Um Agente Inteligente para Simulação de Voo Usando Jason e X-Plane

Tielle da Silva Alexandre, Carlos Eduardo Pantoja

[1] CEFET/RJ - Campus Nova Friburgo
Av. Gov. Roberto da Silveira, 1900 – Prado – 22.635-000 –
Nova Friburgo – RJ – Brasil

tiellesa@gmail.com, pantoja@cefet-rj.br

Abstract. *This paper presents an intelligent agent which connected to a professional flight simulator, performs the pilot function in the task of controlling an airplane to achieve the missions of takeoff; reach determined point based on the latitude longitude and altitude; and remain stable during the flight. This work also presents an integration between Jason framework and the X-Plane flight simulator.*

Resumo. *Este trabalho apresenta um agente inteligente que, conectado a um simulador de voo profissional, realizará a função do piloto na tarefa de controlar uma aeronave na missão de levantar voo; atingir determinado ponto levando em consideração a latitude, longitude e altitude; e se manter estável durante o voo. O trabalho também apresentará uma integração entre o framework Jason e o simulador de voo X-Plane.*

1. Introdução

Um agente inteligente pode ser tanto físico como virtual e tem a capacidade de atuar em um ambiente, que pode ser físico, como um robô atuando através de efetuadores, ou simulado, atuando através de execução de comandos de software. Além disso, possui objetivos de satisfação e subsistência individuais, pode se comunicar com outros agentes, disponibilizar serviços e tem pró-atitude por satisfazer seus objetivos [Wooldridge 2009].

Um agente pode possuir crenças, que são as ideias que este tem do ambiente e sua compreensão da atmosfera em que ele está presente. Os desejos são os eventuais planos que o agente realiza a partir de suas crenças, entretanto, o agente só irá atuar sobre algum plano se estiver comprometido em atingir determinado objetivo, apesar de desejá-lo. Já as intenções se dão quando um agente se compromete em seguir um plano para atingir determinado objetivo [Bratman 1987].

Os Veículos Aéreos Não Tripulados (VANT) têm sido cada vez mais utilizados em diversas áreas de conhecimento, com o objetivo de auxiliar em tarefas onde haja riscos a vidas humanas. Primeiramente idealizados para realizarem missões de estratégia militar, hoje os VANT realizam também operações de monitoramento agrícola, ambiental, segurança e defesa civil. Um VANT não necessita de pilotos embarcados e pode ser guiado à distância, por meio eletrônico ou computacional, manipulado especificamente por pilotos treinados ou serem completamente autônomos.

Neste processo, visto que a pilotagem da aeronave é feita a distância, pode-se detectar problemas como o de comunicação ou de falha humana que podem comprometer o voo e a missão imposta à aeronave. Além disso, a decolagem e aterrissagem são ações críticas que demandam experiência do piloto. Atualmente, mesmo em modelos com piloto automático acionado por controle remoto a determinada altura, a decolagem e aterrissagem ainda é um processo unicamente manual e dependente de um piloto qualificado, como o AG PLANE da AGX Tecnologia.

O VANT necessita ainda de uma estação de controle, podendo essa ser portátil ou móvel. Englobando todos os equipamentos para se operar um VANT, um defeito ou até mesmo a falta de energia na estação de controle compromete toda uma operação. Dessa forma, um VANT inteligente seria uma solução para se contornar tais problemas. Contudo, devido o alto custo do projeto de construção de aeronaves, faz-se necessária uma fase de simulação e testes.

Diversos trabalhos utilizam o paradigma orientado a agentes e visam a autonomia de um VANT, como o caso do [Wallis et al. 2002], que utiliza a linguagem orientada a agentes JACK [Winikoff 2005] para prover um ambiente de programação simples para comportamentos táticos de voo; e [Huff et al. 2003], onde um simulador representa um VANT como um conjunto de agentes que podem simular diferentes abordagens de planejamento de voo.

O objetivo deste trabalho é desenvolver um agente de software capaz de realizar um voo de maneira autônoma em um simulador profissional, utilizando a comunicação de pacotes UDP para: receber informações acerca do ambiente através de crenças; e poder executar ações no simulador através de comandos para a aeronave simulada. Para implementação do agente serão utilizados: o simulador de voo profissional X-Plane, para simulação do agente inteligente; uma API [Cantoni 2010] em Java para comunicação entre o simulador de voo e o ambiente simulado do agente; e, por fim, o framework Jason para a codificação do agente inteligente.

Este artigo está estruturado da seguinte forma: na seção 2 apresenta-se a metodologia utilizada na integração das tecnologias utilizadas no trabalho; na seção 3 é apresentada a implementação do ambiente e do agente inteligente e também serão apresentados os resultados obtidos através da integração com o simulador de voo; na seção 4 serão apresentados alguns trabalhos relacionados; e por fim, na seção 5 é apresentada a conclusão e alguns trabalhos futuros.

2. A Metodologia

Nesta seção serão apresentados os conceitos básicos sobre as tecnologias utilizadas no desenvolvimento do projeto, onde, o ambiente simulado utilizado é o simulador X-Plane integrado a linguagem de programação Java e para a implementação do agente é utilizado o framework Jason. Além disso, será apresentada a metodologia de integração entre o agente e o simulador de voo.

O X-Plane é um simulador de voo de alta realidade, que pode ser usado para prever as qualidades de voo de aeronaves de asa fixa e rotativa com precisão. O simulador prevê o desempenho e manuseio de quase qualquer aeronave, dessa forma é uma ferramenta útil para pilotos treinarem e se aperfeiçoarem, para engenheiros testarem o comportamento

aéreo de novas aeronaves, bem como a exploração da dinâmica de voo de aeronaves. O X-Plane foi escolhido, pois é um simulador de alta precisão de aeronaves comercias, assim como é utilizada para testes de comportamento aéreo e das dinâmicas de voos de protótipos em fase de estudo.

O Jason é um framework baseado em Java e AgentSpeak para desenvolvimento de sistemas multiagentes onde a linguagem AgentSpeak representa uma tentativa de refinar as características principais da arquitetura BDI. O Jason foi escolhido por ser um framework orientado a agentes que utiliza uma arquitetura cognitiva, além de ser uma linguagem de licença livre [Bordini et al. 2007].

O agente inteligente usando o framework Jason deve ser capaz de se comunicar com o simulador de voo através de comunicação de pacotes UDP em uma rede de computadores. A comunicação UDP permite tanto que o agente inteligente tenha acesso às variáveis relacionadas ao ambiente e a aeronave como crenças, que serão usadas na deliberação das ações de voo, quanto no envio de instruções para o simulador, com a finalidade de controlar as funções da aeronave. A figura 1 ilustra a metodologia de comunicação entre o agente e o simulador.

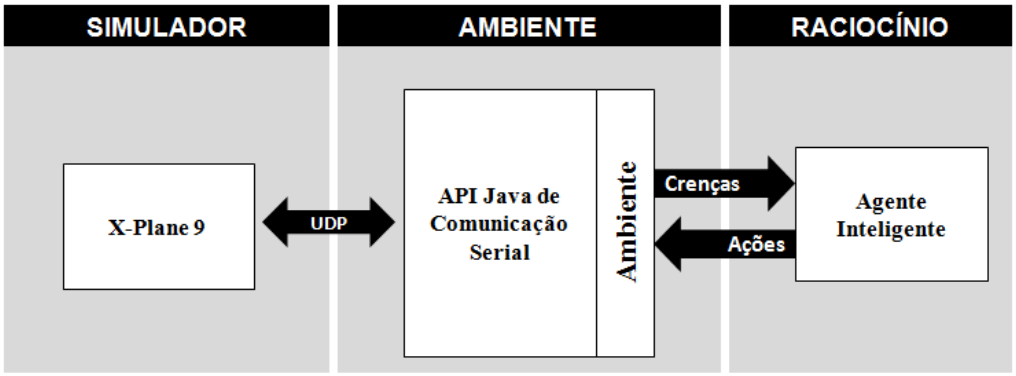

Figure 1. A metodologia de integração entre as tecnologias.

A integração entre o framework Jason e o X-Plane funciona através da importação de uma API em Java para comunicação Serial na classe de ambiente do Jason, que é responsável por receber os pacotes UDP vindos do simulador, interpretá-los e enviá-los ao ambiente do agente, onde serão transformados em crenças. Da mesma forma, o ambiente identifica as ações externas do agente transformando-as em comandos, que são enviados ao simulador de voo através de pacotes UDP.

3. A Implementação

Nesta seção será apresentada a implementação do agente inteligente em Jason, que é capaz de pilotar uma aeronave modelo P180 Avanti Ferrari e possui um plano de voo que é responsável por identificar variáveis de aceleração, posicionamento global, velocidade do vento, para que seja possível um posicionamento adequado na pista para realizar a decolagem até uma altitude pré-estabelecida. Após atingir tal altitude, o agente inteligente estabiliza a aeronave.

Foi elaborada a conexão entre os aplicativos Jason e X-Plane, de forma que a partir da classe Java utilizada como ambiente fosse possível acessar os dados do simulador,

obtendo informações como velocidade, latitude, longitude, altura e as angulações dos vetores do plano cartesiano transmitindo-as como crenças para o agente.

Para que seja possivel realizar uma integração com o ambiente utilizado pelo Jason é necessário instanciar a classe *XPlaneJasonComm*, que é responsável pela comunicação com o simulador através da interpretação dos pacotes UDP, dentro da classe do ambiente do agente. A classe *XPlaneJasonComm* foi adaptada para ser uma *Thread*, para que sempre que forem requisitadas as informações do simulador, esta esteja com os dados mais recentes. Nessa mesma classe são identificadas quais são as informações que serão transmitidas do simulador ao ambiente como percepções. A figura 2 exibe o código relativo ao ambiente do agente.

```
public class XPlaneEnv extends Environment {
    private float lat, lon, alt, speed, vY, joy_elev, joy_ailrn, joy_ruddr, joy_vect;
    private XPlaneJasonComm xData;

    public XPlaneEnv(){
        super();
        xData = new XPlaneJasonComm();
        xData.start();
    }

    public void init(String[] args){
        try {
            updatePercept();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

Figure 2. O ambiente do agente com a integração a API de comunicação UDP.

Além da modificação no ambiente do agente e na classe *XPlaneJasonComm*, é necessário configurar o simulador identificando o IP da máquina que hospeda o agente e quais as informações estão habilitadas para serem transferidas pelo protocolo UDP. O simulador permite a conexão de diversas máquinas com diferentes IPs ao mesmo tempo, possibilitando voos em missão conjunta. As informações que representam as crenças são criadas como atributos na classe do ambiente do agente e são atualizadas pelo simulador de voo. Em seguida o método *updatePercept* adiciona as informações como crenças ao agente especificado. O codigo referente ao método pode ser visto na figura 3.

```
public void updatePercept() throws InterruptedException {
    clearPercepts("plane");
    this.lat = xData.getLat();
    this.lon = xData.getLon();
    this.alt = xData.getAlt();
    this.speed = xData.getSpeed();
    this.vY = xData.getvY();
    this.vX = xData.getvX();
    this.joy_elev = xData.getElev();
    this.joy_ailrn = xData.getAilrn();
    this.joy_ruddr = xData.getRuddr();
    this.joy_vect = xData.getVect();
    addPercept("plane",Literal.parseLiteral("lat("+this.lat+")"));
    addPercept("plane",Literal.parseLiteral("lon("+this.lon+")"));
    addPercept("plane",Literal.parseLiteral("alt("+this.alt+")"));
}
```

Figure 3. As crenças do agente baseadas nas informações do simulador de voo.

Com isso, a integração é possível ser executada através da programação das ações externas do agente no ambiente do Jason e em seguida programar a ação relativa que o agente deseja realizar no simulador. Para isso é necessário invocar o método *setValue* da classe *XPlaneJasonComm* que enviará, através pacotes UDP, a informação e o valor a ser modificado no simulador.

Inicialmente o avião começa acelerando exponencialmente mantendo as coordenadas de longitude de forma a permanecer na pista, ao aproximar-se do fim da pista é levantado voo em uma angulação de 20° e mantendo até atingir 1500 pés de altitude. Dada esta altura, a aeronave estabiliza-se com o plano horizontal.

As intenções iniciais do agente são *getInitialPosition* e *startMission*. A intenção *startMission* só é executada com a condição da crença *missionStarted* não estiver presente na base de crenças do agente. Após serem obtidas a crenças da posição inicial é dado o início a missão soltando os freios e acelerando a aeronave como se pode observar na figura 4.

```
/* Initial goals */
!getInitialPosition.
!startMission.

/* Plans */
+!getInitialPosition: not missionStarted <-
    .print("Pegando a Posição Inicial.");
    getPosition.

+!startMission: not missionStared <-
    +missionStarted;
    .print("Missão Iniciada.");
    throttle;
    .print("Acelerando.");
    !atPosition.
```

Figure 4. Objetivos iniciais do agente.

A partir de então é mantida a ação *throttle*, acelerando a aeronave e movendo-se somente na latitude para que a aeronave não saia da pista. A ação *getPosition* atualiza as crenças do agente com a posição atual da aeronave no simulador. Quando é percebida a latitude que representa a proximidade do final da pista, é acionado o plano *flying* que atua alterando o eixo do avião ao plano horizontal em 20° e mantendo a aceleração. Já o plano *!atPosition* verifica se o ponto de decolagem foi atingido, enquanto isso o agente continua acelerando. A figura 5 mostra os planos *!atPosition* do agente.

```
!atPosition: missionStarted & lat(Y) & lon(X) & Y <= 41.334 <-
   getPosition;
   centralize;
   !atPosition.

!atPosition: missionStarted & lat(Y) & lon(X) & Y > 41.334 <-
   .print("Levantando Voo.");
   !flying.

!atPosition: true <-
   !!atPosition.
```

Figure 5. O plano atPosition.

O plano *flying* é responsável por verificar se a aeronave chegou ao ponto de se estabilizar. Para isso o plano *flying* analisa se a posição da aeronave está entre 1000 e 1500 pés e enquanto essa valoração não for ultrapassada, o plano continua executando e atualizando a posição para o agente. Após atingir a altura desejada, o plano *stabilizing* é acionado com o objetivo de estabilizar a aeronave, centralizar e acionar o piloto automático da aeronave. A figura 6 exibe o plano *flying* e *stabilizing*.

```
+!flying: missionStarted & alt(A) & A < 1000<-
    .print("Subindo até 1500. Altitude atual:", A);
    getPosition;
    getup;
    !flying.

+!flying: missionStarted & alt(A) & A >= 1000 & A < 1500 <-
    .print("Valor atual:", A);
    getPosition;
    !flying.

+!flying: missionStarted & alt(A) & A >= 1500 <-
    !stabilizing.

+!stabilizing: missionStarted  <-
    .print("Piloto Automático Ativado.");
    centralize;
    stabilize;
    .wait(60000).
```

Figure 6. Os planos flying e stabilizing

3.1. Resultados

Esta seção apresenta os resultados obtidos na simulação do voo realizada pelo agente *plane* no simulador X-Plane. O agente pode controlar adequadamente a aeronave na maioria das simulações. Em algumas simulações a aeronave não seguiu um percurso retilínio na decolagem e em outras simulações a aeronave inclinava-se horizontalmente e não retornava a posição de estabilidade. Tais comportamentos se deram devido a simplicidade do raciocínio do agente e a alta realidade do simulador, que provê condições climáticas aleatórias que podem interferir no controle de voo. Contudo, uma codificação de planos e o acesso a outras informações do simulador como crenças do agente permitem um melhor controle da aeronave. A figura 7 exibe o agente *Plane* controlando o simulador de voo.

Foi fixado um aeroporto para que o agente pudesse realizar a decolagem, visto que era necessário determinar um ponto limite para que a aeronave não saísse da pista. Dessa forma, também é necessário a criação de planos para que o agente possa decolar de qualquer aeroporto sem uma prévia configuração.

A integração do Jason com o simulador X-Plane ocorreu conforme o esperado, mantendo uma comunicação através de pacotes UDP sem interferências e satisfatória para o raciocínio do agente. Sobre a performance e o controle do simulador, o ciclo de racioncínio do agente é mais rápido do que o tempo de chegada das informações vindo do simulador, facilitando o processamento das informações pelo agente. Contudo, o agente contém planos simples, dessa forma, testes com planos mais complexos devem ser realizados. As ações realizadas pelo agente são executadas com diferenças mínimas entre a deliberação do agente e a efetiva execução da ação no simulador, não comprometendo a

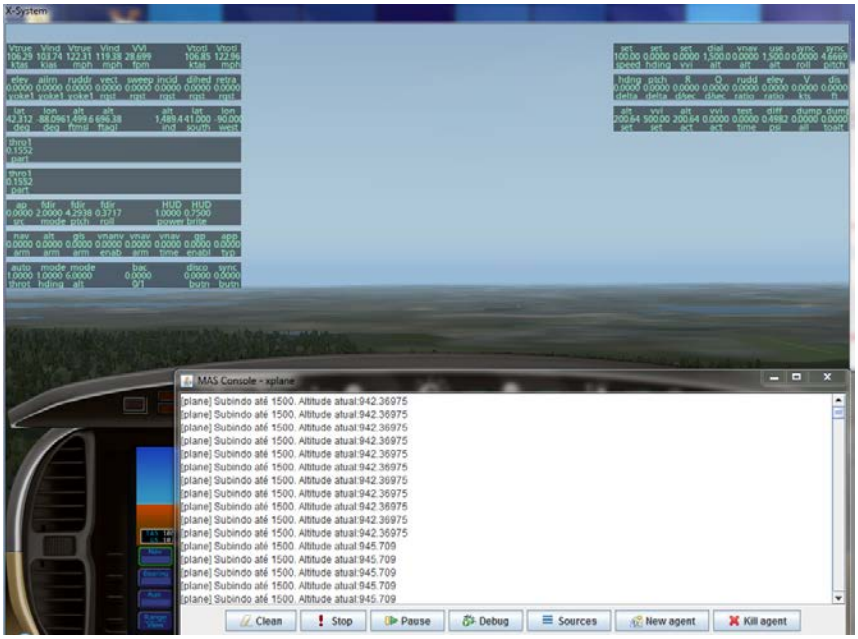

Figure 7. O agente controlando o simulador de voo.

ação do agente. Contudo, nenhuma simulação que exigisse um tempo de resposta quase que instântanea foi realizado.

4. Trabalhos Relacionados

Esta seção apresenta brevemente alguns trabalhos relacionados que utilizam linguagens de programação a agentes em simulações como [Wallis et al. 2002], que apresenta um conjunto de comportamentos táticos em um ambiente simulado de voo. Contudo o trabalho foi desenvolvido utilizando JACK, que é uma linguagem orientada agentes proprietária e reativa. Em [Huff et al. 2003] foi desenvolvido um ambiente de simulação para diferentes abordagens de voo usando o Java em conjunto com outras arquiteturas de softwares. Ambos os trabalhos não utilizam um simulador de voo profissional, sendo desenvolvidas rotinas próprias de simulação de voos, o que impossibilita uma análise para testes em voos em protótipos e ambientes reais.

Este trabalho apresenta um agente inteligente usando o Jason, que é uma linguagem orientada a agentes gratuíta e cognitiva que utiliza a aquitetura BDI. A arquitetura BDI permite que o agente seja programado baseados em crenças, desejos e intenções com a utilização de planos e ações aproximando a programação de ações cognitivas humanas. O trabalho integra o ambiente simulado do agente em Jason a um simulador de voo profissional utilizado para treinamentos que contabiliza hora de voos a pilotos. A utilização do simulador de voo garante que as ações realizadas pelo agente são seguras em um ambiente real de execução.

5. Conclusão

Este artigo apresentou uma integração entre um simulador de alta realidade chamado X-Plane e *framework* orientado a agentes Jason através de uma adaptação de uma biblioteca para comunicação através de pacotes UDP. Foi apresentado também um agente inteligente

que é capaz de realizar uma decolagem usando suas percepções de seu posicionamento global atual e, após atingir determinada altitude, estabilizar o voo e ativar o piloto automático da aeronave.

A integração permitiu que o controle da aeronave através do envio e recebimento dos comandos e informações através de pactoes UDP. O pacote em Java de comunicação com o simulador foi integrado ao ambiente do agente, que por vez passa as informações ao agente inteligente codificado em Jason. As simulações em condições climáticas normais e levando em consideração apenas as informações passadas ao agente foi satisfatória.

Como trabalhos futuros, é necessário ampliar os planos do agente para que ele possa ter um maior controle da aeronave em diversas situações. Também é possivel integrar outras aeronaves para atuarem em rede no mesmo simulador, permitindo a realização de missões em conjunto ou realizar voos em formação, integrando os agentes a alguma plataforma organizacional como o Moise+.

References

Bordini, R. H., Hubner, J. F., and Wooldridge, W. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*. Jonh Wiley and Sons, London.

Bratman, M. (1987). *Intentions, Plans, and Practical Reason*. Harvard University Press.

Cantoni, L. (2010). Avaliação do uso da linguagem pddl no planejamento de missões para robôs aéreos. Dissertação, Universidade Federal de Minas Gerais, Belo Horizonte, Brasil.

Huff, N., Kamel, A., and Nygard, K. (2003). An agent based framework for modeling uavs. In *Computer Applications in Industry and Engineering*, page 139–144. Springer-Verlag.

Wallis, P., Ronnquist, R., and Lucas, A. (2002). The automated wingman: Using jack intelligent agents for unmanned autonomous vehicles. In *Aerospace Conference*, page 2615–2622. IEEE.

Winikoff, M. (2005). Jack intelligent agents: An industrial strength platform. In Bordini, R., Dastani, M., Dix, J., Fallah Seghrouchni, A., and Weiss, G., editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer US.

Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. John Wiley & Sons.