**UNIVERSIDADE FEDERAL DE ITAJUBÁ**
**PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

# Planejamento de Missão para VANTs em Ambientes Estocásticos

**Vinicius Veloso Eleutério Nogueira**

Itajubá, 3 de dezembro de 2018

**UNIVERSIDADE FEDERAL DE ITAJUBÁ**
**PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

**Vinicius Veloso Eleutério Nogueira**

# Planejamento de Missão para VANTs em Ambientes Estocásticos

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica.

**Área de Concentração: Automação e Sistemas Elétricos Industriais**

**Orientador: Prof. Dr. Luiz Edival de Souza**

**3 de dezembro de 2018**

**Itajubá**

**UNIVERSIDADE FEDERAL DE ITAJUBÁ**
**PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

# Planejamento de Missão para VANTs em Ambientes Estocásticos

**Vinicius Veloso Eleutério Nogueira**

Dissertação aprovada por banca examinadora em 30 de Novembro de 2018, conferindo ao autor o título de **Mestre em Ciências em Engenharia Elétrica.**

***Banca Examinadora:***
Prof. Dr. Luiz Edival de Souza (Orientador)
Prof. Dr. Guilherme Sousa Bastos
Prof. Dr. Leonardo Rocha Olivi

**Itajubá**
**2018**

Vinicius Veloso Eleutério Nogueira

# Planejamento de Missão para VANTs em Ambientes Estocásticos

Dissertação submetida ao Programa de Pós-Graduação em Engenharia Elétrica como parte dos requisitos para obtenção do Título de Mestre em Ciências em Engenharia Elétrica.

Trabalho aprovado. Itajubá, 30 de Novembro de 2018:

---

**Prof. Dr. Luiz Edival de Souza**
Orientador

---

**Prof. Dr. Guilherme Sousa Bastos**

---

**Prof. Dr. Leonardo Rocha Olivi**

Itajubá
3 de dezembro de 2018

# Agradecimentos

Concluir este mestrado foi um tanto desafiador, necessitando de muita dedicação e horas de estudos, mas o esforço valeu a pena e estou muito grato à Deus por completar mais esta etapa da minha vida.

Algumas pessoas foram essenciais durante esta trajetória, em especial aos meus pais, Afrânio e Cléa, pelo amor e apoio incondicionais durante o mestrado. As minhas irmãs, Letícia, Valéria, Vanessa e Isabella, as quais me manteram motivado. Aos meus amigos, em especial a Dannilo e Leonardo, pelo companheirismo e colaboração. A Bianca pelo auxílio na revisão final deste trabalho. A república Fake Doi pelos momentos agradáveis vivenciados em Itajubá.

Agradeço ao orientador pelos conselhos, suporte e ensinamentos, os quais foram essenciais para a execução e conclusão do trabalho. Aos membros da banca examinadora que disporam seu tempo e conhecimento para analisar e lapidar este trabalho. Aos colaboradores que disponibilizaram seus trabalhos e impactaram direta ou indiretamente no desenvolvimento do trabalho. E todos aqueles que acreditaram no meu potencial, meus sinceros agradecimentos. À Capes pelo apoio financeiro durante a maior parte do mestrado.

*"Descobrir consiste em olhar*
*para o que todo mundo está vendo*
*e pensar uma coisa diferente."*
*(Roger Von Oech)*

# Resumo

Os veículos aéreos não tripulados (VANTs) têm atraído a atenção das pessoas civis pelo seu crescente número de aplicações e pela redução de custos. A grande maioria das tarefas atuais do UAV ainda possui um baixo grau de autonomia, o que pode levar à falta de eficiência, segurança, viabilidade e praticidade. Neste trabalho é analisado o emprego de técnicas probabilística em planejamento automático para melhorar a autonomia e robustez nas aplicações de UAV. O trabalho contribui ainda com um sistema de planejamento probabilístico, denominado DOTPlan, que integra o planejador Prob-PRP em sistemas ROS. Este sistema permite a visualização, execução e monitoramento de seus planos com base no formato DOT, usando a Máquina de Estados Finitos para execução e uma representação gráfica por meio de grafos gerados automaticamente. Para realizar o estudo de viabilidade do uso do Prob-PRP, o *framework* desenvolvido é analisado em um domínio de entregas com VANTs. Por fim o sistema é incorporado no controle de missões de alto nível em um UAV autônomo simulado no Gazebo.

**Palavras-chaves**: Veículos Aéreos Não Tripulados. Sistema de Planejamento. Planejamento Automático. Planejamento Probabilístico.

# Abstract

Unmanned aerial vehicles (UAVs) have been attracting civilian attention by its increasing number of applications and decreasing cost. The vast majority of current UAV tasks still have a low degree of autonomy, which can lead to lack of efficiency, safety, feasibility and practicality. This work analyzes the use of probabilistic techniques in automatic planning to improve the autonomy and robustness of UAV applications. One of the main contributions of this work is probabilistic planning system, called DOTPlan, that integrates the Prob-PRP planner into ROS systems. This system allows the visualization, execution and monitoring of its plans based on the DOT format, using the Finite State Machine for execution and clean graphical representation by automatically generated graphs. To carry out the feasibility study on the use of Prob-PRP, the proposed framework is applied in a delivery using UAVs. The planning system was then incorporated into the control of high level missions in an automated simulated UAV in the Gazebo.

**Key-words**: UAV. Automated Planning. Probabilistic Planning. Planning System.

# Lista de ilustrações

# Lista de tabelas

# Lista de abreviaturas e siglas

| | |
|---|---|
| ANAC | Agência Nacional de Aviação Civil |
| ABNT | Associação Brasileira de Normas Técnicas |
| API | *Application Programming Interface* |
| CBR | Competição Brasileira de Robótica |
| FSM | *Finite State Machine* |
| GUI | *Graphical User Interface* |
| ICAPS | *Internacional Conference on Automated Planning and Scheduling* |
| IPPC | *Internacional Probabilistic Planning Competition* |
| MDP | *Markov Decision Process* |
| PDDL | *Planning Domain Definition Language* |
| PPDDL | *Probabilistic Planning Domain Definition Language* |
| ROS | *Robot Operating System* |
| UAV | *Unmanned Aerial Vehicle* |
| UAS | *Unmanned Aerial System* |
| UNIFEI | Universidade Federal de Itajubá |
| VANT | Veículo Aério Não Tripulado |
| XML | *eXtensive Markup Language* |
| PNP | *Petri Net Plans* |
| SLAM | *Simultaneous Localization And Mapping* |

# Sumário

# 1 Introdução

O desenvolvimento e aplicação de Veículos Aéreos Não Tripulados (VANTs), também conhecidos como *drones*, iniciou-se em operações militares (TURNER; LUCIEER; WATSON, 2012), especificamente durante a Guerra do Vietnã ou Guerra Fria (WATTS; AMBROSIA; HINKLEY, 2012). Com o desenvolvimento da tecnologia que possibilitou principalmente miniaturização e barateamento de seus componentes, estes veículos se tornaram viavéis também para uso civil.

As primeiras aplicações civis dos VANTs ocorreram na área acadêmica e por hobistas, porém atualmente também são empregados comercialmente e por orgãos do governo. No presente, as aplicações para VANTs incluem: fotografia aérea, monitoramento e controle de plantações, mapeamento, operações de busca e resgate, contenção de desastres, monitoramento, patrulhamento de fronteira, inspeção, controle de meio ambiente, sensoriamento remoto e relé de comunicação. Essas aplicações já eram realizadas com o emprego de veículos áreos tripulados, porém o emprego de VANTs se faz mais conveniente principalmente pela redução expressiva do custo de operação. Além disso, ainda há um grande número de estudos que visam aumentar suas capacidades. Aplicações futuras podem requerer maior nível de inteligência do sistema, já que tendem a se tornar mais complexas. Os VANTs são amplamente explorados na literatura porém a maioria dos estudos envolvendo esses tipos de veículos ainda se concentra em estratégias de percepção e controle (BERNARDINI; FOX; LONG, 2014). Além disso, altos níveis de autonomia exigirão que o sistema aja deliberadamente.

Apesar da rápida expansão das aplicações de VANTs por civis, o estado da arte desta tecnologia ainda está presente na esfera militar. Os principais objetivos para a aplicação militar destas aeronaves são aumentar a eficiência das operações militares e diminuir o risco humano envolvido. A ausência de piloto possibilita uma maior gama de designs, o que pode ser refletido no custo da aeronave, manobrabilidade, furtividade e economia de combustível (GLADE, 2000). Apesar do grande avanço dos veículos não tripulados militares, não há presença de um alto nível de autonomia destes dispositivos. Isso reflete em uma maior relação de pessoal em terra por aeronave além de gerar uma dependência de uma comunicação com uma alta largura de banda e confiável.

Isto se evidencia observando uma das principais referências na área, o modelo Global Hawk pertencente ao governo dos Estados Unidos. Em levantamento feito em 2005, o Global Hawk apresentava taxa de acidentes de aproximadamente 21 vezes maior que o F-16, uma nave tripulada (GERTLER, 2012). Este modelo ainda demanda uma conexão dedicada de 500 Mbps. Estes fatores diminuem a confiança e escalabilidade do

sistema não tripulado que reduzem sua viabilidade em ambientes reais.

No Brasil o uso de VANTs para civis é regulamentado pela Agência Nacional de Aviação Civil (ANAC). Recentemente o regulamento sobre este tipo de veículos foi revista e tornou-se mais restritiva buscando a proteção de terceiros. Os principais pontos desta nova norma para veículos com menos de 25kgs que operem a no máximo 120m de altura são: drones com mais de 250g devem ser registrados e assegurados independente de sua finalidade, seu uso não recreativo restrito a operadores maiores de 18 anos, mantenham a distância horizontal mínima de 30 metros de pessoas não envolvidas na operação e que seu voo autônomo deve permitir a interferência de um operador responsável. O documento possibilita que apenas orgãos do governo operem próximos a terceiros (Agência Nacional de Aviação Civil, 2017), o que limita as aplicação deste dispositivo em principalmente em zonas urbanas.

Atualmente a maior parte dos VANTs são utilizados comercialmente para aquisição de imagens aéreas, sendo estes remotamente rádio controlados. Os veículos remotamente controlados exigem que o operador tenha perícia e mantenha contato visual com o mesmo. Além disso, estes veículos possuem baixa autonomia e exigem uma comunicação confiável e robusta entre o aparelho e o operador, o que é difícil de se garantir em situações reais. Alguns dispositivos comerciais mais sofisticados já apresentam alguma autonomia, como retornar diretamente ao ponto de partida ou ações reativas, sendo que a tomada de decisões é feita pelo operador. Por meio da implementação de técnicas de inteligência artificial e ações deliberativas, o sistema aumentaria a eficiência e eficácia para se atingir os objetivos e consequentemente elevar o grau de autonomia do sistema. Isto possibilita um maior número de aplicações com grau de complexidade superior.

Um campo promissor dentro da Inteligência Artificial capaz de fornecer aos sistemas autônomos a inteligência necessária é o Planejamento Automático. Planejamento Automático é o raciocínio das ações com base em um modelo abstrato e geral do problema com a função de produzir um plano capaz de orientar o agente para atingir seus objetivos(GHALLAB; NAU; TRAVERSO, 2004). O principal propósito desta área é desenvolver uma metodologia para solução de problemas que possa ser aplicado em cenários e problemas gerais. A principal vantagem dessa abordagem é a possiblidade de um desenvolvimento de um sistema inteligente mais rapidamente, evitando que um planejador seja criado para cada aplicação sem exploração das similaridades.

Outro aspecto relevante é que nada impede um planejador automático independente do domínio de ser integrado como uma parte de um planejador específico. Há uma diversidade de planejadores de inteligência artificial disponíveis em pleno desenvolvimento que podem ser utilizados como ferramentas. Sendo que estes recebem como entrada arquivos em um formato padrão, escritos em PDDL - *Planning Domain Definition Language*, que descrevem o problema e o domínio de conhecimento necessário para o planejamento

de ações. O planejador retorna um plano que consiste em uma lista de ações.

Os VANTs são sistemas naturalmente instáveis que possuem alta sensibilidade a distúrbios, como vento e chuva. Além disso, a limitação das fontes de energia torna o consumo de energia destes veículos crucial para atingir seus objetivos (SYDNEY; SMYTH; PALEY, 2013). Portanto, deve-se considerar a incerteza ambiental durante o planejamento de tarefas, ao ponderar previamente sobre as incertezas o agente pode ainda raciocinar antecipadamente sobre mecanismos de recuperação de falha e estratégias de tentativa e erro.

Uma área dentro do Planejamento Automático é o Planejamento sobre Incerteza que leva em conta as incertezas no estado inicial, nas consequências das ações ou ainda na observação do estado atual (GHALLAB; NAU; TRAVERSO, 2004). Para este trabalho o cenário abordado será a execução de tarefas em um contexto aonde o VANT não possui um gasto fixo de recursos para executar as ações com o objetivo de evitar a falta de recursos independente das ocasionalidades. Estes gastos podem ser resultados de falhas de modelagem ou ainda de eventos externos no ambiente.

A combinação de Planejamento Automatizado e Veículos Aéreos já foi proposta em (CANTONI; CAMPOS; CHAIMOWICZ, 2011), onde foi estudado o planejamento clássico em um cenário de combate a incêndios e simulado no X-Plane. Outra pesquisa semelhante foi feita por (BERNARDINI; FOX; LONG, 2014) no planejamento de vigilância aérea para pesquisa e rastreamento operações. O planejamento automatizado também é aplicado comumente em outros tipos de aplicações robóticas (QUINTERO et al., 2011) (CROSBY et al., 2017).

Como o Planejamento Automático tornou-se a ferramenta padrão para solução de problemas independentes de domínio e o Sistema Operacional Robótico (ROS)(QUIGLEY et al., 2009) para aplicações robóticas, estas ferramentas foram fundidas em (CASHMORE et al., 2015) resultando no ROSPlan. Este pacote começou suportando um planejador temporal determinístico com a proposta de ser uma ferramenta para diferentes aplicações, no qual foi aplicado originalmente em veículos subaquáticos autônomos. Mais tarde, foi expandido para dar suporte a geração de planos de contingência, usando Contingent-FF (SANELLI et al., 2017). Atualmente, o ROSPlan é a principal biblioteca para planejadores automáticos, mas ainda não suporta nativamente planejadores não-determinísticos. Existem outras bibliotecas na literatura que suportam planejadores não-determinísticos porém, apresentam uma estrutura específica para sua aplicação.

Dadas as vantagens de se operar VANTs de forma autônoma e que eles são bastante susceptíveis a perturbações em ambientes externos, a motivação do trabalho está em criar uma estrutura em um ambiente de desenvolvimento robótico que possibilite experimentar com a operação de VANTs com base em planos gerados por planejadores não-determinísticos.

## 1.1 Objetivos

O principal objetivo deste trabalho é propor um sistema autônomo de coordenação de aeronaves não tripuladas que utilize um planejamento não determinístico e independente do domínio para tomada de decisões de alto nível. Este sistema será desenvolvido dentro do *framework* do ROS e deverá conter ferramentas que possibilitem o deliberação e designação de tarefas em VANTs simulados de ambientes hipotéticos. Para isto o sistema irá contar com ferramentas de planejamento, supervisão e monitoramento do plano com base no dominio e seus objeitivos. Com ele analisar a aplicabilidade deste tipo de planejamento para elevar o grau de autonomia destes sistemas.

## 1.2 Contribuições

Este trabalho tem como principais contribuições no desenvolvimento de um *framework* que possibilita a utilização de planejados não-deterministicos em problemas gerais e o desenvolvimento do ambiente de simulação de VANTs. Esta contribuição inclui:

- Modelagem de domínios com VANTs na linguagem em PPDDL;
- Inclusão e calibração de funções de navegação no VANT simulado;
- Desenvolvimento de um *framework* que possibilite a elaboração, visualização e supervisão dos planos gerados;
- Comparação com bibliotecas similares disponíveis.
- Análise da aplicação da proposta nos domínios simulados.
- Implementação genérica de um planejador probabilístico no ROS.
- Incorporação e adaptação da ferramenta de análise de grafos padrão do Ubuntu, XDot, no ROS.
- Análises e melhorias nos algoritmos disponibilizados por terceiros.

## 1.3 Organização do trabalho

No Capítulo 2 é realizada uma revisão bibliográfica sobre planejamento automático clássico, incluindo definição, representações de problema, a linguagem utilizada para definição do problema e as extensões do planejamento clássico. O Capítulo 3 discute sobre o planejamento sob incerteza, apresenta técnicas de tratamento de incertezas e mostra a tradução de políticas em planos condicionais. O Capítulo 4 exibe o sistema de desenvolvimento robótico (ROS) e seus principais sistemas de planejamento. O Capítulo 5

mostra o domínio estudado assim como o ambiente de estudo. O Capítulo 6 trata sobre o desenvolvimento do sistema de planejamento DOTPlan, detalhando a operação de cada módulo. Em seguida, no Capítulo 7, são realizadas e discutidas experiência no domínio proposto com base no sistema de planejamento desenvolvido. Finalmente, no Capítulo 8, são discutidas as conclusões e possibilidades de melhorias futuras.

# 2 Planejamento Automático Clássico

Agente é aquele capaz de perceber e agir sobre o ambiente que está inserido, caso ele raciocine sobre a melhor forma de agir é denominado agente inteligente. Planejamento é o processo em que o agente organiza suas ações através da antecipação das suas consequências com intuito de atingir certos objetivos pré-definidos de forma consciente(GHALLAB; NAU; TRAVERSO, 2004). Dessa forma para planejar o agente deve estar ciente deste processo para que ele ocorra de fato.

O ser humano não planeja frequentemente suas ações e apenas o faz quando há uma grande necessidade, geralmente em situação novas, complexas, de alto risco ou que exijam coordenação com outros agentes. Consequentemente as ações humanas não buscam uma solução ótima mas sim uma solução boa o suficiente. O Planejamento Automático consiste na compreensão do raciocínio para que ele possa ser feito de forma automática para auxiliar o ser humano ou máquinas na tomada de decisões. Esta ferramenta pode ser então utilizada para tomar várias decisões desde as mais monótonas até as mais complexas.

Há duas formas distintas de Planejamento Automático, o planejamento específico e o independente do domínio. Atualmente, as soluções específicas de domínio superam planos independentes de domínio, mas eles precisam de muito esforço de modelagem que requer conhecimento especializado e capacidades deliberativas limitadas à sua área. Por tanto, as estratégias independentes de domínio atraem mais interesse de pesquisa sendo assim esta área esteve em relevância nos últimos anos.

Não se espera que o planejamento independente do domínio supere futuramente o específico em questão de performance porém há um grande esforço para aumentar a expressividade da modelagem do domínio e também planejadores que funcionam melhor em certo grupo de problemas. Sempre haverá o *trade-off* entre performance e *lead-time*, situação que é comum entre produtos padronizados e customizados. Neste trabalho será abordado o planejamento independente do domínio. O Planejamento Automático é descrito neste capítulo segundo (GHALLAB; NAU; TRAVERSO, 2004).

## 2.1 Modelo Conceitual

O Modelo Conceitual é uma forma simplificada de descrever os elementos de planejamento. A estrutura do modelo conceitual da Figura 1 situa a organização dos principais elementos na atuação de um agente inteligente, são eles o planejador, o controlador e o sistema (GHALLAB; NAU; TRAVERSO, 2004). O planejador gera um plano composto por uma sequência de ações de alto nível a partir do modelo abstrato do problema. O

controlador traduz o plano e executa as tarefas sequenciais em baixo nível. O ambiente que pode ser representado como um sistema de transição de estados que se modifica a partir de ações do controlador e de eventos externos. Esta estrutura ainda pode incluir um *feedback* do controlador para o planejador com intuito de adequar o plano às observações do sistema e tornar-lo mais robusto. As observações são importantes pois exprimem a diferença do sistema real e sua descrição abstraída para o controlador.

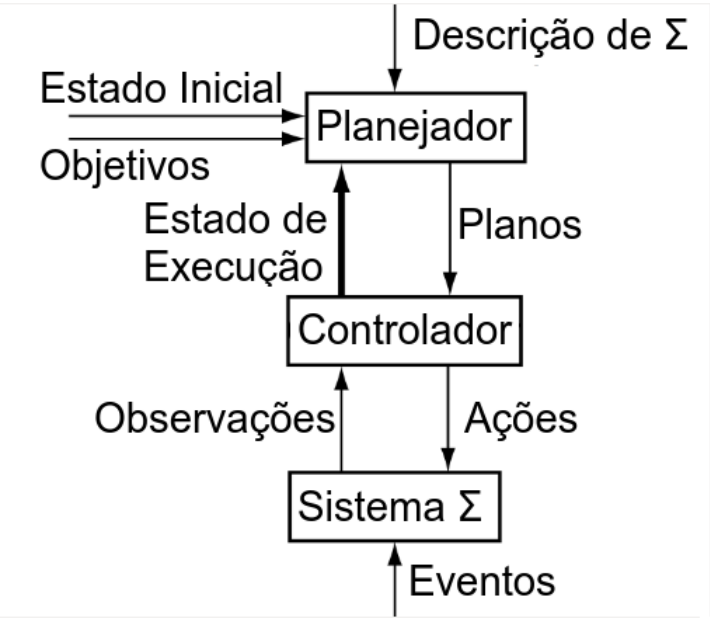



Figura 1 – Estrutura do Modelo Conceitual. Imagem adaptada de (GHALLAB; NAU; TRAVERSO, 2004)

O domínio de planejamento é modelado no modelo conceitual como um sistema à eventos discretos composto por estados e transições descritos formalmente como uma 4-tupla $(S, A, \Sigma, \gamma)$ onde :

- $S$ é um conjunto finito de estados;
- $A$ é um conjunto finito de ações;
- $\Sigma$ é um conjunto finito de eventos externos;
- $\gamma$: $S$ x $A$ x $\Sigma \rightarrow 2^S$ é uma função de transição de estados;

Neste modelo, um estado é dado por um conjunto de fatos que podem ser alterados a partir de ações ou eventos externos. Ele pode ser representado através de grafo direcionado em que os estados representam os nós e as transições as ações do agente ou eventos. Para exemplificar este conceito utilizaremos um domínio hipotético, nele o agente:

- ganha dinheiro ao trabalhar porém fica triste e sujo, enquanto o trabalho exige que ele esteja limpo;

- precisa estar limpo e com dinheiro para que possa sair e se divertir, ficando feliz, sujo e sem dinheiro;
- toma banho quando está sujo para ficar limpo;

Neste modelo os estados são definidos como a combinação de todas instâncias positivas e negativas de todos os fatos do domínio, que no caso são os predicados *limpo*,$feliz$ e *dinheiro*. As ações e eventos externos modificam o estado atual do sistema, levando um agente de um estado ao outro. O sistema de transição do domínio dos trabalhadores pode ser representado gráficamente conforme a Figura 2.

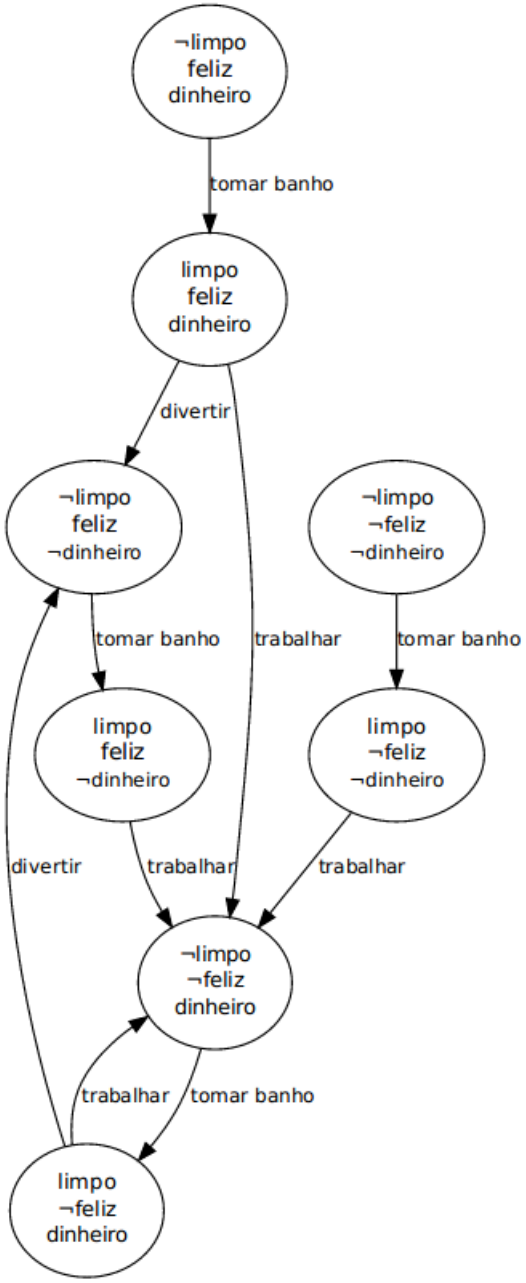



Figura 2 – Modelo Conceitual do dominio do trabalhador.

Este domínio pode ser descrito formalmente conforme a descrição:

- $A : A1 = tomarbanho, A2 = trabalhar, A3 = divertir$;
- $S$ : S1= $(limpo, \neg feliz, \neg dinheiro)$, S2= $(limpo, \neg feliz, dinheiro)$, S3= $(limpo, feliz, dinheiro)$, S4= $(limpo, feliz, \neg dinheiro)$, S5= $(\neg limpo, \neg feliz, \neg dinheiro)$, S6= $(\neg limpo, \neg feliz, comdinheiro)$, S7= $(\neg limpo, feliz, dinheiro)$, S8= $(\neg limpo, feliz, \neg dinheiro)$;
- $\Sigma$ :{};
- $\gamma$ : (S1,A2,S7), (S1,A3,S6), (S2,A2,S7), (S3,A2,S7), (S3,A2,S6), (S4,A2,S7), (S5,A1,S1), (S6,A1,S2), (S7,A1,S3), (S8,A1,S4) .

Na Figura 2 é possível observar a evolução dos sistemas de acordo com as ações tomadas e visualizar os planos capazes de levar o sistema de um estado a outro. A determinação do plano consiste em realizar e explicitar todos os estados e transições possíveis e a partir disso realizar uma busca com intuito de definir uma sequência de ações capazes de levar o agente de um dado estado inicial a algum dos estados objetivos. Nem sempre há possibilidade encontrar um plano pois pode não existir uma sequência de ações que façam o agente atingir seu objetivo. Para ilustrar, definindo o estado inicial (*limpo*, $feliz$, *dinheiro*) e o estado final como sendo ($\neg$*limpo*, $feliz$, $\neg$*dinheiro*), existiriam várias soluções como p1=(A3) e p2=(A2,A1,A3).

## 2.2 Planejamento Clássico

O Modelo Conceitual é bastante didático porém em termos práticos não é aplicado por conta de sua complexidade computacional. Em problemas de planejamento o número de estados cresce exponencialmente com o número de predicados, o que torna impraticável a representação dos mesmos de forma explícita. Sendo assim, a comunidade científica sentiu a necessidade de explorar novas formas de representação dos componentes do domínio e de desenvolver melhores dispositivos de controle de buscas.

Com intuito de desenvolver metodologias para viabilizar as buscas do planejador se originou o Planejamento Clássico, procurando solucionar os problemas menos complexos. O Planejamento Clássico parte de premissas mais restritivas, são elas: número finito de estados, sistema totalmente observado e estático, com um único objetivo, planos sequenciais, tempo implícito e planejamento *off-line*. Estas premissas simplificam o problema de planejamento a solução de um sistema de transição de estados. A maior vantagem deste modelo é sua escalabilidade. Posteriormente esse tipo de planejador tornou-se base para o desenvolvimento de algoritmos mais abrangentes de planejamento.

### 2.2.1 Representação Clássica

Há três formas populares de representação de Problemas Clássicos: representação por conjuntos teóricos, representação clássica e por variáveis de estados. Neste trabalho será apresentada a Representação Clássica por ser uma das representações compactas mais popular. Na representação clássica, os predicados têm um papel fundamental em expressar o conhecimento, porém podem não representar conhecimento por si. Eles também podem conter símbolos variáveis para definir os fatos, necessitando assim a instanciação das porções variáveis a fim de constituir uma informação. Um predicado totalmente determinado é definido como um *átomo* bem como *literal*, podendo ser uma afirmação ou negação.

Um estado representa uma circunstância do domínio se e somente se possui todos os *literais* positivos e nenhum dos negativos. Enquanto alguns predicados são fluentes e suas variáveis têm seu valor alterado com a mudança de estados outros expressam uma relação rígida entre símbolos.

Os predicados rígidos retratam as regras lógicas fixas do domínio enquanto os fluentes remetem às configurações mutáveis. Esta representação assume que quando um *átomo* não é explicitamente expresso ele é admitido como uma negação, tal suposição é chamado de suposição de mundo fechado.

Um exemplo de planejamento clássico é o problema dos missionários e canibais, neste problema um barco deve atravessar três missionário e três canibais por um rio, se os canibais estiverem em maior número que os missionários em alguma das margens são comidos. O barco deve ir de uma margem a outra carregando no mínimo um tripulante e no máximo dois. Neste domínio o predicado *localbarco* $?x$ pode ser utilizado para denominar a margem que se encontra o barco sendo que $?x$ pode assumir qualquer valor de objeto, devendo ser instanciado por um símbolo como $margem1$ resultando no átomo $localbarco?margem1$. Enquanto o predicado localbarco é fluente, outro predicado como *ladooposto* $?x$ $?y$ possui as relações rígidas.

Os operadores são expressões genéricas de ações cujos predicados não são instânciados. Formalmente os operadores são definidos por uma tripla $o = N(o), P(o), E(o)$, sendo:

- $N(o)$ é a descrição do operador dado por $n(x1, x2, ...)$ onde n é um identificador único do operador e xn são símbolos operacionais que constituem todas os símbolos variáveis presentes nas precondições e efeitos;
- $P(o)$ são as precondições dadas por um conjunto de literais que necessitam ter suas condições sustentadas no estado atual para a execução do operador;

- $E(o)$ são os efeitos dados por um conjunto de literais do estado atual que serão modificados pelo operador;

Nesta formalização, os símbolos operacionais representam os símbolos variáveis que deverão necessariamente ser substituidos pelos mesmos valores em todo o operador para caracterizar uma ação. Sendo assim a lógica de atuação do operador é dada pelos símbolos operacionais dos predicados. A lógica de mover o barco no domínio dos canibais pode ser expressa através do operador abaixo:

```
mover_barco (l1,l2)
            precond: localbarco(l1),localoposto(l1,l2),tripulado
            efeito : ¬localbarco(l1), localbarco(l2)
```

Neste operador o barco tripulado deve atravessar para a margem oposta. Nele a relação rígida de margem oposta deve estar previamente definida, o barco presente em uma margem l1 executa a ação e passa se situar em l2. Este operador também depende do fluente *tripulado*, que é modificado através de outros operadores de carga e descarga de passageiros. A computação da transição de estados por meio de operadores é realizada conforme as definições estabelecidas nas definições 2.1 e 2.2, retiradas de (GHALLAB; NAU; TRAVERSO, 2004).

**Definição 2.1** Sendo $L$ um conjunto de literais. $L^+$ é o conjunto o qual representa os átomos e $L^-$ o conjunto com a negação de átomos. Para uma determinada instância $o$, $precond^+(o)$ e $precond^-(o)$ são as precondições positivas e negativas respectivamente. Do mesmo modo, $efeitos^+(o)$ e $efeitos^-(o)$ são os efeitos positivos e negativos.

**Definição 2.2** A instanciação dos símbolos operacionais caracteriza uma ação. Se $a$ for uma ação e s for um estado tais que $precond^+(a) \subseteq s$ e $precond^-(a) \cap s = \emptyset$, então $a$ é aplicavél em s, e o resultado dessa aplicação é dado por:

$$\gamma(s,a) = (s - efeitos^-(a)) \cup efeitos^+(a) \tag{2.1}$$

## 2.3 Linguagem em PDDL

A linguagem PDDL, referente a sigla em inglês *Planning Domain Definition Language*, é a principal forma de estruturação dessa representação. Ela surgiu para padronizar a comparação de planejadores na principal competição internacional de planejamento (MCDERMOTT et al., 1998), a IPC (*Internacional Planning Competition*), durante a conferência de maior referência na área, o *Internacional Conference on Automated Planning and Scheduling* (ICAPS).

Houve uma evolução no PDDL para aumentar sua expressividade com a meta de se aproximar adequeadamente de problemas reais. Dentre as várias versões propostas a variante amplamente utilizada por planejadores é a versão 2.1. Esta linguagem possui uma expressividade que vai além da representação clássica e permite a comparação direta entre diferentes sistemas de planejamento. Nesta representação, os estados não são explicitamente definidos pois são computados a partir do estado inicial ou de algum estado objetivo utilizando a equação 2.1 da definição 2.2.

O problema de planejamento é dividido em duas partes: a descrição do domínio e a descrição do problema. A descrição do domínio inclui a estruturação dos operadores e dos predicados de forma geral. A descrição do problema já estabelece os objetos presentes no plano, todos os literais positivos do estado inicial e os literais que satisfazem os estados objetivo do problema. Dessa forma, o domínio é composto de elementos constantes e variáveis que definem o funcionamento lógico do sistema, já o problema é constituido de instanciações do domínio que especificam contexto do problema. Sendo assim a descrição domínio pode ser utilizada para resolver diferentes problemas com diversos objetos, estados iniciais e objetivos variados. A descrição completa em PDDL do domínio e do problema dos missionários e canibais está disponível no Apêndice A. Esta é uma abstração mais didática do domínio, onde o movimento do barco e a subida e descida de passageiros são explicitos, o que aumenta significamente o número de estados. Em uma representação mais compacta, os participantes vão de uma margem a outra instantaneamente, reduzindo os operadores e o número de ações do plano, no entanto os operadores ficam mais complexos e possuem mais variavéis. Este fato demonstra a não trivialidade em se modelar os mecanismos de sistemas em planejamento.

A representação clássica pode ser expandida para aumentar sua expressividade ou intuitividade em descrever domínios. Algumas extensões incluem:

**Tipificação**: Classificação em grupos dos objetos, símbolos variáveis e símbolos operacionais. Isto resulta em uma restrição na instanciação do predicado. No domínio dos canibais, nada impede o planejador de instanciar o predicado *localbarco* ?*lado* como *localbarco um*. Na descrição do problema os objetos podem ser representados como *margem1 margem2 - local* e na descrição do domínio o predicado *localbarco ?lado - local*. Dessa forma apenas os objetos do tipo *local* seriam substituídos na variável *?lado*.

**Condicionalidade**: Condicionamento de uma parcela dos efeitos nos operadores à determinados predicados. Em uma possível expansão do domínio do trabalhador, o agente seria promovido a gerente se trabalhasse e fosse graduado. Em PDDL esta condição pode ser representada nos efeitos do operador da seguinte forma *(when (graduado) then (gerente))*:

**Concorrência de ações**: Paralelismo na execução de ações. Permite que ações que tenham suas precondições atendidas sejam executadas simultaneamente. Muito im-

portante em sistemas multi-tarefas e multi-agentes.

**Temporização**: Representação explícita do tempo nos operadores. Permite que ações concorrentes tenham durações distintas podendo se iniciar ou finalizar em tempos diferentes e ainda possibilita que predicados componham o operador antes, durante ou após a execução da ações.

**Funções algébricas**: Expansão das variáveis lógicas para númericas. Possibilita a comparação e operação algébrica entre variáveis lógicas.

**Métrica**: A métrica pode ser determinada como o tempo, número de ações ou ainda valor de uma variável númerica. Podendo ser maximizada ou minizada no planejamento de acordo com a necessidade. Utilizada na busca de um plano ótimo e na orientação da busca de soluções.

**Objetivos Expandidos**: Expansão do propósito do planejador para objetivos intermediários além do objetivo principal. Estes objetivos podem ser obrigatórios ou facultativos.

**Não-Observabilidade inicial**: Relaxamento do valor de alguns átomos do estado inicial para uma lista de possibilidades. Representa quando o estado inicial não é totalmente conhecido.

**Não-Determinismo**: Admissão de efeitos incertos no operador. Um operador não-determinístico possui efeitos com predicados alternativos que ocorrem de forma aleatória. Utilizada quando as consequências não podem ser determinadas *a priori* da execução, sabendo apenas as consequências prováveis.

**Probabilismo**: Extende o não-determinismo permitindo explicitar a probalibilidade de ocorrência de cada efeito de uma ação.

Nenhum planejador atualmente suporta todas as expansões do PDDL. Os planejadores em geral suportam uma parcela das expansões e focam em uma classe de problemas específica. O planejador OPTIC (BENTON; COLES; COLES, 2012), permite a expressão de uma métrica baseada em funções númericas dependentes do tempo. O planejador (HARRIS, 2015) busca a otimização da utilidade do plano em problemas probabilísticos determinando quais objetivos facultativos a ser cumpridos e assim a sequência de ações apropriadas. Sendo assim, a solução de um problema real inicia-se na escolha de um planejador mais adequado que suporte a expressidade exigida pelo ambiente.

# 3 Planejamento Automático sobre Incerteza

O Planejamento Clássico se mostrou capaz de encontrar soluções para diversos jogos e enigmas. Estes problemas de raciocínio lógico se mostram bem importantes no desenvolvimento do planejamento, pois expressam diferentes aspectos do raciocínio e inclusive são utilizados como base de comparação entre os planejadores. Um problema deste tipo de planejador é a modelagem do domínio diante de tantas restrições pois enquanto as regras do ambientes são bem definidas nos jogos e enigmas, na prática muitos sistemas reais apresentam incertezas. Muitos problemas reais podem de fato serem aproximados por problemas clássicos dependendo do grau de abstração ou expressividade exigida. Geralmente as ações são tratadas em um alto nível de abstração, delegando ao controlador a função de lidar com o rompimento das premissas. Essa aproximação resulta em uma aceleração e simplificação do processo de planejamento.

Em um ambiente totalmente conhecido, o estado atual é totalmente sabido e as ações o modificam de forma previsível. Por mais que a premissa de completa observalibilidade e determinismo não se mantenham verdadeiras em muitas aplicações reais, é possível se aproximar por um sistema livre de incertezas considerando-se apenas a consequência mais provável para cada ação. No entanto, quando o sistema é submetido a perturbações exógenas ou quando ações possuem efeitos alternativos críticos se faz necessário a deliberação das ações considerando estas incertezas (GHALLAB; NAU; TRAVERSO, 2004).

Este capítulo apresentará uma série de estratégias de planejamento que lidam com a presença de incertezas do ambiente. Primeiramente, o replanejamento é apresentado como uma técnica mais básica de tratar situações adversas. Posteriormente, o planejador de contingência é descrito como uma forma de planejamento com observabilidade parcial. Em seguida, técnicas de planejamento probabilístico são retratadas, iniciando com a técnica clássica, o MDP, desenvolvendo os conceitos até o planejador aplicado neste trabalho, o Prob-PRP. Encerrando com a apresentação da extensão do PDDL para problemas probabilísticos juntamente com um controlador por máquinas de estado.

## 3.1 Replanejamento

A forma básica de se lidar com as incertezas é o replanejamento. Nele o controlador lida com o sistema não determinístico e mantém o controle sobre o estado atual do sistema e do plano. Caso seja identificada uma discrepância entre eles, o planejador replaneja suas ações tendo como base o estado atual real como estado inicial. Na modelagem clássica de domínios com incertezas o efeito de cada ação é normalmente determinado como a consequência mais provável, o mesmo ocorre com o valor dos literais desconhecidos. Desse

modo, cada vez que o agente se depara com um efeito menos provável de suas ações ou outras situações adversas, ele replaneja (HARRIS, 2015).

**Algoritmo 1** Execução de um Plano com Replanejamento

**Entrada:** $operadores, predicados, simbolos, estadoinicial, objetivos$
**Saída:** $objetivoatingido$

```
estadoobs = {}
enquanto estadoobs ⊈ objetivos faça
    ações ← Planejamento(operadores, predicados, simbolos, estadoinicial, objetivos)
    se ações = {} então
        retorne falso
    para cada ação ∈ ações faça
        estadoplano ← estadoinicial
        estadoplano ← Progride(estadoplano,ação)
        sucesso ← Executa(ação)
        se sucesso ≠ verdadeiro então
            retorne falso
        estadoobs ← ObservaEstado()
        se estadoplano ≠ estadoobs então
            estadoinicial ← estadoobs
            interrompa
retorne verdadeiro
```

O algoritmo 1 mostra uma estrutura de execução de um plano que utiliza replanejamento. Por intermédio deste algoritmo, o agente é guiado para o objetivo mesmo que ocorram adversidades. Como visto no algoritmo ele também pode falhar, caso não seja possível encontrar um plano válido ou ainda se houver falha na execução de uma ação. Um problema na implementação do replanejamento é que ele não raciocina sobre os diferentes cenários possíveis. A deliberação sobre as prováveis situações pode evitar bloqueios mortais (*dead-ends*) ou bloqueios com interação repetitiva (LITTLE; THIEBAUX et al., 2007). Estes *dead-ends* podem representar estados críticos e irrecuperáveis do sistema, onde o agente tem potencial de causar danos.

Mesmo sendo um algoritmo simples que não oferece garantias quanto a evasão de estado sem saída evitáveis, o planejador FF-Replan que utiliza este tipo de estratégia foi capaz de superar planejadores probabilísticos e vencer a primeira IPPC (*International Probabilistic Planning Competition*) em 2004. Além disto, este planejador foi capaz de exceder todos os participantes da IPPC em 2006, apenas com modificações mínimas (HARRIS, 2015). O sucesso deste planejador na competição foi atribuído principalmente à simplicidade dos problemas de planejamento propostos. Muitos domínios tinham seus efeito mais frequentes com a probabilidade muito maior e não havia presença de muitas situações críticas. O replanejamento apresenta uma alternativa simples e válida para resolver um conjunto de problemas probabilísticos.

## 3.2 Planejamento de Contingência

O agente pode ser dotado fisicamente de sensores para auxiliar na tomada de decisões. Neste modo de planejamento o sensoriamento é considerado na elaboração do plano fazendo parte também do plano gerado (MUISE; BELLE; MCILRAITH, 2014). Ele é representado por um operador de observação capaz de modificar literais indiferenciados. Sendo assim, o planejador permite a presença de literais com valores indeterminados nas observações e no estado inicial, a serem revelados durante a execução do plano. O papel do planejador é encontrar uma sequência de ações capaz de levar o agente ao objetivo para todo valor possível dos literais indeterminados e organizá-lo em torno de pontos de decisão, em que a ramificação ocorre de acordo com os valores observados (HOFFMANN; BRAFMAN, 2005). A incerteza neste caso é representada pelo conhecimento incompleto do sistema sobre os estados.

O planejador de contingência é utilizado quando a configuração do estado inicial não é totalmente conhecido, devendo o agente adquirir meios de cumprir seus objetivos através de ações e sensoriamento. Outra aplicação é na expressão de ações não determinísticas em uma intercalação entre agir e sentir, onde os efeitos da ação são conhecidos na observação (HOFFMANN; BRAFMAN, 2005). Sendo utilizados para lidar com contingências na execução do plano, como por exemplo em emergências e desastres. Uma desvantagem da representação dos efeitos não-determinísticos da ação por observabilidade parcial é que o agente é incapaz de observar um literal mais de uma vez para assegurar seu valor, ou seja, é incapaz de expressar efeitos de uma mesma ação em momentos distintos como eventos independentes. Sendo assim os mecanismos de recuperação e percepção também ficam limitados.

Uma forma de solucionar um problema de contingência a incerteza é levada ao estado inicial e posteriormente é feita a tradução em problemas de planejamento conformantes, que são resolvidos por planejadores clássicos (ALBORE; PALACIOS; GEFFNER, 2009). De forma que problemas conformantes são aqueles que possuem diferentes situações possíveis e que possuem uma solução capaz de alcançar os objetivos independente das incertezas com uma sequência de ações. As principais referências na área de planejamento de contingências, os planejadores CLG (ALBORE; PALACIOS; GEFFNER, 2009) e Contingent-FF (HOFFMANN; BRAFMAN, 2005). Já o PO-PRP realiza a solução através da adaptação de um planejador não determinístico de observabilidade completa e alega superar o CLG, considerado o estado-da-arte no campo.

## 3.3 Processos de Decisão de Markov

Os processos de decisão de Markov, do inglês *Markov Decision Processes* (MDP) é uma metodologia clássica de representação e solução de problemas probabilísticos. Em

sua representação conceitual, pode ser expressa como um sistema de transição de estados de tempo discreto com um número finito de estados, aonde as transições possuem probabilidade de ocorrência e custo relacionados. Sendo assim, uma MDP pode ser definida, segundo (RUSSELL; NORVIG, 2016),como uma 5-tupla $(S, A, T, R, \gamma)$:

- $S$ é um conjunto finito de estados;
- $A$ é um conjunto finito de ações;
- $T$: $S$ x $A$ x $S \rightarrow [0, 1]$ é uma função de probabilidade de ocorrência de um estado $s' \in S$ ao executar uma ação $a \in A$ em um estado $s \in S$, ou seja $T(s'|s, a)$;
- $R$: $S$ x $A \rightarrow \mathbb{R}$ é uma função que relaciona uma recompensa ou custo à aplicação de uma ação $a \in A$ no estado $s \in S$, $R(s, a)$;
- $\gamma \rightarrow [0, 1]$ é o desconto dado à uma recompensa futura;

Um problema de solução de uma MDP consiste em encontrar uma *política* $\Pi$ que maximize a recompensa média do agente e determine a melhor ação a se tomar em um estado $s$. Sendo assim, uma *política* $\Pi$ relaciona um estado $s$ à uma ação $a$, de forma $\Pi : S \rightarrow A$. Desta forma, o agente inteligente identifica o estado atual, consulta $\Pi$ e executa a ação correspondente. A MDP permite que o sistema parta de qualquer estado possível e atinja seus objetivos. Portanto, o estado inicial apenas é utilizado para originar a execução da política.

As formas tradicionais de solução são a iteração por valor e por política, ambas baseadas na equação de Bellman:

$$V_{i+1}^{\pi}(s) = \max_{a}\{\sum_{s'} T(s'|s, a)(R(s, a) + \gamma V_i^{\pi}(s'))\} \tag{3.1}$$

Na equação 3.1 cada estado é associado a um valor, referente a esperança de recompensa na execução de uma ação $a$ segundo $\pi$ que possui maior expectativa de valor. Ela é dada pela soma das recompensas associadas aos possíveis cenários da execução de uma ação $a$ em um estado $s$ com suas respectivas probabilidades. Para a determinação dos valores é feito um processo iterativo onde inicialmente são associadas recompensas aos estados objetivo e valores nulos ao restante dos estados. Os valores das iterações seguintes $i + 1$ são resolvidas com base nos valores obtidos na iteração anterior $i$.

A solução de uma MDP a partir de Bellman pode ser adquirida de duas formas, com horizonte finito e infinito, sendo que no primeiro são determinados o número de iterações e no segundo é definido um critério de parada. Na solução por iteração de valor, o critério de parada é estabelecido como a convergência dos valores de recompensa de

todos os estados. Em grande parte destes problemas este valor cresce indeterminadamente conforme o número de iterações, a convergência é garantida então com a introdução de um fator $\gamma \rightarrow [0, 1]$ que deprecia as recompensas de estados futuros. Na iteração por política é realizada a iteração de valores monitorando-se a política a cada iteração $\pi(i)$, o critério de parada é expresso na conversão da política, ou seja quando $\pi(i) = \pi(i + 1)$. A complexidade do algoritmo é $O(A \text{ x } S^2)$ por iteração sendo que a solução por política é subótima e exige um número menor de iterações.

O MDP ainda possui grande relevância na área pois possibilita a solução de problemas probabilísticos de forma abrangente. Os procedimentos apresentados são apenas as metodologias clássicas de solução por MDP, existindo várias outras técnicas mais recentes como o PROST (KELLER; EYERICH, 2012). Este planejador foi o vencedor das IPPC-2011 e IPPC-2014 sendo considerado o estado-da-arte em planejamento por MDPs.

## 3.4 Planejamento FOND

Um problema do tipo FOND (*Fully Observable Non-Deterministic*) é não determinístico e totalmente observável. Ou seja, o sistema tem total visibilidade de todos os estados e é capaz de discernir entre eles durante a execução do plano. As ações do FOND podem conter efeitos alternativos em que as probabilidades não são levadas em consideração, e os efeitos ocorrem aleatoriamente (MUISE; MCILRAITH; BECK, 2012).

Um exemplo de aplicação em um problema FOND é resolver a situação onde um robô que possui imprecisão em sua movimentação se desloca em um ambiente dividido em células. Neste domínio a imprecisão ocorre de forma diferente conforme a orientação do movimento, com isto a célula em que o robô se encontrará após a execução da ação só pode ser determinado posteriormente a ação. Os prováveis efeitos de cada tipo de movimentação são conhecidos porém as suas probabilidades de ocorrência são desconhecidos ou irrelevantes. O ambiente que este robô se encontra apresenta células perigosas que devem ser evitadas. O robô deve então encontrar uma forma de chegar a uma determinada célula sem passar pelos locais perigosos. Um problema de planejamento FOND pode ser definido como uma 4-tupla $(\mathcal{V}, s_0, s_*, \mathcal{A})$, onde:

- $\mathcal{V}$ é um conjunto de variáveis $\nu$ em um domínio finito $D_v$;
- $s_0$ é o estado inicial;
- $\varsigma_*$ é um estado objetivo;
- $\mathcal{A}$ é um conjunto de todas as ações.

Nele as variáveis $\nu$ podem estar $indefinidas(\perp)$ nos estados $s$. Caso um estado apresente todas suas variáveis definidas, ou seja $s(\nu) \neq \perp$, ele é denominado *estado*

*completo*, se o mesmo não ocorrer ele é designado como *estado parcial*. Para este trabalho, as definições originais foram adaptadas diferenciando a nomenclatura de estados completos $s$ e estados parciais $\varsigma$. Esta modificação tem o intuito de diferir de definições anteriores e assim facilitar a compreensão do texto.

O $s_0$ é um estado completo uma vez que o problema é totalmente observável. O $\varsigma_*$ é um estado parcial, pois ele representa um conjunto mínimo de variáveis que devem ser atendidas para definir o propósito do plano. As $\mathcal{A}$s são todas as ações, determinísticas ou não, que o agente é capaz de realizar. Elas são representadas a partir de causa e efeito do mesmo modo que na representação clássica, onde a causa é exprimida através de precondições $Pre_a$ e efeitos $Eff_a$ que podem ter diferentes desfechos $e \in Eff_a$. Estes efeitos são expressos utilizando a cláusula *oneof*, que significa que *uma das* alternativas irá necessariamente ocorrer como consequência de uma ação não-determinística. Além disso, as variáveis podem ser representadas por predicados assim na representação clássica. No domínio do robô impreciso, a ação não-determinística de se mover para a direita, o robô deixa de se posicionar em um local e se estabelece em outro dentre uma série de alternativas, esta ação é definida através do operador como:

```
mover-direita-não-deterministicamente (x1,x2,x3,x4)
        precond: localrobo(x1), direita(x1,x2),
                 direita(x2,x3), acima(x2,x4)
        efeito : ¬localrobo(x1)
                 oneof(localrobo(x2), localrobo(x3),
                       localrobo(x4))
```

As definições de 2.3 a 2.7 foram extraídas de (MUISE; MCILRAITH; BECK, 2012), e remetem a operações envolvendo estados parciais para a representação de problemas FOND. Elas são de grande importância pois auxiliam o processo de construção de políticas por planejadores.

**Definição 2.3** Para dois estados parciais $\varsigma$ e $\varsigma'$, $\varsigma$ engloba $\varsigma'$ , definido como $\varsigma \models \varsigma'$, se e somente se $\forall \nu \in \varsigma' \neq \bot, \varsigma(\nu) = \varsigma'(\nu)$.

**Definição 2.4** Para dois estados parciais $\varsigma$ e $\varsigma'$, $\varsigma$ é dito consistente em $\varsigma'$ , definido como $\varsigma \approx \varsigma'$, se e somente se $\forall \nu \in \mathcal{V}, \varsigma(\nu) = \varsigma'(\nu) \vee \varsigma(\nu) = \bot \vee \varsigma'(\nu) = \bot$.

**Definição 2.5** Para dois estados parciais $\varsigma$ e $\varsigma'$, sendo $\varsigma \approx \varsigma'$, o resultado da aplicação de $\varsigma$ em $\varsigma'$ é definido como $\varsigma'' = \varsigma \oplus \varsigma'$ e é determinado da forma:

$$\forall \nu \in \mathcal{V}, ((\varsigma'(\nu) \neq \bot) \Rightarrow (\varsigma''(\nu) = \varsigma'(\nu))) \vee$$

$$((\varsigma'(\nu) = \bot) \Rightarrow (\varsigma''(\nu) = \varsigma(\nu)))$$

**Definição 2.6** Uma ação $a$ é aplicável em $\varsigma$ desde que $\varsigma \models Pre_a$. E possivelmente aplicável caso $\varsigma \approx Pre_a$. A progressão de estado para um efeito $e \in Eff_a$ é estabelecido

como $\varsigma' = Prog(\varsigma, a, e) = (\varsigma \oplus Pre_a) \oplus e$.

**Definição 2.7** Um estado $\varsigma$ regride para uma ação $a$ com efeito $e \in Eff_a$, caso $\varsigma \approx e$. A regressão denominada $\varsigma' = Reg(\varsigma, a, e)$ é estipulada como:

$$\forall \nu \in \mathcal{V}, ((Pre_a(\nu) \neq \bot) \Rightarrow (\varsigma'(\nu) = Pre_a(\nu)))\vee$$

$$((Pre_a(\nu) = \bot) \wedge (\varsigma(\nu) = e(\nu)) \Rightarrow (\varsigma'(\nu) = \bot)))\vee$$

$$((Pre_a(\nu) = \bot) \wedge (\varsigma(\nu) \neq e(\nu)) \Rightarrow (\varsigma'(\nu) = \varsigma(\nu)))$$

O estado-da-arte de planejamento para problemas FOND é a solução por meio da *determinização* das ações $a \in \mathcal{A}$ para cada desfecho provável, resultando em ações determinísticas $a' \in \mathcal{A}'$ que são resolvidas por planejadores clássicos. Desta maneira, cada ação determinizada representa uma combinação de precondição $Pre_a$ e um efeito $e \in Eff_a$. Sendo assim, a determinização da ação mover-direita, expressa anteriormente, é retratada por:

```
mover-direita-x2 (x1,x2,x3,x4)
        precond: localrobo(x1), direita(x1,x2),
                 direita(x2,x3), acima(x2,x4)
        efeito : ¬localrobo(x1), localrobo(x2)

mover-direita-x3 (x1,x2,x3,x4)
        precond: localrobo(x1), direita(x1,x2),
                 direita(x2,x3), acima(x2,x4)
        efeito : ¬localrobo(x1), localrobo(x3)

mover-direita-x4 (x1,x2,x3,x4)
        precond: localrobo(x1), direita(x1,x2),
                 direita(x2,x3), acima(x2,x4)
        efeito : ¬localrobo(x1), localrobo(x4)
```

Após a determinação das ações, o planejador FOND é capaz de resolver um plano a partir de um planejador clássico. A ideia básica desta técnica consiste na geração de um plano clássico que leve o sistema ao seu objetivo, ele é capaz de atingir os objetivos caso os efeitos alternativos coincidam com os escolhidos pelo planejador. O plano sequencial encontrado é de fato um *plano fraco* em um meio determinístico, porém caso ocorra o tratamento de todos os efeitos alternativos com novos planos suplementares é possível encontrar uma solução para que o sistema alcance seus objetivos independente dos efeitos apresentados. Este procedimento tende a formar *planos fortes*.

O planejamento FOND se assemelha a um replanejamento que atua de forma *offline*. A diferença entre eles fica explícita na perspectiva de otimização do plano que é viabilizada pela deliberação das consequências. Mecanismos de seleção de planos, eva-

são de bloqueio e simplificação do planejamento são os principais fatores que diferem os planejadores que utilizam desta técnica.

## 3.5 Planejamento de Ciclos Fortes

Por conta da complexidade computacional, as metodologias clássicas de solução de MDPs são aplicadas em problemas com um número reduzido de estados. Sendo assim, existem algoritmos alternativos que oferecem diferentes aproximações e representações na resolução de problemas deste tipo que buscam uma melhor escalabilidade. O Planejamento de Ciclos Fortes é uma metodologia recente de problemas probabilísticos focada principalmente em atingir os objetivos enquanto evita as situações com bloqueio. A estratégia neste tipo de técnica é a solução de problemas desta classe por meio da indução de laços no plano de ação para maximizar as chances de chegar aos objetivos (MUISE; MCILRAITH; BECK, 2012). Uma vantagem deste tipo de representação é que apenas os estados atingíveis a partir do estado inicial são analisados e gerados conforme a necessidade.

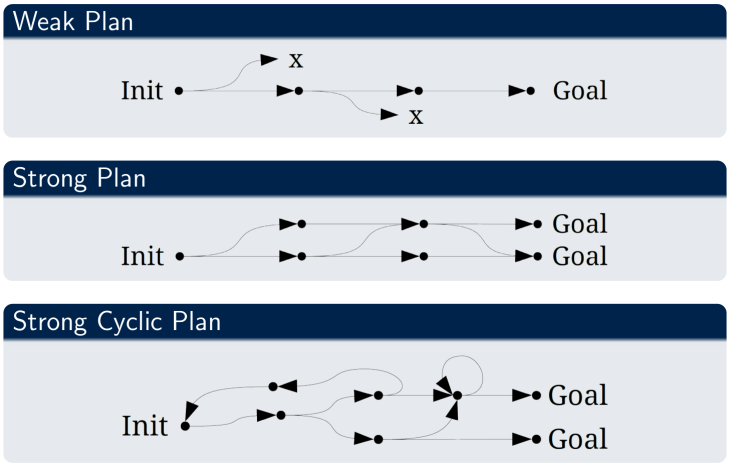



Figura 3 – Comparação entre os diferentes tipos de solução: solução fraca (*weak plan*), solução forte (*strong plan*) e solução ciclica forte (*strong cyclic plan*). Imagem retirada da apresentação de (MUISE; MCILRAITH; BECK, 2012).

Existem três tipos de solução para um plano, são eles: Plano Fraco, Plano Forte e Plano Cíclico Forte (CIMATTI et al., 2003). Em um plano fraco, o agente é capaz de atingir seus objetivos em determinadas sequências de consequências. Um plano forte é dito como um plano preparado para atingir sua finalidade independente das consequências em um número finito de ações. Já um plano cíclico forte é aquele em que o agente assegura alcançar seu propósito após um número infinito de ações. A Figura 3 expressa graficamente os diversos tipos de planos. A solução ideal para problemas é o plano forte, na ausência de tais soluções o papel do planejador é encontrar um plano cíclico forte se possível. Este processo consiste em prevenir os bloqueios evitáveis.

### 3.5.1 Planejador PRP de Ciclos Fortes

O PRP (MUISE; MCILRAITH; BECK, 2012), do inglês *Planner for Relevant Policies*, é uma ferramenta de planejamento desenvolvido para solucionar problemas FOND. Ele é considerado o estado-da-arte na sua área e é baseado no antecessor do título, o planejador FIP (FU et al., 2011). Seu nome remete a sua proposta de providenciar soluções por meio de políticas compactas. Para isto, o planejador busca operar com estados parciais com o menor número possível de variáveis ao invés de estados completos, o que requer a utilização das definições 2.3 a 2.7.

A finalidade do PRP é a produção de uma política $P(s) = \langle \varsigma, a \rangle$, capaz de designar uma ação $a$ mais adequada a ser executada em um estado $s \models \varsigma$, sendo que, a política deve abranger todos os estados alcançáveis pelo sistema a partir do $s_0$ até $\varsigma_*$. O algoritmo 2 mostra a estratégia utilizada pelo PRP para determinar uma política adequada ao sistema de planejamento que será descrito posteriormente.

**Algoritmo 2** Geração de Ciclos Fortes pelo PRP, retirado de (MUISE; MCILRAITH; BECK, 2012)

**Entrada:** $\mathcal{V}, s_0, \varsigma_*, \mathcal{A}$
**Saída:** $P$
**enquanto** $P$ *muda* **faça**
  $Aberto \leftarrow \{s_0\}$
  $Visto \leftarrow \{\}$
  **enquanto** $Aberto \neq \emptyset$ **faça**
    $s \leftarrow Aberto$.pop()
    **se** $s \not\models \varsigma_* \wedge s \notin Visto$ **então**
      $Visto$.adiciona($s$)
      **se** $P(s)$ é $indefinido$ **então**
        GerarParesPlanos($\mathcal{V}, s, \varsigma_*, \mathcal{A}$,P)
      **se** $P(s)$ é $definido$ **então**
        $(\varsigma, a) \leftarrow P(s)$
        **para** $e \in Eff_a$ **faça**
          $Aberto$.adiciona($Progride(s, a, e)$)
  ProcessaDeadEnds()
**retorne** P

A função $GerarParesPlanos$ realiza a computação da *instância* da política $P$ que englobe um estado $s'$. Para isto, é realizado um planejamento clássico que leva um $s'$ até $\varsigma_*$ a partir de ações determinizadas $a \in \mathcal{A}'$. O planejamento clássico incorporado em PRP é um planejador de busca progressiva adaptado que possui uma heurística que almeja planos com o menor número de ações até o objetivo e incorpora mecanismo de evasão de bloqueios. Após a geração do plano completo, é extraída a primeira ação $a'$ e com sua versão não determinitizada pode ser definida a política do estado $s'$. Um artifício para compactar a política pelo PRP é a representação do estado $s'$ por um estado parcial $\varsigma'$, denominado *estado relevante*, que expresse de forma relevante a situação de $s'$ e também

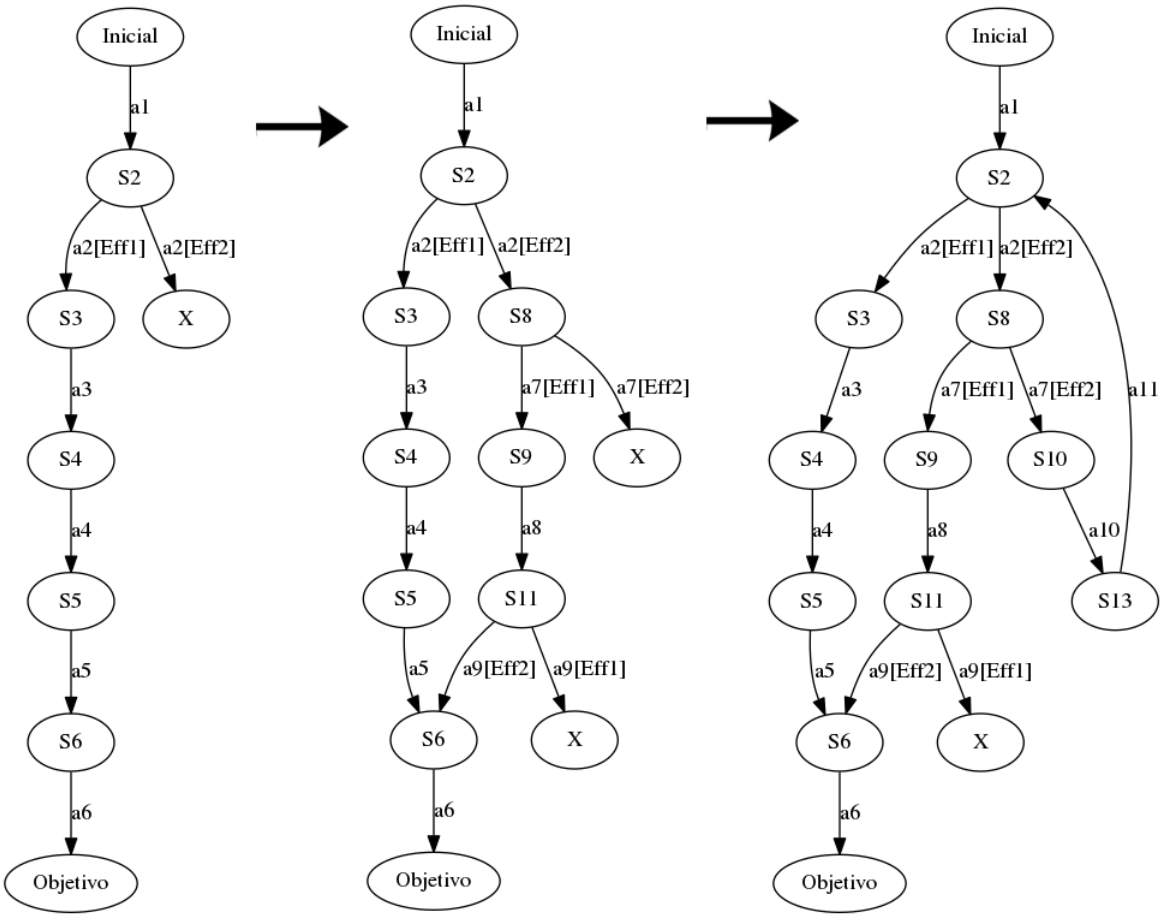



Figura 4 – Exploração dos estados para construção de políticas no PRP.

outros estados que compartilhem situação comum. O meio de extração de $\varsigma'$ utilizado é a regressão do estado parcial objetivo, $\varsigma_*$, até o estado $s'$ segundo as ações geradas no plano clássico. Dessa forma, a função retorna uma instância da política do estado $\varsigma'$ capaz de lidar com o $s'$, $P(s') = \langle \varsigma', a' \rangle$.

Em uma iteração completa do algoritmo 2 para a geração de $P$, são geradas instâncias da política para todos os estados alcançáveis do sistema a partir da execução da mesma. Desta forma, todos os estados não analisados são empilhados e explorados sequencialmente. A cada estado, o algoritmo verifica se ele não foi previamente analisado ou se não compõe o estado objetivo, caso uma dessas condições seja confirmada o estado é descartado. Após as verificações iniciais o estado é adicionado a uma lista de estados explorados e busca-se uma política que o englobe, que é computada através do método $GerarParesPlanos$ caso não esteja definida. Com posse da política $P(a)$, o algoritmo gera novos estados a partir do estado atual com base na progressão do mesmo. A iteração continua até que todos os estados alcançáveis sejam explorados, a figura 4 exemplifica este processo.

A primeira iteração do algoritmo não possui qualquer sistema de tratamento efetivo de $dead-ends$, porém a cada iteração são registrados e evitados na iteração

posterior. Os becos sem saída podem ser encontrados durante a execução do método $GerarParesPlanos$, durante as buscas do planejador clássico ou quando o planejamento falha. A forma de registro de becos sem saída pelo PRP é o mapeamento de pares de estados parciais e ações de forma $F(s) = \langle \varsigma, a \rangle$. Essas combinações são denominadas *proibidas* e são filtradas durante o planejamento em $GerarParesPlanos$. O ponto de parada das iterações é quando não são encontrados mais bloqueios e a política $P(s)$ não muda de uma iteração para a outra.

O algoritmo 2 é suficiente para a construção de uma política que gere planos fortes. Contudo o PRP conta ainda com artifícios adicionais para acelerar a computação de política que podem ser ativados ou desativados durante a chamada do planejador. Dentre eles se destaca o planejamento local, o qual é baseado no FIP. Enquanto o algoritmo 2 gera as políticas para o estado de forma indiscriminada, no planejamento local os efeitos de uma ação são tratadas de forma especial. Em uma determinada situação é realizada a geração da política de um estado $s$ de forma convencional retornando uma determinizada ação $a$ para o efeito calculado $e$ que também possui um efeito $e'$. A consequência da aplicação da ação determinizada $a$ planejada é $s' = Prog(s, a, e)$ que também possui o efeito não selecionado que resulta em $s'' = Prog(s, a, e')$. No planejamento local, a geração de políticas busca levar o estado $s''$ ao estado $s'$ visando simplificar e acelerar o processo. Na prática este procedimento força a geração de ciclos no plano. Caso não se encontre um plano de $s''$ a $s'$, a geração de políticas ocorre normalmente.

Em comparação aos planejadores *online* que utilizam replanejamento, o planejador PRP é capaz de aumentar a chance de se atingir os objetivos. Em alguns problemas mais complexos que são exigidos um grande número de replanejamentos, o tempo total de planejamento do PRP pode ainda ser inferior. Com relação ao seu antecessor FIP, ele se mostrou mais eficiente e eficaz, sendo considerado o estado-da-arte para problemas FOND. Este sucesso é atribuído principalmente a sua forma de tratamento de situações de bloqueio e a representação por estados *relevantes*. Existem quatro expansões do PRP original Cond-PRP, PO-PRP, PROB-PRP e MA-PRP, que possuem as seguintes características:

- O Cond-PRP (MUISE; MCILRAITH; BELLE, 2014), *Conditional PRP*, trata ações com efeitos condicionais. Ele foi a primeira expansão e atualmente está presente nativamente no PRP;
- O PO-PRP (MUISE; BELLE; MCILRAITH, 2014), *Partial Observability PRP*, abrange planejamento de contingência a partir da tradução do problema de observação incompleta para FOND.
- O Prob-PRP (CAMACHO et al., 2015), *Probabilistic PRP*, soluciona problemas probabilísticos do tipo MAXPROB.

- O MA-PRP (MUISE et al., 2015), *Multi-Agent PRP*, gera planos para agentes em ambiente multiagente, onde as ações dos outros agentes são conhecidas mas não controláveis.

Tanto PRP quanto suas extensões são ferramentas com código aberto disponíveis *online*[1]. Elas são *prontas para uso*, necessitando apenas a execução prévia de um script de compilação do código. Também estão disponíveis gratuitamente os artigos referente a cada uma delas. O trabalho é suportado por Christian Muise que também encoraja a comparação e utilização do $software$. Vale destacar que seu código ainda passa por eventuais aperfeiçoamentos, o que mostra sua relevância na área.

### 3.5.2 Planejador PROB-PRP

O Planejador Prob-PRP é uma extensão do planejador PRP que considera a expressão de probabilidades em seu planejamento. Ele busca a solução de um grupo de problemas em que o critério de desempenho seja dado pela probabilidade de se atingir os objetivos, denominados MAXPROB. Os MAXPROB são problema probabilísticos equivalentes a uma MDP de horizonte infinito sem a presença de recompensas ou coeficiente de depreciação. O Prob-PRP possui modificações pontuais em relação ao PRP convencional. Ele ainda utiliza o algoritmo 2 para geração de políticas, com as seguintes alterações:

- A função $GerarParesPlanos$, tem a heurística do planejador clássico modificado para maximizar a função $log(perspectiva)$. A $perspectiva$ é pela probabilidade do plano gerado conduzir o sistema até determinado estado podendo ser calculada de forma $PP(s_n) = \Pi_{i=0}^{n-1} T(s_{i+1}|s_i, a)$, sendo $s_n$ o ponto da busca, $s_0$ estado em que a política é analisada, T a função de probabilidade de transição do estado $s_i$ ao estado $s_{i+1}$ por meio da execução de $a$. Desta maneira ela é o produto da probabilidade de ocorrência dos efeitos selecionados até o estado buscado. Uma forma implícita de representação da $perspectiva$ é a função utilizada pelo Prob-PRP chamada $log(PP) = \Sigma_{i=0}^{n-1} log(T(s_{i+1}|s_i, a))$ que possui vantagem computacional. Inevitavelmente ocorre a redução de $log(perspectiva)$ com o tamanho do plano em problemas probabilísticos, uma vez que $T \rightarrow [0, 1]$, o que reduz o espaço de buscas;
- Ao final do algoritmo é feita uma última iteração para tentar lidar com os estados que não poderam ser tratados previamente. Nesta iteração os estados *proibidos* são ignorados e a detecção de bloqueios é desativada. Este mecanismo é necessário principalmente na presença de bloqueios inevitáveis em que os mecanismos de processamento impedem que o planejamento explore áreas próximas.

[1] https://bitbucket.org/haz/planner-for-relevant-policies/wiki/Home

O vencedor da IPPC-2011 e IPPC-2014 foi o planejador PROST, porém a meta da competição foi a solução de problemas de maximização de recompensa em MDPs. O foco do PROST não é voltado para solução de problemas orientados a objetivo, sendo assim ele não é otimizado para solução de problemas como o MAXPROB. Portanto, os autores do Prob-PRP se compararam com o RFF (TEICHTEIL-KOENIGSBUCH; INFANTES; KUTER, 2008), vencedor da IPPC-2008, por o considerarem o estado-da-arte. Nesta comparação o Prob-PRP se mostrou superior em problemas que ele foi capaz de solucionar, ou seja, nos domínios que não ultrapassou a memória ou tempo máximo. Em alguns domínios chegou a apresentar resultados melhores com diferença de magnitudes de tempo.

## 3.6 PPDDL

O *PPDDL (Probabilistic Planning Domain Language)* é uma formalização capaz de representar problemas probabilisticos extendendo o PDDL. Esta é a linguagem de entrada aceita pelo Prob-PRP, que será utilizada durante o trabalho. Na PPDDL, as probabilidades são simbolizadas por efeitos condicionais associados a números reais de 0 à 1 e a recompensa por uma variável numérica reservada. Nesta convenção um problema de movimento impreciso probabilístico pode ser expresso através da seguinte sintaxe:

```
(:action  mover-direita-probabilisticamente
    :parameters (?x1 ?x2 ?x3 ?x4)
    :precondition (and(localrobo ?x1)(direita ?x1 ?x2)
                      (direita ?x2 ?x3)(acima ?x2 ?x4))
    :effect (and (not(localrobo ?x1))
                 (decrease (reward) 0.8)
                 (probabilistic 0.6 (localrobo ?x2)
                                0.25 (localrobo ?x3)
                                0.15 (localrobo ?x4))))
```

O PPDDL foi o padrão de linguagem até a IPPC-2008, sendo posteriormente substituída pela RDDL (*Relational Dynamic Influence Diagram Language*). O RDDL é uma linguagem que representa problemas probabilísticos na forma de MDP fatorada com intuito de eliminar a restrição sobre ações probabilisticas concorrentes do PPDDL (SANNER, 2010). No entanto, o objetivo do RDDL não é substituir o PPDDL, mas apenas exprimir uma classe de problemas explorados nas IPPCs de forma mais fácil. Mesmo assim, o PPDDL ainda é bastante utilizado por vários planejadores orientados a objetivo. Observando a representação acima pode-se concluir que o PPDDL pode também ser utilizado na representação de vários problemas solucionados por MDPs, inclusive a IPPC disponibiliza um *script* de tradução do PPDDL para RDDL.

## 3.7 Controlador por Máquina de Estados

Uma Máquina de estados finitos, do inglês *Finite State Machine* (FSM), é um modelo de computação representado por um número finito de estados e transições. A FSM é uma máquina abstrata que recebe um conjunto de símbolos alfabéticos de entradas, chamados *string*, e os processa sequencialmente de acordo com as funções de transição. Depois de processar todas as entradas, se ela terminar em um dos estados de meta, a máquina aceitará a sequência, caso contrário a rejeitará. Uma FSM pode ser representada graficamente como um grafo no qual os nós são equivalentes aos estados e as transições às arestas. Uma máquina de estados do tipo Autômato Finito Determinístico (DFA) é definido em (HOPCROFT; MOTWANI; ULLMAN, 2006) como uma tupla $(Q, \Sigma, \delta, q0, F)$, onde:

- Q é um conjunto finito de estados;
- $\Sigma$ é um conjunto finito de símbolos, chamado alfabeto;
- $\delta$: $Q$ x $\Sigma \rightarrow Q$ é uma função de transição determinística;
- $q_0 \in Q$ é o estado inicial;
- $F \subseteq Q$ é um conjunto de estados que completam o objetivo.

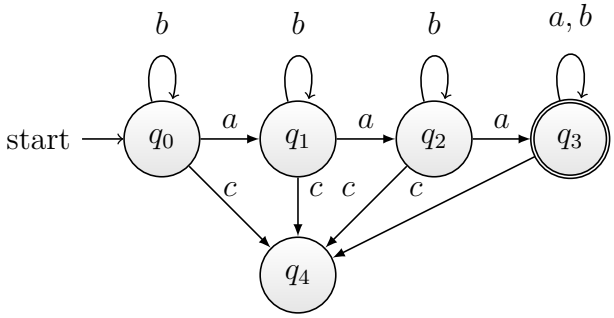



Figura 5 – Autômato Finito Determinístico (DFA)

A figura 5 mostra um DFA com $(\{q_0, q_1, q_2, q_3, q_4\}, \{a, b, c\}, \delta, q_0, \{q_3\})$, com $\delta$ sendo descrito pelas arestas do gráfico. Este autômato aceita qualquer string que tenha mais de três $a$ e nenhum $c$. Quando uma máquina de estados é extraída da política, ela pode ser expressa por elementos de planejamento com as seguintes relações:

- $Q \supset s$, onde $s$ é um estado *completo único* dado por um arranjo de literais;
- $\Sigma = \mathcal{A}'$, onde $\mathcal{A}'$ é um conjunto de ações *determinalizadas* $a'$;
- $\delta$: $Q$ x $\Sigma \rightarrow Q$, com $(\delta(s'|s, a'))$, ou seja o mapeamento de evolução dos estados.

- $F$, é um conjunto de estados objetos representado por estado *parcial* que contém os literais almejados.

Por definição a FSM gerada é uma máquina não determinística, uma vez que o formalismo da DFA requer que as transições sejam definidas para todo os símbolos do alfabeto, no entanto a política define apenas uma ação por estado. Para simplificar as explicações, a FSM resultante será considerada uma DFA relaxada. Esta máquina pode ser construída simulando a execução da política, a partir do estado inicial até os estados objetivo para cada resultado possível das ações. Os autores do planejador do PRP (SARDINA; D'IPPOLITO, 2015) disponibilizaram seu código-fonte contendo um *script* usado para sua validação. Este script gera uma FSM, em formato DOT, a partir de políticas em domínios FOND. O método de geração de FSM usado por esse *script* é descrito pelo algoritmo 3. Esse algoritmo é semelhante ao apresentado em (IOCCHI et al., 2016), no qual é usado para converter políticas em Planos de Rede de Petri.

**Algoritmo 3** Tradução de Política para FSM

**Entrada:** $s_0, \varsigma_*, politica$
**Saída:** $fsm$

```
Aberto ← {s_0}
Visto ← {s_0,ς_*}
adicionaEstado(s_0, fsm)
adicionaEstado(ς_*, fsm)
enquanto Aberto ≠∅ faça
    s ← Aberto.pop()
    a ← avalia(s,politica)
    se a = {} então
        adicionaEstadoMorto(s, fsm)
    se não
        para cada e ∈ Eff_a faça
            s' ← progride(s, e)
            se ς_* ⊆ s' então
                s' ← ς_*
            se não se s' ∉ Visto então
                Visto.adiciona(s')
                adicionaEstado(s', fsm)
                Aberto.adiciona(s')
            adicionaTransição(s, s', a, fsm)
retorne fsm
```

O algoritmo 3 traduz uma política para uma máquina de estados. O procedimento realizado é empilhar e explorar os estados inexplorados iterativamente. Na exploração do estado $s$, a *política* é consultada retornando uma ação $a$. Na hipótese de não ser encontrada, o estado não é tratado pelo política e possivelmente é um ciclo infinito ou um beco sem saída. Posteriormente é realizada a simulação de cada consequência $e$ dentre o conjunto de possibilidade $Eff_a$ de $a$, que leva a outro estado $s'$. Em seguida é analisado se

$s'$ é um estado objetivo, se confirmado o estado é tratado com um estado objetivo comum $\varsigma_*$. Caso $s'$ não tenha sido explorados anteriormente ele é adicionado aos estados visitados e para a pilha de inexplorados. Então é adicionado a transição de $s$ para $s'$ conforme $a$. O algoritmo termina quando todos os estados alcançáveis são explorados. Este sistema de solução por exploração se assemelha a uma iteração do algoritmo 2.

Um controlador por máquina de estados coordena as ações dos agentes ditando a ação a ser executada com base nas ações previamente realizadas e nos $feedbacks$ apresentados. Este controlador considera que apenas um único estado é ativo e que o progresso ocorre conforme a ativação de uma transição por vez. Esta forma de controle é amplamente utilizada principalmente por ser uma técnica com: fácil entendimento, fácil implementação, representação visual, baixa complexidade computacional e adequada a várias situações reais. Algumas limitações também estão presentes neste sistema como a incapacidade de reproduzir ações simultâneas. Isto dificulta sua aplicação principalmente em sistemas multi-robôs e multi-tarefas.

# 4 ROS - Robot Operating System

O ROS (QUIGLEY et al., 2009), do acrônimo *Robot Operating System*, é um *middleware* modular direcionado para propiciar o desenvolvimento colaborativo na área da robótica. Este *framework* age como uma ponte entre o sistema operacional e a aplicação robótica, providenciando recursos esperados de um sistema operacional como: abstração de hardware, comunicação entre processos, controle em baixo nível, implementação de funcionalidades comuns e gerenciamento de pacotes. A figura 6 demonstra o intermédio feito pelo ROS entre os diferentes componentes do sistema, adicionando uma camada de abstração no sistema operacional.

Seus objetivos incluem: permitir a conexão de diferentes *hosts* através de uma topologia ponto-a-ponto, ser multi-linguagem, providenciar ferramentas de análise e configuração de sistemas robóticos, ter uma estrutura minimalista, possuir código aberto e ser gratuito.

Um ponto chave do ROS é o favorecimento da reutilização de códigos dentro da área de robótica, o que oportuniza o aproveitamento do trabalho de outros pesquisadores e direcionamento dos esforços para os componentes em análise. Esta característica é indispensável em sistemas complexos, onde os componentes exigem um desenvolvimento e análise especializados. A estrutura minimalista do ROS permite o aproveitamento de códigos de outros *frameworks* com poucas adaptações e de forma modular. Tudo isso aliado ao suporte de diversas linguagens que fomenta o cooperativismo na área.

O ROS possui também gestão do conhecimento que é realizada em boa parte por meio de sites relacionados ao ROS.org. O ROS-Wiki, que possui documentação colaborativa com informações, tutoriais e divulgação de trabalhos relacionados. No ROS-Answers a comunidade responde perguntas relacionadas ao ROS. As notícias são divulgadas através do *Blog*. Deste modo, a comunidade tem um papel muito importante não só na criação de *softwares* disponibilizados em seus repositórios mas também no suporte da ferramenta.

No ROS, a execução de um sistema contendo diversos executáveis é realizada conforme uma estrutura denominada *grafo de computação*. Ela define a conexão entre

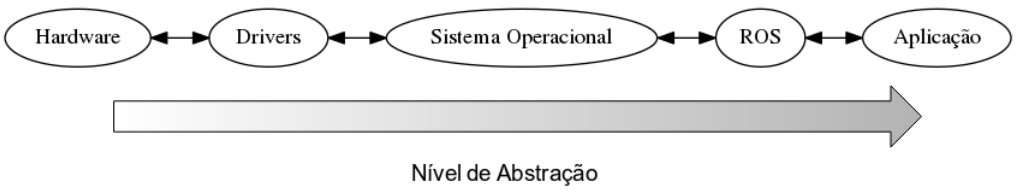



Figura 6 – Posicionamento do ROS no desenvolvimento de *software*

unidades de computação denominados *nós*, que realizam processamento de forma paralela. Esta estrutura é uma rede ponto-a-ponto compostas por nós capazes de trocar informações entre si. A comunicação entre nós ocorre por meio de mensagens e pode ser realizada por *tópicos* ou *serviços*.

A comunicação por tópicos é utilizada quando deseja-se que um nó forneça informações a outros, buscando manter os nós de interesse atualizados com um fluxo contínuo de dados. Os serviços são requisições de computação que um nó realiza a outros, seja ela uma consulta, mudança de estado ou outra operação qualquer (ROS, 2007). Estes serviços funcionam por intermédio de interrupções no sistema, consequentemente possuem prioridade no processamento e necessitam ter computação rápida.

Quanto ao fluxo de informações, a comunicação em tópicos é de forma unidirecional, da origem para os destinos, enquanto nos serviços é bidirecional. Toda esta comunicação é gerenciada por um *nó mestre*, detentor das informações de tópicos e serviços além dos protocolos de rede. Nesta representação o tópico parte do nó que o anunciou para os que se inscreveram. Já o serviço é uma rotina que é anunciada e pode ser chamada por qualquer outro nó da rede.

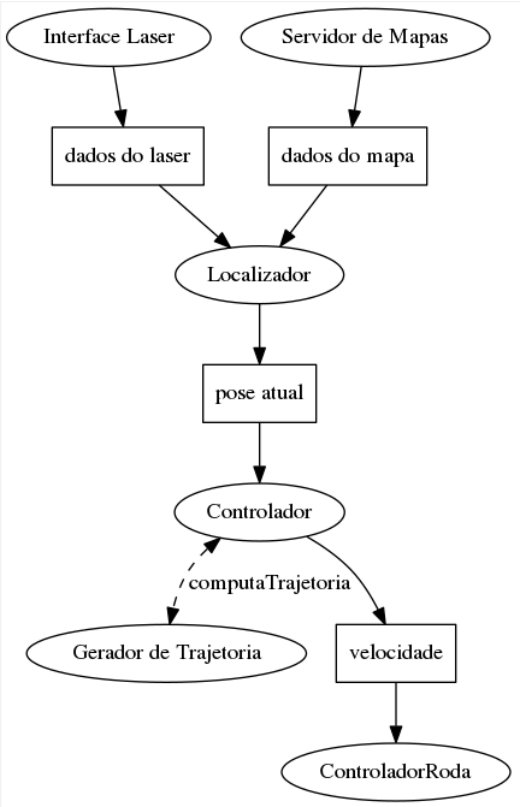



Figura 7 – Grafo de computação de um controle de trajetória hipotético

A figura 7 representa um sistema hipotético cuja função é guiar um robô com restrições de movimento equipado com sensores *laser*. Neste sistema a localização do robô é feita com base em um mapa e em sensores laser. O controlador de trajetória consulta um

gerador de trajetórias e calcula velocidades angulares e lineares para o robô se manter na rota com base na localização atual e a desejada. Por fim, essas velocidades são traduzidas e enviadas para cada roda. Neste grafo os objetos circulares são nós, os quadrados são tópicos e a linha tracejada é um serviço.

Os grafos computacionais possibilitam uma modulação dos sistemas robóticos, o que induz a colaboração. Por exemplo, na aplicação apresentada os executáveis *Laser* e $ControladorRodas$ podem ser parte de um pacote fornecido pela fabricante e ainda $Localizador$, $Controlador$ e $CalculoTrajetoria$ podem ter sido projetos de outros pesquisadores ou os elementos de interesse. Desta forma, as aplicações conseguem apresentar um grau mais elevado de sofisticação através do aproveitamento de técnicas previamente implementadas.

Atualmente, o ROS possui suporte oficial apenas ao Ubuntu com compatibilidade para o Mac OS X, contudo a comunidade tem ajudado no suporte de outros sistemas operacionais baseados no Unix. Ele possui 12 distribuições, das quais quatro estão em sua vida útil, são elas: Indigo, Kinectic, Lunar e Melodic. A política de lançamentos do ROS determina um calendário de lançamentos anuais de distribuições onde são intercalados entre a versão ROS, com suporte de dois anos, e o LTS ROS, com suporte de cinco anos (ROS, 2007). A compatibilidade entre as distribuições ocorre de forma que uma versão ROS é compatível com a versão LTS ROS anterior e as versões LTS não são compatíveis entre si. Hoje ele é uma das principais referências na área de desenvolvimento robótico com milhares de pacotes atraindo o interesse de fabricantes, pesquisadores e desenvolvedores.

Uma das maiores dificuldades vivenciada na utilização do ROS é a falta de compatibilidade entre pacotes gerada principalmente pela falta de suporte em parte de sua comunidade, especialmente nos que possuem desenvolvimento independente. Deste modo, muitos pacotes são desenvolvidos mas perdem a sua compatibilidade com o passar do tempo, de modo a comprometer seu funcionamento. A própria política de lançamentos do ROS já prevê uma retrocompatibilidade limitada entre as distribuições. Desta forma, o pesquisador acaba buscando um conjunto de pacotes compatíveis mesmo que não sejam os mais adequados. A tentativa de adaptação de códigos de terceiros com intuito de compatibilizar *softwares*, na prática, acaba sendo muitas vezes frustradas por conta da falta suporte também de bibliotecas. Isto atrapalha o desenvolvimento colaborativo e faz com que os usuários se mantenham nas distribuições LTS mais bem estabelecidas do *software* até o fim da sua vida útil.

## 4.1 ROSPlan

O ROSPlan (CASHMORE et al., 2015) é um pacote do ROS capaz de gerar e executar planos, construídos por meio de planejamento automático. Esta biblioteca não

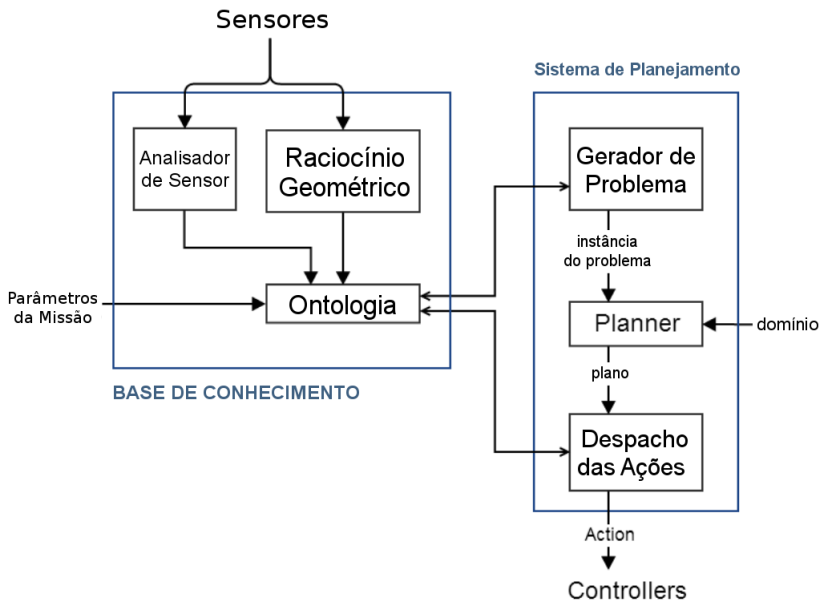



Figura 8 – Arquitetura do ROSPlan. Adaptado de (CASHMORE et al., 2015)

foi pioneira na utilização de planejamento dentro do ambiente ROS, existiram trabalhos anteriores como é o caso de (BERNARDINI; FOX; LONG, 2014). No entanto sua contribuição foi na implementação de uma estrutura generalista que permite a integração entre aplicações e planejadores independentes do domínio dentro do ambiente ROS.

A Figura 8 mostra a arquitetura do ROSPlan, onde é descrito o funcionamento do *software* em dois módulos: a base de conhecimento e o sistema de planejamento. A base de conhecimento traz informações sobre o estado atual do sistema e informações do domínio, elas são adquiridas através das entradas do usuário e inferidas com base nas medições dos sensores. O gerador de problema sintetiza um arquivo problema em PDDL, para guiar o sistema do estado atual até um estado objetivo, cujas informações se encontram na base de conhecimento. O problema sintetizado é passado na chamada do planejador em conjunto com o arquivo domínio e o plano gerado determina as ações despachadas ao sistema robótico. O controlador de ações dita a ação a ser executada e atualiza a base de conhecimento conforme as observações realizadas das consequências da ação. Caso o estado atual não satisfaça as precondições da próxima ação ou a ação falhe o sistema replaneja com base no estado observado.

As informações do problema são inicialmente inseridas no ROSPlan conforme a descrição do mesmo em um formato próprio por meio de comandos que podem ser enviados por via terminal do sistema operacional. Estes comandos normalmente são agrupados em forma de *scripts* de execução do próprio sistema operacional e o arquivo problema em PDDL não pode ser utilizado como entrada. A desvantagem deste método é que além de forçar o usuário a aprender um novo formato, as extensões do PDDL devem

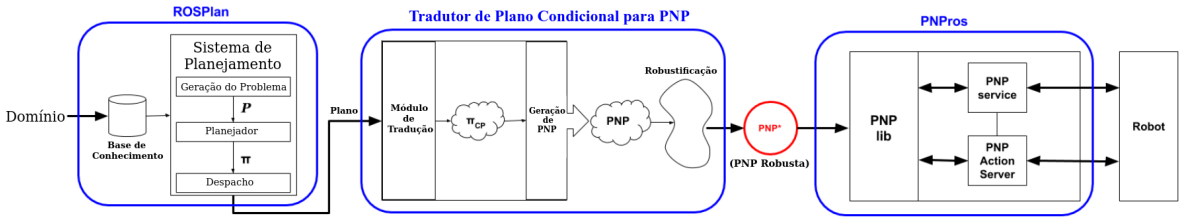



Figura 9 – Arquitetura do ROSPlan Contingent, adaptado de (SANELLI et al., 2017)

ser implementadas no sistema, caso o usuário deseje utilizá-las, sendo que o PDDL já é um padrão bem estabelecido dentre as linguagens para planejadores independentes do domínio. Outra desvantagem é que no caso do ROSPlan o sistema demora a carregá-la, muitas vezes levando mais tempo que o planejamento. A interface do ROSPlan com a aplicação é dada por tópicos de envio de ações e de retroalimentação.

O ROSPlan suportou originalmente o planejador POPF, um planejador determinístico temporal e o aplicou em veículos subaquáticos autônomos. Posteriormente, o suporte foi expandido para o Contingent-FF, que será discutido na próxima sessão. Recentemente a biblioteca anunciou o suporte a vários planejadores, são eles: OPTIC, FF, Metric-FF, Contingent-FF, LPG, Fast Downward, TFD, SMTPlan, and UPMurphi.

## 4.2 ROSPlan Contingent

O ROSPlan Contingent (SANELLI et al., 2017) é uma variação do ROSPlan que utiliza o Contingent-FF e Planos por Rede de Petri (PNP) para a geração e execução de planos contingentes. A PNP é a utilização de Redes Petri para a representação e execução de planos. Nesta rede as transições são disparadas pelo despacho de ações ou pelo seu $feedback$, os lugares têm marcações unitárias e mais de um lugar pode possuir marcação. Ela permite o paralelismo de ações e é conveniente em sistemas multi-robôs. A visualização da Rede é realizada através de uma versão modificada do JARP.

A ferramenta possui a arquitetura disposta na figura 9. As entradas do pacote são os arquivos PDDL do domínio e do problema, dos quais o planejador de contingência calcula um plano condicional. Em seguida o plano é enviado a biblioteca de geração de PNPs, que retorna uma Rede de Petri. Posteriormente, a rede passa por um processo denominado robustificação, onde a Rede é adaptada conforme afirmações condicionais transmitidas ao programa. Com a rede robustificada uma biblioteca de execução de PNPs indica a ação a ser realizada conforme o $feedback$ das observações que determina o resultado dos pontos de decisão.

# 5 Planejamento em VANTs

O problema analisado neste trabalho será a aplicação de VANTs na entrega de pacotes, disposto nas Figuras 10 e 11. Este domínio representa situação comum em aplicações com VANTs, a incerteza no consumo de bateria durante a execução das suas ações em conjunto com a capacidade limitada de armazenamento de energia. A sua relevância vêm da flexibilidade na ordenação das ações e na capacidade de se representar um domínio genéricos para várias situações problema. Neste domínio a navegação é realizada por meio de pontos de referência em um mapa onde as rotas são previamente conhecidas.

O mapa contém ainda estações onde o veículo pode se recarregar, deixar e retirar pacotes. Os pacotes podem ser entregues em ordem arbitrária em seus destinos correspondentes. O objetivo do sistema é entregar todos os pacotes. O consumo de combustível é incerto e varia de acordo com a carga. A falha na operação pode ocorrer inerente a navegação ou quando a bateria atinge um nível mínimo, portanto deve ser evitada a todo custo, pois possui potencial de dano.

O domínio apresentado é modelado em alto nível, onde é determinado um plano de ações em que os meios de execução da ação são deixados para o controlador. Desta forma a resistência à perturbações externas, o desvio de obstáculos, manobras evasivas não são consideradas no planejamento mas impactam indiretamente no feedback do controlador e assim na execução do plano.

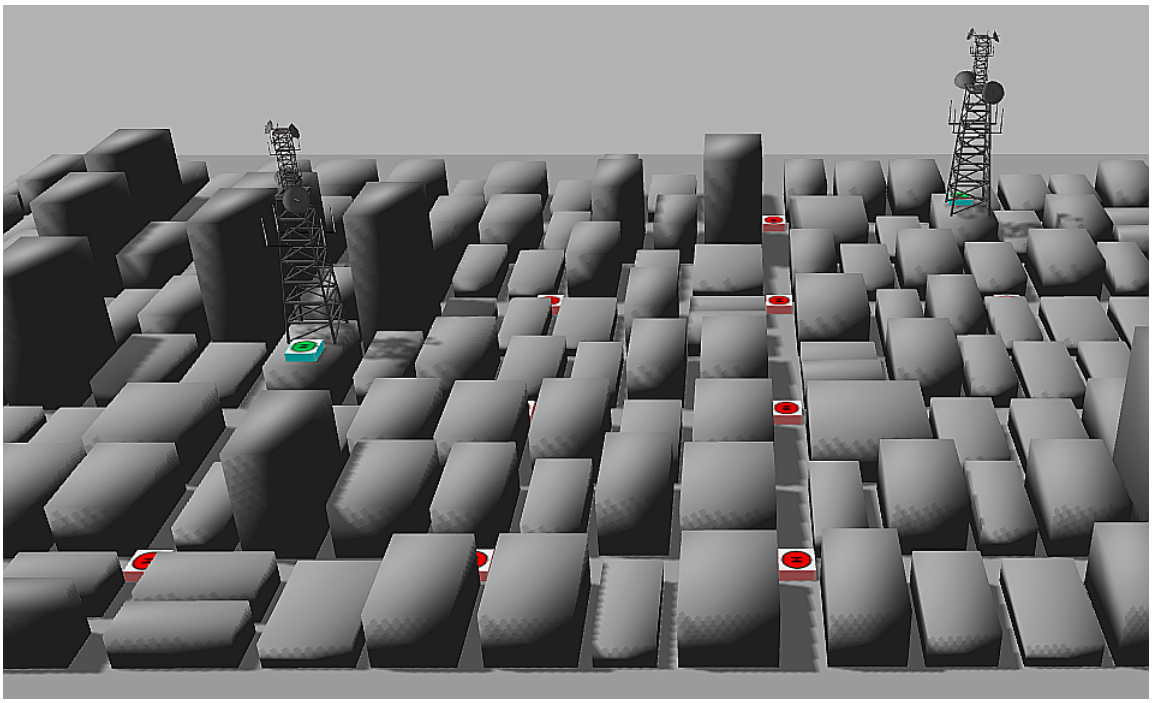


Figura 10 – Ambiente de Entregas com Visão Geral

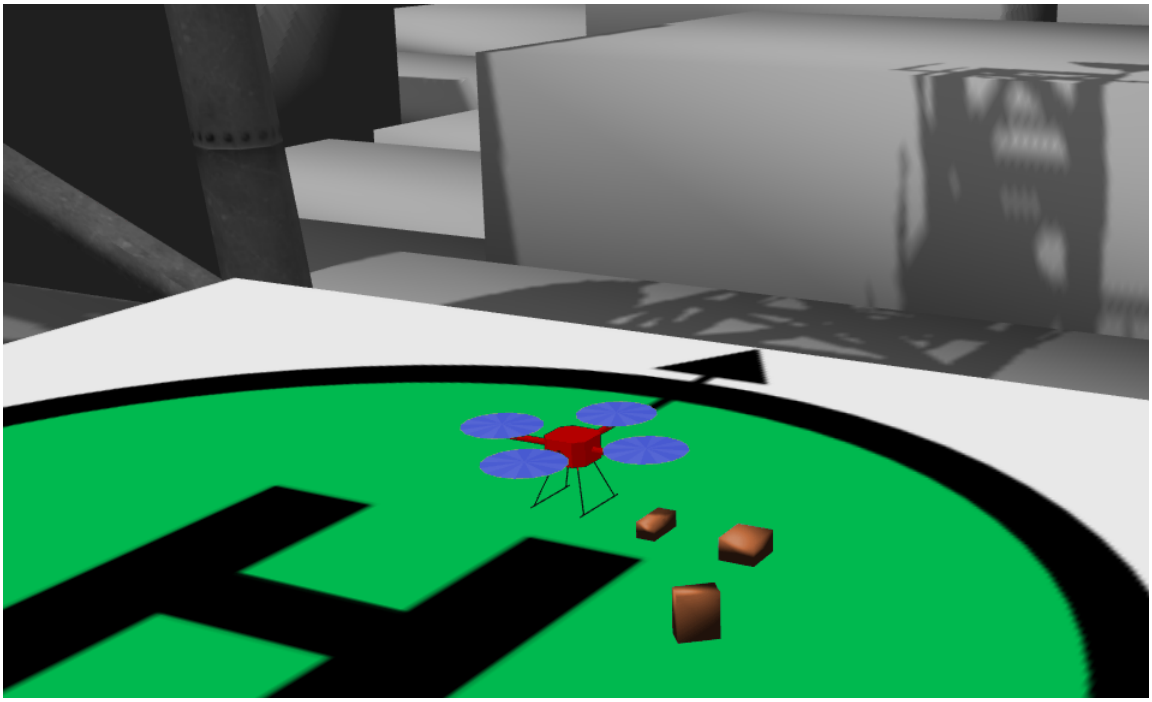


Figura 11 – Ambiente de Entregas com Visão da Estação

Neste modelo o recurso de combustível é discretizado em níveis, por dois motivos: complexidade computacional e suporte do planejador. A modelagem da navegação do VANT será realizada através da abstração em pontos de referências, *waypoints*. A ação de se movimentar horizontalmente é representada por meio de um único operador com diferentes possibilidades de consumo de combustível. Esta incerteza pode estar relacionada a condições climáticas, manobras necessárias ou erro de discretização. Neste operador as possibilidades não variam com os pontos navegados, sendo assim durante a instanciação do problema é interessante posicionar os *waypoints* o mais equidistantes possível, sendo o erro considerado como incertezas. O sistema permitirá recargas infinitas de combustível durante a execução.

## 5.1 Representação em Linguagem de Planejamento

A formalização do problema de entrega de pacotes apresentado previamente foi realizada como um problema probabilístico totalmente observável $(\mathcal{V}, s_0, s_*, \mathcal{A})$, onde as probabilidades se encontram incluídas em $\mathcal{A}$. Esta representação foi codificada em PPDDL e está disposta no Apêndice C.

### 5.1.1 Estrutura Base

As variáveis $\mathcal{V}$ que definem os estados são expressas através dos predicados:

- deliverat (?pkg - package ?wp - waypoint): Local de Entrega, onde um pacote *pkg*

deve ser entregue uma localização *wp*;

- (pkgat ?pkg - package ?wp - waypoint): Localização do Pacote, onde um pacote *pkg* está localizado em *wp*;
- (quadat ?wp - waypoint): Localização *wp* do VANT;
- (inquad ?pkg - package): Pacote *pkg* inserido no VANT;
- (delivered ?pkg - package): Pacote *pkg* entregue;
- (connected ?wp1 ?wp2 - waypoint): Existe uma rota que conecta o ponto *wp*1 ao *wp*2;
- (quadfuel ?fuel - fuellvl): O VANT se encontra com o nível $fuellvl$ de combustível.
- (fuelmapping ?lowerlvl ?higherlvl - fuellvl): Mapeamento dos níveis, onde o nível *lowerlvl* é imediatamente anterior ao *higherlvl*. Sendo o *higherlvl* um nível com mais combustível.
- (maximumfuel ?maxfuel - fuellvl): Nível máximo $maxfuel$ de combustível que o VANT é capaz de armazenar.
- (stationat ?wp - waypoint): Localização *wp* de uma estação.
- (onair): Estado em que o VANT está voando.
- (grounded): Estado em que o VANT está em terra.
- (notloaded): Estado em que o VANT está sem cargas.

As ações não-determinísticas $\mathcal{A}$ que alteram os estados são expressas através dos operadores:

- land (?fuel1 ?fuel2 - fuellvl): Aterriza VANT, consome 1 nível de combustível saindo do nível $fuel2$ para o nível $fuel1$;
- takeoff (?fuel1 ?fuel2 - fuellvl): Decola o VANT, consome 1 nível de combustível saindo do nível $fuel2$ para o nível $fuel1$;
- load-package(?pkg - package ?wp - waypoint): Carrega o VANT com o pacote *pkg* localizado no ponto *wp*;
- deliver-package(?pkg - package ?wp - waypoint): Entrega o pacote *pkg* no ponto *wp*;

- move-to(?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl): Movimenta o VANT com nível de combustível $fuel3$ do ponto $wp1$ para o ponto $wp2$ com duas possibilidades de consumo resultando nos níveis $fuel2$ ou $fuel1$;
- refuel (?wp1 - waypoint ?fuel1 ?fuel2 - fuellvl): Reabastece o VANT de nível $fuel1$ na estação do ponto $wp1$ para sua capacidade máxima $fuel2$.

### 5.1.2 Extensão 1

Na primeira extensão o VANT é capaz de carregar mais de um pacote, porém os pacotes agora possuem peso e impactam no consumo da aeronave. Para isso foram adicionados os seguintes predicados:

- (weightquad ?w - weightlvl): Peso do VANT, a aeronave se encontra com nível de peso líquido $w$;
- (weightpkg ?pkg - package ?w - weightlvl): Peso do pacote, a entrega possui com nível de peso bruto $w$;
- (addweight ?w1 ?w2 ?w3 - weightlvl): Soma de Pesos, o peso $w1$ somado a $w2$ resulta em $w3$;

Os operadores das seguintes ações foram modificados:

- load-package(?pkg - package ?wp - waypoint ?w1 ?w2 ?w3 - weightlvl): Estende o carregamento adicionando um pacote de peso $w2$ no VANT com peso $w1$ resultando em um peso $w3$ ;
- deliver(?pkg - package ?wp - waypoint ?w1 ?w2 ?w3 - weightlvl): Estende o descarregamento removendo um pacote de peso $w2$ no VANT com peso $w3$ resultando em um peso $w1$ ;

O operador $move - to$ poderia ser modificado para incorporar efeitos condicionais dependentes do nível de peso, porém o Prob-PRP e o *script* de tradução de políticas em planos têm dificuldade de lidar com esta tipo de configuração. Consequentemente este operador teve que ser particionado em diversas ações para cada nível de peso.

### 5.1.3 Extensão 2

A segunda extensão incorpora as falhas inerentes a operação de se mover. Este bloqueio é mortal e inevitável mas suas chances de ocorrência ao longo da execução do plano devem ser reduzidas. A possibilidade de ocorrência de falha é bem improvável e

varia com o peso. Na representação utilizada o predicado *operating* simboliza a operação normal do VANT e permite que execute suas ações. Este predicado tem chance de ser negado através de um efeito probabilístico nas ações.

## 5.2 Ambiente de Simulação

A simulação do domínio foi realizada utilizando o pacote do Gazebo dentro do ambiente ROS. A criação de novos ambientes pode ser efetuada pelo próprio pacote que possui formas básicas como cubos, esferas e cilindros além de um repositório com diversos objetos mais complexos como aviões, casas e robôs. A versão do Gazebo compatível com ROS Indigo é a 2.2.3 do ano de 2014, esta variante contém muitos problemas na criação e modificação de ambientes. Como resultado, muitas modificações são realizadas diretamente no arquivo do modelo. Para criação do ambiente de simulação foi desenvolvido um programa em C++ utilizando a biblioteca SDFormat. Esta API tem como objetivo descrever objetos e ambientes robóticos por meio de arquivos XML. A vantagem, para este trabalho, da criação através desta API é o posicionamento automatizado de diversos elementos, o que seria uma tarefa exaustiva se realizado da forma convencional.

O ambiente escolhido para o transporte de pacotes é uma cidade como visto na Figura 10. Como o ambiente apresentado ocorre em um meio urbano, é importante conter características urbanas em sua concepção como: casas próximas, prédio, ruas, quarteirões e interseções. Foi realizado uma representação minimalista destes elementos com intuito de diminuir o esforço computacional envolvido. Desta maneira, a formulação do local foi idealizada a partir de figuras geométricas simples. Esta estratégia é inclusive encorajada pelos desenvolvedores do $software$. A geração do ambiente foi automática com base na organização em quarteirões de 43m x 100m. Os quarteirões possuem duas linhas de edificações cujas dimensões são aleatoriamente variadas dentre uma faixa de valores. Adicionalmente, a variação na altura das construções foi incorporada para dar uma identidade maior ao ambiente. Posteriormente as marcações de *waypoints* são adicionadas.

## 5.3 Plataforma UAV e Navegação

Veículos aéreos não tripulados é uma terminologia geral para vários tipos de veículos como os de asa fixa, dirigíveis e helicópteros. Recentemente, os helicópteros de múltiplas hélices, denominados multirotores, têm se destacado na aviação civil principalmente por sua manobrabilidade, simplicidade mecânica além de decolagem e pouso verticais. O multirotor é um dos modelos naturalmente mais instáveis e desafiadores dentre as aeronaves, sua ascensão só ocorreu com o avanço da tecnologia que possibilitou a minimização dos componentes e avanço da eletrônica embarcada, nas técnicas de controle e nos motores elétricos. O controle de voo, do multirotor, é realizado através da variação das velocidades

relativas de cada hélice. A aplicação comercial dos multirotores atualmente se delimita em aeronaves de pequena e média escala, por conta da sua eficiência energética e da inércia das hélices em grande escala. Sendo assim, a simulação foi realizada em um multicóptero de pequeno porte e com quatro hélices (denominado *quadricóptero*). O modelo de quadricóptero utilizado foi o pacote de terceiros chamado hector_quadrotor[1] (MEYER et al., 2012). Este modelo desenvolvido para operação em conjunto com o Gazebo e ROS, incorporando:

- Descrições do quadricóptero como modelos visuais em 3D, cinemática e dinâmica;
- Controladores de vôo de alto nível e baixo nível;
- Modelo de Sensores como IMU, laser, GPS, sonar e câmera;
- Interfaces de teleoperação;
- Demonstrações de utilização no Gazebo em um cenário interno e externo;

No trabalho realizado foi utilizado o drone padrão da biblioteca desfrutando principalmente do sensor laser Hokuyo Utm30lx, da IMU e do sonar. A IMU é composta por um acelerômetro e giroscópio, com papel de estabilizar o voo dessa aeronave. O sensor sonar é posicionado na parte inferior com orientação vertical para baixo, tem a função de mensurar a altitude do quadricóptero em relação aos objetos em nível inferior. O sensor laser possui faixa de medição de $-135^o$ a $135^o$ em relação a sua orientação sensoriando a distância de 1081 pontos, oportunizando o desvio de obstáculos, mapeamento e localização. Foi utilizado o controlador de alto nível simples oferecido pela biblioteca. Sua forma de operação consiste em receber os valores de referência das velocidades lineares e angulares, e atribuir as velocidades correspondentes aos motores. Esta informação é passada ao controlador por meio do tópico *cmd_vel*.

Em uma das demonstrações do pacote, apresenta-se um mapeamento 2D através de técnicas de SLAM (*Simultaneous Localization and Mapping*) em um ambiente interno mediante a navegação por teleoperação. Por simplificação esta demonstração foi adaptada para mapear o ambiente proposto pelo trabalho em uma altura fixa. A variação nas dimensões das construções implementadas no ambiente auxiliam a localização e mapeamento através de SLAM, pois facilita a diferenciação dos locais. Um dos problemas vivenciados foi causado pelas grandes dimensões do mapa e pela técnica de mapeamento escolhida, o qual causava consumo excessivo de memória RAM que era limitado pela capacidade da máquina de 8GB. Empiricamente definiu-se a resolução de 0.25m por pixel do mapa gerado. Outra dificuldade causada pelo tamanho do mapa foi o extenso trabalho de mapeamento por controle manual. O mapeamento do ambiente utilizado necessitava que a

[1] http://wiki.ros.org/hector_quadrotor

operação fosse realizada em baixas velocidades, o que prolongava o tempo por tentativa. As tentativas estavam sujeitas ainda ao acúmulo de erro de medição e erros de operação, causados principalmente por ilusão ótica. O resultado está disposto na Figura 12.

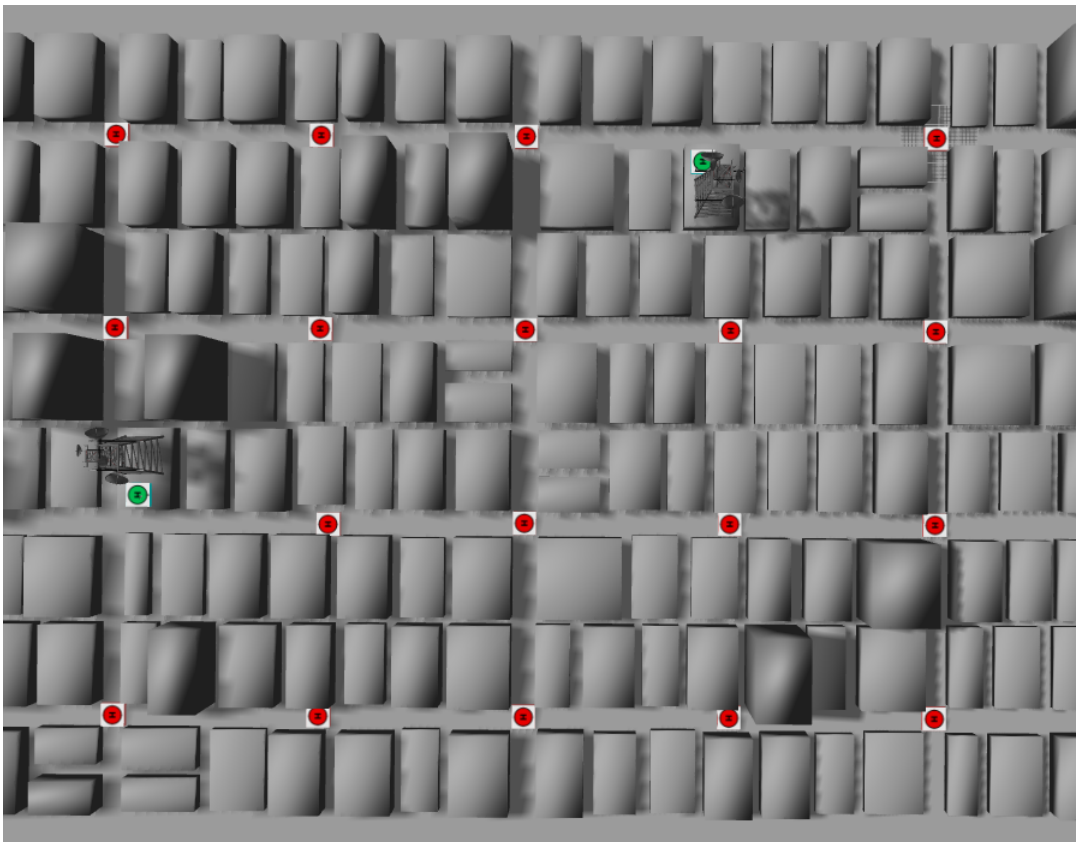
Por motivos de simplificação e conveniência, a navegação é ativada quando a aeronave está em uma altura fixa abaixo do nível das casas, ou seja, a navegação é planar. A navegação entre pontos de referência foi realizada utilizando o ROS Navigation[2], um pacote de navegação 2D do ROS. Este pacote possui módulos de localização e navegação para robôs móveis. Um destes módulos é o AMCL capaz de realizar localização a partir de um mapa de ocupação do ambiente previamente fornecido e da leitura de sensores laser. Outro módulo é o *move_base* que realiza o planejamento de trajetória entre dois pontos com base em um mapa, nos obstáculos captados pelos sensores e parâmetros de configuração. Estes dois módulos foram integrados ao sistema do VANT, tendo seus parâmetros calibrados para a aplicação.

## 5.4 Interação com outros objetos

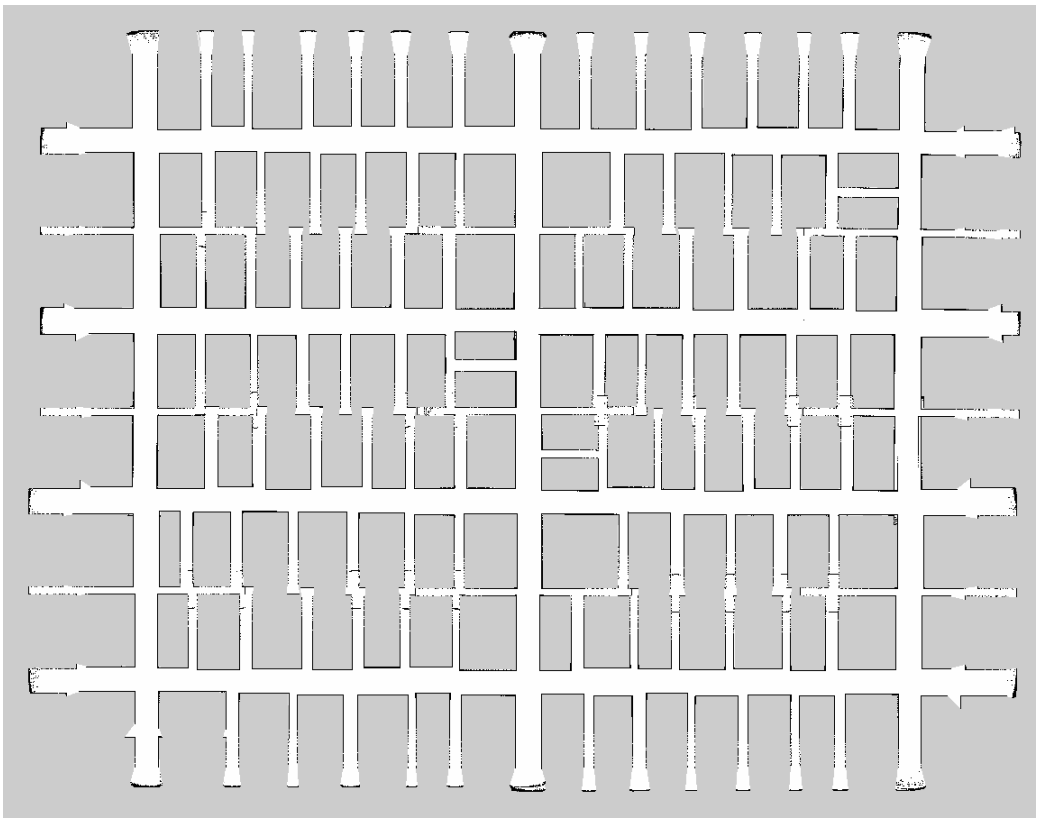
No Gazebo a interação entre diferentes objetos pode ocorrer através de mecanismos físicos como colisão ou através de comandos de *software*. Há duas forma de comandos por *software* através de *plugins* ou por intermédio da comunicação com o ROS. A comunicação por tópicos é utilizada para o posicionamento dos objetos ou de seus elos. Já a comunicação por serviços é mais ampla, ela possibilita que os objetos: sejam criados ou destruídos, tenham seu estado consultado ou ainda que ocorra a aplicação de forças nas juntas. A comunicação com o ROS foi preferida para simular a interação uma vez que os meios de interação não são priorizados neste trabalho.

Na versão do Gazebo utilizada o serviço de criação de objetos não era capaz de desempenhar sua função, porém um *script* em Python disponibilizado era capaz de realiza-lá. Assim sendo, este *script* foi adaptado e incorporado ao ambiente ROS para desempenhar sua tarefa através de tópicos. Os pacotes então apareciam nas posições desejadas no início da simulação. A interação com o pacote ocorria de modo que quando o drone carregava o pacote, o mesmo começava a acompanhar seu movimento e ao descarrega-lo, o pacote deixava de ter este comportamento.

[2] http://wiki.ros.org/navigation



(a) Visão Superior do Domínio Planejado



(b) Mapa extraído

Figura 12 – Mapamento do ambiente por meio de técnicas de SLAM

# 6 DOTPlan - Sistema Proposto de Planejamento

Durante o desenvolvimento do trabalho inicialmente a versão original do ROSPlan foi experimentada em domínios envolvendo VANTs, sendo que o mecanismo de tratamento de incertezas era o replanejamento disponibilizado pela própria ferramenta. Dadas as desvantagens do replanejamento discutidas anteriormente, decidiu-se pela utilização de técnicas mais complexas para tratamento de incertezas. Deste modo, houve a necessidade de incorporar um novo tipo de planejador ao ROSPlan que apresentasse um raciocínio *offline* que implementasse táticas efetivas de maximizar a eficácia e eficiência em atingir os objetivos dentre diversos cenários possíveis. Em meio aos estudos foi lançada uma variante do ROSPlan que utilizava o Contingent-FF e PNP com intuito de gerar planos que tratassem contingências ocorridas durante sua execução. Esta ferramenta contava ainda com um $software$, externa ao ROS, chamado $JARP$ para visualização de Redes de Petri o qual foi adaptado ao PNP.

Ao utilizar esta versão para a aplicação pretendida foi possível perceber que a ferramenta não permitia a observação de predicados que continham variáveis, o que foi implementado e compartilhado com a comunidade. Após a implementação da aplicação no sistema proposto, notou-se que o planejamento por contingência só era capaz de oferecer soluções efetivas em problemas não determinísticos que pudessem ser resolvidos com um número finito de passos. A produção de ciclos de reabastecimento é comum na aplicação estudada neste trabalho, além disto seria interessante a consideração das probabilidades de ocorrências dos efeitos alternativos. A alternativa clássica de planejamento probabilístico é a MDP, porém na busca de planejadores probabilísticos orientados a objetivo optou-se pelo Prob-PRP. O Prob-PRP possui uma complexidade computacional inferior a planejadores por MDPs, além disso a ponderação entre recompensa e risco não é interessante no problema proposto, onde deve-se evitar o risco a qualquer custo.

Efetuou-se por fim a tentativa de adaptação do ROSPlan com PNP para agregar o Prob-PRP, utilizando o próprio código de geração de PNPs intitulado PNPgen. Foi desenvolvido então um analisador de sintaxe que adequa o plano gerado pelo Prob-PRP à estrutura requerida pelo algoritmo de geração. Houve a tentativa de conversão da política e do plano condicional para PNPs, a rede gerada está disposta na Figura 13. Na figura é possível notar alguns problemas na rede: sobreposição de lugares e transições, sobreposição nas legendas, as ligações entre estados e transições são linhas retas e o posicionamento não busca otimizar a visualização. Estes problemas ocorrem pois a ferramenta é feita para o posicionamento manual dos elementos e a biblioteca de geração não os tratas. Como os

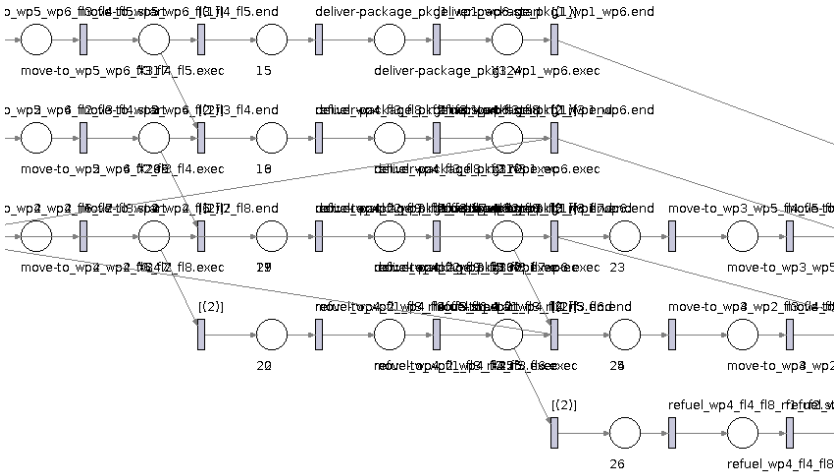



Figura 13 – Visualização da PNP gerada por meio do JARP.

exemplos apresentados pelo *software* tem poucos estados estas falhas não ocorrem, porém em problemas com maior número de estados ela fica mais evidente. Outra contrariedade dessa biblioteca que inviabilizou seu uso no trabalho foi a incapacidade de representação de ciclos pela biblioteca de geração.

Por fim, decidiu-se pelo desenvolvimento de um pacote próprio para integrar o planejamento, supervisão, visualização e execução de tarefas. Estes elementos de planejamento em conjunto com o replanejamento compõem um modelo mais realista entre planejar e agir, segundo (GHALLAB; NAU; TRAVERSO, 2004). O sistema desenvolvido deve então ser aplicado na entrega de pacotes por VANTs. O objetivo deste pacote é se assemelhar ao ROSPlan na implementação de uma estrutura genérica, desacoplando o sistema de planejamento da aplicação. Sendo assim, o pacote proposto adapta algumas implementações do ROSPlan: (1) a interface gráfica do supervisório, (2) a interface entre o servidor de ações e controlador, (3) formato de configuração e (4) chamada do planejador.

O pacote, denominado DOTPlan, apresenta também algumas diferenças: (1) forma de representação do plano, (2) visualização gráfica de progresso do plano, (3) *feedback* de efeitos do controlador, (4) servidor de ações por máquina de estado, (5) base de conhecimento implícita e (6) implementação do PRP e suas variantes. A estrutura do pacote é apresentada na Figura 14. Esta figura mostra que o sistema é composto por módulos: planejamento (AI Planning), supervisão (DOTSupervisor), execução(DotActionServer) e visualização (DotViz). Nesta seção, descrevemos a estrutura para executar planos probabilísticos por meio da representação da máquina de estados finitos.

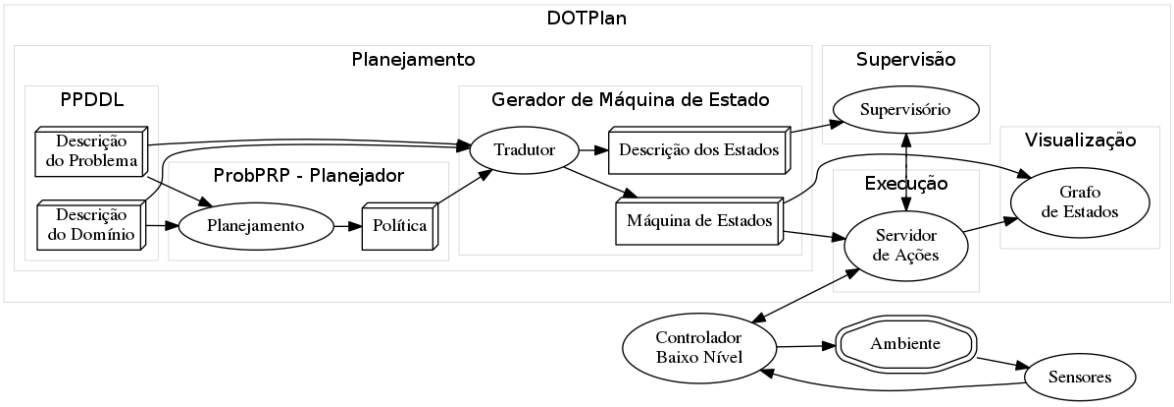



Figura 14 – Visão Geral do DotPlan

## 6.1 Formato Dot

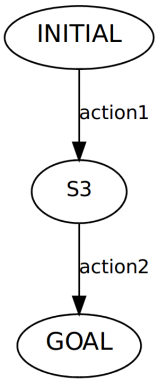
O DOT é um formato de arquivo amplamente aplicado para representação gráfica de grafos. Embora este formato oferece suporte a algumas ferramentas sofisticadas, o uso básico requer apenas um cabeçalho mínimo. A Figura 15a mostra o DOT do grafo disposto na Figura 15b. Na Figura 15a, pode-se notar que a posição dos nós pode ser deixada sem especificação. Isto acontece porque *softwares* que leem este formato, usam a biblioteca Graphviz. As atribuições de coordenadas do Graphviz permitem uma visualização gráfica limpa e gerada automaticamente, o que justifica seu uso em um grande número de aplicações. De fato, grande parte dos diagramas deste trabalho foram projetados em DOT. Nesta biblioteca, o posicionamento dos nós é resolvido como um problema de minimização do comprimento das arestas, com a progressão geral do grafo ocorrendo de cima para baixo, respeitando a orientação das arestas. O posicionamento dos nós e aresta busca ainda não obstruir as legendas das arestas, facilitando a leitura das mesmas. O DOT já está presente em pacotes essenciais do ROS como rqt_graph e rq_tf_tree, encapsulado pelo Pydot. Além disso, existe uma grande variedade de bibliotecas disponíveis em diversas linguagens de programação para criar automaticamente um grafo no formato DOT, o que aumenta sua flexibilidade, conveniência e suporte desta ferramenta.

Há alguns empecilhos na implementação convencional dos Planos de Rede de Petri no DOT, onde há presença de marcações nos lugares e a legenda destes é externa. A principal dificuldade é o Graphviz não considerar a posição da legenda dos nós no posicionamento das arestas, o que leva em alguns casos a desorganização do grafo. Outro inconveniente é que a movimentação das marcações requer uma modificação mais profunda nos *softwares* que utilizam DOT, pois essa não é uma operação comum em grafos. Na PNP os lugares só possuem marcação unitária, eles poderiam ser representados como destacados ou não destacados, porém isto poderia descaracterizar a Rede.

Os FSMs são frequentemente citados na literatura como controladores simples e compacto. Este controlador é normalmente usado para coordenação de ações sequenciais.

```
digraph  {
node [label="\N"];
1 [label=INITIAL];
3 [label=S3];
1 -> 3 [key=0,
label="action1"];
4 [label=GOAL];
3 -> 4 [key=0,
label="action2"];
}
```

(a) DOT




(b) Grafo

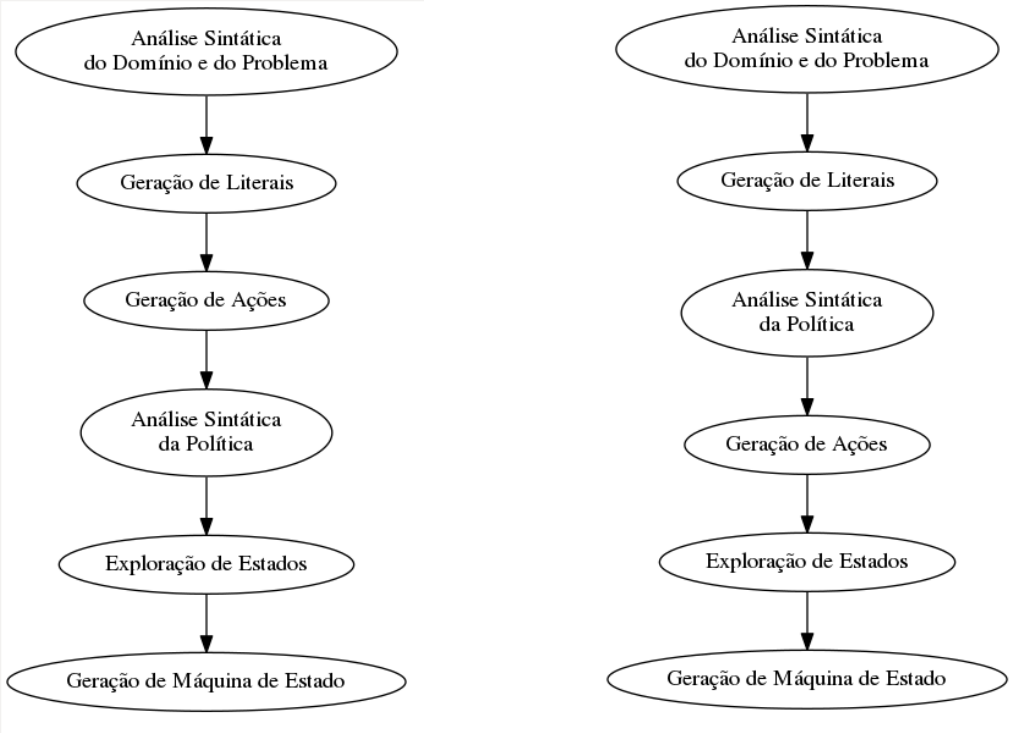
Figura 15 – DOT e seu grafo gerado pelo Graphviz

A representação por máquinas de estados finitas é suficiente para a aplicação proposta e possui computação mais simples que a PNP. Ao olhar para a Figura 15a, pode-se observar que o DOT possui um formato estruturado simples e eficiente. Com base nisso, o formulário DOT foi definido como o formato de entrada padrão, não apenas para monitoramento, mas também para execução das tarefas. Finalmente, foi definido que o formato seria o método de interface entre o planejamento e os outros módulos do pacote.

## 6.2 Módulo AI Planning

A execução do módulo de planejamento é fundamental, uma vez que fornece as informações que os outros módulos precisam para operar. O sistema de planejamento é executado *offline*, anterior a execução, e deve gerar uma máquina de estados no formato Dot da Graphviz. Na aplicação, o Prob-PRP foi usado como o planejador resultando em uma política que foi convertida usando o algoritmo 3. Informações adicionais sobre os estados também são coletadas durante a exploração dos mesmos na geração da FSM. Esses dados devem ajudar o operador durante a supervisão do sistema.

O *script* denominado *validation*, presente no *software* do PRP é capaz de realizar a tradução de problemas FOND em máquinas de estados no formato DOT. Enquanto (SARDINA; D'IPPOLITO, 2015) usa uma FSM como ferramenta de visualização, no sistema proposto esta máquina é também o controlador de tarefas. Algumas modificações foram realizadas com intuito de acelerar o código e melhorar a interface com o controlador. A primeira modificação foi a melhora na expressão dos possíveis efeitos de uma ação, que originalmente eram diferenciados apenas por um identificador inteiro. Com a adaptação, a identificação dos efeitos se tornaram os literais que o destaca dos demais efeitos da mesma ação. Ou seja, os literais que definem um efeito são obtidos por meio da diferença do conjunto com todos seus literais e a interseção dos conjuntos dos outros efeitos, de




(a) Fluxo original do algoritmo de Geração (b) Fluxo modificado do algoritmo de Geração

Figura 16 – Alteração no fluxo do algoritmo de Geração

forma:

$$Eff_a^{1*} = Eff_a^1 - (Eff_a^2 \cap Eff_a^3 \cap Eff_a^4 \cap ...) \tag{6.1}$$

Neste *framework*, adotamos a seguinte convenção para expressar uma FSM no DOT: o identificador do estado deve ser um número inteiro único, o nome do estado é arbitrário, mas não deve conter colchetes e traços, *action [efeitos dos resultados]* deve ser o formato do nome da borda, # é um caractere reservado para representar em branco ou nenhum. Assim como mostrado na Figura 18.

Outra adaptação foi modificar o fluxo do algoritmo para gerar apenas as ações tratadas pela política. A F*igura* ˚effig:flux mostra o fluxo original do algoritmo. Neste algoritmo primeiramente é realizada a análise de sintaxe do domínio com propósito de extrair os predicados, objetos, operadores, e os estados: inicial e objetivo. Logo após, os predicados são instanciados com base nos objetos para a geração de literais, produzindo todas combinações possíveis de objetos nos predicados. Posteriormente são geradas as ações baseadas também em todas as combinações de objetos, nos predicados previamente gerados e nos operadores. Em seguida, a política é extraída da saída do planejador. Por fim, a exploração de estados é realizada a partir do estado inicial utilizando as ações e a política, resultando na máquina de estados. Este processo é funcional, porém gera várias ações que não são utilizadas ou até que são impossíveis, pois não há um raciocínio

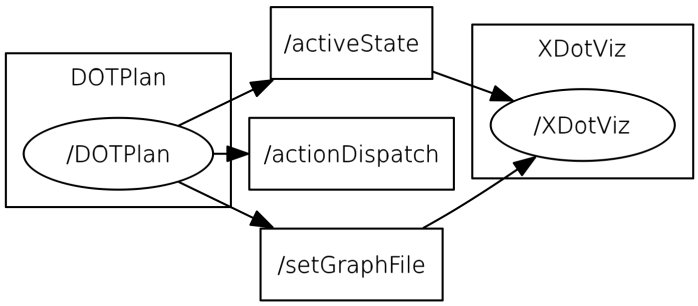



Figura 17 – Interação do DotViz com o framework proposto

geométrico envolvido. Partindo do pressuposto que apenas as ações presentes na política são tratadas no plano, é interessante que sejam analisados antes da geração das ações, evitando assim que ações desnecessárias sejam geradas. A performance era tão prejudicada da forma que foi implementada originalmente, que em alguns casos o algoritmo de geração durava mais tempo que o próprio planejamento.

A mudança do planejador neste módulo requer que o analisador de sintaxe do arquivo de saída seja adequado para a extração da política. Caso a saída do planejador preterido não seja uma política ou a entrada do planejador não seja PDDL, o desenvolvedor pode criar ainda seu próprio algoritmo utilizando de diversas bibliotecas disponíveis para geração de grafos no formato DOT, como é o caso da biblioteca empregada, $NetworkX$. Ressaltando que a exploração dos estados é interessante para a extração de informações para o supervisório.

## 6.3 Módulo DotViz

DotViz é o módulo de visualização do DOTPlan que exibe de forma gráfica o plano condicional, destaca e centraliza o estado atual da execução. O seu objetivo é facilitar a análise e compreensão do plano atual. Este módulo organiza automaticamente os nós e arestas do grafo de forma compacta e clara para o usuário. A exibição do grafo é realizada a partir de uma interface gráfica de uma versão modificada do XDot incorporada ao ROS. O DotViz é uma ferramenta suplementar do sistema de planejador, com objetivo de dar suporte de informações.

Xdot (FONSECA, 2017) é um *software* de código aberto escrito em Python que exibe arquivos DOT usando a biblioteca Graphviz em uma interface de usuário GTK3. A aplicação do Xdot se torna conveniente, já que usa o Graphviz em um nível mais baixo e possui algumas funções internas interessantes como busca de nós, destaque de nós e movimentos animados. Este *software* foi modificado para ser incorporado no ROS usando a sincronização de *threads* Gobject e denominado DotViz. Este *software* é capaz de mostrar de forma organizada todos os estados do controlador, destacar estados de

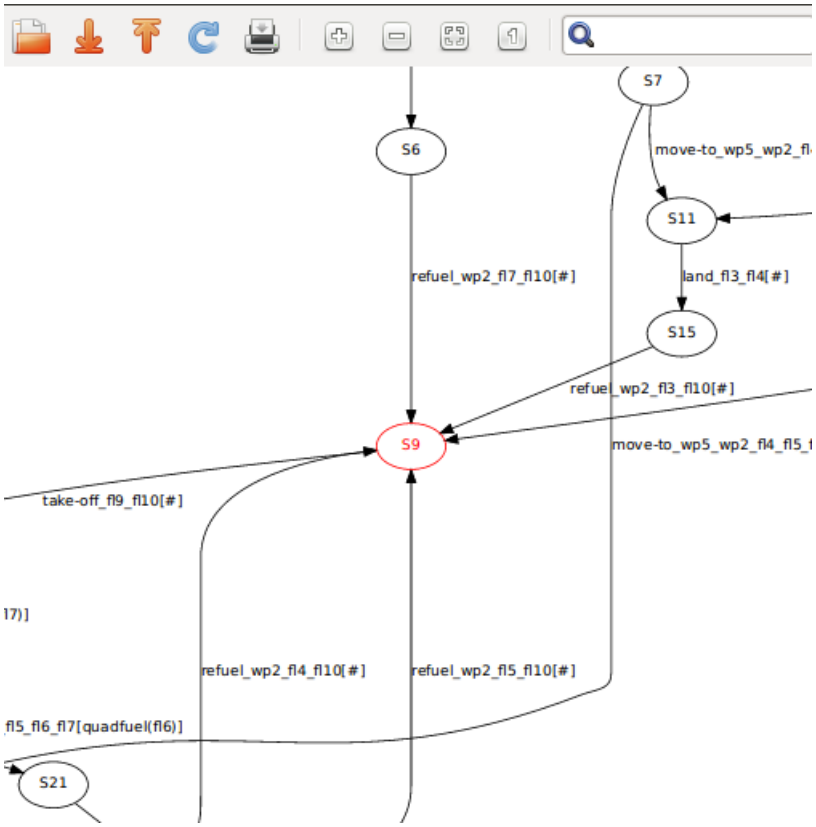



Figura 18 – Interface Gráfica do XDotViz

forma automática e realizar animações com base na troca de informações em tópicos ou serviços, sua interface gráfica está disposta na Figura 18. A Figura 17 representa a interação do DotViz dentro do *framework*. Este módulo recebe dois tipos de informação, o estado atual e o arquivo do grafo.

A última versão do Xdot não é suportada pelo ROS, uma vez que faz uso do Python 3. Consequentemente foi empregada uma versão anterior, a 0.6, compatível com o Python 2. Sendo que a diferença perceptível entre as duas é que a versão mais recente também destaca as arestas. Quando incorporada ao ROS, a interface só era atualizada mediante uma interação manual, o que foi corrigido com a sincronização de *threads*. Outra modificação foi na forma que os elementos são buscados, na configuração original os elementos que continham a sequência de caracteres eram localizados, na nova configuração a localização ocorre apenas no nó com a combinação exata de caracteres. Esta modificação impede que estados como $S1$ e $S10$ sejam destacados simultâneamente quando o sistema tenta localizar o estado $S1$.

O Xdot nativamente não salva nenhuma informação, portanto tem que o grafo deve ser organizado sempre que o arquivo é aberto. Nesse aspecto, uma nova função foi implementada para salvar e carregar posições de nós em um arquivo intermediário para acelerar a inicialização do programa de arquivos pré-abertos. Esse recurso é útil uma vez que planos extensos ou complexos podem resultar em gráficos grandes ou embaralhados,

pode levar algum tempo para organizar automaticamente os nós.

## 6.4 DotActionServer

DotActionServer é o módulo central para execução do plano computado pelo módulo de planejamento. Seu papel é designar ações de alto nível para o controlador com propósito de orientar o sistema até seu estado objetivo. Este nó concentra as informações de planejamento e execução. Estas informações são utilizadas para coordenar o controlador de baixo nível e auxiliar os outros módulos. Os tipos de informações são:

- Informações das ações: nome, possíveis resultados, e parâmetros.
- Informações do estado: nome, identificador;
- Máquina de estado: mapeamento de estado e ação, de resultados de ação e estado além de estado e identificador;
- Parâmetros de Planejamento: localização dos arquivos de planejamento no sistema, comandos de chamada do planejador, comandos de chamada da geração da máquina de estado;
- Informações de Execução: *status* da execução, localização do arquivo DOT, registro de erros de execução.

Estes dados são parte transmitidas como parâmetros na chamada do nó, parte adquiridas pela máquina de estado no formato DOT e parte decorrentes da operação. Este módulo é capaz de chamar planejadores externos ao ROS com os parâmetros dos arquivos de domínio e problema. Posteriormente, a saída do planejador é adequada ao formato de execução por meio de um gerador de máquinas de estados. Em seguida, o DotActionServer realiza a análise sintática do arquivo DOT e extrai as informações da máquina, dos estados e das ações. Todo este processo é ativado mediante a requisição por meio de serviços.

A execução deste controlador é muito simples, com função de rastrear o estado atual da máquina e despachar a ação especificada em suas transições. O servidor de ações inicialmente despacha a ação correspondente ao estado inicial, e o despacho das ações consecutivas ocorre com base na retroalimentação do controlador de baixo nível. O algoritmo 4 mostra como ocorre este procedimento de despacho de ações quando acontece um $feedback$. O controlador de baixo nível deve executar a ação e retornar o estado anterior a ação, qual ação foi executada, a consequência da ação dentre a lista de consequências e se houve êxito na execução da tarefa. Esta retroalimentação tem como finalidade verificar

a sincronia do controlador de alto nível e de baixo nível, verificar o *status* de execução e guiar a progressão do plano condicional.

**Algoritmo 4** Método de Despacho de Ações

**Entrada:** $estadoAtual, resultadoAcao, acaoAtual, sucesso$
  **se** $sucesso = falso$ **então**
    $planStat \leftarrow falha$
    ***retorne***
  **se** verificaFeedback($estadoAtual, resultadoAcao, acaoAtual$)=falso **então**
    $planStat \leftarrow falha$
    ***retorne***
  $estadoAtual \leftarrow$ Progride($resultadoAcao$)
  **se** estadoObjetivo($estadoAtual$) = verdadeiro **então**
    $planStat \leftarrow concluído$
    ***retorne***
  $acaoAtual \leftarrow$ consultaAção($estadoAtual$)
  $resultadosAcao \leftarrow$ consultaConsequencias($estadoAtualState$)
  despachaAcao($estadoAtual, acaoAtual, resultadosAcao$)
  publicaEstadoAtual($estadoAtual$)
  $planStat \leftarrow despachando$

## 6.5 DotSupervisor

Este sistema inclui ainda uma interface gráfica interativa, usada para apresentar informações adicionais sobre os estados e possibilitar a interação com o DotActionServer de forma amigável. Algumas das informações exibidas são coletadas durante a geração do FSM. Esses dados incluem informações do estado atual, o status do servidor de ações e a ação despachada. Essa interface também permite que o usuário: planeje a partir de arquivos PPDDL, recarregue o arquivo DOT do servidor de ações e envie *feedback* de ação.

A interface gráfica do usuário (GUI) deste módulo é baseada na do ROSPlan, como mostra a Figura 19. Ambos os programas foram desenvolvidos em Python utilizando o plugin do Qt no ROS para construção da interface gráfica. A estrutura de exibição de informações dos elementos de planejamento como fatos, objetivos e objetos é equivalente. Estas informações são no entanto obtidas de formas diferentes, enquanto o ROSPlan consulta sua base de conhecimento, o DOTPlan examina as informações extraídas durante a geração da máquina de estado. A interface proposta conta ainda com a exibição dos fatos relevantes, que é o conjunto de literais utilizados para determinação da política. A importância de explicitar estes literais vem do fato que eles exprimem melhor o estado da execução, uma vez que os estados completos possuem uma porção de literais para descrever como o mundo funciona. No entanto, os estados completos também são importantes para

| | ROSPlan (Popf) | ROSPlan (Cont) | DOTPlan (Prob-PRP) |
|---|---|---|---|
| Supporte PDDL | X | X | X |
| Supporte PPDDL | | | X |
| Mecanismos de Recuperação | X | X | X |
| Recuperação Offline | | X | X |
| Representação Visual | X | X | X |
| Supervisão do Plano | X | | X |
| Base de Conhecimento | X | | |
| Sensoriamento Repetitivo | X | | X |
| Observabilidade Parcial | | X | X' |

Tabela 1 – Comparação entre o ROSPlan usando o POPF, o ROSPlan usando o Contingent-FF com o PNP e o DOTPlan usando o Prob-PRP

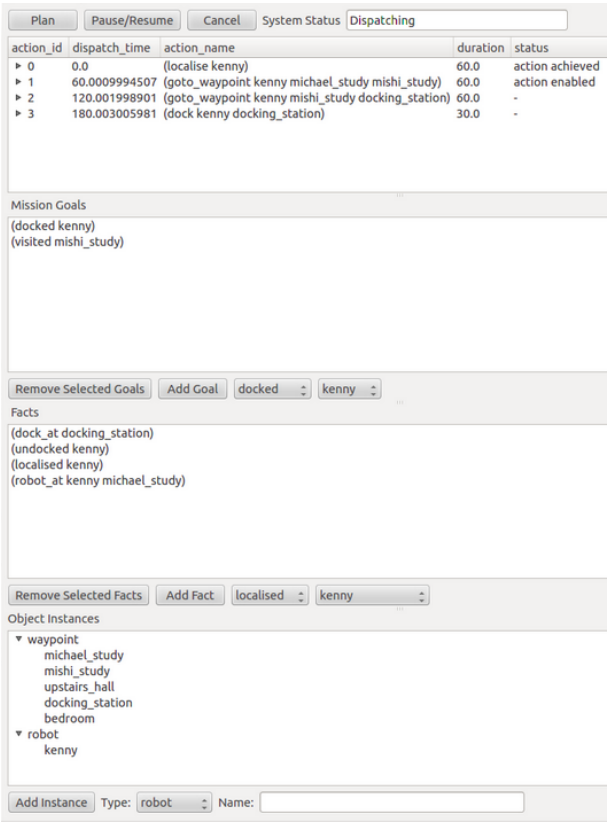
verificar a modelagem do problema de planejamento e assim também foram incorporados na interface.

A forma de representação para acompanhamento da execução do plano utilizada pelo ROSPlan, como visto na Figura 19a, é uma lista organizada por meio de tabelas. Esta representação é adequada a planos determinísticos pois não possuem ramificações ou ciclos, o que poderia dificultar o entendimento da sequência de execução. O DOTPlan utiliza por sua vez a representação por grafos, os quais podem ser visualizados no DOTViz.

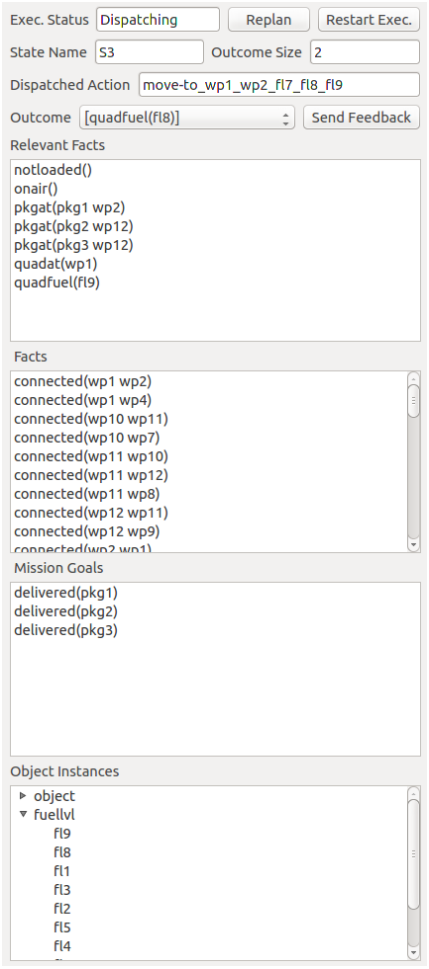
O ROSPlan possibilita através de sua interface, a manipulação dos elementos de planejamento exibidos na interface. Este mecanismo funciona como uma forma interativa de descrição do problema que posteriormente é traduzida em PDDL e transmitida para o planejador. No DOTPlan, as modificações devem ser realizadas diretamente no arquivo PDDL do problema em tempo de execução, o que não é possível no outro programa que apenas faz uso do PDDL indiretamente. Por fim, a interface proposta é capaz de realizar a interação direta com o plano, com intuito de simulá-lo para entendê-lo ou testá-lo de maneira anterior a execução. Nesta interface o usuário é capaz de visualizar detalhes do despacho de ações e determinar sua consequência.

## 6.6 Comparação com Outros Sistemas de Planejamento

O *framework* proposto foi comparado ao principal pacote ROS para planejamento e execução, o ROSPlan, em suas duas principais variantes usando POPF e Contingent-FF. Essa comparação é apresentada na tabela 1. Todos eles usam planejadores independentes de domínio que aceitam a estrutura do PDDL e apresentam algum tipo de comportamento




(a) Interface Gráfica do ROSPlan




(b) Interface Gráfica do DOTPlan

Figura 19 – Comparação entre a interface gráfica do ROSPlan e do DOTPlan

de recuperação para lidar com a incerteza. Enquanto o ROSPlan-POPF usa replanejamento *online* quando encontra incerteza, o ROSPlan-Cont.FF e o DOTPlan confiam em seguir um plano condicional. Tanto o replanejamento quanto o plano condicional precisam de uma estimativa do estado atual para guiar seu plano. Mas enquanto os planejadores não-determinísticos fazem o raciocínio *offline* para diferentes cenários prováveis para evitar becos sem saída, o replanejamento não, o que pode levar o agente a um estado sem saída ou interação repetida na presença de incerteza (LITTLE; THIEBAUX et al., 2007).

O ROSPlan-POPF representa seu plano em formato de lista, o que é suficiente para representar um plano clássico, já que existe apenas uma seqüência possível de ações. É útil acompanhar o progresso do agente. Um problema de visualizar planos clássicos é que seu comportamento pode ser imprevisível se o replanejamento for acionado. O ROSPlan-Cont.FF usa os Planos por Rede de Petri, do inglês *Petri Net Plans* (PNP), através da interface de uma ferramenta PNP fora do ROS chamada PNPJarp. O sistema utiliza os Planos por Rede de Petri que é a aplicação de redes de Petri para representar e executar planos. Sua teoria também suporta ações simultâneas e sistemas multiagentes, o que não

se encontra implementado ainda no $software$. A biblioteca de geração de PNP não parece oferecer suporte a ciclos em sua política de tradução PNP.

O ROSPlan-Cont.FF tem uma percepção limitada de eventos exógenos no ambiente após a observação. Uma vez que sistema implementado não suporta ciclos e assim a detecção repetida do mesmo fato. O que torna o sistema de planejamento inadequado para esse tipo de problema. De fato, é capaz de resolver problemas de observabilidade parciais que Prob-PRP e POPF não são. O DOTPlan é capaz de utilizar o PO-PRP modificando apenas o caminho do planejador nos parâmetros do DOTActionServer, uma vez que todos os variantes do PRP possuem mesmo formato de entrada e saída. Esta variante é capaz de computar problemas com observabilidade incompleta, mostrando inclusive resultados superiores ao Contingent-FF.

O ROSPlan-POPF mantém o controle de todo o conhecimento usando sua base de conhecimento. Neste pacote, as ações de detecção podem ocorrer com frequência infinita e influenciam o modo como o sistema estima seu estado e atualiza seus fatos conhecidos. Uma vez que os fatos do sistema mudam, o replanejamento deve ser acionado. O DOTPlan deve observar os efeitos de suas ações após cada ação não determinística para escolher qual ramificação de seu plano será despachado.

# 7 Experimentos

Esta seção consiste na análise da aplicação do planejador Prob-PRP por meio do sistema de planejamento proposto DOTPlan em domínios de entrega com VANTs. Primeiramente serão realizadas experiências em domínios mais didáticos, buscando extrair características dos domínios com VANTs e da ferramenta de planejamento. Posteriormente, o sistema é aplicado no simulador de aplicações Gazebo, onde também serão realizadas considerações. O trabalho foi realizado em um computador portátil com processador i7-4500U, placa de vídeo Geforce 745M e 8 GB de memória RAM.

## 7.1 Experimento 1 - Considerações Iniciais

Neste experimento o objetivo do planejamento é realizar entregas em ambiente cujas rotas são mapeadas conforme a Figura 20. O VANT é capaz de recarregar sua bateria na estação localizada no *waypoint* $wp4$. A ação de se movimentar entre estes pontos de referência pode custar um ou dois níveis de combustível, com probabilidade de 70% e 30% respectivamente, o pouso e decolagem custam um nível de combustível e as demais ações não gastam nada.

E uma configuração do problema o VANT possui 12 níveis de combustível, inicia seu percurso em $wp1$ em terra totalmente abastecido. Seu objetivo é entregar um pacote, localizado em $wp1$, no local $wp6$ e pousar em $wp5$. O plano é calculado pelo Prob-PRP em 0.02s e o resultado é um plano factível com as características de um plano cíclico forte.

A primeira característica explicita é que o número de ações e consumo de combustível não é considerado durante o planejamento, como mostra Figura 21. Em uma análise no algoritmo de planejamento, sabe-se que o Prob-PRP busca maximizar as chances de um plano clássico, com as ações determinilizadas, ser executado. Sendo assim, uma sequência de ações determinísticas possui a mesma chance de seguir o plano que apenas uma ação, que é 100%. Dado este fato, infere-se que o planejador faz uso indiscriminado de ações determinísticas para simplificar e acelerar o processo de planejamento. Ainda na Figura

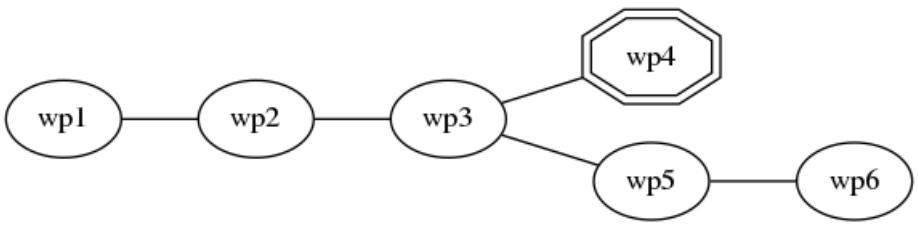



Figura 20 – Mapa de rotas do experimento I

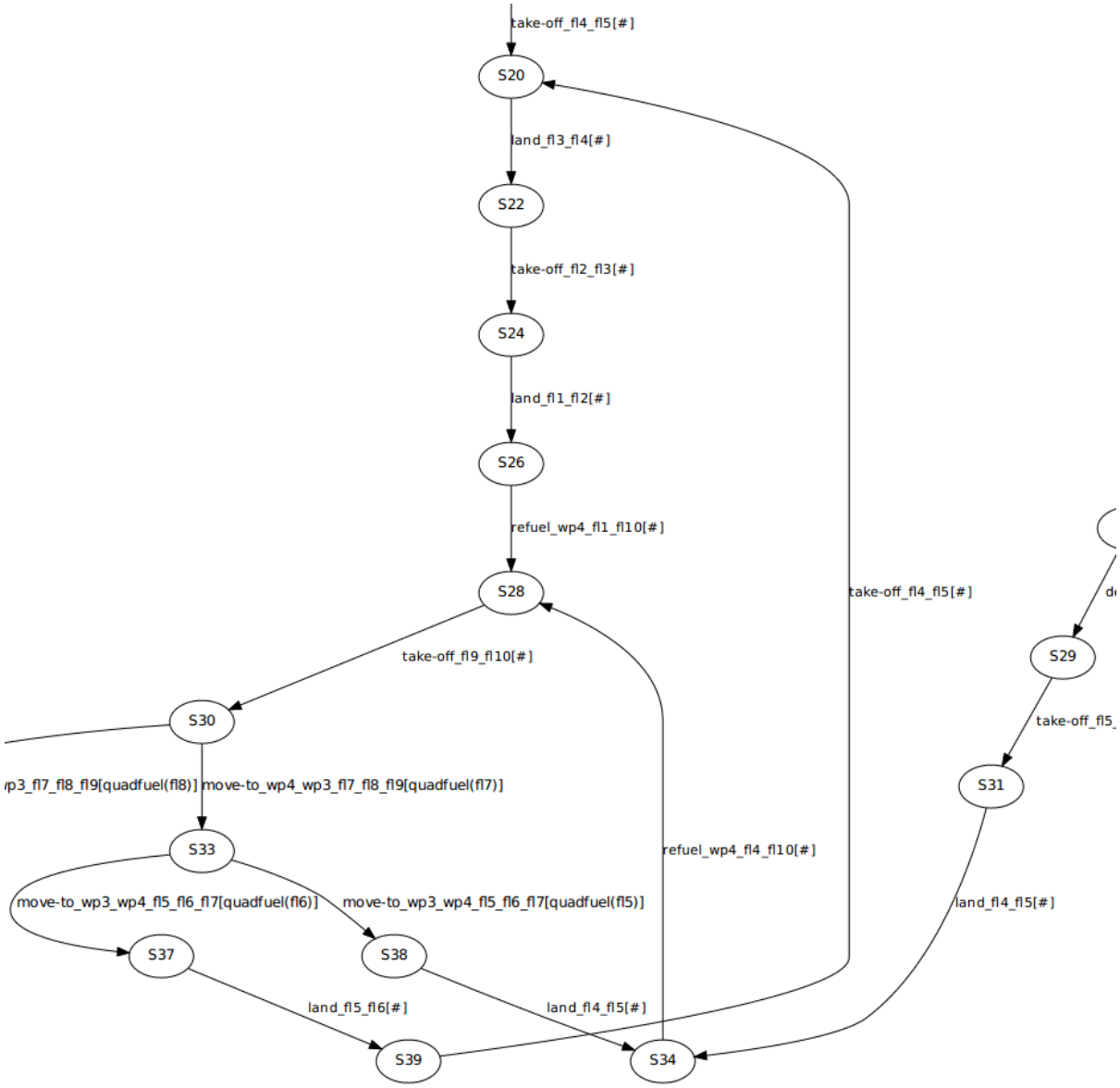



Figura 21 – Despedício de combustível para comprimir o plano

21, o planejador define a execução das ações determinísticas de decolar e pousar para modificar o nível de combustível e assim convergir os ramos. Este comportamento é indesejável no domínio analisado, pois estas ações são custosas e desnecessárias. Uma solução é tornar as ações determinísticas custosas, com os diferentes efeitos são irrelevantes para o plano. O problema desta solução é o aumento no tempo de planejamento e a poluição do plano gerado. A solução utilizada foi impedir a execução seguida da decolagem e pouso por meio de modificações em seu predicado. Esta solução é específica para o domínio e ajuda o planejamento reduzindo as opções de planos.

Outra característica é a imposição de ciclos fortes no plano, mesmo que exista uma solução forte e estes não sejam necessários. No domínio em questão implica no reabastecimento e tomada de ações desnecessárias para retornar a um estado garantir os

| Recargas Permitidas | Tempo de Planejamento (s) | Estados do Controlador | Bloqueios na Máquina | Chance de Atingir Objetivo (%) | Número Médio de Ações |
|---|---|---|---|---|---|
| 0 | 0.24 | 28 | 3 | 74.2 | 11 |
| 1 | 1.92 | 100 | 3 | 96.1 | 18.08 |
| 2 | 5.56 | 168 | 3 | 99.4 | 20.47 |
| 3 | 9.92 | 123 | 4 | 99.7 | 18.56 |
| 4 | 16.42 | 289 | 7 | 98.9 | 20.5 |
| 5 | 27 | 348 | 3 | 99.9 | 20.7 |
| $\infty^1$ | 0.2 | 99 | 0 | 100 | 30.9 |
| $\infty^2$ | 0.16 | 70 | 0 | 100 | 19.92 |

Tabela 2 – Características das políticas pelo número de recargas permitidos no Prob-PRP. Onde $\infty^1$ e $\infty^2$ representam recargas ilimitadas com e sem planejamento local respectivamente.

efeitos do ramo principal. Em outras palavras, a execução leva de volta a um estado visitado anteriormente para tentar obter o resultado desejado. Este ciclo tende a aumentar o comprimento do plano e pode não ser necessário para evitar becos sem saída. Uma das soluções foi limitar o número de reabastecimentos para forçar o planejador a não reabastecer de forma desregrada.

A tabela 2 foi elaborada realizando simulações de Monte Carlo no plano proposto a partir de 4000 execuções. Em cada execução, as ações eram despachadas e o plano progredia conforme o *feedback* dos efeitos observados até que o sistema atinja o estado objetivo. Os *feedbacks* eram determinados pelo simulador conforme um sorteio entre os efeitos da ação despachada com suas respectivas probabilidades ocorrência. Ao atingir o estado objetivo ou algum bloqueio, eram registradas o número de ações executadas até atingir aquele estado e se o objetivo foi cumprido.

Inicialmente foram feitas experiências limitando o número de recargas permitidas pelos VANTs, linhas de 0 a $\infty^1$. Ele introduz no problema um bloqueio inevitável, quando o VANT não possui mais recargas e deve se arriscar. O planejador busca tratar os bloqueios no plano por meio do racionamento do recurso de recargas, que indiretamente impacta o consumo de combustível. Na tabela é possível observar comparando os tamanho dos planos que quando impostos limites de recargas eles tendem a atingir seus objetivos com um número menor de ações. Isto indica que o plano cíclico com reabastecimentos infinitos não está sendo eficiente, porém ao comparar o número de estados do controlador nota-se que o plano é compacto.

Quanto ao tempo de planejamento, o planejamento cíclico evita que o plano condicional tenha muitas ramificações, diminuindo assim o número de planejamentos clássicos requeridos e acelerando o processo. Para a configuração analisada do problema não existe

uma solução forte apenas uma solução cíclica forte, neste caso o número de recargas limita o número de tentativas do agente. Outra característica que não está explícita na tabela é que quando o sistema está em um estado com um grande número de recargas restantes o planejamento não economiza ações, quanto mais próximo de acabar, mais ele fica restritivo. Quanto a escalabilidade desta abordagem, o tempo varia quase que quadraticamente com o número de recargas. Um estratégia para a utilização deste plano poderia ser utilizada até que o sistema atinja os estados com apenas uma recarga e posteriormente replanejar com base no estado atual.

Um problema desta abordagem de limitar o reabastecimento é que o número de recargas varia de acordo com a configuração do problema e assim também um número ideal de recargas. Sendo assim, o planejamento necessitaria um certo número de ajustes. Em busca de uma nova alternativa, foi descoberto que o planejamento local descrito na seção 3.5.1 era responsável por este tipo de comportamento de impor ciclos. Este mecanismo tem preferência no planejamento e caso falhe o planejamento ocorria de forma normal. No planejador é possível desativar este mecanismo por meio de uma combinação de parâmetros passados na chamada do PRP. Ainda na tabela 2, o planejamento local teve uma performance superior em todos os quesitos em relação às demais configurações. No experimento em questão, o planejamento global foi capaz de gerar soluções mais compactas e em menor tempo, o que não é esperado. No planejamentos global, os efeitos alternativos ao efeito selecionado pelo planejamento clássico ficam incondicionados, e prosseguem o planejamento normalmente buscando se encontrar com o plano principal de forma mais conveniente. Deste modo, os efeitos alternativos podem gerar mais ramos que demandam um maior número de planejamentos provocando um aumento no tempo de planejamento e no tamanho da máquina de estados. Porém no geral, desativar o planejamento local propicia a geração de planos condicionais melhores, evitando ciclos desnecessários e o impacto no tempo de planejamento é justificável e não passa da ordem de grandeza.

Para o mesmo domínio foram desenvolvidas outras instâncias do problema para testar o comportamento geral do plano com a variação da capacidade de armazenar combustível, são elas:

- Níveis $\geq 14$: O VANT se desloca diretamente para o local da entrega;
- $12 \leq$ Níveis $< 14$: O VANT se dirige a estação, realiza a recarga e cumpre seu objetivo;
- $11 \leq$ Níveis $< 12$: O VANT recarrega, e volta a estação caso não seja capaz de cumprir seus objetivos no pior caso;
- $9 \leq$ Níveis $< 11$: O VANT tenta maximizar suas chances através de recargas, se houver riscos no trajeto até a estação, o veículo opta por não se recarregar se arriscando

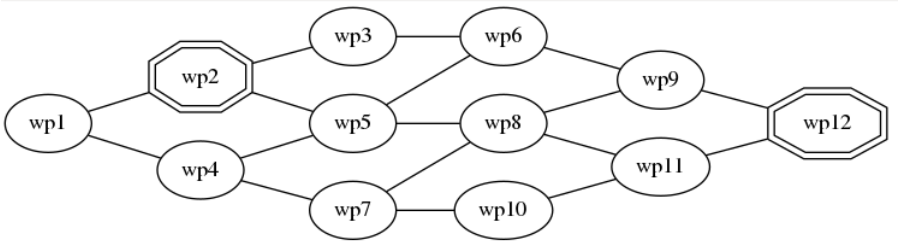



Figura 22 – Mapa de Rotas do Experimento 2

até o objetivo;

- Níveis $\leq 8$: O VANT não é capaz de operar;

O comportamento apresentado no plano, mostrou a capacidade do planejador de lidar tanto com situações favoráveis quanto desfavoráveis. Algumas considerações são relevantes na análise: (1) ele foi capaz de gerar planos fracos, forte e cíclicos fortes conforme a necessidade; (2) Se o planejamento fosse gerado exclusivamente por um planejamento clássico pessimista não seria permitido que as tarefas serem cumpridas caso o VANT tivesse capacidade inferior a 12 níveis, enquanto um plano otimista não consideraria ir até uma estação para recarregar e poderia sofrer com a falta de combustível; (3) O raciocínio sobre os bloqueios permite com que o planejador delibere sobre quando é melhor recarregar e quando é melhor seguir o plano analisando as diferentes consequências. O planejador não tem informações sobre qual consequência é desejável ou não, tentando apenas encontrar um plano que seja seguro; (4) A operação sobre risco, como no caso analisado de se operar com capacidade inferior a 11 níveis, é indesejada no domínio analisado porém pode ser útil em outras aplicações com VANTs em que os benefícios da operação compensam os riscos necessários.

## 7.2 Experimento 2 - Análise do Plano

O experimento 2 é uma instância mais complexa do domínio base apresentado. O mapa de rotas deste experimento está expresso na Figura 22. Nele o VANT possui 10 níveis de combustível, inicia sua execução em $wp1$ e tem como objetivo entregar os pacotes localizados: (1) em $wp2$ no $wp7$,(2) em $wp12$ no $wp11$ e em (3) $wp12$ no $wp2$. As estações estão localizadas nos locais $wp2$ e $wp12$.

O resultado foi gerado em 0.66s com 100% de chance de atingir o objetivo. O plano condicional gerado possui 99 estados e está presente na Figura 23. O objetivo da imagem é esboçar o plano gerado em que há a presença de um plano central, destacado na figura. Este plano central tem grande probabilidade de ser executado e suas ramificações tendem a segui-lo, sem terminarem em bloqueios. Este ramo principal tende a ser o caminho mais

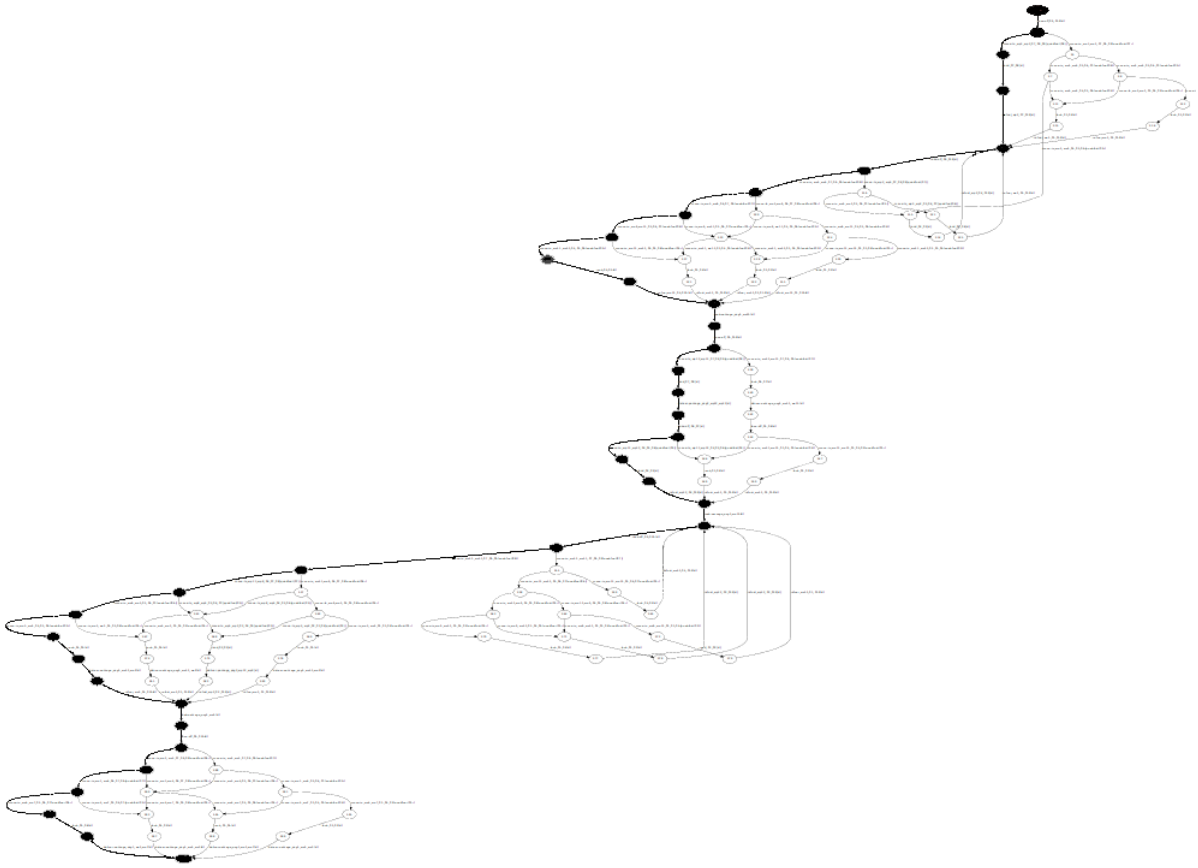


Figura 23 – Plano Condicional do Experimento 2

curto, uma vez que contém os efeitos desejados pelo planejador e os outros efeitos são tratados posteriormente.

A deliberação sobre as probabilidades de ocorrência dos efeitos no Prob-PRP para a escolha de ações na geração do plano faz com que os efeitos mais prováveis tenham prioridade do tratamento. Desta forma, elas têm maior possibilidade de ser encontradas nos efeitos selecionados pelo planejador clássico. Como o ramo principal normalmente é o plano que requer um menor número de ações, incentivar a presença de efeitos com maior probabilidade de ocorrência, faz com o tamanho médio do plano diminua.

A sequência de execução do plano gerado pode ser representado por meio de etapas, da seguinte forma:

1. Se dirigir a estação em $wp2$ e recarregar;
2. Se dirigir a estação em $wp12$, caso gaste mais combustível de $wp2$ a $wp12$ retorna a $wp2$ e recarrega;
3. Recarrega, pega o pacote 2, o entrega em $wp11$, retorna a estação em $wp12$;
4. Recarrega, pega o pacote 3, o entrega em $wp2$, caso gaste mais combustível de $wp12$ a $wp11$ retorna a $wp12$ e recarrega;

5. Recarrega, pega o pacote 1, o entrega em $wp7$;

No plano gerado, o VANT passa pelo $wp2$, reabastece e não tenta entregar o pacote localizado neste ponto de referência, mesmo passando próximo do destinatário $wp7$. Apesar da possibilidade da entrega ser concretizada em uma certa combinação de efeitos e que este plano seja mais direto, o planejador delibera por não fazê-lo uma vez que o tratamento dos efeitos alternativos resulta em becos sem saída. Ou seja, o planejador possui mecanismo de adaptação no planejamento clássico conforme são encontrados bloqueios.

## 7.3 Experimento 3 - Impacto do Peso no Plano

Nestes experimentos o objetivo é testar a expansão do domínio de entregas onde os pacotes tem peso e a aeronave pode carregar mais de um pacote, porém possui uma capacidade máxima de carga. Os pesos são divididos em 6 níveis e impactam no consumo durante o movimento entre *waypoints*, conforme mostra a Tabela 3

| Consumo | Probabilidade de Consumo | | | | | |
|---|---|---|---|---|---|---|
| | w0 | w1 | w2 | w3 | w4 | w5 |
| 1 nível | 0.6 | 0.55 | 0.5 | 0.45 | 0.4 | 0.35 |
| 2 níveis | 0.4 | 0.45 | 0.5 | 0.55 | 0.6 | 0.7 |

Tabela 3 – Probabilidade de consumo de acordo com o nível de peso do VANT.

O domínio testado inicialmente é um expansão da experiência anterior, onde os pacotes pesam 1, 2 e 3 respectivamente. O plano foi gerado em 12.2s, com as seguintes etapas:

1. Se dirigir a estação em $wp2$ e recarregar;
2. Se dirigir a estação em $wp12$, caso gaste mais combustível de $wp2$ a $wp5$ retorna a $wp2$ e recarrega;
3. Recarrega, pega o pacote 2 e 3, o entrega em $wp11$, retorna a estação;
4. Recarrega, se dirige ao $wp2$ para realizar mais combustível, caso gaste mais combustível de $wp12$ a $wp11$ retorna a $wp12$ e recarrega;
5. Recarrega, pega o pacote 1, o entrega em $wp7$;

A diferença entre o plano da experiência 2 e da experiência 3 é que o planejador decide carregar o pacote 3 durante a entrega do pacote 2. No problema proposto não é interessante que a aeronave carregue mais de um pacote mesmo que ela tenha capacidade,

pois isto impacta no gasto de combustível e não gera vantagens na logística. Analisando o funcionamento do planejador é possível inferir as causa deste comportamento. Ao deliberar sobre pegar ou não o pacote 1, o sistema calcula que o plano mais conveniente se baseia em gastar menos combustível e que pegar o pacote 1 diminui a probabilidade de segui-lo. Ao raciocinar sobre o pacote 2, o planejador delibera que o plano de gastar mais combustível é curto, seguro e o mais provável, sendo assim ele carrega o pacote 3 para aumentar mais ainda a probabilidade de se gastar mais combustível e seguir o plano pessimista. O tamanho total do plano não varia mesmo carregando o pacote 3 na entrega do pacote 2, pois esta ação teria que ocorrer cedo ou tarde. Isto acontece porque o planejador não enxerga o consumo de combustível como um custo, apenas como um recurso que ele não pode deixar acabar por completo.

Este comportamento pode ser ocasionado por causa do nível de discretização utilizada que mantém o consumo sempre em no máximo dois níveis. Assim sendo, optou-se por investigar um problema onde o combustível é discretizado em mais níveis. Nesta nova configuração o nível de combustível é discretizado em 30 níveis e os níveis de consumo combustível gastos são adequados de acordo, cujos valores estão dispostos na tabela 4. Estes valores foram extraídos de distribuições normais com variância de 0.6 e médias 3,1, 3,48, 3,86, 4,24, 4.62 e 5, conforme a Figura 24.

| Consumo | Probabilidade de Consumo | | | | | |
|---|---|---|---|---|---|---|
| | w0 | w1 | w2 | w3 | w4 | w5 |
| 1 nível | - | - | - | - | - | - |
| 2 níveis | 0.12 | 0.05 | - | - | - | - |
| 3 níveis | 0.66 | 0.49 | 0.24 | 0.08 | - | - |
| 4 níveis | 0.22 | 0.46 | 0.65 | 0.62 | 0.39 | 0.17 |
| 5 níveis | - | - | 0.11 | 0.3 | 0.56 | 0.66 |
| 6 níveis | - | - | - | - | 0.05 | 0.17 |

Tabela 4 – Probabilidade de consumo de acordo com o nível de peso do VANT.

Aumentar o nível de discretização ajuda a aproveitar melhor oportunidades e assim tirar mais vantagem dos níveis. Deste modo, o planejamento desconsidera algumas situações impossíveis e vê a redução do peso como uma possibilidade de encurtar o plano. Porém a mesma situação descrita anteriormente ocorre, pois caso o plano em que o VANT se sobrecarrega seja seguro e tenha maior probabilidade de ocorrência, o planejador deliberará por realizá-lo. Na instância analisada o nível w2 tem como maior probabilidade de ocorrência de gastar 3 níveis e o nível w5 de 5 níveis, como $max(p(w2)) < max(p(w5))$ então o pacote 3 é carregado durante a entrega do pacote 2 mesmo gastando mais combustível. Assim o aumento do número de níveis é importante para representar melhor o ambiente e as situações, porém não evitaria este comportamento.

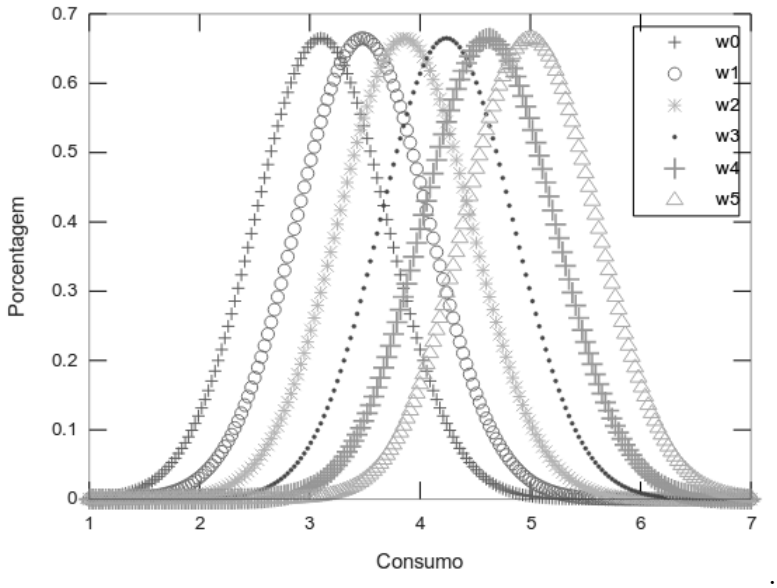



Figura 24 – Consumo do VANT de acordo com o peso

## 7.4 Experimento 4 - Falhas Inevitáveis

As falhas estudadas nesta seção são as que não podem ser tratadas pelo planejamento. A falha analisada é o mal funcionamento do VANT durante a movimentação entre *waypoints*. Inicialmente a falha foi adicionada como uma possível consequência da ação de se mover, quando ela ocorria, o estado de um literal que bloqueava as outras ações era alterado. A probabilidade de falha ocorria conforme a tabela 5 e o cenário foi o mesmo discutido anteriormente.

| Efeitos | Probabilidade dos Efeitos | | | | | |
|---|---|---|---|---|---|---|
| | w0 | w1 | w2 | w3 | w4 | w5 |
| 1 nível | 0.6 | 0.55 | 0.53 | 0.5 | 0.47 | 0.34 |
| 2 níveis | 0.399 | 0.4489 | 0.4687 | 0.4984 | 0.528 | 0.5475 |
| Falha | 0.001 | 0.0011 | 0.0013 | 0.0016 | 0.002 | 0.0025 |

Tabela 5 – Probabilidade de ocorrência dos efeitos de acordo com o nível de peso do VANT.

O resultado do planejamento foi produzido em 31.68s com 13.3% de probabilidade de chegar ao objetivo, sendo bem abaixo do esperado. Se as probabilidades de falha fossem desprezadas na modelagem do domínio, o VANT teria pelo menos mais de 99% de chance de chegar ao objetivo. A causa da deficiência no plano é que o planejador identifica a ação e o estado que causou as situações de bloqueios e os coloca na lista de par de ação e estados proibidos, como discutido na seção 3.5.1. O plano então evita estes estados, os quais são essenciais para a chegada no objetivo, gerando um plano com baixa probabilidade. O resultado é ainda pior para 8% se desativada a iteração final que desconsidera os estados proibidos. Ou seja, os estados se tornam restrições tão fortes neste planejador que ele

os exclui da topologia. As probabilidades de falha dispostas na tabela são hipotéticas, porém a situação é comum quando ações indispensáveis possuem uma falha inevitável, independente de sua probabilidade. Esta é uma limitação do planejador Prob-PRP e não do sistema de planejamento.

Um artifício para evitar que os estados e ações que devem ser executados se encontrem na lista de estados proibidos é colocar uma ação intermediária entre os bloqueios mortais e a ação de mover em que o VANT tenha chance de se recuperar. Mesmo assim o resultado não é satisfatório com 44.1% de chance de atingir o objetivo, calculado em 362.2s. Em situações como a apresentada onde os $dead - ends$ são inevitáveis e eminentes, os planos apresentaram uma quantidade reduzida de ciclos mesmo quando necessários. Sem os ciclos o VANT teria que se arriscar e não seria bem sucedido em algumas situações. Por fim, a ferramenta não se mostrou capaz de lidar com este tipo de situação.

## 7.5 Experimento 5 - Simulação Gazebo

A simulação no Gazebo mostra como o sistema poderia atuar em uma aplicação mais prática. Este software possibilita uma simulação em um ambiente 3D, com um motor de física. Este pacote também é responsável por simular o modelo de objetos, atuadores e sensores com suas incertezas e assim informar o sistema robótico a respeito do meio. Como o foco do trabalho é o planejamento, alguns componentes da simulação foram simplificados. Um dos elementos simplificados foi o controle do quadricóptero utilizado em alto nível, sendo que a operação em baixo nível seria interessante uma vez que permite calcular o impacto da operação no gasto de energia e simular a presença de ventos.

Para a representação da incerteza na operação é realizado um sorteio com base nas probabilidades descritas no domínio. Para a manipulação de objetos é realizada a comunicação direta com o Gazebo. A navegação é realizada utilizando a biblioteca Navigation, com a localização realizada através de sensores laser e os *waypoints* previamente mapeados em posições do mapa. No sistema utilizado o servidor de ações atribui sequencialmente as ações ao quadricóptero com base em seu $feedback$, como mostra a Figura 25. Um grafo de computação mais completo está disto no apêndice D, o módulo de supervisão não aparece no ROS pois é um plugin e o módulo de planejamento por ser externo ao ROS.

Esta simulação é baseada da interface_de_quadricóptero [1] com o ROSPlan, sendo que nativamente o quadricóptero é capaz de realizar apenas as ações de decolar e pousar em outro mapa não mapeado. O ambiente testado inicialmente foi o play_pen, o qual já é mapeado, disponível no repositório do Gazebo, uma vez que é um ambiente menor e assim a execução do plano é mais rápida. Posteriormente o sistema foi simulado no

[1] https://github.com/fairf4x/ROSPlan_interface_quadrotor

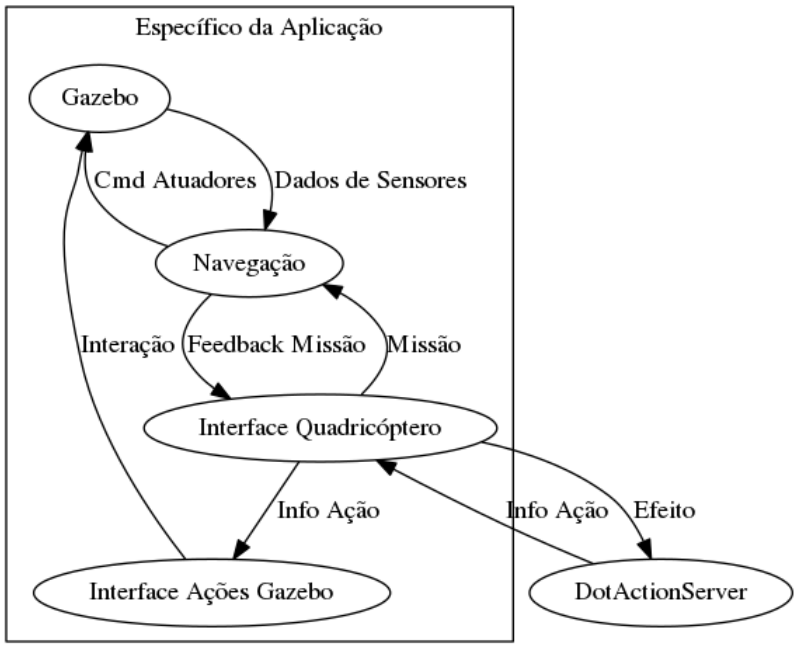



Figura 25 – Diagrama de funcionamento do sistema aplicado

domínio de entregas proposto, como mostra a Figura 26. Neste mapa são adicionadas latas de refrigerante e uma porta para exemplificar um plano, disposto no Apêndice C. Nesta simulação o papel do VANT é transportar latas de refrigerante pelo mapa, para isto este veículo deveria se locomover entre os pontos de referência e abrir uma porta. Quando o VANT carregava a lata, ela era posicionada acima da linha dos lasers e acompanhava o VANT, já quando descarregada se comportava naturalmente.

O papel da interface do quadricóptero era receber a ação, extrair seus parâmetros, executa-lá em baixo nível de acordo com seus parâmetros. Com esta interface e o sistema de planejamento, o VANT foi capaz de atingir seus objetivos com sucesso com base nas designações do plano. O que mostra que o sistema de planejamento é preparado não só para a síntese e análise mas também para a execução do plano. O sistema foi capaz de interagir automaticamente com o plano, que agrega inteligência ao VANT a partir da deliberação prévia das consequências. A figura 27 mostra como o sistema auxilia na operação do VANT. A esquerda está disposto o supervisor com informações do plano e do estado atual, acima está posicionado o visualizado com o estado centralizado que permite a análise visual do plano e do progresso e por fim o Gazebo com execução das ações.

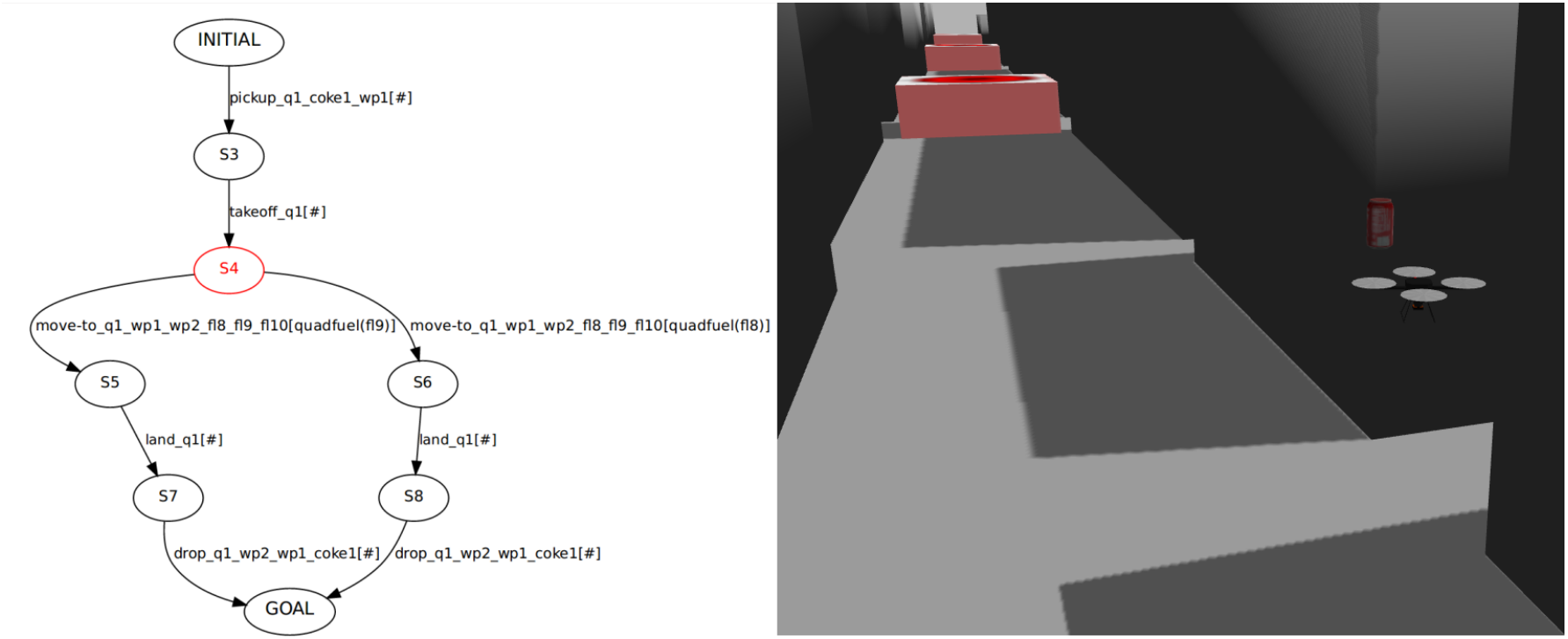



Figura 26 – Simulação no Domínio Proposto

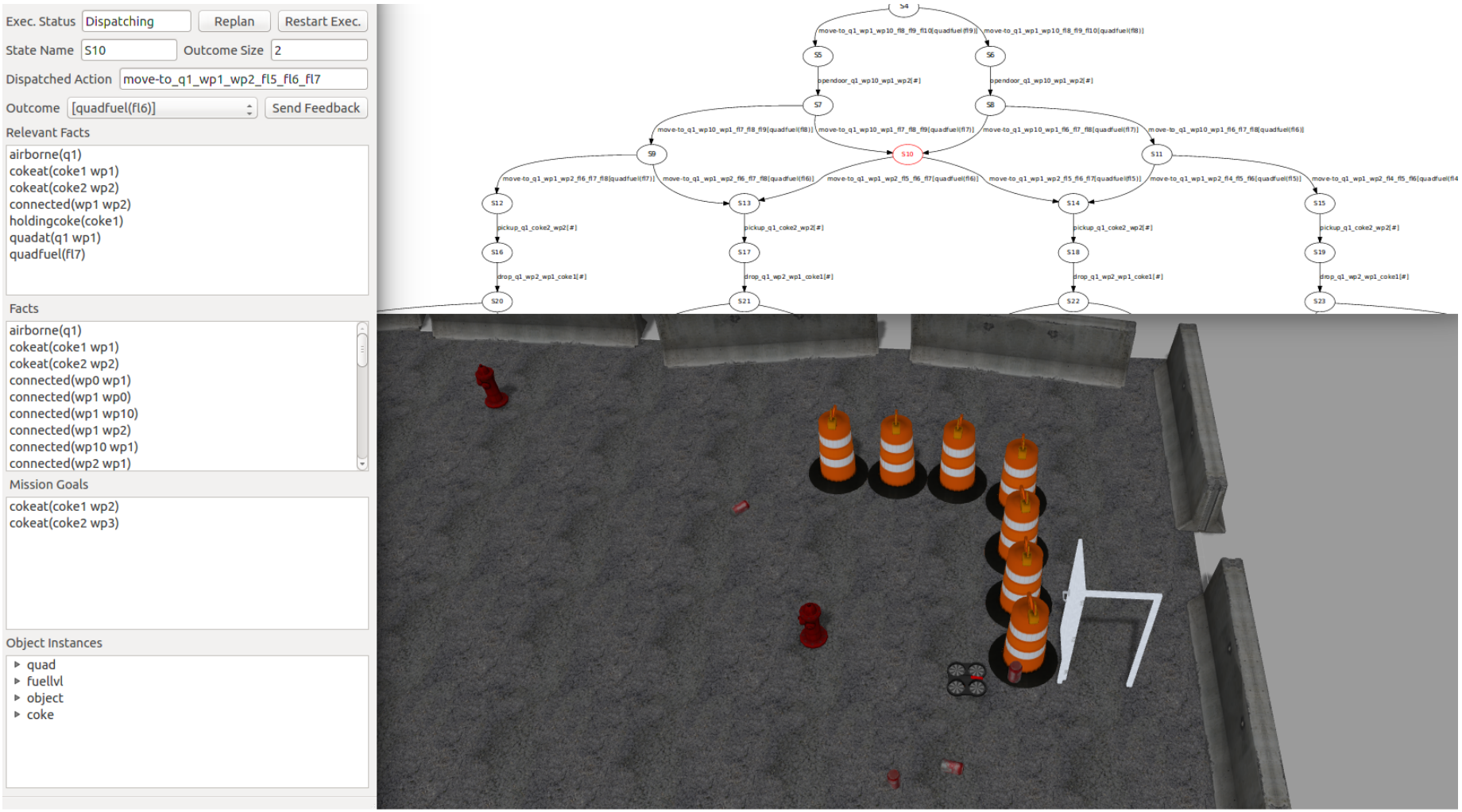



Figura 27 – Ambiente de simulação

# 8 Conclusão e Trabalhos Futuros

Neste trabalho, apresentamos uma estrutura para executar planos probabilísticos em sistemas ROS e com intuito de investigar sua aplicação em VANTs. Este *software* se baseia no ROSPlan buscando implementar uma estrutura de planejador desacoplada da aplicação que permita a execução de diferentes planejadores em diversas plataformas robóticas.

O pacote proposto utiliza o plano representado por meio de máquinas de estados finitos no formato arquivo DOT para a visualização organizada e execução do plano. Este sistema suporta o Prob-PRP como seu planejador probabilístico padrão, mas também suporta outras variantes do PRP, uma vez que todos eles têm o mesmo formato de saída. A proposta foi comparada a outros sistemas de planejamento padrão dentro do ROS, evidenciando a possibilidade de resolver problemas probabilísticos por meio de ciclos fortes. Este *solver* foi aplicado no domínio de entrega de pacotes por VANTs e foram realizadas considerações sobre sua aplicabilidade em sistemas com VANTs em geral, assim como artifícios de ajuste para o planejador. Por fim, o sistema foi simulado no *software* Gazebo, mostrando como seria a interação entre o sistema de planejamento e uma aplicação mais prática. O artigo extraído deste trabalho, anexo, foi apresentado e publicado no XXII Seminário Brasileiro de Automática.

Dentre as limitações do sistema de planejamento proposto, podemos citar a incapacidade de realizar replanejamento. Embora o domínio simulado possa apresentar apenas os resultados modelados, isso pode não acontecer em uma aplicação prática. Portanto, é relevante incorporar o replanejamento na estrutura, o que requer uma ferramenta para geração de problemas e um gerenciamento do conhecimento assim como no ROSPlan. No presente momento, esta estrutura não suporta ações simultâneas. Essa é uma limitação conhecida de execução em se usar máquinas de estados. Esta limitação poderia ser reduzida com a expansão das máquinas de estado como na implementação de um *buffer* de *feedback* de ação, assim como em um Autômato com Pilha, que poderia examinado em outro trabalho. Outra implementação pertinente seria a simulação de Monte-Carlo que extraísse automáticamente os dados dos arquivos PDDL, para facilita a análise do plano. No futuro, essa estrutura também poderia ser mesclada em outras estruturas de planejamento como o ROSPlan.

Quanto ao planejador escolhido, este *software* foi capaz de cumprir com sua proposta de maximizar a probabilidade de execução do plano na maioria das situações. No entanto alguns problemas foram identificados, principalmente nos quais o sistema se encontrava com abundância de recursos. O principal problema foi a deliberação por sequên-

cias de ações desnecessárias que não impactavam na eficácia do plano, em que o planejador faz uso principalmente para simplificação do planejamento e compactação do plano.

Outra caracteristíca observada é que a estratégia do planejador não aparenta ser oportunistas, onde o tratamento de estados busca conectá-los a planos pré calculados ao invés de buscar uma melhor alternativa. No raciocínio entre diferentes ações probabilísticas, a heurística implementada pelo Prob-PRP, as vezes falha em selecionar os efeitos mais prováveis ao invés dos mais benéficos. Os desperdícios no plano comprometem a sua eficiência mas não sua eficácia, pois o planejador têm como meta principal elevar as chances de atingir os objetivos.

O planejador apenas falhou em maximizar suas chances quando inseridas probabilidades de falha inerentes a operação, onde a meta de evitar operações arriscadas compromete a exploração de alternativas pelo planejador. As soluções dadas por este planejador são compactas e possuem complexidade inferior a outros planejadores probabilísticos. Nele as probabilidades são utilizadas para a minimização do caminho médio do plano. O PRP e suas variantes ainda estão em desenvolvimento e melhorias nos mecanismos de buscas ou surgimento de outras variantes ainda são esperadas. A aplicação deste planejador torna a aplicação em VANTs mais segura através da deliberação dos possíveis efeitos. Outra alternativa na mesma linha de planejadores probabilísticos de baixa complexidade computacional que poderia ser explorado é os planejadores para problemas de menor caminho, o qual teve uma categoria específica no IPPC 2018.

O pacote foi desenvolvido para operar dentro do ambiente ROS. Este ambiente de robótica oferece uma estrutura que permite o desenvolvimento modular do sistema de planejamento DOTPlan e facilita a comunicação entre aplicações desenvolvidas. Outra vantagem deste *middleware* é o aproveitamento de pacotes de terceiros, o que viabiliza o emprego de ferramentas mais sofisticadas em sistemas robóticos. A simulação da aplicação utiliza técnicas complexas como modelagem de sensores, modelagem da aeronave, técnicas de navegação e simulador 3D de forma gratuita, a partir do aproveitamento de trabalhos de colaboradores neste ambiente de desenvolvimento.

A simulação demonstrou a interação entre a aplicação e o sistema de planejamento, contudo sua fidelidade deve ser aprimorada para que possam ser extraídos resultados mais conclusivos. Uma implantação relevante seria o controle em baixo nível para avaliar o impacto do controle no consumo de bateria sobre a influência de distúrbios como vento e de manobras evasivas. Outra possível implantação é a navegação acima do nível das construções o que aproveitaria melhor a operação com VANTs. Poderiam ser exploradas também outras aplicações como o resgate, buscas, vistorias e agricultura que já utilizam VANTs.

# Referências


Agência Nacional de Aviação Civil. *REQUISITOS GERAIS PARAAERONAVES NÃO TRIPULADAS DE USO CIVIL*. 2017. RBAC-E nº 94, Resolução nº 419. 16

ALBORE, A.; PALACIOS, H.; GEFFNER, H. A translation-based approach to contingent planning. In: *IJCAI*. [S.l.: s.n.], 2009. p. 1623–1628. 30

BENTON, J.; COLES, A. J.; COLES, A. Temporal planning with preferences and time-dependent continuous costs. In: *ICAPS*. [S.l.: s.n.], 2012. v. 77, p. 78. 27

BERNARDINI, S.; FOX, M.; LONG, D. Planning the behaviour of low-cost quadcopters for surveillance missions. In: PORTSMOUTH, NH. *ICAPS*. [S.l.], 2014. p. 445–453. 15, 17, 47

CAMACHO, A. et al. From fond to probabilistic planning: Guiding search for quality policies. In: *Workshop on Heuristic Search and Domain Independent Planning, ICAPS*. [S.l.: s.n.], 2015. 38

CANTONI, L. F.; CAMPOS, M. F.; CHAIMOWICZ, L. Investigacao da linguagem pddl no planejamento de missoes para rob^os aéreos. In: *X SBAI*. [S.l.: s.n.], 2011. 17

CASHMORE, M. et al. Rosplan: Planning in the robot operating system. In: *ICAPS*. [S.l.: s.n.], 2015. p. 333–341. 9, 17, 46, 47

CIMATTI, A. et al. Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, Elsevier, v. 147, n. 1-2, p. 35–84, 2003. 35

CROSBY, M. et al. Integrating mission, logistics, and task planning for skills-based robot control in industrial kitting applications. In: *Ceur Workshop Proceedings*. [S.l.: s.n.], 2017. v. 1782. 17

FONSECA, J. *Xdot.py: An interactive viewer for graphs written in Graphviz's dot language*. 2017. Https://github.com/jrfonseca/xdot.py. 62

FU, J. et al. Simple and fast strong cyclic planning for fully-observable nondeterministic planning problems. In: *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 2011. v. 22, n. 3, p. 1949. 36

GERTLER, J. *US unmanned aerial systems*. 2012. Library of Congress Washington DC Congressional Research Service. 15

GHALLAB, M.; NAU, D.; TRAVERSO, P. *Automated Planning: theory and practice*. [S.l.]: Elsevier, 2004. 9, 16, 17, 20, 21, 25, 28, 58

GLADE, D. *Unmanned aerial vehicles: Implications for military operations*. 2000. Air Univ Press Maxwell Afb Al. 15

HARRIS, C. A. *Online plan modification in uncertain resource-constrained environments*. Tese (Doutorado) — University of Birmingham, 2015. 27, 29

HOFFMANN, J.; BRAFMAN, R. Contingent planning via heuristic forward search with implicit belief states. In: *Proc. ICAPS*. [S.l.: s.n.], 2005. v. 2005. 30

HOPCROFT, J. E.; MOTWANI, R.; ULLMAN, J. D. Automata theory, languages, and computation. *International Edition*, v. 24, 2006. 41

IOCCHI, L. et al. A practical framework for robust decision-theoretic planning and execution for service robots. In: *ICAPS*. [S.l.: s.n.], 2016. p. 486–494. 42

KELLER, T.; EYERICH, P. Prost: Probabilistic planning based on uct. In: *ICAPS*. [S.l.: s.n.], 2012. 32

LITTLE, I.; THIEBAUX, S. et al. Probabilistic planning vs. replanning. In: *ICAPS Workshop on IPC: Past, Present and Future*. [S.l.: s.n.], 2007. 29, 67

MCDERMOTT, D. et al. Pddl-the planning domain definition language. 1998. 25

MCDERMOTT, D. M. The 1998 ai planning systems competition. *AI magazine*, v. 21, n. 2, p. 35, 2000. 92

MEYER, J. et al. Comprehensive simulation of quadrotor uavs using ros and gazebo. In: SPRINGER. *International conference on simulation, modeling, and programming for autonomous robots*. [S.l.], 2012. p. 400–411. 54

MUISE, C. et al. Leveraging fond planning technology to solve multi-agent planning problems. *Distributed and Multi-Agent Planning (DMAP-15)*, p. 83, 2015. 39

MUISE, C. J.; BELLE, V.; MCILRAITH, S. A. Computing contingent plans via fully observable non-deterministic planning. In: *AAAI*. [S.l.: s.n.], 2014. p. 2322–2329. 30, 38

MUISE, C. J.; MCILRAITH, S. A.; BECK, J. C. Improved non-deterministic planning by exploiting state relevance. In: *ICAPS*. [S.l.: s.n.], 2012. 9, 32, 33, 35, 36

MUISE, C. J.; MCILRAITH, S. A.; BELLE, V. Non-deterministic planning with conditional effects. In: *ICAPS*. [S.l.: s.n.], 2014. 38

QUIGLEY, M. et al. Ros: an open-source robot operating system. In: KOBE. *ICRA workshop on open source software*. [S.l.], 2009. v. 3, n. 3.2, p. 5. 17, 44

QUINTERO, E. et al. Control of autonomous mobile robots with automated planning. In: CITESEER. *Journal of Physical Agents*. [S.l.], 2011. 17

ROS. *Wiki ROS*. 2007. Disponível em: <http://wiki.ros.org/>. 45, 46

RUSSELL, S. J.; NORVIG, P. *Artificial intelligence: a modern approach*. [S.l.]: Malaysia; Pearson Education Limited,, 2016. 31

SANELLI, V. et al. Short-term human-robot interaction through conditional planning and execution. In: *ICAPS*. [S.l.: s.n.], 2017. 9, 17, 48

SANNER, S. Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University*, p. 32, 2010. 40

SARDINA, S.; D'IPPOLITO, N. Towards fully observable non-deterministic planning as assumption-based automatic synthesis. In: *IJCAI*. [S.l.: s.n.], 2015. p. 3200–3206. 42, 60

SYDNEY, N.; SMYTH, B.; PALEY, D. A. Dynamic control of autonomous quadrotor flight in an estimated wind field. In: IEEE. *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*. [S.l.], 2013. p. 3609–3616. 17

TEICHTEIL-KOENIGSBUCH, F.; INFANTES, G.; KUTER, U. Rff: A robust, ff-based mdp planning algorithm for generating policies with low probability of failure. *Sixth International Planning Competition at ICAPS*, v. 8, 2008. 40

TURNER, D.; LUCIEER, A.; WATSON, C. An automated technique for generating georectified mosaics from ultra-high resolution unmanned aerial vehicle (uav) imagery, based on structure from motion (sfm) point clouds. *Remote Sensing*, Molecular Diversity Preservation International, v. 4, n. 5, p. 1392–1410, 2012. 15

WATTS, A. C.; AMBROSIA, V. G.; HINKLEY, E. A. Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use. *Remote Sensing*, Molecular Diversity Preservation International, v. 4, n. 6, p. 1671–1692, 2012. 15

# Apêndices

# APÊNDICE A – Representação do Problema dos Missionários e Canibais em PDDL

## A.1 Arquivo Domínio

```
(define (domain canibais)
  (:requirements :strips)
  (:predicates (localbarco ?lado)
               (localoposto ?lado1 ?lado2)
               (posterior ?maior ?menor)      ;;sequencia numeros
               (seguromc ?m ?c)               ;;seguro para missionarios
               (missionarios ?numero ?lado)
               (canibais ?numero ?lado)
               (tripulado)
               (carregado_um_m)
               (carregado_dois_m)
               (carregado_um_c)
               (carregado_dois_c)
               (carregado_um_cm)
               )


  (:action mover_barco
            :parameters (?l1 ?l2)
            :precondition (and (localbarco ?l1) (localoposto ?l1 ?l2)(tripulado))
            :effect
            (and (not (localbarco ?l1))
                 (localbarco ?l2)))


  (:action carregar_um_missionario
            :parameters (?x ?y ?w ?l1)
            :precondition (and (missionarios ?x ?l1)(posterior ?x ?y)(localbarco ?l1)
                          (not(tripulado))(canibais ?w ?l1)(seguromc ?y ?w))
            :effect
            (and (missionarios ?y ?l1)(not(missionarios ?x ?l1))(tripulado)
            (carregado_um_m)))


  (:action carregar_dois_missionario
            :parameters (?x ?y ?z ?w ?l1)
            :precondition (and (missionarios ?x ?l1)(posterior ?x ?y)(posterior ?y ?z)
                          (localbarco ?l1)(not(tripulado))(canibais ?w ?l1)
                          (seguromc ?z ?w))
            :effect
            (and (missionarios ?z ?l1)(not(missionarios ?x ?l1))(tripulado)
            (carregado_dois_m)))


  (:action carregar_um_canibal
            :parameters (?x ?y ?l1)
            :precondition (and (canibais ?x ?l1)(posterior ?x ?y)(localbarco ?l1)
                          (not(tripulado)))
```

```
            :effect
            (and (canibais ?y ?l1)(not(canibais ?x ?l1))(tripulado)(carregado_um_c)))

  (:action carregar_dois_canibais
            :parameters (?x ?y ?z ?l1)
            :precondition (and (canibais ?x ?l1)(posterior ?x ?y)(posterior ?y ?z)
                        (localbarco ?l1)(not(tripulado)))
            :effect
            (and (canibais ?z ?l1)(not(canibais ?x ?l1))(tripulado)(carregado_dois_c)))

  (:action carregar_um_missionario_um_canibal
            :parameters (?x ?y ?v ?z  ?l1)
            :precondition (and (canibais ?x ?l1)(posterior ?x ?y)(missionarios ?v ?l1)
                        (posterior ?v ?z)(localbarco ?l1)(not(tripulado)))
            :effect
            (and (canibais ?y ?l1)(not(canibais ?x ?l1))(missionarios ?z ?l1)
            (not(missionarios ?v ?l1))(tripulado)(carregado_um_cm)))

  (:action descarregar_um_missionario
            :parameters (?x ?y ?l1)
            :precondition (and (missionarios ?x ?l1)(posterior ?y ?x)(localbarco ?l1)
                        (carregado_um_m))
            :effect
            (and (missionarios ?y ?l1)(not(missionarios ?x ?l1))(not(tripulado))
            (not(carregado_um_m))))

  (:action descarregar_dois_missionarios
            :parameters (?x ?y ?z ?l1)
            :precondition (and (missionarios ?x ?l1)(posterior ?y ?x)(posterior ?z ?y)
                        (localbarco ?l1)(carregado_dois_m))
            :effect
            (and (missionarios ?z ?l1)(not(missionarios ?x ?l1))(not(tripulado))
            (not(carregado_dois_m))))

  (:action descarregar_um_canibal
            :parameters (?x ?y ?w ?l1)
            :precondition (and (carregado_um_c)(canibais ?x ?l1)(posterior ?y ?x)
                        (missionarios ?w ?l1)(seguromc ?w ?y)(localbarco ?l1))
            :effect
            (and (canibais ?y ?l1)(not(canibais ?x ?l1))(not(tripulado))
            (not(carregado_um_c))))

  (:action descarregar_dois_canibais
            :parameters (?x ?y ?z ?w ?l1)
            :precondition (and (carregado_dois_c)(canibais ?x ?l1)(posterior ?y ?x)
                        (posterior ?z ?y)(missionarios ?w ?l1)(seguromc ?w ?z)
                        (localbarco ?l1))
            :effect
            (and (canibais ?z ?l1)(not(canibais ?x ?l1))(not(tripulado))
            (not(carregado_dois_c))))

  (:action descarregar_um_missionario_um_canibal
            :parameters (?x ?y ?z ?v ?l1)
            :precondition (and (carregado_um_cm)(canibais ?x ?l1)(posterior ?y ?x)
                        (missionarios ?v ?l1)(posterior ?z ?v)(localbarco ?l1)
                        (seguromc ?z ?y))
            :effect
            (and (canibais ?y ?l1)(not(canibais ?x ?l1))(missionarios ?z ?l1)
            (not(missionarios ?v ?l1))(not(tripulado))(not(carregado_um_cm))))
)
```

## A.2 Arquivo Problema

```
(define (problem problema_canibais)

(:domain canibais)

(:objects margem1 margem2 zero um dois tres )

(:INIT (localbarco margem1) (localoposto margem1 margem2) (localoposto margem2 margem1)
(posterior um zero) (posterior dois um) (posterior tres dois)
(seguromc zero zero) (seguromc zero um)(seguromc zero dois) (seguromc zero tres)
(seguromc um um) (seguromc dois um) (seguromc tres um)
(seguromc dois dois) (seguromc tres dois)
(seguromc tres tres)
(missionarios tres margem1)(missionarios zero margem2)(canibais tres margem1)
(canibais zero margem2))


(:goal (AND (missionarios zero margem1)(canibais zero margem1)(missionarios tres margem2)
       (canibais tres margem2)(localbarco margem2)))
)
```

## A.3 Solução

Solução obtida utilizando o planejador FF:

```
0: Carregar_um_missionario_um_canibal Tres Dois Tres Dois Margem1
1: Mover_barco Margem1 Margem2
2: Descarregar_um_missionario_um_canibal Zero Um Um Zero Margem2
3: Carregar_um_missionario Um Zero Um Margem2
4: Mover_barco Margem2 Margem1
5: Descarregar_um_missionario Dois Tres Margem1
6: Carregar_dois_canibais Dois Um Zero Margem1
7: Mover_barco Margem1 Margem2
8: Descarregar_dois_canibais Um Dois Tres Zero Margem2
9: Carregar_um_canibal Tres Dois Margem2
10: Mover_barco Margem2 Margem1
11: Descarregar_um_canibal Zero Um Tres Margem1
12: Carregar_dois_missionario Tres Dois Um Um Margem1
13: Mover_barco Margem1 Margem2
14: Descarregar_dois_missionarios Zero Um Dois Margem2
15: Carregar_um_missionario_um_canibal Dois Um Dois Um Margem2
16: Mover_barco Margem2 Margem1
17: Descarregar_um_missionario_um_canibal Um Dois Dois Um Margem1
18: Carregar_dois_missionario Dois Um Zero Dois Margem1
19: Mover_barco Margem1 Margem2
20: Descarregar_dois_missionarios Um Dois Tres Margem2
```

21: Carregar_um_canibal Um Zero Margem2
22: Mover_barco Margem2 Margem1
23: Descarregar_um_canibal Dois Tres Zero Margem1
24: Carregar_dois_canibais Tres Dois Um Margem1
25: Mover_barco Margem1 Margem2
26: Descarregar_dois_canibais Zero Um Dois Tres Margem2
27: Carregar_um_canibal Dois Um Margem2
28: Mover_barco Margem2 Margem1
29: Descarregar_um_canibal Um Dois Zero Margem1
30: Carregar_dois_canibais Dois Um Zero Margem1
31: Mover_barco Margem1 Margem2
32: Descarregar_dois_canibais Um Dois Tres Tres Margem2

# APÊNDICE B – Competições Internacionais de Planejamento

## B.1 IPC - Internacional Planning Competition

A IPC (*Internacional Planning Competition*) é organizada juntamente com a *Internacional Conference on Automated Planning and Scheduling* (ICAPS). Esta competição surgiu em 1998, com o objetivo de:(1) realizar uma comparação significativa de diferentes sistemas de planejamento,(2) evidenciar o progresso e desafios da área e (3) providenciar problemas relevantes para comparação e referencia em pesquisas futuras (MCDERMOTT, 2000). Nela os planejadores automáticos independentes do domínio são avaliados empiricamente com base em um grupo de problemas. Em sua origem, esta competição abrangia sobre planejamento clássico porém a partir da IPC-2004 começou a avaliar também outras categorias. Já houveram 9 edições desta competição, são elas IPC-1998, IPC-2000, IPC-2002, IPC-2004, IPC-2006, IPC-2008, IPC-2011, IPC-2014, IPC-2018 e IPC-2018. Os categorias abordadas são: clássica, probabilistica, aprendizado e temporal, sendo presentes conforme a necessidade.

## B.2 IPPC - Internacional Probabilistic Planning Competition

A IPPC ( *Probabilistic Planning Competition*) é uma competição que avalia os planejadores probabilisticos segundo uma série de problemas propostos. Esta competição é uma categoria das IPCs e ocorre sempre que são considerados avanços expressivos na área. Já foram realizadas 7 edições desta competição são elas: IPPC-2004, IPPC-2006, IPPC-2008, IPPC-2011, IPPC-2014, IPPC-2016 e IPPC-2018. Sua existência é muito importante para definir padrões para comparação, as técnicas mais relevantes e o futuro da área. A IPPC-2018, ocorrida no final de junho de 2018, foi a última competição da área porém os resultados não foram divulgados até a presente data. Estes resultados devem ser anunciados em breve uma vez que outras categorias da IPC-2018 o fizeram. A IPPC-2018 possui três subcategorias: MDPs dicretas, MDPs contínuas e SPPs (*Shortest Path Problems*).

# APÊNDICE C – Domínio de Entrega em VANTs

## C.1 Básico

### C.1.1 Arquivo Domínio

```
(define (domain quaddeliver)
(:requirements :typing)
(:types package waypoint fuellvl - object)

(:predicates (deliverat ?pkg - package ?wp - waypoint)
             (pkgat ?pkg - package ?wp - waypoint)
             (quadat ?wp - waypoint)
             (inquad ?pkg - package)
             (delivered ?pkg - package)
             (connected ?wp1 ?wp2 - waypoint)
             (quadfuel ?fuel - fuellvl)
             (fuelmapping ?lowerlvl ?higherlvl - fuellvl)
         (minimumfuel ?minfuel - fuellvl)
             (stationat ?wp - waypoint)
         (onair)
         (grounded)
         (free)
         (notloaded))

(:action land
 :parameters (?fuel1 ?fuel2 - fuellvl)
 :precondition (and (onair)(quadfuel ?fuel2)(fuelmapping ?fuel1 ?fuel2)(free))
 :effect (and (not(free))(grounded) (not(onair)) (quadfuel ?fuel1)
              (not(quadfuel ?fuel2))))

(:action take-off
 :parameters (?fuel1 ?fuel2 - fuellvl)
 :precondition (and (grounded)(quadfuel ?fuel2)(fuelmapping ?fuel1 ?fuel2)(free))
 :effect (and (not(free))(onair) (not(grounded)) (quadfuel ?fuel1
              (not(quadfuel ?fuel2))))

(:action load-package
 :parameters (?pkg - package ?wp - waypoint)
 :precondition (and (pkgat ?pkg ?wp) (quadat ?wp)(grounded)(notloaded))
 :effect (and(free)(inquad ?pkg)(not(notloaded)) ))

(:action move-to
 :parameters (?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and(quadat ?wp1)(connected ?wp1 ?wp2)(quadfuel ?fuel3)
                   (fuelmapping ?fuel1 ?fuel2)(fuelmapping ?fuel2 ?fuel3)(onair))
 :effect (and (free)(not (quadat ?wp1)) (quadat ?wp2)
               (probabilistic 0.7 (and(quadfuel ?fuel2)(not(quadfuel ?fuel3)))
                              0.3 (and(quadfuel ?fuel1)(not(quadfuel ?fuel3)))
                              )))
```

```
(:action refuel
 :parameters (?wp1 - waypoint ?fuel1 ?fuel2 - fuellvl)
 :precondition (and(quadat ?wp1)(stationat ?wp1)(quadfuel ?fuel1)
                   (maximumfuel ?fuel2)(grounded))
 :effect (and (free)(quadfuel ?fuel2)(not(quadfuel ?fuel1))))

(:action deliver-package
 :parameters (?pkg - package ?wp1 ?wp2 - waypoint)
 :precondition (and (pkgat ?pkg ?wp1) (quadat ?wp2) (deliverat ?pkg ?wp2)
                    (inquad ?pkg)(grounded))
 :effect (and (free)(not (pkgat ?pkg ?wp1)) (pkgat ?pkg ?wp2) (not(inquad ?pkg))
              (delivered ?pkg)(notloaded)))
)
```

### C.1.2 Arquivo Problema - Experimento 1

```
(define (problem pquaddeliver)
(:domain quaddeliver)
(:objects
          pkg1 - package
          wp1 wp2 wp3 wp4 wp5 wp6 - waypoint
          fl1 fl2 fl3 fl4 fl5 fl6 fl7 fl8 fl9 fl10 fl11 fl12 - fuellvl)

(:init
        (quadat wp1)
        (pkgat pkg1 wp1)
        (deliverat pkg1 wp4)
        (connected wp1 wp2)
        (connected wp2 wp1)
        (connected wp3 wp2)
        (connected wp2 wp3)
        (connected wp4 wp3)
        (connected wp3 wp4)
        (connected wp5 wp3)
        (connected wp3 wp5)
        (connected wp5 wp6)
        (connected wp6 wp5)
        (fuelmapping fl1 fl2)
        (fuelmapping fl2 fl3)
        (fuelmapping fl3 fl4)
        (fuelmapping fl4 fl5)
        (fuelmapping fl5 fl6)
        (fuelmapping fl6 fl7)
        (fuelmapping fl7 fl8)
        (fuelmapping fl8 fl9)
        (fuelmapping fl9 fl10)
        (fuelmapping fl10 fl11)
        (fuelmapping fl11 fl12)
        (quadfuel fl12)
        (maximumfuel fl12)
        (stationat wp4)
        (grounded)
        (free))

(:goal (and
          (delivered pkg1)
          (quadat wp5)
          (grounded)
          )))
```

### C.1.3 Arquivo Problema - Experimento 2

```
(define (problem pquaddeliver)
(:domain quaddeliver)
(:objects
          pkg1 pkg2 pkg3 - package
          wp1 wp2 wp3 wp4 wp5 wp6 wp7 wp8 wp9 wp10 wp11 wp12 - waypoint
          fl1 fl2 fl3 fl4 fl5 fl6 fl7 fl8 fl9 fl10 - fuellvl)

(:init
        (quadat wp1)
        (pkgat pkg1 wp2)
        (deliverat pkg1 wp7)
        (pkgat pkg2 wp12)
        (deliverat pkg2 wp11)
        (pkgat pkg3 wp12)
        (deliverat pkg3 wp2)
        (connected wp1 wp2)
        (connected wp1 wp4)
        (connected wp2 wp1)
        (connected wp2 wp3)
        (connected wp2 wp5)
        (connected wp3 wp2)
        (connected wp3 wp6)
        (connected wp4 wp1)
        (connected wp4 wp5)
        (connected wp4 wp7)
        (connected wp5 wp2)
        (connected wp5 wp4)
        (connected wp5 wp6)
        (connected wp5 wp8)
        (connected wp6 wp3)
        (connected wp6 wp5)
        (connected wp6 wp9)
        (connected wp7 wp4)
        (connected wp7 wp8)
        (connected wp7 wp10)
        (connected wp8 wp5)
        (connected wp8 wp7)
        (connected wp8 wp9)
        (connected wp8 wp11)
        (connected wp9 wp6)
        (connected wp9 wp8)
        (connected wp9 wp12)
        (connected wp10 wp7)
        (connected wp10 wp11)
        (connected wp11 wp8)
        (connected wp11 wp10)
        (connected wp11 wp12)
        (connected wp12 wp9)
        (connected wp12 wp11)
        (fuelmapping fl1 fl2)
        (fuelmapping fl2 fl3)
        (fuelmapping fl3 fl4)
        (fuelmapping fl4 fl5)
        (fuelmapping fl5 fl6)
        (fuelmapping fl6 fl7)
        (fuelmapping fl7 fl8)
        (fuelmapping fl8 fl9)
        (fuelmapping fl9 fl10)
```

```
        (quadfuel fl10)
        (maximumfuel fl10)
        (minimumfuel fl1)
        (stationat wp2)
        (stationat wp12)
        (grounded)
        (free)
        (notloaded))

(:goal (and
          (delivered pkg1)
          (delivered pkg2)
          (delivered pkg3)
          )))
```

## C.2 Extensão I

### C.2.1 Arquivo Domínio

```
(define (domain quaddeliver)
(:requirements :typing)
(:types package waypoint fuellvl weightlvl - object)
(:constants w0 w1 w2 w3 w4 w5 - weightlvl)
(:predicates (deliverat ?pkg - package ?wp - waypoint)
             (pkgat ?pkg - package ?wp - waypoint)
             (quadat ?wp - waypoint)
             (inquad ?pkg - package)
             (delivered ?pkg - package)
             (connected ?wp1 ?wp2 - waypoint)
             (quadfuel ?fuel - fuellvl)
             (fuelmapping ?lowerlvl ?higherlvl - fuellvl)
             (minimumfuel ?minfuel - fuellvl)
             (maximumfuel ?maxfuel - fuellvl)
             (stationat ?wp - waypoint)
             (onair)
             (grounded)
             (free)
             (notloaded)
             (addweight ?w1 ?w2 ?w3 - weightlvl)
             (weightquad ?w - weightlvl)
             (weightpkg ?pkg - package ?w - weightlvl))


(:action land
 :parameters (?fuel1 ?fuel2 - fuellvl)
 :precondition (and (onair)(quadfuel ?fuel2)(fuelmapping ?fuel1 ?fuel2)(free))
 :effect (and (not(free))(grounded) (not(onair)) (quadfuel ?fuel1)
              (not(quadfuel ?fuel2))))

(:action take-off
 :parameters (?fuel1 ?fuel2 - fuellvl)
 :precondition (and (grounded)(quadfuel ?fuel2)(fuelmapping ?fuel1 ?fuel2)(free))
 :effect (and (not(free))(onair) (not(grounded)) (quadfuel ?fuel1)
              (not(quadfuel ?fuel2))))

(:action load-package
 :parameters (?pkg - package ?wp - waypoint ?w1 ?w2 ?w3 - weightlvl )
 :precondition (and (pkgat ?pkg ?wp) (quadat ?wp)(grounded)(weightquad ?w1)
```

```
                    (weightpkg ?pkg ?w2)(addweight ?w1 ?w2 ?w3)
                    (not(delivered ?pkg))(not(inquad ?pkg)))
 :effect (and(free)(inquad ?pkg)(not(weightquad ?w1))(weightquad ?w3)))


(:action move-with-w0-to
 :parameters (?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and(quadat ?wp1)(connected ?wp1 ?wp2)(quadfuel ?fuel3)
                   (fuelmapping ?fuel1 ?fuel2)(fuelmapping ?fuel2 ?fuel3)
                   (onair)(weightquad w0))
 :effect (and (free)(not (quadat ?wp1)) (quadat ?wp2)(not(quadfuel ?fuel3))
               (probabilistic 0.6 (quadfuel ?fuel2)
                              0.4 (quadfuel ?fuel1)
                              )))
(:action move-with-w1-to
 :parameters (?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and(quadat ?wp1)(connected ?wp1 ?wp2)(quadfuel ?fuel3)
                   (fuelmapping ?fuel1 ?fuel2)(fuelmapping ?fuel2 ?fuel3)
                   (onair)(weightquad w1))
 :effect (and (free)(not (quadat ?wp1)) (quadat ?wp2)(not(quadfuel ?fuel3))
               (probabilistic 0.55 (quadfuel ?fuel2)
                              0.45 (quadfuel ?fuel1)
                              )))
(:action move-with-w2-to
 :parameters (?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and(quadat ?wp1)(connected ?wp1 ?wp2)(quadfuel ?fuel3)
                   (fuelmapping ?fuel1 ?fuel2)(fuelmapping ?fuel2 ?fuel3)
                   (onair)(weightquad w2))
 :effect (and (free)(not (quadat ?wp1)) (quadat ?wp2)(not(quadfuel ?fuel3))
               (probabilistic 0.5 (quadfuel ?fuel2)
                              0.5 (quadfuel ?fuel1)
                              )))
(:action move-with-w3-to
 :parameters (?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and(quadat ?wp1)(connected ?wp1 ?wp2)(quadfuel ?fuel3)
                    (fuelmapping ?fuel1 ?fuel2)(fuelmapping ?fuel2 ?fuel3)
                    (onair)(weightquad w3))
 :effect (and (free)(not (quadat ?wp1)) (quadat ?wp2)(not(quadfuel ?fuel3))
               (probabilistic 0.45 (quadfuel ?fuel2)
                              0.55 (quadfuel ?fuel1)
                              )))
(:action move-with-w4-to
 :parameters (?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and(quadat ?wp1)(connected ?wp1 ?wp2)(quadfuel ?fuel3)
                    (fuelmapping ?fuel1 ?fuel2)(fuelmapping ?fuel2 ?fuel3)
                    (onair)(weightquad w4))
 :effect (and (free)(not (quadat ?wp1)) (quadat ?wp2)(not(quadfuel ?fuel3))
               (probabilistic 0.4 (quadfuel ?fuel2)
                              0.6 (quadfuel ?fuel1)
                              )))
(:action move-with-w5-to
 :parameters (?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and(quadat ?wp1)(connected ?wp1 ?wp2)(quadfuel ?fuel3)
                   (fuelmapping ?fuel1 ?fuel2)(fuelmapping ?fuel2 ?fuel3)
                   (onair)(weightquad w5))
 :effect (and (free)(not (quadat ?wp1)) (quadat ?wp2)(not(quadfuel ?fuel3))
               (probabilistic 0.35 (quadfuel ?fuel2)
                              0.65 (quadfuel ?fuel1)
                              )))
```

```
(:action refuel
 :parameters (?wp1 - waypoint ?fuel1 ?fuel2 - fuellvl)
 :precondition (and(quadat ?wp1)(stationat ?wp1)(quadfuel ?fuel1)
                   (maximumfuel ?fuel2)(grounded))
 :effect (and (free)(quadfuel ?fuel2)(not(quadfuel ?fuel1))))


(:action deliver-package
 :parameters (?pkg - package ?wp1 ?wp2 - waypoint ?w1 ?w2 ?w3 - weightlvl)
 :precondition (and (pkgat ?pkg ?wp1) (quadat ?wp2) (deliverat ?pkg ?wp2)
                    (inquad ?pkg)(grounded)(weightquad ?w3)
                    (weightpkg ?pkg ?w2)(addweight ?w1 ?w2 ?w3))
 :effect (and (free)(not (pkgat ?pkg ?wp1)) (pkgat ?pkg ?wp2) (not(inquad ?pkg))
              (delivered ?pkg)(not(weightquad ?w3))(weightquad ?w1)))
)
```

### C.2.2 Arquivo Problema - Experimento 3

```
(define (problem pquaddeliver)
(:domain quaddeliver)
(:objects
          pkg1 pkg2 pkg3 pkg4 - package
          wp1 wp2 wp3 wp4 wp5 wp6 wp7 wp8 wp9 wp10 wp11 wp12 - waypoint
          fl1 fl2 fl3 fl4 fl5 fl6 fl7 fl8 fl9 fl10 - fuellvl)

(:init
        (quadat wp1)
        (pkgat pkg1 wp2)
        (deliverat pkg1 wp7)
        (pkgat pkg2 wp12)
        (deliverat pkg2 wp11)
        (pkgat pkg3 wp12)
        (deliverat pkg3 wp2)
        (connected wp1 wp2)
        (connected wp1 wp4)
        (connected wp2 wp1)
        (connected wp2 wp3)
        (connected wp2 wp5)
        (connected wp3 wp2)
        (connected wp3 wp6)
        (connected wp4 wp1)
        (connected wp4 wp5)
        (connected wp4 wp7)
        (connected wp5 wp2)
        (connected wp5 wp4)
        (connected wp5 wp6)
        (connected wp5 wp8)
        (connected wp6 wp3)
        (connected wp6 wp5)
        (connected wp6 wp9)
        (connected wp7 wp4)
        (connected wp7 wp8)
        (connected wp7 wp10)
        (connected wp8 wp5)
        (connected wp8 wp7)
        (connected wp8 wp9)
        (connected wp8 wp11)
        (connected wp9 wp6)
        (connected wp9 wp8)
```

```
        (connected wp9 wp12)
        (connected wp10 wp7)
        (connected wp10 wp11)
        (connected wp11 wp8)
        (connected wp11 wp10)
        (connected wp11 wp12)
        (connected wp12 wp9)
        (connected wp12 wp11)
        (fuelmapping fl1 fl2)
        (fuelmapping fl2 fl3)
        (fuelmapping fl3 fl4)
        (fuelmapping fl4 fl5)
        (fuelmapping fl5 fl6)
        (fuelmapping fl6 fl7)
        (fuelmapping fl7 fl8)
        (fuelmapping fl8 fl9)
        (fuelmapping fl9 fl10)
        (quadfuel fl10)
        (maximumfuel fl10)
        (minimumfuel fl1)
        (stationat wp2)
        (stationat wp12)
        (grounded)
        (notloaded)
        (weightquad w0)
        (addweight w0 w1 w1)(addweight w0 w2 w2)(addweight w0 w3 w3)(addweight w0 w4 w4)
        (addweight w0 w5 w5)
        (addweight w1 w1 w2)(addweight w1 w2 w3)(addweight w1 w3 w4)(addweight w1 w4 w5)
        (addweight w2 w1 w3)(addweight w2 w2 w4)(addweight w2 w3 w5)
        (addweight w3 w1 w4)(addweight w3 w2 w5)
        (addweight w4 w1 w5)
        (weightpkg pkg1 w1)
        (weightpkg pkg2 w2)
        (weightpkg pkg3 w3)
        (free))

(:goal (and
          (delivered pkg1)
          (delivered pkg2)
          (delivered pkg3)
          )))
```

## C.3 Extensão 2

### C.3.1 Arquivo Domínio

```
(define (domain quaddeliver)
(:requirements :typing)
(:types package waypoint fuellvl weightlvl - object)
(:constants w0 w1 w2 w3 w4 w5 - weightlvl)
(:predicates (deliverat ?pkg - package ?wp - waypoint)
             (pkgat ?pkg - package ?wp - waypoint)
             (quadat ?wp - waypoint)
             (inquad ?pkg - package)
             (delivered ?pkg - package)
             (connected ?wp1 ?wp2 - waypoint)
             (quadfuel ?fuel - fuellvl)
             (fuelmapping ?lowerlvl ?higherlvl - fuellvl)
```

```
            (minimumfuel ?minfuel - fuellvl)
            (maximumfuel ?maxfuel - fuellvl)
            (stationat ?wp - waypoint)
            (onair)
            (grounded)
            (free)
            (notloaded)
            (addweight ?w1 ?w2 ?w3 - weightlvl)
            (weightquad ?w - weightlvl)
            (weightpkg ?pkg - package ?w - weightlvl))


(:action land
 :parameters (?fuel1 ?fuel2 - fuellvl)
 :precondition (and (onair)(quadfuel ?fuel2)(fuelmapping ?fuel1 ?fuel2)(free))
 :effect (and (not(free))(grounded) (not(onair)) (quadfuel ?fuel1)
              (not(quadfuel ?fuel2))))

(:action take-off
 :parameters (?fuel1 ?fuel2 - fuellvl)
 :precondition (and (grounded)(quadfuel ?fuel2)(fuelmapping ?fuel1 ?fuel2)(free))
 :effect (and (not(free))(onair) (not(grounded)) (quadfuel ?fuel1)
              (not(quadfuel ?fuel2))))

(:action load-package
 :parameters (?pkg - package ?wp - waypoint ?w1 ?w2 ?w3 - weightlvl )
 :precondition (and (pkgat ?pkg ?wp) (quadat ?wp)(grounded)(weightquad ?w1)
                    (weightpkg ?pkg ?w2)(addweight ?w1 ?w2 ?w3)
                    (not(delivered ?pkg))(not(inquad ?pkg)))
 :effect (and(free)(inquad ?pkg)(not(weightquad ?w1))(weightquad ?w3)))


(:action move-with-w0-to
 :parameters (?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and(quadat ?wp1)(connected ?wp1 ?wp2)(quadfuel ?fuel3)
                   (fuelmapping ?fuel1 ?fuel2)(fuelmapping ?fuel2 ?fuel3)
                   (onair)(weightquad w0))
 :effect (and (free)(not (quadat ?wp1)) (quadat ?wp2)(not(quadfuel ?fuel3))
               (probabilistic 0.6 (quadfuel ?fuel2)
                              0.399 (quadfuel ?fuel1)
                         0.001 (and(quadfuel ?fuel1)(recovering)(not(operating)))
                              )))
(:action move-with-w1-to
 :parameters (?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and(quadat ?wp1)(connected ?wp1 ?wp2)(quadfuel ?fuel3)
                   (fuelmapping ?fuel1 ?fuel2)(fuelmapping ?fuel2 ?fuel3)
                   (onair)(weightquad w1))
 :effect (and (free)(not (quadat ?wp1)) (quadat ?wp2)(not(quadfuel ?fuel3))
               (probabilistic 0.55 (quadfuel ?fuel2)
                              0.4489 (quadfuel ?fuel1)
                         0.0011 (and(quadfuel ?fuel1)(recovering)(not(operating)))
                              )))
(:action move-with-w2-to
 :parameters (?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and(quadat ?wp1)(connected ?wp1 ?wp2)(quadfuel ?fuel3)
                   (fuelmapping ?fuel1 ?fuel2)(fuelmapping ?fuel2 ?fuel3)
                   (onair)(weightquad w2))
 :effect (and (free)(not (quadat ?wp1)) (quadat ?wp2)(not(quadfuel ?fuel3))
               (probabilistic 0.53 (quadfuel ?fuel2)
                              0.4687 (quadfuel ?fuel1)
```

```
                    0.0013 (and(quadfuel ?fuel1)(recovering)(not(operating)))
                           )))
(:action move-with-w3-to
 :parameters (?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and(quadat ?wp1)(connected ?wp1 ?wp2)(quadfuel ?fuel3)
                   (fuelmapping ?fuel1 ?fuel2)(fuelmapping ?fuel2 ?fuel3)
                   (onair)(weightquad w3))
 :effect (and (free)(not (quadat ?wp1)) (quadat ?wp2)(not(quadfuel ?fuel3))
              (probabilistic 0.5 (quadfuel ?fuel2)
                           0.4984 (quadfuel ?fuel1)
                    0.0016 (and(quadfuel ?fuel1)(recovering)(not(operating)))
                           )))
(:action move-with-w4-to
 :parameters (?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and(quadat ?wp1)(connected ?wp1 ?wp2)(quadfuel ?fuel3)
                   (fuelmapping ?fuel1 ?fuel2)(fuelmapping ?fuel2 ?fuel3)
                   (onair)(weightquad w4))
 :effect (and (free)(not (quadat ?wp1)) (quadat ?wp2)(not(quadfuel ?fuel3))
              (probabilistic 0.47 (quadfuel ?fuel2)
                           0.528 (quadfuel ?fuel1)
                    0.002 (and(quadfuel ?fuel1)(recovering)(not(operating)))
                           )))
(:action move-with-w5-to
 :parameters (?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and(quadat ?wp1)(connected ?wp1 ?wp2)(quadfuel ?fuel3)
                  (fuelmapping ?fuel1 ?fuel2)(fuelmapping ?fuel2 ?fuel3)
                  (onair)(weightquad w5))
 :effect (and (free)(not (quadat ?wp1)) (quadat ?wp2)(not(quadfuel ?fuel3))
              (probabilistic 0.45 (quadfuel ?fuel2)
                           0.5475 (quadfuel ?fuel1)
                    0.0025 (and(quadfuel ?fuel1)(recovering)(not(operating)))
                           )))
(:action recover
 :parameters ()
 :precondition (recovering)
 :effect (and(not(recovering))(probabilistic 0.8 (operating))))

(:action refuel
 :parameters (?wp1 - waypoint ?fuel1 ?fuel2 - fuellvl)
 :precondition (and(quadat ?wp1)(stationat ?wp1)(quadfuel ?fuel1)
                  (maximumfuel ?fuel2)(grounded))
 :effect (and (free)(quadfuel ?fuel2)(not(quadfuel ?fuel1))))


(:action deliver-package
 :parameters (?pkg - package ?wp1 ?wp2 - waypoint ?w1 ?w2 ?w3 - weightlvl)
 :precondition (and (pkgat ?pkg ?wp1) (quadat ?wp2) (deliverat ?pkg ?wp2)
                    (inquad ?pkg)(grounded)(weightquad ?w3)
                    (weightpkg ?pkg ?w2)(addweight ?w1 ?w2 ?w3))
 :effect (and (free)(not (pkgat ?pkg ?wp1)) (pkgat ?pkg ?wp2) (not(inquad ?pkg))
              (delivered ?pkg)(not(weightquad ?w3))(weightquad ?w1)))
)
```

### C.3.2 Arquivo Problema - Experimento 4

```
(define (problem pquaddeliver)
(:domain quaddeliver)
(:objects
          pkg1 pkg2 pkg3 - package
```

```
        wp1 wp2 wp3 wp4 wp5 wp6 wp7 wp8 wp9 wp10 wp11 wp12 - waypoint
        fl1 fl2 fl3 fl4 fl5 fl6 fl7 fl8 fl9 fl10 - fuellvl)

(:init
        (quadat wp1)
        (pkgat pkg1 wp2)
        (deliverat pkg1 wp7)
        (pkgat pkg2 wp12)
        (deliverat pkg2 wp11)
        (pkgat pkg3 wp12)
        (deliverat pkg3 wp2)
        (connected wp1 wp2)
        (connected wp1 wp4)
        (connected wp2 wp1)
        (connected wp2 wp3)
        (connected wp2 wp5)
        (connected wp3 wp2)
        (connected wp3 wp6)
        (connected wp4 wp1)
        (connected wp4 wp5)
        (connected wp4 wp7)
        (connected wp5 wp2)
        (connected wp5 wp4)
        (connected wp5 wp6)
        (connected wp5 wp8)
        (connected wp6 wp3)
        (connected wp6 wp5)
        (connected wp6 wp9)
        (connected wp7 wp4)
        (connected wp7 wp8)
        (connected wp7 wp10)
        (connected wp8 wp5)
        (connected wp8 wp7)
        (connected wp8 wp9)
        (connected wp8 wp11)
        (connected wp9 wp6)
        (connected wp9 wp8)
        (connected wp9 wp12)
        (connected wp10 wp7)
        (connected wp10 wp11)
        (connected wp11 wp8)
        (connected wp11 wp10)
        (connected wp11 wp12)
        (connected wp12 wp9)
        (connected wp12 wp11)
        (fuelmapping fl1 fl2)
        (fuelmapping fl2 fl3)
        (fuelmapping fl3 fl4)
        (fuelmapping fl4 fl5)
        (fuelmapping fl5 fl6)
        (fuelmapping fl6 fl7)
        (fuelmapping fl7 fl8)
        (fuelmapping fl8 fl9)
        (fuelmapping fl9 fl10)
        (quadfuel fl10)
        (maximumfuel fl10)
        (minimumfuel fl1)
        (stationat wp2)
        (stationat wp12)
        (grounded)
```

```
        (notloaded)
        (weightquad w0)
        (addweight w0 w1 w1)(addweight w0 w2 w2)(addweight w0 w3 w3)(addweight w0 w4 w4)
        (addweight w0 w5 w5)
        (addweight w1 w1 w2)(addweight w1 w2 w3)(addweight w1 w3 w4)(addweight w1 w4 w5)
        (addweight w2 w1 w3)(addweight w2 w2 w4)(addweight w2 w3 w5)
        (addweight w3 w1 w4)(addweight w3 w2 w5)
        (addweight w4 w1 w5)
        (weightpkg pkg1 w1)
        (weightpkg pkg2 w1)
        (weightpkg pkg3 w1)
        (free)
        (operating))

(:goal (and
          (delivered pkg1)
          (delivered pkg2)
          (delivered pkg3)
          )))
```

## C.4 Simulação

### C.4.1 Arquivo Domínio

```
(define (domain quaddeliver)
(:requirements :typing)
(:types quad coke waypoint fuellvl - object)

(:predicates (connected ?wp1 ?wp2 - waypoint)
             (grounded ?q - quad)
             (airborne ?q - quad)
             (finished ?q - quad)
             (squaredone ?q - quad)
             (quadat ?q - quad ?wp - waypoint)
             (visited ?wp - waypoint)
             (cokeat ?c - coke ?wp - waypoint)
             (holdingcoke ?c - coke)
             (keyat ?wp ?from ?to - waypoint)
             (quadfuel ?fuel - fuellvl)
             (fuelmapping ?lowerlvl ?higherlvl - fuellvl)
             (minimumfuel ?minfuel - fuellvl)
             (maximumfuel ?maxfuel - fuellvl)
             (stationat ?wp - waypoint))


(:action move-to
 :parameters (?q - quad ?wp1 ?wp2 - waypoint ?fuel1 ?fuel2 ?fuel3 - fuellvl)
 :precondition (and (quadat ?q ?wp1) (connected ?wp1 ?wp2)(airborne ?q)
                    (quadfuel ?fuel3)(fuelmapping ?fuel1 ?fuel2)
                    (fuelmapping ?fuel2 ?fuel3))
 :effect (and (not (quadat ?q ?wp1)) (quadat ?q ?wp2)
          (probabilistic 0.6 (and(quadfuel ?fuel2)(not(quadfuel ?fuel3)))
                             0.4 (and(quadfuel ?fuel1)(not(quadfuel ?fuel3)))
                             )))

(:action refuel
 :parameters (?q - quad ?wp1 - waypoint ?fuel1 ?fuel2 - fuellvl)
```

```
 :precondition (and(quadat ?q ?wp1)(stationat ?wp1)(quadfuel ?fuel1)
               (maximumfuel ?fuel2)(grounded ?q))
 :effect (and (quadfuel ?fuel2)(not(quadfuel ?fuel1))))

(:action takeoff
 :parameters (?q - quad)
 :precondition (grounded ?q)
 :effect (and (not (grounded ?q)) (airborne ?q)))

(:action land
 :parameters (?q - quad)
 :precondition (airborne ?q)
 :effect (and (not (airborne ?q)) (finished ?q) (grounded ?q)))

(:action flysquare
 :parameters (?q - quad)
 :precondition (and (airborne ?q))
 :effect (squaredone ?q))

(:action pickup
 :parameters (?q - quad ?c - coke ?wayp - waypoint)
 :precondition (and (quadat ?q ?wayp) (cokeat ?c ?wayp))
 :effect (holdingcoke ?c))

(:action drop
 :parameters (?q - quad ?wayp ?wayp2 - waypoint ?c - coke)
 :precondition (and (quadat ?q ?wayp)(cokeat ?c ?wayp2)(holdingcoke ?c))
 :effect (and (cokeat ?c ?wayp) (not(cokeat ?c ?wayp2))
              (not (holdingcoke ?c))))

(:action opendoor
        :parameters (?q - quad ?wayp ?from ?to - waypoint)
        :precondition (and (quadat ?q ?wayp)(keyat ?wayp ?from ?to))
        :effect (and(connected ?from ?to)(connected ?to ?from)))
)
)
```

### C.4.2 Arquivo Problema - Experimento 5 Playpen

```
(define (problem rosplanquadtask)
(:domain quaddeliver)
(:objects
    coke1 coke2 coke3 - coke
    q1 - quad
    wp0 wp1 wp2 wp3 wp4 wp10 - waypoint
    fl1 fl2 fl3 fl4 fl5 fl6 fl7 fl8 fl9 fl10 - fuellvl
)
(:init
    (grounded q1)
    (cokeat coke2 wp2)
    (cokeat coke1 wp1)
    (connected wp0 wp1)
    (connected wp1 wp0)
    (connected wp2 wp3)
    (connected wp4 wp1)
    (connected wp3 wp4)
    (connected wp1 wp10)
    (connected wp10 wp1)
    (keyat wp10 wp1 wp2)
```

```
    (quadat q1 wp1)
    (squaredone q1)
    (fuelmapping fl1 fl2)
    (fuelmapping fl2 fl3)
    (fuelmapping fl3 fl4)
    (fuelmapping fl4 fl5)
    (fuelmapping fl5 fl6)
    (fuelmapping fl6 fl7)
    (fuelmapping fl7 fl8)
    (fuelmapping fl8 fl9)
    (fuelmapping fl9 fl10)
    (quadfuel fl10)
    (maximumfuel fl10)
    (minimumfuel fl1)
    (stationat wp2)
)
(:goal (and
    (cokeat coke1 wp2)
    (cokeat coke2 wp3))
))
```

### C.4.3 Arquivo Problema - Experimento 5 Domínio de Entregas

```
(define (problem rosplanquadtask)
(:domain quaddeliver)
(:objects
    coke1 coke2 coke3 - coke
    q1 - quad
    wp1 wp2 wp3 wp4 wp5 wp6 wp7 wp8 wp9 wp10
    wp11 wp12 wp13 wp14 wp15 wp16 wp17 wp18 wp19 wp20 - waypoint
    fl1 fl2 fl3 fl4 fl5 fl6 fl7 fl8 fl9 fl10 - fuellvl
)
(:init
    (grounded q1)
    (cokeat coke2 wp2)
    (cokeat coke1 wp1)
    (connected wp1 wp2)
    (connected wp1 wp5)
    (connected wp2 wp1)
    (connected wp2 wp3)
    (connected wp2 wp6)
    (connected wp3 wp2)
    (connected wp3 wp4)
    (connected wp3 wp7)
    (connected wp4 wp3)
    (connected wp4 wp8)
    (connected wp5 wp1)
    (connected wp5 wp6)
    (connected wp5 wp9)
    (connected wp6 wp2)
    (connected wp6 wp5)
    (connected wp6 wp7)
    (connected wp6 wp10)
    (connected wp7 wp3)
    (connected wp7 wp6)
    (connected wp7 wp8)
    (connected wp7 wp11)
    (connected wp8 wp4)
    (connected wp8 wp7)
```

```
    (connected wp8 wp12)
    (connected wp9 wp5)
    (connected wp9 wp10)
    (connected wp9 wp13)
    (connected wp10 wp6)
    (connected wp10 wp9)
    (connected wp10 wp11)
    (connected wp10 wp14)
    (connected wp11 wp7)
    (connected wp11 wp10)
    (connected wp11 wp12)
    (connected wp11 wp15)
    (connected wp12 wp8)
    (connected wp12 wp11)
    (connected wp12 wp16)
    (connected wp13 wp9)
    (connected wp13 wp14)
    (connected wp13 wp17)
    (connected wp14 wp10)
    (connected wp14 wp13)
    (connected wp14 wp15)
    (connected wp14 wp18)
    (connected wp15 wp11)
    (connected wp15 wp14)
    (connected wp15 wp16)
    (connected wp15 wp19)
    (connected wp16 wp12)
    (connected wp16 wp15)
    (connected wp16 wp20)
    (connected wp17 wp13)
    (connected wp17 wp18)
    (connected wp18 wp17)
    (connected wp18 wp14)
    (connected wp18 wp19)
    (connected wp19 wp18)
    (connected wp19 wp15)
    (connected wp19 wp20)
    (connected wp20 wp19)
    (connected wp20 wp16)
    (keyat wp10 wp1 wp2)
    (quadat q1 wp1)
    (squaredone q1)
    (fuelmapping fl1 fl2)
    (fuelmapping fl2 fl3)
    (fuelmapping fl3 fl4)
    (fuelmapping fl4 fl5)
    (fuelmapping fl5 fl6)
    (fuelmapping fl6 fl7)
    (fuelmapping fl7 fl8)
    (fuelmapping fl8 fl9)
    (fuelmapping fl9 fl10)
    (quadfuel fl10)
    (maximumfuel fl10)
    (minimumfuel fl1)
    (stationat wp5)
    (stationat wp19)
)
(:goal (and
    (quadat q1 wp2(grounded q1)
    (cokeat coke1 wp2))))
```

# APÊNDICE D – Grafo de Computação do Sistema Aplicado

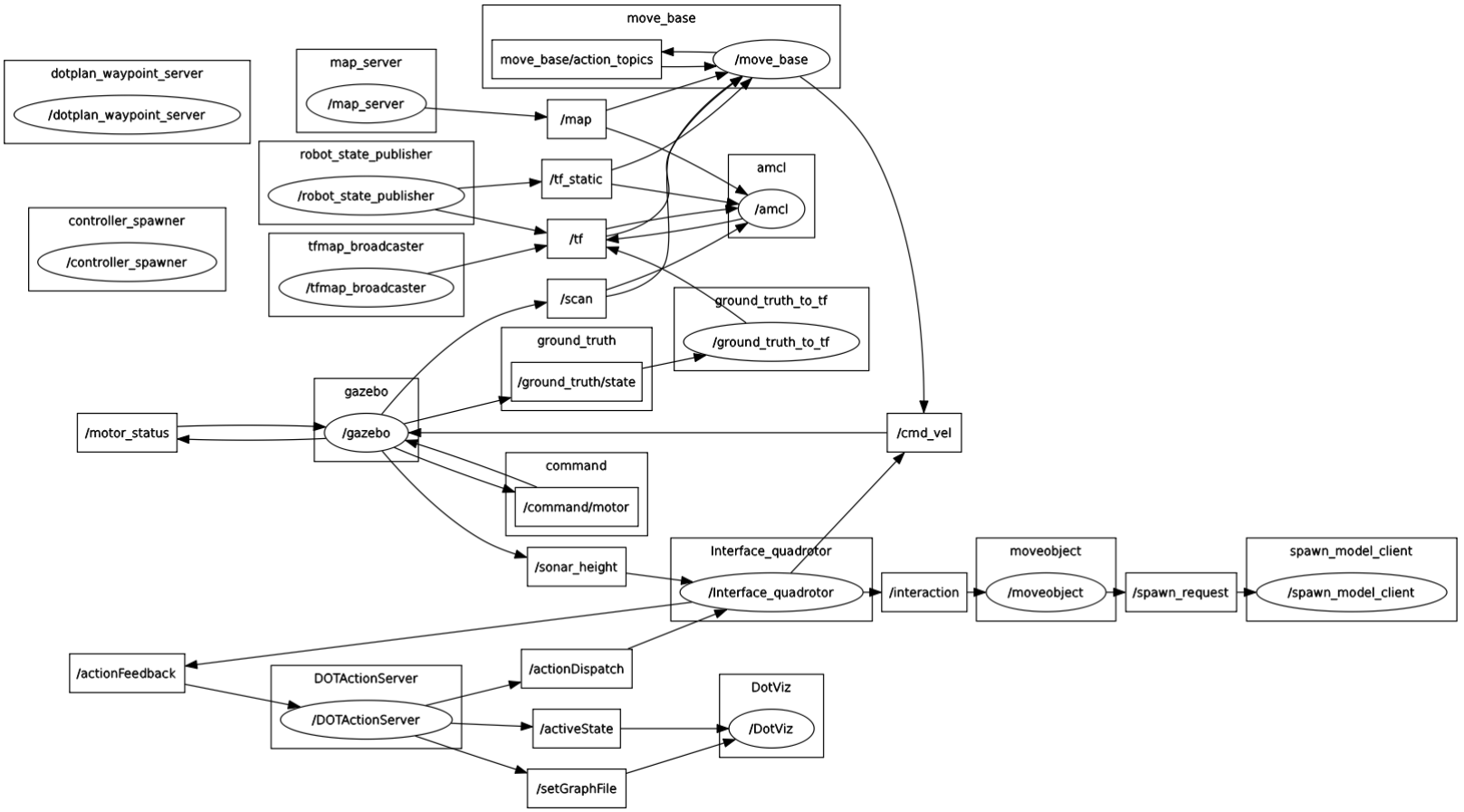



Figura 28 – Grafo de Computação do Sistema VANT implementado no Gazebo

# Anexos

# UAV MISSION PLANNING AND EXECUTION VIA NON-DETERMINISTIC AI PLANNING ON ROS

Vinícius Veloso Eleuterio Nogueira*, Luiz Edival de Souza†

* *Universidade Federal de Itajubá, av. B P S, 1303 - Pinheirinho, Itajubá- MG, Brazil*

Emails: nogueira.vinicius@hotmail.com, edival@unifei.edu.br

**Abstract**— Unmanned aerial vehicles (UAVs) have been attracting civilian attention by its increasing number of applications and decreasing cost. The vast majority of current UAV tasks still have a low degree of autonomy, which can lead to lack of efficiency, safety, practicality and viability. This paper proposes using Probabilistic AI Planning to improve autonomy and robustness in UAV applications. Our main contribution is a framework that integrates Prob-PRP planner on ROS systems and enables the visualization, execution and monitoring of its plans using Finite State Machine (FSM). On this article, we describe our framework and apply it for high-level mission control on a simulated autonomous UAV.

**Keywords**— Automated Planning, Artificial Intelligence, Robotics Sytems, Unmanned Aerial Vehicles.

**Resumo**— Os veículos aéreos não tripulados (UAV) têm atraído a atenção das pessoas civis pelo seu crescente número de aplicações e pela redução de custos. A grande maioria das tarefas atuais do UAV ainda possui um baixo grau de autonomia, o que pode levar à falta de eficiência, segurança, viabilidade e praticidade. Este artigo propõe o uso de planejamento de AI probabilístico para melhorar a autonomia e robustez nas aplicações de UAV. Nossa principal contribuição é uma estrutura que integra o planejador Prob-PRP em sistemas ROS e permite a visualização, execução e monitoramento de seus planos usando a Máquina de Estados Finitos (FSM). Neste artigo, descrevemos o nosso quadro e aplicamos o controle de missão de alto nível em um UAV autônomo simulado.

**Palavras-chave**— Planejamento Automático, Inteligência Artificial, Sistemas Robóticos, Veículos Aéreos Não Tripulados

## 1 INTRODUCTION

The development of unmanned aerial vehicles (UAVs) began in applications involving military operations (Turner et al., 2012) during the Vietnam War or Cold War (Watts et al., 2012) and became popular for civilian use with the rise of multicopters. These vehicles gained relevance mainly due to their mechanical simplicity, low cost, user-friendliness, and safety (Cutler, 2012).

Nowadays, UAVs can be found in a wide number of civil applications. These applications include: aerial photography; agriculture; mapping; search and rescue; disaster management; monitoring; border patrol; inspection; environmental control; remote sensing and wireless relay. On top of that, there are still a large of number of studies that aims to increase its capabilities. Future applications may require higher level of intelligence from the system as they tend to become more complex. Most of the studies involving these types of vehicles still focus on perception and control strategies(Bernardini et al., 2014). Moreover, high levels of autonomy will require the system to act deliberately.

A promising field within Artificial Inteligence that could provide intelligence to the system is *Automated Planning*, also called AI Planning. Automated Planning is the reasoning side of acting (Ghallab et al., 2004) that relies on a model of the agent and its environment to produce a plan that should guide an agent to reach its objectives. This model can be domain specific planning (Mersheeva and Friedrich, 2015)(Ghamry et al., 2016)(Lee and Morrison, 2015) (Albore et al., 2015) or domain-independent (Bernardini et al., 2014) (Munoz-Morera et al., 2015). Nowadays, domain specific solutions outperform domain independent plans, but they need much modeling effort which requires much specialized knowledge and limits deliberative capabilities to specific areas. On the other hand, domain independent strategies attract more research interest, and had received much contribution in the past few years. These contributions envolve algorithms that either increase expressivity or decrease the complexity of domain independent approach.

UAVs are naturally unstable systems that have high sensitivity to disturbances, such as wind and rain. In addition, the limitation of power sources makes the energy consumption of these vehicles crucial to meet their objectives (Sydney et al., 2013). Therefore, considering the environmental uncertainty during task planning show to be necessary in this type of vehicles, once that deterministic approaches do not avoid dead-ends. We selected Prob-PRP(Camacho et al., 2015) non-deterministic solver that aims to increase the plan probability of success. It is a goal-oriented and domain-independent solver that uses PPDDL model as input.

The combination of Automated Planning and aerial vehicles was already proposed in (Cantoni et al., 2011), where it was studied classical planning on a fire fighter scenario and simulated it on X-Plane®. Another similar research was made

by (Bernardini et al., 2014) in aerial surveillance planning for searching-and-tracking operations. Automated Planning is also applied commonly to other types of robotic applications (Quintero et al., 2011) (Crosby et al., 2017). As automated planning became the standard tool for domain-independent problem solving and Robotic Operating System (ROS) for robotics applications they were first merged in (Cashmore et al., 2015) resulting in ROSPlan. In its first version, this package implemented a classical planner and was used in an application using autonomous underwater vehicles. Later, it has been extended to support contingency plans, using Contingent-FF (Sanelli et al., 2017). While contingency plans are able to handle eventualities, monotonicity of knowledge prevents construction of cycles which can be required in some domains.

ROS (Quigley et al., 2009) is popular modular framework that allows the integration of libraries intended for robotic systems. It provides hardware abstraction and communication to interface different robotic application. The implementation in this platform is convenient once it already possess many state-of-the-art algorithm from scientific collaborative work.

Our approach was implemented in ROS using both Python and C++. Our presented library is based on ROSPlan (Cashmore et al., 2015) and conditional planning in (Sanelli et al., 2017). Our framework contributes to the community by incorporating PRP and allowing its plan execution, visualization and supervision on ROS systems. Its purpose is to designate high-level tasks that guides an UAV to its goal according to the outcomes of its actions using a FSM controller. We call this framework *DOTPlan*.

This paper first provides some preliminaries on Finite State Machines and their relation to our work in section 2. On section 3, we do a overview on the system, describing its architecture, features and purpose. Following, we apply Prob-PRP on UAV delivery domain using our framework and discuss the results.Following, we present the integration with simulated application and compare our work with others in section 5. Future work and conclusion are discussed in section 6.

## 2 Preliminaries

Finite State Machines (FSMs) are a computation model represented by finite number of states and transitions. A deterministic finite automaton (DFA) is defined in (Hopcroft et al., 2006) as a tuple $(Q, \Sigma, \delta, q0, F)$, where Q is a finite set of states; $\Sigma$ is a finite set of symbols, called alphabet; $\delta : Q$ x $\Sigma \rightarrow Q$ is a deterministic transition function; $q0 \in Q$ is the initial state; $F \subseteq Q$ is a set of goal states. It is a abstract machine that receives a set of inputs alphabet symbols, called string, and process them sequentially according to the transition functions. After processing all inputs, if it ends on one of the goal states, the machine accepts the string, if not it rejects it. A FSM can be graphically represented as graph in which the nodes are equivalent to the states and the transition to the edges. Figure 1 shows a DFA with $(\{q0, q1, q2, q3, q4\}, \{a, b, c\}, \delta, q0, \{q3\})$, with $\delta$ being described by the graph edges. This automaton will accept any string that have three *a*s and no *c*s.

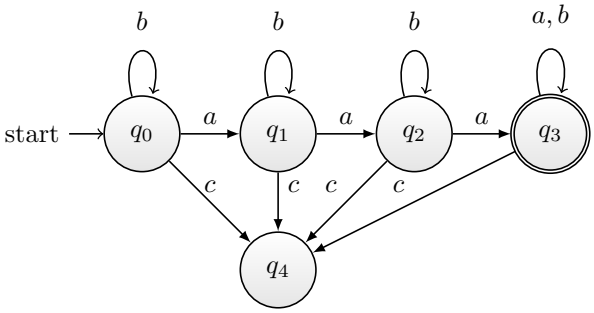



Figure 1: Deterministic Finite Automata(DFA)

### *2.1 From Non-Deterministic Problem to FSM*

A non-deterministic planning problem is described as $P = (S, I, A, T, G)$ in (Kissmann and Edelkamp, 2009), where $S$ is a set of states; $I$ is the initial state; $A(s)$ is a set of applicable actions; $G$ is the goal state; $T$ is a non-deterministic transition relation $T(s, a, s)$ that represents the probability of occurring the outcome, $o$, when applying action $a$ in the state $s$. The non-deterministic planning problem is determined when the probabilistic planner parses $I$,$G$ and other domain knowledge provided in PPDDL format and infers $A$, $T$, $S$. The planner task is to calculate a policy $\Pi(s) \rightarrow a$ that maps a state $s$ into an action $a$ that will guide $s$ to $G$.

A FSM $(Q, \Sigma, \delta, q0, F)$ can be produced from the policy, if we let $Q$ be a set of states, where a state $s$ is characterized by a set of facts; $\Sigma$ be a set of all possible determinized actions, *ao*; $\delta$ the transition function that relates $s$ x $ao \rightarrow s$; q0 the initial state, F the state that contains all goal facts. FSM can be created by simulating the policy execution from the initial state to the goal state for every possible outcome. The PRP planner authors (Sardina and D'Ippolito, 2015) made their source code available in which contained a script used for their validation. This script generates a FSM from policies in FOND domains and outputs it in a DOT format. The FSM generation method used by this script is very similar to the algorithm described in (Iocchi et al., 2016) to convert polices into conditional plans.

## 3 System Overview

The aim of the proposed paper is to introduce a new framework which aims to allow execution of probabilistic plans with ROS and to apply it to a common UAV delivery domain problem, represented in figure 2. This figure shows that the system is composed by modules: planning, supervision, execution and visualization. In this section we describe the framework for executing probabilistic plans via Finite State Machine representation.

### *3.1 AI Planning*

The execution of the planning module is fundamental once it gives the information that the other modules needs to operate. The Planning System is off-line and should output a FSM in Graphviz's Dot format. In our application, the Prob-PRP was applied as the third-party planner resulting in a policy that was converted using algorithm 1. Additional information is also collected during FSM generation, this data should help the system operator during system supervision. FSMs are often cited in literature as a simple and compact controller. This controller is often used for sequential action coordination. While (Sardina and D'Ippolito, 2015) use a FSM as validation tool, we apply it as a plan controller.

### *3.2 DotViz*

DotViz is the visualization module of DOTPlan, it displays the conditional plan graph, highlights and centers the execution's current state. It facilitates the analysis and comprehension of the current plan. This module auto arranges of the graph's nodes and edge in a way which is both compact and clear.

DOT format is a widely used output format for graph representation. While it supports some fancy tools, the basic usage requires only minimal data overhead. Figure 3a shows the DOT of the graph displayed in figure 3b. In figure 3a, it can be noted that the nodes position can be left unspecified. It happens because softwares that reads this format, uses Graphviz library. Graphviz's coordinates assignments enables a clean auto generated graph visualization, what justify its usage in a wide number of application. In this library the problem of arranging is solved as a cost minimization task. It is already present in core ROS package as rqt_graph and rq_tf_tree, wrapped by Pydot. Also, there is a wide variety of libraries available in many different programming languages to automatic create a graph in DOT format, which its increases flexibility, convenience and support. In fact, all diagrams in this paper (except figure 1) were designed in DOT.

Xdot (Fonseca, 2017) is an open source software written in Python that displays DOT files using Graphviz library on a GTK3 user interface. The appliance of Xdot becomes convenient, since it uses Graphviz in a lower level and has some interesting built-in functions like: node_search and animate_to_position. This software was modified to be incorporated on ROS using gobject thread synchronization and called XDotViz. This software is able to show the FSM, auto highlight states and animate through the graph based on information exchange in either topics or services. Figure 4 represents the XDotViz interaction within the framework. Xdot natively does not save any information, therefore it has to arrange the whole graph every time the dot file is opened. In that matter, a new function was implemented to save and load nodes positions in a intermediate file to speed up the program start-up of pre opened files. This feature is useful once that a complex plan can result in a large graph, it could take sometime to auto arrange nodes. XDotViz main purpose is to monitoring the execution of the FSM. The system was implemented in a way that it works independently from XDotViz.

### *3.3 DotActionServer*

DotActionServer is the core module for execution the computed plan. It coordinates the agent high-level actions to the desired goal. It works by passing action information to low-level control and receives its feedback to compute the next action. This information includes possible outcomes, action name, action parameters. The application system should then execute the action and return which was the resulting outcome.

By looking at figure 3a, it can be observed that a DOT form has a simple and efficient structured format. Based on this, the DOT form was defined as the standard input format, not only for monitoring but also for execution. In order to obtain its DOT file the system calls the planner and the converter, which is responsible to transform the plan/policy into a FSM in DOT. This DOT file is referenced to XDotViz and parsed into the plan executor. In this framework, we adopt the following convention to express a FSM in DOT: the node id must be a unique integer; the node name can be arbitrary, with exception to brackets and dashes; *action[outcomes effects]* must be the format of the edge name; # is a reserved character to represents either blank or none.

The action dispatching algorithm parses the DOT file, and extract the following relations: $(s \rightarrow (state\ name))$, $(s \rightarrow a)$, $(s \rightarrow O(s))$, and $(s \text{ x } o \rightarrow s)$. The action outcomes may contain many shared effects, the solution was to express it to its exclusive effects that differs it from the other outcomes. This operation makes the graph

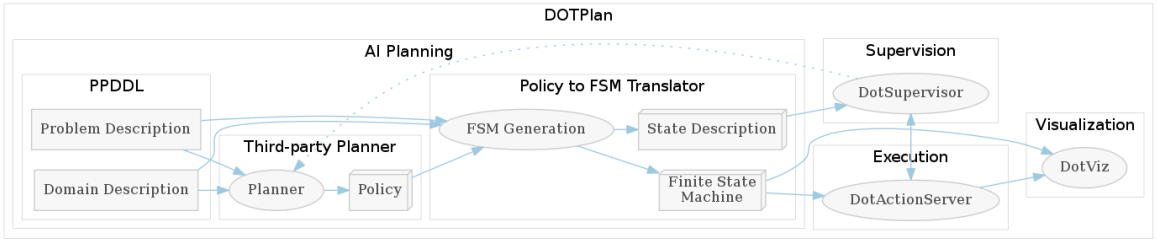
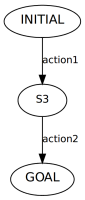



Figure 2: DotPlan System Overview

```
digraph  {
node [label="\N"];
1 [label=INITIAL];
3 [label=S3];
1 -> 3 [key=0,
label="action1"];
4 [label=GOAL];
3 -> 4 [key=0,
label="action2"];
}
```

(a) DOT




(b) Graph

Figure 3: DOT and its corresponding graph

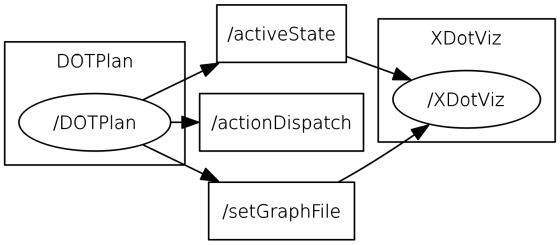



Figure 4: XDotViz interaction with the framework

visualization clearer and eases the outcome verification. The execution of this controller is very straightforward, the controller should keep track of the FSM current state and dispatch the action specified in its transitions. When the action server is activated, it dispatches the initial action and dispatches the consecutive action based on their feedback.

Algorithm **??** shows the procedure to dispatch actions based on their feedback. A plan fails if: the action did not succeed; the action server goes out of synchrony with the action executor; the outcome is not present in the FSM. If a plan doesn't fail it will be able to retrieve the necessary information from the relations until it achieves the goal state.

### *3.4 DotSupervisor*

This framework also includes an interactive graphical user interface, that is used for information output and plan interaction. Some of the information displayed is gathered during the FSM generation. This data includes the active state, action server status, current action, facts $s$ and key facts

**procedure ActionFB**
**Require:** $actState, curOut, curAct, Succeed$
  **if** $Succeed = false$ **then**
    $planStat \leftarrow failed$
    ***return***
  **if** checkFB$(actState, Out, Act)$=false **then**
    $planStat \leftarrow failed$
    ***return***
  $actState \leftarrow$ progress$(curOut)$
  **if** stateIsGoal$(actState)$ = true **then**
    $planStat \leftarrow concluded$
    ***return***
  $curAct \leftarrow$ getStateAction$(actState)$
  $stateOuts \leftarrow$ getStateOutcomes$(actState)$
  publishAction$(actState, curAct, stateOuts)$
  publishActiveState(getStateName$(s)$)

$s'$ of the active state. This interface also enables the user to: (re)plan from PPDDL files, to reload the action server DOT file, send action feedback.

## 4 Prob-PRP in UAV domain

Planner for Relevant Policies (PRP) is the state-of-the-art FOND planner. Prob-PRP is an extended version of PRP to solve probabilistic problems. An interesting characteristic of Prob-PRP is that it guarantees to avoid avoidable dead-end state. This solver relays on finding strong cyclic solution for the problem. A strong cyclic solution is defined as a solution that reaches the goal after an infinite number of actions. One issue is that this planner does not guarantee a solution in the presence of unavoidable dead ends.

A fragment of our domain is displayed in figure 5. The UAV domain is a discrete probabilistic problem that models the task of using a single aerial vehicle for package delivery. This domain considers that routes between locations of interest are discretized with equidistant waypoints. It contains uncertainty in fuel consumption while moving between waypoints. This uncertainty could be related either to weather conditions, necessaries maneuvers or discretization error. The actions $land$ and $take-off$ also consumes fuel but does considers uncertainty on its usage. In this model, the fuel resource is discretized into fuel levels. For

```
(:domain(quaddelivery)
(:types pack way fuellvl - object)
(:predicates
  (deliverat ?pkg - pack ?wp - way)
  (stationat ?wp -way)
  (pkgat ?pkg - pack ?wp - way)
  (quadat ?wp - way)
  (inquad ?pkg - pack)
  (delivered ?pkg - pack)
  (connected ?wp1 ?wp2 - way)
  (quadfuel ?fuel - fuellvl)
  (fuelmap ?lowlvl ?highlvl fuellvl)
  (maximumfuel ?maxfuel - fuellvl)
  (onair)
   (grounded))
(:action move-to
  :parameters (?wp1 ?wp2 - way
               ?f1 ?f2 ?f3 - fuellvl)
  :precondition (and(quadat ?wp1)
    (connected ?wp1 ?wp2)
    (quadfuel ?f3)
    (fuelmap ?f1 ?f2)
    (fuelmap ?f2 ?f3)
    (onair))
  :effect (and (not (quadat ?wp1))
    (quadat ?wp2)
    (not(quadfuel ?f3))
    (probabilistic
                0.6 (quadfuel ?f2)
                0.4 (quadfuel ?f1))))
(:action land (?f1 ?f2 - fuellvl)
(:action takeoff(?f1 ?f2 - fuellvl))
(:action load(?pkg - pack
              ?wp - way))
(:action deliver(?pkg - pack
                 ?wp ?wp2 - way))
(:action refuel(?wp - way
                ?f1 ?f2-fuellvl))
```

Figure 5: Fragment of the UAV Delivery Domain in PPDDL

this problem we suppose that the system holds full observability.

After experimenting with multiple different problem file and domain variations, we could analyze the behavior of Prob-PRP in our domain. When we let the UAV refuel infinite number of times some aspects can be observed: the planner always avoided dead ends when possible; the most probable outcome tends to have the shortest path; when the action results in a least probable outcome; it forces a cycle; it cycles the plan even when the worst set of outcomes would lead to the goal. Bringing these characteristics to the analyzed domain several points can be remarked. For the sake of state reduction and problem simplification, it forces cycles in its plan, in its plan, when is possible to make the choice infinite times, called *strong cycles*. Another problem is when it refuels and takes actions to waste its fuel and return to a state which it visited earlier. In other words, the execution leads back to a state that it previously visited to try to get another outcome. This cycle may increase the plan length and may not be needed to avoid dead ends.

In this planner, the length of the plan is shorten for UAV domain by limiting its number of refuels. By doing that, we take of the guarantee of reaching the goal and permit an increase in the number of states for the FSM. We implicitly impose a heuristic, limiting the mission length by giving it limited fuel. Table 1 shows how the number of allowed refuels affects a domain with 12 waypoints, 10 fuel levels, 2 stations and 3 packages. In this table we can extract the relation between the number of allowed refuel and the probability of success. In that relation the planner percentage of success increases and converges to a value, while the planning time tend to increase in a exponentially. In non-deterministic planning the trade off between success rate and plan length is inevitable, once conservative behavior do not take unnecessary risks. In the experiments, even when the 10% longest plan lengths of the infinite refuels scenario are discarded, the average still 112 actions. This shows that the plan is taking unnecessary actions.

| R | PT (s) | S | P | L |
|---|---|---|---|---|
| 2 | 0.44 | 0 | 0 | 0 |
| 3 | 9.84 | 80 | 0.448 | 18.42 |
| 4 | 35.34 | 163 | 0.911 | 34.25 |
| 5 | 53.26 | 256 | 0.966 | 38.98 |
| 6 | 59.9 | 258 | 0.968 | 43.85 |
| 7 | 102.7 | 388 | 0.973 | 45.02 |
| $\infty$ | 0.82 | 150 | 1 | 123.04 |

Table 1: Policy caracteristics by number of allowed refuels in Prob-PRP. R represents the maximum refuels allowed; PT is the planning time, S is the number of the states in the FSM, P is the probability of reaching the goal state; L is the average number of actions to achieve the objective

This approach breaks the cycles and makes a tree that is more likely to converge in the sub-goals and refueling. Even without *strong cycles*, Prob-PRP avoided avoidable dead-ends. Despite that this works and give good solutions, this method breaks Prob-PRP proposed method of assuring plan maximum probabilities using *strong cycles*. This leads to no guarantees in terms of performance. It is also evident that any other approach than the strong cyclic takes much more planning time. This solver seems very practical in problems where the most problem outcome is very likely, in fault recovery and in assuring action desired

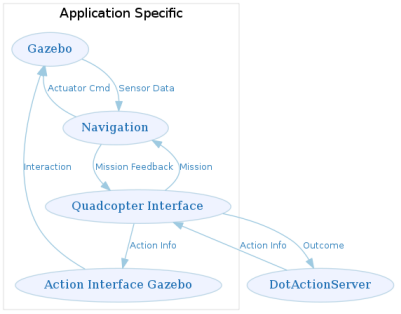



Figure 6: Interfacing DOTPlan's execution with UAV application.

effect(e.g. fire fighting, search and monitoring). Like other model based approaches, the effectiveness of executing this plan will depend on how accurate is the model domain. This planner is a promissing algorithm that is still under development, it is capable of avoiding avoidable dead-ends while being a fast algorithm that gives compact representations.

## 5 RESULTS

To validate our approach, the proposed framework was implemented in a simulated UAV system using the architecture represented in figure 6. This figures shows how the application interfaces the execution module inside ROS using topics. In this simulation the agent, environment and problem was described in PDDL file format and the actions of the quadrotor are deliberated by the planner and dispatched by the FSM executor. The simulator used for this application was Gazebo, which is the default 3D simulator in ROS. Our platform was the hector quadrotor which is natively compatible with Gazebo, which is equipped by a laser scan, sonar altitude sensor and a RGB camera. In order to keep the rest of the system as simple as possible a 2D navigation stack was applied for moving and other actions interact directly inside gazebo. The resulting system is displayed in figure 7, it exposes the aerial vehicle executing a plan to achieve its objectives.

We compared our framework to the main ROS package for planning and execution, ROSPlan, in its two main variants using POPF and Contingent-FF. This comparison is represented in table 2. All of them use domain-independent planners that accept PDDL structure and present some sort recovery behavior to deal with uncertainty. While ROSPlan-POPF uses online replanning when it encounters uncertainty, ROSPlan-Cont.FF and DOTPlan rely on following a conditional plan. Both replanning and conditional plan need estimation of the current state to guide its plan. But while non-deterministic planners do offline reasoning for different probable sce-

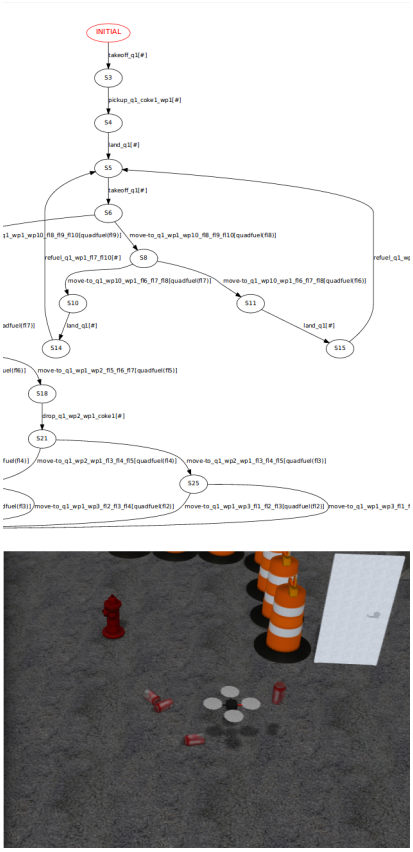



Figure 7: Plan execution in a simulated environment using Gazebo.

narios to avoid dead-ends, replanning does not, which may lead the agent into a dead-end state or repeated interaction in presence of uncertainty (Little et al., 2007). ROSPlan-POPF represents its plan in a list format, which is enough to represent a classical plan since there is only one possible sequence of actions. It is useful to keep track of the agent progress. A problem of visualizing classical plans is that its behavior can be unpredictable if the replanning is triggered. ROSPlan-Cont.FF uses Petri Net Plans (PNP) by interfacing a PNP tool outside ROS called PNPJarp. Petri Net Plans is the application of Petri Nets to represent and execute plans. Its theory also supports concurrent actions and multi-agent systems. We tried to implement our policy in their visualization program, but we had some difficulties with the automatic layout of the states. Also, their PNP generation library doesn't seem to support loops in their policy to PNP translation.

ROSPlan-Cont.FF has a limited perception exogenous events on the environment after it does its observation. This happens because contingent planning assumes that once a fact is unknown it can only changed by an agent action. This is the principle of contingent planning which allows the uncertainty to be compiled in the initial state and provide monotonicity of knowledge. In this way, this system does not support repeated sensing on

| | ROSPlan (Popf) | ROSPlan (Cont) | DOTPlan (Prob) |
|---|---|---|---|
| PDDL Support | X | X | X |
| Recovery Behavior | X | X | X |
| Off-line Recovery | | X | X |
| Visual Representation | X | X | X |
| Plan Supervision | X | | X |
| Knowledge Management | X | | |
| Repeated Sensing | X | | X |
| Partial Observability | | X | |

Table 2: Comparison between ROSPlan using POPF, ROSPlan using Contingent-FF with PNP and DOTPlan using Prob-PRP

the same fact. This doesn't make these technique worst, just not adequate for this kind of problem. In fact, it is able to solve partial observability problems which Prob-PRP and POPF are not. ROSPlan-POPF keeps track of all knowledge using its knowledge base. In this package, sensing actions can occurs infinitely often and influences on how the system estimates its state and update its known facts. Once that the system facts changes, replanning should be triggered. DOTPlan must observe the effects of its actions after every non-deterministic action to choose which branch of its plan that it will dispatch.

## 6 Conclusion and Future Work

In this paper we presented a framework for executing probabilistic plans with ROS Systems and applied it into the UAV delivery domain. The proposed package is based in the representation of plans with finite state machines (FSM) in DOT format. This system utilizes Prob-PRP as its default probabilist planner but it also supports PRP and PO-PRP, once that they all have the same output format. We applied this solver into the UAV domain and made several considerations about its applicability in UAV systems. Finally, it was compared to other standard planning tools inside ROS, highlighting the possibility of solving probabilistic problems by strong cycles.

While simulated domain might presented only the modeled outcomes, this might not happen in a real-world application. Thereat, it is relevant to incorporate replanning into the framework. This would require a tool for problem generation. Since we already keep track of the state facts, defining the initial state of the new plan shouldn't be a big issue. In the present time, this framework doesn't support concurrent actions. This is a known limitation of doing execution using FSMs. We believe that this could be overcome with an action feedback buffer, similar to Determinist Pushdown Automaton. In the future, this framework could also be merged into other planning frameworks.

## References


Albore, A., Peyrard, N., Sabbadin, R. and Teichteil-Königsbuch, F. (2015). An online replanning approach for crop fields mapping with autonomous uavs., *ICAPS*, pp. 259–267.

Bernardini, S., Fox, M. and Long, D. (2014). Planning the behaviour of low-cost quadcopters for surveillance missions., *ICAPS*, Portsmouth, NH, pp. 445–453.

Camacho, A., Muise, C., Ganeshen, A. and McIlraith, S. A. (2015). From fond to probabilistic planning: Guiding search for quality policies, *Workshop on Heuristic Search and Domain Independent Planning, ICAPS*.

Cantoni, L. F., Campos, M. F. and Chaimowicz, L. (2011). Investigacao da linguagem pddl no planejamento de missoes para robËos aéreos, *X SBAI*.

Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carrera, A., Palomeras, N., Hurtós, N. and Carreras, M. (2015). Rosplan: Planning in the robot operating system., *ICAPS*, pp. 333–341.

Crosby, M., Petrick, R. P., Toscano, C., Dias, R. C., Rovida, F. and Krüger, V. (2017). Integrating mission, logistics, and task planning for skills-based robot control in industrial kitting applications, *Ceur Workshop Proceedings*, Vol. 1782.

Cutler, M. J. (2012). *Design and control of an autonomous variable-pitch quadrotor helicopter*, PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Fonseca, J. (2017). Xdot.py: An interactive viewer for graphs written in graphviz's dot language., https://github.com/jrfonseca/xdot.py.

Ghallab, M., Nau, D. and Traverso, P. (2004). *Automated Planning: theory and practice*, Elsevier.

Ghamry, K. A., Kamel, M. A. and Zhang, Y. (2016). Cooperative forest monitoring and fire detection using a team of uavs-ugvs, *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, IEEE, pp. 1206–1211.

Hopcroft, J. E., Motwani, R. and Ullman, J. D. (2006). Automata theory, languages, and computation, *International Edition* **24**.

Iocchi, L., Jeanpierre, L., Lazaro, M. T. and Mouaddib, A.-I. (2016). A practical framework for robust decision-theoretic planning and execution for service robots., *ICAPS*, pp. 486–494.

Kissmann, P. and Edelkamp, S. (2009). Solving fully-observable non-deterministic planning problems via translation into a general game, *KI 2009: Advances in Artificial Intelligence* pp. 1–8.

Lee, S. and Morrison, J. R. (2015). Decision support scheduling for maritime search and rescue planning with a system of uavs and fuel service stations, *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, IEEE, pp. 1168–1177.

Little, I., Thiebaux, S. et al. (2007). Probabilistic planning vs. replanning, *ICAPS Workshop on IPC: Past, Present and Future*.

Mersheeva, V. and Friedrich, G. (2015). Multi-uav monitoring with priorities and limited energy resources., *ICAPS*, pp. 347–356.

Munoz-Morera, J., Maza, I., Fernandez-Aguera, C. J., Caballero, F. and Ollero, A. (2015). Assembly planning for the construction of structures with multiple uas equipped with robotic arms, *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, IEEE, pp. 1049–1058.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R. and Ng, A. Y. (2009). Ros: an open-source robot operating system, *ICRA workshop on open source software*, Vol. 3, Kobe, p. 5.

Quintero, E., García-Olaya, Á., Borrajo, D. and Fernández, F. (2011). Control of autonomous mobile robots with automated planning, *Journal of Physical Agents*, Citeseer.

Sanelli, V., Cashmore, M., Magazzeni, D. and Iocchi, L. (2017). Short-term human-robot interaction through conditional planning and execution, *ICAPS*.

Sardina, S. and D'Ippolito, N. (2015). Towards fully observable non-deterministic planning as assumption-based automatic synthesis., *IJCAI*, pp. 3200–3206.

Sydney, N., Smyth, B. and Paley, D. A. (2013). Dynamic control of autonomous quadrotor flight in an estimated wind field, *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, IEEE, pp. 3609–3616.

Turner, D., Lucieer, A. and Watson, C. (2012). An automated technique for generating georectified mosaics from ultra-high resolution unmanned aerial vehicle (uav) imagery, based on structure from motion (sfm) point clouds, *Remote Sensing* **4**(5): 1392–1410.

Watts, A. C., Ambrosia, V. G. and Hinkley, E. A. (2012). Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use, *Remote Sensing* **4**(6): 1671–1692.