

UNIVERSIDADE FEDERAL FLUMINENSE  
ESCOLA DE ENGENHARIA  
CURSO DE GRADUAÇÃO EM ENGENHARIA DE  
TELECOMUNICAÇÕES

Lúcio Folly S. Zebendo  
e  
João Luiz de Amorim Pereira Neto

Aplicações de *Drones* em Redes de Computadores:  
Utilização da plataforma *Raspberry Pi* como  
computador de bordo.

Niterói – RJ  
2022

Lúcio Folly S. Zebendo  
e  
João Luiz de Amorim P. Neto

Aplicações de *Drones* em Redes de Computadores: Utilização da plataforma *Raspberry Pi* como computador de bordo.

Trabalho de Conclusão de Curso apresentado ao  
Curso de Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense,  
como requisito parcial para obtenção do Grau de  
Engenheiro de Telecomunicações.

Orientador: Prof. Dr. Alexandre Santos de la Vega  
Coorientador - Prof. Dr. Lauro Eduardo Kozovits

Niterói – RJ  
2022

A figura referente ao arquivo

*FichaCatalografica.jpg*

fornecido pela Biblioteca

deverá aparecer aqui.

**ATENÇÃO:** Na versão impressa, essa página deverá  
ficar no verso da página anterior.

Lúcio Folly S. Zebendo  
e  
João Luiz de Amorim P. Neto

Aplicações de *Drones* em Redes de Computadores: Utilização da plataforma *Raspberry Pi* como computador de bordo.

Trabalho de Conclusão de Curso apresentado ao  
Curso de Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense,  
como requisito parcial para obtenção do Grau de  
Engenheiro de Telecomunicações.

Aprovada em DIA de MÊS de ANO.

#### BANCA EXAMINADORA

---

Prof. Dr. Alexandre S. de la Vega - Orientador  
Universidade Federal Fluminense - UFF

---

Prof. Dr. Lauro Eduardo Kozovits - Co-Orientador  
Universidade Federal Fluminense - UFF

---

Prof. Alexandre S. de la Vega  
INSTITUIÇÃO

---

Prof. Lauro Eduardo Kozovits  
INSTITUIÇÃO

Niterói – RJ  
2022

# Resumo

Este trabalho é o fruto de um esforço conjunto entre vários discentes e docentes da UFF e iniciativa privada, todos com mútuo interesse por VANTs e suas aplicações. O projeto inicial era ideia da Equipe UFFO [Equipe UFFO, ] - construir um *drone* para vigilância dos campi da UFF. O hardware a ser utilizado era um quadricóptero equipado com um sistema FPV de transmissão analógica. Posteriormente, houve-se o interesse de equipar um computador de bordo no *drone* em questão, para fazer aplicações no campo da visão computacional com processamento em tempo de voo. Para isso foi utilizado como referência o material da comunidade [ArduPilot Docs, 2021] sobre computadores de bordo [Companion Computers, 2021].

Para a escolha do computador de bordo foram levados mais em conta fatores de custo e popularidade, com isso a plataforma *Raspberry Pi* [Foundation, 2020] foi a selecionada para esse fim.

A partir da arquitetura proposta, várias aplicações além da visão computacional são possíveis. Como o computador de bordo em questão possui um sistema operacional com propósito geral baseado em *debian* [Software in the Public Interest, Inc, 2021], além de outras capacidades de Hardware como portas *GPIO* para acoplamento de sensores e atuadores e outros barramentos para conexão de periféricos, as possibilidades para esse hardware são muitas.

Com isso, a aplicação proposta nesse trabalho será a conexão desse computador de bordo à uma rede IP. Várias capacidades da plataforma *Raspberry Pi* serão exploradas nos experimentos que sucederão, além disso, serão desenvolvidas aplicações para testar o conceito de *Drones* em Redes de computadores.

Palavras-chave: Raspberry Pi. Drones. VANT. ROS. Equipe UFFO. Robótica.  
Ardpilot. Pixhawk

# Abstract

This work is a result of a collective effort between some students and teachers from UFF and the private sector, all of them having the mutual interest in UAVs and their applications. The first idea of this project was made by UFFO Team [Equipe UFFO, ] - to build a drone for surveillance of all UFF campuses. The required hardware would have been a quadcopter equipped with an analog transmission FPV system. Posteriorly, there was an interest to equip to the same drone an on-board computer, so it could provide some application in computer vision by doing real-time processing, The reference was made by studying the community files [Ardupilot Docs, 2021] about on-board computers [Companion Computers, 2021].

The on-board computer was chosen taking into account the cost and popularity, thus, the Raspberry Pi [Foundation, 2020] was selected for this purpose.

Based on the proposed architecture, countless features may be implemented by using its hardware. The on-board computer has an operating system based on debian [Software in the Public Interest, 2021] moreover it has numerous GPIO ports that enables the communication to sensors or actuators and also other buses to connect peripherals devices. Hence, there are a plenty possibilities of addons and functionalities.

Therefore, the proposed application of this thesis will be the connection between the on-board computer to an IP network. Many possibilities of Raspberry Pi will be studied and explored in future experiments, besides that, there will be developed some applications to test the concept of drones on a computer network.

**Keywords:** Raspberry Pi. Drones. UAV. ROS. UFFO Team. Robotics. Ardupilot.  
Pixhawk

Espaço reservado para a dedicatória.

# Agradecimentos

Agradecemos ao Prof. Alexandre pela orientação durante a nossa estadia no grupo do PET-Tele. Através, da nossa participação no grupo foi possível termos um primeiro contato com as plataformas de desenvolvimento Arduino e *Raspberry Pi* [Foundation, 2020], além de outras atividades que contribuíram para nosso desenvolvimento acadêmico e profissional. Agradecemos ao Prof. Raul pelo acolhimento na *Equipe UFFO* e pelo incentivo ao estudo e desenvolvimento de VANTs. Agradecemos ao Prof. Lauro Eduardo pelo direcionamento durante a elaboração desse projeto de TCC. Agradecemos à Equipe UFFO e especialmente ao Raphael Miranda, pelo apoio no desenvolvimento do drone quadrcóptero utilizado nos experimentos desse trabalho. Agradecemos à Empresa 6DDrones e especialmente ao Fabio, pelo apoio com infraestrutura e equipamentos para o desenvolvimento desse projeto. Agradecemos à comunidade Arducopter pelo grande esforço colaborativo em desenvolver varias das ferramentas utilizadas nesse projeto. Agradeço à Larissa, minha namorada, por todo apoio, carinho e incentivo durante a minha graduação. Agradecemos aos nossos familiares pela cobrança e incentivo nos estudos.

# Lista de Figuras

2.1	Classificação de VANTs por tipo de pouso, aerodinâmica e quantidade de rotores. . . . .	6
2.2	Veículos suportados pelo ArduPilot. . . . .	9
2.3	Raspberry Pi Revision 2.0. . . . .	14
4.1	Conexões de hardware para Omnibus F4 Pro V3. . . . .	20
4.2	Bancada de trabalho com o drone quadricóptero. . . . .	21
4.3	Pixhawk 2 Cube Hero. . . . .	22
4.4	Conexões de hardware para a Pixhawk 2 Cube Hero. . . . .	22
4.5	Bancada de trabalho com o drone quadricóptero. . . . .	23
4.6	Demonstração do Software In The Loop. . . . .	24
4.7	Arquitetura do Software In The Loop . . . . .	25
4.8	sim_vehicle.py . . . . .	25
4.9	startstl_nogazebo.sh . . . . .	25
4.10	startstl.sh com Gazebo . . . . .	26
4.11	Software In The Loop com Gazebo . . . . .	26
4.12	Nó do Gazebo - ROS . . . . .	26
4.13	apm.launch do Pacote Mavros . . . . .	27
4.14	Execução do Mavros . . . . .	28
4.15	Tópicos do Mavros . . . . .	28
4.16	Tópicos do Mavros . . . . .	29

# **Lista de Tabelas**

2.1	Especificações técnicas do modelo Raspberry Pi Revision 2 - Element14.	13
2.2	Descrição dos pinos do Raspberry Pi Rev 2.0 - Element 14.	14

# Sumário

<b>Resumo</b>	<b>iv</b>
<b>Abstract</b>	<b>v</b>
<b>Agradecimentos</b>	<b>vii</b>
<b>Lista de Figuras</b>	<b>viii</b>
<b>Lista de Tabelas</b>	<b>ix</b>
0.1 Siglas . . . . .	1
<b>1 Introdução</b>	<b>2</b>
1.1 Motivações . . . . .	3
1.2 Objetivo . . . . .	3
1.3 Resultados esperados . . . . .	4
1.4 Trabalhos correlacionados . . . . .	4
1.5 Organização do documento . . . . .	4
<b>2 Introdução teórica</b>	<b>6</b>
2.1 Definições iniciais . . . . .	6
2.1.1 Drones e VANTs . . . . .	6
2.1.2 Classificação de VANTs . . . . .	6
2.1.3 Componentes dos VANTs de Asa Rotativa . . . . .	7
2.1.4 ArduPilot . . . . .	8
2.1.5 Ardupilot . . . . .	10
2.1.6 UART . . . . .	11
2.1.7 Telemetria . . . . .	11
2.1.8 Protocolo Mavlink . . . . .	11
2.1.9 Computadores de Bordo . . . . .	12
2.2 Série de computadores Raspberry Pi . . . . .	13
2.2.1 ROS - Robot Operative System . . . . .	15

2.2.2	Django . . . . .	15
2.3	Arquitetura do sistema proposto . . . . .	17
<b>3</b>	<b>Metodologia</b>	<b>18</b>
3.1	Pesquisa . . . . .	18
3.2	Objetivos do método aplicado . . . . .	18
3.3	Abordagem . . . . .	18
3.4	Descrição do problema . . . . .	18
<b>4</b>	<b>Detalhamento da solução</b>	<b>20</b>
4.1	Primeiros testes com o drone quadricóptero . . . . .	20
4.1.1	Diagrama de conexão dos componentes de hardware . . . . .	20
4.2	Montagem do drone hexacóptero . . . . .	22
4.2.1	Diagrama de conexão dos componentes de hardware . . . . .	22
4.3	Ambiente de simulação . . . . .	24
4.4	Ambiente de Simulação com o Mavros . . . . .	27
4.5	Arquitetura da aplicação . . . . .	29
4.6	Arquitetura da aplicação de rede . . . . .	29
4.7	Outras arquiteturas possíveis para o sistema proposto . . . . .	30
<b>5</b>	<b>Resultados</b>	<b>31</b>
<b>6</b>	<b>Conclusão</b>	<b>32</b>
<b>7</b>	<b>Sugestões para trabalhos futuros</b>	<b>34</b>
	<b>Referências</b>	<b>36</b>
<b>A</b>	<b>Apêndice 1</b>	<b>37</b>
A.1	Intalando o ArduPilot e MavProxy . . . . .	37
A.1.1	Clonando o repositório <i>git</i> para sua máquina . . . . .	37
A.1.2	Instalando as dependências e recompilando o perfil . . . . .	37
A.1.3	Mudando para a <i>branch</i> do ArduCopter . . . . .	37
A.1.4	Rodando SITL ( <i>Software In The Loop</i> ) . . . . .	37
A.2	Instalando o Gazebo e o <i>plugin</i> do ArduPilot . . . . .	38
A.2.1	Atualizando a lista de fontes para <i>download</i> . . . . .	38
A.2.2	Instalando o <i>plugin</i> do Gazebo para comunicação com o ArduPilot .	38
A.2.3	Executando a simulação . . . . .	38
A.3	Instalando o <i>ROS</i> e o configurando o <i>Catkin</i> . . . . .	39
A.3.1	Atualizando a lista de fontes para <i>download</i> . . . . .	39

A.3.2	Instalando o <i>ROS</i> . . . . .	39
A.3.3	Configurando o ambiente . . . . .	39
A.3.4	Instalando as dependências para os pacotes dos <i>ROS</i> . . . . .	39
A.3.5	Configurando o <i>Catkin</i> . . . . .	40
A.3.6	Instalando as dependências para o <i>Catkin</i> . . . . .	40
A.3.7	Instalando as <i>MAVROS</i> e <i>MAVLink</i> . . . . .	40
A.3.8	Atualizando o arquivo <i>.bachrc</i> . . . . .	40
A.4	Instalando as dependências geográficas e clonando o repositório de simulação do <i>Intelligent Quads</i> . . . . .	41
A.4.1	Instalando as dependências geográficas . . . . .	41
A.4.2	Clonando pacote ROS de simulação do <i>Intelligent Quads</i> . . . . .	41
A.5	Instalando o <i>QGround Control</i> . . . . .	41
A.5.1	Alterando permissões e instalando o <i>QGround Control</i> . . . . .	41

## 0.1 Siglas

ROS - *Robot Operating System*

VANT - Veículo aéreo não tripulado

MAVLink - *Micro Air Vehicle Communication Protocol*

UART - *Universal asynchronous receiver/transmitter*

RXD - *Receive Data*

TXD - *Transmit Data*

# Capítulo 1

## Introdução

«««< HEAD Atrelado à popularização da tecnologia, o barateamento dos *drones* tornaram-los capazes de serem usados para além do âmbito militar, ganhando espaço nas indústrias cinematográficas, esportivas e do entretenimento, fora para o uso pessoal.

Somado à isso, com o barateamento e miniaturização dos componentes eletrônicos nas últimas décadas, surgiram diversas plataformas eletrônicas de prototipagem, como o Arduíno e também mini computadores como o Raspberry Pi, o que popularizou o estudo das mesmas no âmbito dos drones.

O uso de mini computadores como o Raspberry Pi já é difundido em aplicações de IOT e robótica e recentemente a comunidade do Ardupilot também explorou e documentou a utilização da plataforma como um computador de bordo, abrindo um leque de possibilidades para as atividades que um VANT (Veículo Aéreo Não Tripulado) pode executar.

Um computador de bordo permite a utilização de um sistema operacional completo, e por consequência oferece vários benefícios, como por exemplo capacidades de conexão à rede e a utilização de diversas tecnologias que facilitam o desenvolvimento de drones inteligentes. ===== Atrelado à popularização da tecnologia, o barateamento dos *drones* tornaram-los capazes de serem usados para além do âmbito militar, ganhando espaço nas indústrias cinematográficas, esportivas e do entretenimento, fora para o uso pessoal.

Somado à isso, com o barateamento e miniaturização dos componentes eletrônicos nas últimas décadas, surgiram diversas plataformas eletrônicas de prototipagem, como o Arduíno e também mini computadores como o Raspberry Pi, o que popularizou o estudo das mesmas no âmbito dos drones.

O uso de mini computadores como o Raspberry Pi já é difundido em aplicações de IOT e robótica e recentemente a comunidade do Ardupilot também explorou e documentou a utilização da plataforma como um computador de bordo, abrindo um leque

de possibilidades para as atividades que um VANT (Veículo Aéreo Não Tripulado) pode executar.

Um computador de bordo permite a utilização de um sistema operacional completo, e por consequência oferece vários benefícios, como por exemplo capacidades de conexão à rede e a utilização de diversas tecnologias que facilitam o desenvolvimento de drones inteligentes.

»»»> 8f3b1bbf034ec2d3e90e71f2004f3703b5ccfa48

Dessa forma, partindo da tecnologia para a utilização de computadores de bordo em drones é possível explorar as capacidades desse sistema conectado à rede. Além disso, é inevitável a criação de sistemas capazes de realizar o gerenciamento e controle dessas aeronaves através da própria internet, o que será abordado nesse documento.

## 1.1 Motivações

O sentimento de interesse e empolgação pela tecnologia dos drones dos autores desse texto foram as forças primordiais para a concepção do trabalho que se segue. Tal interesse nasceu dentro da Universidade Federal Fluminense através da equipe UFFO [Equipe UFFO, ], formada por discentes e docentes da universidade.

Com a experiência adquirida na confecção de um drone quadrocoptero simples, descobriu-se a possibilidade de acoplar a este um Raspberry Pi [Foundation, 2020] como computador de bordo da aeronave, expandindo suas capacidades. A partir desta ideia, foram realizados estudos para buscar os métodos e tecnologias mais recentes utilizados nessa configuração de drone. Felizmente, muitas dessas tecnologias utilizadas já estavam disponíveis na documentação Open Source do ArduPilot [ArduPilot Docs, 2021] para o equipamento utilizado.

## 1.2 Objetivo

««< HEAD O primeiro objetivo deste trabalho será o estudo e planejamento da arquitetura do sistema proposto, desde o hardware a ser utilizado, às ferramentas de software e a arquitetura da aplicação que será desenvolvida.

O segundo objetivo é o desenvolvimento de uma aplicação utilizando as ferramentas e a arquitetura proposta no objetivo anterior. Além disso, o funcionamento do sistema será validado através de um ambiente de simulação. A validação será feita a partir da análise dos resultados.

Por último, com a apuração do sistema, visa-se a construção e configuração de um drone, com as mesmas especificações do usado no segundo objetivo, isso inclui todo o sis-

tema embarcado proposto. Finalmente, serão analisados os resultado obtidos. ====== O primeiro objetivo deste trabalho será o estudo e planejamento da arquitetura do sistema proposto, desde o hardware a ser utilizado, as ferramentas de software e a arquitetura da aplicação que será desenvolvida.

O segundo obejtivo é o desenvolvimento de uma aplicação utilizando as ferramentas e a arquitetura proposta no objetivo anterior. Além disso, o funcionamento do sistema será parcialmente validado através de um ambiente de simulação.

Por último, visa-se a construção e configuração de um drone com o sistema embarcado proposto e serão analisados os resultado obtidos. »»»> 8f3b1bbf034ec2d3e90e71f2004f3703b5ccfa

### **1.3 Resultados esperados**

- Criação de um ambiente de simulação para a validação dos experimentos propostos.
- Um sistema de controle e navegação para o drone via rede IP.
- Um drone capaz de realizar comunicação de dados numa rede IP.

### **1.4 Trabalhos correlacionados**

- Artigos de Drones LTE [Lassi Sundqvist, 2015]

### **1.5 Organização do documento**

#### 1. Introdução teórica

##### (a) Definições

Apesar da crescente difusão dos drones na sociedade geral, muitos termos ainda não são conhecidos e devem ser explicados a fim de se compreender a teoria em volta do nosso projeto. Resumidamente, discutiremos os tipos de drones mais comumente comercializados, qual a utilidade de cada parte essencial para um drone operar, os *softwares* embarcados, o computador de bordo e alguns complementos para aperfeiçoar o projeto.

##### (b) Arquitetura Proposta

Desejamos que, após agregar todas as tecnologias introduzidas, elaborar um sistema composto por uma controladora de *firmware ardupilot* e um computador de bordo capaz de se conectar com uma rede de computadores. O *software*

embarcado será desenvolvido utilizando *ROS* e uma *API* será programada em *django* para centralizar a conexão do drone via rede.

## 2. Metodologia

### 3. Detalhamento da solução

A nível de *hardware*, para realizar testes iniciais da solução, escolheu-se um drone quadricóptero, a controladora Omnibus f4 pro, um Raspberry Pi, além do *software MavProxy*. Por outro lado, a parte de *software* do projeto, gira em torno da conexão entre o pacote *MAVros*, baseado no protocolo *MAVLink*, e o *django framework*.

Em resumo, a arquitetura se baseia num modelo cliente-servidor, onde o Raspberry é o cliente, e o *django* o servidor, ambos conectados numa mesma rede via VPN. Idealmente, outras possíveis soluções seria a conexão de um modem LTE (4G) ou NR (5G), o qual permitiria a conexão sem limitações de distância, bastaria estar na área de cobertura do sinal de rede móvel.

## 4. Resultados

Estabelecemos com sucesso a conexão entre cliente e servidor. O cliente, composto pelo drone, controladora e *Raspberry*, coleta, processa e envia ao servidor os dados de telemetria. Já o servidor, formado principalmente pelo *django*, recebe esses dados e os imprime numa página *web*.

# Capítulo 2

## Introdução teórica

### 2.1 Definições iniciais

#### 2.1.1 Drones e VANTs

Drones são sistemas aéreos automatizados que incluem tanto os VANTs - Veículos Aéreos Não Tripulados, capazes de voar por milhares de quilômetros, quanto drones pequenos que voam em locais confinados. São veículos aéreos não tripulados, automatizados ou controlados remotamente, podem carregar diversos tipos de carga e fazer vários tipos de missões.

Devido a avanços nas tecnologias envolvidas na fabricação, nos sistemas de navegação, nos sensores e sistemas de armazenamento de energia, uma grande variedade de tipos de drones surgiram nas últimas décadas.

#### 2.1.2 Classificação de VANTs

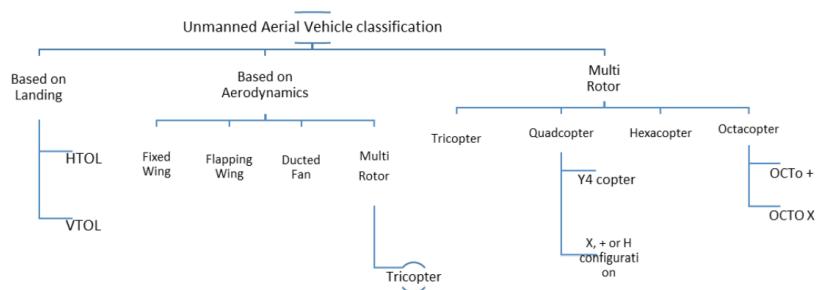


Figura 2.1: Classificação de VANTs por tipo de pouso, aerodinâmica e quantidade de rotores.

Como mostrado no organograma da figura 2.1, as categorizações mais comuns para os VANTs são obtidas pelos atributos "tipo de pouso", "aerodinâmica" e "quantidade de rotores".

tidade de rotores". <<<< HEAD

O tipo de aerodinâmica responsável pela sustentação das aeronaves no ar é bastante diversificado. Uma categoria bem comum de drones são os VANTs de asa fixa, que semelhantemente aos aviões, tiram proveito da geometria das asas e da velocidade gerada por um ou mais motores para obterem sustentação.

A classificação de aterrissagem, com a nomenclatura VTOL e HTOL diz respeito apenas aos VANTs de asa fixa visto que apenas esses tipos de VANTs tiram proveito tanto da aterrissagem vertical quanto da autonomia proveniente do vôo horizontal com asas fixas.

=====

O tipo de aerodinâmica responsável pela sustentação das aeronaves no ar é bastante diversificado. Uma categoria bem comum de drones são os VANTs de asa fixa, que semelhantemente aos aviões, tiram proveito da geometria das asas e da velocidade gerada por um ou mais motores para obterem sustentação.

A classificação de aterrissagem, com a nomenclatura VTOL e HTOL diz respeito apenas aos VANTs de asa fixa visto que apenas esses tipos de VANTs tiram proveito tanto da aterrissagem vertical quanto da autonomia proveniente do vôo horizontal com asas fixas.

»»> 8f3b1bbf034ec2d3e90e71f2004f3703b5ccfa48 No presente documento focaremos mais acutuadamente nos drones de asa rotativa, visto que esse tipo de drone será utilizado nos experimentos que sucederão.

### **2.1.3 Componentes dos VANTs de Asa Rotativa**

Os componentes dos drones de asa rotativa variam conforme o tipo de missão. Alguns dos componentes principais podem ser listados abaixo:

1. Frame
2. Motores e hélices
3. Escs
4. Controladora de voo
5. Bateria e Placa distribuidora de energia

#### **Frame**

O Frame de um drone é a base estrutural na qual todos os outros componentes são fixados, ele geralmente é feito de um material leve e resistente como fibra de carbono e

outros polímeros plásticos.

### **Bateria e Placa distribuidora de energia**

A bateria é a fonte de energia do drone, geralmente são utilizadas baterias Li-Po (Lítio-Polímero) pela sua boa relação massa/desempenho enérgético, essas baterias possuem uma nomenclatura específica com base na quantidade de células <explicar melhor as baterias>.

### **Motores e hélices**

Os motores do drone são os responsáveis por converter a energia elétrica proveniente das baterias em energia cinética. Geralmente os motores utilizados são motores de corrente contínua não escovados (*brushless DC*) <colocar as características físicas desse tipo de motor>. A partir da corrente elétrica fornecida ao motor ele movimenta a hélice, que devido à sua aerodinâmica gera uma força resultante para cima no drone, a qual é denominada empuxo.

### **Escs**

««< HEAD =====

Os Escs, ou Electronic Speed Controllers são os componentes eletrônicos responsáveis por controlar a velocidade dos motores. Eles são comandados pela controladora de voo, geralmente a partir de sinais PWM <Falar sobre PWM>. Assim como o PWM é utilizado para comandar os ESCs, os ESCs utilizam essa modulação para chavear o circuito entre os motores e a bateria controlando assim a velocidade dos motores.

### **Controladora de Voo**

A controladora de voo é um microcontrolador com software embarcado dedicado às funções básicas da aeronave. Ela possui sensores iniciais embutidos para comandar a dinâmica de voo da aeronave, além de possuir interface eletrônica com outros componentes como módulos GPS, módulos receptores de rádio, módulos de telemetria entre outros. Sem a controladora de voo, seria uma tarefa humanamente impossível controlar o voo da aeronave, visto que a mesma depende de um feedback contínuo e instantâneo dos parâmetros iniciais.

#### **2.1.4 ArduPilot**

O ArduPilot é um projeto de código aberto que possui um conjunto de software destinado ao piloto automático de veículos não tripulados, os quais incluem:



Figura 2.2: Veículos suportados pelo ArduPilot.

- 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.

Além de suportar o firmware para todos esses tipos de veículos, a comunidade ArduPilot também se preocupa em desenvolver outras plataformas que auxiliam o desenvolvimento e a utilização do próprio firmware, como por exemplo:

- Antenna Tracker - firmware para mirar uma antena automaticamente para o veículo.
- Mission Planner - interface de estação de controle em solo escrita em C# para Windows.
- APM Planner 2.0 - interface de estação de controle específica para APM escrita em C++ usando os módulos Qt, pode ser executada em Linux, Windows e Mac.
- MAVProxy - interface de estação de controle em linha de comando ou via script.
- DroneKit - APM SDK para aplicações executadas de forma embarcada, em mobile, e/ou na nuvem.
- MinimOSD - visualização de informações de voo em tempo real no visor da interface de FPV.
- Tower - interface de estação de controle para celular.

- QGroundControl é uma interface de estação de controle alternativa escrita em C++ usando os modulos Qt.
- PX4 - firmware pixhawk para a controladora pixhawk.
- MAVLink - o protocolo para comunicação entre estação de controle, controladora de voo e alguns componentes incluindo OSD.
- UAVCAN - protocolo leve e confiável para aplicações de comunicação aeroespacial e robótica, via CAN bus.

<LISTAR CONTROLADORAS DE VOO COMPATÍVEIS COM O FIRMWARE>  
 »»»> 8f3b1bbf034ec2d3e90e71f2004f3703b5ccfa48

Os Escs, ou Electronic Speed Controllers são os componentes eletrônicos responsáveis por controlar a velocidade dos motores. Eles são comandados pela controladora de voo, geralmente a partir de sinais PWM <Falar sobre PWM>. Assim como o PWM é utilizado para comandar os ESCs, os ESCs utilizam essa modulação para chavear o circuito entre os motores e a bateria controlando assim a velocidade dos motores.

### **Controladora de Voo**

A controladora de voo é um microcontrolador com software embarcado dedicado às funções básicas da aeronave. Ela possui sensores iniciais embutidos para comandar a dinâmica de voo da aeronave, além de possuir interface eletrônica com outros componentes como módulos gps, módulos receptores de rádio, módulos de telemetria entre outros. Sem a controladora de voo, seria uma tarefa humanamente impossível controlar o voo da aeronave, visto que a mesma depende de um feedback contínuo e instantâneo dos parâmetros iniciais.

#### **2.1.5 Ardupilot**

ArduPilot enables the creation and use of trusted, autonomous, unmanned vehicle systems for the peaceful benefit of all. ArduPilot provides a comprehensive suite of tools suitable for almost any vehicle and application. As an open source project, it is constantly evolving based on rapid feedback from a large community of users. The Development Team works with the community and commercial partners to add functionality to ArduPilot that benefits everyone. Although ArduPilot does not manufacture any hardware, ArduPilot firmware works on a wide variety of different hardware to control unmanned vehicles of all types. Coupled with ground control software, unmanned vehicles running

ArduPilot can have advanced functionality including real-time communication with operators. ArduPilot has a huge online community dedicated to helping users with questions, problems, and solutions

### 2.1.6 UART

UART, que vem das palavras *universal asynchronous receive/transmit*, realiza o papel principal na comunicação serial, já que converte os dados entre série e paralelo. Há então uma conversão de paralelo-serial para os dados transmitidos, e uma outra serial-paralelo para os dados recebidos. Existe ainda um *buffer*, cujo objetivo é armazenar dados por tempo limitado em transmissões com alta taxa, sem que haja descarte de dados e consequente perda de informação. Para que não haja fluxo de transmissão caso ambas as partes estiverem prontas, em adição as características citadas, são incorporados circuitos reguladores de fluxo.

A divisão do protocolo é feita em três sub-módulos, gerador de taxa de transmissão, módulo de transmissão e módulo de recepção. O primeiro, se responsabiliza por gerar um sinal de relógio localmente, o qual deve ser muito maior que a taxa de transmissão de dados entre transmissor-receptor. O módulo receptor, por sua vez, recebe os sinais RXD e os converte para dados paralelos. Por fim, o módulo transmissor converte os bytes em bits seriais, com base no padrão de quadro utilizado, e os envia através de sinais TXD. [Laddha and Thakare, 2013]

### 2.1.7 Telemetria

A telemetria é uma funcionalidade indispensável nos drones mais modernos, ela permite receber informações dos sensores do drone em tempo real como altitude, velocidade, gps, temperatura, informações da quantidade de carga na bateria dentre outras. Essa transmissão de dados geralmente é feita por um módulo dedicado a isso ou então em módulos receptores de rádio que já possuem essa funcionalidade integrada.

### 2.1.8 Protocolo Mavlink

MAVLink é um protocolo de comunicação com drones e entre os componentes acoplados ao drone. Essa transmissão consiste no envio de um fluxo de dados, os quais são chamados de tópicos, e escritos em arquivos XML. Dentre as principais vantagens de se utilizar o MAVLink estão a sua eficiência e versatilidade. Em números, a segunda versão do MAVLink tem apenas 14 *bytes* de *overhead* no pacote de comunicação. Além disso, provê métodos de detectar perda de pacotes e dados corrompidos, e garante a autenticação dos pacotes. Outro ponto positivo é o suporte a diversas linguagens, já que o protocolo

tem suporte a diferentes tipos de sistemas operacionais e microcontroladores. Por fim, apesar de menos importante para o nosso projeto, vale citar que o protocolo suporta até 255 usuários simultâneos em um mesma rede. [MAVLink, ]

### 2.1.9 Computadores de Bordo

Computadores de bordo (ou *Companion Computers*, como são chamados na documentação do ArduPilot) podem ser utilizados para interfacear com o firmware ArduPilot numa controladora de voo através do protocolo MAVLink. Dessa forma, o computador de bordo consegue computar todos os dados MAVLink produzidos pela controladora e com estes pode fazer decisões inteligentes durante o voo.

Geralmente, no quesito de hardware para esses computadores são escolhidos mini computadores de arquitetura baseada em ARM. Os mini computadores suportados pela comunidade ArduPilot são listados abaixo:

- Arduino family
- LYCHEE (Cube Carrier Board for Raspberry Pi Compute Module)
- NVidia TX1
- NVidia TX2
- ODroid
- Raspberry Pi

Já no quesito de software, existem ferramentas, programas e sistemas operacionais específicos para computadores de bordo, tais programas conseguem computar os dados MAVLink vindos da controladora. Alguns dos softwares suportados pela comunidade ArduPilot são listados abaixo:

- APSync
- DroneKit
- FlytOS
- Maverick
- ROS
- Rpanion-server

## 2.2 Série de computadores Raspberry Pi

O Raspberry Pi é uma série de computadores de placa única e tamanho reduzido, que recebem a denominação SoC (*System on Chip*) [Members, 2019b] à qual são conectados os seguintes dispositivos: monitor, teclado, e mouse.

Desenvolvido no Reino Unido pela Raspberry Pi Foundation tendo, como principais objetivos, contribuir para inclusão digital, promoção de ensino básico em ciência da computação e empoderamento social. Uma alternativa de ensino com baixo custo para escolas e estudantes [Members, 2019a]. A Tabela 2.1 apresenta as especificações técnicas do modelo Raspberry Pi Revision 2 - Element 14, utilizado pelo grupo PET-Tele.

Chip	Broadcom BCM2835 SoC Full HD Processador de Aplicações Multimídia
CPU	700 Mhz ARM1176JZ-F Processador de Baixa Potência de aplicações
GPU	Dois Núcleos, VideoCore IV, Co-Processador de Multimídia
Memória	512MB SDRAM
Ethernet	Onboard 10/100 Ethernet com conector RJ-45
USB 2.0	Dois Conectores USB 2.0
Saída de Vídeo	HDMI e Composição RCA (PAL e NTSC)
Saída de Áudio	Conecotor 3.5 mm ou HDMI
Armazenamento	SD, MMC, SDIO Card Slot
Dimensões	8.6 cm X 5.4 cm X 1.7 cm

Tabela 2.1: Especificações técnicas do modelo Raspberry Pi Revision 2 - Element14.

Na Figura 2.3 é mostrado uma fotografia do modelo. Dentre os 25 Pinos presentes neste Raspberry, 17 podem ser usados como entradas ou saídas de uso geral, 5 como terminais comuns (GND), 2 como fontes de tensão de valor +5V e 2 como fontes de tensão de valor +3.3V.

A Tabela 2.2 reúne uma descrição de todos os pinos.



Figura 2.3: Raspberry Pi Revision 2.0.

N	Descrição	N	Descrição
1	Saída de +3.3V	14	Terminal Comum GND
2	Saída de +5V	15	GPIO22
3	GPIO2   Pull-Up   I2C-SDA	16	GPIO23
4	Saída de +5V	17	3V3
5	GPIO3   Pull-Up   I2C-SCE	18	GPIO24
6	Terminal Comum GND	19	GPIO10   SPI   MOSI
7	GPIO4	20	Terminal Comum GND
8	GPIO14   UART   TXD	21	GPIO9   SPI   MISO
9	Terminal Comum GND	22	GPIO25
10	GPIO15   UART   RXD	23	GPIO11   SPI   CLK
11	GPIO17   UART-RTS	24	GPIO8   SPI   CE0
12	GPIO18   PWM	25	Terminal Comum GND
13	GPIO27	26	GPIO7   SPI   CE1

Tabela 2.2: Descrição dos pinos do Raspberry Pi Rev 2.0 - Element 14.

### 2.2.1 ROS - Robot Operative System

Robot Operating System, ROS ou ros, é um middleware, com foco em robótica, open-source. O ROS não é considerado um sistema operacional, mas um conjunto de software frameworks para desenvolvimentos de softwares para robôs. Oferece serviços a uma vasta gama de computadores, como abstração de hardware, controle de dispositivos de baixo nível, implementação de funcionalidades usadas comumente, troca de mensagens entre processos e gerenciamento de pacotes. A arquitetura de grafos pode ser usada para representar processos baseados em ROS, onde os processos são tidos como vértices, os quais podem receber, enviar, e multiplexar dados de sensores, controlar, planejar e trocar outras mensagens. Apesar da necessidade de baixa latência e interações rápidas, o ROS não é um *Real-Time Operating System* (RTOS). Entretanto, embora pouco explorado, é possível integrar códigos executados em tempo real. Essa falta de um suporte nativo a códigos de tempo real é uma preocupação real dos desenvolvedores, os quais planejam integrá-los ao ROS 2, projeto que visa ter proveito de bibliotecas e tecnologias mais novas.

As principais bibliotecas do ROS são voltadas para sistemas Unix, isso se deve ao fato da dependência com diversos *softwares open-source*. Para essas bibliotecas, o Ubuntu é listado como "*supported*", entretanto, para outras distros Linux, como o Fedora, macOS e Windows, todos são listados como "*experimental*". Ainda assim, existem exceções, como a biblioteca rosjava, a qual foi escrita para Android OS. Apesar disso, oferece uma integração nativa com o MATLAB, programa que pode ser usado no Linux, macOS ou Windows. Outro caso a parte é a biblioteca roslibjs, voltada para desenvolvimento *web* com JavaScript, que garante compatibilidade com qualquer navegador padrão.

#### Mavros

This package provides communication driver for various autopilots with MAVLink communication protocol. Additional it provides UDP MAVLink bridge for ground control stations (e.g. QGroundControl).

<listar os tópicos>

### 2.2.2 Django

Django é um *web framework* cuja intenção é tornar o desenvolvimento *web* mais veloz e fácil. Possui uma integração nativa entre banco de dados, *front-end* e *back-end*, apesar de que não seja necessária a utilização de um banco de dados. Primeiramente, podemos citar que todo o esquema do banco é feito através de códigos *python*, pela técnica de mapeamento objeto-relacional, como podemos ver no exemplo a seguir[Foundation, ].

```
from django.db import models
```

```

class Reporter(models.Model):
    full_name = models.CharField(max_length=70)

    def __str__(self):
        return self.full_name

class Article(models.Model):
    pub_date = models.DateField()
    headline = models.CharField(max_length=200)
    content = models.TextField()
    reporter = models.ForeignKey(Reporter, on_delete=models.CASCADE)

    def __str__(self):
        return self.headline

```

No exemplo acima, são instanciadas duas tabelas, *Reporter* e *Article*, e suas respectivas colunas. A fim de criá-las no banco deve-se executar os comandos:

```

$ python manage.py makemigrations
$ python manage.py migrate

```

A partir do momento que os modelos estão criados, podemos registrá-los no arquivo referente ao administrador, o que possibilita a criação, alteração e remoção dos objetos através da *interface web* criada pelo próprio Django. Para habilitar as operações citadas sobre a classe *Article*, por exemplo, basta configurar o arquivo *admin.py* tal como:

```

from django.contrib import admin

from . import models

admin.site.register(models.Article)

```

Outro ponto interessante é a possibilidade de configuração de *URLs*. Para isso, devemos criar um código de mapeamento de *URLs* dentro do arquivo *urls.py*, como exemplificado a seguir:

```

from django.urls import path

from . import views

```

```
urlpatterns = [
    path('articles/<int:year>/', views.year_archive),
    path('articles/<int:year>/<int:month>/', views.
        month_archive),
]
```

## 2.3 Arquitetura do sistema proposto

Com todas as tecnologias devidamente estudadas, deseja-se criar um sistema composto pelas mesmas e explorar as capacidades de tal sistema. Para isso, utilizaremos um drone com controladora de firmware ArduPilot e um computador de bordo capaz de se conectar com uma rede de computadores. O software embarcado será desenvolvido utilizando ROS e uma API será desenvolvida em Django para se comunicar com o drone via rede.

# **Capítulo 3**

## **Metodologia**

### **3.1 Pesquisa**

### **3.2 Objetivos do método aplicado**

A metodologia aplicada possui objetivo descritivo e exploratório. Após a realização de cada etapa do projeto serão feitos experimentos para comprovar o devido funcionamento do sistema e para este serão comentadas as possibilidades de aplicações.

### **3.3 Abordagem**

Adotar-se-á uma análise qualitativa dos resultados obtidos a partir da implementação do novo sistema, em observância aos seguintes questionamentos:

1. Com o sistema desenvolvido será possível receber dados de telemetria via rede?
2. Com o sistema desenvolvido será possível ao operador do sistema, controlar o drone remotamente?
3. O quanto seguro é o sistema desenvolvido? Quais riscos de segurança estão submetidos a tal sistema?
4. pergunta avaliativa 3
5. pergunta avaliativa 4

### **3.4 Descrição do problema**

De forma resumida, deseja-se, a partir de um drone montado com todos os seus componentes essenciais, acoplar um Raspberry Pi à controladora do mesmo para conec-

tar o conjunto à rede de computadores e com isso expandir suas capacidades. Algumas questões surgem quanto a esse projeto, como por exemplo: Qual arquitetura de software a ser utilizada, ou seja como o software a ser desenvolvido será organizado? Qual arquitetura de rede a ser utilizada, por exemplo, par-a-par ou cliente-servidor? No quesito de comunicação na internet, na qual muitas vezes não se tem o controle da rede, como seria feita essa comunicação?

# Capítulo 4

## Detalhamento da solução

### 4.1 Primeiros testes com o drone quadricóptero

De posse de um drone quadricóptero com a controladora de voo Omnibus f4 pro, foram estabelecidos os primeiros testes de conexão entre o Raspberry Pi e o firmware ArduPilot. O diagrama de montagem do equipamento foi o seguinte:

#### 4.1.1 Diagrama de conexão dos componentes de hardware

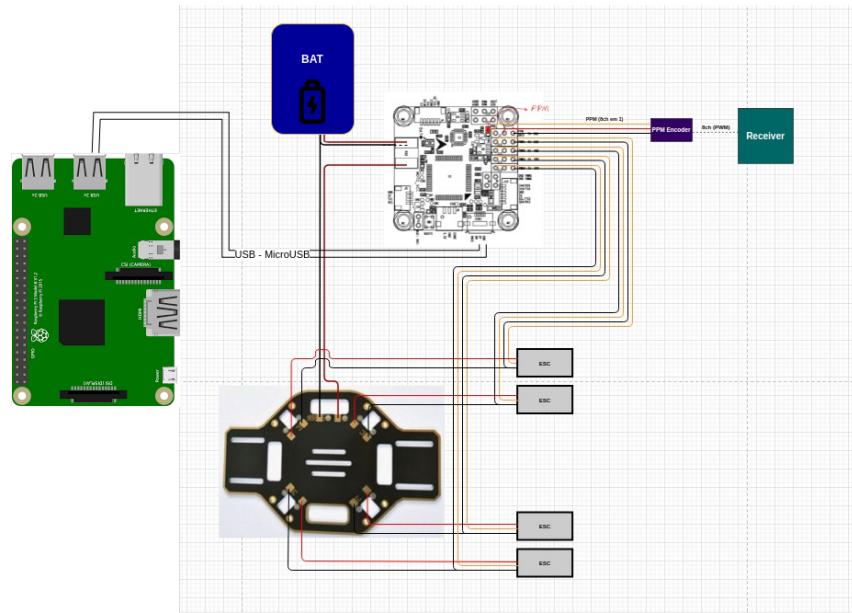


Figura 4.1: Conexões de hardware para Omnibus F4 Pro V3.

A motivação para a escolha dessa controladora foi o preço da mesma no mercado em relação às outras, valor que na época era em torno de 260 reais, enquanto que a Pixhawk custava em torno de 1000 reais. O resto dos componentes também foi escolhido com o intuito de deixar o projeto em baixo custo.

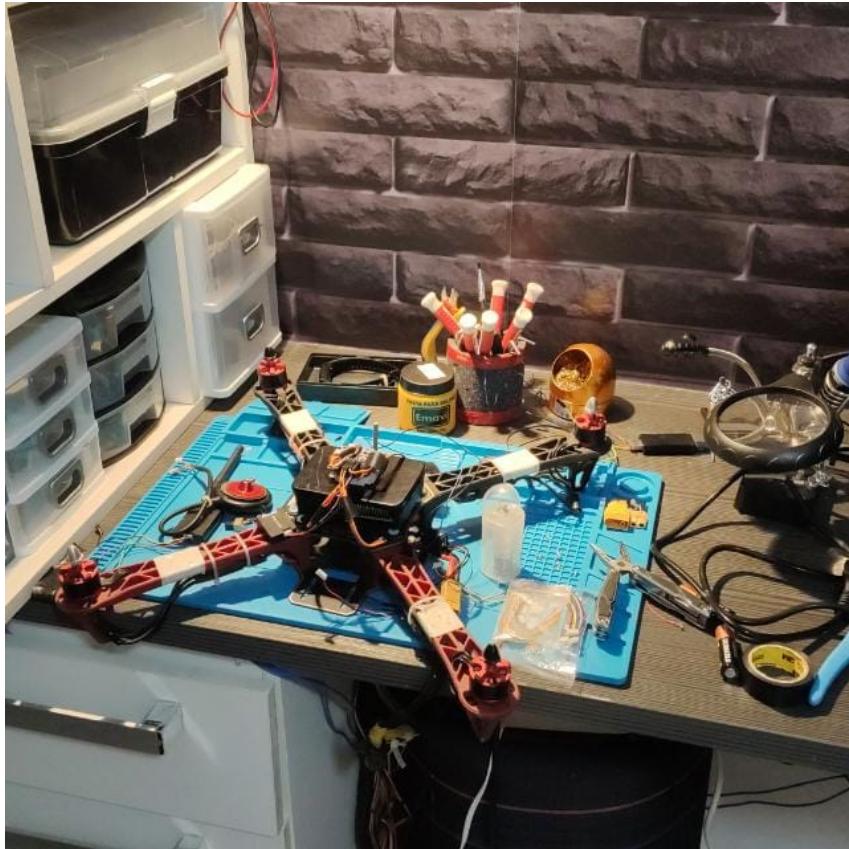


Figura 4.2: Bancada de trabalho com o drone quadricóptero.

Após a ligação dos componentes eletrônicos, o teste de conexão entre o Raspberry e a controladora foi realizado utilizando o software MavProxy no minicomputador e foi possível obter dados em tempo real da controladora através da conexão serial assim como também foi possível mandar comandos MAVLink para a controladora de voo através do terminal da aplicação.

Dessa forma, conseguiu-se comandar o drone a partir do computador de bordo e receber dados de telemetria do mesmo. Entretanto, não foi possível acoplar o módulo de GPS na controladora já que o módulo em questão não se mostrou compatível com a mesma.

As pesquisas e soluções de problemas oriundos da execução dessa etapa foram cruciais para adquirir maturidade na montagem, configuração e acoplamento dos componentes. Além disso, o conhecimento adquirido das ferramentas utilizadas para conectar o computador de bordo ao drone também foram essenciais. Entretanto, escolhemos mudar o hardware utilizado para uma configuração de drone mais convencional na comunidade ArduPilot, com a controladora Pixhawk Cube.

## 4.2 Montagem do drone hexacóptero

Após os problemas obtidos no drone quadcóptero com a controladora Omnibus, trocamos o hardware utilizado para um drone hexacóptero com a controladora Pixhawk Cube. Tal aeronave já estava operacional antes de acoplarmos o RaspberryPi, isto é, era possível realizar todas as funções básicas que a controladora por si só é capaz de fazer.



Figura 4.3: Pixhawk 2 Cube Hero.

Com isso, obtemos a seguinte conexão dos componentes:

### 4.2.1 Diagrama de conexão dos componentes de hardware

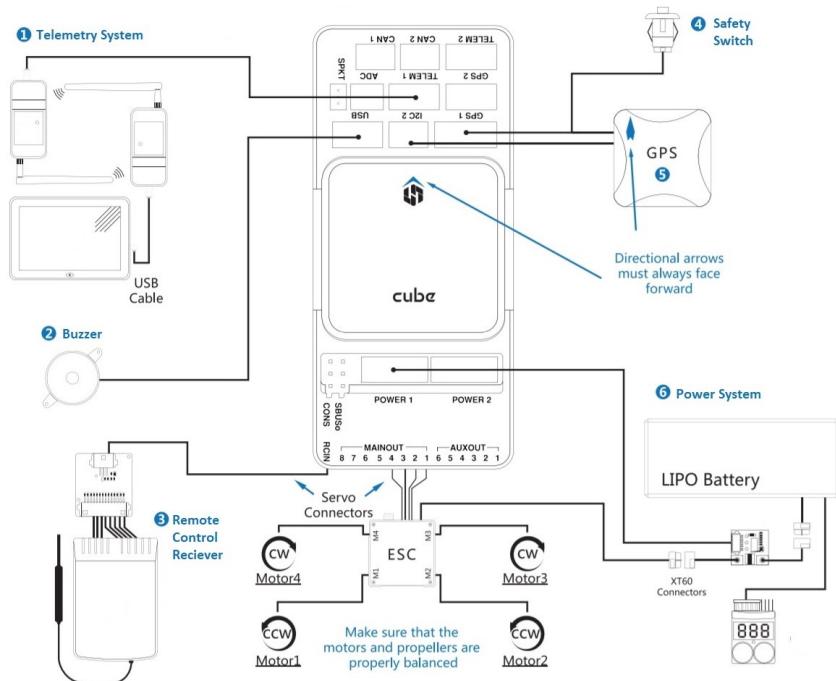


Figura 4.4: Conexões de hardware para a Pixhawk 2 Cube Hero.

<motivação da escolha desse hardware> <características desse hardware> <capacidades desse drone> <conseguir mais fotos do drone> <mostrar ligação dos componentes

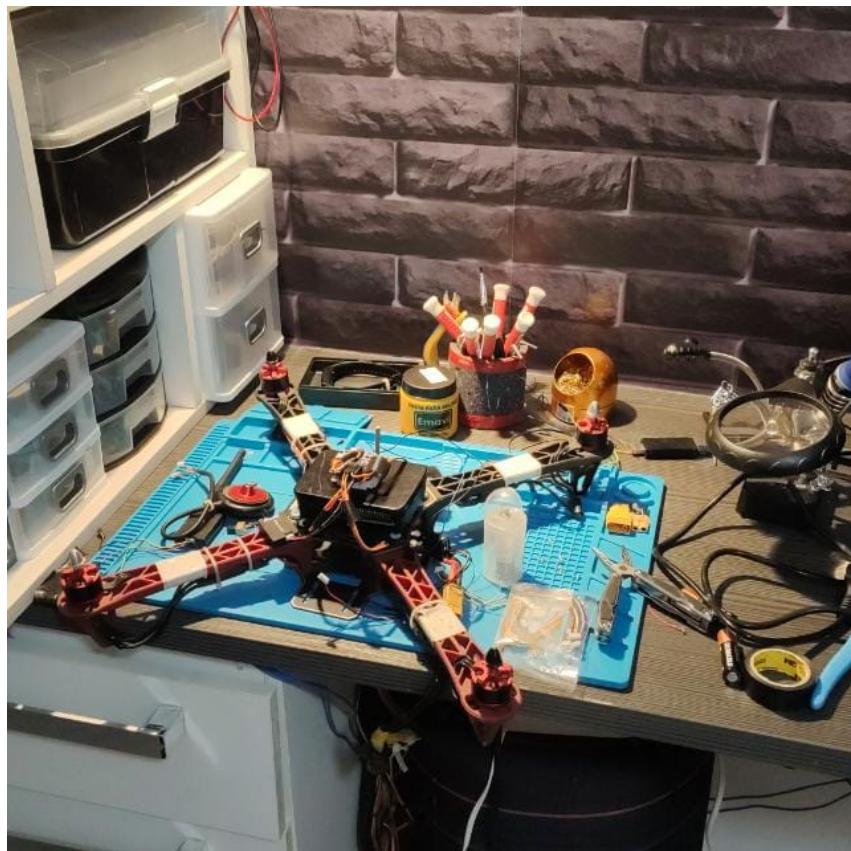


Figura 4.5: Bancada de trabalho com o drone quadricóptero.

e tudo funcionando perfeitamente> <motivação da escolha desse hardware> <características desse hardware> <capacidades desse drone>

### 4.3 Ambiente de simulação

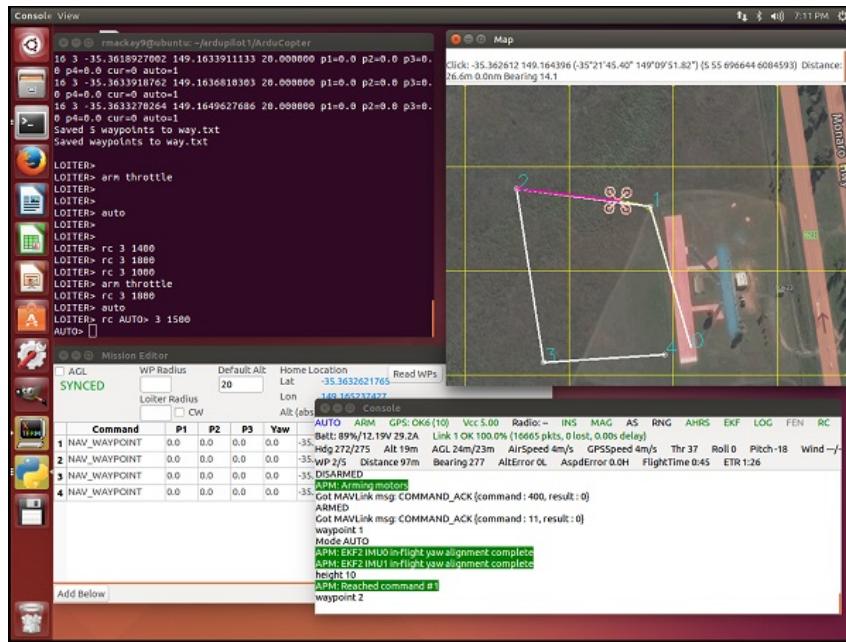


Figura 4.6: Demonstração do Software In The Loop.

O projeto ArduPilot possui um rico conjunto de ferramentas que auxiliam no desenvolvimento de aplicações para drones. O próprio código fonte do firmware ArduPilot pode ser compilado e carregado em um ambiente de simulação. O executável responsável por isso é chamado SITL (Software In The Loop), cujo diagrama é apresentado na figura 4.7, esse programa consegue integrar-se com outros formando um ambiente de simulação completo. O MavProxy é utilizado para fazer a interface de comunicação entre o operador e a controladora em ambiente simulado. Além disso, é possível integrar com um simulador de física e dessa forma simular a dinâmica de voo da aeronave. A simulação da dinâmica de voo da aeronave geralmente vem acompanhada com a integração de algum software de visualização como no caso do Flight Gear e do Gazebo, o qual estaremos utilizando. A integração de todos esses componentes no ambiente de simulação forma um loop, e com isso temos o nome Software In The Loop. Essa aplicação é essencial para o desenvolvimento pois permite a execução do ArduPilot sem a utilização de nenhum hardware.

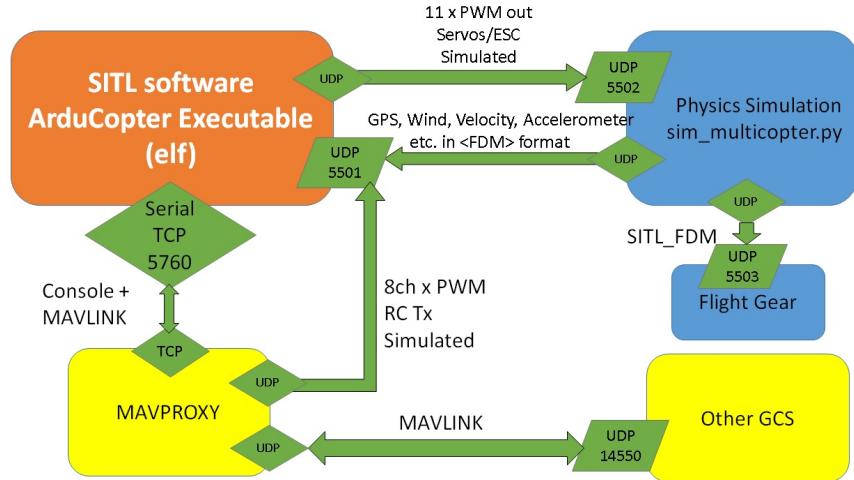


Figura 4.7: Arquitetura do Software In The Loop

A inicialização do SITL é feita utilizando o script python demonstrado na figura 4.8, que é disponibilizado junto ao código fonte do ArduPilot no seu repositório do github.

```
joaoneto@robotmaker:~$ cat startsitl_nogazebo.sh
#!/bin/bash
cd ~/ardupilot/ArduCopter/ && sim_vehicle.py -v ArduCopter -L UFF --console --map
joaoneto@robotmaker:~$ which sim_vehicle.py
/home/joaoneto/ardupilot/Tools/autotest/sim_vehicle.py
joaoneto@robotmaker:~$
```

Figura 4.8: sim\_vehicle.py

Como demonstrado na figura 4.8 criamos um script bash para executar o ambiente de simulação, nos parametros especificamos que queremos simular o firmware ArduCopter, utilizando as interfaces de mapa e console. Além disso, inserimos como parametro a localização em coordenadas do campo de futebol da UFF no Campus de Gragoatá. O resultado da execução desse script pode ser visualizado na figura 4.10.

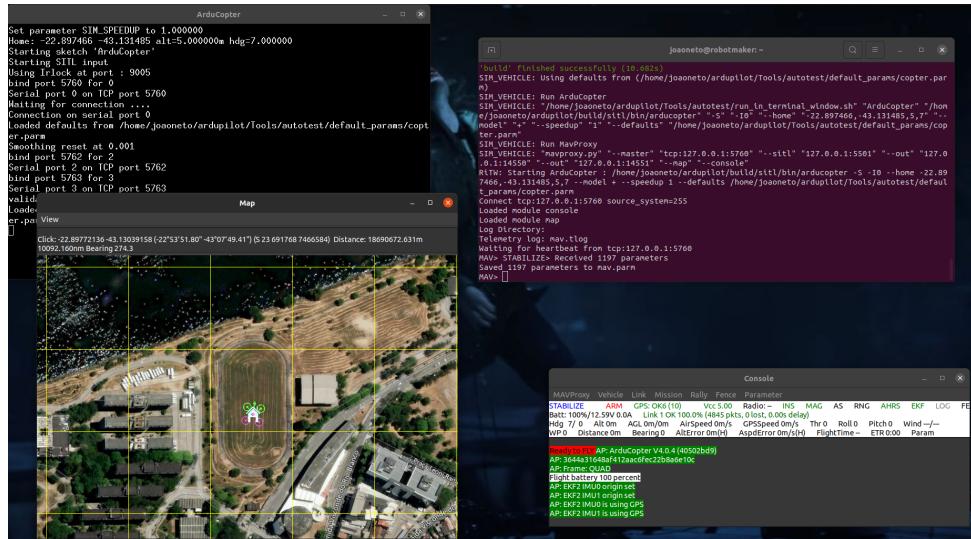


Figura 4.9: startsitl\_nogazebo.sh

Fizemos também um segundo script bash com os parametros necessários para a conexão com o Gazebo, que é o software que simula um mundo em 3D para o drone. Se paramos nesses dois scripts, pois na maioria das vezes a simulação em 3D não é necessária e consome muito dos recursos da máquina que a executa.

```
joaoneto@robotmaker:~$ cat startsitl.sh
#!/bin/bash
cd ~/ardupilot/ArduCopter/ && sim_vehicle.py -v ArduCopter -f gazebo-iris --console
joaoneto@robotmaker:~$
```

Figura 4.10: startsitl.sh com Gazebo

O modelo de mundo 3D utilizado no Gazebo para a integração com o SITL foi obtido a partir do repositório de um canal do *Youtube* chamado [Quads, 2022].

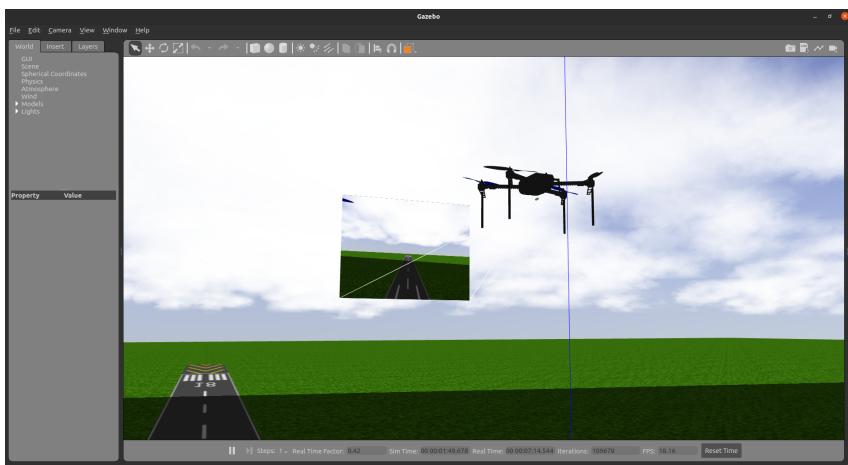


Figura 4.11: Software In The Loop com Gazebo

Essa instancia do Gazebo é iniciada a partir de um nó do ROS, que é executado com o comando da figura 4.12.

```
joaoneto@robotmaker:~$ roslaunch iq_sim runway.launch
... logging to /home/joaoneto/.ros/log/88ed4d48-62e1-11ed-a254-45cf1174f2cd/roslaunch-robotmaker-66411.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://robotmaker:40573/
SUMMARY
========
PARAMETERS
  * /gazebo/enable_ros_network: True
  * /rosdistro: noetic
  * /rosversion: 1.15.14
  * /use_sim_time: True
NODES
  /
    gazebo (gazebo_ros/gzserver)
    gazebo_gui (gazebo_ros/gzclient)
```

Figura 4.12: Nô do Gazebo - ROS

Com isso, obtemos um ambiente de teste de desenvolvimento para as aplicações que sucederão. Os benefícios de se utilizar um ambiente de simulação como este é que não

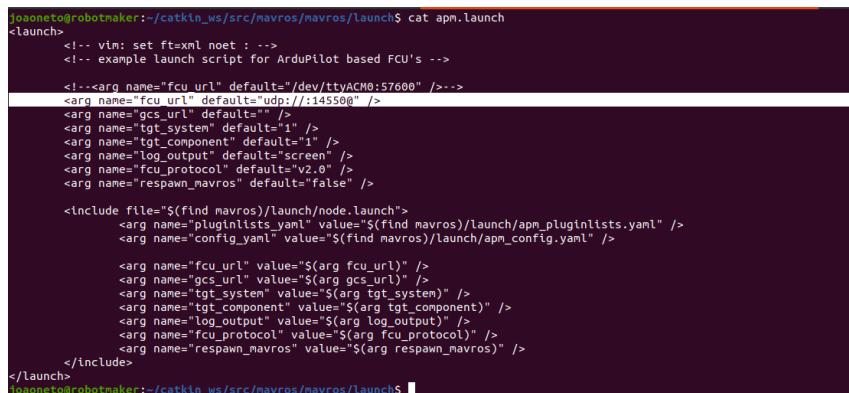
colocamos em risco o equipamento de hardware, nem as pessoas que seriam envolvidas em testes de campo. Além disso, o teste no ambiente computacional se torna mais prático, descartando toda a logística necessária para se deslocar com todo o equipamento para um campo aberto e com baixa densidade populacional.

O processo de instalação do ambiente de desenvolvimento do ArduPilot está bem documentado nos appendices deste documento, nesse processo também está incluído a instalação de dependencias para executar o ambiente de simulação.

## 4.4 Ambiente de Simulação com o Mavros

O Mavros como dito na parte introdutória, é um nó de ROS que publica e subscreve dados da controladora de voo, essas mensagens são trocadas utilizando o protocolo MAVLink. O Mavros é geralmente configurado para realizar a comunicação através de um dispositivo serial, que no linux é identificado por um arquivo no diretório /dev. Entretanto, para fazer o Mavros se comunicar com o SITL, deve-se fazer uma modificação no arquivo de inicialização, arquivo com extensão *.launch* no diretório launch do pacote.

No diretório launch do pacote são observados vários arquivos *.launch* utilizados para iniciar o nó do Mavros com diferentes parâmetros. O *apm.launch* foi modificado como indicado na figura 4.13, para utilizar a porta de rede específica do MavProxy no SITL e não o dispositivo serial, comentado na linha superior.



```

joaoneto@robotmaker:~/catkin_ws/src/mavros/mavros/launch$ cat apm.launch
<launch>
  <!-- vim: set ft=xml noet : -->
  <!-- example launch script for ArduPilot based FCU's -->
  <!--<arg name="fcu_url" default="/dev/ttyACM0:57600" />-->
  <arg name="fcu_url" default="udp://:14550@" />
  <arg name="gcs_url" default="" />
  <arg name="tgt_system" default="1" />
  <arg name="tgt_component" default="1" />
  <arg name="log_output" default="screen" />
  <arg name="fcu_protocol" default="v2.0" />
  <arg name="respawn_mavros" default="false" />

  <include file="$(find mavros)/launch/node.launch">
    <arg name="pluginlists_yaml" value="$(find mavros)/launch/apm_pluginlists.yaml" />
    <arg name="config_yaml" value="$(find mavros)/launch/apm_config.yaml" />

    <arg name="fcu_url" value="$(arg fcu_url)" />
    <arg name="gcs_url" value="$(arg gcs_url)" />
    <arg name="tgt_system" value="$(arg tgt_system)" />
    <arg name="tgt_component" value="$(arg tgt_component)" />
    <arg name="log_output" value="$(arg log_output)" />
    <arg name="fcu_protocol" value="$(arg fcu_protocol)" />
    <arg name="respawn_mavros" value="$(arg respawn_mavros)" />
  </include>
</launch>
joaoneto@robotmaker:~/catkin_ws/src/mavros/mavros/launch$ 

```

Figura 4.13: *apm.launch* do Pacote Mavros

Com essa modificação, o nó do Mavros pode ser iniciado normalmente em conjunto com o SITL e é possível subscrever e publicar mensagens MAVLink nos tópicos para a controladora de voo no ambiente simulado.

```
joaoneto@robotmaker:~$ roslaunch mavros apm.launch
... Logging to /home/joaoneto/.ros/log/5a7edd4c-02f1-11ed-a254-45cf1174f2cd/roslaunch-robotmaker-150544.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://robotmaker:41853/
SUMMARY
========
CLEAR PARAMETERS
  * /mavros/
PARAMETERS
  * /mavros/cmd/use_comp_id_system_control: False
  * /mavros/conn/heartbeat_mav_type: ONBOARD_CONTROLLER
  * /mavros/conn/heartbeat_rate: 1.0
  * /mavros/conn/system_time_rate: 1.0
  * /mavros/conn/timeout: 10.0
  * /mavros/conn/timesync_rate: 10.0
  * /mavros/distance_sensor/rangefinder_pub/field_of_view: 0.0
  * /mavros/distance_sensor/rangefinder_pub/frame_id: lidar
  * /mavros/distance_sensor/rangefinder_pub/id: 0
  * /mavros/distance_sensor/rangefinder_pub/send_tf: False
  * /mavros/distance_sensor/rangefinder_pub/sensor_position/x: 0.0
  * /mavros/distance_sensor/rangefinder_pub/sensor_position/y: 0.0
  * /mavros/distance_sensor/rangefinder_pub/sensor_position/z: -0.1
```

Figura 4.14: Execução do Mavros

Alguns dos tópicos provados pelo Mavros são listados pelo comando rostopic list na figura 4.15.

```
joaoneto@robotmaker:~$ rostopic list
/diagnostics
/mavlink/from
/mavlink/gcs_ip
/mavlink/to
/mavros/adsb/send
/mavros/adsb/vehicle
/mavros/battery
/mavros/battery2
/mavros/cam_imu_sync/cam_imu_stamp
/mavros/companion_process/status
/mavros/distance_sensor/rangefinder_pub
/mavros/distance_sensor/rangefinder_sub
/mavros/esc_info
/mavros/esc_status
/mavros/esc_telemetry
/mavros/estimator_status
/mavros/extended_state
/mavros/fake_gps/mocap/pose
/mavros/geofence/waypoints
/mavros/global_position/compass_hdg
/mavros/global_position/global
/mavros/global_position/gp_lp_offset
/mavros/global_position/gp_origin
/mavros/global_position/home
/mavros/global_position/local
/mavros/global_position/raw/fix
/mavros/global_position/raw/gps_vel
/mavros/global_position/raw/satellites
/mavros/global_position/rel_alt
/mavros/global_position/set_gp_origin
/mavros/gps_input/gps_input
/mavros/gps_rtk/rtk_baseline
```

Figura 4.15: Tópicos do Mavros

Por exemplo, o tópico correspondente a localização GPS da controladora de voo é o global\_position. É possível subscrever e acompanhar as mensagens que chegam da controladora nesse tópico pelo comando da figura 4.16.

```
lasonata@robotmaker: $ rostopic echo /mavros/global_position/global
header:
  seq: 17
  stamp:
    secs: 1668302825
    nsecs: 754978665
    frame_id: "base_link"
status:
  status: 0
  service: 1
latitude: -22.897466
longitude: -43.131485
altitude: -0.5059762761688713
position_covariance: [0.040000000000000001, 0.0, 0.0, 0.0, 0.0, 0.040000000000000001, 0.0, 0.0, 0.0, 0.0, 0.040000000000000001]
position_covariance_type: 2
...
header:
  seq: 18
  stamp:
    secs: 1668302825
    nsecs: 972859208
    frame_id: "base_link"
status:
  status: 0
  service: 1
latitude: -22.897466
longitude: -43.131485
altitude: -0.5059762761688713
position_covariance: [0.040000000000000001, 0.0, 0.0, 0.0, 0.0, 0.040000000000000001, 0.0, 0.0, 0.0, 0.0, 0.040000000000000001]
position_covariance_type: 2
...
```

Figura 4.16: Tópicos do Mavros

Os tópicos do ROS são facilmente acessados através de linguagens de programação como C++ e Python. Em particular esta última, possui um módulo chamado rospy que permite a criação de aplicações do ROS, com este módulo é possível desenvolver nós inteiramente usando Python. Agora que temos o Mavros agindo como interface da aeronave, é possível desenvolver o software embarcado que também será um nó, ele irá se comunicar com o Mavros e com a aplicação em Django que será desenvolvida.

## 4.5 Arquitetura da aplicação

<mostrar diagrama estrutural>

1. mavros
2. iq\_gnc
3. drone\_telemetry\_over\_network
4. drone\_control\_over\_network
5. Drone\_CCS\_API (Cloud Control Station)

## 4.6 Arquitetura da aplicação de rede

Modelo cliente servidor, no qual o raspberry e o servidor da API estão numa mesma VPN.

## 4.7 Outras arquiteturas possíveis para o sistema proposto

LTE e 5G.

# **Capítulo 5**

## **Resultados**

apresentação de resultados (numéricos e/ou gráficos) de cálculos e/ou de simulações, requeridos na especificação do trabalho.

# Capítulo 6

## Conclusão

Hipótese testada Se realizarmos o experimento em outro ambiente com maior capacidade para que todos possam assistir de forma homogênea atenderemos a demanda da turma, teremos um aprendizado mais efetivo e melhoria no tempo dedicado a esse experimento.

Responda as perguntas avaliativas para definir a conclusão

1. Com o sistema desenvolvido será possível ao professor usuário controlar o experimento proposto remotamente?
2. pergunta avaliativa 2
3. pergunta avaliativa 3
4. pergunta avaliativa 4

Esta monografia tem como base a pesquisa, a elaboração e o teste, em ambiente simulado, de um drone capaz de se comunicar por uma rede IP, no nosso caso, uma rede *wi-fi*.

Aproveitou-se conhecimentos adquiridos na equipe UFFO [Equipe UFFO, ], como *softwares* e bibliotecas específicas para automação. A partir dessa base, surgiu a ideia de incrementar o *hardware* de um drone comum acoplando um *raspberry*. A grande adição é o acesso ao protocolo IP, permitindo o drone trocar dados sem limitações de distância. Para realizar a comunicação com o drone, foi elaborada uma arquitetura de cliente-servidor, cujo cliente, em resumo o drone, envia e recebe dados do servidor, ambos conectados numa mesma rede, via VPN. Por fim, realizou-se uma página *web* como *interface* gráfica final, a qual permite o envio de missões ao drone e exibe os dados essenciais de telemetria em tempo real.

No geral, os resultados foram compatíveis aos objetivos propostos, realizaram-se testes no ambiente de simulação que demonstraram a viabilidade de um projeto prático.

O projeto como um todo ainda deve ser aperfeiçoado caso haja interesse em comercialização. Faz-se necessário a realização de testes com um drone fora do ambiente de simulação, e como citado no Capítulo 1 3, idealmente, conectar um modem LTE ou NR ao *raspberry*. Ambos, os testes e a conexão do modem, podem ser feitos por futuros integrantes da equipe UFFO.

# Capítulo 7

## Sugestões para trabalhos futuros

Tendo em mente as características do projeto apresentadas neste trabalho, podemos imaginar diversas aplicações para serem acrescentadas. Somos capazes, portanto, de listar algumas das possíveis aplicações e seus possíveis desdobramentos.

1. Melhorias na interface de controle
  - (a) Adição de novos funcionalidades
  - (b) Coleta e apresentação de mais dados do drone
2. Integração de um modem de internet móvel
  - (a) Controle e coleta de dados a qualquer distância, desde que haja sinal de internet móvel.
3. Integração de uma câmera
  - (a) Visão em tempo real do ambiente o qual o drone sobrevoa
  - (b) Adição de inteligência artificial capaz de tratar e processar imagens capturadas

Para cada uma dessas adições deve-se realizar testes de viabilidade, bem como estudos mais profundos do impacto que cada complemento teria no modelo final deste trabalho.

Portanto, geradas essas necessidades, podemos utilizar o ambiente de simulação do gazebo, junto aos módulos ROS, para prever como se comportará o sistema. Comparações a respeito do desempenho geral podem e devem ser feitas a fim de comprovar bons resultados.

# Referências Bibliográficas

- [Ardupilot Docs, 2021] Ardupilot Docs (2021). **Ardupilot Docs**. Disponível em: “<http://ardupilot.org/dev/#welcome-to-the-ardupilot-development-site>”. Acesso em: 25/01/2021.
- [Companion Computers, 2021] Companion Computers (2021). **Companion Computers**. Disponível em: “<https://ardupilot.org/dev/docs/companion-computers.html>”. Acesso em: 25/01/2021.
- [Equipe UFFO, ] Equipe UFFO. **Equipe UFFO**. Disponível em: “<https://uffoequipe.github.io/website/uffo/>”. Acesso em: 25/01/2021.
- [Foundation, ] Foundation, D. S. **Django at a glance**. <https://docs.djangoproject.com/en/4.1/intro/overview/>. Accessed: 2022-9-25.
- [Foundation, 2020] Foundation, R. P. (2020). **URL da plataforma Raspberry Pi**. Disponível em: “<https://www.raspberrypi.org/>”. Acesso em: 25/01/2021.
- [Laddha and Thakare, 2013] Laddha, N. R. and Thakare, A. P. (2013). **a review on serial communication by uart** . Disponível em: “[https://d1wqtxts1xzle7.clo  
udfront.net/35186800/V3I1-0220-with-cover-page-v2.pdf?Expires=1660489200&Signature=L1W0joCWNEXiG0JGR8xEk8ERpygEDR08dkWQxiz0iYxoG~45wkWuOrH69ar14G0X01YputTpTYBYY8ELUEDnj7x88wnXtNB8xu05kZo6BtKcN80pkWIA99tLMgZcYHfoIHvzsU143iLiAa~5VLWo-EHH1HCtXdcJ2lazswkgqT3CG-D0wpTfjfLaGsqRzV4aKoBju3DufAqSBefW8T8dHMkUEd3hX0x2Jmi2HJcfuMCTHtFRQX4dn1CkeyjABUWE1MMXc2adpzIzhNGcU4yk3cVFV22IVutoZi2G-xfRwxRooF1AR81fIBJA3pkf0Fx8KLokzQEyPmC4S81D1NSh7A\\_\\_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA](https://d1wqtxts1xzle7.cloudfront.net/35186800/V3I1-0220-with-cover-page-v2.pdf?Expires=1660489200&Signature=L1W0joCWNEXiG0JGR8xEk8ERpygEDR08dkWQxiz0iYxoG~45wkWuOrH69ar14G0X01YputTpTYBYY8ELUEDnj7x88wnXtNB8xu05kZo6BtKcN80pkWIA99tLMgZcYHfoIHvzsU143iLiAa~5VLWo-EHH1HCtXdcJ2lazswkgqT3CG-D0wpTfjfLaGsqRzV4aKoBju3DufAqSBefW8T8dHMkUEd3hX0x2Jmi2HJcfuMCTHtFRQX4dn1CkeyjABUWE1MMXc2adpzIzhNGcU4yk3cVFV22IVutoZi2G-xfRwxRooF1AR81fIBJA3pkf0Fx8KLokzQEyPmC4S81D1NSh7A__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA)”. Acesso em: 25/01/2021.
- [Lassi Sundqvist, 2015] Lassi Sundqvist (2015). **cellular controlled drone experiment: evaluation of network requirements**. Disponível em: “[https://aaltooc.aalto.fi/bitstream/handle/123456789/19152/master\\_Sundqvist\\_Lassi\\_2015.pdf?sequence=1&isAllowed=y](https://aaltooc.aalto.fi/bitstream/handle/123456789/19152/master_Sundqvist_Lassi_2015.pdf?sequence=1&isAllowed=y)”. Acesso em: 25/01/2021.

[MAVLink, ] MAVLink. ***mavlink developer guide***. Disponível em: “<https://mavlink.io/en/>”. Acesso em: 25/01/2021.

[Members, 2019a] Members, W. (2019a). **URL do termo Raspberry Pi**. Disponível em: “[https://pt.wikipedia.org/wiki/Raspberry\\_Pi](https://pt.wikipedia.org/wiki/Raspberry_Pi)”. Acesso em: 25/01/2021.

[Members, 2019b] Members, W. (2019b). **URL do termo System on a chip (SoC)**. Disponível em: “<https://pt.wikipedia.org/wiki/System-on-a-chip>”. Acesso em: 25/01/2021.

[Quads, 2022] Quads, I. (2022). **URL Tutoriais do Canal de Youtube Intelligent Quads**. Disponível em: “[https://github.com/Intelligent-Quads/iq\\_tutorials](https://github.com/Intelligent-Quads/iq_tutorials)”. Acesso em: 25/01/2021.

[Software in the Public Interest, Inc, 2021] Software in the Public Interest, Inc (2021). **Debian OS**. Disponível em: “<https://www.debian.org/>”. Acesso em: 25/01/2021.

# Apêndice A

## Apêndice 1

Segue abaixo um guia de instalação completo dos *Softwares* de simulação e suas respectivas dependências para Ubuntu 20.04 LTS.

### A.1 Intalando o ArduPilot e MavProxy

#### A.1.1 Clonando o repositório *git* para sua máquina

```
$ cd ~  
$ sudo apt install git  
$ git clone https://github.com/ArduPilot/ardupilot.git
```

#### A.1.2 Instalando as dependências e recompilando o perfil

```
$ cd ardupilot/Tools/environment_install/install-prereqs-  
ubuntu.sh -y  
$ . ~/.profile
```

#### A.1.3 Mudando para a *branch* do ArduCopter

```
$ git checkout Copter-4.0.4  
$ git submodule update --init --recursive
```

#### A.1.4 Rodando SITL (*Software In The Loop*)

```
$ cd ~/ardupilot/ArduCopter
$ sim_vehicle.py -w
```

## A.2 Instalando o Gazebo e o *plugin* do ArduPilot

### A.2.1 Atualizando a lista de fontes para *download*

```
$ sudo sh -c 'echo "deb http://packages.osrfoundation.org/
gazebo/ubuntu-stable `lsb_release -cs` main" > /etc/apt/
sources.list.d/gazebo-stable.list'
$ wget http://packages.osrfoundation.org/gazebo.key -O - |
  sudo apt-key add -
$ sudo apt update
```

### A.2.2 Instalando o *plugin* do Gazebo para comunicação com o ArduPilot

```
$ cd ~
$ git clone https://github.com/khancyr/ardupilot_gazebo.git
$ cd ardupilot_gazebo
$ mkdir build
$ cd build
$ cmake ..
$ make -j4
$ sudo make install
$ echo 'source /usr/share/gazebo/setup.sh' >> ~/.bashrc
$ echo 'export GAZEBO_MODEL_PATH=~/ardupilot_gazebo/models'
>> ~/.bashrc
$ . ~/.bashrc
```

### A.2.3 Executando a simulação

No primeiro terminal **do** linux rode o Gazebo:

```
$ gazebo --verbose ~/ardupilot_gazebo/worlds/
iris_arducopter_runway.world
```

```
No segundo terminal do linux rode o SITL:  

$ cd ~/ardupilot/ArduCopter/  

$ sim_vehicle.py -v ArduCopter -f gazebo-iris --console
```

## A.3 Instalando o *ROS* e o configurando o *Catkin*

### A.3.1 Atualizando a lista de fontes para *download*

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'  

$ sudo apt install curl  

$ curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -  

$ sudo apt update
```

### A.3.2 Instalando o *ROS*

```
$ sudo apt install ros-noetic-desktop-full
```

### A.3.3 Configurando o ambiente

```
$ source /opt/ros/noetic/setup.bash  

$ echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc  

$ source ~/.bashrc  

$ echo "source /opt/ros/noetic/setup.zsh" >> ~/.zshrc  

$ source ~/.zshrc
```

### A.3.4 Instalando as dependências para os pacotes dos *ROS*

```
$ sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential  

$ sudo apt install python3-rosdep  

$ sudo rosdep init  

$ rosdep update
```

### A.3.5 Configurando o *Catkin*

```
$ sudo apt-get install python3-wstool python3-rosinstall-
  generator python3-catkin-lint python3-pip python3-catkin-
  tools
$ pip3 install osrf-pycommon
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws
$ catkin init
```

### A.3.6 Instalando as dependências para o *Catkin*

```
$ sudo apt-get install python3-wstool python3-rosinstall-
  generator python3-catkin-lint python3-pip python3-catkin-
  tools
$ pip3 install osrf-pycommon
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws
$ catkin init
```

### A.3.7 Instalando as *MAVRROS* e *MAVLink*

```
$ cd ~/catkin_ws
$ wstool init ~/catkin_ws/src
$ rosinstall_generator --upstream mavros | tee /tmp/mavros.
  rosinstall
$ rosinstall_generator mavlink | tee -a /tmp/mavros.
  rosinstall
$ wstool merge -t src /tmp/mavros.rosinstall
$ wstool update -t src
$ rosdep install --from-paths src --ignore-src --rosdistro 'echo $ROS_DISTRO' -y
$ catkin build
```

### A.3.8 Atualizando o arquivo *.bachrc*

```
$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
$ source ~/.bashrc
```

## A.4 Instalando as dependências geográficas e clonando o repositório de simulação do *Intelligent Quads*

### A.4.1 Instalando as dependências geográficas

```
$ sudo ~/catkin_ws/src/mavros/mavros/scripts/
install_geographiclib_datasets.sh]
```

### A.4.2 Clonando pacote ROS de simulação do *Intelligent Quads*

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/Intelligent-Quads/iq_sim.git
$ echo "GAZEBO_MODEL_PATH=${GAZEBO_MODEL_PATH}:$HOME/catkin_ws
/src/iq_sim/models" >> ~/.bashrc
$ cd ~/catkin_ws
$ catkin build
$ source ~/.bashrc
```

## A.5 Instalando o *QGround Control*

### A.5.1 Alterando permissões e instalando o *QGround Control*

```
$ sudo usermod -a -G dialout $USER
$ sudo apt-get remove modemmanager
$ wget https://s3-us-west-2.amazonaws.com/qgroundcontrol/
latest/QGroundControl.AppImage
$ chmod +x ./QGroundControl.AppImage
$ ./QGroundControl.AppImage
```