Plataforma e ambiente de simulação para testes de algoritmos de guiamento de VANTs

Vitor C. F. Gomes[1], Felipe L. L. Medeiros[1], Márcia R. C. de Aquino[1], Diego Geraldo[1], Luiz H. M. Dias[2], Marcos E. L. Honorato[2]

1. Divisão de Geointeligência (EGI), Instituto de Estudos Avançados (IEAv), São José dos Campos – SP - Brasil
2. Universidade Federal de São Paulo (UNIFESP), São José dos Campos – SP - Brasil

***Resumo* — Este trabalho apresenta a estruturação de uma plataforma e a montagem de um ambiente de simulação para a realização de testes de algoritmos que fazem o guiamento de VANTs. Os dois sistemas são estruturados a partir de hardware e software livres e permitem que aplicações de guiamento de VANTs atuem no controle do veículo através do envio de comandos de guinada, arfagem, rolamento e aceleração ao Piloto Automático (PA). Para a comunicação entre PA e as aplicações de guiamento, foi desenvolvida uma biblioteca de comunicação, que utiliza o protocolo aberto MAVLink. Essa biblioteca permite que aplicações embarcadas no VANT possam comandar o piloto automático e monitorar os parâmetros de vôo de maneira facilitada. Um teste com uma aplicação para pouso autônomo através da detecção de heliponto em imagens foi conduzido no ambiente simulado e um teste preliminar foi realizado utilizando a plataforma estruturada.**

***Palavras-Chave* — Integração de Sistemas Embarcados, Ardupilot, MAVLink.**

I. INTRODUÇÃO

Veículos Aéreos Não Tripulados (VANTs) são aeronaves projetadas para operar sem piloto a bordo. Esses veículos têm se popularizado nos últimos anos devido à vasta gama de aplicações em atividades militares e civis e por possuírem características que se destacam em relação às aeronaves tripuladas. Entre elas podemos citar: a realização de tarefas que colocariam em risco a tripulação e o custo de operação reduzido na realização de determinadas tarefas. VANTs com capacidade de decolagem e pouso na vertical (VTOL, do inglês *Vertical Takeoff and Landing*) podem, ainda, ser utilizados em locais de difícil acesso e no emprego de busca e salvamento, devido às suas características de manobrabilidade.

Atualmente, o principal foco de pesquisa com VANTs é o aumento da autonomia desses veículos, reduzindo a dependência de operadores remotos. Tradicionalmente, a autonomia dessas aeronaves está associada à utilização de meios computacionais embarcados para a realização de tarefas como: a decolagem, a navegação e o pouso autônomos; a detecção automática de obstáculos para a correção da navegação; o monitoramento e o rastreamento de alvos; entre outras.

V. Gomes, vitor@ieav.cta.br, Tel +55-12-39475313, F. Medeiros, felipe@ieav.cta.br, Tel +55-12-39475311; M. Aquino, marcia@ieav.cta.br, D. Geraldo, diegogeraldo@ieav.cta.br, Tel. +55-12-39475307, Fax +55-12-3944-1177.

VANTs modernos projetados para realizar tais tarefas são sistemas distribuídos complexos, compostos de hardware e softwares heterogêneos. Estes sistemas incluem microcontroladores, processadores de propósito geral, atuadores, um conjunto de sensores, algoritmos para a análise dos dados de sensores e controle dos atuadores e programas para o planejamento e monitoramento de missões. A integração de múltiplos componentes, assegurando um bom funcionamento é um desafio na construção de sistemas de VANTs com capacidades autônomas [1]. Além disso, a capacidade de testar, através de simulação e em aplicações reais, algoritmos que fazem o guiamento de VANTs é essencial para a validação de novas técnicas [2].

De maneira geral, um VANT é composto por [3]: uma estrutura (*frame*); um conjunto de motores e hélices; um PA com sensores inerciais, magnetômetro e barômetro; um receptor de GPS; um Rádio-Controle (RC) com receptor; módulos transceptores para o monitoramento e o controle via Estação de Controle de Solo (ECS); um conjunto de controladores eletrônicos de velocidade (ESC, do inglês *Electronic Speed Control*) para os motores; e bateria.

A Fig. 1 ilustra de maneira esquemática a estrutura típica dos componentes que realizam o controle de um sistema VANT. À esquerda, está representada uma ECS, unidade responsável por comandar e monitorar as ações do VANT. Essa estação comunica-se de duas formas distintas com o PA do VANT. A primeira é através de comandos enviados via RC, onde um piloto controla diretamente os movimentos da aeronave. A segunda forma é através de uma aplicação do tipo Sistema de Controle de Solo (SCS) executada em um computador que, através de transceptores sem fio, envia e recebe mensagens do PA da aeronave. Essas mensagens podem requisitar parâmetros de vôo do VANT, como velocidades, altitude e carga da bateria, ou enviar ações a serem executadas pelo VANT, como a alteração de modo de vôo ou pontos de controle (*waypoints*) a serem visitados.

Na Fig. 1, estão ilustrados o PA e o módulo receptor de RC, embarcados no veículo. As setas indicam o sentido do fluxo das informações. Para destacar os componentes de controle do VANT, neste diagrama, foram omitidos os demais componentes que também são embarcados no VANT, citados anteriormente.

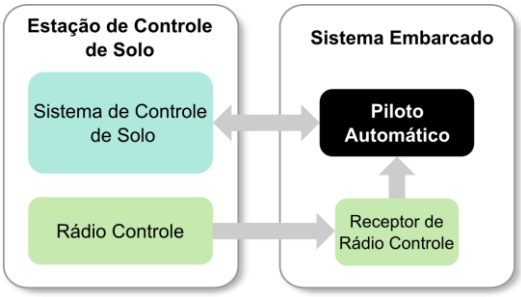

Fig. 1. Diagrama típico de componentes da Estação de Controle de Solo integrada com o Piloto Automático de um VANT.

Considerando a estrutura típica de controle de um VANT apresentada, as opções para a execução de algoritmos de guiamento automático da aeronave seriam a ECS ou o PA. O uso da ECS para esse fim é restrito pela distância máxima de comunicação entre os recursos, o que reduziria os cenários de emprego do VANT. O uso do PA como plataforma para a execução dos algoritmos é limitado por, principalmente, dois motivos. O primeiro é quanto aos recursos computacionais disponíveis. Frequentemente os algoritmos internos para o controle do VANT esgotam o uso de processamento e memória. O segundo motivo é quanto à restrição de conexão de sensores externos (câmeras, radares e etc), a serem utilizados pelos algoritmos de guiamento. Sob esse cenário, a inclusão de um módulo de computação extra embarcado no VANT é essencial.

Neste sentido, este trabalho tem como objetivo a estruturação de uma plataforma para a execução embarcada de algoritmos de guiamento de VANTs. Além disso, deseja-se que esses algoritmos possam ser testados em ambiente simulado e que a transição entre o Ambiente Simulado (AS) e a Plataforma de Operação (PO) exija o mínimo de reprogramação e reconfiguração.

O restante desse trabalho está estruturado da seguinte forma: a seção II apresenta os materiais e métodos empregados, onde são indicados os sistemas e os componentes utilizados neste trabalho. Na seqüência, são apresentados detalhes da biblioteca de comunicação desenvolvida e a seção IV descreve o ambiente simulado estruturado. Na seção V, são apresentados dois experimentos realizados e, por fim, são feitas as conclusões na seção VI.

II. Materiais e Métodos

A solução adotada para a integração de um novo módulo de computação aos componentes embarcados no VANT é apresentada na Fig. 2. Do ponto de vista do fluxo de dados, o Módulo de Computação (MC), onde os algoritmos de guiamentos de VANT serão executados, foi posicionado entre o SCS e o PA. O objetivo é permitir que as aplicações possam receber e enviar comandos de forma transparente ao PA, ou seja, como se as mensagens viessem do próprio SCS.

Na estrutura apresentada, um aplicativo Gerenciador de Comunicação (GC) é responsável por coordenar o fluxo de mensagens do SCS e das aplicações de guiamento executadas no MC. Para facilitar a integração das aplicações com o GC, uma Biblioteca de Comunicação foi projetada para fornecer os recursos básicos para o envio e recebimento de mensagens do PA.

A comunicação via RC não foi alterada, permitindo que um piloto remoto possa tomar o controle da aeronave caso seja necessário. Além disso, a abordagem utilizada na solução apresentada permite que o VANT continue funcional mesmo sem uma aplicação de guiamento, com a SCS recebendo e enviando comandos ao VANT.

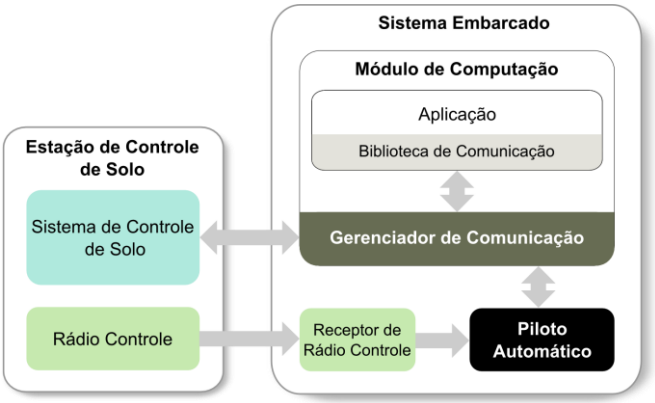

Fig. 2. Diagrama de componentes da Estação de Controle de Solo integrada com o Piloto Automático do VANT via Gerenciador de Comunicação.

A partir da estratégia adotada, buscou-se identificar dispositivos e aplicações que atendessem o cenário planejado. Nesta etapa, foi dada prioridade à escolha de hardware e software livres, de fácil acesso e que houvesse ampla documentação. O objetivo foi utilizar, ao máximo, aplicações já disponíveis, como forma de reduzir o esforço de desenvolvimento e manutenção futura.

O ponto central da estruturação da plataforma é a escolha do PA. Para essa função foi escolhido o Ardupilot Mega (APM) [4]. Essa plataforma, amplamente utilizada em trabalhos de pesquisa [2, 3], tem hardware e códigos-fonte abertos e integra sensores inerciais, magnetômetro e barômetro em sua arquitetura. Além disso, permite a utilização de receptor GPS e sensor ultrassom para o posicionamento do veículo. O Ardupilot é uma plataforma completa, que tem capacidade de auto-estabilização, navegação por *waypoints* e comunicação com SCS através de transceptores sem fio. Ele é projetado para ser uma plataforma flexível, sendo capaz, através da alteração de *firmware*, de controlar VANTs de asa fixa, de asa rotativa ou veículos terrestres e aquáticos.

O APM utiliza o protocolo de comunicação MAVLink (*Micro Air Vehicle Communication Protocol*) [5] para as suas comunicações externas e nas comunicações com os recursos internos. É através desse protocolo que o APM se comunica com o SCS.

O MAVLink é um protocolo aberto, que permite a personalização de mensagens para aplicações específicas e que possui 8 bytes de sobrecarga de controle (*overhead*). Seu uso vem sendo adotado em diversos pilotos automáticos e sistemas de controle de solo, podendo ser considerado um padrão para plataformas abertas para pequenos VANTs. Este fato é importante, pois indica a possibilidade de substituição

do PA ou do SCS por outros sistemas que também utilizem esse protocolo de comunicação.

Para o papel de GC na solução adotada neste trabalho, é utilizado o MAVProxy [6]. Esse sistema foi desenvolvido para ser um SCS leve e completo. Ele possui interface via linha de comando, a qual permite o envio de comandos e a leitura de parâmetros de vôo do VANT. Suas funcionalidades podem ser estendidas através de *plugins* escritos em linguagem de programação Python.

Apesar de seus recursos como SCS, a adoção do MAVProxy foi motivada por outra característica. Esse software é capaz de receber, via comunicação serial ou de rede, mensagens MAVLink de diferentes fontes e encaminha-las para o PA. Além disso, quando ele recebe uma mensagem do PA, ele a reencaminha para todos os recursos que estão conectados a ele, funcionando como um multi/demultiplexador de mensagens. Outros fatores que favorecem sua escolha é a capacidade de ser executado em segundo plano, consumir poucos recursos de processamento e memória e ser multiplataforma [6].

Para permitir a comunicação de aplicações de guiamento de VANT com o PA, através do GC, foi desenvolvida uma biblioteca. As funcionalidades e os detalhes de desenvolvimento dessa biblioteca são descritos na próxima seção.

III. Biblioteca de Comunicação: MAVCom

Os desenvolvedores do protocolo MAVLink disponibilizam o código-fonte de duas bibliotecas para a utilização desse protocolo, uma em linguagem C e outra em Python [5]. Essas bibliotecas fornecem meios para a construção mensagens de tipos definidos no protocolo ou a criação de novos tipos, para casos particulares. Para o envio de mensagens, as funções dessas bibliotecas permitem a conversão de uma estrutura de dados, com os parâmetros a serem enviados, em uma seqüência binária reconhecida pelo protocolo e pronta para ser transmitida por um canal de comunicação. Para o recebimento, são fornecidas funções que convertem uma seqüência binária em uma estrutura de dados tratável pelas aplicações.

O estabelecimento e manutenção de conexão, a certificação que um comando foi recebido pelo PA ou o envio de um comando complexo, que exige uma seqüência definida de comandos simples, não são tratados pelas bibliotecas distribuídas pelos desenvolvedores do MAVLink. Para essas situações, necessárias às aplicações que fazem o guiamento do VANT, foi desenvolvida uma biblioteca de comunicação chamada MAVCom. Essa biblioteca foi escrita em linguagem C e funciona como uma camada de alto nível sobre a biblioteca em C do MAVLink.

Para a comunicação entre a MAVCom e o MAVProxy, foi escolhido o uso de rede através do protocolo UDP. A escolha é motivada pela simplicidade de implementação e a baixa sobrecarga de comunicação, frente ao TCP, por exemplo. A comunicação foi desenvolvida utilizando *sockets*, havendo versão para execução em sistema operacional Windows e Linux.

Uma das funcionalidades implementadas na MAVCom é a manutenção da comunicação entre a aplicação e o APM. O protocolo MAVLink possui uma mensagem do tipo HEARTBEAT que deve ser enviada ao PA a cada período predefinido (para o APM a freqüência é de 1Hz). Essa mensagem indica que a comunicação continua ativa. No caso do APM não receber mensagens HEARTBEAT por um período, ele poderá entrar em modo de segurança (*Fail-safe*), pousando ou retornando ao ponto de decolagem. Mensagens do tipo HEARTBEAT também são enviadas pelo PA aos demais sistemas para informar seu estado ativo.

Além da mensagem HEARTBEAT, o APM também envia mensagens com os principais parâmetros de vôo do VANT. Essas mensagens, que também são tratadas pela MAVCom, fornecem, por exemplo, posição GPS, atitude do VANT, altura relativa e absoluta, velocidades e carga da bateria. A freqüência com que essas mensagens são enviadas pelo PA é configurável através de uma mensagem do tipo REQUEST_DATA_STREAM.

Para permitir que o envio de mensagens HEARTBEAT e o recebimento de mensagens seja feito de forma isolada do restante da aplicação, a MAVCom cria uma *thread* para cada uma dessas funcionalidades.

A *thread* de envio de mensagens HEARTBEAT é a mais simples. Através de um laço de repetição é feito o envio da mensagem e a execução da *thread* é interrompida por um tempo predefinido através da função *sleep*.

A *thread* responsável pelo recebimento de mensagens, aguarda a chegada de novas mensagens através de uma chamada blocante de recebimento via *socket*. Dessa forma, usa-se processamento somente quando uma mensagem é recebida, evitando o uso contínuo do processador. A cada nova mensagem recebida, essa *thread*, realiza seu desempacotamento, através da biblioteca do MAVLink, e armazena os parâmetros do VANT em uma estrutura de dados própria. Essa estrutura de dados é passada para a aplicação através de uma chamada de *callback*.

Para o envio de comandos para o APM, a MAVCom fornece as seguintes funções:

- *mavcom_arm*: realiza o procedimento para a habilitação do VANT para início do vôo;
- *mavcom_takeoff*: realiza a decolagem do VANT na vertical. A função é retornada quando a altura passada como parâmetro é atingida ou em caso de falha;
- *mavcom_send_data_stream*: permite alterar a freqüência de recebimento de determinado tipo de mensagem com parâmetros do VANT;
- *mavcom_send_rc_channel_override*: envia um sinal de RC ao APM, como aceleração, guinada, rolagem, arfagem e etc;
- *mavcom_reset_rc_channel_override*: retorna todos os sinais de RC para os valores padrão;
- *mavcom_send_mode*: altera o modo de vôo, retornando com sucesso quando o APM confirmar a alteração ou com um erro em caso de falha;

Além dessas, a MAVCom fornece funções que facilitam o acesso aos dados recebidos e armazenados pela biblioteca.

Essas funções permitem que a aplicação possa acessar parâmetros de vôo do VANT a qualquer instante, mesmo fora da chamada de *callback*. Essas funções permitem, por exemplo, o acesso à atitude, à posição GPS e à carga da bateria. O horário de recebimento está associado a cada informação, para que a aplicação possa avaliar a validade do dado armazenado.

Para o envio de mensagens não tratadas pela MAVCom, como mensagens personalizadas, é fornecida a função *mavcom_send_mavlink_msg*. Essa função utiliza os mesmos recursos de rede gerenciados pela MAVCom, facilitando o envio de mensagens MAVLink pela aplicação.

Para a utilização da MAVCom, é necessária a invocação de uma função de inicialização da biblioteca. Nessa chamada, *mavcom_initialize*, devem ser informados os parâmetros: IP do GC, porta para envio dos dados, porta para recebimento dos dados, freqüência de envio de HEARTBEAT e ponteiro da função de *callback*. Ao encerrar o uso da biblioteca, a aplicação deve invocar a função *mavcom_finalize*, responsável por desalocar os recursos utilizados e finalizar as *threads* em execução.

IV. Ambiente Simulado

Para permitir a realização de testes iniciais de algoritmos de guiamento de VANT, um AS foi montado. Esse ambiente utiliza como referência os componentes apresentados na Fig. 2, alterando o APM por um simulador. A Fig. 3 ilustra este novo cenário. A estrutura apresentada permite que as aplicações utilizem a mesma interface utilizada na plataforma de testes, visando pouca ou nenhuma alteração de código entre a simulação e a utilização da plataforma.

Para simular o PA, foi utilizado um simulador do tipo *Software in the Loop (*SITL*)*, que executa o código fonte do próprio APM. Neste simulador, disponibilizado pelos desenvolvedores do APM, os dados dos sensores da aeronave são produzidos por um simulador de vôo [7]. O controle do APM via RC não é utilizado no AS. Neste ambiente, o controle do VANT pode ser feito através do SCS, do terminal do MAVProxy ou de uma aplicação que utiliza a MAVCom.

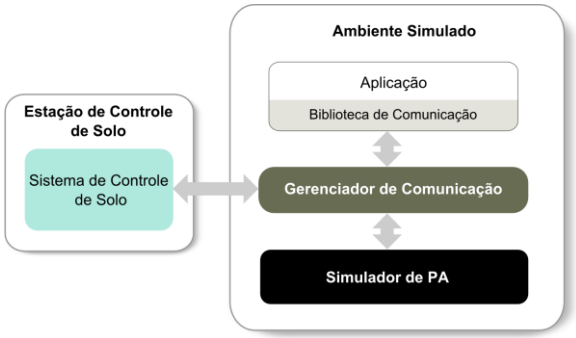

Fig. 3. Diagrama de componentes de controle de um sistema de VANT integrados com um simulador de PA.

Para facilitar a replicação do AS, foi criada uma máquina virtual com todos os componentes necessários. Foi escolhido como Sistema Operacional o Ubuntu 14.04. O simulador foi instalado conforme as orientações fornecidas pelos desenvolvedores [5]. Na instalação do simulador, o MAVProxy é instalado automaticamente para ser utilizado como SCS, mas, neste caso, ele é utilizado como GC. Um projeto básico (*template)* com a biblioteca MAVCom e aplicações de exemplo também são disponibilizados nesse ambiente.

Para ser utilizado junto ao AS, foi escolhido como SCS o software *Mission Planner* [8]. Esse sistema pode ser conectado ao APM através de conexão serial ou via rede. Para o ambiente montado, é utilizada a rede para conectar o *Mission Planner* ao MAVProxy. Dessa forma, o SCS não precisa, necessariamente, ser executado no mesmo ambiente do simulador.

V. Experimentos

Como forma de avaliar a Plataforma de Operação e o Ambiente Simulado foram conduzidos dois testes. Para o AS, uma aplicação de pouso autônomo de VANT VTOL, a partir da detecção de heliponto em imagens [10], foi integrada com a MAVCom. Neste teste, foi avaliada, visualmente, a resposta do VANT simulado em relação aos comandos enviados. Como no AS não é possível obter a imagem sob o VANT, foi utilizado um vídeo gravado previamente em um voo de teste. A partir da detecção do heliponto do vídeo, a aplicação calcula os comandos a serem enviados ao VANT para posicioná-lo sobre a área de pouso. O envio foi feito via MAVCom e a trajetória produzida pelo VANT foi observada no SCS. Uma aplicação de monitoramento (*logger)* foi utilizada para registrar os parâmetros de vôo do VANT durante a simulação. Ambas as aplicações, de pouso autônomo e *logger*, foram executadas concorrentemente no AS. A primeira fez recebimento e envio de mensagens ao APM, enquanto que a segunda somente recebeu os parâmetros do VANT.

O segundo teste foi feito com a PO. Como módulo de computação, foi utilizado o minicomputador Raspberry PI (RPI), com o sistema operacional Raspbian [9]. Neste sistema, foi instalado o MAVProxy e o *template* com a biblioteca MAVCom. Para monitoramento via SCS, foram utilizados dois módulos transceptores de 430MHz. Um dos módulos foi conectado ao RPI via porta GPIO e o outro instalado em um computador, onde foi executado o software *Mission Planner*. O APM foi conectado ao RPI através da porta USB. O diagrama da Fig. 4 ilustra os componentes utilizados neste teste e suas conexões. Esses componentes foram instalados em um VANT quadcóptero para a realização do teste.

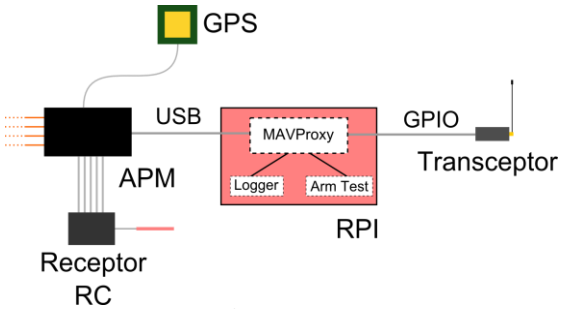

Fig. 4. Componentes da plataforma para testes de algoritmos de guiamento de VANTs.

A aplicação de *logger* utilizada no teste do AS também foi utilizada no teste da PO. Outra aplicação, *arm test*, foi desenvolvida para avaliar o envio de comandos ao VANT. Essa aplicação habilita o APM (*Arm*) e envia comandos de arfagem, guinada e rolagem para o VANT. Por segurança, os testes desse segundo experimento foram conduzidos com o VANT sem hélices, em laboratório.

Para a execução do MAVProxy no RPI, foi utilizado o comando apresentado na Fig. 5. Na primeira linha, é possível verificar a indicação da localização do APM (/dev/ttyACM0) através do parâmetro *master*. As demais linhas indicam os canais onde haverá conexões com o MAVProxy. Na linha 2, é indicado o caminho da GPIO, onde está conectado o transceptor, e a taxa de transmissão (*baud rate*). Nas linhas 3 e 4, são disponibilizadas conexões de rede locais, através do IP 127.0.0.1, para as portas 14550 e 14551. Essas últimas são utilizadas pelas aplicações em execução no RPI.

```
1  mavproxy.py --master=/dev/ttyACM0 \
2  --out /dev/ttyAMA0,57600 \
3  --out 127.0.0.1:14550 \
4  --out 127.0.0.1:14551
```

Fig. 5. Execução de MAVProxy e parâmetros de configuração.

VI. Resultados e Discussão

Uma imagem do experimento realizado no AS pode ser vista na Fig. 6. Nela é possível ver na parte superior, à esquerda, o vídeo utilizado como entrada para o algoritmo e, na parte inferior, a estrutura do heliponto detectado pela aplicação de pouso autônomo. Na direita, é possível visualizar, na parte superior, a posição do VANT no simulador. Na parte inferior direita, é visualizado um console do simulador, que apresenta os principais parâmetros de vôo do VANT simulado. O *logger* não é ilustrado nessa figura, pois foi executado em segundo plano.

Fig. 6. Execução da aplicação de detecção de heliponto no ambiente simulado.

A possibilidade de visualizar o comportamento do VANT em relação à posição do heliponto no vídeo foi importante para a realização de ajustes na aplicação, a qual utiliza um controlador proporcional para o cálculo das intensidades dos comandos enviados ao VANT. Além disso, com esse teste, verificou-se a facilidade de integrar a MAVCom com uma aplicação. Além das funções de inicialização e finalização, a aplicação fez uso das funções de envio de comandos de RC (*mavcom_send_rc_channel_override*), para o guiamento do VANT, e de alteração de modo de navegação (*mavcom_send_mode)*, para colocar o VANT em estado de pouso. Com o teste realizado, o funcionamento das *threads* para a manutenção da comunicação e para o recebimento de parâmetros do vôo foi confirmado.

A estrutura montada para o segundo experimento é apresentada na Fig. 7. Nessa figura, é possível identificar os componentes e suas conexões. O APM (1) está conectado ao Raspberry PI (2) através de cabo USB (3). O transceptor (4) é conectado ao RPI através da porta GPIO (5). O receptor de GPS (6) e o receptor de Rádio Controle (7) foram ligados ao APM utilizando os conectores específicos do PA. Os demais cabos que aparecem na imagem são utilizados para a conexão de ESCs e a alimentação com a bateria.

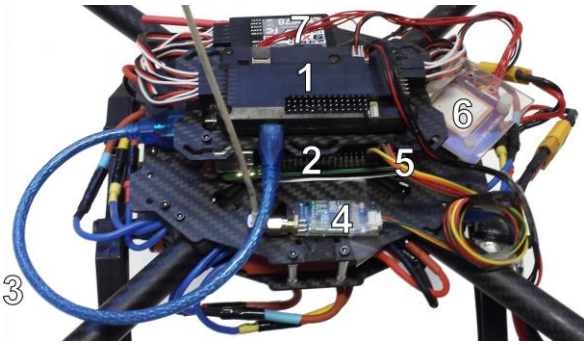

Fig. 7. Plataforma para execução de testes de algoritmos de guiamento de VANTs

Os testes executados com a plataforma montada permitiram verificar que o APM reconheceu os comandos enviados pela aplicação através da MAVCom. Além disso, a conexão entre o APM com o SCS, através do transceptor ligado ao RPI, funcionou corretamente. A aplicação em execução no RPI conseguiu registrar os parâmetros coletados do APM, como posição GPS, atitude e carga da bateria, indicando sucesso nos testes realizados.

VI. Conclusões

Este trabalho apresentou a estruturação de uma plataforma para a realização de testes de guiamento de VANT e a montagem de um ambiente simulado. Dois experimentos foram realizados para avaliar o comportamento dos sistemas. Os resultados indicam o correto funcionamento de ambos para os testes realizados.

A biblioteca desenvolvida, MAVCom, permitiu uma fácil integração entre as aplicações e os ambientes, tanto a plataforma operacional como o ambiente simulado. Além disso, nenhuma alteração em código-fonte foi necessária para a migração entre um ambiente e outro.

O retorno visual da posição do VANT simulado em resposta aos comandos enviados ao APM é bastante útil no desenvolvimento das aplicações para guiamento desses veículos. A arquitetura estruturada para o Ambiente Simulado permite, inclusive, que a aplicação seja executada em outra máquina, por conta da utilização de comunicação via rede. Esta característica facilita o desenvolvimento das aplicações em um ambiente que o desenvolvedor esteja mais familiarizado.

As próximas etapas deste trabalho são: a) a inclusão de novas funcionalidades à MAVCom, como a manipulação de pontos de controle (*waypoints*) e a criação de um modo com limiares de segurança (*fail-safe*), para restringir o envio à APM de certos valores de parâmetros, que poderiam colocar o VANT em situações de risco; b) realizar testes com a plataforma com uma aplicação de guiamento de VANT em campo; e c) desenvolver uma biblioteca para fornecer imagens da região sobrevoada pelo VANT, para ser utilizada no Ambiente Simulado.

REFERÊNCIAS

[1] A. El-Sayed, M. ElHelw, “Distributed component-based framework for unmanned”. Proceeding of the IEEE International Conference on Information and Automation. China, 2012, p. 45-50.

[2] M. Coombes, O. McAree, W. Chen, P. Render, “Development of an autopilot system for rapid prototyping of high level control algorithms”. UKACC International Cofnerence on Control. UK. 2012.

[3] Guowei Cai, Ben M. Chen, Tong Heng Lee, “Unmanned rotorcraft systems”. Advances in Industrial Control. 2011.

[4] 3DROBOTICS, “APM autopilot suite”. Disponível em: <http://ardupilot.com>. Acesso em: 14 de nov. 2014.

[5] L. Meier, ”Micro Air Vehicle Message Marshalling Library”. Disponível em: <https://github.com/mavlink/mavlink>. Acessado em: 2 de abril de 2015.

[6] A. Tridgell, “A UAV ground station software package for MAVLink based systems”. Disponível em: <http://tridge.github.io/MAVProxy> Acessado em: 2 de abril de 2015.

[7] 3DROBOTCS, “Setting up SITL on Linux”. Disponível em: <http://dev.ardupilot.com/wiki/simulation-2/sitl-simulator-software-in-the-loop/setting-up-sitl-on-linux>. Acessado em 2 de abril de 2015.

[8] M. Oborne, “Mission Planner – ground station”. Disponível em: <http://planner.ardupilot.com>. Acessado em 2 de abil 2015.

[9] Raspberry PI Foundation, “Raspberry PI”. Disponível em <http://www.raspberrypi.org>. Acessado em 2 de abril de 2015.

[10] F. Medeiros, V. Gomes, M. Aquino, D. Geraldo, L. Dias, M. Honorato, N. Lacerda, “A computer vision system for guidance of vtol uavs autonomous landing”. 4th Brazilian Conference on Intelligent Systems. 2015.