Skyrats - Grupo de desenvolvimento de drones autônomos da POLI-USP

Subsistema de Software

Firmware e scripts de Lua

Autores
Alessandro Menegon
Bárbara Bueno
Bruno Sarmento

Introdução ao firmware e scripts de missão do Ardupilot em Lua.

2024

Conteúdo

1 Introdução

Todo drone compreende uma estrutura física, componentes eletrônicos e controles inteligentes em seu funcionamento. Sendo ele autônomo ou rádio-controlado, depende da integração de software e hardware para processar informações dos sensores e comandar o próprio voo. Desse modo, para automatizar tarefas com drones inteligentes, é fundamental compreender o funcionamento dos sistemas vitais programados em baixo nível, os componentes eletrônicos envolvidos e como realizar a comunicação para envio de comandos e recebimento de informações de telemetria.

Ademais, será introduzida a linguagem Lua e sua aplicação específica para os sistemas integrados com o Ardupilot. Esta iniciativa é fruto das pesquisas anteriores do subsistema de Software e pretente introduzir uma nova forma de realizar as tarefas de inteligência com maior precisão e rapidez. Assim, trata-se de algo muito recente na comunidade de drones e nunca implementado pela equipe em competições anteriores, necessitando ainda de pesquisas e testes.

Assim, os propósitos desse guia são:

- Introduzir o conceito de Firmware;
- Relacionar o Software com os componentes de Hardware;
- Explorar o funcionamento do piloto automático;
- Introduzir a linguagem Lua;
- Mostrar as aplicações de Lua no Ardupilot.

2 Firmware

2.1 O que é um firmware?

Você sabe qual é a diferença física fundamental entre drones e aviões?

De forma simplificada, o avião utiliza asas fixas como fonte de sustentação e, assim, dispõe de uma estabilidade mecânica nativa, com momentos restauradores em seus eixos de rotação. Isso significa que ele é projetado de forma a se manter em voo estável sem nenhuma atuação de seus sistemas de controle, embora necessite deles para mudar sua velocidade e realizar manobras.

Em contrapartida, drones não são veículos áereos estáveis e, portanto, dependem de sensores, atuadores e controles inteligentes que sejam capazes de mantê-lo em voo estabilizado. É aí que entra o **firmware**: uma arquitetura de **software de baixo nível, rápido e eficiente** que processa os sinais dos sensores e ditam o ritmo dos motores para evitar que o drone tombe ou saia voando para o infinito.

Os firmwares estão presentes não só em drones, mas em qualquer **interface de hardware de dispositivos eletrônicos**, como celulares, televisores, consoles, impressoras e até computadores dos mais variados, na forma de BIOS (você ja deve ter ouvido falar). Assim, são capazes de fornecer um **ambiente operacional** padronizado para o software mais complexo do dispositivo (Windows e Linux, por exemplo) ou, para dispositivos menos complexos, atuar como o sistema operacional completo, executando todas as funções de **controle, monitoramento e manipulação de dados**. [1]

Empresas maiores, como a DJI, utilizam firmwares próprios e fechados, que não podem ser alterados sem permissão, porém existem várias opções abertas mantidas por comunidades de engenheiros, desenvolvedores e hobbystas [2]. Alguns exemplos:

- Ardupilot (Comunidade Open-source)
- PX4 Autopilot (Drone Foundation)
- Betaflight (Open-source FPV)

2.2 A controladora de voo

A controladora de voo (Flight Controller Unit - FCU) é a parte central do drone, que realiza conexão com os módulos de energia, telemetria, GPS, rádio, computador de bordo, entre outros. Essa peça de Hardware pode custar de 400 até dois mil reais e faz o papel de cérebro primitivo do drone, com acelerômetros, giroscópios e outros sensores internos que nutrem os programas de estabilização e controle do firmware.

É importante ressaltar que existem diversas controladoras de voo disponíveis, provenientes de empresas ou comunidades de hardware open-source, com diferentes capacidades de processamento, memória, sensores internos, consumo de energia, protocolos de comunicação, entre outras configurações. Entre elas, separamos algumas famílias de FCU's importantes:

- Pixhawk (Hardware aberto);
- Pixracer (Hardware aberto);
- CubePilot (Hardware aberto);
- Kakute (Hardware fechado);
- Mamba (Hardware fechado).

Em geral, na Skyrats, usamos a Pixhawk para drones autônomos (Figura 1), por ser mais versátil, possuir melhor documentação e maior capacidade de processamento. Entretanto, a Kakute, por exemplo, é utilizada no drone FPV, por ser menor, mais leve e mais barata. [3]

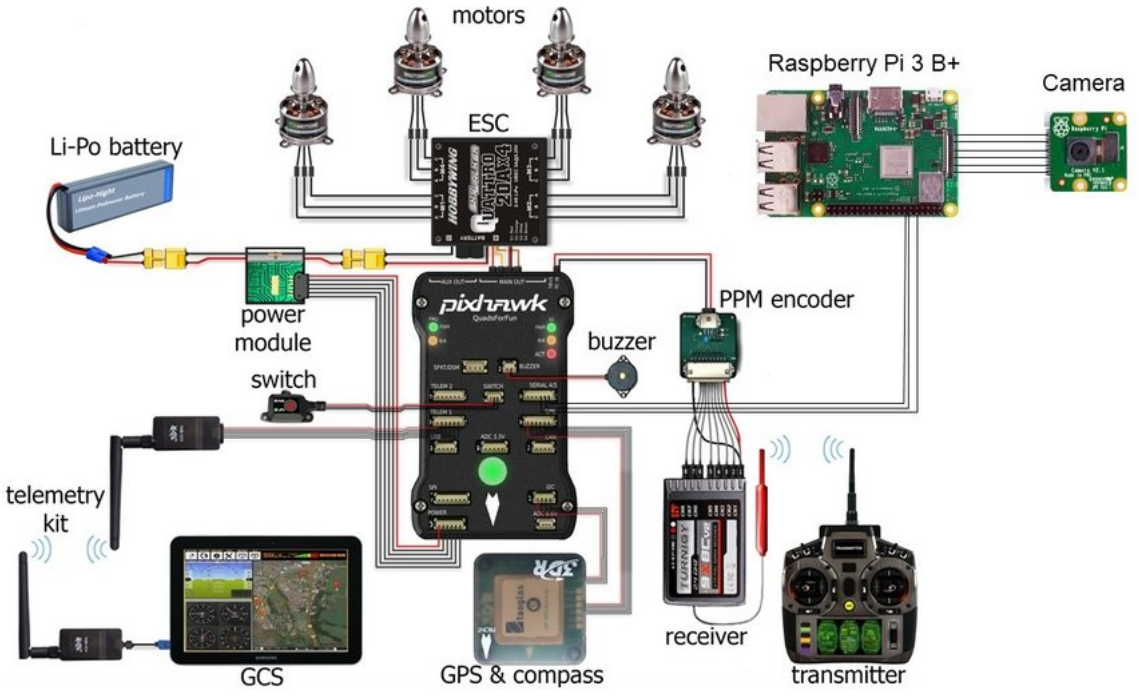


Figura 1: Esquema de montagem dos componentes do drone com a Pixhawk.

2.3 O piloto automático

Uma controladora de voo, necessariamente, contém um firmware capaz de integrar os sensores, acionar os motores e receber inputs do usuário. Entretanto, alguns firmwares vão além e também são capazes de controlar os veículos aéreos de forma completamente autônoma, com modos de missão guiados por programas de computador.

É aí que entra a necessidade de um subsistema de software na equipe!

O piloto automático é a base para que os programadores da equipe possam criar as missões autônomas e pontuar nas competições. Assim, nossa tarefa é criar automações em software para substituir os comandos dados via controle remoto, por meio de computadores.

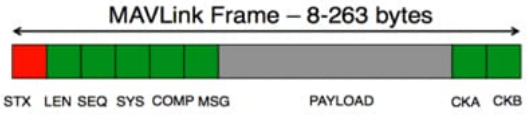
Entretanto, essa tarefa não é simples, e para cumprí-la, precisamos de um piloto automático robusto e confiável, capaz de estimar a própria posição no espaço, compreender as instruções dadas pelos códigos programados e que saiba se comunicar com o usuário. Assim, veremos mais adiante a importância dos protocolos de comunicação e plataformas de controle de solo.

2.3.1 Protocolo MAVLink

Todo protocolo de comunicação estabelece uma série de regras para descrever como transferir dados por uma rede. Essas regras funcionam como um dicionário para definir o que cada sequência de dados digitais significa na comunicação entre dois computadores. Dessa forma, o protocolo MAVLink (Micro Air Vehicle Link) foi estabelecido como principal protocolo de comunicação para enviar comandos e receber informações da controladora de voo. [4]

Ele é extremamente leve e segue uma estrutura baseada em Publishers e Subscribers para envio de dados e retransmissão ponta a ponta das mensagens no caso de subprotocolos de configuração. Ou seja, para garantir que algum parâmetro ou modo de voo tenha sido corretamente alterado, por exemplo, aquele que solicitou a mudança deve aguardar um retorno de confirmação da FCU, enquanto simples dados de telemetria são consumidos e publicados livremente.

A Figura 2 mostra a estrutura básica de uma mensagem de MAVLink. Nela, é possível perceber que toda mensagem apresenta IDs, tanto do sistema, quanto do componente que a está enviando. Isso é essencial para o receptor saber de onde a mensagem está vindo. Além disso, ela também apresenta um ID para cada tipo de mensagem, que indica como os dados devem ser interpretados e também define a estrutura da parcela de dados. [5]



Byte Index	Content	Value	Explanation
0	Packet start sign	v1.0: 0xFE (v0.9: 0x55)	Indicates the start of a new packet.
1	Payload length	0 - 255	Indicates length of the following payload.
2	Packet sequence	0 - 255	Each component counts up his send sequence. Allows to detect packet loss
3	System ID	1 - 255	ID of the SENDING system. Allows to differentiate different MAVs on the same network.
4	Component ID	0 - 255	ID of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
5	Message ID	0 - 255	ID of the message - the id defines what the payload "means" and how it should be correctly decoded.
6 to (n+6)	Data	(0 - 255) bytes	Data of the message, depends on the message id.
(n+7) to (n+8)	Checksum (low byte, high byte)	ITU X.25/SAE AS-4 hash, **excluding packet start sign, so bytes 1..(n+6)** Note: The checksum also includes MAVLINK_CRC_EXTRA (Number computed from message fields. Protects the packet from decoding a different version of the same packet but with different variables).	

Figura 2: Significado de cada parcela da mensagem MAVLink

Alguns exemplos de mensagens do protocolo MAVLink:

- **HEARTBEAT:** É transmitido numa taxa de 1Hz e fornece informações sobre o status do sistema e da aeronave, como modo de voo, status da bateria, tipo de veículo, entre outros. Ele é utilizado para informar a existência de um sistema conectado na rede MAVLink, como um sinal de vida;
- **GPS_RAW_INT:** Transmite informações brutas do GPS, incluindo latitude, longitude, altitude, velocidade, entre outros.;
- **COMMAND_LONG:** Permite enviar comandos para o sistema, como alteração de modos de voo, acionamento de funções específicas, configurações de parâmetros, entre outros;
- **ATTITUDE:** Fornece informações de rotação da aeronave, como ângulos de row, pitch, yaw.

O conceito do protocolo MAVLink é fundamental para compreender como as bibliotecas e pacotes utilizados na criação das missões se comunicam com o drone, como a MAVROS, Dronekit e Ground Control Station, que serão abordados mais para frente. Ademais, saiba que um dia você precisará consultar a documentação do MAVLink para saber a estrutura de uma mensagem ao tentar realizar alguma tarefa de posicionamento ou alterar parâmetros do firmware. Portanto, fique ligado!

2.3.2 Como funciona o ArduPilot?

Como dissemos, o Ardupilot é um piloto automático confiável, versátil e Open Source. Ele serve não só para drones, mas aviões radiocontrolados, helicópteros, multicópteros, barcos, submarinos, carrinhos, e diversos outros veículos de pequeno porte. [6]. Seu surgimento se deu a partir do projeto DIY Drones em 2007 e começou com o desenvolvimento em conjunto de Hardware e Software, com o primeiro repositório aberto publicado em 2009. Hoje, funciona em diversos tipos de Hardware independentes e conta com mais de 700 mil linhas de código!

Sua comunidade é enorme e muito ativa, cheia de pesquisadores, profissionais e entusiastas da área. Inclusive, recomendamos a entrada no canal de discord da comunidade, que está repleta de documentações e projetos desenvolvidos no mundo todo, com membros do time de desenvolvimento sempre disponíveis para tirar dúvidas e discutir ideias. [7]

O ArduPilot possui recursos avançados de estabilização, controle de altitude e orientação muito complexos para explicarmos no momento, mas a Figura 3 mostra um esquema geral de como funciona a arquitetura do sistema. Esse é o primeiro ano da equipe em que foi iniciada uma busca pelo conhecimento mais a fundo do firmware e espera-se que o subsistema de Software continue nesse caminho. [8]

Como uma breve introdução ao ArduCopter, suporte do ArduPilot para drones, saiba que existem diversos modos de voo pré configurados e utilizados pela equipe para cada tipo e etapa das missões. No total, são 25 modos de voo diferentes, dos quais separamos alguns mais importantes:

- **Stabilize:** Modo de voo padrão para pilotagem com ângulos de inclinação do drone estabilizados (conhecido como manual);
- **AltHold:** Igual ao modo anterior, porém, com altitude estabilizada via sensor de distância ou altitude, ou seja, laser, barômetro, ultrassônico, etc;
- **Loiter:** Além de controle de altitude, também utiliza informações de posição (GPS, Lidar, Visão Computacional, etc) para manter a própria posição;

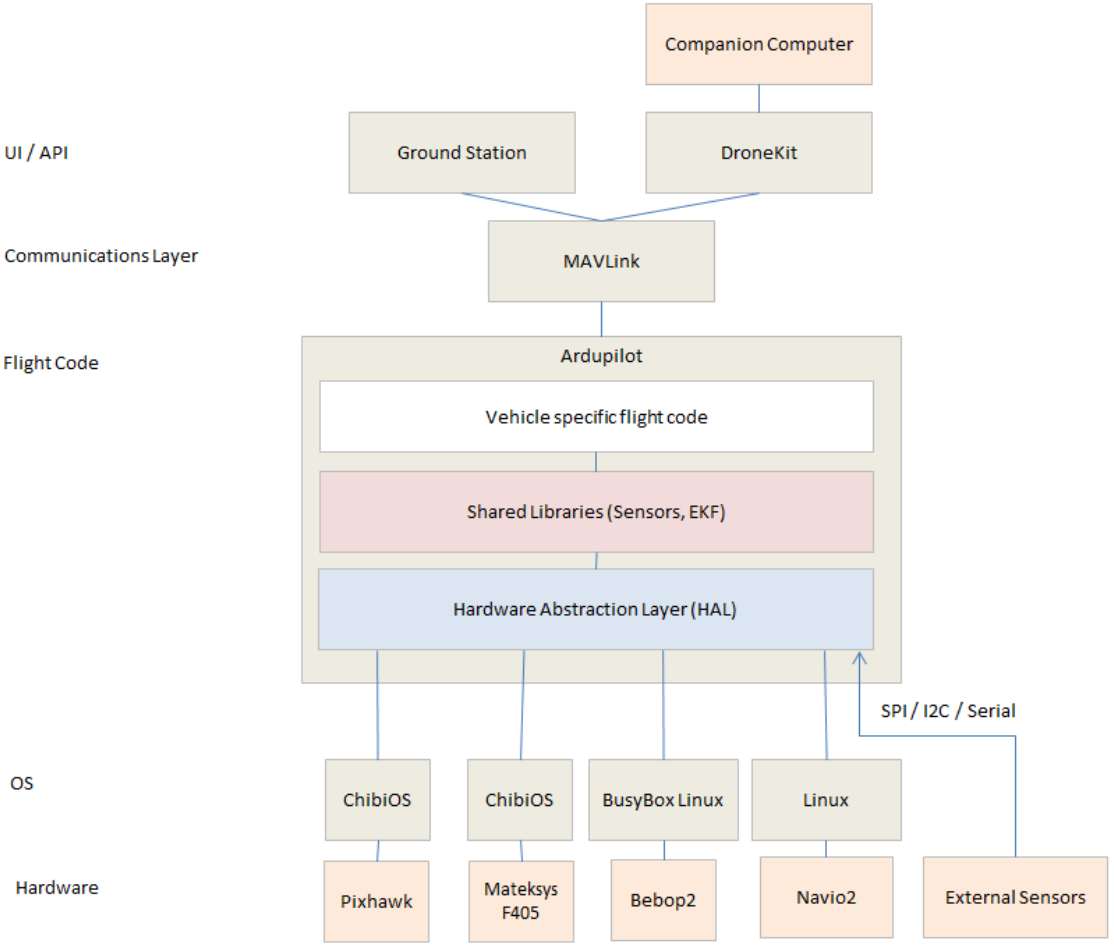


Figura 3: Arquitetura básica do piloto automático.

- **Guided:** Utilizado para navegação em coordenadas determinadas por comandos externos (pela GCS, ou computador de bordo, por exemplo);
- **Auto:** Executa uma missão pré-determinada (modo autônomo);
- **Land:** Reduz a altitude até o ground level em linha reta (ou seja, pouso simples);
- **RTL:** Retorna para a posição de takeoff (Return To Home);

Note o poder do ArduPilot em adequar o comportamento do drone a cada contexto exigido, a fim de aprimorar o seu desempenho em cada tarefa, utilizando sensores, controles e atuadores.

2.3.3 Ardupilot vs PX4

Em 2023, a Skyrats utiliza o Ardupilot como principal firmware, no entanto, anteriormente, era utilizado o PX4-autopilot. Essa mudança ocorreu após a necessidade de uso do ArduPilot para aplicação da câmera Intel Realsense, que culminou em uma pesquisa realizada por membros da equipe, do Software e Hardware, no final de 2022. Assim, tal pesquisa definiu o uso do ArduPilot como novo firmware principal.

A PX4 é outro firmware de piloto automático popular, também de código aberto e com uma comunidade ativa de desenvolvedores. Assim como o Ardupilot, a PX4 fornce recursos avançados de controle de voo, estabilização e navegação.

Algumas das diferenças entre os dois são:

- **Licença de uso:** Para o mercado de drones, a principal diferença entre os dois firmwares é a Licensa de uso dos softwares. O ArduPilot é baseado na GPL (General Purpose License), enquanto a PX4 é baseada na BSD (Berkeley Software Distribution), ou seja, qualquer empresa que modificar o código do ArduPilot é obrigada a divulgar suas alterações, entretanto, modificações da PX4 podem ser protegidas e ocultadas do público. Dessa forma, nossa visão como desenvolvedores Open-source de drones é muito mais compatível com o ArduPilot e podemos nos beneficiar de uma base de projetos muito mais ampla em domínio público.
- **Comunidade:** O ardupilot é um firmware mais antigo e, assim, possui uma comunidade grande e estabelecida de desenvolvedores e usuários que contribuem ativamente com o projeto. A PX4 já é mais recente, no entanto, apresenta uma comunidade expressiva e em constante crescimento.
- **Arquitetura de software:** O ArduPilot adota uma abordagem modular, onde diferentes componentes do firmware são executados como módulos sobre um sistema operacional em tempo real (RTOS). Em contraste, a PX4 segue uma arquitetura mais monolítica, integrando a maioria dos componentes no kernel NuttX. Essas abordagens podem influenciar a flexibilidade e a personalização do firmware.
- **Aplicações e casos de uso:** O ArduPilot tem sido amplamente utilizado em uma variedade de aplicações, desde drones de uso recreativo até drones de pesquisa científica e missões de busca e salvamento. A PX4 também é adequada para várias aplicações, mas é especialmente popular em ambientes industriais e comerciais, onde é usada em drones profissionais.
- **Integração com hardware:** Ambos os firmware têm suporte para uma variedade de plataformas de hardware, no entanto, o ArduPilot possui uma base

de hardware mais ampla e estabelecida. Isso significa que existem mais opções disponíveis para a escolha de hardware compatível com o ArduPilot. Além disso, a câmera Intel Realsense que a Skyrats utiliza para posiconamento indoor se comporta melhor com o firmware ardupilot.

- **Customização e Interação com o firmware:** O ArduPilot oferece diversas ferramentas que permitem a personalização do firmware de forma abrangente, como modificar parâmetros, criar novos modos de voo e desenvolver módulos e plugins. Além disso, também apresenta recursos avançados, como o suporte a scripts de Lua, que será visto mais adiante.

Desse modo, foi decidido que o ArduPilot apresenta vantagens tanto pro Hardware quanto para o Software da equipe. Entretanto, sua adoção ainda exige muita pesquisa para a familiarização da equipe com esse novo firmware. Ressalta-se também que, apesar das vantagens mencionadas, o firmware PX4 ainda tem muita relevância e pode se encaixar melhor em alguns contextos, podendo até ser utilizado novamente pela equipe no futuro, caso necessário.

3 Softwares e ferramentas

3.1 Ground Control Station

As Ground Control Stations (GCS) são aplicativos que rodam em um computador em solo (desktop, tablet, smartphone) e promovem uma interface de comunicação com o drone. Assim, é por meio delas que visualizamos informações em tempo real do estado e da posição do drone (seja ele real ou em simulação). Além disso, elas permitem criar e adicionar missões de voo (que indicam onde o drone deve ir e o que deve fazer) e modificar parâmetros que adequam o veículo a cada tipo de missão.

Na Skyrats, utilizamos o notebook como computador de solo e algumas das GCS que podemos escolher são:

- QGroundControl;
- MAVProxy;
- APM Planner 2;
- Mission planner;
- UgCs.

A escolha é influenciada pelo firmware utilizado, pelo sistema operacional do aparelho (computador) e pela finalidade do drone. Na equipe, optamos pelo uso da QGround na maior parte do tempo, por sua praticidade e acesso a diversas configurações e ferramentas de análises.

Além da QGround, utilizamos também o Mavproxy para aplicações de desenvolvimento, principalmente em simulação. Por fim, destacamos a GCS Mission Planner, que é própria do Ardupilot e contém muitas funcionalidades interessantes, mas que, infelizmente, existe apenas no Windows.

Para saber mais sobre cada uma, consulte a documentação do Ardupilot. [9]

3.1.1 QGroundControl

Todos os aplicativos de GCS possuem características comuns, como visualização do veículo em um mapa, posição em tempo real obtida do GPS a bordo, diveros painéis de equipamentos (altitude, velocidade, altitude, carga de bateria, sensores) e sistema de gravação do link de vídeo, caso exista.

O modelo QGround Control permite o controle e planejamento do voo de qualquer veículo que utilize o protocolo de comunicação MAVLink, que foi discutido anterior-

mente. Além das características gerais mencionadas, ela se destaca por sua facilidade de uso e sua abrangência em diversas plataformas (Windows, Linux, iOS e Android).

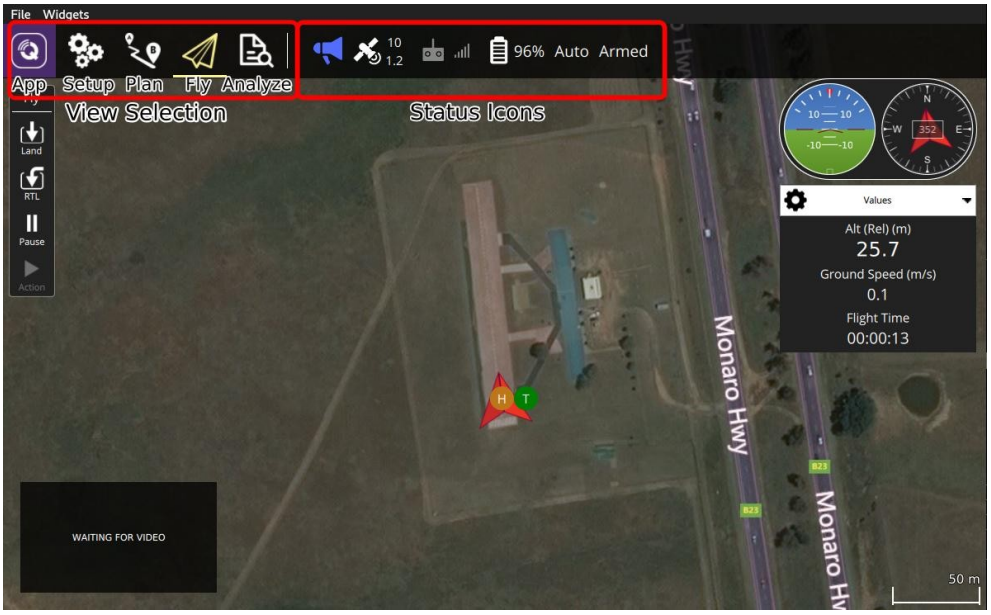


Figura 4: Interface da QGroundControl

3.1.2 MAVProxy

Antes de explicar o que é o MAVProxy, vamos entender primeiro o que é um Proxy. Um Proxy é um intermediário que facilita a comunicação entre dois sistemas ou dispositivos. Assim, ele recebe solicitações de um lado e repassa-as para o outro lado, processando as respostas e retornando-as ao sistema original.

Desse modo, o MAVProxy é uma ferramenta de linha de comando, ou seja, um terminal, que atua como um Proxy para o protocolo MAVLink, agindo como um intermediário entre o drone e a GCS. Ele estabelece uma conexão via MAVLink tanto com a aeronave, quanto com a estação de solo (GCS), recebendo e repassando as mensagens MAVLink entre os dois e sendo capaz de modificar, filtrar ou redirecionar essas mensagens de acordo com as configurações definidas. [10]

Para entender melhor o conceito, podemos fazer uma comparação com uma ligação telefônica. A GCS seria como o telefone principal, que permite que você faça chamadas e receba informações diretamente. O MAVProxy seria como um operador telefônico que gerencia as chamadas, filtra informações, registra chamadas e redireciona chamadas para vários telefones adicionais. Ele adiciona flexibilidade e recursos extras à comunicação.

Mas então, por que dissemos antes que o MAVProxy era uma GCS?

Embora tenha sido criado com o intuito de ser um agente intermediário de uma GCS (ou seja, um middleware), o MAVProxy apresenta funcionalidades básicas de controle em baixo nível e customização que podem ser utilizados como uma GCS primitiva, diretamente pelo terminal do Linux. Isso, aliado a diversas outras aplicações auxiliares (plug-ins), tornou o MAVProxy uma ferramenta muito poderosa para os desenvolvedores da comunidade de drones, principalmente para simulação.

3.2 Pymavlink

O Pymavlink é uma biblioteca de processamento de mensagens MAVLink de baixo nível e de propósito geral, escrita em Python. Ele é usado para implementar o protocolo em vários tipos de sistemas, incluindo GCS (MAVProxy), APIs de desenvolvedor (como o DroneKit) e várias aplicações de computador auxiliares que se comunicam por MAVLink, de alguma forma. [11]

Dentro do Pymavlink, existe um módulo chamado **Mavutil** que fornece mecanismos simples para configurar conexões, enviar e receber mensagens e consultar algumas propriedades básicas do piloto automático, como o modo de voo atualmente ativo. Ele também permite acessar o módulo de dialeto usado para codificar, decodificar e assinar mensagens por meio de um atributo. Os dialetos são os modulos que traduzem o que cada mensagem MAVLink significa, tanto para o MAVLink 1, quanto MAVLink 2.

Essa biblioteca é otimizada para o ArduPilot, mas programar missões com ela diretamente não é recomendado. Assim, seu poder de fato surge quando aplicada em APIs, como o Dronekit, que utilizam os módulos dessa biblioteca para facilitar a programação de missões autônomas.

3.3 Dronekit

O DroneKit é uma plataforma de desenvolvimento de software de código aberto que permite que você controle e interaja com drones usando o Python. É uma ferramenta poderosa para desenvolver aplicativos autônomos e personalizados para drones de forma rápida e prática. [12]

Imagine que você possui um drone equipado com sensores como GPS, giroscópio e câmera. Você quer desenvolver um aplicativo que permita ao drone decolar automaticamente, seguir uma rota predefinida, tirar fotos em pontos específicos e, em seguida, retornar ao local de decolagem para pousar. Usando o DroneKit, você pode escrever esse aplicativo em Python, tanto no computador de bordo, com conexão por telemetria, quanto no computador de bordo, conectado por porta serial.

A API permite que os desenvolvedores criem aplicativos que se comunicam com veículos através do protocolo MAVLink. Ela fornece acesso programático à telemetria, estado e informações de parâmetros de um veículo conectado e possibilita tanto o gerenciamento de missões quanto o controle direto sobre o movimento e as operações do veículo.

O Dronekit possui suporte para as seguintes funções:

- Conectar a uma aeronave (ou várias aeronaves) a partir de um script;
- Obter e definir o estado/telemetria e informações de parâmetros da aeronave;
- Receber notificações assíncronas de mudanças de estado;
- Guiar um drone para uma posição especificada (modo GUIDED);
- Enviar mensagens personalizadas para controlar o movimento do drone;
- Criar e gerenciar missões de pontos de passagem (modo AUTO);
- Sobrescrever as configurações dos canais RC.

3.4 MAVROS

Para não complicar ainda mais esse workshop, falaremos em outra oportunidade sobre o pacote MAVROS. Porém, saiba que essa é uma outra forma de programar tarefas autônomas, utilizando os benefícios da estrutura de software otimizada para robótica do ROS. Esse pacote foi escrito em C++ e também interpreta mensagens MAVLink, disponibilizando uma interface para comunicação com o drone via computador. [13]

Por mais que o ROS seja feito em C++, a biblioteca **rclpy** permite a programação em Python para receber informações de telemetria, mudar parâmetros, alterar modos de voo, enviar comandos de movimentação, e por aí vai. É assim que utilizamos o ROS no nosso subsistema, como será passado mais adiante.

3.5 Software in the Loop

Quando queremos testar o comportamento de um código no drone, é possível fazer isso sem a necessidade de um hardware: rodando em simulação. Para isso, usamos o Software in the Loop (SITL) [14], que permite que o firmware seja executado no próprio computador e simula a física do mundo real e o comportamento dos sensores e atuadores do drone nesse cenário virtual, assim como descrito na Figura 5.

Ademais, é possível conectar o drone simulado com todos os softwares utilizados normalmente, que vão apresentar as mesmas ferramentas disponíveis que para um

drone físico. Desse modo, é possível testar novas funcionalidades e missões sem arriscar a integridade de um drone real.

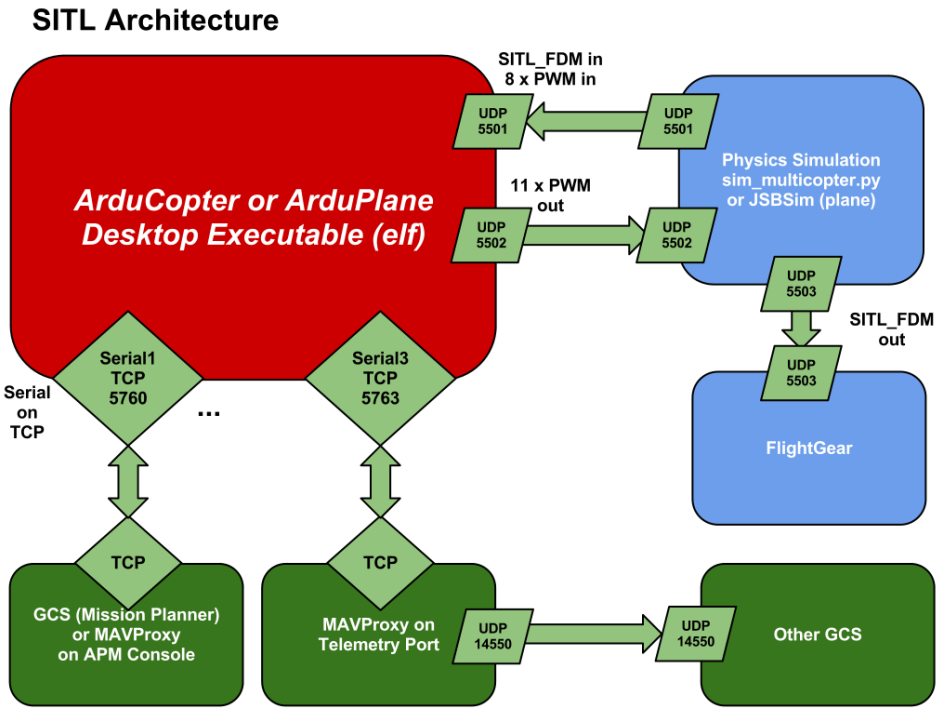


Figura 5: Esquema explicativo do SITL no ArduPilot.

3.6 Exemplo de simulação

Para instalar o ArduPilot SITL e os demais softwares citados, basta seguir o tutorial em nosso GitHub.

Após a instalação, há um exemplo de código comentado utilizando o dronekit para testar em simulação. As principais funções desse código são:

- **vehicle.connect('127.0.0.1:14550')** Conecta o veículo na porta do SITL;
- **vehicle.location.global_frame:** Acessa telemetria do GPS;
- **vehicle.mode.name:** Acessa o modo de voo do drone;
- **vehicle.mode = VehicleMode("GUIDED"):** Troca o modo de voo;
- **vehicle.armed = True:** Arma o drone;
- **vehicle.simple_takeoff(altitude):** Executa um Takeoff;

Para executar o código em simulação, basta rodar o SITL em um terminal e, em outro, rodar o script de python que deseja testar. Para o exemplo mencionado, o resultado final deve ser como mostrado na Figura 6.

```
alessandro@ubuntu20:~/sky_ws/src/sky_sim/missions$ python3 dronekit_test.py

STATUS
Mode: LAND
 Global Location: LocationGlobal:lat=-35.3632624,lon=149.1652381,alt=584.06
 Global Location (relative altitude): LocationGlobalRelative:lat=-35.3632624,lon=149.1652381,alt=0.003
 Local Location: LocationLocal:north=-0.03321707248687744,east=0.057546477764844894,down=0.02666935697197914
 Gimbal status: Gimbal: pitch=None, roll=None, yaw=None
 EKF OK?: True
 Last Heartbeat: 0.10747044900199398
 Is Armable?: True
 System status: STANDBY
 Armed: False

TELEMETRIA
Posição GPS: Lat = -35.3632624, Lon = 149.1652381, Alt = 584.06
Atitude: Roll = -0.0016177181387320l6, Pitch = -0.0017977257957682014, Yaw = -0.12761850655078888
Velocidade: Vx = 0.01, Vy = -0.01, Vz = 0.0
Altitude relativa (m): 0.003

Armando os motores...
Esperando para armar...
Decolando...
Altitude de 10 metros alcançada.

TELEMETRIA
Posição GPS: Lat = -35.3632625, Lon = 149.1652382, Alt = 594.04
Atitude: Roll = -0.0018923040479421616, Pitch = -0.002207038691267371, Yaw = -0.11308390647172928
Velocidade: Vx = 0.01, Vy = -0.01, Vz = 0.0
Altitude relativa (m): 9.989

Iniciando o procedimento de pouso...
Pousando...
Drone pousou com sucesso.

TELEMETRIA
Posição GPS: Lat = -35.3632625, Lon = 149.165238, Alt = 584.14
Atitude: Roll = -0.0014666977804154158, Pitch = -0.0018381497357040644, Yaw = -0.11319892108440399
Velocidade: Vx = 0.01, Vy = -0.01, Vz = 0.02
Altitude relativa (m): 0.088
```

Figura 6: Output do código com dronekit

4 Introdução à linguagem Lua

4.1 O que é Lua?

Lua é uma linguagem de programação leve, poderosa e de propósito geral que foi desenvolvida na Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) no início dos anos 90. Tem uma sintaxe simples e elegante que a torna fácil de aprender e usar, e, portanto, é frequentemente usada em jogos, aplicativos de desktop, servidores web, dispositivos móveis e muitas outras áreas.

No contexto da Skyrats, utilizamos lua para fazer scripts que modificam o comportamento do drone sem que tenhamos que alterar diretamente o código principal do firmware.

4.2 Noções básicas de Lua

Cobriremos a seguir apenas noções bem básicas da linguagem lua, necessárias para programar nossos scripts na equipe. Assim, pressupomos um conhecimento prévio em pelo menos uma linguagem de programação.

4.2.1 Comentários

```
1 -- Comentário de uma linha
2 --[[ Comentário
3     de mais de
4     uma linha ]]
```

4.2.2 Funções

Lua não se importa com identação, no entanto é útil na organização do código. Toda vez que for criar uma função deve-se usar o **function** no inicio e o **end** no final. Variáveis podem receber uma função e serem chamadas, no entanto o mais usual é o primeiro jeito do exemplo.

```
1 -- Jeito usual de criar funções
2 function drone()
3         print("Drone")
4 end
5
6 -- Atribuindo uma função a uma variável
```

```
tello = function()
        print("Tello")
end

-- Chamando as funções
drone() -- Drone
tello() -- Tello
```

A seguir, há exemplos de funções utilizando argumentos. Observe que a concatenação de strings em Lua ocorre por meio de **..**

```
function square(n)
        return n * n
end

function say_hello(name)
        print("Hello" .. name)
end
```

4.2.3 Símbolos

```
-- Aritméticos --
+ -- 3 + 2 = 5
- -- 3 - 2 = 1
* -- 3 * 2 = 6
/ -- 3 / 2 = 1.5
% -- 3 % 2 = 1
^ -- 3 ^ 2 = 9

-- Relacionais --
== -- igual
~= -- não igual
>  -- maior que
<  -- menor que
>= -- maior ou igual
<= -- menor ou igual

-- Lógicos --
and
```

```
19 or
20 not
21
22 -- Outros --
23 .. -- concatenação de strings
24 #  -- retorna o tamanho de uma string ou lista
```

Além disso, o equivalente ao null de outras linguagens, em Lua, é o **nil**

4.2.4 Condicionais

Em Lua, todos os blocos de condições devem ser terminados com **end** e a cada condição, deve-se utilizar o **then** (exceto o else).

```
1  skyrats = true
2
3  if skyrats then
4          print("Que bom que você faz parte da Skyrats!!")
5  end
6
7  nota = 7
8
9  if nota >= 5 then
10         print("Parabéns, você foi aprovado")
11 elseif nota >= 3 then
12         print("Você ainda não foi aprovado, mas pode fazer a rec")
13 else
14         print("Você foi reprovado")
15 end
16
```

4.3 Lua Demo

Para testar códigos simples e checar comandos e sintaxe, o site de Lua apresenta um ambiente para você escrever e rodar códigos nessa linguagem: Lua Demo

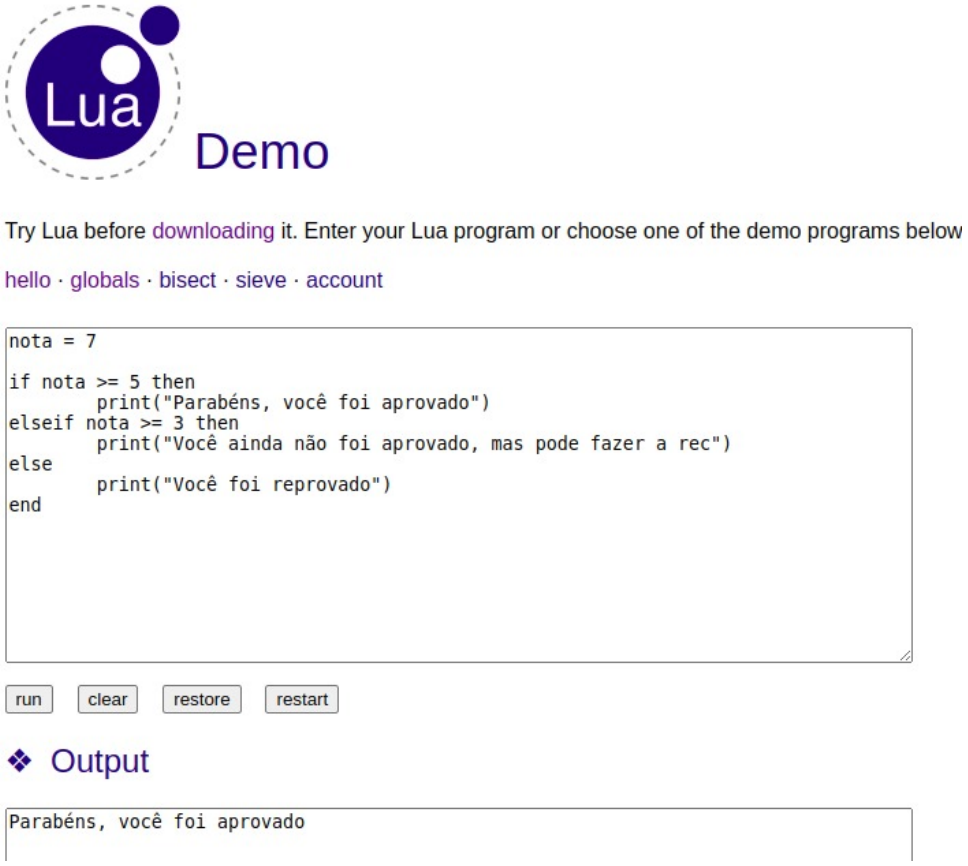


Figura 7: Lua Demo

5 Scripts de Lua no Ardupilot

No contexto do Ardupilot, os scripts de lua sevrem para modificar e adicionar comportamentos ao autopilot, sem que seja necessário alterar o código principal do firmware. Assim, é possível rodá-los em paralelo ao código principal, monitorando e alterando o estado do drone. Os scripts são armazenados no cartão SD da controladora de voo. Uma visão geral e instrucional sobre scripts de lua encontra-se no site do ardupilot. [15]

5.1 Estrutura dos scripts

No código abaixo, criamos uma função chamada **update**, que é a função princial do código. A última linha indica que, na primeira vez que o código rodar, após 1000ms, a função update será chamada. Dentro da função update, a linha 4 indica que, 2000ms após ela terminar sua execução, a função update será chamada novamente,

ou seja, ela será chamada recursivamente enquanto o código estiver rodando. A função pronta **gcs:send_text()** utilizada já vem na API no ardupilot e será discutida mais adiante.

```
function update()
    gcs:send_text(6, "Hello Skyrats :) ")
    return update, 2000
end
return update(), 1000

```

5.2 API e métodos

No GitHub do ardupilot, é possível encontrar a API com todos os métodos e funções que podem ser utilizados nos drones.

No exemplo abaixo, temos a função gcs:send_text(), cujos parâmetros **severity** e **text** são explicados nos comentários do código. Desse modo, a API é útil para consultar como utilizar as funções e entender o que cada um de seus parâmetros representa.

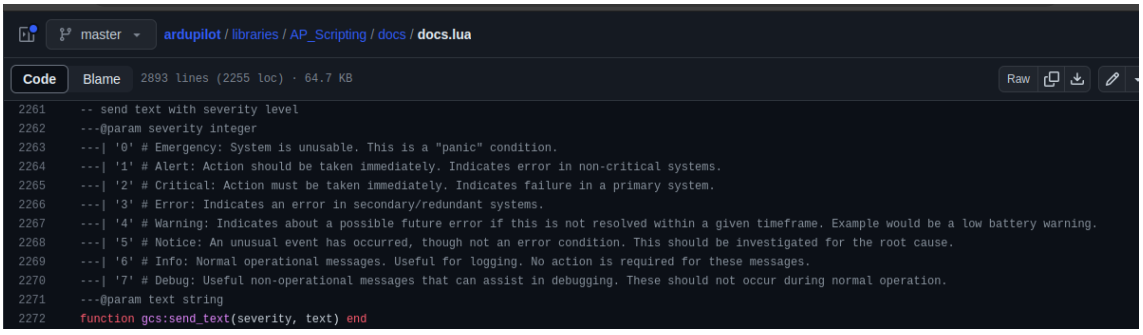


Figura 8: Explicação da função gcs:send_text() na API do ardupilot.

5.3 Objetos e Bibliotecas

Ao fazer códigos mais complexos em lua, devemos utilizar alguns tipos de dados e métodos especiais para a interação com o drone. Assim, fazemos usos de alguns objetos e bibliotecas específicos. Alguns deles são:

Objetos

- **Location:** armazena uma localização, expressa em latitude, longitude e altitude;
- **Vetor2f:** vetor 2D que armazena floats;
- **Vetor3f:** vetor 3D que armazena floats;

Bibliotecas

- **AHRS:** Acessa informações da posição e estado do drone
- **Arming:** Acessa comandos para armar, desarmar e verificar o estado de arming do drone
- **Battery:** Acessa informações da bateria do drone
- **GPS:** Acessa informações do GPS do drone
- **GCS:** Envia mensagens para a Ground Control Station
- **Vehicle:** Acessa e modifica o modo de voo do drone

Cada uma dessas bibliotecas apresenta diversos métodos para acessar dados do drone e controlá-lo. Para saber quais são e entender o que cada um deles faz, consulte a página de scripts de lua do ardupilot.

5.4 Exemplos e Applets

No GitHub do ardupilot é possível encontrar diversos exemplos de scripts e também alguns applets que, diferentemente dos exemplos, não requerem nenhuma edição de código e já estão prontos para serem usados, contendo, cada um, um arquivo README associado, explicando sua função e como usá-lo.

A Skyrats também tem um repositório (Lua-scripts), ainda em construção, com exemplos de scripts de lua, todos testados em simulação e alguns, também no drone físico. Além disso, alguns scripts possuem um arquivo **.md** associado, com a explicação do código.

5.5 Entendendo um código

No repositório mencionado no tópico anterior, há o passo a passo detalhado de um código que faz o drone ir para a frente e para trás um certo número de vezes. Ele apresenta funções que são úteis em diversos contexto. Assim, o bom entendimento dele ajudará na elaboração de outros códigos nesse mesmo nível. O código completo pode ser encontrado aqui.

5.6 Rodar o script em simulação

Para testar os scripts de Lua em simulação utilizando a QGround e o SITL do ardupilot, basta seguir os seguintes passos:

1. Rodar a qground (./QGroundControl.AppImage);
2. Rodar o SITL (ardupilot/ArduCopter/sim_vehicle.py);
3. Habilitar os scripts na QGS :
    - Clicar no ícone com o Q no canto superior esquerdo;
    - Ir em Vehicle Setup;
    - Ir em parameters;
    - Buscar por "SCR_ENABLE"e selecioná-lo;
    - Mudar o valor de None para Lua (ou de 0 para 1);
    - Salvar (Talvez seja necessário encerrar e rodar novamente a simulação);
4. Será criada uma pasta "scripts"na pasta em que a simulação estiver rodando (ardupilot/ArduCopter)
5. Adicione os script de lua nessa pasta e, ao rodar novamente o STIL, todos os scripts da pasta serão executados;

5.7 Rodar o script no drone real

Para rodar os scripts no drone real, devemos enviar os arquivos de lua para o cartão SD do drone. Abaixo está o passo a passo:

1. Retirar o cartão SD do drone;
2. Abrir o conteúdo do cartão SD no computador (Talvez seja necessário o uso de um adaptador para leitura de micro SD);
3. O SD deve conter uma pasta **APM** e, dentro dela, uma pasta **scripts**. Se não houver, crie as duas;
4. Adicione ou modifique scripts de lua na pasta APM/scripts;
5. Devolva o SD para o drone (Certifique-se de que encaixou certo, pois é bem fácil colocar errado);

Agora, para configurar os scripts pela QGround:

1. Rodar a qground (./QGroundControl.AppImage);

2. Conecte o Drone ao seu computador (com USB diretamente ou por telemetria);
3. Habilitar os scripts na QGS :
    - Clicar no ícone com o Q no canto superior esquerdo;
    - Ir em Vehicle Setup;
    - Ir em parameters;
    - Buscar por "SCR_ENABLE"e selecioná-lo;
    - Mudar o valor de None para Lua (ou de 0 para 1);
    - Salvar (Talvez seja necessário conectar e desconectar o drone);
4. Caso esteja ocorrendo um erro de scripts out of memory:
    - Clicar no ícone com o Q no canto superior esquerdo;
    - Ir em Vehicle Setup;
    - Ir em parameters;
    - Buscar por "SCR_DIR_DISABLE e marcar a caixinha com ROMFS (significa que ele não irá buscar scripts na pasta ROMFS, e sim na pasta APM/scripts);
    - Salvar (Talvez seja necessário conectar e desconectar o drone);

Pronto, agora assim que você ligar o drone os scripts já serão executados.

6 Exercício

O exercício será dividio em pequenas etapas, para aplicar cada conhecimento desse workshop. As instalações e instruções de como rodar são baseadas nesse tutorial.

1. Instalar e rodar o SITL no terminal;
2. Instalar e baixar a QGround. Fazer uma missão de waypoints na QGround;
3. Utilizar o MAVProxy em simulação para: mudar o modo de voo para GUIDED, armar o drone, dar um takeoff de 10m e mudar para o modo LAND;
4. Rodar o código teste do dronekit e controlar o drone para printar as informações de telemtria de cada função;
5. Seguir o tutorial da seção 5.6 e rodar um hello_world em Lua e simulação;
6. **DESAFIO** Pesquisar, criar e testar em simulação uma nova funcionalidade para os scripts de Lua, ou programar uma missão completa utilizando o dronekit. Sejam criativos!

Os integrantes do grupo devem disponibilizar um vídeo com a demonstração de todas as etapas solicitadas acima funcionando em simulação. Além disso, devem produzir um arquivo *markdown* para documentar as dificuldades encontradas e resultados obtidos, assim como a descrição do que foi criado no desafio e explicação dos respectivos scripts.

Referências

[1] O que é um firmware. https://en.wikipedia.org/wiki/Firmware.

[2] Lista de firmwares open-source. https://www.libhunt.com/topic/drone.

[3] Lista de controladoras de voo. https://ardupilot.org/copter/docs/common-autopilots.html#autopilot-hardware-options.

[4] Protocolo mavlink. https://mavlink.io/en/.

[5] Estrutura da mensagem mavlink. https://ardupilot.org/dev/docs/mavlink-basics.html.

[6] Vehicle types. https://ardupilot.org/ardupilot/docs/common-all-vehicle-types.html.

[7] Discord do ardupilot. https://ardupilot.org/discord.

[8] Como funciona o ardupilot. https://ardupilot.org/dev/docs/learning-the-ardupilot-codebase.html.

[9] Descrição das ground control stations. https://ardupilot.org/plane/docs/common-choosing-a-ground-station.html.

[10] Mavproxy. https://ardupilot.org/mavproxy/.

[11] Pymavlink. https://mavlink.io/en/mavgen_python/.

[12] Dronekit. https://dronekit-python.readthedocs.io/en/latest/about/index.html.

[13] Mavros. http://wiki.ros.org/mavros.

[14] Sitl no ardupilot. https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html.

[15] Scripts de lua no ardupilot. https://ardupilot.org/copter/docs/common-lua-scripts.html.