UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Fernando Rodrigues Santos

AVALIAÇÃO DO USO DE AGENTES NO DESENVOLVIMENTO DE APLICAÇÕES COM VEÍCULOS AÉREOS NÃO-TRIPULADOS

Florianópolis (SC)
2015

Fernando Rodrigues Santos

AVALIAÇÃO DO USO DE AGENTES NO DESENVOLVIMENTO DE APLICAÇÕES COM VEÍCULOS AÉREOS NÃO-TRIPULADOS

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas para obtenção do grau de "Mestre em Engenharia de Automação e Sistemas".

Orientador: Prof. Dr. Jomi Fred Hübner, UFSC.

Co-orientador: Prof. Dr. Leandro Buss Becker, UFSC.

Florianópolis (SC)
2015

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Santos, Fernando Rodrigues
AVALIAÇÃO DO USO DE AGENTES NO DESENVOLVIMENTO DE APLICAÇÕES COM VEÍCULOS AÉREOS NÃO-TRIPULADOS / Fernando Rodrigues Santos ; orientador, Jomi Fred Hübner ; coorientador, Leandro Buss Becker. - Florianópolis, SC, 2015.
89 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Inclui referências

1. Engenharia de Automação e Sistemas. 2. VANT. 3. Agente BDI. 4. Sistema Embarcado. I. Hübner, Jomi Fred. II. Becker, Leandro Buss. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. IV. Título.

Fernando Rodrigues Santos

AVALIAÇÃO DO USO DE AGENTES NO DESENVOLVIMENTO DE APLICAÇÕES COM VEÍCULOS AÉREOS NÃO-TRIPULADOS

Esta dissertação foi julgada aprovada para a obtenção do grau de "Mestre em Engenharia de Automação e Sistemas" e aceita em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Florianópolis (SC), 22 de Maio de 2015.

Prof. Dr. Rômulo Silva de Oliveira
Coordenador do Programa de Pós-Graduação em Engenharia de Automação e Sistemas

---

Prof. Dr. Jomi Fred Hübner - UFSC
Orientador

Prof. Dr. Leandro Buss Becker - UFSC
Co-orientador

Banca Examinadora:

Prof. Dr. Jomi Fred Hübner - UFSC
Presidente

Prof. Dr. Felipe Rech Meneguzzi - PUC/RS

Prof. Dr. Guilherme Vianna Raffo - UFMG

Prof. Dr. Jean-Marie Farines - UFSC

Dedico este trabalho a Deus como uma oferta e a todos os meus familiares e amigos.

Agradecimentos

Agradeço, primeiramente, a Deus por tudo que Ele é, por tudo aquilo que Ele tem feito e ainda vai fazer, por sua fidelidade, misericória, graça e cuidado para comigo. Por me conduzir até aqui durante todo esse período do mestrado. A Ele seja dada a honra e a glória, para sempre.

Aos meus pais, Eduardo e Neuza, por todo amor e apoio dado a mim em cada etapa e nova jornada em busca de um sonho maior. Muito obrigado por acreditarem em mim.

Ao meu irmão e meus familiares, Lopes e Rodrigues, pelo apoio e incentivo. E também aos amigos que mesmo de longe, estiveram torcendo por mim.

Aos meus orientadores, os professores Jomi e Leandro, pela paciência e auxílio no desenvolvimento desse projeto. Também a todos das equipes do Projeto ProVant e do Grupo SMA, foi muito bom fazer parte desses times e trabalhar com vocês foi uma experiência fantástica.

Agradeço, também, aos meus grandes amigos e colegas do mestrado, que sempre me auxiliaram, ajudaram e participaram de vários momentos.

Aos amigos que tive o privilégio de conhecer em Florianópolis e compartilhar de momentos únicos, que de alguma forma marcaram e comtribuíram de maneira especial em todo esse período em Floripa.

Ao PPGEAS, CNPQ e ao CTC pelo apoio, suporte e infraestrutura.

E por fim, a todos que direta ou indiretamente contribuíram para o desenvolvimento desse trabalho.

Muito obrigado!

"O futuro pertence àqueles que acreditam na beleza de seus sonhos."
– Elleanor Roosevelt

Resumo

O uso de agentes em aplicações com Veículos Aéreos Não-Tripulados (VANTs) tem sido explorado nos últimos anos, principalmente como alternativa para dotar o veículo de autonomia na realização de suas missões. Este trabalho tem como objetivo desenvolver um modelo de comportamento autônomo para um VANT com o uso de um agente com arquitetura BDI, explorando sua capacidade de reagir rapidamente a mudanças em seu ambiente e de ter objetivos de longo prazo a serem cumpridos até se finalizar uma dada missão.

O trabalho também busca avaliar a implementação do agente na plataforma de sistema embarcado de um VANT real, a aeronave do projeto ProVant, além de apresentar uma análise e comparação do sistema proposto, baseado em lógica de predicados, com uma abordagem usual empregando uma programação imperativa, organizada em uma sequencia de comandos e ações.

Palavras-chave: VANT, Agente BDI, Sistema Embarcado.

Abstract

The use of agents in applications with Unmanned Aerial Vehicles (UAVs) has been explored in recent years, mainly as an alternative to provide autonomy to the vehicle in carrying out their missions. This study aims to develop and evaluate a model for a UAV with the use of an agent with BDI architecture, exploring its ability to react quickly to changes in the environment and while still having goals to be accomplished even finish a given mission. The proposed model was implemented and embedded in a real UAV system and then compared against a usual approach employing an imperative programming.

Keywords: UAV, Agent BDI, Embedded System.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

As aplicações utilizando Veículo Aéreo Não-Tripulado (VANT) são inúmeras, indo desde monitoramento de regiões por grupos de VANTs até filmagens de eventos, passando por entrega de pizzas, como abordado em [FAHLSTROM; GLEASON, 2012].

Na maioria destas aplicações um operador humano é responsável pela execução remota destes equipamentos. Em algumas circunstâncias, porém, é necessário, ou pelo menos desejável, que o veículo possua autonomia para realizar algumas tarefas. Por exemplo, o controle do VANT por rádio frequência limita o alcance das missões. Um veículo com maior autonomia pode tanto dispensar o operador de atividades repetitivas e, portanto, sujeitas a erro, quanto possuir mecanismos automáticos mais precisos e otimizados. Embora existam várias interpretações para o termo autonomia, neste momento iremos considerar autônomo um veículo que não precise da operação constante de um humano e para o qual possam ser atribuídos objetivos. Neste sentido, o veículo tem autonomia para escolher um plano de ação para atingir um objetivo, dado por um operador humano, porém deve se comprometer com tal objetivo, ou seja, ele não pode ignorá-lo. Do ponto de vista do operador humano, ele passa a controlar o VANT por meio da delegação de objetivos, tais como passar por determinados lugares (waypoints) ou encontrar uma pessoa perdida em uma floresta.

Além disso, o processo de desenvolvimento de um VANT apresenta diversos desafios [GONÇALVES, 2014], e o desenvolvimento de uma implementação da autonomia da aeronave deve levar em consideração as restrições de hardware do sistema embarcado do VANT, as quais limitam o uso de técnicas mais complexas que exijam grande capacidade de processamento da plataforma.

O tema autonomia é estudado há vários anos na área de Sistemas Multi-Agentes (SMA), tendo produzido teorias, arquiteturas de software e mesmo linguagens de programação especificamente voltadas para o desenvolvimento deste tipo de sistema computacional, chamado

de agente autônomo, segundo [WOOLDRIDGE, 2002]. Uma das vantagens deste tipo de linguagem é seu alto nível de abstração, com primitivas como objetivos, planos e ações que permitem ao desenvolvedor definir claramente o comportamento e a autonomia do agente. Em especial, se destacam na área as linguagens baseadas na arquitetura BDI (*Belief, Desire, Intention*) que facilitam o desenvolvimento de agentes que apresentam tanto a capacidade de reagir rapidamente a mudanças em seu ambiente quanto possuir objetivos de longo prazo [RAO; GEORGEFF, 1995] [BRATMAN, 1987].

Assim, a proposta deste trabalho consiste em analisar a viabilidade do uso de um agente com arquitetura BDI no contexto do sistema computacional embarcado de um VANT, dadas as restrições do hardware do dispositivo. Para isso, foi desenvolvido um agente para tomada de decisão da aeronave durante o cumprimento de sua missão, tendo como aplicação o planejamento de rota. Também foi realizada uma análise comparativa entre o sistema proposto e uma abordagem usual com programação imperativa.

1.1. Motivação

Diante da possibilidade de explorar o cenário de aplicações de VANTs aliado ao uso de agentes BDI embarcados nesses veículos, surge então algumas questões que motivaram a escolha do tema de pesquisa e o problema a ser abordado, sendo eles:

Problema de Pesquisa: consiste em investigar a viabilidade do uso de uma abordagem com agentes BDI para modelagem e desenvolvimento de um comportamento autônomo de um VANT, bem como a integração deste sistema na plataforma embarcada da aeronave.

Pergunta de Pesquisa: no projeto de comportamentos autônomos de VANTs, espera-se que essas aeronaves tenham como características reagir às alterações do ambiente e ter um comprometimento com os objetivos da missão. Através de um modelo usando a abordagem com arquitetura de agentes BDI é possível ter esse sistema embarcado em um VANT e qual a sua vantagem perante às técnicas de implementação usualmente utilizadas no desenvolvimentos destes veículos?

1.1.1. Veículos Aéreos Não-Tripulado (VANT)

A definição de Veículos Aéreos Não-Tripulado - VANT, do inglês *Unmanned Aerial Vehicle* - UAV, apresentada em [CORRÊA, 2008], diz que o VANT é um veículo aéreo motorizado que não transporta um

operador humano, utiliza as forças aerodinâmicas para sua sustentação aérea, pode voar de maneira autônoma ou ser operado através de um controle remoto, pode ser recuperável ou descartável e pode transportar uma carga útil letal ou não-letal.

Segundo descreve [CHAVES, 2013], os VANTs podem ser classificados conforme algumas de suas características, sendo assim chamados de VANTs: de asas fixas (fixed-wing) ou asas rotativas (rotary-wing); mais leves ou mais pesados que o ar; com decolagem vertical (VTOL - Vertical take-off and landing) ou em curto espaço (STOL - Short take-off and landing); com controle on-board (autônomo) ou off-board (controlado remotamente).

Outro tipo de classificação de VANTs, quanto a sua funcionalidade, conforme exposto em [HAMA, 2012], 8descreve que os VANTs podem ser:

- **Falso Alvo:** Fornecem cobertura aérea e funcionam como alvos-iscas com o objetivo de enganar unidades hostis;
- **Reconhecimento:** Fornecem dados geográficos, informações de campo ou da infraestrutura de instalações;
- **Monitoramento:** Englobam o contexto de monitoramento de áreas de difícil acesso ou ambientes hostis;
- **Combate:** Fornecem apoio bélico às missões de alto-risco às vidas dos pilotos, seja como força de frente ou cobertura e apoio;
- **Transporte:** Possuem a tarefa de transportar cargas em geral;
- **Resgate:** Utilizados em missões de resgate tático, em lugares de difícil acesso;
- **Pesquisa e Desenvolvimento:** Utilizados para o desenvolvimento da tecnologia em VANTs através de testes de software e integrações com outras plataformas e sistemas;
- **Civil e Comercial:** Especificados e concebidos com o objetivo de atender aos propósitos civis, comerciais ou de entretenimento, como é o caso de hobbie em aeromodelismo e apresentações de marketing.

1.1.2. Projeto ProVant

O presente trabalho faz parte do projeto ProVant[1], que é um projeto iniciado em 2012 no Departamento de Automação e Sistemas (DAS) da Universidade Federal de Santa Catarina (UFSC) e possui parceria com a Universidade Federal de Minas Gerais (UFMG). Ele propõe o desenvolvimento de um VANT de baixo custo, com caráter geral, vol-

[1] `http://provant.das.ufsc.br`

tado para aplicações civis. O desenvolvimento prevê não somente sua modelagem e controle, mas também sua concepção e construção, o que é um diferencial diante da grande maioria dos projetos com VANTs.

O ProVant é um projeto aberto, que tem por objetivo documentar e apresentar todos os detalhes de projeto do VANT desenvolvido. As informações do projeto eletromecânico e também do desenvolvimento de algoritmos são disponibilizados tanto no website do projeto, quanto por meio de publicações em eventos relacionados à área.

O projeto ProVant consiste de uma proposta diferenciada em termos de desenho do projeto e de custo reduzido em relação ao que existe no mercado. Ao invés de se trabalhar com veículos aéreos em escala reduzida nas configurações comumente encontradas no mercado, como por exemplo, aeronaves de asas fixas (aviões) ou aeronaves de asas rotativas como os helicópteros com quatro rotores (usualmente chamados quadrirotores), o VANT deste projeto possui apenas dois rotores, cada um associado a um servomotor para realizar a rotação longitudinal (perpendicular ao eixo) dos rotores, constituindo assim uma aeronave na configuração Tiltrotor[2], conforme ilustrado na figura 1.1 do protótipo desenvolvido.

Na concepção da aeronave do projeto, optou-se por desenvolver um VANT bi rotor VTOL, de configuração tiltrotor, visando explorar as pesquisas com esse modelo, o qual tem por característica a realização de decolagens e pousos na vertical.

Uma das aplicações do ProVant é permitir que o VANT realize voos autônomos de acordo com trajetórias pré-definidas. Estas trajetórias possuem um conjunto de pontos (*waypoints*) que são fornecidas por uma estação base, sendo definidas na fase de planejamento das missões. A comunicação entre o VANT e a estação base é realizada por meio de uma conexão sem fio do tipo *wireless*, como pode ser visto na figura 1.2.

Através da interface de software na estação base, o usuário pode realizar algumas funções de comando para o VANT, tais como: definição dos objetivos da missão e os pontos que o VANT deve passar durante a sua execução; iniciar e abortar as missões; monitorar as informações de vôo e de execução da missão; além de realizar testes de verificação do funcionamento dos sensores e atuadores da aeronave.

A realização do vôo autônomo ocorre a partir de uma missão previamente carregada no sistema embarcado do VANT e com o co-

[2]O tiltrotor é uma aeronave caracterizada pela movimentação por meio da atuação de seus rotores. Seu deslocamento ocorre devido à inclinação (do inglês *tilt*) de seus rotores.

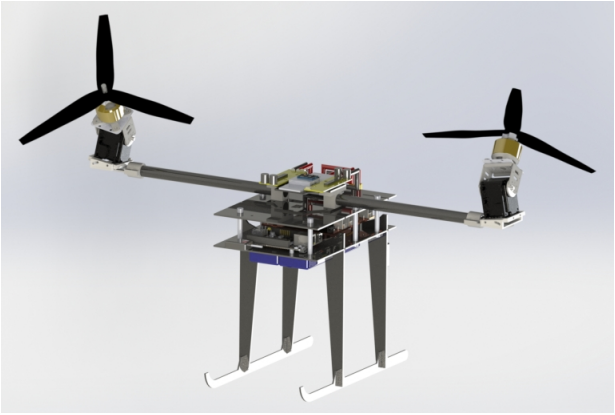

Figura 1.1: Modelo da estrutura mecânica do VANT

mando de autorização do usuário da estação-base para o início da sua operação.

O projeto ProVant possui em seu escopo três dissertações concluídas, seis em andamento e uma tese em desenvolvimento. São trabalhos que mostram a potencialidade de casos e estudos que essa área tem a oferecer para o desenvolvimento de pesquisas e aplicações.

1.2. Objetivos

1.2.1. Objetivo Geral

A proposta deste trabalho consiste em analisar a possibilidade do uso de um agente com arquitetura BDI em uma aplicação embarcada em um VANT, para aplicações de tomada de decisão da aeronave, bem como suas vantagens e limitações em um hardware de restrito.

1.2.2. Objetivos Específicos

Para se alcançar o objetivo geral proposto, pretende-se atingir também os seguintes objetivos específicos:

- Investigar a viabilidade do uso de uma abordagem com uma arquitetura de agentes BDI em VANTs autônomos.
- Desenvolver um modelo de comportamento de VANTs autônomos em uma linguagem com arquitetura de agentes BDI.

Figura 1.2: Cenário da aplicação proposto para o ProVant.

- Integrar o modelo proposto com o hardware do sistema embarcado do VANT.
- Realizar testes com modelo proposto por meio de simulação com o hardware.
- Estabelecer critérios de avaliação para o modelo proposto.
- Comparar o modelo proposto com uma técnica usual de progração imperativa.

1.3. Organização

O trabalho está organizado da seguinte forma: No Capítulo 2 tem-se a fundamentação teórica com a apresentação dos temas tratados nessa dissertação, sendo abordado sobre agentes, sistemas multiagentes, arquitetura BDI, arquitetura customizada do agente, AgentSpeak e a linguagem Jason. Também é apresentado a estrutura de simulação Hardware In the Loop (HIL).

O Capítulo 3 discute os trabalhos relacionados ao tema de pesquisa da presente dissertação, com uma visão de como é tratado o assunto de agentes e VANTs.

O Capítulo 4 apresenta a proposta desenvolvida neste trabalho, descreve o planejamento de rota, o sistema proposto, o cenário de teste, a modelagem do problema em ambas as abordagens utilizadas, a arquitetura do sistema e o desenvolvimento da aplicação.

O capítulo 5 traz os resultados obtidos e as análises dos testes realizados, através de uma análise qualitativa e quantitativa do problema.

Por fim, o Capítulo 6 apresenta as conclusões do presente trabalho e descreve algumas propostas de trabalhos futuros.

2 Fundamentação Teórica

Nesta seção são apresentados fundamentos teóricos relacionados ao tema dessa dissertação que serviram de base para o desenvolvimento deste projeto.

2.1. Agentes e Sistemas Multi-Agentes

No desenvolvimento de software e tecnologia de programação, tem-se os programas funcionais. Esse programas executam a partir de uma conjunto de entradas, realizam operações, produzem uma saída e param. Eles são assim chamados porque, matematicamente, podemos considerá-los como funções **f: I -> O**, de algum domínio **I** de entradas possíveis para alguma gama de possíveis saídas **O**. Existem outros tipos de sistemas, dentre eles os chamados reativos, que devem manter por um longo período interação contínua com o ambiente; eles não simplesmente calculam alguma função de uma entrada e, em seguida, encerram. Como exemplos deste tipo de programa, incluem os sistemas operacionais, os sistemas de controle de processo, sistemas bancários on-line, servidores web, e afins. Uma classe ainda mais complexa de sistemas é um subconjunto de sistemas reativos chamado de agentes [BORDINI et al., 2007].

O conceito de agentes possui algumas definições, dentre elas, tem-se a definida por [WOOLDRIDGE, 2002] como um sistema computacional capaz de executar, de forma autônoma, ações em um ambiente em que se situa, com o propósito de realizar os objetivos a ele delegados. De acordo com [RUSSELL; NORVING, 2009], um agente é um programa de computador capaz de atuar de forma autônoma, perceber seu ambiente, persistir ao longo de um período de tempo prolongado, se adaptar à mudanças, além de criar e buscar objetivos. Ele age de modo a alcançar o melhor resultado ou, quando há incerteza, o melhor resultado esperado.

Segundo [BORDINI et al., 2007], agente é um sistema reativo que

apresenta algum grau de autonomia no sentido de delegar alguma tarefa a ele e o próprio sistema determina a melhor forma de realizar esta tarefa. Esses sistemas são assim chamados por serem pensados como sendo ativos, produtores intencionais de ações: eles são enviados para seu ambiente para alcançar os seus objetivos, prosseguir ativamente essas metas, descobrir por si a melhor forma de atingir esses objetivos, ao invés de dar detalhes em baixo nível de como fazer.

Os agentes são sistemas que estão situados em algum ambiente, ou seja, os agentes são capazes de perceber o seu ambiente (através de sensores), e têm um repertório de ações possíveis que eles podem executar (através de atuadores), a fim de modificar o ambiente em que estão inseridos. O ambiente que um agente ocupa pode ser físico (no caso de robôs no mundo físico) ou um ambiente de software (no caso de um agente de software em um sistema operacional de computador ou rede), conforme [BORDINI et al., 2007].

Segundo [WOOLDRIDGE; JENNINGS, 1995], os agentes devem ter algumas propriedades, tais como: autonomia, proatividade, reatividade e habilidade social:

- A autonomia se refere a capacidade do agente de operar de forma independente, a fim de alcançar as metas que lhe foram delegadas. Um agente autônomo toma suas próprias decisões sobre a forma de alcançar seus objetivos, além disso, suas decisões e suas ações estão sob seu próprio controle, e não são movidos por outros.
- A proatividade significa ser capaz de apresentar um comportamento dirigido a objetivos. Quando um agente recebe um objetivo particular, então espera-se que ele tente alcançar esse objetivo através de suas ações.
- A reatividade diz respeito a ser sensível às alterações do ambiente, responder a estímulos de forma reflexiva. Quando o agente tem consciência de que um plano deu errado, ele deve responder e escolher um curso de ação alternativo.
- E a habilidade social considera a capacidade dos agentes de cooperar e coordenar suas atividades com outros agentes, a fim de alcançar os objetivos. O agente pode se comunicar não apenas em termos de troca de dados, mas em nível de conhecimento, ou seja, comunicando as suas crenças, metas e planos para outros agentes.

Os agentes podem ser classificados em três classes, de acordo com o comportamento utilizado para atingir determinado objetivo, conforme abordado em [BERGENTI et al., 2004]:

- reativos: os agentes reativos simplesmente reagem de forma rápida à percepção de eventos e estímulos que ocorrem no ambiente, sem executar um raciocínio complexo;
- cognitivos: os agentes cognitivos contém um modelo simbólico do seu ambiente externo e de outros agentes, sobre o qual desenvolve planos e toma as suas decisões;
- híbridos: os agentes híbridos combinam as vantagens dos modelos reativos e cognitivos, sendo compostos de módulos de natureza reativa ou cognitiva. Assim, o agente pode se adaptar às aplicações usando ambos os módulos para atender de forma eficaz as necessidades da aplicação em determinado instante.

Existem sistemas nos quais diversos agentes, sendo eles iguais ou não, dividem o mesmo ambiente e interagem para solucionar os problemas. Esses sistemas são chamados de sistemas multi-agentes (SMA), onde os agentes agem de forma coordenada, interagem e cooperam uns com os outros para atingir tanto os seus objetivos individuais quanto o objetivo geral do sistema. A figura 2.1 ilustra uma visão geral de um sistema multi-agente. Na parte inferior, tem-se o ambiente compartilhado que os agentes ocupam; cada agente tem uma "esfera de influência" neste ambiente, isto é, uma parte do ambiente que eles são capazes de controlar de forma total ou parcialmente. Um agente pode ter a capacidade única de controlar parte do seu ambiente, mas de modo geral, tem-se a possibilidade de que as esferas de influência se sobreponham e o ambiente seja controlado em conjunto. Dessa forma, para conseguir o resultado que ele deseja no ambiente, o agente terá que levar em consideração os outros agentes. Acima do ambiente, tem-se os próprios agentes, que estão relacionados organizacionalmente (hierarquias, por exemplo). Além disso, os agentes podem ter algum conhecimento dos outros agentes do sistema.

2.1.1. Arquitetura BDI

Dentre os diversos modelos de arquiteturas propostas para o desenvolvimento de agentes, esse trabalho utiliza o modelo *Belief-Desire-Intention* (BDI). As origens dessa arquitetura foram inspiradas e baseadas no modelo filosófico proposto por [BRATMAN, 1987] para explicar o raciocínio prático humano. A ideia central do modelo BDI é de que os programas de computador tenham algum tipo de estado mental, ou seja, análogos computacionais de crenças (beliefs), desejos (desires) e intenções (intention), e assim decidam como agir.

No modelo BDI, conforme [BORDINI et al., 2007], temos:

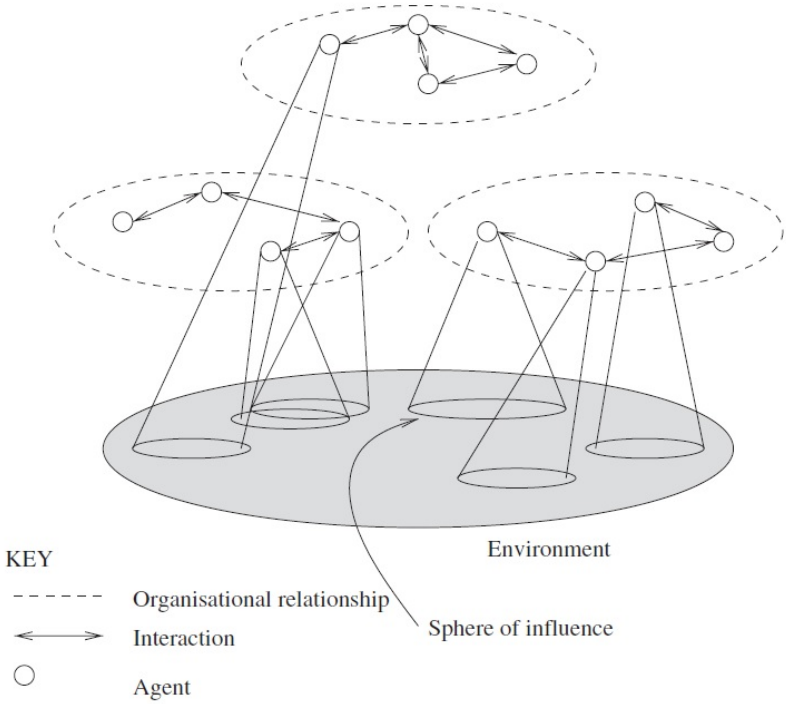

Figura 2.1: Estrutura típica de um sistema multi-agente [BORDINI et al., 2007].

- **Beliefs (crenças):** são as informações que o agente possui sobre o ambiente, outros agentes ou sobre o próprio agente. Essas informações são adquiridas a partir das percepções que o agente tem do ambiente, e essas podem estar imprecisas ou desatualizadas.
- **Desires (desejos):** são estados do ambiente que o agente gostaria de atingir, é um influenciador potencial das suas ações, ou seja, ter um desejo não significa necessariamente que o agente vai tentar atingi-lo. É com base nesses desejos que o agente seleciona suas intenções.
- **Intentions (intenções):** são estados do ambiente em prol dos quais o agente decidiu trabalhar, podendo ser os objetivos que são delegados ao agente ou resultar de quando ele considera alguma opção dentre as metas existentes.

Para ir das crenças, desejos e intenções para às ações, o agente faz uso de um modelo particular de tomada de decisão subjacente ao modelo BDI que é conhecido como raciocínio prático. Este é um raciocínio direcionado por ações, sendo o processo de descobrir o que fazer [BORDINI et al., 2007].

O processo de raciocínio prático consiste em duas atividades: a deliberação, que decide quais desejos devem ser promovidos à intenção; e a análise de meios-fins, também conhecida como planejamento, que decide como essas intenções serão atingidas. Durante o processo de deliberação, o agente seleciona dentre as suas intenções qual ele irá tentar atingir, ou seja, qual o seu objetivo. A seleção dos desejos na fase de deliberação é feita com base nas crenças, nos desejos e nas intenções atuais do agente. No processo de análise de meios-fins, o agente define um de seus planos de ação (meio) que deve ser executado para alcançar uma determinada intenção (fim) escolhida. A figura 2.2 ilustra como é o modelo de raciocínio do agente BDI.

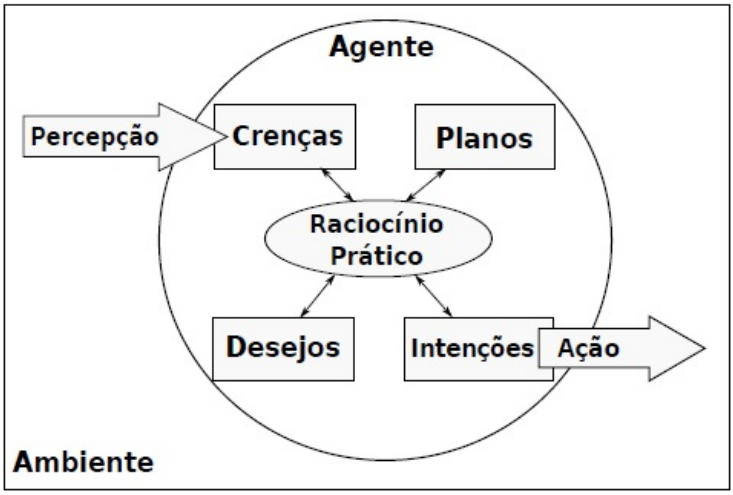

Figura 2.2: Modelo de raciocínio prático utilizado pelos agentes BDI. Adaptado de [WOOLDRIDGE, 2002].

2.1.2. AgentSpeak e Jason

A linguagem AgentSpeak, originalmente introduzida por [RAO, 1996], é uma linguagem baseada em programação lógica para arqui-

tetura de agentes BDI. Os principais construtores da linguagem são: beliefs (crenças), goals (objetivos/metas) e plans (planos). A arquitetura de software da AgentSpeak possui quatro componentes, sendo eles: base de crenças, a biblioteca de planos, o conjunto de eventos e o conjunto de intenções.

Em AgentSpeak, as crenças representam as informações que um agente possui sobre o ambiente ou outros agentes; os objetivos/meta representam estados de tarefas que o agente quer fazer; e os planos são receitas de como agir, que representam o conhecimento do agente.

Um dos componentes dessa arquitetura do agente é uma base de crença, que é atualizada em conformidade com as percepções do ambiente. Outro componente importante são os objetivos do agente, que são alcançados pela execução dos planos. Os agentes são concebidos para estar permanentemente em execução, reagindo a alguma forma de evento, através da execução dos planos. Os planos são cursos de ação que os agentes se comprometem a executar, de forma a lidar com tais eventos. As ações, por sua vez mudam o ambiente do agente, de tal forma que se espera que os objetivos sejam alcançados. Existe um componente da arquitetura do agente que é responsável por efetuar as alterações no ambiente com base nas escolhas sobre o curso de ações realizadas pela interpretação do programa do agente. Um agente está constantemente percebendo o ambiente, raciocinando sobre como agir de forma a atingir os seus objetivos, e então atuar de forma a mudar o ambiente. O raciocínio prático do agente, em um agente AgentSpeak, é feito de acordo com os planos que o agente tem em sua biblioteca de planos. Inicialmente, esta biblioteca é formada pelos planos que o programador escreve como um programa AgentSpeak [BORDINI et al., 2007].

Em AgentSpeak, existem dois tipos de metas: metas de realização e metas de teste. As metas de realização podem ser processuais ou declarativas. Na meta processual, o nome da meta é semelhante ao nome do método/procedimento em linguagem de programação tradicional e pode ser bastante útil, por exemplo, para agrupar as ações que são frequentemente utilizadas no programa. Já a meta declarativa é vista como uma representação simbólica de um estado de coisas que o agente tem que alcançar. As metas do teste são usadas simplesmente para recuperar a informação que está disponível na base de crenças do agente [BORDINI et al., 2007].

A linguagem Jason é baseada na arquitetura BDI, em lógica de predicados e é uma linguagem interpretada que extende AgentSpeak. O interpretador Jason executa um programa de agente que opera por

meio de um ciclo de raciocínio, conforme ilustrado na figura 2.3.

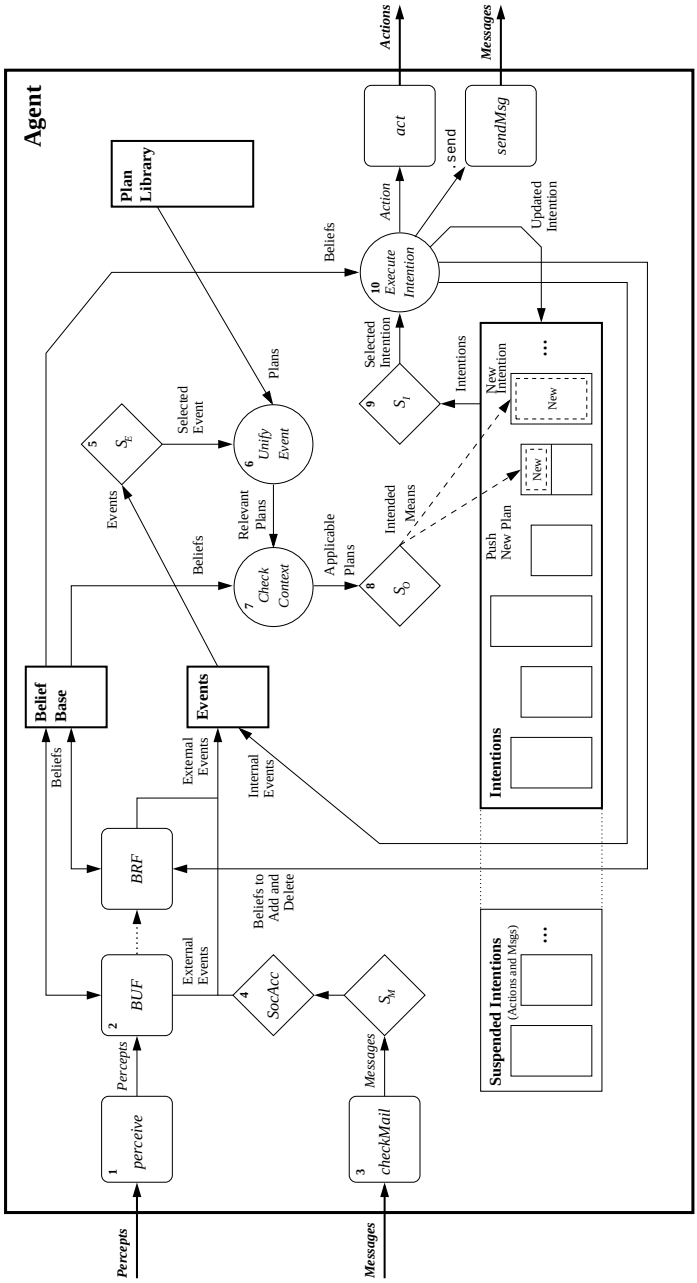

Figura 2.3: O ciclo de raciocínio Jason [BORDINI et al., 2007].

No ciclo de raciocínio do Jason, apresentado por [BORDINI et al., 2007], as 10 etapas são:

Passo 1 - Percepção do ambiente: o agente sente o ambiente, a fim de atualizar suas crenças sobre o estado do ambiente.

Passo 2 - Atualização da base de crenças: uma vez que a lista de percepções é obtida, a base de crença precisa ser atualizada para refletir as mudanças percebidas no ambiente.

Passo 3 - Recepção de comunicação de outros agentes: o interpretador verifica as mensagens que poderiam ter sido entregues para a "caixa de entrada" do agente.

Passo 4 - Seleção de Mensagens "socialmente aceitáveis": antes das mensagens serem processadas, elas passam por um processo de seleção para determinar se elas podem ser aceitas pelo agente ou não.

Passo 5 - Seleção de um evento: os agentes funcionam através tratamento contínuo de eventos, que representam cada mudança percebida no ambiente ou mudanças nos objetivos próprios do agente. Em cada ciclo de raciocínio, apenas um caso pendente será tratado.

Passo 6 - Recuperação de todos os planos relevantes: a partir de um evento selecionado, o agente encontra um plano que irá permitir que atue de modo a lidar com esse evento. Assim, busca-se na biblioteca de planos todos aqueles que são relevantes para o evento determinado.

Passo 7 - Determinação dos planos aplicáveis: selecionar, a partir dos planos relevantes, todos aqueles que são atualmente aplicáveis, ou seja, um plano que dado o conhecimento do agente e suas crenças atuais, parece ter uma chance no sucesso.

Passo 8 - Seleção um plano aplicável: dado o conhecimento do agente, expresso por sua biblioteca de planos e as informações atualizadas sobre o mundo expresso por sua base de crenças, determina-se qual dentre os planos atuais no conjunto de plano aplicáveis que é a alternativa adequada para lidar com o evento selecionado.

Passo 9 - Seleção de uma intenção para posterior execução: escolher uma intenção particular entre aquelas atualmente prontas para execução.

Passo 10 - Execução de uma etapa de uma intenção: a execução de uma ação que o agente se compromete a realizar sobre o ambiente.

2.1.3. Arquitetura Customizada do Agente

A linguagem Jason permite customizar a arquitetura de software do agente para modificar a interação com o ambiente e com outros agentes do sistema. O interpretador AgentSpeak é apenas o módulo

de raciocínio dentro de uma "arquitetura geral do agente" que faz a interface com o mundo exterior, visto que a arquitetura BDI é apenas a parte cognitiva do agente.

A arquitetura geral do agente fornece percepção (modela os sensores do agente), atuação (modela a atuação do agente), e como o agente recebe mensagens de outros agentes. Assim, esses aspectos também podem ser personalizados para cada agente individualmente, conforme [BORDINI et al., 2007]. A implementação padrão dessa arquitetura está em uma classe chamada `AgArch` que pode ser estendida. Essa classe é uma "ponte" para a infraestrutura multi-agente subjacente que pode ser personalizada sem se preocupar com a aplicação concreta da percepção e da comunicação feita pela infraestrutura.

Na classe `AgArch`, temos os métodos `perceive()` e `act()` que estão associados a percepção e a atuação do agente, respectivamente:

- `perceive()`: este método retorna uma lista de literais que representam o que acaba de ser percebido pelo agente. A implementação padrão simplesmente recebe as percepções enviadas pelo ambiente e as retorna. Ele pode ser personalizado, por exemplo, para mudar capacidades de percepção do agente conforme a necessidade da aplicação.
- `act(ActionExec action, List<ActionExec> feedback)`: quando a execução de alguma intenção contém uma ação, este método é chamado para realizar a ação no ambiente, onde o primeiro argumento contém detalhes dessa ação. Em ambientes simulados, significa simplesmente enviar uma mensagem de solicitação para a implementação do ambiente para simular a execução da ação. Em agentes que possuem atuadores concretos, este método deve interagir com o hardware para executar a ação. Enquanto a ação está sendo realizada, a intenção que fez a escolha dessa execução está suspensa. Quando a ação for concluída (com sucesso ou não), esta deve ser adicionada na lista de `feedback`, para que a intenção possa ser retomada ou não.

2.2. Técnicas de Avaliação

A avaliação do sistema desenvolvido para aplicações embarcadas pode ser realizada diretamente no protótipo real de um robô móvel ou através de simulação. O ambiente simulado permite a realização de testes em um ambiente seguro, sem causar riscos a integridade física do modelo do robô móvel em questão. A seguir, tem-se a descrição da técnica de emulação HIL utilizada neste trabalho para avaliação.

2.2.1. Hardware In the Loop (HIL)

No desenvolvimento de aplicações de sistemas embarcados, para se obter um ambiente seguro e confiável para a concepção e teste do sistema, é possível utilizar a técnica de emulação Hardware In the Loop (HIL). O HIL tem por objetivo a união entre uma plataforma embarcada real e um ambiente de simulação, que visa abstrair os riscos físicos e busca aplicar o sistema computacional diretamente na estrutura final da aplicação, neste caso, o VANT [LOUALL et al., 2011].

O HIL é descrito por [LOUALL et al., 2011] como uma técnica de emulação que é caracterizada pela integração entre a plataforma real de controle e um modelo computacional da planta, que por meio dessa estrutura permite que o processo real possa ser descrito e analisado computacionalmente, como ilustrado na figura 2.4. Esta técnica é aplicada para desenvolvimento, teste e validação de sistemas de tempo real embarcados complexos, proporcionando uma plataforma efetiva e permitindo a adição de complexidade, com a segurança de uma plataforma de testes.

Figura 2.4: Estrutura HIL [GONÇALVES, 2014].

Com o HIL, o sistema embarcado pode ser submetido a diferentes cenários, visto que os sinais de entrada e de saída apresentam seus valores de acordo com o comportamento do processo real [SHIXIANJUN et al., 2006]. O HIL inclui a representação dos sensores e atuadores do sistema. Esses sinais gerados estabelecem uma interface entre a planta/processo emulado, o VANT, e o sistema embarcado. Os dados da planta são enviados como leitura para o sistema embarcado, que processa e atua no modelo computacional da planta de modo a controlá-la. Com base nesta troca de informações, o sistema é executado de acordo com os testes propostos, perimitindo que o comportamento da planta e do sistema embarcado sejam ser analisados.

O uso do HIL permite verificar efetivamente os aspectos com-

portamentais e temporais do sistema desenvolvido. Assim, a utilização desse modelo de simulação tem sido bastante utilizado na verificação do desempenho. Além disso, ajustes e modificações podem ser realizados no sistema antes da sua aplicação definitiva no ambiente real, prevenindo a ocorrência de acidentes e minimizando os riscos [CAI et al., 2008].

O HIL tem sido aplicado com diferentes finalidades no contexto dos VANTs, tais como a validação dos algoritmos de controle e análise de softwares de simulação de vôo. Dessa forma, observa-se que o modelo de emulação HIL tem sido utilizado no desenvolvimento de VANTs com o objetivo de auxiliar no processo de concepção dos mesmos, sendo essa a estrutura utilizada neste trabalho para testes do sistema proposto.

3 Trabalhos Relacionados

Embora a existência de VANTs venha de longa data, foram nas últimas décadas que grandes avanços surgiram nessa área. Alguns desses avanços são na área de inteligência artificial e desenvolvimento de comportamentos complexos. Nesta abordagem os sistemas multi-agentes estão tomando papel de destaque em algumas pesquisas ao redor do mundo, com vários estudos de caso em planejamento de trajetória, trabalhos de coordenação em grupo, a gerenciamento de voos livres, entre outros.

Existem vários avanços no sentido de dotar o VANT de mais autonomia em suas decisões. Como foi abordado por [CHAVES, 2013], o Departamento de Defesa Americano aponta dez níveis de autononia para VANTs, conforme ilustrado na figura 3.1. A curva no gráfico indica a evolução observada e esperada para o nível de autonomia dos VANTs ao longo dos anos, indo desde a operação controlada remotamente até enxames de totalmente autônomos.

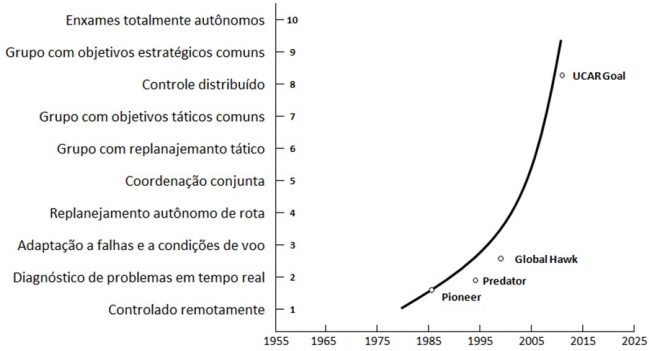

Figura 3.1: Níveis de autonomia em VANTs [CHAVES, 2013].

Além disso, nota-se que o paradigma de agentes tem sido empre-

gado em projetos com VANTs e que a maioria das abordagens apresentam soluções com agente e SMA. Os trabalhos estudados modelam um VANT como um SMA, alguns trazem resultados de coordenação em ambiente simulado, enquanto outros realizam testes com um VANT real acoplado a um simulador. Esses trabalhos exploram as potencialidades do SMA para tomada de decisão em grupo de VANTs, e em algumas aplicações utilizam outras técnicas computacionais ou específicas da área do problema. Essas contribuições servem de base para o presente projeto e ajudaram nas definições e formas de como foi resolvido o problema de pesquisa.

A seguir, serão apresentados alguns trabalhos relacionados ao tema dessa dissertação. Esses trabalhos aqui listados foram selecionados por utilizarem a abordagem de SMA em aplicações com VANTs e servirem de base para desenvolvimento do presente projeto.

3.1. Modelo UAVAS - Unmanned Aerial Vehicles AgentsSpeak

Os VANTs vêm sendo utilizados em pesquisas no meio acadêmico com o desenvolvimento de diversas tecnologias e algoritmos. O uso de agentes inteligentes e sistemas multi-agentes tem sido explorado para o controle de VANTs como forma de prover auxílio na abstração, concepção e no desenvolvimento desses sistemas para possibilitar a implementação de comportamentos inteligentes nessas aeronaves. O trabalho de [HAMA, 2012] aborda a aplicação do paradigma da programação orientada a agentes para controle de comportamento de VANTs, com a concepção de um *framework* através do uso da arquitetura, teoria e ferramentas orientadas a agentes. Neste trabalho é proposto o modelo UAVAS - Unmanned Aerial Vehicles AgentsSpeak, que é um *framework* de programação de comportamentos para VANTs. Os resultados para validação e avaliação da proposta foram obtidos através de simulações com a infraestrutura implementada. Foram realizados dois estudos de caso, um que enfatizava a comunicação entre os VANTs e a cooperação do time e outro com ênfase na verificação dos mapeamentos de sinais com o envio de dados da infraestrutura. A figura 3.2 mostra o modelo conceitual do fluxo de dados da arquitetura do modelo UAVAS.

A arquitetura do modelo baseia-se em um esquema de fluxo e conversão de dados. O agente possui dois tipos de interação com o ambiente: agir sobre ele (ação) ou percebê-lo (percepção). O *framework* é executado dentro do sistema operacional do hardware embarcado do VANT. O VANT envia e recebe bytes de dados à plataforma UAVAS

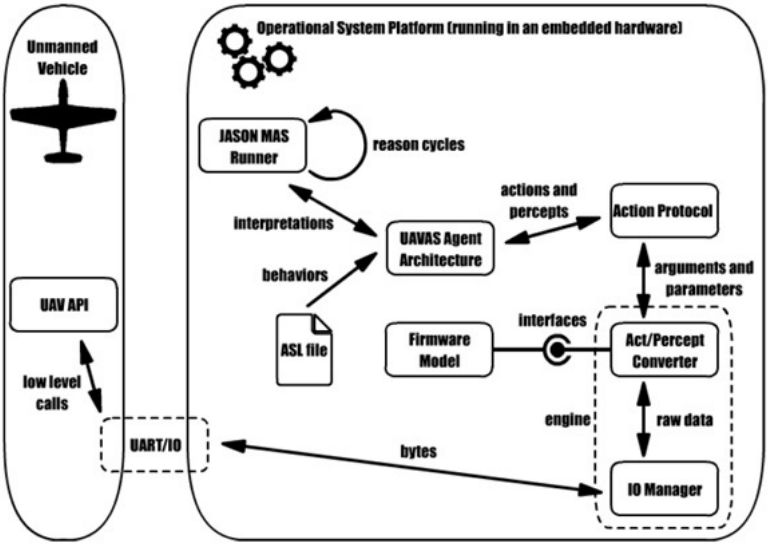

Figura 3.2: Modelo conceitual do *framework* UAVAS [HAMA, 2012].

através do UART/IO, que faz chamadas diretamente à API do firmware do VANT com chamadas de baixo nível. Os bytes de dados são convertidos em fluxos de dados por um gerenciador de IO (Input/Output), e processados por um conversor de ações/percepções que utiliza um modelo de interface com o firmware do VANT para construção de valores válidos a partir de parâmetros e argumentos. Estes valores são enviados ou recebidos pelas ações geradas no Protocolo de Ações, onde tais ações são chamadas pela implementação da Arquitetura dos Agentes. Os agentes são interpretados por um sistema multi-agente Jason, onde ocorrem os ciclos de raciocínio deliberativo.

Esse trabalho utiliza um sistema com múltiplos agentes (SMA) para controlar um único VANT, também enfatiza a comunicação entre esses agentes e a cooperação deles como time. Nele tem-se o uso de agentes BDI, mas o foco é dado no *framework* e seu modelo de abstração que mapeia um veículo aéreo tripulado em um SMA para um veículo aéreo não-tripulado. Porém, não foram realizados testes deste *framework* em uma aplicação embarcada real, tendo resultados somente em simulação.

3.2. VANTs cooperativos aplicados a operações de busca e salvamento

Os VANTs são utilizados em operações de risco e estressantes para o ser humano. Uma aplicação importante é o uso de robôs aéreos em operações de busca e salvamento envolvendo múltiplos VANTs cooperativos, com risco de colisões e tempo de voo limitado. No trabalho de [CHAVES, 2013], foram estudados diferentes algoritmos de navegação e padrões de busca adequados, assim como uma visão geral sobre mecanismos de coordenação multi-agente para coordenação distribuída de agentes (VANTs) visando cooperação. O projeto propõe um modelo de VANTs cooperativos que combina mecanismos de coordenação multi-agente, algoritmos de navegação e padrões de busca estabelecidos pelos principais órgãos responsáveis pelas operações de busca e salvamento. A arquitetura geral do VANT é ilustrado na figura 3.3. O estudo de caso proposto por esses autores corresponde a operações de busca e salvamento, cujo objetivo é varrer uma área de busca para localizar pessoas que se encontram em situações difíceis. A abordagem multi-agente é adotada devido à autonomia ser sua principal característica, também desejada no comportamento do VANT nessas aplicações. A avaliação da sensibilidade do percentual médio de detecção de objetos e o tempo médio de busca foi realizada através de um simulador.

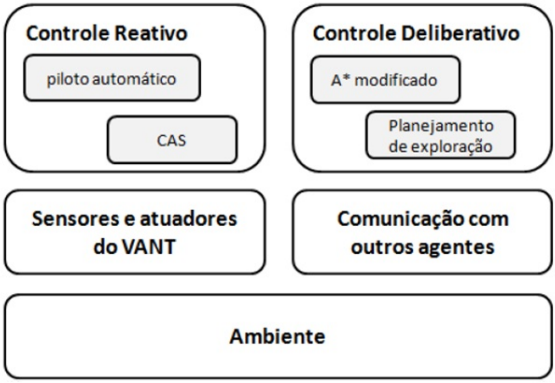

Figura 3.3: Arquitetura Geral do VANT [CHAVES, 2013].

A arquitetura geral do VANT possui dois mecanismos, um reativo e outro deliberativo. O controle reativo possui prioridade sobre o deliberativo, e é responsável por manter a estabilidade da aeronave,

manter a rota e responder a situações de emergência, por exemplo. O controle reativo recebe estímulos dos sensores e utiliza os atuadores para executar as funções do piloto automático e do mecanismo anticolisão CAS - Collision Avoidance Systems. Já no controle deliberativo, observa-se as funções de planejamento de trajetória e de coordenação com outros VANTs. Também busca-se evitar colisões por meio do planejamento da rota. O seu foco principal é o de deliberar para realizar a busca do alvo cooperando com os outros VANTs e maximizando a eficiência do grupo.

Esse trabalho explora a cooperação em grupo de VANTs com foco na coordenação do SMA. Seus resultados são avaliados por meio de simulações em PC, sem implementação em plataformas de VANTs reais, a qual dependerá da capacidade do hardware utilizado para implementar e executar o modelo proposto.

3.3. Integração de VANTs autônomos e simulação multi-agente

Este trabalho aborda o uso de equipes de múltiplos VANTs autônomos. O mesmo apresenta algumas soluções para os problemas no processo de integração de VANTs reais em um sistema de simulação multi-agente com VANTs virtuais adicionais, compondo em um sistema de realidade mista, onde VANTS reais e VANTs virtuais podem coexistir, coordenar o seu voo e cooperar em tarefas comuns. Esses VANTs reais foram equipados com um computador *Gumstix*, sendo capazes de realizar planejamento on-board, raciocínio e a comunicação com outros VANTs do sistema, podendo cooperar e coordenar seus movimentos uns com os outros, e também com os VANTs virtuais. Dois VANTs reais de asa fixa foram integrados ao sistema multi-agente *AgentFly* (figura 3.4), que é utilizado para simulação de VANTs e tráfego aéreo, permitindo a coordenação e cooperação de agentes complexos, além de fornecer mecanismos de prevenção de colisão. O sistema foi modificado para permitir que tanto os VANTs reais quanto os VANTs virtuais pudessem agir e interferir em um ambiente comum. Este trabalho trata de uma abordagem com o VANT de hardware, em um experimento similar ao Hardware-in-the-loop (HIL), no qual esse VANT de hardware foi acoplado ao sistema *AgentFly* e conseguiu-se simular a sua interação com outros VANTs, reais e virtuais [SELECKY; MEISER, 2012].

Esse trabalho explora a parte de integração de VANTs em ambiente simulado, mas não garante que o sistema possa ser utilizado sem modificações em uma aeronave real. Além disso, não investiga ou trata

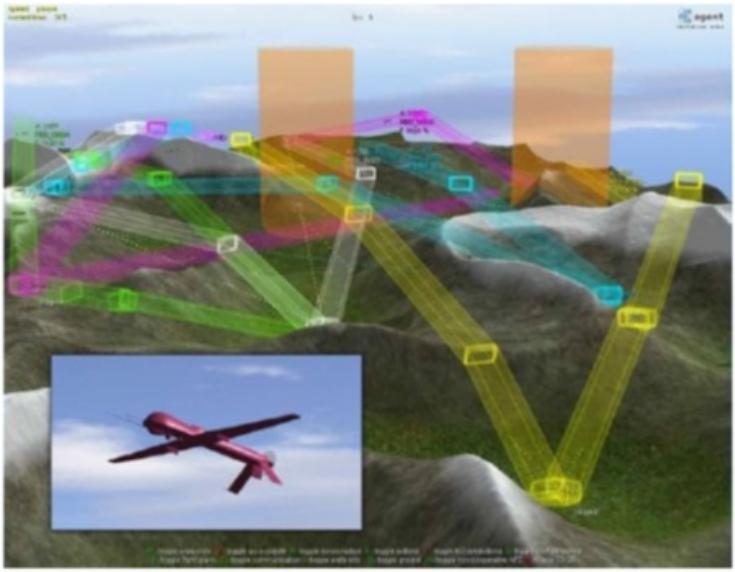

Figura 3.4: AgentFly multi-agent system.

da possibilidade do uso de um agente em um VANT, utilizando uma estrutura do tipo HIL.

3.4. Controle cooperativo de VANTs baseado em Sistema Multi-Agente

Em se tratando do problema de missão de voo coordenado e cooperativo de VANTs, o trabalho de [HAN et al., 2013] traz uma abordagem de um sistema agente híbrido de iniciativa e auto-governo com capacidade de reação fina e raciocínio. Na guerra moderna, por exemplo, os ambientes de batalha variam bastante. O modo de operação ponto-a-ponto tradicional do VANT tem uma grande limitação que não pode resolver problemas dinâmicos e complexos, como o voo coordenado e missão cooperativa. Nesta proposta, o voo coordenado do VANT é realizado através da elaboração de um algoritmo de controle e do método de campo potencial artificial no sistema. Para a missão cooperativa no sistema, o modelo BDI é usado para modelar e classificar a missão. O agente usa o modelo BDI para descrever as informações de missão no sistema. Nele, a "crença" representa a motivação do processamento

da missão que é gerada por considerar a sua capacidade própria, as condições restritivas e do ambiente ao redor. O "desejo" representa o interesse para a missão, geralmente o sistema pode obter benefícios ao terminar essas missões. E a "intenção" representa o método de resolver a missão. A partir da análise do acoplamento e das propriedades dinâmicas da missão, os algoritmos de leilão e licitação são integrados com os algoritmos Potencial Prim e Particle Swarm para a decomposição racional e alocação ótima da missão, visando atingir o máximo lucro do sistema. Este sistema foi avaliado através de cálculo e simulações, considerando sua flexibilidade, estabilidade e capacidade de perceber a viabilidade do voo coordenado e a otimização da missão de cooperação. Porém, esse trabalho apresenta resultados em termos de simulação e não garante a execução da solução proposta em uma plataforma embarcada na aeronave. Além disso, por fazer uso de vários algoritmos complexos em seus cálculos não foi realizada uma análise do desempenho do sistema em casos de restrição do hardware.

3.5. Planejamento de múltiplas rotas para VANTs

A respeito da tarefa de planejamento de múltiplas rotas para VANTs, o trabalho de [CHEN et al., 2013] estabelece um algoritmo híbrido baseado em sistema multi-agentes (MAS) e otimização por enxame de partículas (Particle Swarm Optimization - PSO), denominado Multi-Agent Particle Swarm Optimization (MAPSO). Nele, a estrutura da população tradicional da PSO original é ajustada, de modo que uma partícula em MAPSO é considerada como um agente e representa um trajeto candidato. Todos os agentes vivem em um ambiente de rede, com cada agente fixo em um ponto da estrutura. Por este meio, a velocidade da informação que passa entre as partículas é otimizada. Além disso, um algoritmo de agrupamento do tipo K-means é introduzido para formar subpopulações espaciais distintas. Como resultado, foi encontrada uma maneira eficaz de planejar várias rotas. Os resultados da simulação mostram que a abordagem pode satisfazer os pedidos do planejamento de vários percursos para VANT. Contudo, esse trabalho também analisa a solução a nível de simulação e explora bastante o uso de agente, mas não um agente BDI e suas potencialidades.

3.6. Considerações

Diante do que foi visto, percebeu-se que o paradigma de agentes tem sido empregado em projetos com VANTs e que a maioria das

abordagens observadas apresenta soluções com agentes, sistemas multi-agentes ou sistemas híbridos aplicados a VANTs e times de VANTs. Alguns dos trabalhos modelam um VANT como um sistema multi-agente ao invés de um VANT como um único agente. Outros trazem apenas resultados da proposta em ambiente simulado, enquanto outros realizam um teste com um VANT real acoplado a um simulador. Esses trabalhos também exploram as potencialidades de um sistema multi-agente para tomada de decisão em grupo, no contexto de times de VANTs, e em determinadas aplicações fazem uso de outras técnicas computacionais ou específicas da área do problema, de modo a alcançar melhores resultados. Logo, esses trabalhos não exploram o uso de um único agente BDI em um VANT e as potencialidades da arquitetura, tais como a sua capacidade de reagir rapidamente a mudanças em seu ambiente e de ter objetivos de longo prazo. Assim, com base nessas características, dotar um VANT de autonomia em suas aplicações, garantindo o comprometimento com a missão dada e a reação aos imprevistos durante a sua execução. Da mesma forma, percebe-se que esses trabalhos não exploram o uso da abordagem com agente BDI embarcado diretamente na plataforma computacional de um VANT, sendo este o foco do presente trabalho.

Neste trabalho foi desenvolvido um modelo de comportamento para um VANT baseado no modelo de fluxo e conversão de dados para interação do agente com o ambiente, proposto no trabalho de [HAMA, 2012]. Porém, foi utilizado de um único agente BDI para o VANT e não um SMA. A arquitetura do sistema foi inspirada no trabalho de [CHAVES, 2013], com dois níveis: um nível para os controles de estabilidade e de translação da aeronave; e outro nível para o planejamento de rota, desenvolvido com uma arquitetura de agentes BDI. Já a parte de testes, inspira-se no trabalho de [SELECKY; MEISER, 2012] e seu modelo de integração, mas em vez da integração do VANT real com o simulador, realizou-se a integração da plataforma embarcada da aeronave com o modelo computacional do VANT. Os demais trabalhos de [HAN et al., 2013] e [CHEN et al., 2013] trazem uma abordagem com sistemas híbridos com o uso agentes e outras técnicas computacionais para solução de problemas com VANTs, a qual pode ser uma estratégia utilizada no desenvolvimento uma solução futura ou melhoramento da proposta deste projeto.

4 Agentes em Sistema Embarcado

Para se atingir o objetivo deste projeto de analisar o uso de um agente com arquitetura BDI em uma aplicação embarcada em um VANT, utilizou-se a plataforma do projeto ProVant para análise, bem como o software Jason de programação de agentes BDI. O desenvolvimento do trabalho se deu com uma aplicação de planejamento de rota do VANT, com um agente BDI que define os voos da aeronave.

Neste capítulo, será apresentado o desenvolvimento do **SPR-VANT** - *Sistema de Planejamento de Rota para Veículo Aéreo Não-Tripulado* ou **RPS-UAV** - *Route Planning System for Unmanned Aerial Vehicle*, um sistema de tomada de decisão para planejamento de rota de um VANT autônomo.

O sistema foi desenvolvido para atender à aplicações com aeronaves autônomas. O projeto também tem como objetivo explorar as potencialidades do uso de um agente com arquitetura BDI no contexto dessas aplicações e analisar a viabilidade de executar esse agente no sistema computacional embarcado do VANT.

Este capítulo está organizado da seguinte forma: a seção 4.1 apresenta a visão geral do projeto do sistema desenvolvido; a seção 4.2 mostra a arquitetura de software do sistema; a seção 4.3 descreve um cenário de aplicação para análise do sistema proposto; e a seção 4.4 descreve o desenvolvimento da aplicação.

4.1. Visão Geral

A robótica móvel de modo geral, não só aplicações com VANTs, trabalha com planejamento de rotas [RAFFO; RICO, 2014], onde o planejador de rotas é o módulo responsável por determinar de forma autônoma o caminho a ser seguido pelo robô, de acordo com a tarefa atualmente em execução. Dessa forma, busca-se uma solução com o desenvolvimento de um sistema de planejamento de rota para robôs móveis com o uso de um agente BDI, explorando as potencialidades da arqui-

tetura nessas aplicações.

Geralmente, a estrutura de controle de navegação de um Robô Móvel Autônomo (RMA) é organizada em cascata com quatro níveis de controle segundo [RAFFO; RICO, 2014], como ilustrado na figura 4.1. No nível 4 encontra-se o planejamento dinâmico e a geração das rotas do robô. No nível 3 são executados tipicamente os algoritmos de controle que são responsáveis pela condução do veículo, através do seguimento ou rastreamento de trajetórias, baseados em modelos cinemáticos ou em equações de movimento translacional, os quais garantem o movimento desejado para o robô. No nível 2 é executado o controle da dinâmica do robô, com o objetivo de manter as velocidades longitudinal e lateral do veículo, ou a rotação do robô, e suas derivadas, estabilizadas em torno de um ponto de equilíbrio, que mantém a estabilidade do movimento do veículo. No nível 1 são implementados os sistemas de controle dos sensores e atuadores, garantindo assim as percepções e atuações do robô no ambiente.

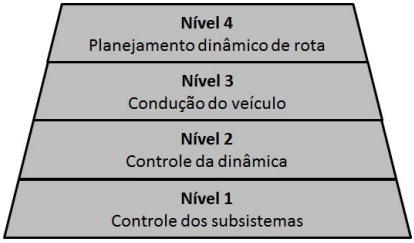

Figura 4.1: Níveis de controle de um RMA [RAFFO; RICO, 2014].

A figura 4.2 ilustra um exemplo da estrutura de controle em cascata de um RMA representada em um diagrama de blocos.

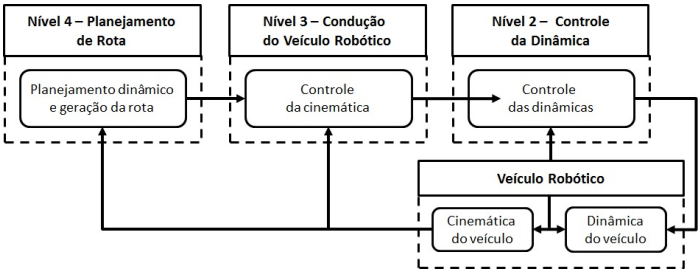

Figura 4.2: Estrutura de controle em cascata para robôs móveis, adaptado de [RAFFO; RICO, 2014].

O planejamento de rota consiste, dadas as configurações inicial e final do robô, em descobrir uma sequência de movimentos a ser executada para que ele saia da primeira e atinja a segunda. Assim, a solução proposta neste projeto trabalha com o enfoque no nível 4 da estrutura de controle de navegação, para o desenvolvimento de um modelo de comportamento autônomo com um agente BDI para planejamento de rota e tomada de decisão em robôs móveis.

4.2. Sistema Proposto

Esse sistema foi desenvolvido de forma modular, como uma boa prática no projeto de sistemas computacionais, visando a estruturação, manutenção e reutilização de alguns módulos comuns para análise e comparação entre as abordagens utilizadas no contexto da aplicação com VANTs.

A solução proposta com agente BDI trata do nível de **Planejamento** da estrutura do controle de navegação. Assim, os níveis 1, 2 e 3 foram implementados em um único nível, chamado de **Controle**, tendo uma nova estrutura como ilustrada na figura 4.3.

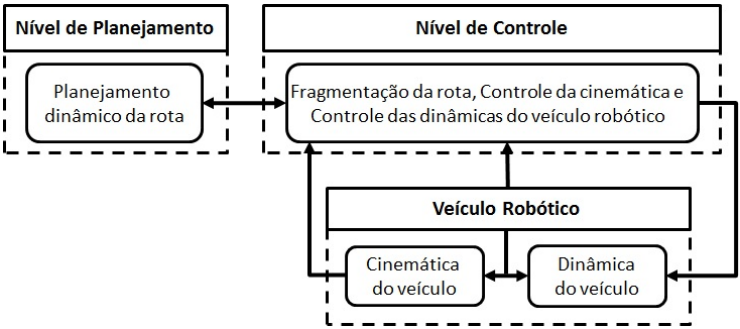

Figura 4.3: Estrutura proposta.

O fluxo de informação entre os níveis ocorre a partir da percepção do ambiente pelos sensores do veículo robótico. Os sinais dos sensores são enviados para o nível de **Controle**, que recebe os dados, processa e comunida para o nível de **Planejamento**, fornecendo as informações das percepções do ambiente requeridas pelo agente, conforme a aplicação. Com base nesses dados, o nível de **Planejamento** realiza o seu processo de tomada de decisão e define as atuações a serem executadas no ambiente, enviando para o nível de **Controle** os pontos de destino

(*waypoints*) da aeronave. sssdsc

Uma interface de comunicação entre os níveis garante a troca de informações. O nível de **Controle** recebe as mensagens de atuação do nível de **Planejamento** com os valores de referência de velocidade e da posição de destino, e envia as percepções com as informações da posição atual (GPS), valores da carga e da taxa de consumo da bateria. A partir de uma nova mensagem do nível de **Planejamento**, ocorre a fragmentação da rota em pontos intermediários entre a posição atual do VANT e o *waypoint* de destino, através de um percurso retilíneo dividido em trechos menores de passo fixo. Também acontece a correção do ângulo de direção do VANT em relação ao ponto a ser alcançado. Dessa forma, as refências de posição e ângulo são passadas ao nível de **Controle** para seguimento da rota definida no nível de **Planejamento**.

No nível de **Controle** estão implementados os algoritmos de controle de estabilidade e de controle de translação do VANT, conforme detalhado em [DONADEL et al., 2014]. Neste nível, são executadas as rotinas do controle de estabilidade (controle rotacional) e de seguimento de trajetória (controle translacional). O controle de estabilização é baseado na posição (x, y, z) e velocidade linear (x', y', z'), assim como na sua orientação. A partir destes, são calculados os erros com relação às posições de referência e então é calculado o controle. Já o controle de seguimento de trajetória se baseia nos ângulos de referência, na velocidade angular de referência, na atitude e velocidade angular do VANT. Calculados os controles de estabilidade e orientação, estes dados precisam ser transformados em valores compatíveis com os atuadores do sistema. Para isto, é aplicada uma transformação de sinais. gerando sinais de força e ângulo.

4.3. Cenário de Teste

O cenário de teste tem o objetivo de testar e ilustrar a aplicação do sistema proposto. Além disso, ele serve para análise e avaliação das abordagens utilizadas neste trabalho.

O cenário escolhido para apresentar o uso do mecanismo proposto consiste em uma aplicação para visitar os nodos de uma rede de sensores sem fio com um VANT. Esses nodos se encontram distribuídos no solo em uma determinada região de monitoramento, a aeronave deve visitá-los, a partir da estação-base, e retornar para a base, conforme ilustrado na figura 4.4. A posição de cada sensor é conhecida pelo VANT desde o início da missão. A medida que a aeronave alcança um *waypoint* de destino, ela tenta obter os dados daquele sensor e, em

seguida, passa para o próximo nodo. Após todos os sensores serem visitados, o VANT finaliza a sua missão na base.

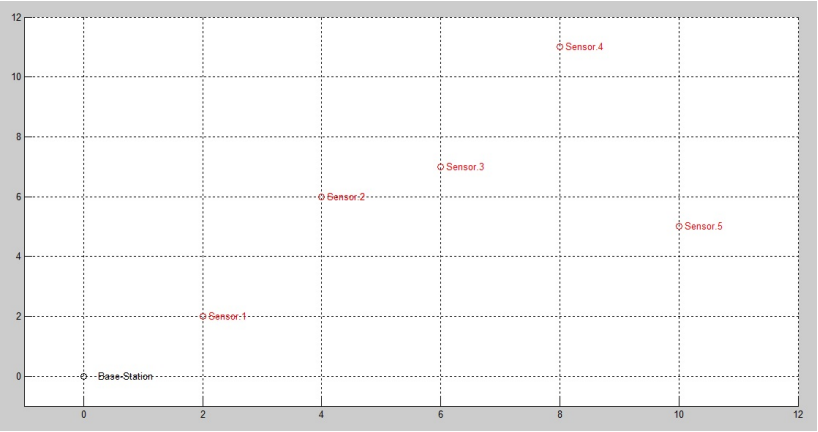

Figura 4.4: Cenário de Teste.

A aeronave parte do solo e deve manter uma altitude de cruzeiro para realizar a coleta dos dados com as informações dos sensores da rede. Além disso, deve-se levar em consideração a carga (B) e a taxa (T) de consumo de bateria do VANT, sendo que essa taxa varia de acordo com a velocidade e direção do vento. As possíveis direções do vento são os 8 sentidos da rosa dos ventos, os 4 pontos cardeais (norte, leste, oeste, sul) e 4 pontos colaterais (nordeste, sudeste, sudoeste, noroeste).

Neste cenário, não foi utilizado um modelo de bateria específico, optou-se por criar um modelo aproximado que descreve o decaimento linear da carga, de modo a analisar a influência da queda do nível da bateria no sistema.

Tem-se que a distância máxima que a aeronave consegue percorrer é dada pela equação (4.1), onde a carga da bateria (B) varia de 0 a 100% e a taxa de consumo (T) é dada em %/m. (Esse modelo não é realista e foi utilizado para os testes).

$$Dmax = B/T \tag{4.1}$$

As informação das condições climáticas, direção e velocidade do vento, são obtidas de uma estação meteorológica, e a missão é iniciada com esses parâmetros. Assim, a medida que a aeronave se desloca, mantendo uma velocidade constante de voo, a carga da bateria decrementa em uma taxa que varia conforme a direção que se encontra o

VANT e o vento. Se o vento está em sentido contrário à aeronave, a taxa de consumo de bateria é maior e com vento a favor o consumo é menor.

O VANT deve considerar também o "ponto de não retorno", que é definido como aquele trecho da rota de um voo em que o combustível restante na aeronave não é suficiente para que ela retorne ao aeroporto de partida, em caso de emergência. Para isso, deve-se levar em consideração a quantidade de carga disponível em sua bateira, a taxa de consumo atual e a distância até a estação base.

Assim, quando o VANT perceber que não conseguirá visitar todos os nodos da rede de sensores devido a sua carga de bateria, estando ele perto do ponto de não retorno, ele deve abortar a missão de coleta de dados no ponto em que estiver e retornar à estação-base para recarregar a bateria. Além disso, durante o percurso de retorno à base, caso a aeronave sofra a incidência de um vento em direção contrária à sua rota e o consumo de bateria seja alto, deve-se analisar se consegue alcançar a base. Caso avalie que não seja possível chegar até a estação-base com segurança, o VANT deve realizar um pouso de emergência no local mais próximo.

4.4. Modelagem

Com base no cenário de aplicação descrito na seção anterior, fez-se então a modelagem do problema para ser implementado com a abordagem de agentes e também para abordagem usual, visando a comparação e análise.

4.4.1. Especificação da abordagem BDI

Na área de SMA, apesar da grande quantidade de pesquisas desenvolvidas, ainda não existe um método que pode ser considerado padrão ou um ferramental de projeto e desenvolvimento desses sistemas. Neste projeto foi utilizado o método Prometheus AEOlus de [UEZ, 2013] para desenvolvimento da aplicação com um agente BDI, o qual tem como objetivo permitir o projeto e a análise de um sistema com agentes.

A modelagem do sistema foi feita com o Diagrama de Objetivos do Sistema do Prometheus AEOlus, o qual permite a análise dos objetivos em forma de uma árvore de decomposição AND/OR, como ilustrado na figura 4.5.

Os objetivos são refinados em sub-objetivos e podem ser interli-

gados através de ligações do tipo AND ou OR. A ligação AND indica que o objetivo será atingido se todos os seus sub-objetivos também forem alcançados. A ligação OR indica que o objetivo será atingido assim que um dos sub-objetivos for alcançado. Ambas as ligações são representadas por setas contínuas rotuladas com AND ou OR. As ligações AND também podem expressar a ordem na qual os sub-objetivos devem ser atingidos. Os sub-objetivos interligados com seta pontilhada que parte de um sub-objetivo que deve ser atingido antes até o sub-objetivo ocorre depois. Esse diagrama possui uma implicação sequencial e paralela, de modo a descrever os objetivos do sistema durante a sua execução.

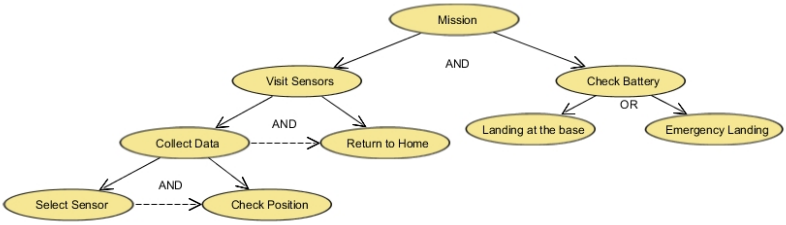

Figura 4.5: Diagrama de Objetivos do Sistema.

O objetivo principal do sistema é a *Mission*, o qual é decomposto nos subobjetivos *Visit Sensors* e *Check Battery*. Por sua vez, o objetivo *Visit Sensors* só é atingido após se completar os objetivos *Collect Data* e *Return to Home*, necessariamente com essa ordem de precedência. O objetivo de *Collect Data* ocorre mediante a completude dos objetivos, *Select Sensor* e *Check Position*, também nessa ordem.

Após os objetivos das sub-árvores da esquerda serem alcançados, para se atigir o objetivo *Visit Sensors*, o sistema vai em busca do objetivo *Return to Home*. Mas de forma paralela, o objetivo *Check Battery* da sub-árvore da direita também deve ser alcançado para o cumprimento do objetivo principal, que é o *Mission*. Para se concluir o objetivo *Check Battery*, um dos objetivos *Landing at the Base* ou *Emergency Landing* devem ser alcançados. Na fase de implementação, cada um desses objetivos será detalhado e traduzido em programa.

Outro diagrama do Sistema do Prometheus AEOlus utilizado para especificação foi o diagrama de Visão Geral do Sistema, o qual permite a visualização da estrutura global do sistema e mostra além do agente, as suas percepções e ações, bem como as suas crenças, conforme ilustra a figura 4.6. Assim, tem-se o agente ***VANT*** com as percepções

de posição, carga e taxa de consumo da bateria, além das atuações do agente com a velocidade e a posição de destino da aeronave.

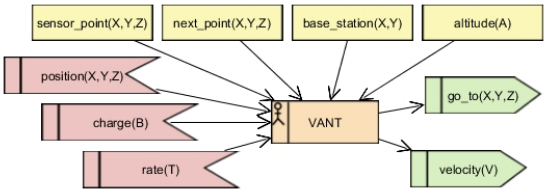

Figura 4.6: Diagrama do Agente.

Essas crenças do agente são as informações que ele precisa para o seu processo de tomada de decisão com relação ao planejamento de rota do VANT. No início da missão, o agente possui em sua base de crenças a informação de todos os nodos sensores que precisa visitar, referido por `sensor_point(X,Y,Z)`. No processo de deliberação do agente, a informação de qual será o próximo nodo a ser vistidado (`next_point(X,Y,Z)`) é atualizada cada vez que esse ponto de referência é alterado pelo agente, dada a sua localização atual e *waypoint* de destino. A informação da localização da estação-base (`base_station(X,Y)`) é fixada desde o início da missão, assim com a altitude de cruzeiro (`altitude(A)`) que o VANT manterá durante o seu voo.

4.4.2. Especificação Usual

Essa versão do sistema de tomada de decisão do VANT foi projetado com o auxílio de uma máquina de estados. Uma máquina de estados possui uma implicação sequencial do sistema, de modo que através dela se especifica as sequências de estados pelos quais um processo passa durante seu tempo de execução em resposta a eventos. Essa é uma das práticas adotatas em projetos com esse tipo de abordagem usual.

Dessa forma, o nível de **Planejamento** do VANT foi projetado com a espeficicação em Máquina de Estados ilustrada na figura 4.7, buscando modelar o comportamento esperado da aeronave no cenário de teste escolhido. Cada estado representa uma condição na qual se encontra o VANT em determinado momento da execução do sistema e as transições correspondem aos eventos que acarretam em uma mudança de estado.

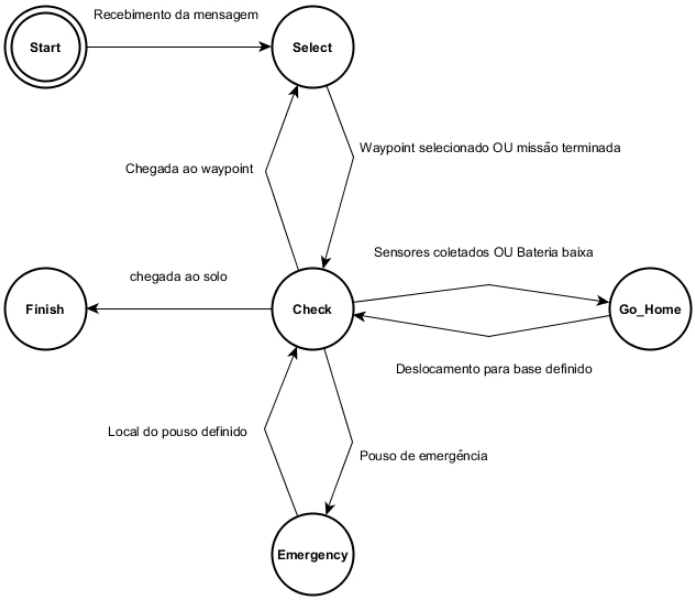

Figura 4.7: Estados de operação do VANT.

Ao iniciar, o sistema se encontra no estado *Start* e permanece nele até que ocorra o recebimento de uma mensagem com a posição do VANT e o estado da bateria, vindo do nível **Controle**. Dessa forma, o VANT consegue se localizar e saber a sua condição atual, passando para o estado *Select*, no qual ele executa a rotina de seleção de qual o nodo sensor (*waypoint*) deseja alcançar.

Após selecionar o sensor alvo, o sistema passa para o estado *Check*, no qual permanece verficando a sua posição atual até que tenha atingido o *waypoint* de destino. Quando o VANT está sobre o *waypoint* de destino, o sistema retorna para o estado *Select* e retoma o processo de seleção.

Depois de alcançar todos os nodos sensores da rede, o sistema passa para o estado *Go-Home*, no qual retorna para estação-base, definindo o próximo *waypoint* como sendo a base. Novamente passa-se a esperar que a aeronave atinja o *waypoint* alvo, permanecendo no estado *Check* até que isso ocorra.

Em uma condição normal de funcionamento, o sistema passa para o estado *Finish* após o VANT chegar ao solo da base e finaliza

a missão de coleta de dados. Porém, quando o VANT se encontra no estado *Check*, pode acontecer da bateria ficar com carga baixa. Dessa forma, o sistema aborta a missão mesmo que não tenha alcançado todos os nodos da rede, muda para o estado *Go-Home* e inicia o percurso de retorno para recarregar. Então o sistema passa para o estado *Check* e espera atingir a estação-base.

Contudo, caso o sistema perceba que o nível de bateria é insuficiente para conseguir chegar até a base, é acionado o pouso de emergência e passa-se para o estado *Emergency*. Neste estado, o VANT decide pousar no ponto mais próximo, visando evitar danos na aeronave. Assim, o sistema passa para o estado *Check* e aguarda até que se alcance o solo no ponto de destino e finaliza a missão no estado *Finish*.

4.5. Arquitetura do Sistema HIL

Para a realização das análises e testes do sistema proposto sem colocar em risco o VANT real, optou-se por implementar o projeto a nível de simulação, através de um modelo de simulação Hardware In the Loop (HIL).

A estrutura geral deste modelo HIL é composta por uma plataforma embarcada com o planejamento de rota e o controle, um computador com o modelo computacional da planta e um mecanismo de comunicação entre eles, como ilustra a figura 4.8.

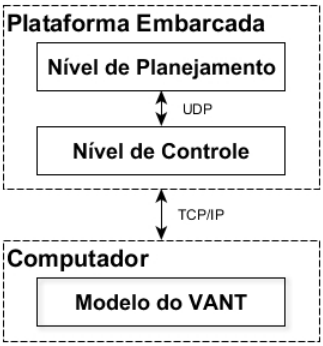

Figura 4.8: Estrutura geral HIL.

Neste trabalho, o modelo computacional da planta, representado pelo **Modelo do VANT** estará rodando em um computador. Já na plataforma embarcada, tem-se a execução do **Nível de Planejamento**

e do **Nível de Controle**, os quais dispõem de um link entre eles através de uma comunicação via sockets UDP. E a comunicação entre a plataforma embarcada e o computador é realizada por meio de um link TCP/IP.

A arquitetura do software desenvolvido para execução na plataforma embarcada é composta por dois processos, sendo estes implementados um no **Nível de Planejamento** e o outro no **Nível de Controle**. As subseções seguintes descrevem esses processos desenvolvidos em cada nível do sistema.

4.5.1. Nível de Controle

O processo implementado no **Nível de Controle** é composto por três módulos: *Control*, *Interface VANT* e *Fragmentation*, conforme a figura 4.9, sendo que cada módulo deste nível é executado de forma específica e concorrente.

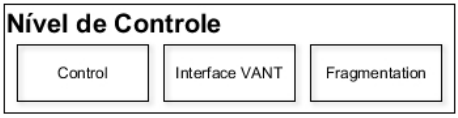

Figura 4.9: Nível de Controle.

O módulo *Control* e o módulo *Interface VANT* foram desenvolvidos e implementados pelo projeto ProVant, sendo estes apenas utilizados como parte integrante do modelo final do sistema proposto. Enquanto as demais partes do sistema e as integrações foram desenvolvidas como parte deste trabalho [GONÇALVES, 2014].

O módulo *Interface VANT* permite a troca de informações entre o **Nível de Controle** e o **Modelo do VANT** através de um link de comunicação TCP/IP, enviando as referências de controle para planta simulada (forças dos motores e ângulos dos servos) e recebendo as informações do comportamento do VANT (posição, atitude e velocidades). Essa tarefa trabalha com um período de 5 ms para executar as funções de envio e recebimento de mensagens.

No módulo *Control* estão implementados os algoritmos de controle de estabilidade e de controle de translação do VANT. Neste módulo, são calculados os erros das posições e das respectivas velocidades nos três eixos cartezianos em relação às referências, bem como a execução das rotinas do controle translacional e do controle rotacional.

Além disso, também é realizada uma operação de transformação dos sinais de saída para o valor de grandeza padrão da comunicação com o **Modelo do VANT**, sendo estes, sinais de força e ângulo. Essa tarefa opera com um período de 5 ms para executar as funções de controle da aeronave.

Já o módulo *Fragmentation* possui um link de comunicação via sockets UDP com o **Nível de Planejamento**, através do qual recebe as mensagens com o valor de referência de velocidade e a posição de destino que a aeronave deve atigir. Por esse mesmo link de comunicação, este módulo envia a informação da posição atual do VANT (GPS), além dos valores da carga e da taxa de consumo da bateria, calculados nesta camada. Neste módulo ocorre a fragmentação da rota em pontos intermediários entre a posição atual da aeronave e o *waypoint* de destino, através de um percurso retilíneo dividido em trechos menores de passo fixo. Também acontece a correção do ângulo de direção do VANT em relação ao nodo sensor a ser alcançado. Dessa forma, as refências de posição e ângulo são passadas para o *Módulo Control* através de variáveis compartilhadas. Essa tarefa é opera com um período de 10 ms para executar as funções descritas.

O envio das informações com os valores de GPS e bateria do **Nível de Controle** para o **Nível de Planejamento** foi implementado de duas formas diferentes. Uma versão foi desenvolvida com a comunicação ocorrendo a cada execução do módulo *Fragmentation*, enviando os dados sempre com a mesma frequência. Outra versão foi implementada com a comunicação sendo realizada após uma mudança significativa no valor da posição da aeronave. Por exemplo, as informações são enviadas para o **Nível de Planejamento** somente quando o VANT altera a sua posição em 0,1 metros. Esse valor foi encontrado de forma empírica, através de testes com o modelo de simulação HIL. Estas duas alternativas serão comparadas nas análises do Capítulo 5.

4.5.2. Nível de Planejamento

O processo que realiza a tomada de decisão e o planejamento da rota do VANT é implementado no **Nível de Planejamento** do sistema. A seguir, será descrito o programa desenvolvido para esse nível, o qual foi implementado segundo as duas especificações apresentadas anteriormente.

4.5.2.1. Arquitetura Orientada a Agentes

Um sistema de tomada de decisão com um agente BDI foi implementado no **Nível de Planejamento**, como ilustrado na figura 4.10. Essa aplicação é formada por um agente BDI desenvolvido em linguagem Jason, com uma arquitetura customizada que permite implementar as percepções e as atuações no ambiente. Essa arquitetura possui uma interface (*Communication Thread* com o **Nível de Controle** que realiza o recebimento das percepções e o envio das atuações do agente.

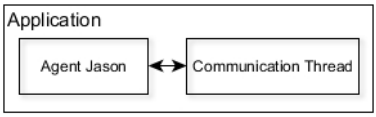

Figura 4.10: Nível de Planejamento com agente BDI.

O agente possui autonomia em suas ações e durante o seu ciclo de raciocínio, ele decide quando deve perceber o ambiente, realizar o seu processo deliberativo e atuar no ambiente. Para não interferir nos ciclos do agente, optou-se por implementar um mecanismo de comunicação que servisse de interface entre o agente e o ambiente, permitido a troca de informação entre o **Nível de Planejamento** e o **Nível de Controle**. Assim, enquanto a interface comunica e mantém as informações mais recentes do ambiente, estas ficam disponíveis para o agente no momento em que ele decide realizar o processo de percepção. Da mesma forma, as atuações do agente permanecem na interface até que uma nova decisão seja tomada pelo agente para atuar no ambiente.

A troca de informações entre os níveis implementados na plataforma embarcada é dada por meio de um link de comunicação via socket UDP. As percepções dos sensores (valores de posição e bateria) são recebidas e modificadas no mecanismo de comunicação do agente, posteriormente essas informações são atualizadas na base de crenças do agente quando ocorre o processo de percepção. Após o agente deliberar, ele executa as atuações a serem aplicadas no ambiente, neste caso no VANT, através de funções implementadas em sua arquitetura customizada do agente (valores de posição e velocidade), os quais são passados para interface de comunicação e depois são encaminhados para o **Nível de Controle**.

Dessa forma, o modelo completo de simulação HIL para o agente Jason possui a estrutura apresentada na figura 4.11, tendo um agente

implementado no **Nível de Planejamento**, os respectivos módulos do *Nível de Controle* e o **Modelo do VANT**.

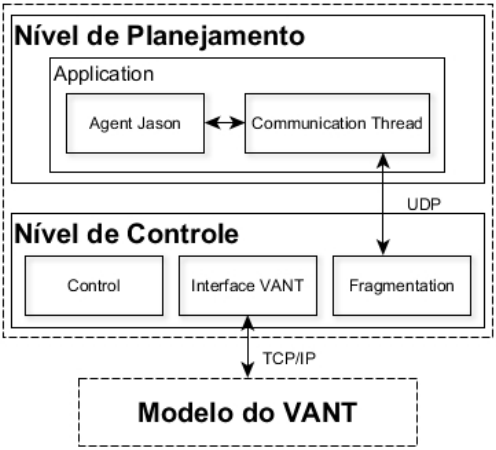

Figura 4.11: Estrutura HIL com agente Jason.

4.5.2.2. Arquitetura Baseada em Programação Imperativa

Outra versão do sistema de tomada de decisão do VANT foi desenvolvida no **Nível de Planejamento** utilizando uma abordagem usual com programação imperativa e implementado em linguagem C++. Neste programa, atende-se aos mesmos requisitos da versão com agente BDI, porém através da linguagem C++ e com seus mecanismos de desenvolvimento.

O sistema opera em um laço de repetição, o qual executa o recebimento das mensagens com os valores de percepção, realiza o processo de tomada de decisão e envio de mensagem com a atuação. Esse sistema recebe os dados com as informações dos sensores (valores de posição e bateria), vindos do **Nível de Controle**, e atualiza as suas variáveis correspondentes. Após isso, ele realiza as operações de tomada de decisão através da execução de suas funções seleção do próximo sensor e do cálculo das distâncias entre a aeronave e os pontos de destino. Em seguida, ocorre o envio das mensagens com os valores de atuação do VANT (valores de posição e velocidade), os quais serão executados no **Modelo do VANT**.

Dessa forma, tem-se o modelo de simulação HIL correspondente para abordagem usual como ilustrado na figura 4.12.

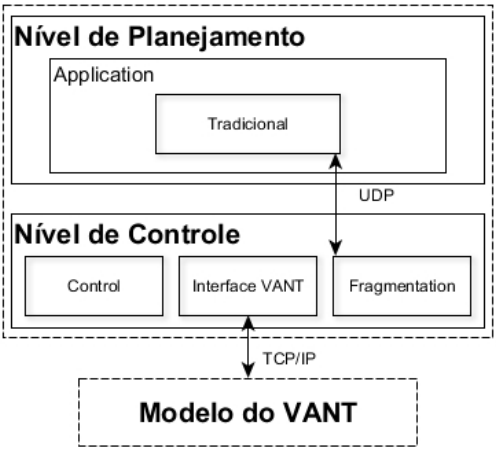

Figura 4.12: Estrutura HIL com abordagem usual.

4.6. Implementação da Aplicação

O uso de duas abordagens diferentes para o desenvolvimento do sistema de tomada de decisão, traz consigo algumas particularidades de cada linguagem e estilo de programação.

As subseções seguintes apresentam a implementação da versão usual e da versão com agentes no contexto Do cenário de teste descrito.

Como o objetivo deste trabalho consiste em uma análise comparativa entre as abordagens de projeto, serão detalhados alguns aspectos de cada alternativa de programação e apresentados os códigos das mesmas.

4.6.1. Abordagem baseado em Agente

4.6.1.1. Implementação

A linguagem Jason possui um elevado nível de abstração que permite ao desenvolvedor trabalhar parâmetros como crenças, objetivos, planos, eventos e intenções. As crenças representam as informações do agente, os objetivos representam os alvos que o agente quer alcançar,

os planos representam a estratégia de ação do agente para atingir um objetivo, os eventos acontecem como consequência de mudanças nas crenças ou objetivos do agente, e as intenções são os planos instanciados para alcançar algum objetivo do sistema, conforme modelado na figura 4.5.

No código Jason é possível criar regras lógicas que são calculadas quando chamadas durante a execução do código, parecidas com as funções em C++. Algumas rotinas do agente foram implementadas dessa forma, como o cálculo da distância entre dois pontos no espaço, a qual é descrito no programa 4.1.

Programa 4.1: Código Jason da regra de cálculo da distância.

```
1 distanceXYZ(X0, Y0, Z0, X, Y, Z, D) :-
2     D = math.sqrt((X-X0)*(X-X0) + (Y-Y0)*(Y-Y0) + (Z-Z0)*(Z-Z0)).
```

Outra rotina que foi implementada como uma regra em código Jason foi a análise da carga disponível na bateria do VANT, conforme o programa 4.2. Essa regra retorna verdadeira quando a aeronave possui carga suficiente para ir até o próximo *waypoint* e retornar para estação-base, dada a sua taxa de consumo naquele momento (`rate(T)`), a carga atual da bateria (`charge(B)`), bem como a posição em que se encontra o VANT e a base (`base_station(BX,BY)`), além do ponto de destino (`next_point(NX,NY,NZ)`). Assim, dado os cálculos das distâncias entre os pontos analisados e a distância máxima que a aeronave consegue percorrer (`Dmax = B/T`) com a carga e a taxa de consumo da bateria analisadas, a regra verifica se essa distância máxima é maior que as distâncias calculadas mais um fator de segurança de 3 metros (`Dmax > D1+D2+3`). Logo, quando a distância máxima for maior, a regra retorna verdadeiro e caso contrário, falso.

Programa 4.2: Código Jason da análise de carga da bateria.

```
1 energy :- rate(T) & charge(B) & position(X,Y,Z) & base_station(BX,BY)
2     & next_point(NX,NY,NZ) & distanceXYZ(X,Y,Z,NX,NY,NZ,D1)
3     & distanceXYZ(NX,NY,NZ,BX,BY,0,D2) & Dmax = B/T & (Dmax > D1+D2+3).
```

Um agente Jason pode ser implementado com os estilos de programação: reativo e proativo. Neste trabalho, o agente foi desenvolvido com características reativas e proativas, dada a necessidade de cada plano da aplicação. Os planos em que o agente deve reagir a mudanças no ambiente foram implementados de forma reativa e os planos em que o agente tem objetivos de longo prazo e necessite de comprometimento com um plano até que ele seja finalizado foram implementados

de forma proativa. A seguir será mostrado como ambas as formas foram utilizadas nos planos do agente.

4.6.1.2. Parte Reativa

Um agente mantém uma interação contínua com o ambiente e responde às mudanças que ocorrem nele. Esse agente pode reagir a cada nova percepção do ambiente ou alteração na sua base de crenças e intenções.

Um plano que foi implementado de forma reativa foi o plano de verificação de carga da bateria do VANT, como descrito no programa 4.3. Esse plano implementa o objetivo *Check Battery* da especificação da figura 4.5. Sua execução é verificada a cada nova perceção do ambiente e atualização da carga da bateria em sua base de crenças (`+charge`). Quando isso ocorre, as condições que estão depois dos dois pontos são analisadas, sendo elas: a negação da regra `energy`, apresentada na seção anterior; e a negação do desejo `go_home` (`not .desire(go_home)`), para que o agente não tenha duas intenções `go_home` de retorno para base - desejos são metas de realização que aparecem no conjunto de eventos ou aparecem nas intenções do agente. Assim, quando a negação da regra `energy` for verdadeira e o agente não estiver com a intenção/desejo `go_home`, esse plano reativo é executado. Tem-se então a execução do comando `.drop_all_desires`, onde todos os eventos e todas as intenções do agente são descartados, por exemplo o desejo de coletar dados. Em seguida é criada a intenção `go_home`, para o VANT retornar para estação-base. O ponto de exclamação se refere a uma intenção do agente e a dupla exclamação (`!!`) é usada para criar uma intenção independente deste plano que a gerou. Dessa forma, o agente aborta os planos e intenções anteriores que estava executando e passa a ter como nova intenção apenas voltar para base, `go_home`.

Programa 4.3: Plano para verificação da carga da bateria (reativo).

```
1 +charge(_) : not energy & not .desire(go_home)
2     <- .drop_all_desires;
3         !!go_home.
```

4.6.1.3. Parte Proativa

Um agente proativo é capaz de executar o seu ciclo de raciocínio de forma autônoma para cumprir seus objetivos, ele não é guiado somente pelos eventos do ambiente e possui iniciativa em suas ações.

Dessa forma, o agente permanece comprometido com um plano até que ele seja finalizado.

O plano de coletar dados dos sensores da rede foi implementado de forma proativa, como descrito no programa 4.4. Esse plano implementa o objetivo *Collect Data* conforme a especificação da figura 4.5. O plano é executado quando o agente tiver a intenção (`!collect_data`) e possuir sensores (`sensor_point(_, _, _)`) em sua base de crenças. A execução desse plano ativa outros planos através de novas intenções, primeiro tem-se a seleção do próximo nodo sensor (`!select_sensor`) e a verificação da chegada ao local de destino (`!check_position`). Após se atingir o *waypoint* alvo, o agente consulta em sua base de crenças o próximo ponto (`?next_point(X,Y,Z)`) que desejava alcançar e remove de suas crenças a informação referente ao sensor (`-sensor_point(ID,X,Y)`) com essa posição. Então, retoma-se novamente a execução do plano com a ativação da intenção (`!collect_data`). Esse processo permance até que todos os sensores mapeados na memória do agente sejam visitados e quando não há mais sensores na base de crenças do agente, o plano (`+!collect_data.`) na última linha do código é executado, encerrando assim o laço de repetição. Isso garante o cumprimento da missão até o final, conforme projetado para essa aplicação.

Programa 4.4: Plano para coletar dados dos sensores da rede (proativo).

```
1 +!collect_data : sensor_point(_, _, _)
2     <-  !select_sensor;
3         !check_position;
4         ?next_point(X,Y,Z);
5         -sensor_point(ID,X,Y);
6         !collect_data.
7
8 +!collect_data.
```

4.6.1.4. Agente Jason

O código do agente Jason possui as regras descritas no programa 4.5, as crenças e o objetivo apresentados no programa 4.6, além dos planos listados no programa 4.7.

Programa 4.5: Regras do agente Jason.

```
1 // Regra para pouso de emergencia
2 emergency :- rate(T) & charge(B) & position(X,Y,Z)
3     & distanceXYZ(X,Y,Z,X,Y,0,H)
4     & Dmax = B/T & (Dmax < H+3).
5
6 // Regra de analise da quantidade de energia da bateria
```

```
energy :- rate(T) & charge(B) & position(X,Y,Z) & base_station(BX,BY)
     & next_point(NX,NY,NZ) & distanceXYZ(X,Y,Z,NX,NY,NZ,D1)
     & distanceXYZ(NX,NY,NZ,BX,BY,0,D2) & Dmax = B/T & (Dmax > D1+D2+3).

// Distancia entre dois pontos em (X, Y)
distance(X0,Y0,X,Y,D) :- D = math.sqrt((X-X0)*(X-X0)+(Y-Y0)*(Y-Y0)).

// Distancia entre dois pontos em (X, Y, Z)
distanceXYZ(X0,Y0,Z0,X,Y,Z,D) :-
     D = math.sqrt((X-X0)*(X-X0)+(Y-Y0)*(Y-Y0)+(Z-Z0)*(Z-Z0)).

// Escolha do sensor mais proximo do VANT
closer(X0,Y0,ID,X,Y) :-
     sensor_point(ID,X,Y) & not(sensor_point(ID1,X1,Y1) & ID \== ID1
     & distance(X0,Y0,X,Y,D) & distance(X0,Y0,X1,Y1,D1) & D1<D).
```

Programa 4.6: Crenças e objetivo do agente Jason.

```
// Localizacao dos nodos sensores na rede
sensor_point(id1,2,2).
sensor_point(id2,4,6).
sensor_point(id3,6,7).
sensor_point(id4,8,11).
sensor_point(id5,10,5).

altitude(5). // atitude de cruzeiro
next_point(0,0,0). // ponto inicial
base_station(0,0). // posicao da estacao-base

/* Initial goals */
!mission.
```

Programa 4.7: Planos do agente Jason.

```
+!mission
     <- velocity(0.1,0.1,0);
         .wait({+charge(_)});
         !collect_data;
         !go_home.

+!collect_data : sensor_point(_, _, _)
     <-  !select_sensor;
         !check_position;
         ?next_point(X,Y,Z);
         -sensor_point(ID,X,Y);
         !collect_data.

+!collect_data.

@lbattery[atomic]
+charge(_) : not energy & not .desire(go_home)
     <-  .drop_all_desires; !!go_home.

@lemergency[atomic]
+charge(_) : emergency & not .desire(emergency_landing)
     <-  .drop_all_desires; !!emergency_landing.

+!select_sensor
```

```
25      <-  ?altitude(Z);
26          ?position(X0,Y0,Z0);
27          ?closer(X0, Y0, ID, X, Y);
28          -+next_point(X,Y,Z);
29          go_to(X,Y,Z).
30
31  +!go_home
32      <-  ?base_station(X,Y);
33          -+next_point(X,Y,0);
34          go_to(X,Y,0);
35          !check_position;
36          !finish.
37
38  +!emergency_landing
39      <-  ?position(X,Y,Z);
40          -+next_point(X,Y,0);
41          go_to(X,Y,0);
42          !check_position;
43          !finish.
44
45  +!check_position : position(X0,Y0,Z0) & next_point(X,Y,Z)
46      & distanceXYZ(X0,Y0,Z0,X,Y,Z,D) & (D < 0.1).
47
48  +!check_position
49      <- .wait({+position(_,_,_)});
50         !check_position.
51
52  +!finish
53      <- .stopMAS.
```

Os planos descritos no código do programa 4.7 representam o comportamento do sistema de tomada de decisão com um agente BDI, conforme a especificação (figura 4.5). Primeiramente, o agente possui o objetivo missão (`!mission`) que é executado com o plano (`+!mission`). Neste plano ocorre a atuação do agente com o envio das velocidades de referências do VANT nos três eixos cartezianos (`velocity(0.1,0.1,0)`) para o **Nível de Controle**. Após isso, ele espera por uma percepção da carga da bateria (`.wait(+charge(_))`) e em seguida, cria o objetivo (`!collect_data`). Somente após este objetivo ser concluído que o agente cria o novo objetivo (`!go_home`), de retorno para estação base. Dessa forma, conclui-se o plano da missão do agente.

O plano (`+!collect_data`) ocorre como descrito na seção 4.6.1.3 da parte proativa do agente. Esse plano é acionado pela intenção (`!collect_data`) do plano anterior.

O plano (`+charge(_)`) ocorre como descrito na seção 4.6.1.2 da parte reativa do agente. Esse é um plano atômico, que executa de modo que as demais intenções do agente são paradas até que este plano atômico seja concluído. Logo, isso garante a execução deste plano que é crítico para o sistema em questão. O outro plano que implementa um comportamento reativo está na linha 21 do programa 4.7 (`+charge(_)`) e ocorre de maneira semelhante ao anterior.

O plano (`+!select_sensor`), na linha 24, é usado para atingir o objetivo (`!select_sensor`). Neste plano ocorre a consulta à base de crenças do agente em busca das informações de altitude (`?altitude(Z)`) e de posição da aeronave (`?position(X0,Y0,Z0)`). Em seguida, faz-se a seleção do sensor mais próximo do VANT através da chamada da regra (`?closer(X0,Y0,ID,X,Y)`). Com a informação da posição do nodo sensor de destino (`X,Y,Z`), executa a remoção do ponto destino anterior e a inserção de um novo alvo com a posição do nodo em questão, através do comando (`-+next_point(X,Y,Z)`). Por fim, o agente executa a atuação de enviar este *waypoint* de destino (`go_to(X,Y,Z)`) para o **Nível de Controle**.

O plano (`+!go_home`), na linha 31, é usado para atingir o objetivo (`!go_home`). Neste plano ocorre a consulta à base de crenças do agente em busca da informação de posição da estação-base (`?base_station(X,Y)`). Com a informação da posição (`X,Y`), executa a remoção do ponto destino anterior e a inserção de um novo alvo com a posição da base, pelo do comando (`-+next_point(X,Y,0)`). Então, o agente executa a atuação de enviar este *waypoint* de destino (`go_to(X,Y,0)`) para o **Nível de Controle**. Em seguida, é criado o objetivo de chegar à posição (X,Y) (`!check_position`), para verificação da cheganda ao local de destino. Após este objetivo ser atingido, a intenção (`!finish`) é instanciada.

O plano (`+!emergency_landing`), na linha 38, é usado para atingir o objetivo (`!emergency_landing`). Neste plano ocorre a consulta à base de crenças do agente em busca da informação de posição atual do VANT (`?position(X,Y,Z)`). Depois, executa-se a remoção do ponto destino anterior e a inserção de um novo alvo (`-+next_point(X,Y,0)`). Então, o agente executa a atuação de enviar este *waypoint* de destino (`go_to(X,Y,0)`) para o **Nível de Controle**. Em seguida, o objetivo (`!check_position`) é criado, para verificação da chegada ao local de destino. Após este objetivo ser atingido, a intenção (`!finish`) é instanciada.

O plano (`+!check_position`), na linha 45, é usado para atingir o objetivo (`!check_position`). Este plano é verificado quando a seguinte condição é satisfeita: dada a posição atual da aeronave (`position(X0,Y0,Z0)`) e a posição de destino (`next_point(X,Y,Z)`), a distância entre esses pontos (`distanceXYZ(X0,Y0,Z0,X,Y,Z,D)`) deve ser menor que 0,1m (`D<0.1`). Dessa forma, tem-se a satisfação do objetivo. Mas enquanto essa condição não é satisfeita, o plano da linha 48 é executado, no qual tem-se a espera por uma nova percepção da posição atual do VANT (`.wait(+position(_,_,_))`) e em seguinda o

objetivo (`!check_position`) é criado novamente, ocorrendo uma nova chamada deste plano.

E o plano (`+!finish`), na linha 52, é usado para atingir o objetivo (`!finish`). Quando este plano é executado, ocorre a operação do comando (`.stopMAS`) que encerra o agente em questão, finalizando a missão.

4.6.2. Abordagem com Programação Imperativa

4.6.2.1. Implementação

A implementação do sistema de tomada de decisão na versão usual, com programação imperativa foi desenvolvida com o uso de *flags* de controle no código, através das quais é possível acompanhar o estado de execução do programa, tais como os estados da máquina de estados da especificação. O programa foi implementado com uma estrutura de seleção para execução dos estados, por meio de instruções `IF-THEN-ELSE` com chamadas de funções para executar uma tarefa específica. Todo programa foi escrito em linguagem C++, com o uso de classes e objetos.

A tabela 4.1, apresenta uma relação entre a *flag* `state` de controle de execução do programa e os estados da máquina de estados da especificação (figura 4.7).

Tabela 4.1: Relação da flag de controle de execução e o estado da aplicação.

Flag	Estado
state = 0	Start
state = 1	Select
state = 2	Check
state = 3	Go-Home
state = 4	Emergency
state = 5	Finish

Com isso, tem-se o programa como descrito no pseudo-código do programa 4.8. Esse processo é executado em um ciclo de repetição em um laço `while` de execução, onde em cada ciclo o sistema tenta receber uma nova mensagem com as atualizações das percepções com os sinais dos sensores. Caso o sistema não receba uma nova mensagem, o processo continua a sua execução normal com as últimas informações

obtidas do ambiente.

Programa 4.8: Pseudo-código da aplicação usual.

```
1  state = 0;
2  comeBack = 0;
3  while (true){
4       /* TENTA RECEBER MENSAGEM DE PERCEPCAO */
5       if (state == 0 && distance() < 0.1 && comeBack == 0){
6            state = 1; // (Start -> Select)
7       }
8       if (state == 1 && countSensor == TotalSensors && comeBack == 0){
9            comeBack = 1;
10           state = 2; // (Select -> Check)
11      }
12      if(state == 2){
13           if (distance() < 0.1){
14                if(comeBack == 0){
15                     state = 1; // (Check -> Select)
16                }else{
17                     state = 5; //(Check -> Finish)
18                }
19           }
20           if(countSensor == TotalSensors && comeBack == 1){
21                state = 3; // (Check -> Go-Home)
22           }
23           if(lowBattery == 0 && energy() == 0){
24                lowBattery = 1;
25                state = 3; //(Check -> Go-Home)
26           }
27           if(lowBattery == 1 && criticalLevel == 0 && emergency() == 1){
28                criticalLevel = 1;
29                state = 4; // (Check -> Emergency)
30           }
31      }
32      /* SELECT */
33      if (state == 1) {
34      // chamada da funcao de selecao do proximo sensor
35           state = 2; // (Select -> Check)
36      }
37      /* GO-HOME */
38      if (state == 3){
39           // chamada da funcao que define a rota para estacao-base
40           state = 2; (Go-Home -> Check)
41      }
42      /* EMERGENCY */
43      if (state == 4){
44           // chamada da funcao que define a rota pouso de emergencia
45           state = 2; (Emergency -> Check)
46      }
47      /* Finish */
48      if (state == 5){
49           // chamada da funcao que finaliza a missao
50      }
51 }
```

Na linha 4 do código, o sistema verifica a *flag* `state` de controle de execução, se a distância entre o ponto atual e o local de destino é menor que 0,1 metro, e a *flag* `comeBack` que diz se o VANT está

realizando a missão ou retornando para estação-base (*0 = missão*, *1 = retorno*) . No início da operação do sistema, tem-se a *flag* `state = 0` e a posição de referência coincide com a posição inicial da aeronave e o VANT está começando a missão. Logo ele passa do estado *Start* para o estado *Select*. Em outras condições de execução, com o sistema no estado *Check* e as condições do `IF` sendo verificadas, tem-se a transição do estado *Check* para o estado *Select*.

Na linha 7 do código, tem-se as condições referentes a *flag* `state` de controle de execução, a quantidade de sensores coletados e a quantidade total de nodos na rede, além da *flag* `comeBack`. Com o sistema no estado *Select*, o número de sensores coletados sendo igual ao total e o VANT estando em missão, tem-se que o estado de execução passa do estado *Select* para *Check* e a *flag* `comeBack` passa a ser igual a 1, retorno para base.

Na linha 11 do código, quando o sistema se encontra no estado *Check*, tem-se a verificação de quatro condições possíveis. Quando a distância entre o ponto atual e o local de destino é menor que 0,1 metros e a *flag* `comeBack` é igual a 1, o sistema passa do estado *Check* para o estado *Finish* ou se a *flag* `comeBack` é igual a 0, o sistema passa do estado *Check* para o estado *Select*, conforme a linha 12. Quando o número de sensores coletados é igual ao total e a *flag* `comeBack` é igual a 1, o sistema passa do estado *Check* para o estado *Go-Home*, conforme a linha 19. Quando a *flag* `lowBattery` é igual a 0 e a função de cálculo da energia da bateria retorna 0, o sistema também passa do estado *Check* para o estado *Go-Home* e a *flag* `lowBattery` passa a valer 1, conforme a linha 22. Quando a *flag* `lowBattery` é igual a 1, a *flag* `criticalLevel` é igual a zero, e a função de cálculo do nível crítico de bateria retorna 1, o sistema passa do estado *Check* para o estado *Emergency* e a *flag* `criticalLevel` passa a valer 1, conforme a linha 26.

Na condição *flag* `state = 1`, linha 32, tem-se o sistema no estado *Select*, no qual é executada a função de seleção do próximo nodo sensor da rede e o envio deste *waypoint* de destino para o **Nível de Controle**. Além disso, o sistema passa do estado *Select* para *Check*.

Para a condição *flag* `state = 3`, linha 37, tem-se o sistema no estado *Go-Home*, no qual é selecionada a estação-base como sendo o local de destino e ocorre o envio deste *waypoint* para o **Nível de Controle**. Além disso, o sistema passa do estado *Go-Home* para *Check*.

Já na condição *flag* `state = 4`, linha 42, tem-se o sistema no estado *Emergency*, no qual é selecionado um ponto mais próximo do local que se encontra a aeronave para pouso de emergência e ocorre

o envio deste *waypoint* de destino para o **Nível de Controle**. Além disso, o sistema passa do estado *Emergency* para *Check*.

Por fim, na condição *flag* `state = 5`, linha 47, tem-se o sistema no estado *Finish*, no qual é executada a função de finalização da missão.

4.7. Resultados Preliminares

Assim, percebeu-se que o a abordagem com agentes apresenta como características uma linguagem descritiva, expressiva em muitos aspectos, permitindo a construção de expressões de código mais inteligíveis e curtas. Dessa forma, a programação dos comportamentos autônomos do VANT se dá em um nível de abstração que permite ao desenvolvedor trabalhar apenas o comportamento desejado em projeto, sem tratar das implementações na arquitetura de baixo nível. Enquanto a abordagem usual possui uma programação que permite o acesso direto ao hardware do dispositivo, permitindo a implementação das funções do VANT no nível da aplicação.

A versão usual possui as mesmas estruturas de dados (como as crenças) da versão com agentes, além das mesmas funcionalidades de atuação no VANT, buscando atingir o mesmo comportamento desejado na missão. Porém, a abordagem usual apresenta também outras estruturas de dados, como o uso das variáveis de flags na sua programação, que auxiliam no controle do fluxo de execução do processo, e também outras variáveis utilizadas como contadores e referentes ao estado do sistema.

5 Resultados e Análises

Este capítulo descreve alguns resultados obtidos com o projeto e realiza uma análise comparativa das abordagens utilizadas, com o uso de um agente BDI e outra usual de programação imperativa com orientação a objetos.

A organização do capítulo é feita em três seções principais, como descrito a seguir. A seção 5.1 mostra como foram feitas simulações para obteção dos resultados. A seção 5.2 apresenta uma análise qualitativa apresentada com objetivo de analisar a viabilidade do uso de um agente BDI em um VANT. Além disso, foi analisada a diferença em inserir nova funcionalidade ao sistema, bem como a modelagem do problema utilizada em cada abordagem. E a seção 5.3 descreve a análise quantitativa que apresenta os resultados de um teste do tempo de uso do processador por cada abordagem e do tamanho do código fonte gerado.

5.1. Resultados

Como resultado deste trabalho, obteve-se duas versões do sistema de tomada de decisões e planejamento de rota de um robô móvel autônomo. Esses sistemas foram implementados no sistema embarcado de um VANT e testados através de um modelo de simulação HIL que descreve o comportamento da aeronave.

A missão do VANT nas simulações foi de percorrer a área da região de monitoramento onde estão os nodos da rede de sensores sem fio, visitar cada um deles e retornar para estação-base, levando em consideração as variações do ambiente durante a sua execução.

Quando o VANT consegue alcançar todos os nodos da rede e retornar para base, tem-se cumprida a missão completa, como ilustrado na figura 5.1.

Outra condição testada em simulação com o agente foi quando o VANT recebe a incidência de um vento contrário ao seu deslocamento, fazendo com que o seu consumo de energia durante o voo aumente.

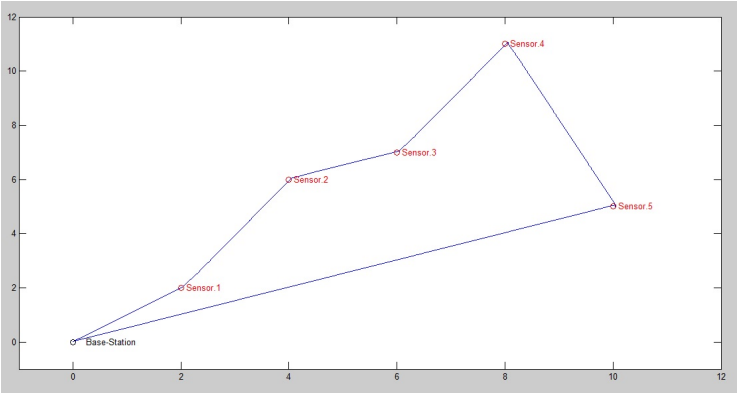

Figura 5.1: Simulação da missão completa do VANT.

Dessa forma, o agente percebe que o nível da carga da bateria está em uma condição inferior a um limite de segurança e que a aeronave não conseguirá visitar todos nodos sensores. Neste caso, o agente decide retornar à estação-base de modo seguro para recarregar, somente com uma parte dos sensores sendo visitados. Esse caso é ilustrado na figura 5.2 com a missão incompleta, onde o VANT visita apenas os sensores 1, 2, 3 e 4. E ao se deslocar para o sensor 5, decide abortar a missão e voltar para base.

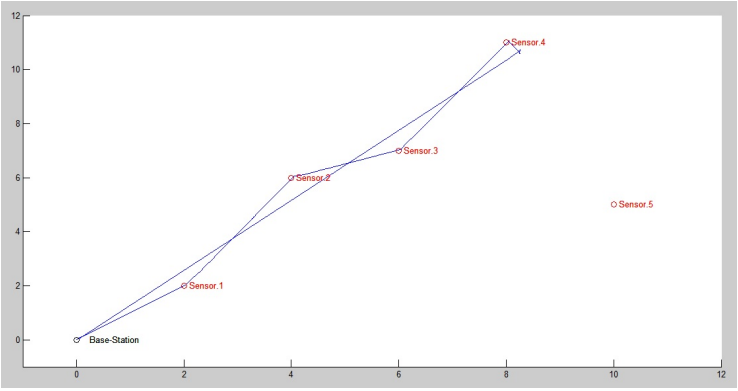

Figura 5.2: Simulação da missão incompleta do VANT.

5.2. Análise Qualitativa

A análise qualitativa busca fazer uma análise no que diz respeito ao projeto e desenvolvimento do sistema com uso de uma linguagem de agentes em uma aplicação de sistemas embarcados, que geralmente utiliza uma abordagem usual de programação e possui restrições de hardware e processamento.

5.2.1. Análises iniciais do uso das abordagens

O desenvolvimento do agente BDI foi inicialmente baseado na implementação da abordagem usual, com a criação de flags de estado como crenças do agente. Porém, após a familiarização com a linguagem, foram realizadas melhorias e refinamentos no código, sendo este processo bastante custoso com relação ao tempo de projeto, até se chegar a uma implementação baseada em objetivos e não em estados.

O uso de uma linguagem de programação de agentes BDI facilita o desenvolvimento de agentes que necessitem de capacidade de reagir rapidamente a mudanças no ambiente (como exemplifica o programa 4.3), bem como possuir objetivos de longo prazo em uma aplicação (como exemplifica o programa 4.4), conforme empregado na implementação dos planos do agente no sistema. Já uma abordagem usual apresenta dificuldade de se definir comportamentos proativos na modelagem em máquina de estados, além disso, o desenvolvimento desse tipo de comportamento requer a implementação de processos concorrentes, em vez de um processo com fluxo sequencial, levando a uma programação mais complexa. Por outro lado, máquinas de estados são amplamente conhecidas e possuem várias ferramentas de análise, além de serem rapidamente e eficientemente implementadas. Além disso, a abordagem usual e a programação imperativa são mais comumente utilizadas pela maioria dos projetistas que possuem familiaridade com esse tipo de paradigma.

Nota-se também que no uso de uma linguagem de programação de agentes, como Jason, muitas vezes, tem-se a necessidade do desenvolvimento de uma infraestrutura de integração para aplicações embarcadas. Visto que estas aplicações são desenvolvidas, geralmente, em linguagem C.

5.2.2. Diferença de programação

Esse ponto de análise foi realizado baseando-se na diferença de programação entre as abordagens para se inserir um novo comporta-

mento ou funcionalidade no sistema do VANT. Dessa forma, pensou-se no cenário de aplicação já apresentado e na questão da aeronave realizar um pouso de emergência por apresentar um nível crítico de bateria.

Na abordagem com agentes BDI, a inserção de um novo comportamento ou alteração de uma funcionalidade na aplicação é realizada através da criação de um novo plano, sem mexer no que já existe no código, como pode ser notado pela independência entre os códigos dos programas 4.3 e 4.4.

Já na abordagem usual de programação, desenvolvida com uma estrutura de seleção baseada nos estados do sistema, a inserção de um novo comportamento ou alteração de uma funcionalidade na aplicação é realizada através da criação de uma nova condição de execução do sistema. Além disso, deve-se verificar a lógica para que não haja conflito com as demais condições do sistema para garantir o fluxo de todos os estados de execução durante o processo. Logo, por serem paradigmas diferentes, apresentam características específicas que interferem no modo como programar o comportamento desejado para a aplicação.

5.2.3. Modelagem

Para análise da modelagem do problema, tem-se que cada abordagem modela o comportamento do sistema de uma forma particular, sendo que a Máquina de Estados especifica a sequência de estados pelos quais um processo passa durante o tempo de execução em resposta aos eventos, enquanto o Diagrama de Objetivos do Sistema descreve os objetivos do sistema durante a execução do processo.

Por outro lado, comportamentos sequenciais podem ser modelados tanto em uma como em outra abordagem, porém, os comportamentos proativos são mais dicíficeis de serem modelados em Máquinas de Estados. Enquanto comportamentos complexos e concorrentes podem ser modelados com o Diagrama de Objetivos do Prometheus AEOlus.

5.3. Análise Quantitativa

A análise qualitativa busca avaliar o desempenho das versões do sistema, visando comparar seus rendimentos em uma aplicação real. Duas formas de análise foram utilizadas para avaliar os sistemas desenvolvidos: o tempo de utilização da CPU e o tamanho do código fonte gerado por cada abordagem.

Para a análise do uso do processador pelo sistema, foram realizadas simulações do modelo HIL com o uso do sistema de tomada

de decisão nas versões com agente e usual. Nessas simulações, foi explorado o cenário de aplicação com 5 nodos sensores, com o VANT tendo que percorrer essa área, visitar todos os *waypoints* e retornar para estação-base.

5.3.1. Utilização da CPU

O tempo de utilização do processador pelo sistema de tomada de decisão no **Nível de Planejamento** do VANT foi medido através do comando *time* do sistema operacional Linux. Esse comando informa o tempo de execução de um processo específico.

Este teste do tempo de utilização da CPU consiste em dado o tempo total da execução da simulação, com o VANT percorrendo todos os sensores e retornando para estação-base, medir o tempo que o processo de planejamento faz uso do processador para executar as suas tarefas. Dessa forma, com base no tempo total de simulação e no tempo gasto pelo planejamneto, obtem-se um percentual de uso da CPU pelo processo. Os outros processos também são executados na mesma plataforma em um processador unicore, sendo eles os processos do **Nível de Controle**, com os controles de rotação e translação, envio e recebimento de mensagens para o modelo da planta (VANT) e para o **Nível de Planejamento**.

Os primeiros testes foram realizados utilizando o envio da percepção para o **Nível de Planejamento** a cada ciclo do processo de controle, que opera a cada 5 ms. Com isso, o agente sempre tinha uma nova informação para atualizar em sua base de crenças a cada período de operação do controle.

Nestes testes, a abordagem com agente BDI obteve 61% de tempo de utilização da CPU e a abordagem usual atingiu 4%. Analisando esse resultado, buscou-se o gargalo nessa operação e percebeu-se que o agente utilizava muito do seu tempo para fazer percepção, já que a frequência de atualização das informações era alta (a cada 5 ms).

Nos testes seguintes, foi utilizada a comunicação com o envio da percepção para o **Nível de Planejamento** somente após uma mudança significativa na posição do VANT, baseado em testes para o cenário de aplicação descrito para os ensaios, sendo o valor de deslocamento da aeronave maior que 0,1 m. Com isso, obteve-se uma redução do tempo de uso da CPU, tanto para a abordagem com o agente como a usual, visto que o sistema já não realizava a percepção constantemente.

Foram realizados experimentos com 10 amostras para cada abordagem, os resultados do agente Jason obteve Média = 15,15 % e Desvio

Padrão = 1,12 - conforme a tabela 5.1. Já a abordagem usual obteve Média = 0,83 % e Desvio Padrão = 0,23 - conforme a tabela 5.2. Assim, a solução com agente obteve um desempenho satisfatório, o que possibilita a sua execução numa aplicação em plataforma embarcada sem comprometer o desempenho do hardware do veículo.

Dessa forma, o sistema com a abordagem de agentes em linguagem Jason usa o processador em média 18,2 vezes mais que o sistema com a abordagem usual em linguagram C++.

Tabela 5.1: Tabela de resultados: Abordagem com Agente.

Tempo Total (s)	Tempo do Processo (s)	Uso da CPU (%)
119,54	17,61	14,73
111,79	18,90	16,91
127,65	17,45	13,67
119,07	18,26	15,34
128,31	18,38	14,32
115,90	18,13	15,64
139,29	18,64	13,38
108,12	17,26	15,97
117,97	18,19	15,42
117,71	19,08	16,21
	Média = 15,16%	Desvio Padrão = 1,13%

5.3.2. Tamanho do Código

Outro ponto analisado foi o tamanho do código final em cada uma das abordagens utilizadas [CARDOSO, 2012] [WESZ, 2015]. Como a quantidade de linhas no código fonte de um programa depende do estilo de programação do desenvolvedor, optou-se por utilizar o comando *gzip* para compactar o arquivo fonte dos programas (sem comentários) e verificar o seu tamanho em bytes.

Além disso, outra métrica que também analisada foi a contagem do número de identificadores[1] utilizados no código fonte de cada uma das abordagens em seus programa.

Dessa forma, tem-se que o código do agente Jason possui 0,700 Kbytes e 81 identificadores, enquanto o código da abordagem tradi-

[1] A contagem do número de identificadores no código Jason foi realizada considerando as regras, crenças, intenções e variáveis utilizadas e consultadas.

Tabela 5.2: Tabela de resultados: Abordagem Usual.

Tempo Total (s)	Tempo do Processo (s)	Uso da CPU (%)
108,53	0,76	0,70
74,47	0,77	1,04
78,28	0,68	0,87
72,89	0,20	0,28
77,19	0,73	0,95
79,09	0,52	0,66
81,13	0,78	0,96
72,82	0,70	0,96
71,82	0,66	0,91
76,91	0,77	1,00
	Média = 0,83%	Desvio Padrão = 0,23%

cional tem 2,164 Kbytes e 165 identificadores. Por ter um nível de abstração maior e utilizar a programação lógica, o código do agente é 67,65% menor em relação ao tamanho que o da versão usual e 50,1% menos identificadores no código fonte.

5.4. Considerações

Os resultados dos testes de simulação HIL foram satisfatórios para as análises e comparações prentendidas. Assim, percebeu-se que o uso de uma abordagem com agentes em uma aplicação de planejamento de rota, embarcada em uma plataforma unicore, obteve 67,65% menos código que uma versão usual com programação imperativa, a um custo 18,2 vezes maior do uso do processador.

Além disso, uma abordagem com agente BDI apresenta uma alternativa para a programação de comportamentos concorrentes, reativos e proativos, em aplicações de tomada de decisão para robôs móveis, bem como apresenta uma maior facilidade para manutenção e inserção de novos objetivos e funcionalidades ao sistema.

Outro ponto importante dos resultados foi a importância da melhoria da percepção do agente, que impactou consideravelmente no custo computacional da solução. Visto que não foi necessária uma atualização em um período tão curto quanto o tempo de operação do processo de controle, mas sim por meio de uma variação significativa no deslocamento da aeronave para o agente, baseada em seu compor-

tamento.

Outras melhorias no código poderiam ser realizadas visando um desempenho superior para o sistema, tais como, o envio das mensagens para o agente somente quando este solicitar. Mas como testado no outro caso, deve-se analisar o seu impacto na aplicação durante a execução das simulações com o modelo HIL e como essa mudança interfere no sistema.

6 Considerações Finais

6.1. Conclusões

Conclui-se que é possível a execução de um agente BDI embarcado em um VANT, tendo em vista a redução do percentual do tempo de uso da CPU apresentado nas análises e testes através de um modelo de simulação HIL. Embora necessite de uma infraestrutura computacional para integração com o sistema embarcado do robô móvel, a solução com agente é viável para aplicações de tomada de decisão e pode ser embarcada em um robô autônomo.

O trabalho trouxe como contribuição o indicativo de pontos importantes no projeto de aplicações com agentes embarcados em plataformas que possuem restrições de hardware, como o gargalo das percepções do agente. Assim, apresentou-se uma forma de reduzir esse tempo com uma melhoria para um cenário de teste utilizado. Também apresentou um indicativo de tempo gasto e do ganho com o uso dessa abordagem. Além disso, mostrou que é possível desenvolver uma aplicação com o software Jason embarcado em um hardware restrito, como a *beaglebone*, visto que muitas das aplicações com agentes são desenvolvidas em computadores de elevado nível processamento.

O projeto também gerou um modelo de comportamento para tomada de decisão e planejamento de rota de um VANT autônomo, bem como a integração junto a plataforma computacional da aeronave. Apresentou algumas potencialidades da arquitetura BDI em aplicações com VANTs, avaliando as vantagens e desvantagens do uso desse tipo de agente para um cenário de teste, podendo essa solução ser utilizada e explorada em outros cenários da robótica móvel, tais como veículos terrestres, aquáticos, entre outros.

Também gerou desenvolvimento do artigo SANTOS, F. R.; HÜBNER, J. F.; BECKER, L. B. Modelo de VANT autônomo baseado em uma arquitetura BDI. **Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC)**, 2014. Esse ar-

tigo aborda a proposta de desenvolvimento de um modelo de comportamento autônomo para VANTs através da arquitetura agentes BDI, abordando o problema da qualidade de código, bem como a análise das diferenças em relação à outras técnicas de implementação.

Outra publicação gerada foi o artigo SANTOS, F. R.; HÜBNER, J. F.; BECKER, L. B. Concepção e análise de um modelo de agente BDI voltado para o planejamento de rota em um VANT. **Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC)**, 2015. Essa publicação apresenta o modelo de comportamento com um agente BDI para um VANT autônomo, explorando sua capacidade de reagir rapidamente a mudanças em seu ambiente e de ter objetivos de longo prazo, além da implementação do agente no sistema embarcado de um VANT real e uma comparação deste sistema com uma abordagem usual de programação imperativa.

6.2. Trabalhos Futuros

Um ponto a ser explorado neste trabalho é a realização de novos testes com o uso de baterias para alimentação do sistema computacional embarcado do VANT, analisando o consumo e a influência no comportamento do agente.

Para dar continuidade nesse projeto, pretende-se realizar testes do modelo proposto com o VANT real. Também pretende-se extender a aplicação para um cenário mais completo de coleta de dados em uma rede de sensores sem fio. Além disso, almeja-se o desenvolvimento de um *middleware* genérico, independente de contexto, que possa ser utilizado em diversas aplicações com agentes e robôs.

Outro ponto que deseja-se explorar é a identificação de padrões de programação gerais no trabalho proposto, definindo alguns modelos de planos comuns para uso em aplicações de robótica móvel com agentes.

Referências

BERGENTI, F.; GLEIZES, M.; ZAMBONELLI, F. **Methodologies and software engineering for agent systems: the agent-oriented software engineering handbook**. 3. ed. [S.l.]: Boston: Kluver Academic Publishers, 2004.

BORDINI, R. H.; HÜBNER, J. F.; WOOLDRIDGE, M. **Programming Multi-Agent Systems in AgentSpeak using Jason (Wiley Series in Agent Technology)**. [S.l.]: Wiley-Interscience, 2007.

BRATMAN, M. E. **Intention, Plans, and Practical Reason**. [S.l.]: Harvard University Press, 1987.

CAI, G.; CHEN, B.; LEE, T.; DONG, M. Design and implementation of a hardware-in-the-loop simulation system for small-scale uav helicopters. **Automation and Logistics, 2008. ICAL 2008. IEEE International Conference**, 2008.

CARDOSO, R. C. **Programação Orientada a Agentes com Aplicações em Robótica**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul, 2012.

CHAVES, q. N. **Proposta de Modelo de Veículos Aéreos Não Tripulados (VANTs) Cooperativos Aplicados a Operações de Busca**. Dissertação (Mestrado) — USP - Universidade de São Paulo, 2013.

CHEN, X.; HE, W.; WU, Z. Research of UAV's multiple routes planning based on multi-agent particle swarm optimization. **Fourth International Conference on Intelligent Control and**

Information Processing (ICICIP), 2013.

CORRÊA, M. **Modelo de Veículos Aéreos Não Tripulados Baseado em Sistemas Multi-agentes**. Tese (Doutorado) — USP - Universidade de São Paulo, 2008.

DONADEL, R.; RAFFO, G. V.; BECKER, L. B. Modeling and control of a tiltrotor uav for path tracking. **19th World Congress The International Federation of Automatic Control**, 2014.

FAHLSTROM, P. G.; GLEASON, T. J. **Introduction to UAV Systems**. [S.l.]: Wiley, 2012.

GONÇALVES, F. S. **Projeto da Arquitetura de Software Embarcado de um Veículo Aéreo Não Tripulado**. Dissertação (Mestrado) — UFSC - Universidade Federal de Santa Catarina, 2014.

HAMA, M. T. **Uma Plataforma Orientada a Agentes para o Desenvolvimento de Software em Veículos Aéreos Não-Tripulados**. Dissertação (Mestrado) — UFRGS - Universidade Federal do Rio Grande do Sul, 2012.

HAN, J.; WANG, C.-h.; YI, G.-x. Cooperative control of uav based on multi-agent system. **IEEE**, 2013.

LOUALL, R.; BELLOULA, A.; DJOUADI, M.; BOUAZIZ, S. Real-time characterization of microsoft flight simulator 2004 for integration into hardware in the loop architecture. **Control Automation (MED) - 19th Mediterranean Conference**, 2011.

RAFFO, G. V.; RICO, J. E. N. Robótica móvel. In: _____. [S.l.]: LTC, 2014. cap. Controle de robôs móveis para seguimento de trajetórias, p. 113–138.

RAO, A. S. Agentspeak(l): BDI agents speak out in a logical computable language. In: **Proceeding of the Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)**. [S.l.]: Springer, 1996. p. 42–55.

RAO, A. S.; GEORGEFF, M. P. Bdi agents: from theory to practice. **Proceedings of the First International Conference on MultiAgent Systems**, 1995.

RUSSELL, S.; NORVING, P. **Artificial Intelligence: A Modern Approach**. 3. ed. [S.l.]: Prentice Hall, 2009.

SANTOS, F. R.; HÜBNER, J. F.; BECKER, L. B. Modelo de VANT autônomo baseado em uma arquitetura BDI. **Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC)**, 2014.

SANTOS, F. R.; HÜBNER, J. F.; BECKER, L. B. Concepção e análise de um modelo de agente BDI voltado para o planejamento de rota em um VANT. **Workshop-Escola de Sistemas de Agentes, seus Ambientes e Aplicações (WESAAC)**, 2015.

SELECKY, M.; MEISER, T. Integration of autonomous UAV's into multi-agent simulation. **Acta Polytechnica**, v. 52, n. 5, 2012.

SHIXIANJUN; JIANKUN, S.; HONGXING, L. Hardware-in-the-loop simulation framework design for a uav embedded control system. **Control Conference**, 2006.

UEZ, D. M. **Método para o Desenvolvimento de Software Orientado a Agentes Considerando o Ambiente e a Organização**. Dissertação (Mestrado) — UFSC - Universidade Federal de Santa Catarina, 2013.

WESZ, R. B. **Integrating Robot Control into the AgentSpeak(L) Programming Language**. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul, 2015.

WOOLDRIDGE, M. **An Introduction to MultiAgent Systems**. [S.l.]: Chichester: John Wiley and Sons Ltd, 2002.

WOOLDRIDGE, M.; JENNINGS, N. R. **Intelligent agents: theory and pratice**. [S.l.]: The Knouledge Engineering Review, 1995.