parecida com a $C2_{s/fator}$ para o parâmetro duração, entretanto, a primeira utilizou o fator de correção para a distância e a última não. A razão para isso, é que o planejamento realizado pelo domínio C2 possui uma duração maior, pois considera as fases de voo. Por exemplo, do WP2 ao WP3 o veículo subiu 8.000 ft e percorreu 45 nm. Planejando com o C1, a duração para esse deslocamento é de 1505 s. Porém, ao utilizar o domínio C2, esse tempo passa a ser de 1703 s, sendo 960 s para o voo de subida e 743 s para o voo de cruzeiro. Este acréscimo acontece, pois o trecho de voo de subida é realizado em uma velocidade menor do que o trecho de voo de cruzeiro e isso, consequentemente, faz a duração aumentar.

Porém, no geral, é possível perceber que com exceção da instância $C1_{s/fator}$, as demais geraram planos de qualidade. Assim, uma boa opção para essa missão é o domínio $C1_{c/fator}$, pois, possui um código mais claro e conciso. Além disso, por utilizar o fator de ajuste para a distância, o plano gerado é mais conservador em relação ao combustível e isso é uma característica interessante quando se trata de planejamento para veículos áereos.

	Duração (s)			Combustível (lb)		
	Plan.	Real.	Dif.	Plan.	Real.	Dif.
$C1_{c/fator}$	18.482	18.690	1,13%	205	162	**-20,98%**
$C1_{s/fator}$	17.463	18.690	**7,03%**	194	162	**-16,49%**
$C2_{c/fator}$	19.579	18.690	-4,54%	188	162	**-13,3%**
$C2_{s/fator}$	18.403	18.690	1,56%	176	162	**-7,95%**

Tabela 6.12. Comparativo entre os domínios C1 e C2 para o experimento final.

6.3.5.1 Desempenho do planejador *LPG-td*

Para testar o desempenho do planejador *LPG-td*, foram construídas instâncias aleatórias com as mesmas regras citadas acima, porém, com um número maior de *waypoints*. Entretanto, em nenhuma das instâncias com mais de 25 *waypoints* o planejador conseguiu retornar um plano em até 40 min utilizando o Computador 2 detalhado na Tabela 6.1. A principal razão para isso é que os domínios tratados

nessa missão são essencialmente númericos e isso dificulta de maneira significativa o trabalho de um planejador PDDL.

6.4 Missão de combate a incêndios florestais

Nesta seção, serão demonstrados os experimentos realizados com a missão de combate a incêndios florestais. Diferentemente da missão de navegação entre *waypoints*, nesse caso, não haverá a execução do plano gerado no simulador. Para esta missão foram exploradas outras características também essenciais para o planejamento, uma delas é o paralelismo na execução das ações.

6.4.1 Metodologia Experimental

Foram desenvolvidas quatro instâncias para os experimentos com a missão de combate a incêndios florestais. O cenário é o mesmo para todas elas e é composto por um aeroporto, dois lagos e três incêndios, todos eles posicionados no mesmo lugar. A quantidade e homogeneidade dos veículos variam de acordo com a instância. A instância 1, considera apenas um veículo, já as instâncias 2, 3 e 4, consideram três, sendo que na 2 os veículos são homogêneos e na 3 e 4 heterogêneos.

Foram definidas as seguintes regras sobre como os locais se alcançam entre si:

- Aeroportos alcançam lagos, mas lagos não alcançam aeroportos;
- Lagos alcançam incêndios e incêndios alcançam lagos;
- Incêndios alcançam aeroportos, mas aeroportos não alcançam incêndios;

Essas regras definem os predicados (*reachable*) que serão adicionados no arquivo de problema. Esses predicados, por sua vez, são utilizados para saber se o veículo pode ou não se deslocar de um local para outro. É importante ressaltar que essas regras tem como principal objetivo reduzir o espaço de buscas do planejador, pois, se todos os locais se alcançassem entre si, o espaço seria consideravelmente maior. Além disso, evita que o planejador aplique ações não relevantes no plano, um exemplo está descrito abaixo:

```
106.0000: (GO UAV2 AIRPORT1 LAKE1) [D:24.00]
130.0000: (GO UAV2 LAKE1 AIRPORT1) [D:24.00]
```

Note que, nesse exemplo, o planejador gerou uma ação de deslocamento (para o UAV2) do Airport1 para o Lake1 aos 106 min e depois de 24 min aplicou uma ação de deslocamento (também para o UAV2) do Lake1 para o Airport1, ou seja, a segunda ação foi irrelevante para o contexo. O correto para essa situação seria aplicar uma ação para obtenção de água no Lake1. Deve-se ressaltar que, esse tipo de anamolia, acontece devido à dificuldade computacional em se gerar planos ótimos.

A Tabela 6.13 mostra os locais (objetos do tipo *Location*) utilizados nos experimentos e a distância entre cada um. Além disso, pode ser observado quando um local alcança (ou não) outro. Note, portanto, que se um veículo está no Airport1 ele só pode se deslocar para o Lake1 ou Lake2.

	Airport1	Fire1	Fire2	Fire3	Lake1	Lake2
Airport1	–	–	–	–	40	60
Fire1	30	–	–	–	28	50
Fire2	50	–	–	–	45	35
Fire3	70	–	–	–	85	20
Lake1	–	28	45	85	–	–
Lake2	–	50	35	20	–	–

Tabela 6.13. Objetos do tipo *Location* definidos para os experimentos com a missão de combate a incêndios florestais. Os números indicam a distância (em nm) entre cada um.

Em todas as quatro instâncias, os incêndios foram considerados com as seguintes intensidades: Fire1 (2), Fire2 (3) e Fire3 (1). Assim, como visto anteriormente, o incêndio Fire3 necessitará de apenas um lançamento de água para que seja extinto. Já o Fire2 precisará de três lançamentos.

6.4.1.1 VAL - Ferramenta de Validação de planos para a PDDL

Os gráficos que serão exibidos em cada experimento foram extraídos utilizando a ferramenta VAL (*The Automatic Validation Tool For PDDL*)[4]. Essa ferramenta é muito útil na validação dos planos gerados. Ela foi inicialmente utilizada para a terceira competição de planejamento, validando milhares de planos produzidos pelos competidores, [Howey et al., 2004]. A ferramenta não só diz quando um plano não está correto, mas também indica o que pode ser alterado para que ele seja validado corretamente. Vale ressaltar que a depuração é um dos maiores problemas

[4]http://planning.cis.strath.ac.uk/VAL/

com a PDDL, pois como o planejador é uma caixa preta, o processo de entender o que está falhando é complicado e, as vezes, é inevitável não utilizar a abordagem de tentativa e erro. Atualmente, a VAL está na versão 4.2.07. Porém, para este trabalho, compilou-se a versão 4.2.04. A saída da VAL é um relatório da validação do plano codificado em LaTeX.

Para a sua utilização é necessário passar como parâmetros os arquivos de domínio, problema e solução. O arquivo de solução contém o plano gerado. Portanto, a ferramenta, validará o plano em conjunto com o domínio e a instância que o produziram.

Para cada função (variável númerica) alterada no efeito de alguma ação, a ferramenta gera um gráfico mostrando como este valor se alterou ao longo do tempo. Assim, é possível verificar, por exemplo, como o nível de combustível de um determinado veículo se alterou ao longo do plano. Além disso, um gráfico de *Gantt* é exibido mostrando a concorrência e o paralelismo na execução das ações.

6.4.2 Instância 1

A instância 1 considerou apenas um veículo para a missão. A Tabela 6.14 mostra os parâmetros que foram utilizados para defini-lo.

O tempo máximo para o cumprimento dessa instância foi definido como sendo de 550 min. Para encontrar este valor foi utilizada uma abordagem empírica. Realizou-se um experimento com o tempo máximo de 1.000 min e este valor foi sendo decrementado de 100 min em 100 min até que fosse possível gerar um plano com maior qualidade e em um tempo razoável (menos de 1 min).

Uma outra forma de abordar esse problema seria saber qual é o menor tempo possível em que essa missão poderia ser completada, entretanto, para isso, seria necessário aplicar algum algoritmo de otimização e esse não era o foco deste trabalho. Note, entretanto, que usar o valor ótimo não traria nenhum benefício para o planejamento, pelo contrário, só mais dificuldades, pois, como visto na Seção 2.4.1, gerar um plano ótimo é um problema PSPACE.

A Tabela 6.15 mostra o resultado de cada uma das cinco execuções para alguns parâmetros. Note que há uma variação alta para o tempo de geração do plano entre as execuções. Isso acontece, pois a heurística de planejamento do *LPG-td* usa valores aleatórios em sua inicialização. Entretanto, é possível fazer com que o planejador gere sempre o mesmo plano em diferentes execuções. Para isso, deve ser passado, via linha de comando, o mesmo valor para o parâmetro *seed*.

Parâmetro	UAV1
Velocidade (knots)	100
Taxa de Consumo de Combustível (lb/h)	80
Capacidade para Combustível (lb)	150
Tempo de Reabastecimento (min)	10
Capacidade de Supressão	3

Tabela 6.14. Instância 1: parâmetros para o veículo UAV1.

	1	2	3	4	5
Tempo para geração do plano (s)	14,50	31,16	41,92	6,64	29,94
Duração total (min)	423,40	517,40	517,40	520,40	509,00
Número de ações	32	36	36	36	34
Total de combustível gasto (lb)	495,20	607,20	616,80	611,20	596,00
Número de reabastecimentos	4	5	5	5	5

Tabela 6.15. Instância 1: resultado das cinco execuções.

Para as análises a seguir foi considerado o primeiro plano dos cinco gerados acima. O trecho inicial deste plano pode ser visualizado abaixo.

```
0.0000:  (REFUEL UAV1 AIRPORT1)    [10.0000]
10.0000: (GO UAV1 AIRPORT1 LAKE1)  [24.0000]
34.0000: (PICKUPWATER UAV1 LAKE1)  [1.0000]
35.0000: (GO UAV1 LAKE1 FIRE2)     [27.0000]
62.0000: (DROPWATER UAV1 FIRE2)    [1.0000]
```

Como apenas um veículo foi utilizado, o trecho do plano acima pode ser analisado de forma sequencial. Assim, o UAV1 reabastece no Airport1, se desloca para o Lake1, obtém água, se desloca para o Fire2 e lança água. A primeira coluna representa o tempo de início da ação e a última a sua duração. Assim, a segunda ação tem início aos 10 min e dura 24 min.

O Gráfico 6.5 mostra a variação do combustível do UAV1 ao longo do tempo. Os pontos gerados na ordenada 150 (capacidade de combustível do UAV1) são as ações de reabastecimento, os demais, representam as ações de deslocamento. Como visto anteriormente, as ações de obter e lançar água não consomem combustível e por isso não aparacem no gráfico. A linha que conecta cada par de ponto representa a duração da respectiva ação. O intervalo fechado indica que o nível de combustível

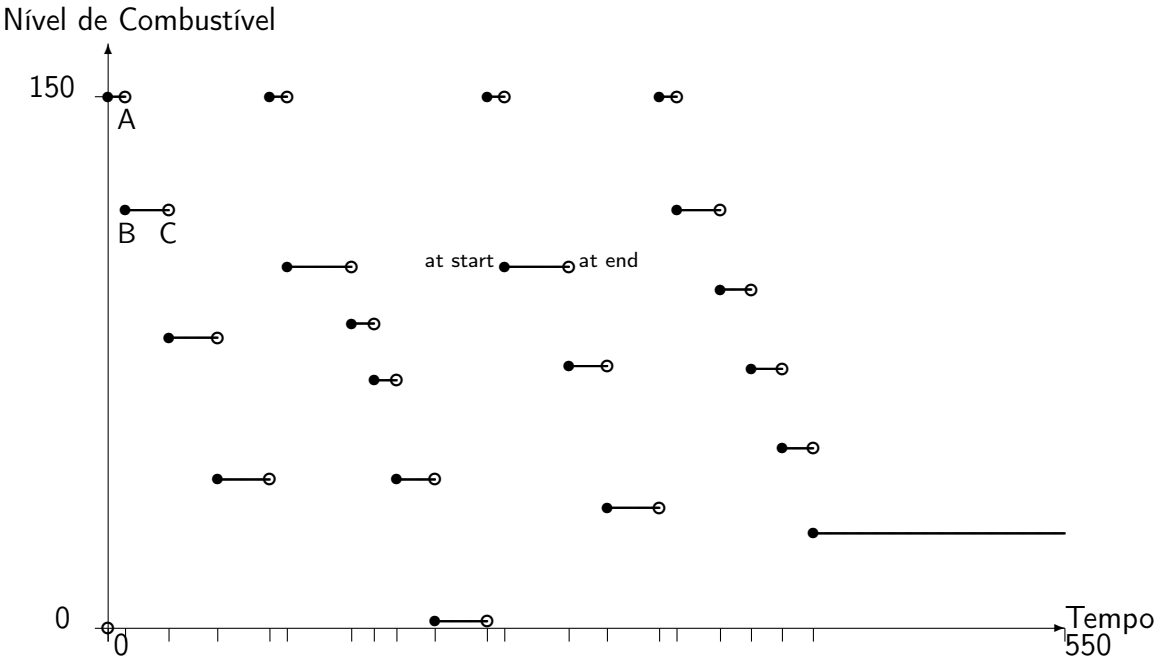

Figura 6.5. Instância 1: Variação do nível de combustível do UAV1 ao longo do tempo. Fonte: VAL.

do UAV1 foi alterado nesse ponto do tempo. Note que, para todas as ações, seja de reabastecimento ou deslocamento, o nível de combustível é sempre alterado no início (*at start*). Já o intervalo aberto, indica que não houve alteração nessa variável, todos eles estão no fim das ações (*at end*).

O ponto A representa o fim da primeira ação de reabastecimento e o ponto B representa o início da primeira ação de deslocamento. Note que, o combustível necessário para o deslocamento foi consumido de uma só vez no início e depois permaneceu constante até o fim da duração da ação de deslocamento (ponto C). É essa maneira discreta de consumir o combustível que gera a descontinuidade no gráfico. Isso é consequência direta da utilização das ações durativas discretas (nível 3 da PDDL2.1).

Um fato curioso a respeito desse gráfico é que mesmo que pareça que o veículo poderia, eventualmente, ficar sem combustível, isso não aconteceria, pois uma das pré-condições da ação de deslocamento é saber se existe combustível suficiente para realizar a navegação de um ponto a outro. Caso não exista, o planejador tentará outras alternativas de caminho até que encontre (ou não) um plano. De qualquer maneira, esse não é um plano conservador, assim, executá-lo é muito arriscado, pois, dependendo das condições atmosféricas e do próprio veículo, o consumo de combustível realizado pode ser significativamente maior do que o planejado.

6.4.3 Instância 2

Na instância 2 foram considerados três veículos, UAV1, UAV2 e UAV3. Ambos são homogêneos entre si. Os parâmetros para cada veículo são os mesmos encontrados na tabela Tabela 6.14. O tempo máximo para o cumprimento dessa instância foi definido como sendo de 200 min. A forma de obter esse valor foi a mesma encontrada na instância 1.

Foram realizadas cinco execuções e os dados podem ser observados na Tabela 6.16. Como explicado anteriormente, a variação no tempo de geração do plano (entre as execuções) acontece devido à escolha aleatória que o planejador faz para uso em sua heurística. Note, entretanto, que essa variação acontece apenas no tempo para geração do plano. Para os parâmetros, duração e consumo, a variação não é tão significativa entre as execuções.

Um dado interessante é que a execução 5 durou menos tempo apesar de ter um número maior de ações do que, por exemplo, a execução 1. Isso aconteceu, pois a métrica utilizada é a de duração (o objetivo é minimizar o tempo total).

	1	2	3	4	5
Tempo para geração do plano (s)	22,52	2,48	19,86	5,82	2,42
Duração total (min)	197,00	198,00	195,80	193,80	189,80
Número de ações	32	32	36	34	34
Total de combustível gasto (lb)	487,20	503,20	568,80	528,80	528,80
Número de reabastecimentos	4	4	5	5	5

Tabela 6.16. Instância 2: resultado das cinco execuções.

Ao comparar a duração média para a realização dessa instância (194,88 min), com a duração média da instância anterior (497,52 min), observa-se uma queda expressiva. Isso aconteceu, pois os três veículos foram utilizados para cumprir a missão. O planejador explorou o paralelismo na execução das ações e o plano deixou de ser uma sequência ordenada de ações para se transformar em uma sequência parcialmente ordenada.

Para as análises a seguir, foi considerado o primeiro plano dos cinco gerados acima. O trecho inicial deste plano pode ser visualizado abaixo.

```
0.0000:  (REFUEL UAV1 AIRPORT1)    [D:10.00]
10.0000: (GO UAV1 AIRPORT1 LAKE2)  [D:36.00]
10.0000: (REFUEL UAV2 AIRPORT1)    [D:10.00]
20.0000: (GO UAV2 AIRPORT1 LAKE1)  [D:24.00]
20.0000: (REFUEL UAV3 AIRPORT1)    [D:10.00]
30.0000: (GO UAV3 AIRPORT1 LAKE1)  [D:24.00]
44.0000: (PICKUPWATER UAV2 LAKE1)  [D:1.00]
45.0000: (GO UAV2 LAKE1 FIRE2)     [D:27.00]
46.0000: (PICKUPWATER UAV1 LAKE2)  [D:1.00]
```

Na sequência acima, é possível observar o paralelismo na execução das ações. Entre o tempo 10 min e o tempo 46 min, o UAV1 se deslocou do Airport1 para o Lake2 e, enquanto isso, outras ações para os outros veículos (UAV2 e UAV3) aconteceram. Para esse comportamento, nada foi alterado no domínio, apenas os dois veículos (UAV2 e UAV3) foram incluídos no arquivo de problema.

Além dessas questões temporais, é interessante observar que a regra que permite apenas um reabastecimento por vez foi seguida corretamente. Note que, o reabastecimento do UAV3, só iniciou depois de terminado o do UAV2 e esse, por sua vez, somente depois de terminado o do UAV1.

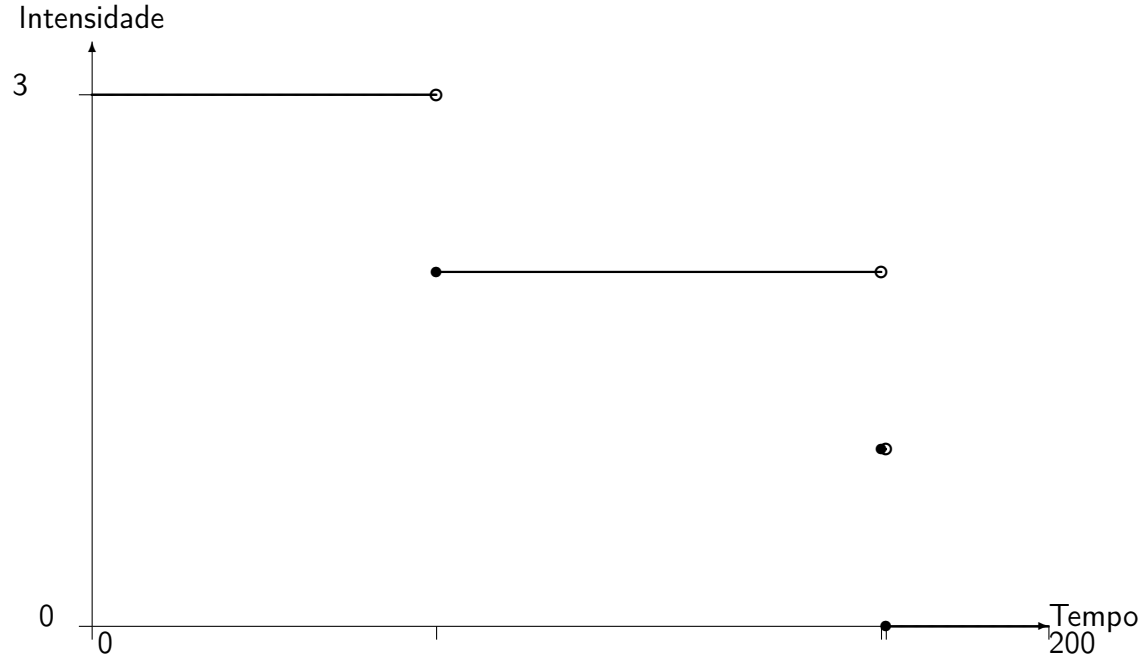

Figura 6.6. Instância 2: Variação da intensidade do incêndio Fire2 ao longo do tempo. Fonte: VAL.

O incêndio Fire2 possui intensidade 3 e foi extinto por dois veículos, o UAV2 e o UAV1. Todas as ações de lançamento de água da missão podem ser observadas abaixo.

```
59.0000:  (DROPWATER UAV1 FIRE3) [D:1.0000]
72.0000:  (DROPWATER UAV2 FIRE2) [D:1.0000]
71.8000:  (DROPWATER UAV3 FIRE1) [D:1.0000]
107.4000: (DROPWATER UAV3 FIRE1) [D:1.0000]
165.0000: (DROPWATER UAV2 FIRE2) [D:1.0000]
166.0000: (DROPWATER UAV1 FIRE2) [D:1.0000]
```

Já o Gráfico 6.6, mostra como a intensidade desse incêndio variou ao longo do tempo. É possível observar que dois lançamentos aconteceram praticamente no mesmo instante (isso também pode ser visto na lista das ações acima).

6.4.4 Instância 3

Para a instância 3, os três veículos foram considerados como sendo heterogêneos entre si. A Tabela 6.17 mostra os parâmetros que foram definidos para cada um. Considera-se que, se o veículo possui uma capacidade de supressão de incêndio alta, então ele deve carregar mais água. Nesse caso, devido ao peso, ele voa mais devagar, carrega menos combustível e possui uma taxa de consumo de combustível maior.

Parâmetro	UAV1	UAV2	UAV3
Velocidade (knots)	100	130	160
Taxa de Consumo de Combustível (lb/h)	80	60	40
Capacidade para Combustível (lb)	150	170	200
Tempo de Reabastecimento (min)	10	15	20
Capacidade de Supressão	3	2	1

Tabela 6.17. Instância 3: parâmetros para os veículos UAV1, UAV2 e UAV3.

O tempo máximo para a realização da missão foi considerado como sendo de 300 min. Note que o tempo aumentou em relação à instância 2, pois os veículos são heterogêneos (principalmente) em relação à capacidade de supressão de incêndio. Na instância anterior, os três veículos tinham capacidade 3 e agora apenas o UAV3 possui essa capacidade.

Foram realizadas cinco execuções e os dados podem ser observados na Tabela 6.18. A tabela exibe uma linha nova denominada de *Veículos Utilizados*. Essa linha mostra quais os veículos foram utilizados na respectiva execução. Apenas na execução 5 o planejador decidiu não utilizar o veículo UAV3. De fato, faz sentido, já que sua capacidade de supressão é apenas de 1, assim, é normal que em alguns planos ele não seja selecionado.

	1	2	3	4	5
Tempo para geração do plano (s)	13,82	10,8	5,4	13,42	3,64
Duração total (min)	291,00	285,00	279,00	285,00	300,00
Número de ações	36	34	36	34	32
Total de combustível gasto (lb)	515,03	472,14	504,58	440,58	481,07
Número de reabastecimentos	5	5	5	5	4
Veículos utilizados	1,2,3	1,2,3	1,2,3	1,2,3	1,2

Tabela 6.18. Instância 3: resultado das cinco execuções.

Para as análises a seguir, foi considerado o quinto plano dos cinco gerados acima. O trecho abaixo mostras as ações de reabastecimentos que aconteceram ao longo da missão. Como pode ser observado, o UAV3 não participou dessa missão e o UAV2 reabasteceu apenas uma vez.

```
0.0000:   (REFUEL UAV2 AIRPORT1) [D:15.0000]
15.0000:  (REFUEL UAV1 AIRPORT1) [D:10.0000]
114.0000: (REFUEL UAV1 AIRPORT1) [D:10.0000]
207.0000: (REFUEL UAV1 AIRPORT1) [D:10.0000]
```

O Gráfico 6.7 mostra o consumo de combustível acumulado para todos os veículos do plano.

6.4.5 Instância 4

A instância 4 tem por objetivo mostrar a capacidade que a PDDL possui de minimizar alguma dimensão que seja diferente do tempo. Nesse caso, será minimizada a quantidade de combustível total utilizada por todos veículos. Para isso, considere que agora, os veículos descritos na Tabela 6.17 tenham a mesma capacidade de supressão de incêndio, sendo esta igual a 3. Portanto, somente a velocidade, a capacidade de combustível e a taxa de consumo os diferenciam.

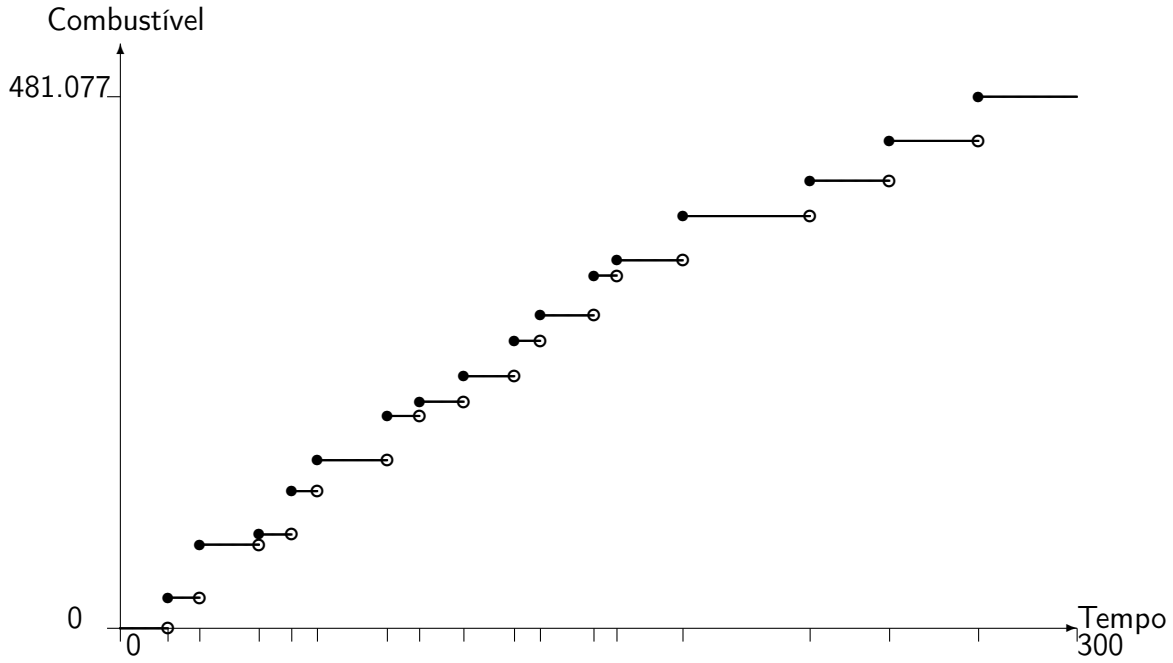

Figura 6.7. Instância 3: consumo de combustível acumulado para todos os veículos do plano. Fonte: VAL.

O tempo máximo para a missão foi reduzido para 200 min, pois, os veículos que antes não podiam lançar água em incêndios com intenisade 3, agora podem e isso, consequentemente, diminui a duração total do plano. Note, entretanto, que o tempo máximo poderia ser reduzido ainda mais, para algo próximo de 110 min. Porém, se isso for feito, o planejador utilizará todos os veículos para completar a missão e, assim, muito combustível será utilizado. Com isso, utilizar a métrica de minimização do consumo de combustível perde o sentido.

Para esses experimentos, o UAV3 inicia a missão já em voo, no local Lake1 e com 150 lb de combustível no tanque. Nas instâncias anteriores todos os veículos iniciavam no Airport1.

O planejador *LPG-td* possui uma opção na linha de comando que permite especificar diversas execuções para a mesma chamada. Cada execução tenta gerar um plano melhor do que o anterior. Para isso, o valor da métrica especificada no arquivo de problema é utilizado para comparar o plano atual com o anterior. Essa é uma opção muito útil na minimização ou maximização de alguma função PDDL, porém, infelizmente, para o domínio e problema tratados essa opção apresentou problemas.

Assim, uma outra alternativa foi adotada. Para cada plano gerado, anota-se o valor retornado pela métrica e o utiliza como um dos objetivos no arquivo de

problema. Por exemplo, se o valor retornado for de 168 lb, então esse valor é utilizado no estado final do arquivo de problema da seguinte maneira:

```
(:goal
  ...
  (<  (totalFuelUsed) 168)
  ...
)
```

	1	2	3	4	5
Tempo para geração do plano (s)	0,04	0,06	0,86	3,56	12,72
Duração total (min)	158,2	166,76	137,5	136,6	169,13
Número de ações	26	26	26	26	24
Total de combustível gasto (lb)	157,56	149,73	130,23	127,23	104,75
Número de reabastecimentos	1	1	1	1	0
Veículos utilizados	2,3	2,3	2,3	2,3	3

Tabela 6.19. Instância 4: resultado das cinco execuções.

Foram realizadas cinco execuções e os dados podem ser observados na Tabela 6.19. A primeira informação que chama atenção nessa tabela é o tempo (menos de 1 s) para a geração do plano nas execuções 1, 2 e 3. Não é possível saber ao certo o motivo desse comportamento, pois o planejador é considerado uma caixa preta sob a ótica desse trabalho, assim, não foi o foco investigar esse tipo de fenômeno. Entretanto, acredita-se que o fato dos três veículos possuírem a mesma capacidade de supressão e diferenciarem de maneira significativa nos outros parâmetros (velocidade e consumo) reduz de maneira significativa o espaço de busca.

A segunda informação interessante e que, de certa forma, explica o comportamento acima mencionado, é que o UAV1 não foi utilizado em nenhuma das execuções. Isso aconteceu, pois a sua taxa de consumo de combustível é a mais alta entre os três, assim, utilizá-lo, prejudicaria a métrica especificada nesse caso. Também é possível observar que o total de combustível gasto foi reduzindo ao longo das execuções, o que mostra a minimização do combustível gasto. A execução de número 5 utilizou apenas o UAV3 que iniciou a missão já em voo. Por isso, não houve reabastecimento.

Uma anamolia foi observada na execução 4. O UAV2, aos 34 min, iniciou o seu deslocamento para o Fire1. O deslocamento deveria durar 13 min, finalizando,

portanto, aos 47 min. Porém, ele só terminou aos 121 min do plano. Provavelmente, esse é um problema no algoritmo de planejamento do *LPG-td*.

Capítulo 7

Conclusão

Neste capítulo será apresentada a avaliação do uso da PDDL dentro no planejamento de missões para robôs aéreos. Esta avaliação é baseada, principalmente, nos resultados apresentados no capítulo anterior e na modelagem dos domínios realizada no Capítulo 4. Feito isso, são citados alguns possíveis trabalhos futuros que têm relação com o tema tratado.

7.1 Conclusões e Avaliação da PDDL

Este trabalho apresentou algumas possibilidades de aplicação da linguagem de planejamento automático PDDL no auxílio ao problema de planejamento de missão para robôs aéreos. A linguagem e os algoritmos de planejamento que operam sobre ela são utilizados para gerar sequências de ações que os robôes aéreos devem seguir para cumprir determinada missão. Até onde se tem conhecimento, essa é a primeira vez que a PDDL é aplicada dentro desse contexto. Assim, este trabalho pode servir como um ponto de partida em pesquisas mais específicas sobre o tema.

Para realizar a avaliação da PDDL dentro do planejamento de missões para robôs aéreos, foram desenvolvidas duas missões com objetivos distintos. Uma delas abordou um cenário hipotético de combate a incêndios florestais. Para essa missão não houve execução no simulador, sendo o objetivo avaliar aspectos como a geração de planos temporais e a maneira que a linguagem explora a concorrência e o paralelismo na execução das ações, principalmente quando múltiplos veículos são considerados.

Já a outra, denominada de *missão de navegação entre waypoints*, trata exclusivamente do deslocamento do veículo. Dentro dela, quatro características foram modeladas em PDDL, são elas: *velocidade*, *distância*, *altitude* e *consumo de combustível*.

Com essa missão, o objetivo foi avaliar o poder de expressividade da linguagem, verificando a possibilidade de modelar missões que envolvam um ou mais VAANTs. Um dos principais aspectos trabalhados, foi a execução dos planos gerados em PDDL no simulador de voo X-Plane® por meio de um arcabouço experimental desenvolvido neste trabalho. Essa execução permitiu extrair os dados necessários para comparar o planejado em PDDL com o realizado no simulador. A partir dessa comparação, refinava-se o domínio em PDDL com o objetivo de aproximá-lo da execução no simulador. Ao final, os resultados mostraram que os modelos construídos foram conservadores em relação às características mencionadas acima. Este é um aspecto importante quando se trata de planejamento para veículos aéreos em geral.

Mesmo com as limitações impostas no modelo, como ausência de ventos e umidade, um dos principais desafios deste trabalho foi expressar, em um linguagem utilizada para o planejamento simbólico, algumas características de um veículo de dinâmica complexa. As diversas modelagens realizadas permitiram explorar várias características da linguagem e, devido a isso, este processo contribuiu de maneira significativa na avaliação (apresentada abaixo) da PDDL dentro do planejamento de missão para robôs aéreos. Entretanto, vale ressaltar que os domínios desenvolvidos abordam características que são comuns a qualquer tipo de planejamento simbólico para robôs aéreos, seja ele realizado em PDDL ou por um sistema de planejamento dependente de domínio (ver Seção 2.6). Dentro dessa perspectiva, a PDDL se torna mais uma forma de instanciar o problema de planejamento. A vantagem de enxergar este processo por meio deste prisma é que as análises realizadas aqui se tornam uma contribuição mais genérica, podendo ser instanciadas utilizando outros tipos de sistemas de planejamento, inclusive para veículos aéreos tripulados.

Verificou-se, principalmente por meio dos domínios desenvolvidos com a missão de navegação entre *waypoints*, que a PDDL possui expressividade limitada para ser utilizada dentro de planejamento de missão para robôs aéreos. O domínio C2 desta missão, soma mais de 250 linhas de código PDDL. Entretanto, diversos trechos são cálculos repetidos dentro de uma mesma ação. Essa é uma característica que dificulta o uso da linguagem. Uma outra limitação significativa é a impossibilidade de utilizar funções como seno, cosseno, raiz quadrada e entre outras dentro da linguagem. Nesse sentido, a PDDL permite apenas o uso das quatro operações aritméticas básicas. Isso impede que problemas mais complexos sejam resolvidos a partir da própria linguagem. Somada a essas, está a limitação de não permitir chamadas a funções externas desenvolvidas em outras linguagens. Isso impede, por exemplo, o cálculo mais realista da distância que o veículo percorrerá em uma navegação entre dois *waypoints*.

Já a modelagem e os resultados obtidos com a missão de combate a incêndios florestais mostraram que a PDDL possui mecanismos interessantes para a geração de planos temporais. Assim, poder fazer uso de linguagem para expressar um domínio temporal, com algoritmos (planejadores) já prontos e consolidados que efetuam cálculos de tempo complexos e exploram o paralelismo e a concorrência das ações de maneira automática é um benefício significativo oferecido pela utilização da PDDL. Entretanto, o desempenho dos planejadores PDDL é algo que se mostrou não muito satisfatório, principalmente para domínios essencialmente númericos e temporais como é o caso dos desenvolvidos neste trabalho.

Dessa maneira, o uso da PDDL dentro do planejamento de missão vai depender diretamente do tipo de missão que será cumprida. Portanto, a complexidade do domínio codificado e o tamanho da instância que deverá ser resolvida, influenciam na qualidade e desempenho do plano gerado. Ferramentas como o itSIMPLE [Vaquero et al., 2009], facilitam o processo de prototipação em PDDL, agilizando o desenvolvimento dos domínios e das instâncias. Assim, uma alternativa interessante é implementar a missão, testar com algumas instâncias e verificar o desempenho. Se este for satisfatório, integra-se a PDDL com o respectivo sistema de planejamento de missão.

7.2 Trabalhos Futuros

Uma das principais limitações impostas por esse trabalho foi considerar que o veículo sempre realizará o voo nas condições padrão de atmosfera (ISA). Entretanto, uma aeronave raramente encontrará essa situação em um voo real. Assim, um possível trabalho futuro, seria realizar experimentos com outros tipos de variações na atmosfera e verificar como ficará a qualidade do plano nessas situações.

Um outro trabalho que poderia ser realizado a partir deste é uma comparação entre a PDDL e o planejamento hierárquivo (HTN), mais especificamente com o planejador SHOP2. Assim, os domínios construídos aqui seriam traduzidos para a linguagem do SHOP2 e testes de desempenho e qualidade do plano poderiam ser efetuados. Uma das vantagens em se utilizar o SHOP2 é que ele permite chamadas externas e, assim, é possível realizar a integração com um planejador especializado. Dessa maneira, um outro trabalho interessante, seria fazer um sistema de planejamento com o SHOP2 que fosse integrado com os planejadores de trajetória citados no Capítulo 3.

Referências Bibliográficas

Adiprawita, W.; Ahmad, A. S. & Sembiring, J. (2007). Hardware in the loop simulation for simple low cost autonomous uav (unmanned aerial vehicle) autopilot system research and development. In *International Conference on Electrical Engineering and Informatics (ICEEI'07)*, pp. 218–221.

Alford, R.; Kuter, U. & Nau, D. (2009). Translating HTNs to PDDL: a small amount of domain knowledge can go a long way. In *IJCAI'09: Proceedings of the 21st international jont conference on Artifical intelligence*, pp. 1629--1634, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Alili, S.; Pandey, A. K.; Sisbot, E. A. & Alami, R. (2010). Interleaving symbolic and geometric reasoning for a robotic assistant. In *CAMP'10: Proceedings of ICAPS Workshop on Combining Action and Motion Planning*, pp. 23–28.

Alves Neto, A. & Campos, M. F. M. (2008). Implementação de um sistema hardware-in-the-loop para veículos aéreos autônomos não-tripulados. In *Congresso Brasileiro de Automática (CBA'08)*, Juiz de Fora, MG, Brazil.

Alves Neto, A. & Campos, M. F. M. (2009a). On the generation of feasible paths for aerial robots with limited climb angle. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'09)*, pp. 2827–2877, Kobe, Japan.

Alves Neto, A. & Campos, M. F. M. (2009b). A path planning algorithm for uavs with limited climb angle. In *The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'09)*, pp. 3894–3899, St. Louis, USA.

Alves Neto, A.; Macharet, D. & Campos, M. (2010). On the Generation of Trajectories for Multiple UAVs in Environments with Obstacles. *Journal of Intelligent & Robotic Systems*, 57:123–141.

Armano, G.; Cherchi, G. & Vargiu, E. (2003). An Extension to PDDL for Hierarchical Planning. In *Proceedings of ICAPS'03 Workshop on PDDL*, Trento, Italy.

Astuti, G. (2007). *Unmanned Aerial Vehicles: State of the Art and the Road to Autonomy, Intelligent Systems, Control, and Automation: Science and Engineering*, volume 33, chapter Hardware in the Loop Tuning for a Volcanic Gas Sampling UAV, pp. 485–505. Springer.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:165--204.

Cantoni, L. F. A.; Alves Neto, A.; Chaimowicz, L. & Campos, M. F. M. (2009). Análise Comparativa entre Microsoft Flight Simulator e Flightgear Flight Simulator em Testes Hardware-in-the-loop. In *IX Simpósio Brasileiro de Automação Inteligente (SBAI'09)*, Brasília, Brazil.

Coles, A.; Fox, M.; Long, D. & Smith, A. (2008). A hybrid relaxed planning graphlp heuristic for numeric planning domains. In *ICAPS*, pp. 52–59.

Edelkamp, S. & Hoffmann, J. (2004). PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195.

Edelkamp, S.; Jabbar, S. & Nazih, M. (2006). Large-Scale Optimal PDDL3 Planning with MIPS-XXL. In *5th International Planning Competition Booklet (IPC-2006)*.

Fikes, R. & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. In *IJCAI*, pp. 608–620.

Fox, M. & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res. (JAIR)*, 20:61–124.

Gancet, J.; Hattenberger, G.; Alami, R. & Lacroix, S. (2005). Task planning and control for a multi-uav system: architecture and algorithms. In *IROS'05: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1017--1022.

Garcia, R. & Barnes, L. (2010). Multi-UAV Simulator Utilizing X-Plane. *Journal of Intelligent and Robotic Systems*, 57(1-4):393--406.

Gerevini, A. & Long, D. (2005). Plan constraints and preferences in PDDL3 - the language of the fifth international planning competition. Technical report.

Gerevini, A.; Saetti, A.; Serina, I. & Toninelli, P. (2004). LPG-TD: a Fully Automated Planner for PDDL2.2 Domains (short paper). In *14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, Whistler, Canada.

Ghallab, M.; Nationale, E.; Aeronautiques, C.; Isi, C. K.; Golden, K.; Penberthy, S.; Smith, D. E.; Sun, Y.; Weld, D. & Mcdermott, C. D. (1998). PDDL - the Planning Domain Definition Language, version 1.2. Technical report.

Gimenes, R.; Castro, D.; Reis, L. & Oliveira, E. (2008). Using Flight Simulation Environments with Agent-Controlled UAVs. In *Autonomous Robot Systems and Competitions: Proceedings of the 8th Conference*, Aveiro, Portugal.

Goktogan, A. & Sukkarieh, S. (2007). Simulation of multi-uav missions in a real-time distributed hardware-in-the-loop simulator. In *Proceeding of the 4th International Symposium on Mechatronics and its Applications (ISMA07)*.

Gravot, F.; Cambon, S. & Alami, R. (2003). aSyMov: A planner that deals with intricate symbolic and geometric problems. In *ISRR'03: International Symposium of Robotics Research*, pp. 100–110.

Guitton, J.; Farges, J. L. & Chatila, R. (2008). A planning architecture for mobile robotics. *AIP Conference Proceedings*, 1019(1):162--167.

Homa, J. M. (1998). *Aerodinâmica e Teoria de Voo: Noções Básicas*, p. 158. Editora ASA.

Howey, R.; Long, D. & Fox, M. (2004). VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *ICTAI '04: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, pp. 294--301, Washington, DC, USA. IEEE Computer Society.

Hsu, C. W.; Wah, B. W.; Huang, R. & Chen, Y. X. (2006). New Features in SGPlan for Handling Soft Constraints and Goal Preferences in PDDL3.0. In *Proc. Fifth International Planning Competition*. International Conf. on Automated Planning and Scheduling.

Ilghami, O. & Murdock, J. W. (2005). An Extension to PDDL: Actions with Embedded Code Calls. In *Proceedings of the ICAPS 2005 Workshop on Plan Execution: A Reality Check*, pp. 84–86, Monterey, California.

Lifschitz, V.; McCain, N.; Remolia, E. & Tacchella, A. (2000). Getting to the airport: the oldest planning problem in AI. In *Logic-Based Artificial Intelligence*, pp. 147--165. Kluwer Academic Publishers, Norwell, MA, USA.

Long, D.; Kautz, H. A.; Selman, B.; Bonet, B.; Geffner, H.; Koehler, J.; Brenner, M.; Hoffmann, J.; Rittinger, F.; Anderson, C. R.; Weld, D. S.; Smith, D. E. & Fox, M. (2000). The AIPS-98 Planning Competition. *AI Magazine*, 21(2):13–33.

Maris, F. & Régnier, P. (2008). Tlp-gp: Solving temporally-expressive planning problems. In *TIME '08: Proceedings of the 2008 15th International Symposium on Temporal Representation and Reasoning*, pp. 137--144, Washington, DC, USA. IEEE Computer Society.

McCarthy, J. (1968). Programs with common sense. In *Semantic Information Processing*, volume 1, pp. 403--418.

Nau, D.; Au, T.-C.; Ilghami, O.; Kuter, U.; Munoz-Avila, H.; Murdock, J. W.; Wu, D. & Yaman, F. (2005). Applications of SHOP and SHOP2. *IEEE Intelligent Systems*, 20(2):34--41.

Nau, D.; Ghallab, M. & Traverso, P. (2004). *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Nelson, M. (1995). UAV mission planning. In *DSTO Formal Reports. Report number: DSTO-TR-0164*, p. 38.

Newell, A.; Shaw, J. C. & Simon, H. A. (1959). Report on a general problem-solving program. In *IFIP Congress*, pp. 256–264.

Oliveira, A. F. (2010). Modelo de simulação de fila de transplantes de fígado baseado em sistemas multiagentes. Dissertação de mestrado, Pontifícia Universidade Católica de Minas Gerais.

Ollero, A.; Hommel, O.; Gancet, O.; Gutierrez, L.-G.; Viegas, D. X.; Forssén, V. & González, M. A. (2004). Comets: a multiple heterogeneous uav system. In *SSRR'04: Proceedings of the 2004 IEEE International Workshop on Safety, Security and Rescue Robotics*.

Olson, C. (2007). Flightgear flight simulator. `http://www.flightgear.org`. Acessado em 15 de março de 2010.

Russell, S. & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*, chapter Planning, pp. 375–416. Prentice-Hall, Englewood Cliffs, NJ, 2a edição.

Ryan, A.; Zennaro, M.; Howell, A.; Sengupta, R. & Hedrick, J. (2004). An overview of emerging results in cooperative uav control. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 1, pp. 602–607 Vol.1.

Shelquist, R. (2010). An Introduction to Air Density and Density Altitude Calculations. `http://wahiduddin.net/calc/density_altitude.htm`. Acessado em 16 de Julho de 2010.

Sinnott, R. (1984). *Virtues of the Haversine*, volume 68:2, p. 158. Sky and Telescope.

Smith, S. J. J.; Nau, D. & Throop, T. (1998). Computer bridge: A big win for AI planning.

Sonnemaker, J. B. (1999). *Meteorologia*, pp. 22,23. Editora ASA.

Sorton, E. F. & Hammaker, S. (2005). Simulated flight testing of an autonomous unmanned aerial vehicle using flightgear. In *American Institute of Aeronautics and Astronautics*, Fairmont, West Virginia. Institute for Scientific Research, Inc.

Valavanis, K. P. (2007). *Advances in Unmanned Aerial Vehicles: State of the Art and the Road to Autonomy*. Springer Publishing Company, Incorporated.

Valdes, R. (2004). *HowStuffWorks* - Como funciona o Predator UAV. `http://www.insidegnss.com/auto/janfeb08-wp.pdf`. Acessado em 10 de Janeiro de 2010.

Vaquero, T. S.; Silva, J. R.; Ferreira, M.; Tonidandel, F. & Beck, J. C. (2009). From Requirements and Analysis to PDDL in itSIMPLE3.0. In *Proceedings of the 3rd International Competition on Knowledge Engineering for Planning and Scheduling. ICAPS-09*, Thessaloniki, Greece.

Veness, C. (2010). Distance between pair of latitude/longitude points. `http://www.movable-type.co.uk/scripts/latlong.html`. Acessado em 16 de Julho de 2010.