Aplicação *WEB* para controle de VANTS: utilização da plataforma *Raspberry Pi* como computador de bordo.

João Luiz de Amorim Pereira Neto (joao_luiz@id.uff.br)
Lúcio Folly Sanches Zebendo (luciozebendo@id.uff.br)

Universidade Federal Fluminense

15/12/2022

Sumário

Contextualização

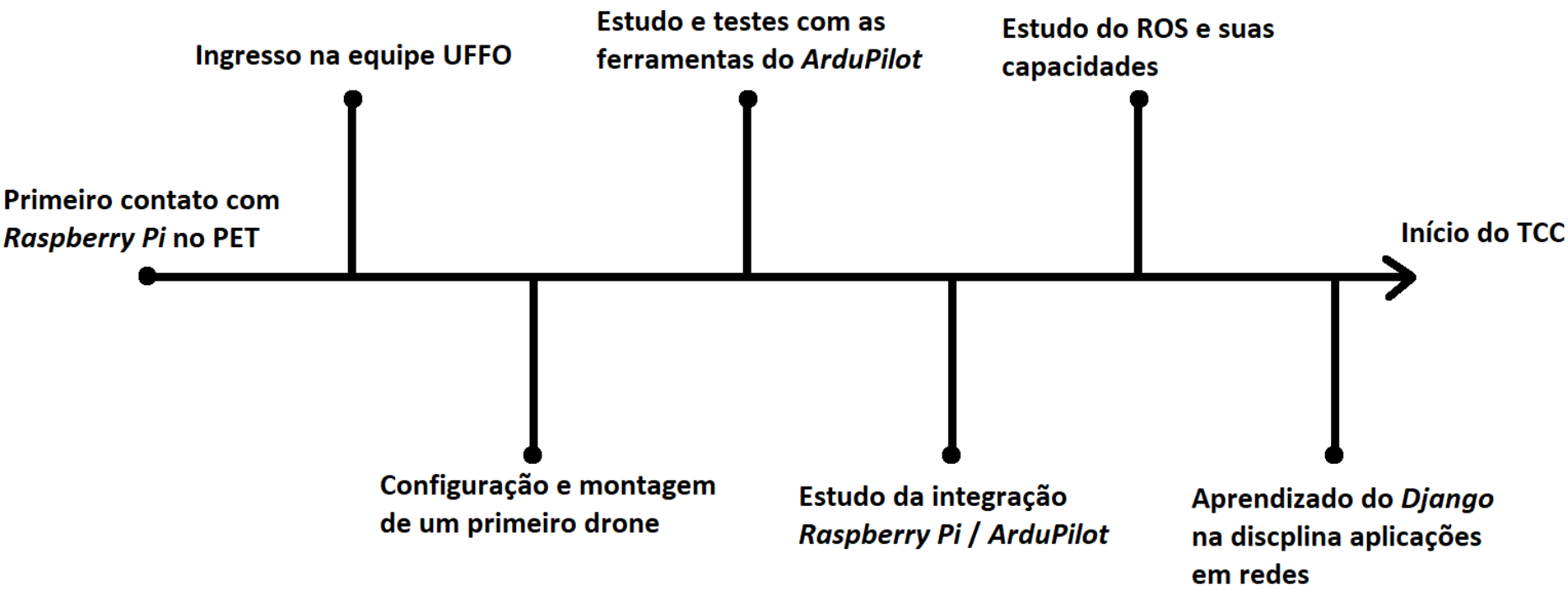


Figura 1: Linha do tempo até o início do TCC.

Motivações

- Neste trabalho, deseja-se elaborar um sistema para controle e navegação de *drones* por meio da *Internet*.

Motivações

- Neste trabalho, deseja-se elaborar um sistema para controle e navegação de *drones* por meio da *Internet*.
- Atualmente, os códigos disponíveis no mercado são pouco customizáveis e restritos a poucas funcionalidades.

Motivações

- Neste trabalho, deseja-se elaborar um sistema para controle e navegação de *drones* por meio da *Internet*.
- Atualmente, os códigos disponíveis no mercado são pouco customizáveis e restritos a poucas funcionalidades.
- Acredita-se que, partindo de tecnologias como o ROS e *Django*, é possível criar um sistema robusto, capaz de prover serviços de navegação e controle. Além disso, pode-se ainda ser altamente customizável, atendendo a boa parte das demandas do mercado.

Sumário

Alguns componentes dos VANTs de asa rotativa

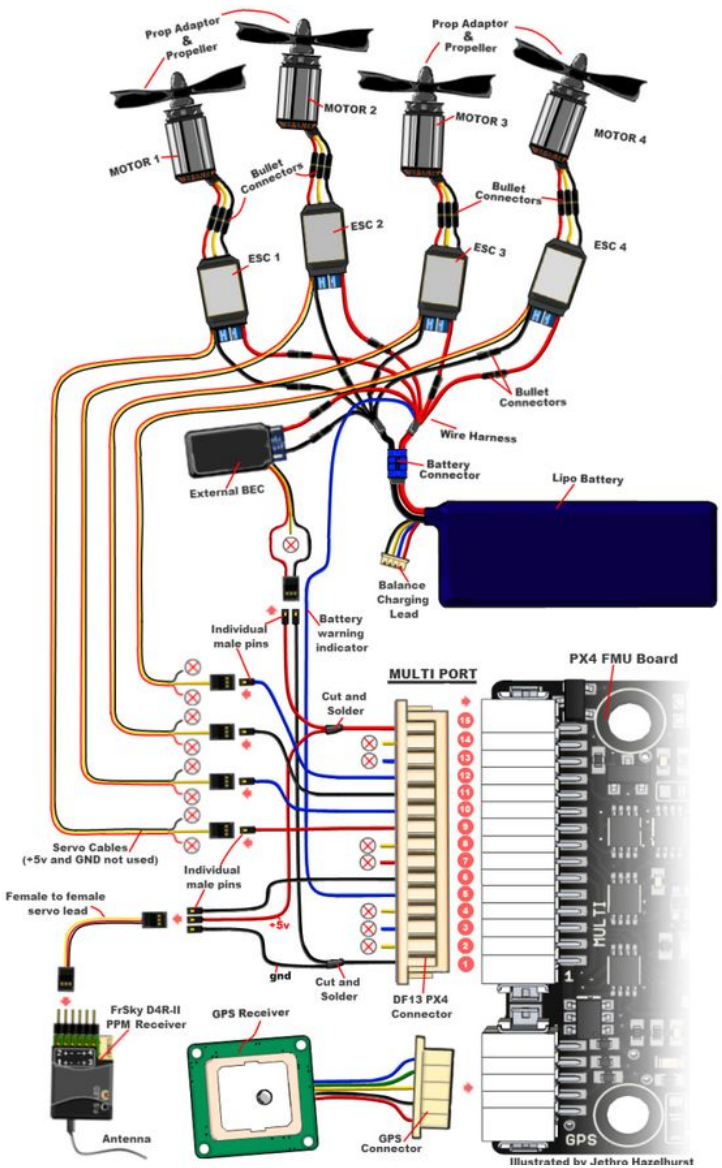
Alguns componentes dos VANTs de asa rotativa:

1. *Frame*
2. Motores e hélices
3. ESCs (*Electronic Speed Controllers*)
4. Controladora de voo
5. Bateria e placa distribuidora de energia

Alguns componentes dos VANTs de asa rotativa

Alguns componentes dos VANTs de asa rotativa:

1. *Frame*
2. Motores e hélices
3. ESCs (*Electronic Speed Controllers*)
4. Controladora de voo
5. Bateria e placa distribuidora de energia



Projeto *ArduPilot*

O *ArduPilot* é um projeto de código aberto que possui um conjunto de *softwares* destinado a realizar a função de piloto automático para veículos não tripulados, tais como:

- Plane: piloto automático para *drones* de asa fixa.
- Copter: piloto automático para *drones* de asa rotativa.
- Rover: piloto automático para veículos terrestres e barcos.
- Sub: piloto automático para veículos submarinos.

Protocolo *MAVlink*

O *MAVlink* é um protocolo de comunicação utilizado com *drones* e entre os componentes acoplados ao *drone*, com as seguintes características:

- O protocolo define um grande conjunto de mensagens, que podem ser encontradas na documentação da comunidade *ArduPilot*.

Protocolo *MAVlink*

O *MAVlink* é um protocolo de comunicação utilizado com *drones* e entre os componentes acoplados ao *drone*, com as seguintes características:

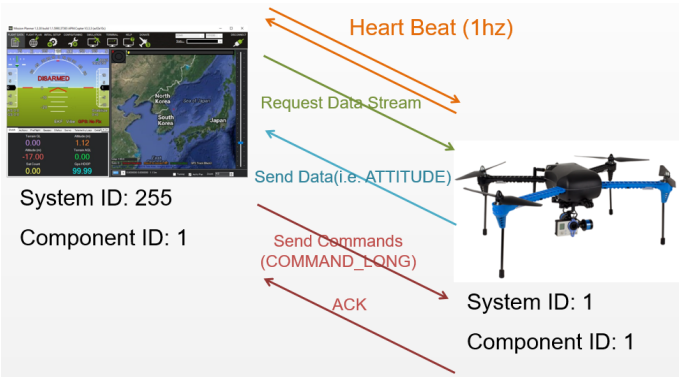
- O protocolo define um grande conjunto de mensagens, que podem ser encontradas na documentação da comunidade *ArduPilot*.
- Mensagens MAVLink podem ser transmitidas através de qualquer conexão serial e independem da tecnologia utilizada, que pode ser WiFi, 900 MHz, rádio, entre outros.

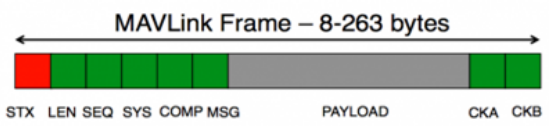
Protocolo *MAVlink*

O *MAVlink* é um protocolo de comunicação utilizado com *drones* e entre os componentes acoplados ao *drone*, com as seguintes características:

- O protocolo define um grande conjunto de mensagens, que podem ser encontradas na documentação da comunidade *ArduPilot*.
- Mensagens MAVLink podem ser transmitidas através de qualquer conexão serial e independem da tecnologia utilizada, que pode ser WiFi, 900 MHz, rádio, entre outros.
- As mensagens transmitidas não possuem garantia de serem entregues. Um sistema *GCS* ou um computador de bordo devem checar o estado do veículo, para determinar se o comando foi executado.

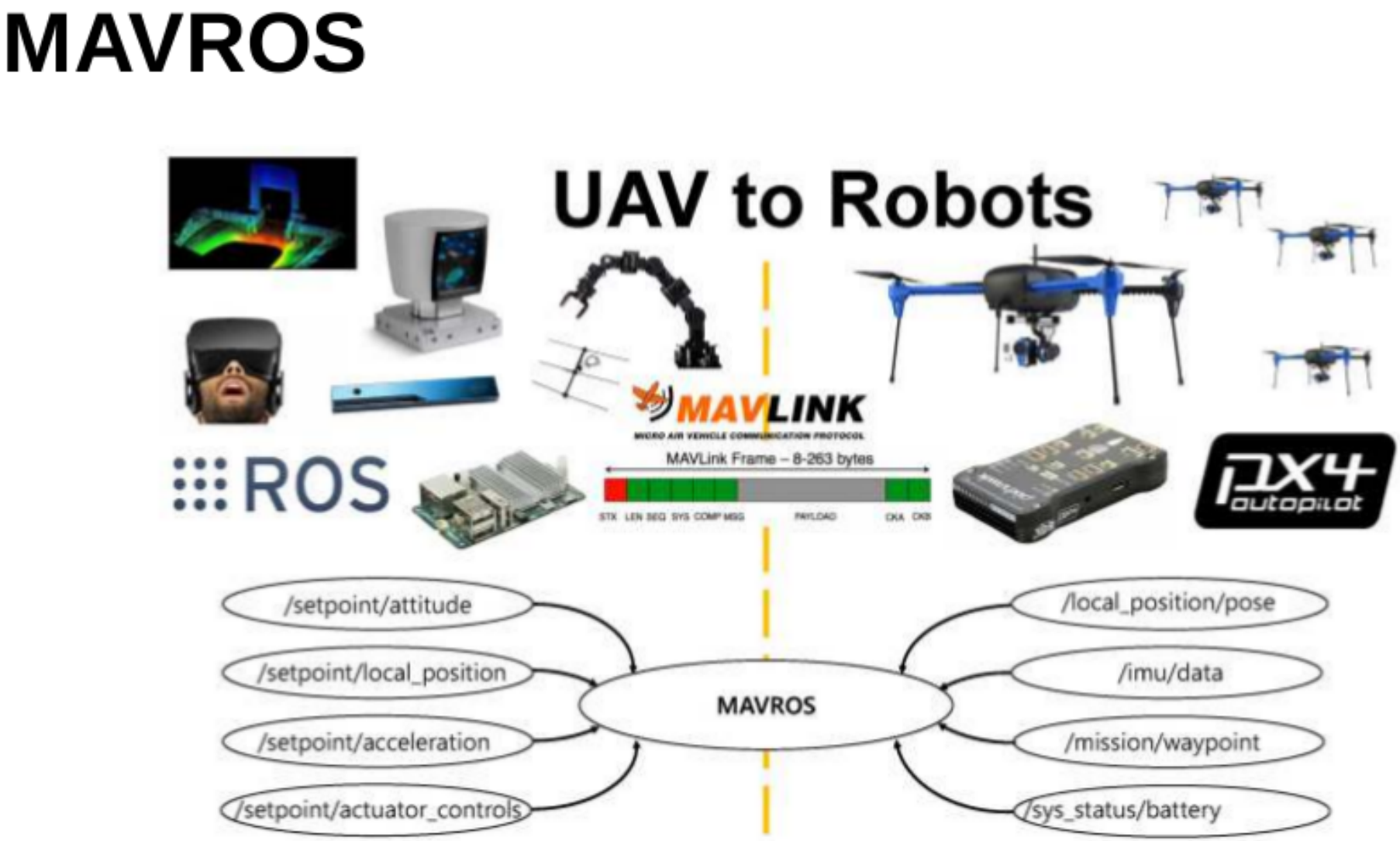
Protocolo *MAVlink*





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).	

MAVROS


MAVROS
UAV to Robots
MAVLINK
MICRO AIR VEHICLE COMMUNICATION PROTOCOL
MAVLink Frame – 8-263 bytes
STX LEN SEQ SYS COMP MSG PAYLOAD CKA CKB
ROS
px4 autopilot
/setpoint/attitude
/setpoint/local_position
/setpoint/acceleration
/setpoint/actuator_controls
MAVROS
/local_position/pose
/imu/data
/mission/waypoint
/sys_status/battery

Sumário

Controle via *Software In The Loop*

O código fonte do *ArduPilot* pode ser compilado em um ambiente de simulação. O *Software In The Loop* (SITL) integra e controla todos os programas durante um primeiro experimento.

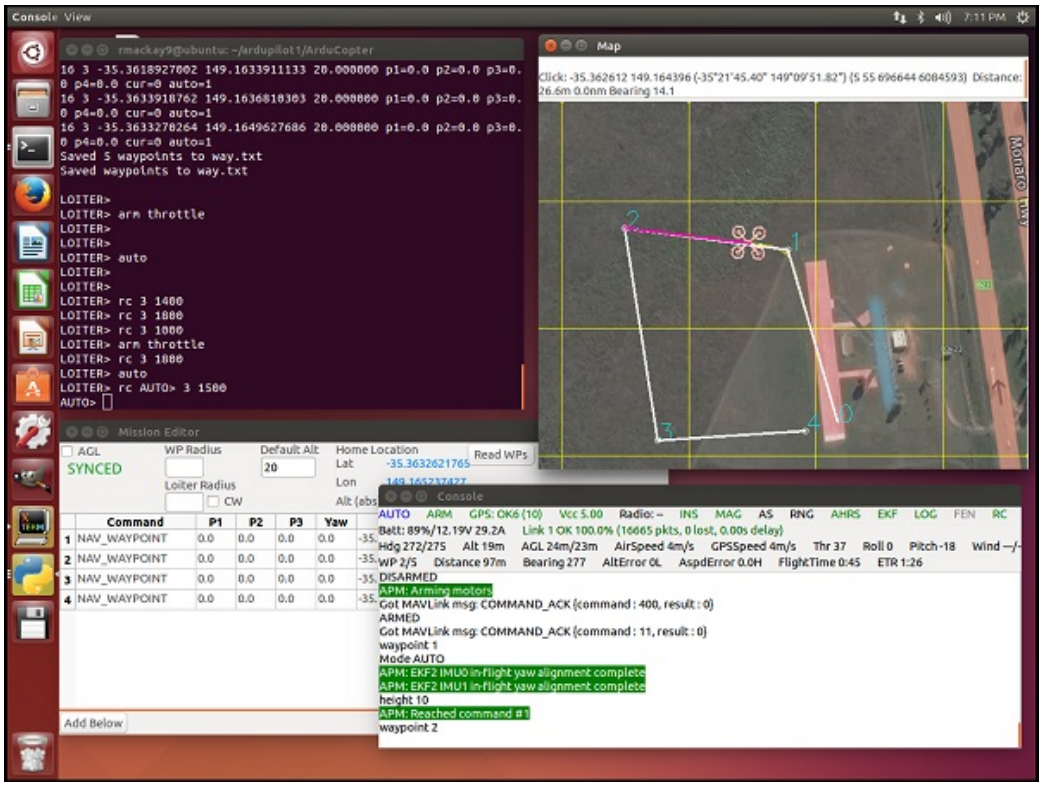


Figura 2: Telas da aplicação *Software In The Loop* (SITL).

Controle via *Software In The Loop* no *campus* da UFF

Em um segundo experimento, foram usados alguns parâmetros para simular um ambiente acessível, onde o *drone* sobrevoa o *campus* da UFF.

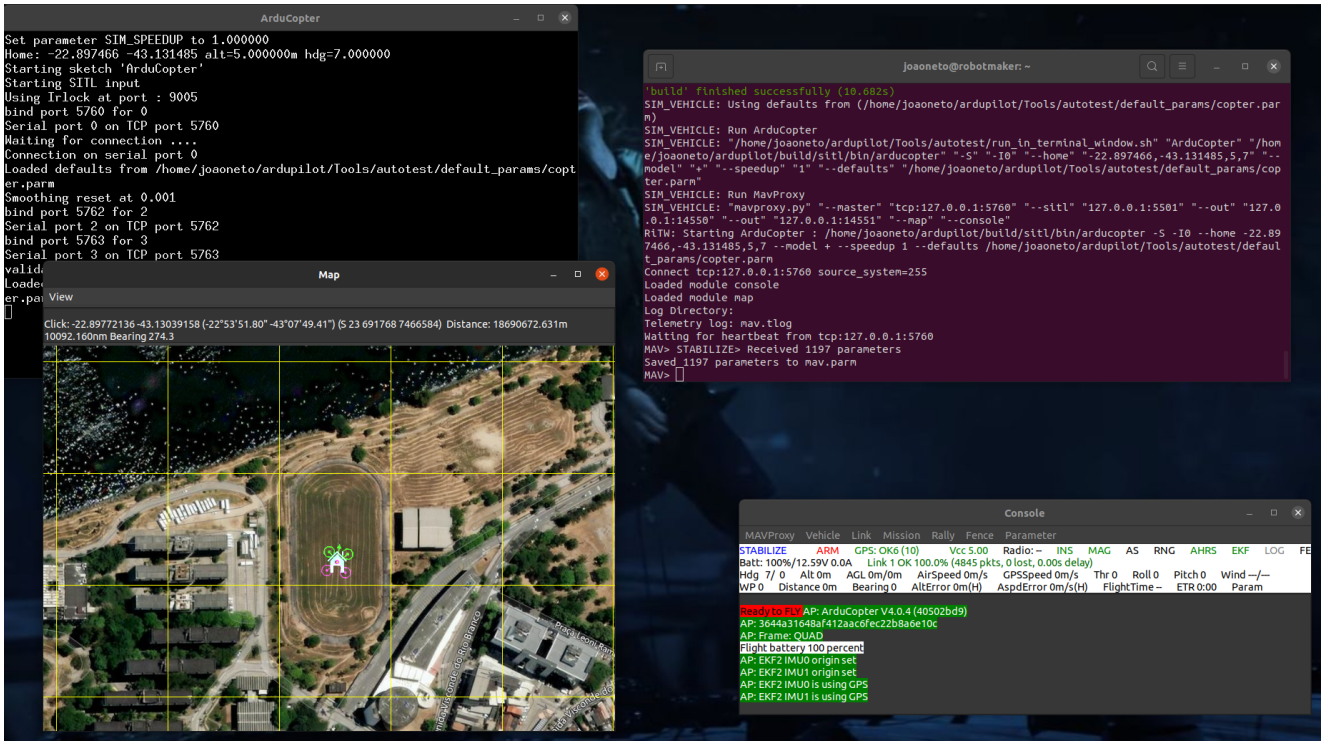


Figura 3: Telas da aplicação *Software In The Loop* (SITL) em ambiente real.

Controle via *Software In The Loop* com *Gazebo*

Em um terceiro experimento, foi passado um parâmetro ao SITL para que houvesse a integração com o *software Gazebo*.

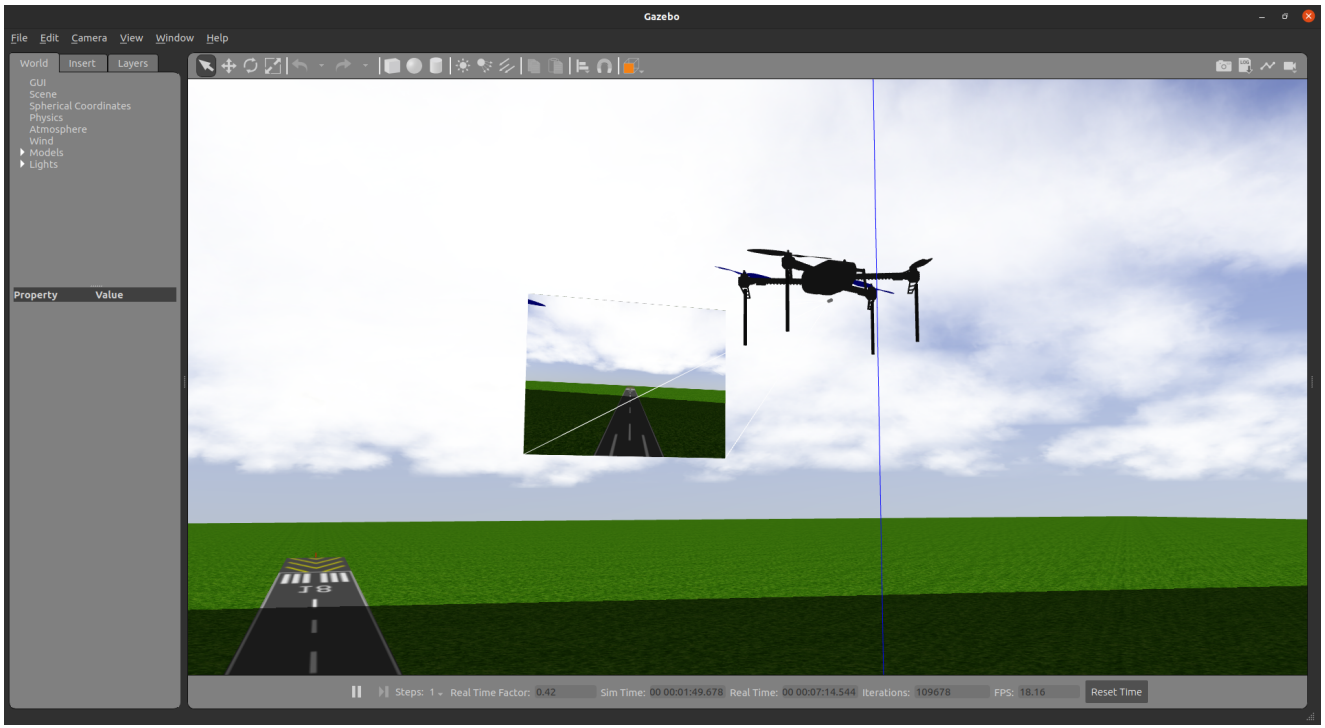


Figura 4: Telas da aplicação *Software In The Loop* (SITL) com *Gazebo*.

Sumário

Descrição do problema

- Atualmente existem poucas alternativas de comunicação com *drones* via *Internet*. Dentre as possibilidades, grande parte está limitada a poucas funcionalidades e nenhuma customização.

Descrição do problema

- Atualmente existem poucas alternativas de comunicação com *drones* via *Internet*. Dentre as possibilidades, grande parte está limitada a poucas funcionalidades e nenhuma customização.
- A partir disso, surgiu a ideia de unir algumas ferramentas, como *ROS*, *Django* e *Javascript* para elaborar um sistema robusto e customizável.

Descrição do problema

- Atualmente existem poucas alternativas de comunicação com *drones* via *Internet*. Dentre as possibilidades, grande parte está limitada a poucas funcionalidades e nenhuma customização.
- A partir disso, surgiu a ideia de unir algumas ferramentas, como *ROS*, *Django* e *Javascript* para elaborar um sistema robusto e customizável.
- Vale destacar que a solução apresentada nesse trabalho oferece alta abstração de *hardware*, permitindo que diferentes controladoras utilizem o sistema.

Perguntas conceituais

As seguintes perguntas surgiram no início do detalhamento do problema:

1. Qual arquitetura de *software* deve ser utilizada?
   Isso indica como o *software* a ser desenvolvido será organizado.
2. Qual arquitetura de rede a ser utilizada?
   Por exemplo, par-a-par ou cliente-servidor?
3. No quesito de comunicação, qual deve ser o melhor conjunto de protocolos a serem selecionados?

Sumário

Drone quadcóptero

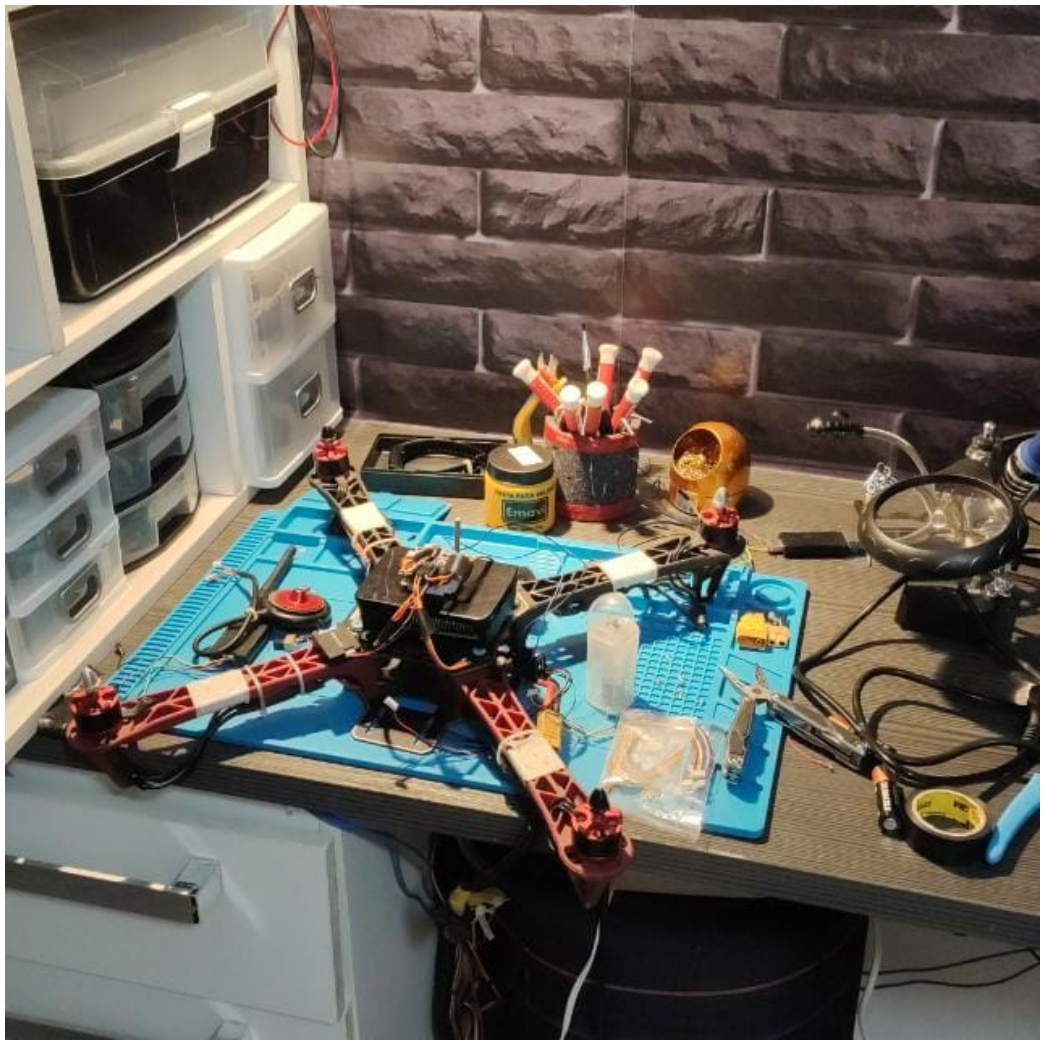


Figura 5: Fotografia do *drone* quadcóptero.

Raspberry Pi

Figura 6: Imagem do *Raspberry Pi 3B*.

Esquemático proposto

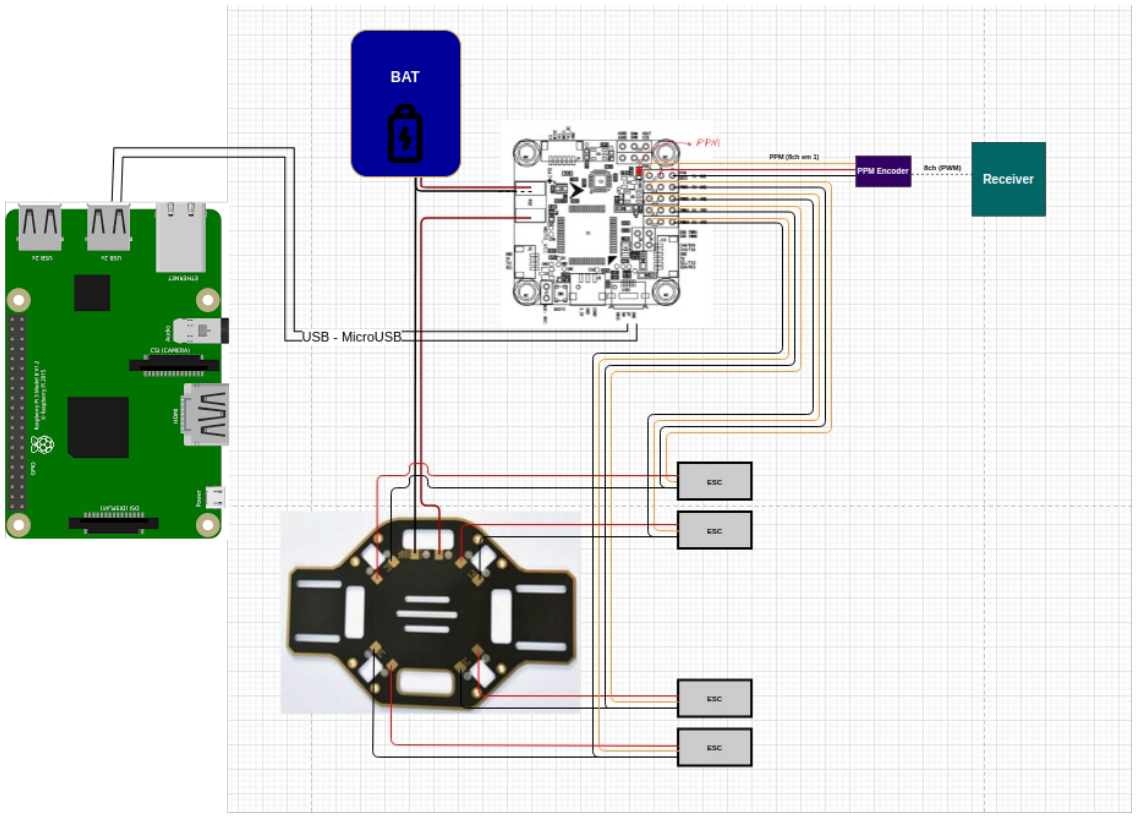


Figura 7: Conexões de *hardware* para Omnibus F4 PRO V3.

Hardware para o *drone* hexacóptero

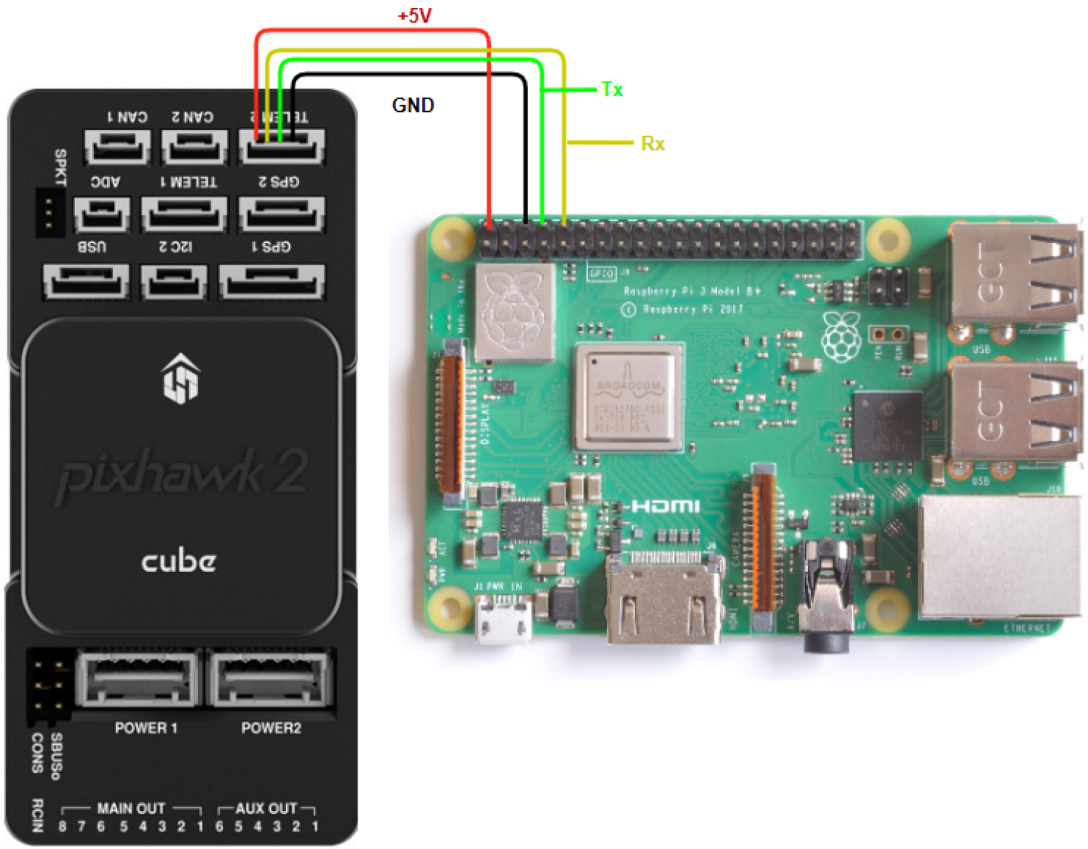


Figura 8: Diagrama de conexão entre o *Raspberry Pi* e a nova controladora de voo (*Pixhawk 2 Cube Hero*).

Sumário

Arquitetura de *software* proposta

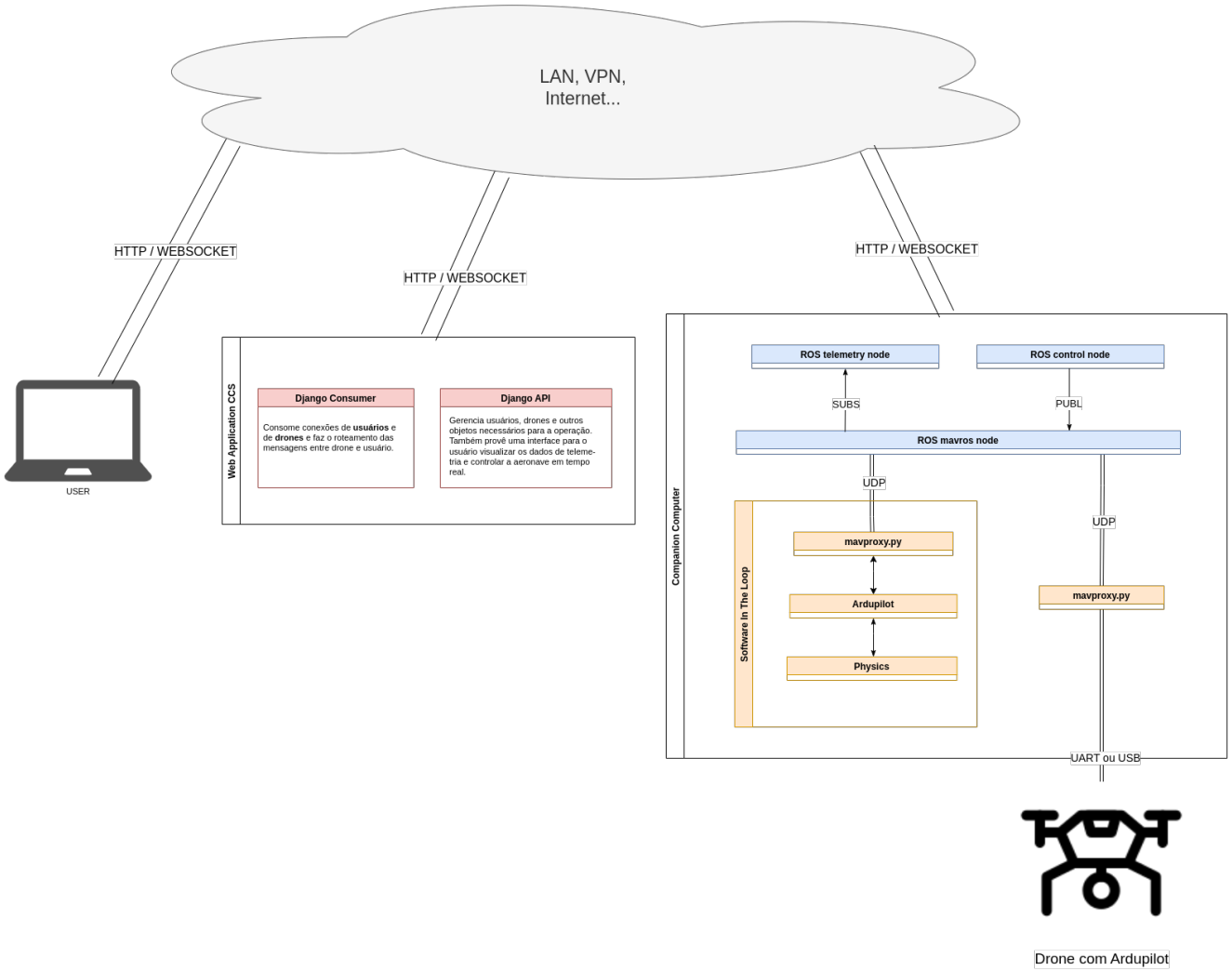


Figura 9: Arquitetura de *software* proposta.

Funcionamento da *Cloud Control Station* (CCS)

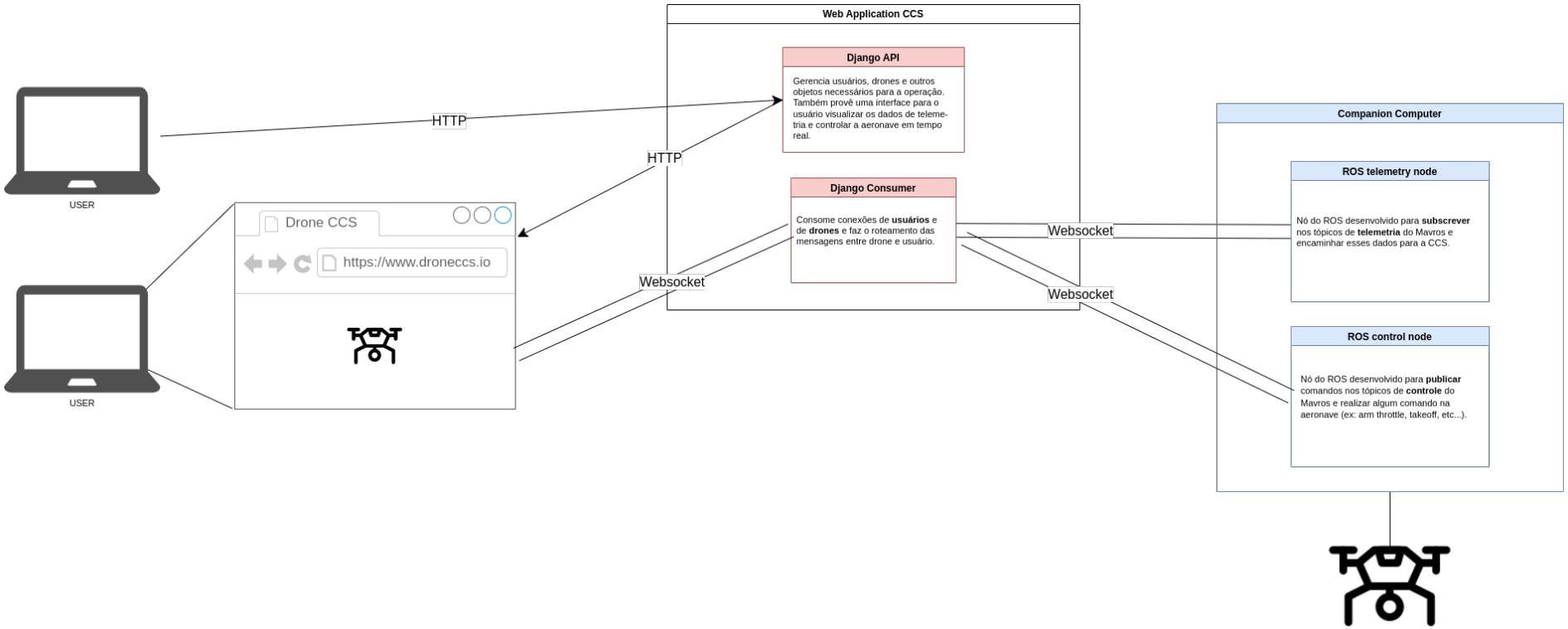


Figura 10: Diagrama de comunicação da CCS.

Autenticações realizadas

- Autenticação via *JWT (JSON Web Token)* para o usuário.

Autenticações realizadas

- Autenticação via *JWT (JSON Web Token)* para o usuário.
- Autenticação via chave de *API* para o *drone*.

Autenticações realizadas

- Autenticação via *JWT (JSON Web Token)* para o usuário.
- Autenticação via chave de *API* para o *drone*.
- Autenticação via chave de sessão para o usuário (necessária para utilização da *interface* do *Django REST Framework*).

Diagrama das autenticações

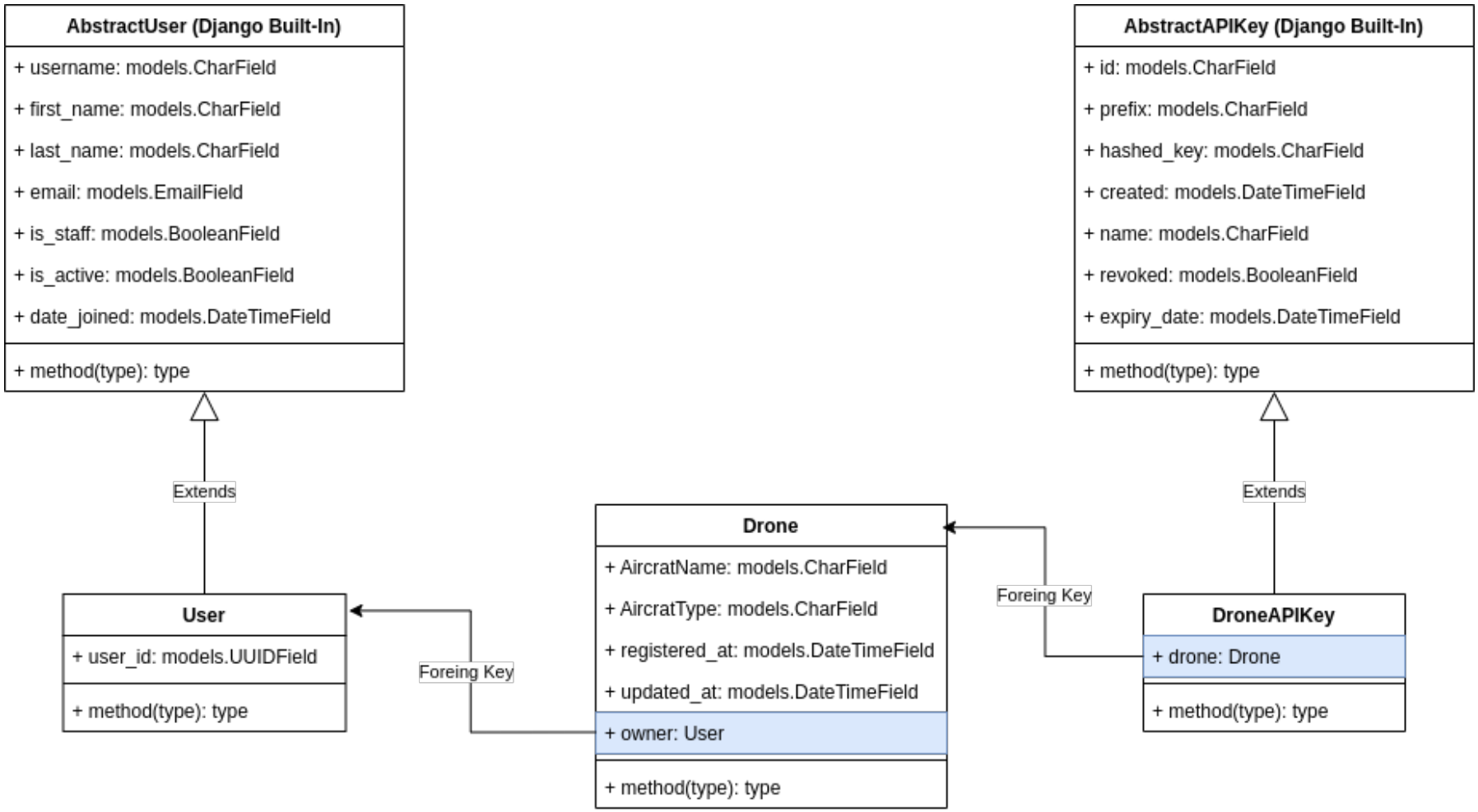


Figura 11: Diagrama de UML dos modelos do *Django* para autenticação.

Registro dos usuários

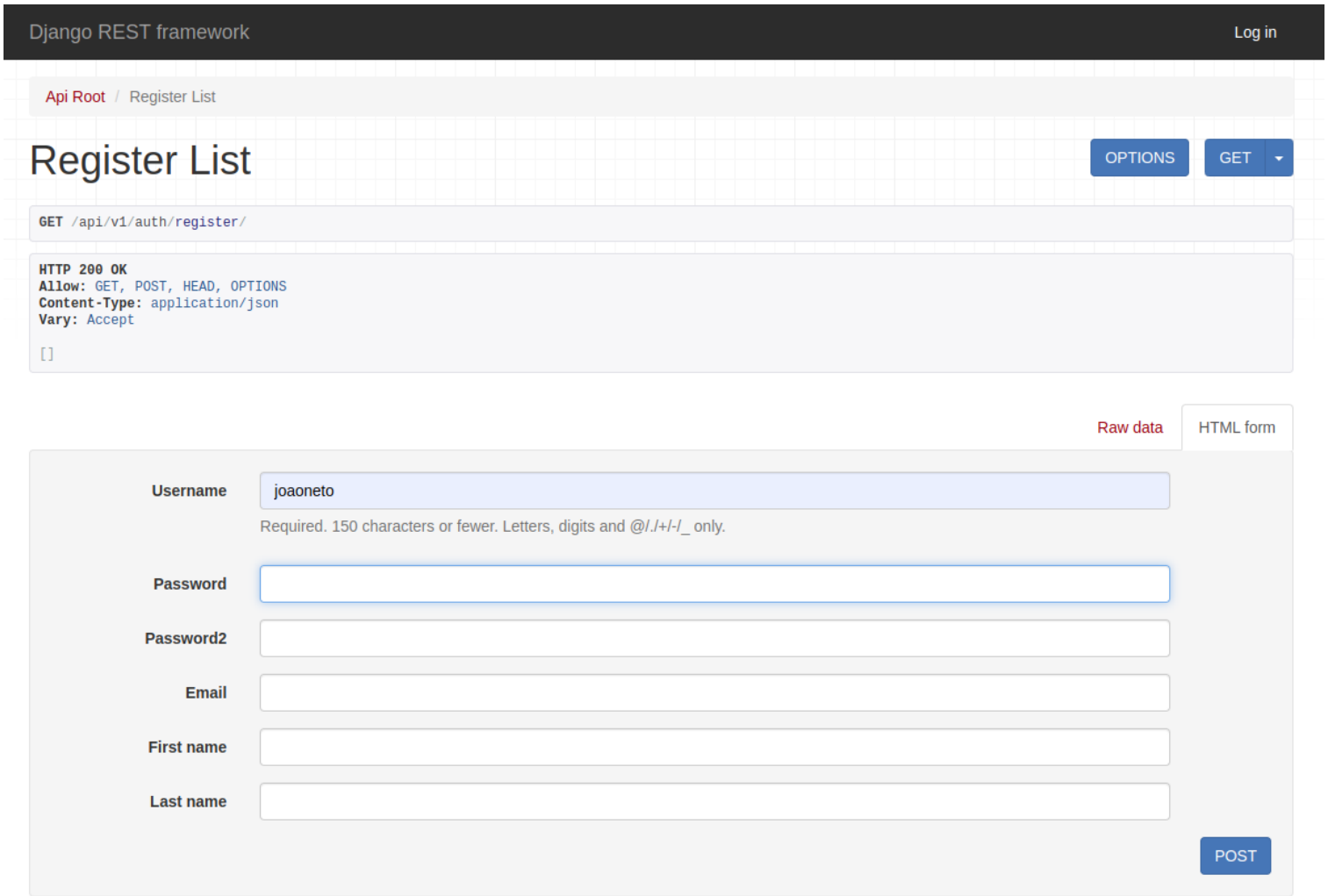


Figura 12: Tela de registro dos usuários.

Registro dos *drones*

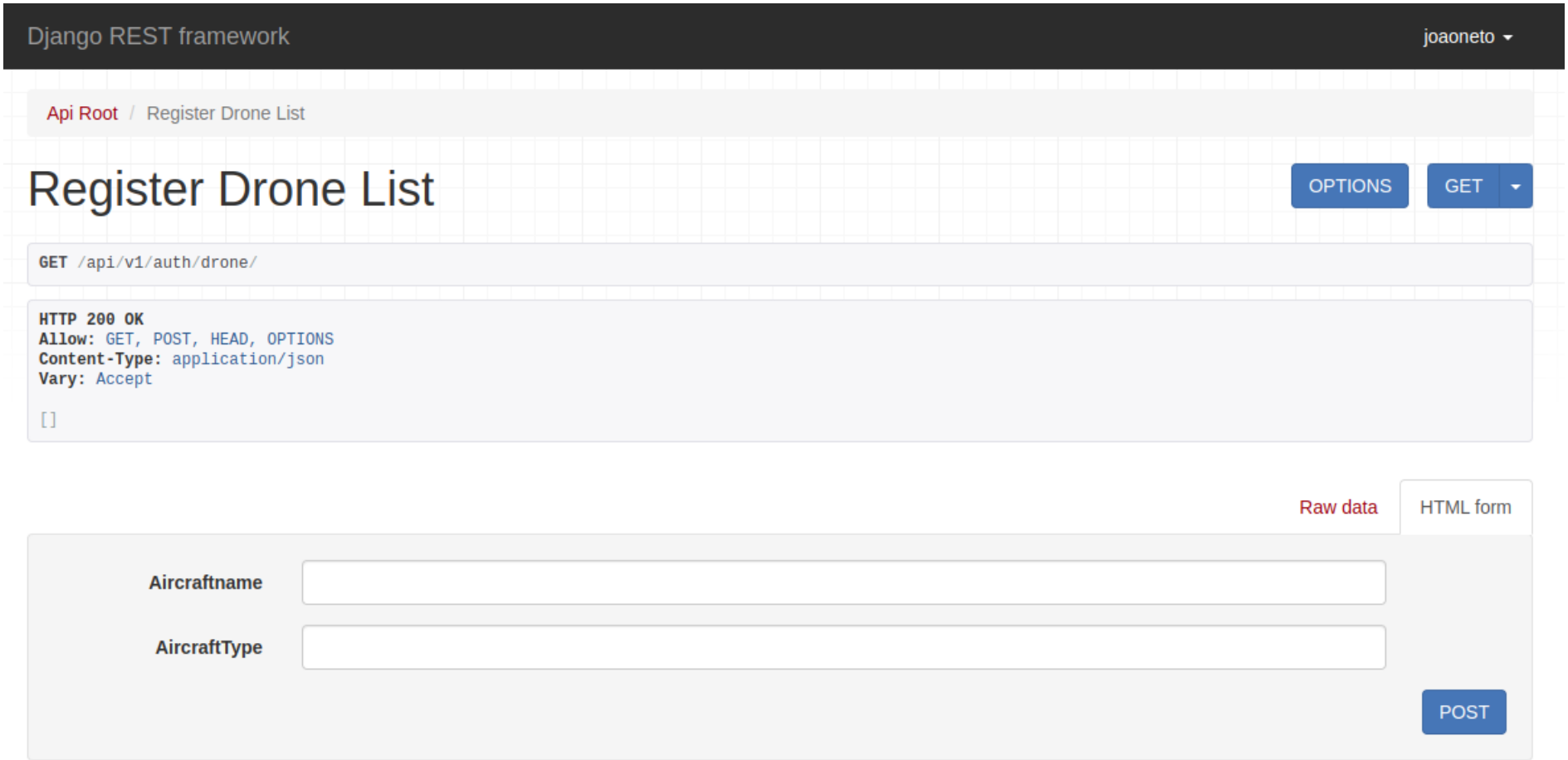


Figura 13: Tela de registro dos *drones*.

Registro dos *drones*

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "api_key": "quad",
    "key": "4m6fUrax.0nSIvSZ4guTF8sGcwgJgifYJ8Oi9mCtu"
}
```

Figura 14: Tela de resposta, após cadastro do *drone*.

Gerenciamento das missões

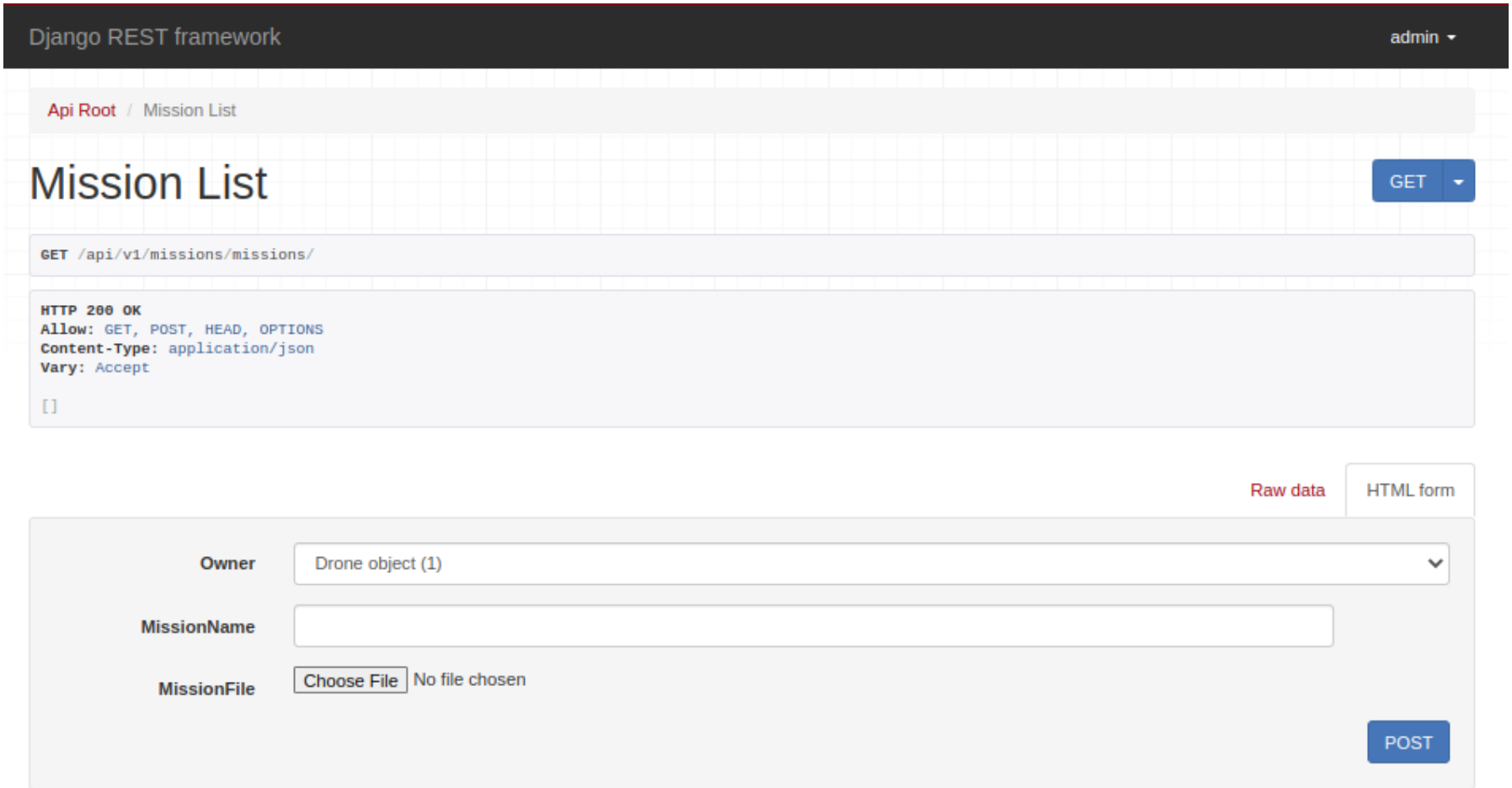


Figura 15: Tela de gerenciamento das missões.

Arquivo de missão

Open | mission.txt ~/apmplanner2/missions | Save

```
QGC WPL 110
0	1	0	16	0	0	0	0	-22.8974659000000003	-43.1314849999999979	5.09000015258789063	1
1	0	3	16	0	0	0	0	-22.8974534210314289	-43.1316885352134705	20	1
2	0	3	16	0	0	0	0	-22.8970383162211135	-43.131726086139679	20	1
3	0	3	16	0	0	0	0	-22.8970185492937048	-43.1312298774719238	20	1
4	0	3	16	0	0	0	0	-22.897243397923063	-43.1312379240989685	20	1
5	0	3	20	0	0	0	0	-22.8974336541644874	-43.1312620639801025	20	1
```

Figura 16: Exemplo do arquivo "*.txt*" de missão.

Criação das missões

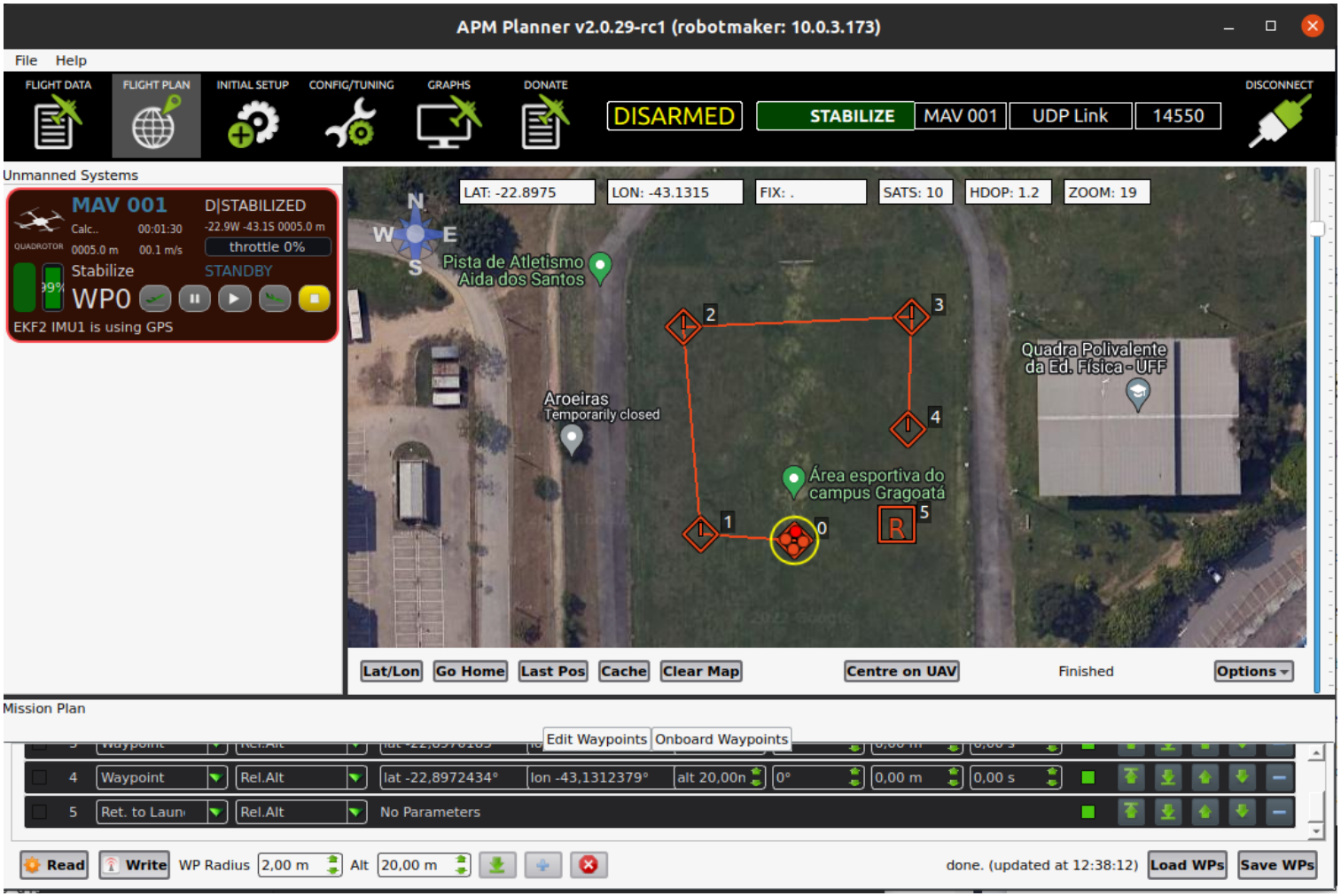


Figura 17: Tela de criação do arquivo de missão.

Sumário

Testes de criação realizados

- Realizaram-se os seguintes testes na *Cloud Control Station* (CCS):
    1. Criação de um usuário.

Testes de criação realizados

- Realizaram-se os seguintes testes na *Cloud Control Station* (CCS):

  1. Criação de um usuário.
  2. Criação de um *drone*.
     Atribuição do *drone* criado a um usuário.

Testes de criação realizados

- Realizaram-se os seguintes testes na *Cloud Control Station* (CCS):

  1. Criação de um usuário.
  2. Criação de um *drone*.
     Atribuição do *drone* criado a um usuário.
  3. Criação de uma missão.
     Atribuição da missão criada a um *drone*.

Testes da página *WEB*

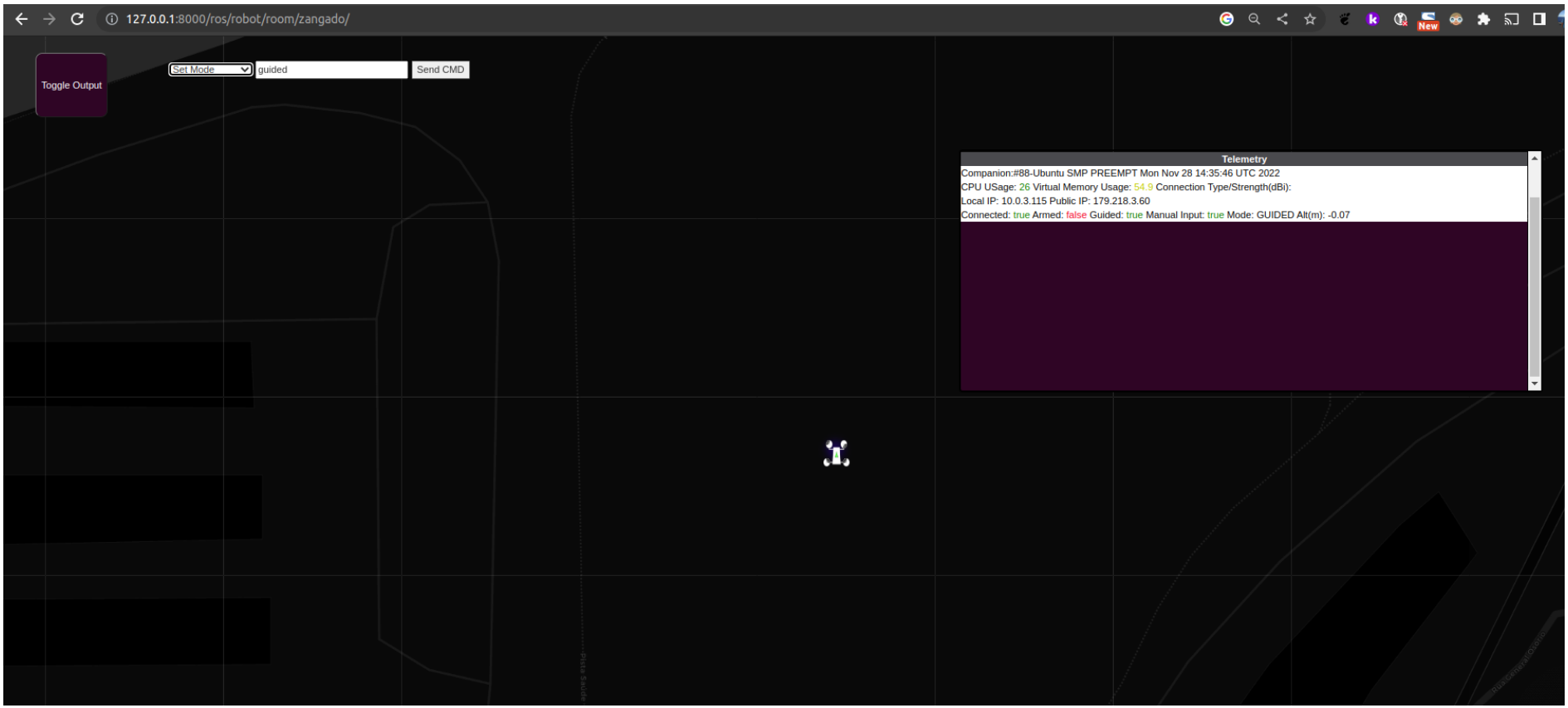


Figura 18: Tela de controle e navegação do *drone*.

Sumário

Sugestões para trabalhos futuros

1. Testes com um drone hexacóptero.

2. Melhorias na interface de controle:
   - Adição de novos funcionalidades.
   - Coleta e apresentação de mais dados do *drone*.

3. Integração de um modem de *Internet* móvel:
   - Controle e coleta de dados a qualquer distância, desde que haja sinal de *Internet* móvel.
4. Integração de uma câmera:
   - Visão, em tempo real, do ambiente sobrevoado pelo *drone*.
   - Adição de inteligência artificial, capaz de tratar e processar imagens capturadas.

Obrigado!

Perguntas?