

**UNIVERSIDADE REGIONAL DE BLUMENAU**  
**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**  
**CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO**

**DRONE AUTÔNOMO: VIGILÂNCIA AÉREA DE ESPAÇOS  
EXTERNOS**

**DIEGO FACHINELLO CORRÊA**

**BLUMENAU**  
**2020**

**DIEGO FACHINELLO CORRÊA**

# **DRONE AUTÔNOMO: VIGILÂNCIA AÉREA DE ESPAÇOS EXTERNOS**

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Prof. Dalton Solano dos reis, M.Sc. - Orientador

**BLUMENAU  
2020**

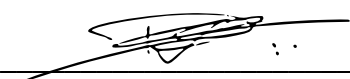
# **DRONE AUTÔNOMO: VIGILÂNCIA AÉREA DE ESPAÇOS EXTERNOS**

Por

**DIEGO FACHINELLO CORREA**

Trabalho de Conclusão de Curso aprovado para  
obtenção dos créditos na disciplina de Trabalho  
de Conclusão de Curso II pela banca  
examinadora formada por:

Presidente:



---

Prof(a). Dalton Solano dos Reis – Orientador(a), FURB

Membro:

---

Prof(a). Francisco Adell Péricas – FURB

Membro:

---

Prof(a). Miguel Alexandre Wisintainer – FURB

Blumenau, 13 de julho de 2020

Dedico este trabalho primeiramente ao universo como um todo, a grandes amizades e oportunidades conquistadas durante o curso, aos professores pela transferência de todo conhecimento e principalmente pelos discursos morais, especialmente a Universidade por proporcionar esses não tão longos anos de muito crescimento pessoal e profissional.

## **AGRADECIMENTOS**

Ao universo e a Deus por permitir a grandeza da vida e por todos os acontecimentos bons durante ela, além das grandezas que estão por vir.

À minha futura esposa, Juliana dos Santos, apoiadora e entusiasta de tecnologia, me proporcionou conforto nos momentos difíceis, apoiando no engajamento da finalização desse trabalho e na conclusão do curso.

Aos professores do curso de Sistemas de Informação que contribuíram com conhecimento e aprendizado ao decorrer desses 5 anos.

Ao meu orientador, Dalton Solano dos Reis por nortear o desenvolvimento desse trabalho desde a proposta até a monografia.

À professora Simone Erbs da Costa por proporcionar aulas diferenciadas e aplicabilidade de novas metodologias e ferramentas de ensino, além de bancar aquela pizza quentinha ao final do semestre.

A todos meus colegas de classe que contribuíram de alguma forma a estudar, aprender e passar nas provas, além da grande parceria no desenvolvimento dos trabalhos e aquelas gargalhadas divertidas durante as apresentações.

I am the bone of my sword  
Steel is my body and fire is my blood  
I have created over a Thousand blades  
Unknown to death  
Nor known to life  
Have withstood pain to create many weapons  
Yet those hands shall never hold anything  
So, as I pray, Unlimited Blade Works.

Emiya Shiro – Fate/Stay night

## RESUMO

Este trabalho descreve o desenvolvimento de uma arquitetura de voo autônomo aplicada no modelo AR.Drone 2.0 da Parrot, e tem como objetivo gerenciar percursos de rotas pré-estabelecidas, visando principalmente aplicabilidade de vigilância aérea em espaços externos. O desenvolvimento foi realizado utilizando o *framework* Node.js integrado a bibliotecas NPM como node-ar-drone, ardrone-autonomy e Geolib. Para a seleção do percurso e coleta das coordenadas foi utilizada biblioteca Leaflet e API do Google Maps. Os algoritmos implementados permitem o cadastramento do percurso, acompanhamento das imagens em tempo real e inclui centro de comandos em tela única, habilitado com opções essenciais de usabilidade amigável. As ferramentas e bibliotecas utilizadas se mostraram adequadas, abstraindo milhares de linhas de código e fórmulas matemáticas. O módulo GPS possui algumas limitações relacionadas a precisão da atual posição do drone, porém ao executar os testes obteve-se êxito em realizar a vigilância de espaços externos, devido ao adequado uso dos recursos técnicos da arquitetura e de hardware do AR.Drone. A arquitetura foi capaz de pilotar autonomamente o drone ao executar os percursos selecionados no mapa, ao final do trajeto trazendo-o de volta a base de origem, finalizando a vigilância.

Palavras-chave: Drone. Ardrone. Parrot. Voo autônomo. GPS. Vigilância. Node. Geolib.

## **ABSTRACT**

This work describes the development of an autonomous flight architecture applied in the Parrot AR.Drone 2.0 model, which aims to manage the courses of pre-applied routes, using the main air applications in external spaces. The development was carried out using the Node.js framework, integrated with npm libraries such as node-ar-drone, ardrone-autonomy and Geolib. For a selection of tracking and coordinate collection, the Leaflet and Google Maps API were used. The implemented algorithms makes available the tracking selection, the first person view of images in real time and include the command center on single screen, enabled with specific options for user friendly. The tools and libraries used had abstracted thousands lines of code and mathematical equations with high complexity. The GPS module has some issues related to the accuracy of the drone current position, however when executing the tests the architecture was able to accomplish the fly with successfully, it were autonomous carried out in external spaces, this was obtained due to the adequate use of the developed architecture and the AR.Drone hardware technical resources. The architecture where able to autopilot the drone and executed the selected routes on the map, at the end of the route the architecture has returned the drone to his initial place, finally ending surveillance.

Key-words: Drone. Ardrone. Parrot. Autonomous fly. GPS. Surveillance. Node. Geolib.



## LISTA DE FIGURAS

Figura 1 - AR.Drone 2.0 Parrot .....	18
Figura 2 – Modulo de GPS Ar.Drone.....	21
Figura 3 - Mapa com <i>waypoints</i> .....	27
Figura 4 – Arquitetura VisEdu-Drone .....	28
Figura 5 - Apresentação e navegação na rota .....	29
Figura 6 - Work Breakdown Structure .....	30
Figura 7 – Casos de uso .....	32
Figura 8 – Atividade de configuração da arquitetura .....	33
Figura 9 – Atividade de selecionar rota .....	35
Figura 10 – Atividade de executar decolagem .....	37
Figura 11 – Atividade de executar trajeto .....	39
Figura 12 – Atividade de retorno a base.....	40
Figura 13 – Atividade de executar pouso .....	42
Figura 14 – Atividade limpar rota .....	43
Figura 15 - Estrutura da arquitetura.....	46
Figura 16 – Botões interface web da arquitetura .....	50
Figura 17 – Interface web da arquitetura.....	52
Figura 18 – Interface web controle e mapa de navegação.....	54
Figura 19 – Interface web FPV, parâmetros e dados de navegação .....	55
Figura 20 – Bateria lipo powerbolt.....	56
Figura 21 – Casco externo customizado parte inferior.....	57
Figura 22 – Casco externo customizado parte superior.....	58
Figura 23 – Cenário de teste 2 .....	60
Figura 24 – Cenário de teste 3 .....	61

## LISTA DE QUADROS

Quadro 1 - Caso de uso configurar arquitetura.....	33
Quadro 2 – Caso de uso selecionar rota.....	34
Quadro 3 – Caso de uso executar decolagem.....	36
Quadro 4 – Caso de uso executar trajeto .....	38
Quadro 5 – Caso de uso executar retorno a base.....	40
Quadro 6 – Caso de uso executar pouso .....	41
Quadro 7 – Caso de uso limpar rota .....	43
Quadro 8 - Arquivo de dependências .....	45
Quadro 9 - Comparativo entre os trabalhos correlatos .....	63

## **LISTA DE TABELAS**

Tabela 1 - Diferenças bateria original e paralela .....	58
Tabela 2 – Diferenças casco original e customizado.....	58
Tabela 3 – Resultados da execução do cenário 1 .....	59
Tabela 4 – Resultados da execução do cenário 2 .....	61
Tabela 5 – Resultados da execução do cenário 3 .....	62

## **LISTA DE ABREVIATURAS E SIGLAS**

API – Application Programming Interface

ARM – Advanced RISC Machine

AT – AuTonomous

AVR – Alf-egil bogen e VegaRd wollan

BIT – BInary digiT

CPU – Central Processing Unit

CSS – Cascading Style Sheets

CSV – Comma Separated Values

DDR – Double Data Rate

DSP – Digital Signal Processor

FPS – Frames Per Second

FURB – Fundação da Universidade Regional de Blumenau

GB – GigaByte

GBIT – GigaBIT

GEO – GEographic LOcation

GHZ – GigaHertZ

GLONASS – GLobal (Orbiting) NAVigation Satellite System

GNSS – Global Navigation Satellite System

GPS – Global Positioning System

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

HZ – HertZ

ID – IDentity

JPEG – Joint Photographics Experts Group

JS – JavaScript

JSON – JavaScript Object Notation

KB – KiloByte

MG - MiliGrama

MHZ – MegaHertz

MIPS – Microprocessor without Interlocked Pipeline Stages

MS – MilisSegundo

NPM – Node Package Manager

OBJ – OBJect file wavefront

PA– Precision and Accuracy

PNG – Portable Network Graphics

QVGA – Quarter Video Graphics Array

RAM – Random Access Memory

RF – Requisitos Funcionais

RNF – Requisitos Não Funcionais

ROS – Robot Operation System

RPM – Rotação Por Minuto

SDK – Software Development Kit

TCP – Transmission Control Protocol

UC – User Cases

UDP – User Datagram Protocol

UI – User Interface

UML – Unified Modeling Language

USB – Universal Serial Bus

UX – User Experience

W – Watt

WI-FI – Wireless Fidelity

## SUMÁRIO

<b>1 INTRODUÇÃO.....</b>	<b>16</b>
1.1 OBJETIVOS.....	16
1.2 ESTRUTURA.....	17
<b>2 FUNDAMENTAÇÃO TEÓRICA .....</b>	<b>18</b>
2.1 ESPECIFICAÇÕES TÉCNICAS E DE HARDWARE AR.DRONE 2.0.....	18
2.2 BIBLIOTECA NPM NODE-AR-DRONE.....	21
2.3 BIBLIOTECA ARDRONE-AUTONOMY .....	23
2.4 BIBLIOTECA GEOLIB.....	25
2.5 API GOOGLE MAPS .....	26
2.6 TRABALHOS CORRELATOS .....	27
2.6.1 VISEDU-DRONE: MÓDULO DE INTEGRAÇÃO COM ROS .....	28
2.6.2 FURB MOBILE: SISTEMA MÓVEL MULTIPLATAFORMA PARA NAVEGAÇÃO EM ROTAS INTERNAS.....	29
2.6.3 AUTONOMOUS NAVIGATION AND SEARCH IN AN INDOOR ENVIRONMENT USING AN AR.DRONE .....	30
<b>3 DESENVOLVIMENTO DA ARQUITETURA .....</b>	<b>31</b>
3.1 REQUISITOS PRINCIPAIS .....	31
3.2 ESPECIFICAÇÃO .....	31
3.2.1 Casos de uso.....	31
3.3 IMPLEMENTAÇÃO .....	44
3.3.1 Técnicas e ferramentas utilizadas.....	44
3.3.2 Estruturas e protocolos de comunicação .....	50
3.4 OPERACIONALIDADE DA IMPLEMENTAÇÃO .....	51
3.4.1 Conectar arquitetura ao AR.Drone 2.0.....	51
3.4.2 Tela principal de operacionalidade da arquitetura .....	52
3.4.3 Tela de operacionalidade da arquitetura .....	53
3.4.4 Tela analítica da arquitetura .....	54
3.5 RESULTADOS E DISCUSSÕES.....	55
3.5.1 Customização AR.Drone 2.0.....	56
3.5.2 Cenário de teste 1 .....	58
3.5.3 Cenário de teste 2 .....	59

3.5.4 Cenário de teste 3 .....	61
3.5.5 Comparativo entre os trabalhos correlatos .....	62
<b>4 CONCLUSÕES.....</b>	<b>64</b>
4.1 EXTENSÕES .....	65
<b>REFERÊNCIAS .....</b>	<b>66</b>



## 1 INTRODUÇÃO

O mundo tecnológico do qual vivemos atualmente se assemelha a uma ficção científica, tecnologias de ponta que estão aplicadas em diversos campos da ciência. Para muitos o espanto ainda é grande e confuso, pois grande parte dessa tecnologia trouxe mudanças em nossa sociedade, tal como novos modelos de trabalho e novas profissões. Uma dessas tecnologias são os drones que se assemelham a um brinquedo ou hobby de criança grande, porém não se limita apenas a isso, há também inúmeras maneiras dessa tecnologia ser utilizada em prol do apoio em diversas áreas, principalmente com vigilância em áreas abertas e de grande circulação de pessoas (SHIRATSUCHI, 2014).

No Brasil, esta máquina é chamada Vant – Veículo Aéreo Não Tripulado) ou “drone” (zangão, na língua inglesa), miniaturas derivadas dos aviões não tripulados produzidos de forma contínua pela indústria bélica há pelo menos 20 anos, principalmente nos Estados Unidos (SHIRATSUCHI, p.16, 2014).

Segundo Estêvão (2014), a preocupação com a vigilância se tornou comum em todas as sociedades e estados, com grande aumento nos investimentos em monitoração e com a recente chegada de novas tecnologias, trazendo uma nova realidade. Atualmente os Estados Unidos da América possui projetos de investigação para detectar ações criminalistas e de terrorismo, utilizando de sofisticados sistemas de vigilância.

Neste contexto, este trabalho disponibiliza uma arquitetura para definir um sistema autônomo de vigilância externa que permite ao usuário definir uma base e rotas que são executadas por um drone. O sistema será disponibilizado numa interface web e o usuário pode informar a sua base e rota através de pontos que deverão ser percorridos, informando no mapa que é carregado previamente. A rota será percorrida pelo drone de forma autônoma. Durante o percurso pré-definido, o drone montará um relatório com imagens da rota e o enviará ao sistema.

### 1.1 OBJETIVOS

Este trabalho tem como objetivo propor uma arquitetura para um sistema de vigilância utilizando drone.

Os objetivos específicos são:

- a) possuir cadastro de base e rotas para o drone;
- b) oferecer recursos para percorrer rotas de forma autônoma;
- c) disponibilizar dados registrados na rota.

## 1.2 ESTRUTURA

O trabalho está estruturado em quatro capítulos principais. No primeiro capítulo a introdução, objetivo geral seguido dos específicos. No segundo capítulo a fundamentação teórica contendo a introdução com as tecnologias e bibliotecas utilizadas. O terceiro capítulo apresenta todo desenvolvimento da arquitetura em detalhes, apresentando os métodos e funções implementados a partir das bibliotecas e frameworks, os requisitos funcionais e não funcionais, diagramas de casos de uso e de atividades.

Na seção da operacionalidade do terceiro capítulo é apresentado em formato de guia com o passo a passo, desde a conexão entre arquitetura e AR.Drone 2.0 até o cadastramento de missões para vigilância aéreas. Ainda no capítulo três é apresentado os resultados e discussões, onde contêm os testes realizados e dados de usabilidade coletados em campo.

O capítulo quatro apresenta as conclusões referente ao trabalho desenvolvido, desde sua pesquisa inicial, levando em consideração os problemas e aprendizado encontrados durante o desenvolvimento da arquitetura.

## 2 FUNDAMENTAÇÃO TEÓRICA

As seções desse capítulo estão fundamentadas na seguinte ordem. A seção 2.1 descreve as especificações técnicas e de hardware do AR.Drone 2.0, adaptações necessárias, e sobre seu respectivo módulo externo que habilita o uso de GPS. A seção 2.2 descreve as estruturas da biblioteca NPM do node-ar-drone, assim como a seção 2.3 descreve a estrutura da biblioteca NPM do ardrone-autonomy. A seção 2.4 descreve o framework NPM Geolib para cálculos geográficos. A seção 2.5 é destinada a integração com API do Google Maps usando o framework NPM leaflet-google, e por fim a seção 2.6 descreve os trabalhos correlatos.

### 2.1 ESPECIFICAÇÕES TÉCNICAS E DE HARDWARE AR.DRONE 2.0

O AR.Drone 2.0 consiste em um quadricóptero no modelo da Figura 1, desenvolvido pela empresa Parrot. O controle deste equipamento pode ser realizado por meio de computadores ou *smartphones* através de uma conexão WI-FI em modo AD-HOC aberta, e operando por bandas b/g/n (RAHN, 2016, p. 18). Se comparado a outros equipamentos semelhantes, o AR.Drone 2.0 oferece facilidade de compra no mercado, pois tem um custo reduzido (SANTANA; BRANDÃO; SARCINELLI, 2016, p. 2).

Figura 1 - AR.Drone 2.0 Parrot



Fonte: Toman (p. 4, 2017).

Conforme Santana, Brandão e Sarcinelli (2016, p. 2) o AR.Drone 2.0 vem de fábrica com acelerômetros, giroscópios, magnetômetros, duas câmeras de vídeo e um computador de bordo que gerencia estes sensores e a rede de comunicação sem fio do veículo.

O hardware do AR.Drone 2.0 conta com especificações técnicas acima da média encontradas em drones disponíveis no mercado, sendo elas:

- a) gravação e *streaming* de vídeo em HD;
- b) câmera HD. 720p 30fps (Frames Per Second);
- c) lente grande angular: 92 ° na diagonal;
- d) perfil base de codificação H264;
- e) *streaming* de baixa latência;
- f) armazenamento de vídeo em tempo real com o dispositivo conectado na entrada *Universal Serial Bus* (USB);
- g) foto Joint Photographics Experts Group (JPEG);
- h) armazenamento de vídeo em tempo real com Wi-Fi diretamente no seu dispositivo remoto.

Já a estrutura corporal do AR.Drone 2.0 conta com tecnologia mais robusta e resistente, pensado no uso geral tanto em campos de pesquisas quanto em projetos mais ousados. Essa estrutura conta com os seguintes diferenciais:

- a) tubos de fibra de carbono: peso total de 380g (gramas) com casco externo, 420g com casco interno;
- b) peças plásticas de nylon com alto teor de fibra de 30% de carga;
- c) espuma para isolar o centro inercial da vibração dos motores;
- d) casco de fibra de isopor injetado por um molde de metal escovado;
- e) nano-repelente a líquidos em sensores de ultrassom;
- f) totalmente reparável: todas as peças e instruções para reparo disponíveis na internet.

A tecnologia de bordo oferece controle de precisão e recursos de estabilização automática. Entre estes recursos se tem:

- a) processador Advanced Risc Machine (ARM) Cortex A8 de 1 GigaHertz (GHZ) e 32 Binary digiT (BIT) com Digital Signal Processor (DSP) de vídeo de 800 MegaHertz (MHZ) TMS320DMC64x;
- b) Linux 2.6.32;
- c) Random Access Memory (RAM) Double Data Rate (DDR2) de 1 GigaBIT (GBIT) a 200 MHz;
- d) USB 2.0 de alta velocidade para extensões;
- e) Wi-Fi b, g, n;
- f) giroscópio de 3 eixos 2000 ° / segundo de precisão;
- g) acelerômetro de 3 eixos + -50mg (Miligrama) de precisão;
- h) magnetômetro de 3 eixos 6 ° de precisão;

- i) sensor de pressão +/- 10 Precision and Accuracy (PA) de precisão;
- j) sensores de ultrassom para medição da altitude do solo;
- k) câmera Quarter Video Graphics Array (QVGA) vertical de 60 FPS para medição da velocidade do solo.

Os motores também chamados de *brushless* são essenciais para que o drone consiga decolar e executar o voo de forma a manter-se estável pairando ao ar. É importante também levar em consideração o total de potência máxima e mínima suportados ao enviar pulsos elétricos de forma variável.

Os motores suportam variações da quantidade de energia enviadas, com isso há total controle da placa principal chamada de *board*. A placa principal envia então pulsos controlados aos motores, fazendo com que o drone consigo flutuar e locomover-se ao ar, aumentando e diminuindo a quantidade de energia enviada.

Algumas das características técnicas que envolvem o controle dos motores, além de suas próprias características:

- a) micro rolamento de esferas;
- b) motores *brushless* sem escova. 14.5W (*Watt*) 28.500 Rotação Por Minuto (RPM);
- c) engrenagens *Nylatron* de baixo nível de ruído para redutor de hélice 1 / 8,75;
- d) eixo da hélice em aço temperado;
- e) rolamento em bronze auto lubrificante;
- f) arrasto específico de alta propulsão para grande manobrabilidade;
- g) Central Processing Unit (CPU) de 8 Microprocessor without Interlocked Pipeline Stages (MIPS) Alf-egil bogen e VegaRd wollan (AVR) por controlador de motor;
- h) parada de emergência controlada por software;
- i) controlador do motor totalmente reprogramável;
- j) controlador eletrônico do motor resistente à água.

Para voar autonomamente para uma determinada coordenada geográfica, é necessário um dispositivo receptor Global Navigation Satellite System (GNSS) termo genérico padrão para sistemas de navegação por satélite, que fornecem posicionamento geográfico espacial autônomo com cobertura global. Este termo inclui o Global Positioning System (GPS), GLobal (Orbiting) NAVigation Satellite System (GLONASS), Galileo, Beidou e outros sistemas regionais.

O módulo original da Parrot visto na Figura 2 é conectado na USB da placa principal. Com isso o drone passa a receber e administrar coordenadas geográficas, tal como latitude e longitude.

Figura 2 – Módulo de GPS Ar.Drone



Fonte: Toman (2017).

## 2.2 BIBLIOTECA NPM NODE-AR-DRONE

O `node-ar-drone` é uma biblioteca em Node.js desenvolvida com base no Software Development Kit (SDK) 2.0 oficial da Parrot, cujo objetivo principal destina-se exclusivamente para controlar o AR.Drone através de um computador conectado na rede WI-FI gerada pelo aparelho. E assim permitir enviar comandos escritos em JavaScript para serem interpretados pelo framework Node.js que envia e recebe pacotes através de conexão User Datagram Protocol (UDP) (GEISENDÖRFER, 2014).

A biblioteca `node-ar-drone` conta com API de alto nível que abstrai as funcionalidades disponíveis no SDK (PISKORSKI, 2012), contendo as seguintes características:

- módulo cliente que disponibiliza uma API de alto nível suportando a grande maioria das funções disponíveis para controlar o drone, de forma um tanto simples na usabilidade e interpretação lógica;
- possibilidade de criar um pequeno programa autônomo que realiza comandos pré-programados via codificação JavaScript;
- disponibilidade de usar os sensores disponíveis no aparelho, de forma fácil bastando chamar métodos específicos, tal como obter altitude;
- possibilidade de acessar a câmera vertical e horizontal, salvando fotos e vídeos ou até transmitir em tempo real usando HyperText Transfer Protocol (HTTP) web servidor;

- e) acesso direto para enviar comandos UDP de baixo nível, através de métodos basta enviar os dados usando a porta 5556.

A biblioteca `node-ar-drone` conta com uma API ampla em quantidades de métodos disponíveis, e suas principais características são:

- a) `arDrone.createClient(opções) :`
  - retorna um novo objeto cliente, opções incluem:
    - `ip`: o *Internet Protocol* (IP) do drone. O padrão é '192.168.1.1',
    - `frameRate`: a taxa de quadros do `PngEncoder`. O padrão é 5,
    - `imageSize`: o tamanho da imagem produzido por `PngEncoder`. O padrão é nulo;
- b) `client.createREPL() :`
  - inicia uma interface interativa com todos os métodos cliente disponíveis no escopo ativo. Além disso, o cliente resolve a própria instância;
- c) `client.getPngStream() :`
  - retorna um objeto `PngEncoder` que emite *buffers* individuais de imagem Portable Network Graphics (PNG) como eventos de 'dados'. Várias chamadas para esse método retornam o mesmo objeto. O ciclo de vida da conexão (por exemplo, reconectar com erro) é gerenciado pelo cliente;
- d) `client.getVideoStream() :`
  - retorna um objeto `TcpVideoStream` que emite pacotes de Transmission Control Protocol (TCP) como eventos de dados. As chamadas para esse método retornam o mesmo objeto. O ciclo de vida da conexão (por exemplo, reconectar com erro) é gerenciado pelo cliente;
- e) `client.takeoff(retorno de chamada) :`
  - define o estado interno do voo como *true*, o retorno de chamada é invocado após o drone relatar que está pairando;
- f) `client.land(retorno de chamada) :`
  - define o estado interno do voo como *false*, o retorno de chamada é invocado após o drone relatar que chegou;
- g) `client.up(velocidade) / client.down(velocidade) :`
  - faz o drone ganhar ou reduzir a altitude, velocidade pode ser um valor de 0 a 1;
- h) `client.clockwise(velocidade) / client.counterClockwise(velocidade) :`
  - faz com que o drone gire no sentido horário e anti-horário, a velocidade pode ser um valor de 0 a 1;

- i) `client.front(velocidade) / client.back(velocidade) :`
  - controla com movimentos para frente e para trás de forma horizontal usando a câmera como ponto de referência, a velocidade pode ser um valor de 0 a 1;
- j) `client.left(velocidade) / client.right(velocidade) :`
  - controla com movimentos para direita e esquerda de forma horizontal usando a câmera como ponto de referência. A velocidade pode ser um valor de 0 a 1;
- k) `client.stop() :`
  - define todos os comandos de movimento do drone para 0, tornando-o efetivamente pairado no lugar;
- l) `client.calibrate(dispositivo) :`
  - pede ao drone para calibrar um dispositivo. Embora o *firmware* AR.Drone seja compatível apenas com um dispositivo que possa ser calibrado. Esta função também inclui FTRIM;
- m) magnetômetro:
  - o magnetômetro só pode ser calibrado enquanto o drone está voando, e a rotina de calibração faz com que o drone gire no lugar a 360 graus;
- n) FTRIM:
  - o FTRIM redefine essencialmente o *Yaw*, *Pitch* e *Roll* para 0. Tenha muito cuidado ao usar esta função e apenas calibre enquanto estiver em uma superfície plana. Nunca usar enquanto estiver voando;
- o) `client.config(chave, valor, retorno de chamada) :`
  - envia um comando de configuração para o drone.

## 2.3 BIBLIOTECA ARDRONE-AUTONOMY

A biblioteca `ardrone_autonomy` é uma API de alto nível para o quadricóptero Parrot AR.Drone 1.0 e 2.0. Esta API foi construída com base a versão oficial do AR.Drone SDK 2.0.1. `ardrone_autonomy` é uma bifurcação da biblioteca anterior `node-ar-drone`. A biblioteca `ardrone-autonomy` foi desenvolvida por Laurent Eschenauer (ESCHENAUER, 2014).

Na biblioteca `ardrone-autonomy` a frequência de atualização do drone: a frequência de atualização da transmissão de dados do drone depende do parâmetro `navdata_demo`. Quando definido como 1, a frequência de transmissão é definida em 15Hz (Hertz), caso contrário, a frequência de transmissão é definida em 200Hz. (`navdata_demo` é um parâmetro numérico que não é booleano, portanto, usa-se (Verdadeiro / Falso) para definir / desabilitar).



Já a frequência de atualização do driver pode operar em dois modos: tempo real ou taxa fixa. Quando o parâmetro `realtime_navdata` é definido como *true*, o driver publica qualquer informação recebida instantaneamente. Quando definido como *false*, o driver armazena em cache os dados recebidos primeiro e depois os envia a uma taxa fixa. Essa taxa é configurada através do parâmetro *looprate*. A configuração padrão é: `realtime_navdata = false` (ESCHENAUER, 2014).

A biblioteca `ardrone_autonomy` possui um módulo que expõe uma API de alto nível. Os métodos disponibilizados são:

- a) `mission.createMission()`:
  - cria missão, disponibilizando acesso a todos métodos da API;
- b) `mission.log(caminho)`:
  - registra os dados da missão, no formato Comma Separated Values (CSV), no arquivo fornecido. Torna realmente útil depurar / plotar o estado e o comportamento do controlador;
- c) `mission.run(retorno de chamada)`:
  - executa a missão. Retorno de chamada da `function(err, resultado)` e será acionado em caso de erro ou no final da missão;
- d) `mission.takeoff()`:
  - adiciona uma etapa de decolagem à missão.
- e) `mission.forward / backward / left / right / up / down (distância)`:
  - adiciona um passo de movimento à missão. O drone se moverá na direção especificada pela distância (em metros) antes de prosseguir para o próximo passo. O drone também tentará manter todos os outros graus de liberdade;
- f) `mission.altitude(altura)`:
  - adiciona um passo de altitude à missão. Subirá até a altura especificada antes de prosseguir para o próximo passo;
- g) `mission.cw/ccw(angle)`:
  - adiciona uma etapa de rotação à missão. Girará pelo ângulo fornecido (em graus) antes de prosseguir para o próximo passo;
- h) `mission.hover(atraso)`:
  - adiciona uma etapa flutuante à missão. Irá pairar no local pelo atraso especificado em MilisSegundos (MS) antes de prosseguir para a próxima etapa;
- i) `mission.wait(atraso)`:
  - adiciona um passo de espera para a missão. Esperará o atraso especificado (em

MS) antes de prosseguir para a próxima etapa;

j) `mission.go(posição)` :

- adiciona um passo de movimento à missão. Irá para a posição especificada antes de prosseguir para a próxima etapa. A posição é um objetivo do controlador, como `{x: 0, y: 0, z: 1, yaw: 90}`;

k) `mission.task(função (retorno de chamada) :`

- adiciona uma etapa da tarefa à missão. Irá executar a função fornecida antes de prosseguir para a próxima etapa. Um argumento de retorno de chamada é passado para a função, que deve ser chamado quando a tarefa estiver concluída;

l) `mission.taskSync(função) :`

- adiciona uma etapa da tarefa à missão. Irá executar a função fornecida antes de prosseguir para a próxima etapa;

m) `mission.zero()` :

- adiciona um passo de *reset* à missão. Isso definirá a posição / orientação atual como o estado base do filtro de Kalman (ou seja, `{x: 0, y: 0, yaw: 0}`). Se não estiver usando uma etiqueta como sua posição base, convém usar `zero()` após a decolagem.

## 2.4 BIBLIOTECA GEOLIB

Biblioteca para fornecer operações geoespaciais básicas como cálculo de distância, conversão de coordenadas decimais em sexagesimal e vice-versa. Atualmente esta biblioteca é 2D, o que significa que a altitude / elevação ainda não é suportada por nenhuma de suas funções.

A biblioteca é escrita em TypeScript. Não é necessário conhecer o TypeScript para usar o Geolib, mas as definições de tipo fornecem informações valiosas sobre o uso geral, parâmetros de entrada.

Valores e formatos suportados, todos os métodos que estão trabalhando com coordenadas aceitam um objeto com uma propriedade `lat` / latitude e `lon` / lng / longitude ou uma matriz de coordenadas `GEographic LOcation` (GEO) JavaScript Object Notation (JSON), como: `[longitude, latitude]`. Todos os valores podem estar no formato decimal (53.471) ou sexagesimal (53 ° 21' 16 "). Os valores da distância são sempre flutuantes e representam a distância em metros (BIEH, 2020).

As principais funções disponíveis na API da biblioteca Geolib são:

a) `getDistance(início, fim, precisão = 1) :`

- calcula a distância entre duas coordenadas geográficas. Esta função leva até 3

argumentos. Os dois primeiros argumentos devem ser Geolib *input coordinates* válidos (por exemplo, {latitude: 52.518611, longitude: 13.408056});

- as coordenadas podem estar no formato sexagesimal ou decimal. O terceiro argumento é precisão (em metros). Por padrão, a precisão é de 1 metro. Se precisar de um resultado mais preciso, poderá defini-lo como valor mais baixo, por exemplo 0,01 para precisão de centímetros;
- poderá configurá-lo mais alto para que o resultado seja arredondado para o próximo valor divisível pela precisão escolhida (por exemplo, 25428 com uma precisão de 100 se torna 25400);

b) `getPreciseDistance (start, end [int precision]):`

- calcula a distância entre duas coordenadas geográficas. Esse método é mais preciso que o `getDistance`, especialmente para longas distâncias, mas também é mais lento. Ele está usando a fórmula inversa Vincenty para elipsoides. Ele usa os mesmos argumentos (até 3) que `getDistance (início, fim, precisão = 1);`

c) `getGreatCircleBearing(origin, destination):`

- cálculo que obtém o ângulo horizontal entre duas coordenadas considerando o círculo angular norte a sul entre dois pontos conectados na terra, esse conceito chama-se grande círculo.

## 2.5 API GOOGLE MAPS

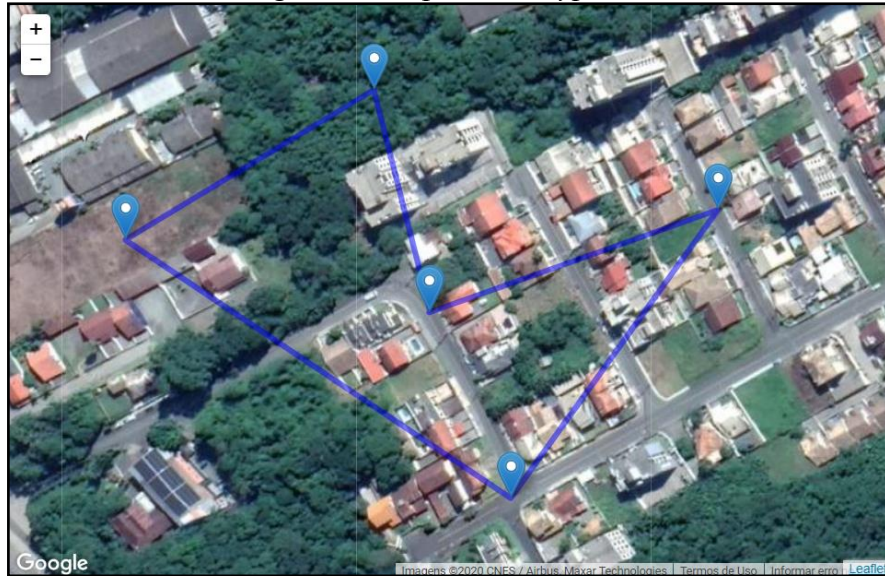
A arquitetura que foi proposta visava a necessidade de abrir um mapa para visualização e escolha da rota a ser percorrida. Desta forma foi realizado testes de implementação com APIs disponíveis no formato *open-source*, porém como validado nos testes, foi identificado na API do Google Maps as funções necessárias com tempo de resposta satisfatório.

O formato de visualização da abertura do mapa é por satélite, para abrir o mapa pela API do Google Maps é requerida uma conexão com a internet, o mapa deverá abrir inicialmente num perímetro de aproximadamente 5km<sup>2</sup> (GOOGLE, 2020).

Para criar as marcações de *waypoints* e traços da rota a ser percorrida como pode ser visto na Figura 3 foi integrado a principal biblioteca JavaScript de código aberto para mapas interativos compatíveis com dispositivos móveis e navegadores. Com apenas 38 Kilobyte (KB) de JavaScript (JS), a biblioteca `leaflet` possui todos os recursos de mapeamento de forma off-line. Foi projetada com simplicidade, desempenho e com a usabilidade em mente. A biblioteca `leaflet` funciona de maneira eficiente em todas as principais plataformas móveis e desktop,

pode ser estendida com muitos plugins, além de possui uma API documentada de código-fonte legível (LIEDMAN, 2020).

Figura 3 - Mapa com *waypoints*



Fonte: Elaborado pelo autor (2020).

Os principais métodos da *leaflet* para o funcionamento adequado, garantindo uma usabilidade amigável e contendo os recursos necessários são:

- a) `L.map('map').setView([latitude, longitude], 20):`
  - esse método gera um mapa com a visualização inicial através das coordenadas de latitude e longitude passadas como parâmetro, além de atribuir o zoom em 20 metros de altitude;
- b) `L.Google('SATELLITE').addLayer(googleLayer):`
  - esse método adiciona através da API do Google Maps o formato de abertura do mapa com visualização por satélite;
- c) `L.marker([latitude, longitude], { ícone: objetoÍcone }).addTo(map):`
  - esse método adiciona ao mapa já aberto o ícone desejado, na posição da latitude e longitude passadas no parâmetro.

## 2.6 TRABALHOS CORRELATOS

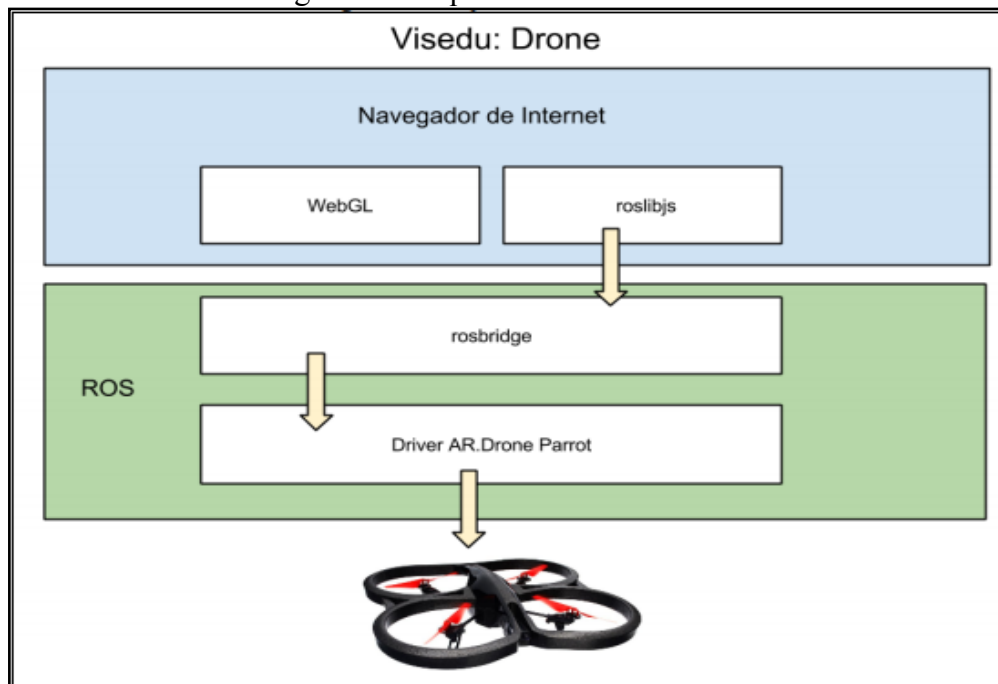
A seguir serão apresentados três trabalhos correlatos. A seção 2.6.1 aborda o trabalho de Vanz (2015) que disponibiliza um protótipo de módulo de integração com Robot Operation System (ROS). Na seção 2.6.2 é apresentado o trabalho de Rocha (2016) que consiste no sistema móvel multiplataforma para navegação em rotas internas. Para finalizar na seção 2.6.3 será apresentado o trabalho de Barrow (2014) que procura integrar a navegação e busca autônoma em ambiente interno usando um drone.

### 2.6.1 VISEDU-DRONE: MÓDULO DE INTEGRAÇÃO COM ROS

Vanz (2015) tem como objetivo criar um simulador de drone integrado com o *framework* para robótica ROS. O simulador estende o VisEdu e foi desenvolvido na linguagem JavaScript utilizando a biblioteca Three.js para abstrair o WebGL e facilitar a manipulação do ambiente virtual. Esse simulador possibilita alterar o comportamento do drone físico simultaneamente, possibilitando ao usuário simular na prática os eventos iguais da realidade. Para controlar o drone físico foi utilizado o driver `ardrone_autonomy`, esse driver efetua a comunicação entre o ROS e o drone. Para disponibilizar a execução através de um navegador web foi utilizado o WebSocket da Rosbridge. Por fim foi constatado que no sistema de navegação implementado, ocorre uma deficiência no tempo de execução ao percorrer o espaço que fora atribuído (VANZ, 2015).

Na Figura 4 é possível observar a arquitetura da aplicação, dividida em duas camadas sendo a primeira referente ao que é executado no navegador do usuário, consistindo na interface gráfica onde o usuário interage e visualiza a cena, executando as animações e controlando o drone. Na segunda camada ocorre a comunicação com o AR.Drone Parrot, a camada destacada em verde corresponde à aplicações que rodam no ecossistemas do ROS, sendo subdividida em dois pacotes principais, Rosbridge e *driver* para AR.Drone (VANZ, 2015).

Figura 4 – Arquitetura VisEdu-Drone



Fonte: Vanz (2015).

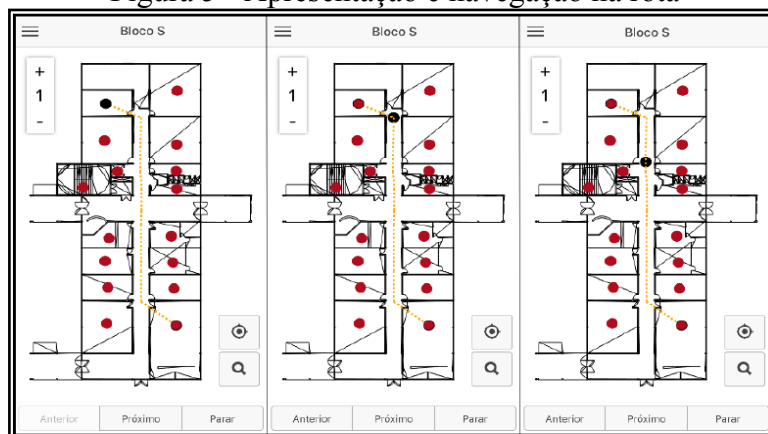
### 2.6.2 FURB MOBILE: SISTEMA MÓVEL MULTIPLATAFORMA PARA NAVEGAÇÃO EM ROTAS INTERNAS

O trabalho de Rocha (2016) trouxe o desenvolvimento de um aplicativo multiplataforma para auxiliar na locomoção dos visitantes pelo campus da Fundação da Universidade Regional de Blumenau (FURB) em dias do evento Interação FURB. Este aplicativo permite buscar e localizar locais específicos, como laboratórios e salas, por exemplo e foi construído com o framework PhoneGap, utilizando de tecnologias web, como HyperText Markup Language (HTML), Cascading Style Sheets (CSS) e JavaScript, com o auxílio da biblioteca AngularJS (ROCHA, 2016).

A fim de dar suporte as funcionalidades do aplicativo, foi construída uma aplicação servidora que dispõe informações pela web através de uma interface RESTful, além de uma ferramenta para a administração dos mapas e de possíveis rotas pelas partes internas dos blocos do campus. Para a construção de um ambiente gráfico para a apresentação e edição dos mapas, além da apresentação de rotas nestes mapas, foi utilizado a biblioteca Three.js, com isso permitiu a apresentação e importação das plantas baixas dos blocos em arquivos em formato *Object file wavefront* (OBJ) (ROCHA, 2016).

Na Figura 5 há apresentação de uma rota de menor custo entre os pontos de origem e destino que foram selecionadas, sendo esta, apresentada ao usuário do aplicativo em forma de uma linha pontilhada. Na parte inferior da tela do aplicativo, estão as ações de navegação, no qual o usuário poderá navegar na rota, através das ações posterior e anterior. Caso o usuário queira parar a navegação e voltar ao estado de navegação do mapa deve utilizar a ação parar. Se a opção de rota do usuário tenha como destino outro pavimento ou bloco, este será apresentado também em linha pontilhada (ROCHA, 2016).

Figura 5 - Apresentação e navegação na rota



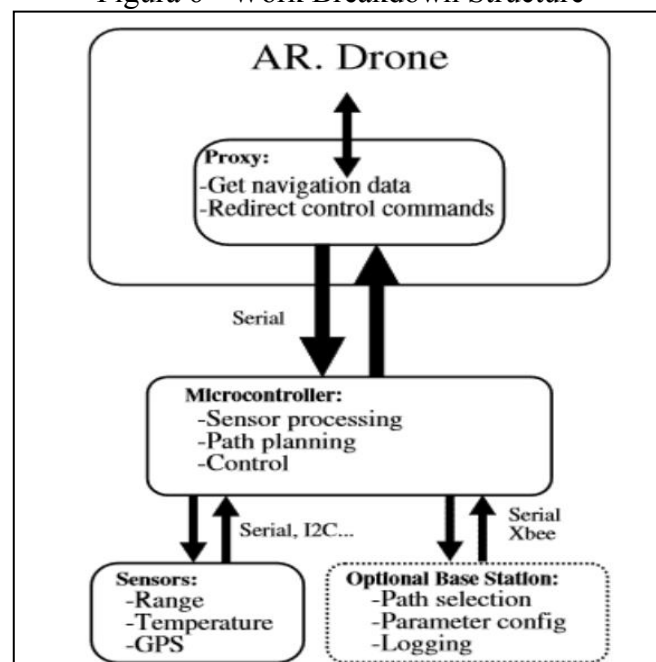
Fonte: Rocha (2016).

### 2.6.3 AUTONOMOUS NAVIGATION AND SEARCH IN AN INDOOR ENVIRONMENT USING AN AR.DRONE

A dissertação de mestrado de Barrow (2014) tem como objetivo entregar um drone autônomo que consegue navegar, buscar e identificar objetos em lugares desconhecidos, excluindo a necessidade de uma pessoa ficar controlando-o. Possui algoritmos de processamento visual que podem ser usados com AR.Drone para identificar objetos e cores, além de processar a rota em tempo real mantendo-se no ar mesmo nos lugares dos quais ainda não foram processados. Ao final de cada processamento o experimento entrega os resultados com possíveis soluções para navegação e busca autônoma.

Na Figura 6 é apresentada a arquitetura do sistema usada para controlar o drone. O bloco ao meio mostra o microcontrolador adaptado ao drone, esse micro controlador processa os dados dos sensores e traça um caminho, esse processo ocorre antes dos comandos de voo enviados ao drone. Também apresenta a opção de escolher uma base para o drone realizar o log dos dados e descarregá-los através de uma conexão do servidor.

Figura 6 - Work Breakdown Structure



Fonte: Barrow (2014).

### 3 DESENVOLVIMENTO DA ARQUITETURA

Este capítulo apresenta toda estrutura de desenvolvimento da arquitetura, na primeira seção os requisitos funcionais e não funcionais. Na segunda seção os casos de uso e diagramas de atividade usando Unified Modeling Language (UML).

A terceira seção detalha toda estrutura usada na implementação da arquitetura, algoritmos aplicados a cálculos de rotas aéreas, arquitetura de conversação entre cliente e servidores, técnicas de transporte de pacotes e mensagens através de redes TCP e UDP e métodos de interpretação da conexão com o hardware do drone.

Por fim na quarta e última seção os resultados e discussões obtidos durante todo ciclo da pesquisa, projeto e trabalho.

#### 3.1 REQUISITOS PRINCIPAIS

Os seguintes requisitos fazem parte da arquitetura:

- a) disponibilizar um sistema web para cadastro de rotas (RF01);
- b) a arquitetura deverá permitir o cadastro de uma base para cada rota (RF02);
- c) a arquitetura deverá gerar um relatório para cada rota a partir das informações obtidas pelo drone (RF03);
- d) o drone deverá possuir um GPS e gravar as coordenadas da rota (RF04);
- e) disponibilizar recurso de monitoramento da porcentagem da carga da bateria do drone (RNF01);
- f) a arquitetura deverá ser desenvolvida em Node.js (RNF02);
- g) a arquitetura deverá possuir integração com a biblioteca NPM Geolib (RNF03);
- h) a arquitetura deverá utilizar cálculos matemáticos para melhorar sua localização (RNF04).

#### 3.2 ESPECIFICAÇÃO

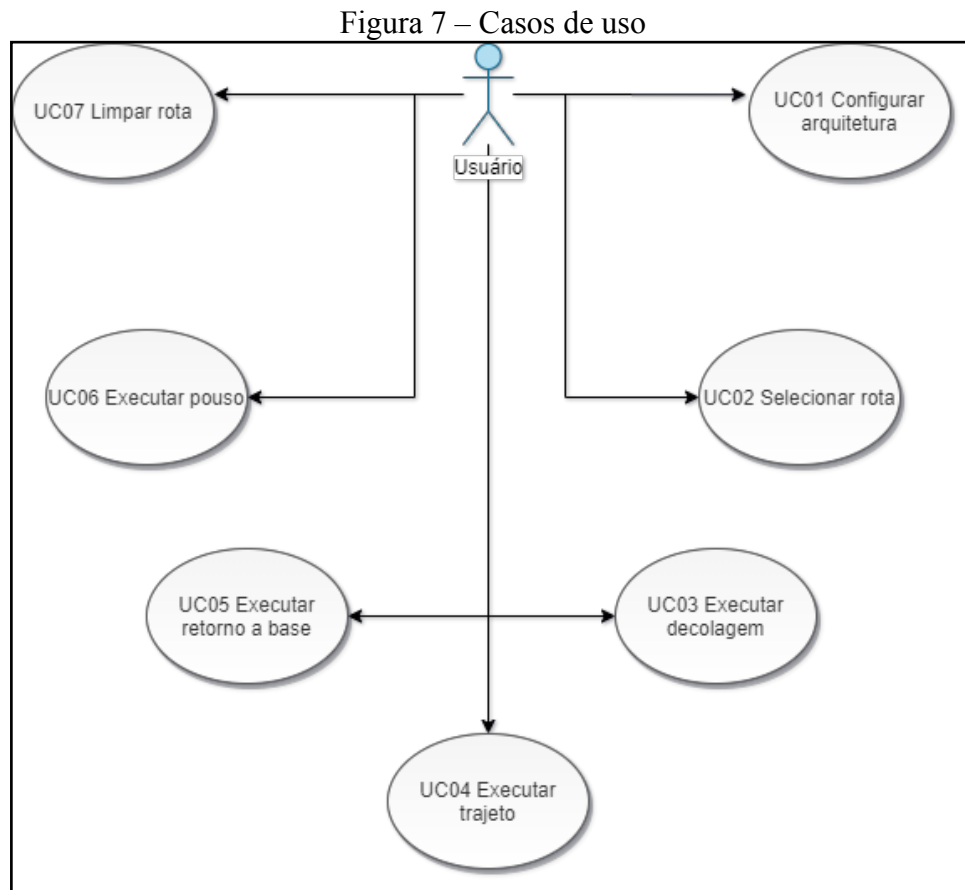
O trabalho segue com a especificação no formato UML com os casos de uso exemplificados através de diagramas de atividades, detalhado os cenários da execução. O software draw.io para desktop foi utilizado na elaboração desses diagramas.

##### 3.2.1 Casos de uso

Os casos de uso a seguir têm como objetivo exemplificar funcionalidades da arquitetura em alto nível, o ator `usuário` representará a entidade de interação que utilizará a interface de pilotagem do drone, cada ação está associada ao seu respectivo ator. Na Figura 7 é apresentado



o diagrama de User Cases (UC) com os detalhes de ações que o ator poderá executar através da arquitetura disponibilizada, as etapas serão executadas e controladas através de uma interface web gerada pela arquitetura, respeitando os conceitos atuais de User Experience (UX) e User Interface (UI).



Fonte: Elaborado pelo autor (2020).

A seguir será apresentada as descrições de cada UC seguidos de seu respectivo cenário de execução detalhado utilizando diagramas de atividades.

#### 3.2.1.1 Caso de uso: Configurar arquitetura

Inicialmente antes de realizar qualquer missão de voo é preciso configurar a arquitetura através da interface, as configurações disponíveis são:

- a) marcar se o drone deve virar sua câmera frontal em direção aos pontos selecionados como detalhado no Quadro 2;
- b) inserir altitude inicial que o drone deverá subir ao realizar a etapa como descrito no Quadro 3;
- c) marcar se deverá realizar a calibração do magnetômetro antes de iniciar o voo.

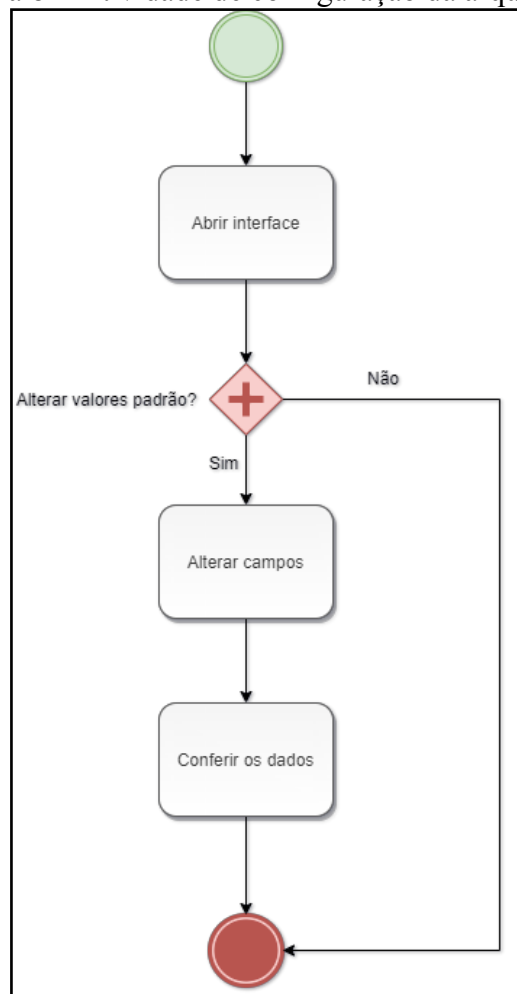
O Quadro 1 apresenta as tarefas e configurações relacionadas, logo após a Figura 8 exemplifica-as através do diagrama de atividades.

Quadro 1 - Caso de uso configurar arquitetura

UC01. Configurar arquitetura	
Descrição	Permitir configurar os parâmetros iniciais da arquitetura.
Requisitos atendidos	RF02, RF01.
Pré-condição	Conectado ao drone.
Cenário principal	1) A interface exibe os campos editáveis; 2) O usuário altera os campos; 3) A arquitetura salva os valores de cada campo; 4) O usuário visualiza conferindo os dados salvos na interface.
Fluxo alternativo	No passo 2, o usuário poderá ignorar os campos editáveis e selecionáveis. 5) A arquitetura irá atribuir os valores padrões.
Pós-condição	A arquitetura atualiza os valores padrões.

Fonte: Elaborado pelo autor (2020).

Figura 8 – Atividade de configuração da arquitetura



Fonte: Elaborado pelo autor (2020).

### 3.2.1.2 Caso de uso: Selecionar rota

É possível selecionar a rota logo ao iniciar a interface e após o mapa carregar, sendo essa uma das tarefas de importantes ao propósito da arquitetura, é possível escolher diversas rotas selecionando-as no mapa através de um click em cima do local desejado

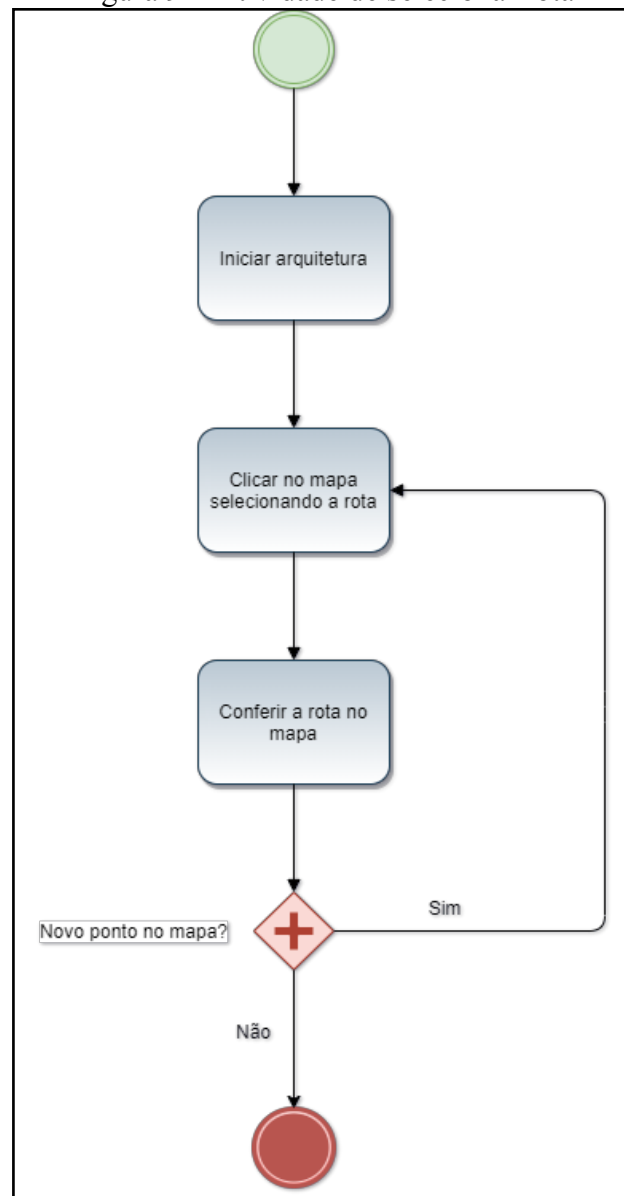
Nesse contexto define-se rota como percurso entre dois ou mais pontos que serão sistematicamente interligados, dos quais o drone deverá percorrer de forma autônoma ao iniciar a missão. Sendo as rotas selecionáveis a critério do usuário no momento da pilotagem. O Quadro 2 apresenta os detalhes da tarefa que define as rotas a serem percorridas, logo após na Figura 9 os detalhes exemplificados através do diagrama de atividades.

Quadro 2 – Caso de uso selecionar rota

UC02. Selecionar rota	
Descrição	Permitir através da interface que o usuário defina uma rota ao seu critério.
Requisitos atendidos	RF01, RF02.
Pré-condição	Conectado ao drone.
Cenário principal	1) O usuário clica nos pontos desejados para definir a rota; 2) A arquitetura salva as coordenadas selecionadas; 3) O usuário confere visualizando a rota no mapa.
Fluxo alternativo	No passo 3 o usuário poderá clicar em mais pontos no mapa. 4) A arquitetura salva as novas coordenadas selecionadas.
Pós-condição	A arquitetura grava a rota que deverá ser percorrida.

Fonte: Elaborado pelo autor (2020).

Figura 9 – Atividade de selecionar rota



Fonte: Elaborado pelo autor (2020).

### 3.2.1.3 Caso de uso: Executar decolagem

Através dessa etapa o usuário define o ponto inicial da missão, após ter realizado a etapa anterior definindo a rota no mapa, ao executar dispara ação de decolagem do drone.

Ao levantar voo a arquitetura ajusta o drone de forma autônoma para que permaneça pairando ao ar com máxima estabilidade possível, sendo sua altitude inicial configurada em 1.5 metros. Caso o usuário tenha configurado para realizar a calibração do magnetômetro na etapa exemplificada no Quadro 1 a arquitetura enviara um comando para que o drone suba mais 2 metros, após faça um giro de 360 graus com o máximo de estabilidade possível.

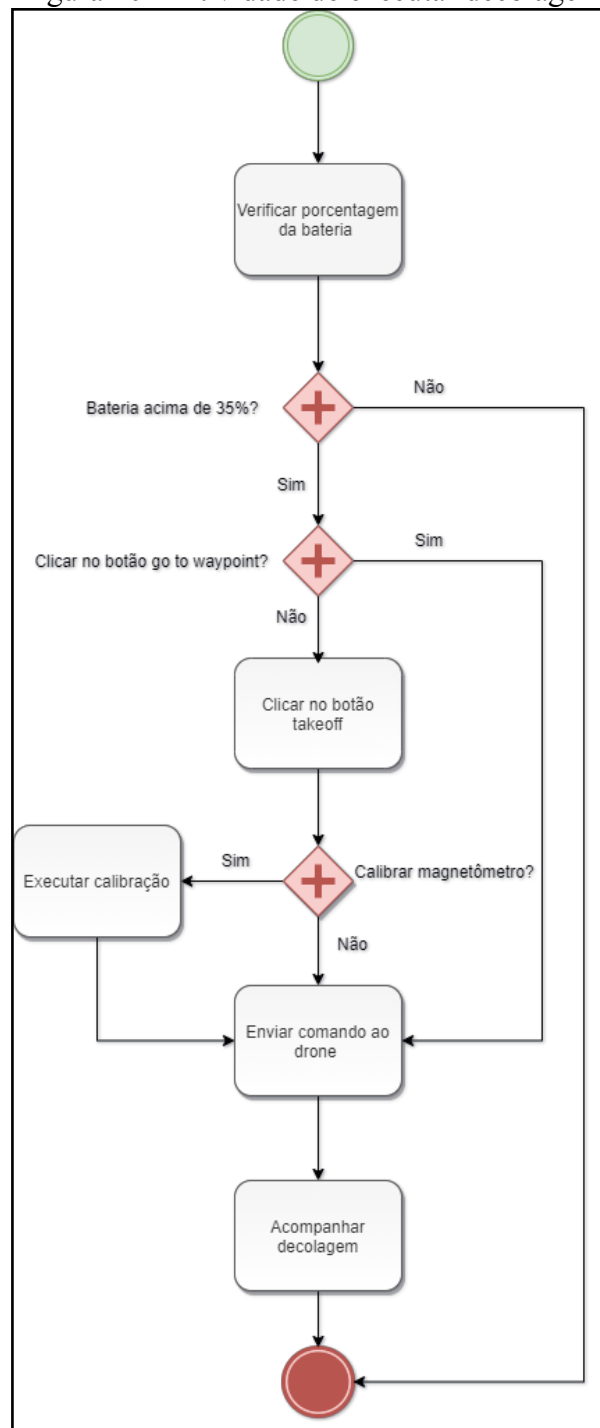
O Quadro 3 apresenta os detalhes da tarefa que define a execução de decolagem, a Figura 10 mostra os detalhes exemplificados através do diagrama de atividades.

Quadro 3 – Caso de uso executar decolagem

UC03. Executar decolagem	
Descrição	Permitir através da arquitetura que o usuário execute a decolagem do drone.
Requisitos atendidos	RF01, RF02, RNF01, RNF02.
Pré-condição	Porcentagem da bateria acima de 35%.
Cenário principal	<ol style="list-style-type: none"> <li>1) A arquitetura valida a porcentagem da bateria do drone;</li> <li>2) O usuário clica no botão de <i>takeoff</i>;</li> <li>3) A arquitetura envia o comando para calibrar o magnetômetro, caso configurado anteriormente;</li> <li>4) A arquitetura envia o comando ao drone para executar a decolagem;</li> <li>5) O usuário acompanha a decolagem.</li> </ol>
Fluxo alternativo	<p>No passo 2 o usuário poderá clicar diretamente no botão <i>go to waypoint</i>.</p> <ol style="list-style-type: none"> <li>6) A arquitetura envia o comando ao drone para executar o fluxo como detalhado na Figura 11 – Atividade de executar trajeto.</li> </ol>
Pós-condição	A arquitetura executa a tarefa salvando os valores dos parâmetros.

Fonte: Elaborado pelo autor (2020).

Figura 10 – Atividade de executar decolagem



Fonte: Elaborado pelo autor (2020).

#### 3.2.1.4 Caso de uso: Executar trajeto

Etapa principal que iniciará o voo coordenado por *waypoints* de ponto a ponto, selecionados pelo usuário e gravados pela arquitetura como visto na Figura 9.

Ao executar essa ação a arquitetura fará os cálculos e ajustes ao direcionar o drone, coordenando-o durante todo trajeto, além de verificar os parâmetros preenchidos pelo usuário como visto na Figura 8, através desses valores o drone voa na altitude estipulada, girando em

sentido horário ou anti-horário, apontando a câmera frontal ao primeiro ponto da rota selecionada no mapa.

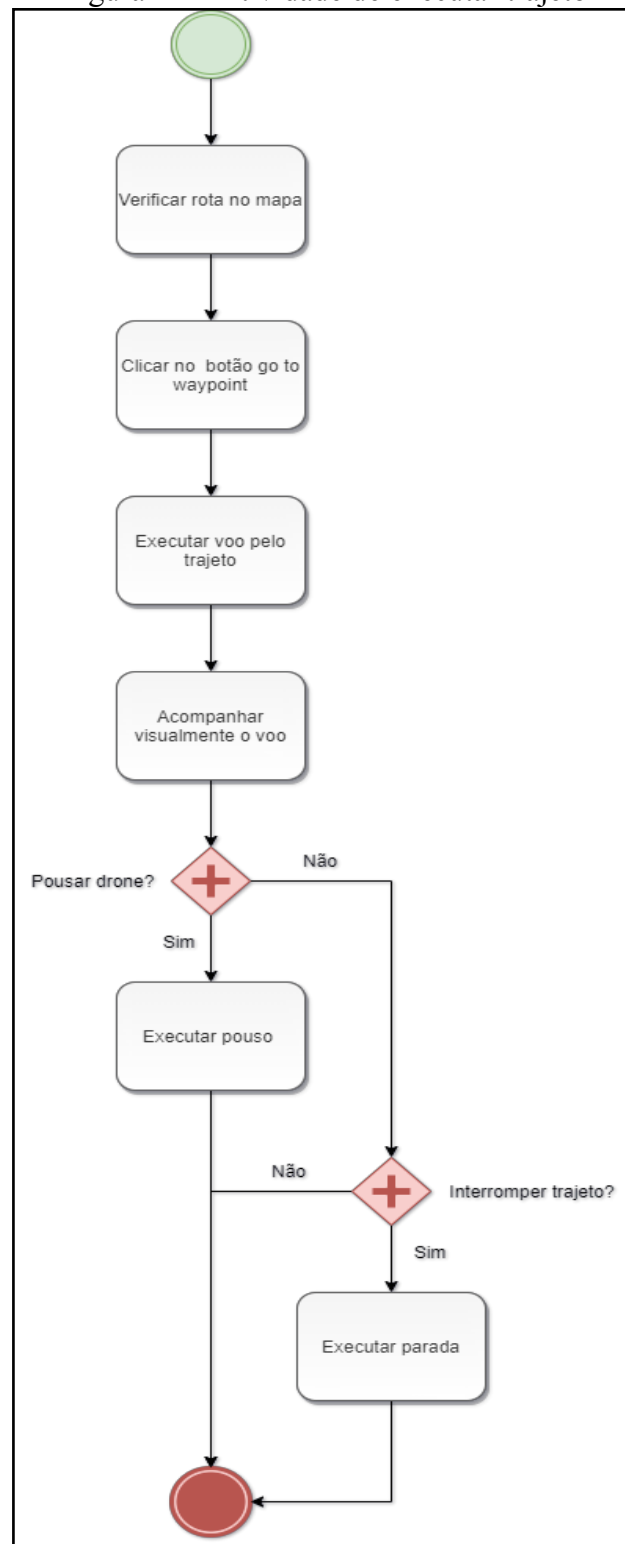
O Quadro 4 apresenta os detalhes da tarefa que define a execução do trajeto a ser percorrido, a Figura 11 exemplifica-os através do diagrama de atividades.

Quadro 4 – Caso de uso *executar trajeto*

UC04. Executar trajeto	
Descrição	Permitir através da arquitetura que o usuário execute o trajeto através da rota selecionada no mapa.
Requisitos atendidos	RF01, RF02, RNF02.
Pré-condição	Rota selecionada no mapa.
Cenário principal	<ol style="list-style-type: none"> <li>1) O usuário verifica a rota no mapa;</li> <li>2) O usuário clica no botão <i>go to waypoint</i>;</li> <li>3) A arquitetura envia o comando ao drone para executar as tarefas de voo pelo trajeto;</li> <li>4) O usuário acompanha o voo pelo mapa exibido na arquitetura e fisicamente o drone sobrevoando a área.</li> </ol>
Fluxo alternativo	<p>No passo 4 caso houver necessidade, o usuário poderá clicar no botão <i>land</i> ou <i>stop</i>.</p> <ol style="list-style-type: none"> <li>5) Botão <i>land</i> a arquitetura envia comando para que o drone pouse imediatamente.</li> <li>6) Botão <i>stop</i> a arquitetura enviará os comandos para que o drone execute a parada imediata e mantenha-se pairando ao ar, mantendo sua última posição.</li> </ol>
Pós-condição	A arquitetura executa a missão pilotando o drone durante o voo.

Fonte: Elaborado pelo autor (2020).

Figura 11 – Atividade de executar trajeto



Fonte: Elaborado pelo autor (2020).

### 3.2.1.5 Caso de uso: Executar retorno a base

Etapa da missão destinada ao retorno a base inicial, gravada pela arquitetura, na etapa da decolagem como visto no Quadro 3, não levando em consideração o trajeto atual. Ao



executar a missão com sucesso e concluir o trajeto, o drone retorna autonomamente para a base, pousando no local designado finalizando a missão.

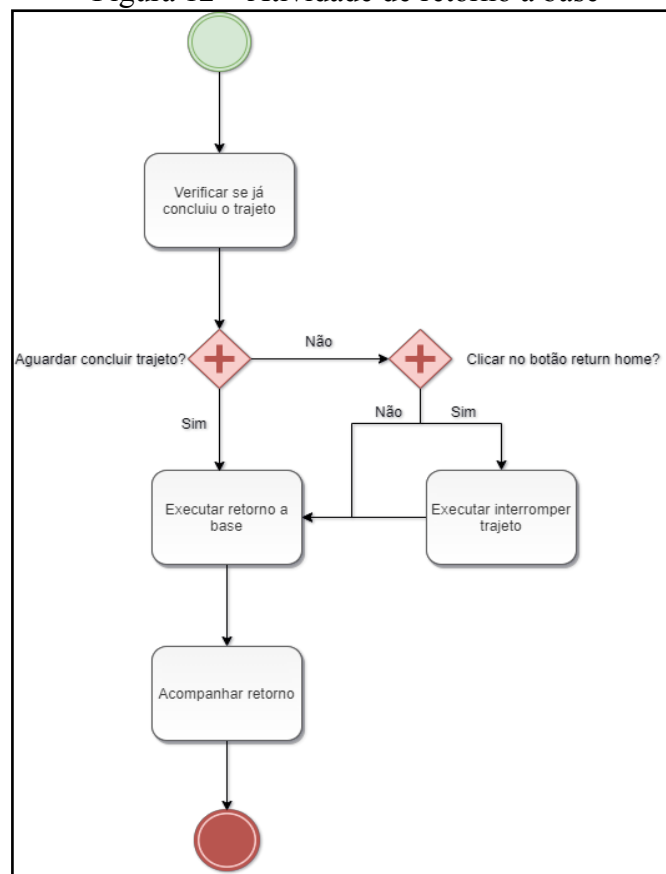
O Quadro 5 apresenta os detalhes da tarefa que define o retorno a base, a Figura 12 exemplifica-o através do diagrama de atividades.

Quadro 5 – Caso de uso executar retorno a base

UC05. Executar retorno a base	
Descrição	Permitir através da arquitetura o retorno autônomo a base.
Requisitos atendidos	RF02, RF04, RNF03.
Pré-condição	Trajeto concluído.
Cenário principal	1) O usuário verifica se o drone concluiu o trajeto selecionado; 2) A arquitetura envia o comando ao drone para executar o retorno a base, caso o trajeto ter sido concluído; 3) O usuário acompanha o drone retornar.
Fluxo alternativo	No passo 1 o usuário pode clicar diretamente no botão <i>return home</i> . 4) A arquitetura envia o comando de retorno a base para o drone, ignorando o trajeto em execução.
Pós-condição	A arquitetura executa a tarefa de forma autônoma.

Fonte: Elaborado pelo autor (2020).

Figura 12 – Atividade de retorno a base



Fonte: Elaborado pelo autor (2020).

### 3.2.1.6 Caso de uso: Executar pouso

Etapa predecessora a missão final, após o drone possivelmente ter concluído com sucesso o trajeto selecionado e retornado a base origem como visto no Quadro 5. Caso o usuário não tenha clicado anteriormente no botão *land*, a arquitetura enviará ao drone um comando de pouso autônomo.

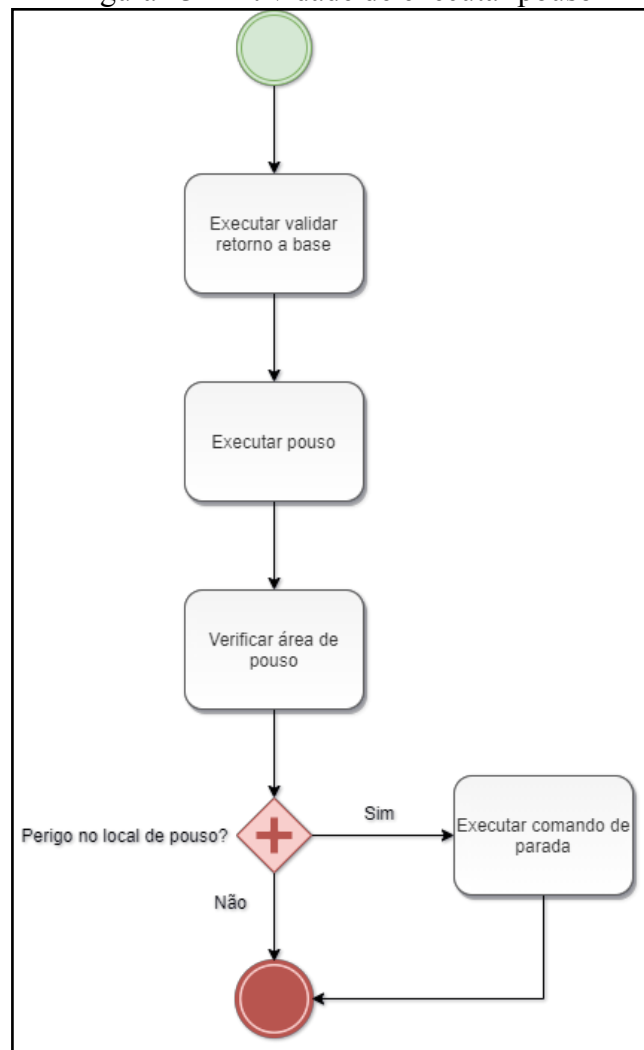
O Quadro 6 apresenta os detalhes da tarefa que executa o pouso concluindo o trajeto, a Figura 13 exemplifica-o através do diagrama de atividades.

Quadro 6 – Caso de uso executar pouso

UC06. Executar pouso	
Descrição	Permitir através da arquitetura o pouso autônomo.
Requisitos atendidos	RF02.
Pré-condição	UC05. Retornado a base.
Cenário principal	1) A arquitetura valida se o drone retornou a base; 2) A arquitetura envia o comando ao drone para executar o pouso; 3) O usuário verifica se a área de pouso está aderente.
Fluxo alternativo	No passo 3, o usuário poderá identificar perigo ao pousar no local. 4) O usuário clica no botão <i>stop</i> obrigando a arquitetura interromper o pouso.
Pós-condição	A arquitetura reinicia limpando os parâmetros gravados durante a missão.

Fonte: Elaborado pelo autor (2020).

Figura 13 – Atividade de executar pouso



Fonte: Elaborado pelo autor (2020).

### 3.2.1.7 Caso de uso: Limpar rota

Etapa final de missão onde o usuário pode optar por limpar a rota selecionada no mapa, na qual executou-se o trajeto. Ao executar essa etapa a arquitetura limpará o mapa e consequentemente os valores de longitude e latitude.

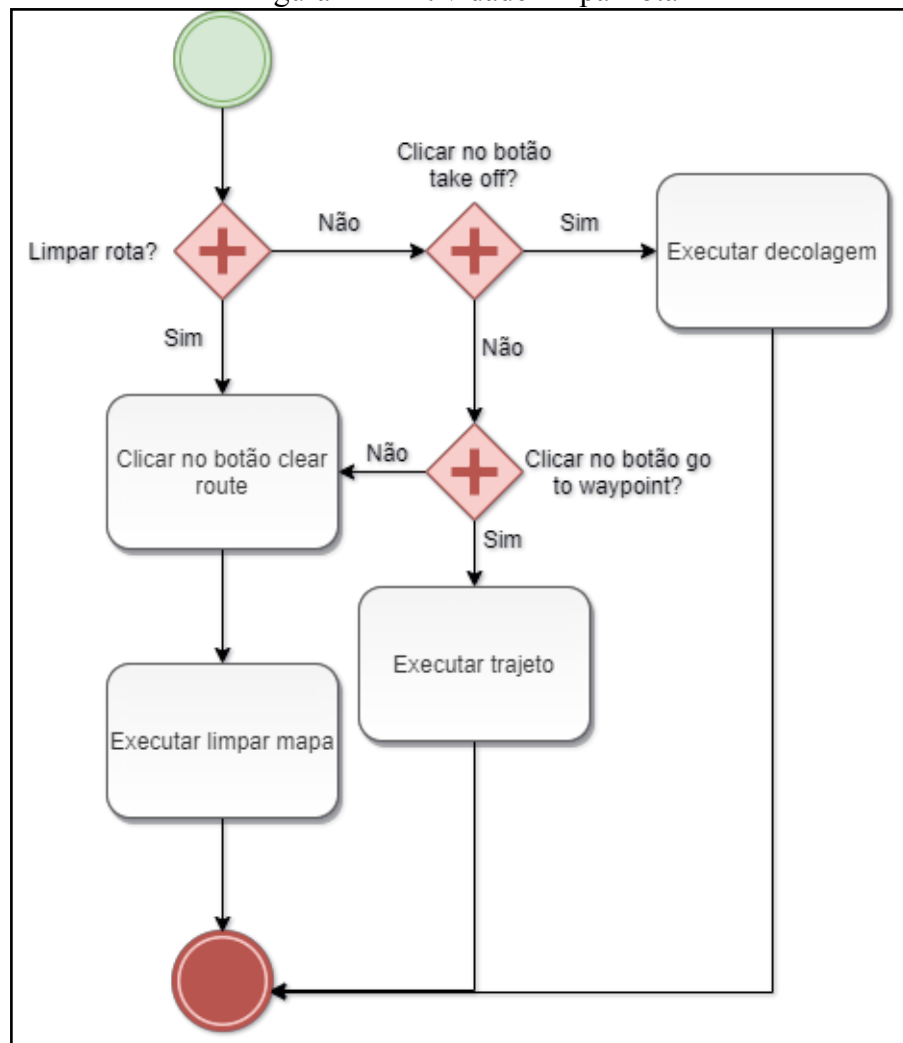
O Quadro 7 apresenta os detalhes da tarefa que limpa o mapa, preparando-o para uma nova missão com um trajeto diferente. A Figura 14 exemplifica-o através do diagrama de atividades.

Quadro 7 – Caso de uso limpar rota

UC07. Limpar rota	
Descrição	Disponibilizar através da arquitetura opção de limpar a rota selecionada no mapa.
Requisitos atendidos	RF01, RF02, RNF02, RNF03.
Pré-condição	UC02. Rota selecionada no mapa.
Cenário principal	1) O usuário clica no botão <i>clear route</i> ; 2) A arquitetura limpa a rota selecionada e valores de longitude e latitude.
Fluxo alternativo	No passo 1, o usuário pode optar por não limpar a rota. 3) O usuário clica no botão <i>take off</i> recomeçando as etapas. 4) O usuário clica no botão <i>go to waypoint</i> para o drone percorrer o mesmo trajeto.
Pós-condição	A arquitetura limpa o mapa e seus parâmetros da rota anterior.

Fonte: Elaborado pelo autor (2020).

Figura 14 – Atividade limpar rota



Fonte: Elaborado pelo autor (2020).

### 3.3 IMPLEMENTAÇÃO

Esse capítulo apresenta toda estrutura de desenvolvimento da arquitetura, contendo especificações dos algoritmos que processam e fazem a gestão dos eventos, além dos cálculos matemáticos envolvendo os pontos de latitude e longitude do mapa, tal como obter a distância precisa entre dois pontos distintos.

Exemplificação e detalhes sobre a referida comunicação entre cliente da arquitetura e hardware servidor do drone, métodos de comunicação e troca de mensagens e pacotes.

O capítulo demonstra como foi realizada integração das bibliotecas na arquitetura, além do SDK e *framework*, através dos métodos extraídos do código fonte, também o uso de APIs para abstrair algumas etapas no desenvolvimento da arquitetura.

#### 3.3.1 Técnicas e ferramentas utilizadas

Essa seção apresenta as técnicas e ferramentas utilizadas no desenvolvimento da arquitetura, detalhando os principais motivadores na escolha pelo Node.js como principal *framework*, além da apresentação das técnicas de integração da arquitetura com as bibliotecas NPM. Apresenta os meios da comunicação entre cliente e servidor, arquitetura e drone propriamente ditos, também o uso das APIs de alto nível. Apresenta a forma de integração entre API do Google Maps e biblioteca *Leaflet*, ambas usadas para exibir e controlar o mapa exibido, mostrando a posição atual do drone e os pontos selecionados no mapa.

No desenvolvimento da arquitetura foi utilizado como ferramenta de trabalho, notebook CCE Win de 14 polegadas com sistema operacional Windows 10 Home 64 BITS, 8 GB de memória RAM, contando com processador core I3-3220 de terceira geração com 3,30 GHZ. Para a pesquisa e os testes dos requisitos funcionais e não funcionais foi utilizado o modelo de drone AR. Drone 2.0 da fabricante francesa Parrot, customizado com bateria estendida.

##### 3.3.1.1 Aplicabilidade Node.js *framework*

A escolha do Node.js como *framework* principal foi dada pela vantagem de construir uma arquitetura assíncrona orientada a eventos, através dessa abordagem foi identificada a possibilidade de executar simultaneamente uma quantidade considerável de tarefas, mantendo a performance e integridade dos métodos. Os eventos são enfileirados e processados numa *pool* de eventos, após esse processo a *thread pool* separa-os por requisição e de forma assíncrona retorna a *promise* da chamada.

No arquivo *server.js* foi desenvolvido o servidor da arquitetura, realiza todo processamento, controle, execução e gestão dos métodos. Para criar o servidor foi utilizado uma

Biblioteca externa em Node chamada `express`, da qual implementa a estrutura de requisições com o protocolo HTTP, tornando possível a troca de informações e comandos através do cliente, nesse contexto refere-se a interface web de pilotagem do drone.

### 3.3.1.2 Dependências da arquitetura

Para executar a arquitetura é preciso instalar algumas dependências no pacote do projeto, o arquivo `package.json` contém as bibliotecas utilizadas, o Node faz a leitura desse arquivo ao executar o comando NPM *install*, que por sua vez realizada toda instalação necessária para o projeto funcionar, inclusive a instalação de forma automatizada dessas bibliotecas.

No Quadro 8 é apresentada a estrutura JSON com as bibliotecas e suas respectivas versões, utilizadas no *build* da arquitetura.

Quadro 8 - Arquivo de dependências

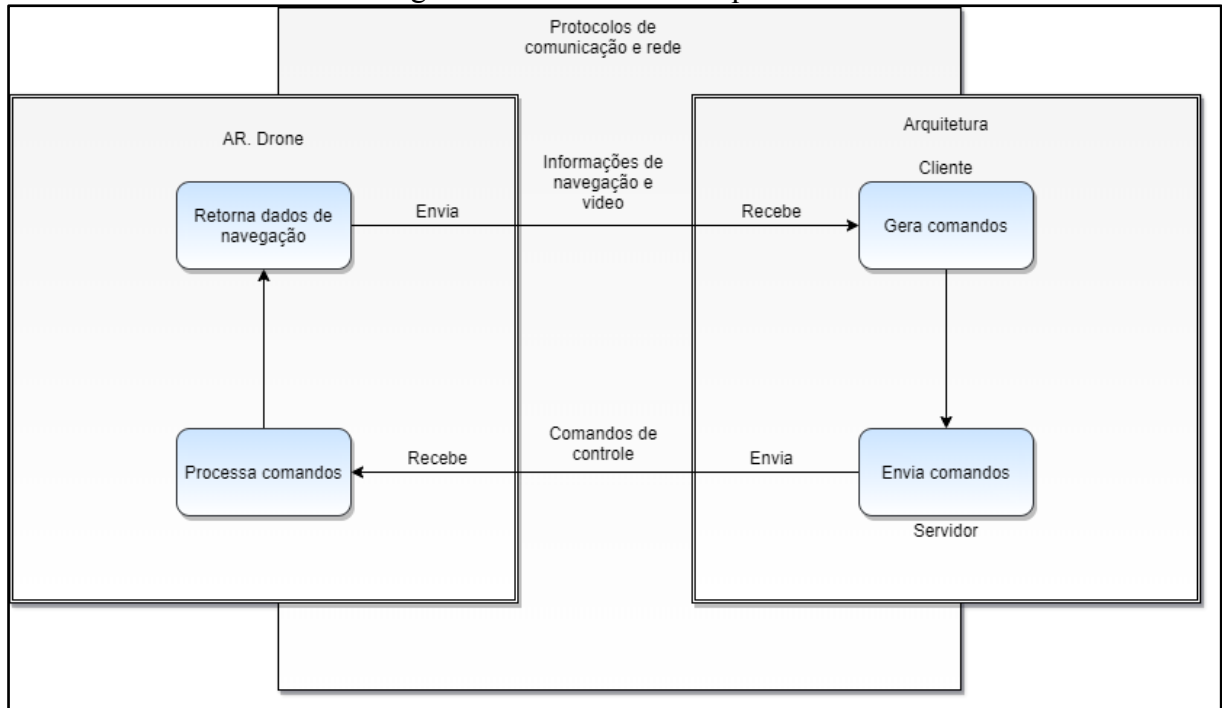
```
{} package.json > {} dependencies
1  {
2    "name": "ar-drone-autonomous",
3    "version": "0.0.1",
4    "description": "Autonomous fly Parrot Ar Drone with GPS module",
5    "main": "index.js",
6    "dependencies": {
7      "ar-drone": "^0.3.3",
8      "ardrone-autonomy": "^0.1.2",
9      "dronestream": "~1.1.1",
10     "express": "~3.4.0",
11     "geolib": "^3.2.1",
12     "leaflet": "^1.6.0",
13     "leaflet-google": "^1.0.1",
14     "node-vincenty": "0.0.6",
15     "nodemon": "^2.0.4",
16     "socket.io": "^2.3.0"
17   },
18   "devDependencies": {},
19   > Debug
20   "scripts": {
21     "test": "echo \"Error: no test specified\" && exit 1"
22   },
23   "repository": {
24     "type": "git",
25     "url": "https://github.com/diegofachinello/ar-drone-autonomous"
26   },
27   "keywords": [
28     "nodecopter",
29     "autonomous",
30     "gps"
31   ],
32   "author": "Diego Fachinello",
33   "license": "FURB",
34   "bugs": {
35     "url": "https://github.com/diegofachinello/ar-drone-autonomous"
36   }
37 }
```

Fonte: Elaborado pelo autor (2020).

Essa seção apresenta os detalhes da aplicabilidade das dependências utilizadas no desenvolvimento da arquitetura, abordagem técnica dos métodos, requisições e processamento das informações.

A Figura 15 exemplifica como a arquitetura está organizada, entre servidor, cliente, protocolos de comunicação e hardware AR.Drone.

Figura 15 - Estrutura da arquitetura



Fonte: Elaborado pelo autor (2020).

A biblioteca NPM `ar-drone` versão 0.3.3 foi utilizada para integrar o drone com arquitetura, servindo também como base obrigatória para executar a biblioteca NPM `ardrone-autonomy` na versão 0.1.2, que por sua vez foi utilizada para desenvolver e estruturar as missões. Além de realizar a conexão entre cliente, servidor e hardware, processar requisições do drone, tanto recebidas quanto enviadas.

A biblioteca NPM `ardrone-autonomy` versão 0.1.2 foi utilizada para executar os métodos disponibilizados através da API do `ar-drone`. Primeiramente foi necessário criar a conexão entre as bibliotecas `ardrone-autonomy` e `ar-drone`, para isso foi estabelecida uma chamada utilizando o método `require('ardrone-autonomy')`, ganhando acesso a todos os métodos da interface propriamente dita. Adquirindo o recurso para acessar métodos de ambas as bibliotecas numa única arquitetura, através da variável `mission`.

Para criar a missão foi usado o método `mission.createMission()` que internamente executa o método `arDrone.createClient(opções)`.

Concedendo acesso aos comandos da API, para a configuração `mission.client().config('general:navdata_demo', 'FALSE')` que executa `client.config(chave, valor, retorno de chamada)`, através desse método é possível enviar comandos de configurações diretamente ao SDK oficial da Parrot, visou-se a utilizar

somente esse, que por sua vez, retorna em tempo real todos dados de navegação do AR.Drone (PISKORSKI, 2015).

Através da biblioteca NPM `dronestream` versão 1.1.1, foi possível integrar na arquitetura a transmissão em tempo real de vídeo capturado pela câmera frontal do drone, chamamos essa técnica de visualização de First Person View (FPV). O protocolo de comunicação utilizado para obter o vídeo do AR.Drone é o TCP.

Após subir o servidor Node e estabelecer a conexão com o drone, foi adicionada a notação `require("dronestream").listen(server)` no arquivo `server.js` para iniciar a transmissão, com isso foi implementada uma `div` no arquivo `index.html` e adicionado o `ID` `dronestream` para visualizar a interface web diretamente no navegador (WEISSHUHN, 2013).

A biblioteca NPM `express` versão 3.4.0 foi utilizada para subir o servidor em Node, criando uma conexão HTTP no endereço de IP 192.168.1.1, adicionando uma estrutura *rest* que processa as requisições disparadas através da interface web. Não limitando-se apenas a isso, também serve para integrar o `socket.io` ao server HTTP, sendo esse a chave para controlar uma arquitetura orientada a eventos (WILSON, 2019).

A biblioteca NPM `geolib` versão 3.3.1 foi utilizada para abstrair fórmulas e cálculos matemáticos, os seguintes métodos foram implementados:

- a) cálculo da distância precisa entre duas coordenadas `getPreciseDistance (start, end [int precision])`, com isso é obtida a referida distância que o drone deverá percorrer até chegar ao seu destino, *waypoints* percorridos ao executar a missão de trajeto;
- b) cálculo para obter o ângulo entre as duas coordenadas geográficas `getGreatCircleBearing(origin, destination)`, com isso é obtido o referido ângulo, necessário para calcular a linha de deslocamento com base na distância entre a posição atual do drone ao próximo *waypoint* da rota selecionada.

O conceito de *great circle* em tradução livre:

A distância entre dois pontos na Terra é calculada usando um Grande Círculo, ou seja, a menor distância que em uma esfera está ao longo do caminho de um círculo cujo centro é o centro da esfera. Como a Terra não é modelada como uma esfera, mas como um elipsoide, calcular essa distância pode ser bastante complicado. Neste documento, vários métodos são apresentados, do simples (e intuitivo) ao mais preciso. Mas, com base em testes feitos para comparar todos os métodos, a diferença real entre os métodos é inferior a 1% em distâncias inferiores a algumas centenas de quilômetros. Esses cálculos não incorporam altitude. (CARTER, 2002, p. 2).



Ao obter a distância e *bearing* angular (ângulo) é executado o cálculo para encontrar os componentes X e Y do vetor de displacement (descolamento), sendo a fórmula utilizada `displacementVector = $V([Math.cos(greatCircleBearing) * preciseDistance, Math.sin(greatCircleBearing) * preciseDistance])`, na sintaxe JavaScript `Math.cos` (cosseno) e `Math.sin` (seno), levando em consideração o cálculo matemático de razões trigonométricas, teremos um vetor de dois argumentos contendo a distância com relação ao *bearing* inicial, aplicado a ambos.

Ao final executa-se o cálculo de hipotenusa entre os valores do vetor, obtendo a distância em linha reta relacionando o drone até o próximo ponto na rota. Cálculo JavaScript `Math.hypot(displacementVector.elements[0], displacementVector.elements[1])`.

O cálculo para obter o ângulo em grau necessário para rotacionar o yaw (rotação no eixo vertical) do drone:

- a) variável `yawAdjustment` recebe `((droneBearing * 180) / por PI)` que representa a proporção entre circunferência de um círculo com o seu diâmetro, aproximadamente 3.14159;
- b) após é executada uma repetição para normalizar e ajustar em graus nos casos que o `yawAdjustment` for maior que 180 recebe `(yawAdjustment - 360)`, quando menor que -180 recebe `(yawAdjustment + 360)`;
- c) ao final executa-se o cálculo do `(yawAdjustment - currentYaw)` que foi obtido diretamente do drone, ou seja, refere-se ao ângulo atual do drone em relação a câmera frontal, com isso obtemos o valor angular em graus que o deve rotacionar verticalmente em sentido horário.

A biblioteca NPM `leaflet` versão 1.6.0 foi utilizada no desenvolvimento de toda interação com o mapa, a escolha deu-se pela simplicidade, performance e usabilidade, também por executar no server pesando apenas 38kb. Ao iniciar a interface da arquitetura pelo navegador é disparado um evento de abertura do mapa interativo, nesse momento é obtido pelo navegador a localização atual do computador (LIEDMAN, 2020).

As tratativas com relação a abertura do mapa, exemplificadas através dos seguintes métodos:

- a) obter a localização atual do computador com alta precisão através do método `navigator.geolocation.getCurrentPosition()`;
- b) iniciar o mapa nas coordenadas de longitude e latitude atuais, através do método `L.map('map').setView([latitude, longitude], 20)`;
- c) exibir o mapa no formato satélite usando a API do Google Maps para ativação com

o método `L.Google('SATELLITE').addLayer(googleLayer);`

- d) adicionar ao mapa na latitude e longitude inicial o ícone do computador, através do método `L.marker([latitude, longitude], { ícone: objetoÍcone }).addTo(map);`
- e) adicionar ao mapa o ícone do drone ativando o evento de atualização em tempo real ao alterar as coordenadas de latitude e longitude, através do método `L.marker([latitude, longitude], { ícone: objetoÍcone }).addTo(map).`

A biblioteca foi utilizada também para processar os marcadores chamados *waypoints*, cada click no mapa é gerado visualmente um marcador na latitude e longitude, capturada no evento de click sobre o local no mapa. Esses dados são armazenados num *array* de *waypoints*, cada qual na respectiva posição do click, ao clicar em mais locais no mapa, através da biblioteca *leaflet* é desenhado um traço que faz a ligação visual entre os pontos, formando uma rota conexa.

A biblioteca *leaflet-google* versão 1.0.1 foi utilizada para processar os movimentos no mapa, tal como diminuir ou aumentar o zoom, mover o mapa arrastando-o através de um click mantendo pressionado, controle de scroll do mouse, zoom por *touch pad* do notebook e demais atalhos do teclado.

A biblioteca NPM *socket.io* versão 2.3.0 foi utilizada para implementar a comunicação em tempo real, bidirecional e baseada em eventos entre a interface da arquitetura pelo navegador e o servidor Node do arquivo *server.js* (ARRACHEQUESNE, 2020).

Através da interface web da arquitetura, são emitidos os eventos gerados pelos clicks nos botões disponíveis.

Eventos emitidos pela interface web cliente, tratados pelo servidor, são:

- a) botão go to *waypoint*:

- `socket.emit('go', { latitude: destino, longitude: destino })`  
dispara evento para o server tratar as tarefas de disparar o drone pelo trajeto selecionado, levando em consideração a exemplificação do método `mission.go(posição);`

- b) botão stop `socket.emit('stop')`

- dispara evento para o servidor tratar a parada do drone, zerando as variáveis locais do trajeto, como exemplificado no método `client.stop();`

- c) botão takeoff `socket.emit('takeoff')`

- dispara evento para o servidor tratar a decolagem do drone, gravando a latitude

e longitude iniciais como home, exemplificado no método `client.takeoff(retorno de chamada);`

d) botão reset `socket.emit('reset')`

- dispara evento para o servidor processar o desativar do modo de aviso de emergência que poderá ocasionar erro durante o voo;

e) botão home:

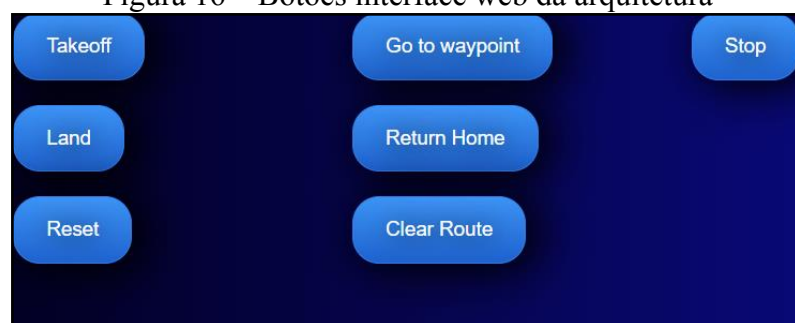
- `socket.emit('go', { latitude: inicial, longitude: inicial })`  
dispara evento para o servidor retornar a base inicial gravado ao executar o `takeoff`.

Eventos emitidos pelo servidor da arquitetura para a interface cliente tratar, são:

- `socket.emit('drone', { latitude: atual, longitude: atual, yaw: atual, distance: atual, battery: porcentagem atual da bateria })`  
- dispara em tempo real de execução evento para o cliente tratar atribuindo as informações na interface para o usuário visualizar;
- `socket.emit('waypointReached', { latitude: destino, longitude: destino })`  
- dispara para o cliente tratar o evento de chegada ao ponto na rota, ao executar o cliente processa o próximo *waypoint* da lista caso existir, disparando evento de volta ao servidor, para voar até o próximo ponto da lista gravada de *waypoints*.

A Figura 16 mostra os botões disponíveis na interface web, cada qual com sua devida ação, como exemplificado anteriormente.

Figura 16 – Botões interface web da arquitetura



Fonte: Elaborado pelo autor (2020).

### 3.3.2 Estruturas e protocolos de comunicação

Os serviços de comunicação entre o AR.Drone 2.0 e arquitetura cliente, o controle do drone é feito através destes principais serviços de comunicação:

- o controle e a configuração do drone são feitos enviando comandos AuTonomous (AT) na porta UDP 5556. A latência de transmissão dos comandos de controle é

crítica para a experiência do usuário. Os comandos descritos anteriormente são enviados regularmente (geralmente 30 vezes por segundo);

- b) informações sobre o drone (como status, posição, velocidade, velocidade de rotação do motor, altitude) são chamados de *navdata*, são enviados pelo drone para o cliente (arquitetura) na porta UDP 5554. A *navdata* também inclui a identificação de informações de detecção que podem ser usadas para realidade aumentada, não aplicados nesse trabalho;
- c) o fluxo de vídeo em tempo real capturado é enviado pelo AR.Drone para o dispositivo cliente (arquitetura) na porta 5555 TCP para AR.Drone 2.0. As imagens do fluxo de vídeo são decodificadas usando o *codec* do SDK oficial Parrot (PISKORSKI, 2012).
- d) um quarto canal de comunicação, chamado porta de controle, é estabelecido na porta TCP 5559 para transferir dados críticos, por oposição a outros dados que podem ser perdidos sem efeito perigoso, é utilizado para recuperar dados de configuração e reconhecer informações importantes, como o envio de informações de configuração *navdata*.

### 3.4 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Na interface web gerada pela arquitetura o usuário tem como objetivo principal a interação com o mapa e botões de ação, além da opção de configurar os parâmetros antecedentes a missão de voo.

Esta seção apresenta a tela principal da arquitetura, o mapa interativo, a transmissão do vídeo em tempo real, os botões operacionais, parâmetros configuráveis e informações *navdata* obtidas através do drone como supramencionado. Também apresenta a operacionalidade da arquitetura, subdivida em etapas para conectar, configurar, interagir e executar.

#### 3.4.1 Conectar arquitetura ao AR.Drone 2.0

Antes de executar a arquitetura, subir servidor Node e abrir interface *localhost*, conectando a arquitetura ao AR.Drone 2.0, é necessário seguir os seguintes passos:

- a) ligar o Parrot AR.Drone 2.0 e verificar se as quatro hélices fazem o movimento inicial de teste, caso sim estão ok, caso contrário estão com problema, a fabricante recomenda não realizar o voo;
- b) aguardar 10 segundos após o passo anterior, nesse tempo o AR.Drone 2.0 gera a rede WI-FI;

- c) conectar na rede WI-FI `ardrone2_251829` gerada pelo AR.Drone 2.0. Observação importante, o nome da rede pode não ser o mesmo, pode variar conforme modelo.

Os passos são:

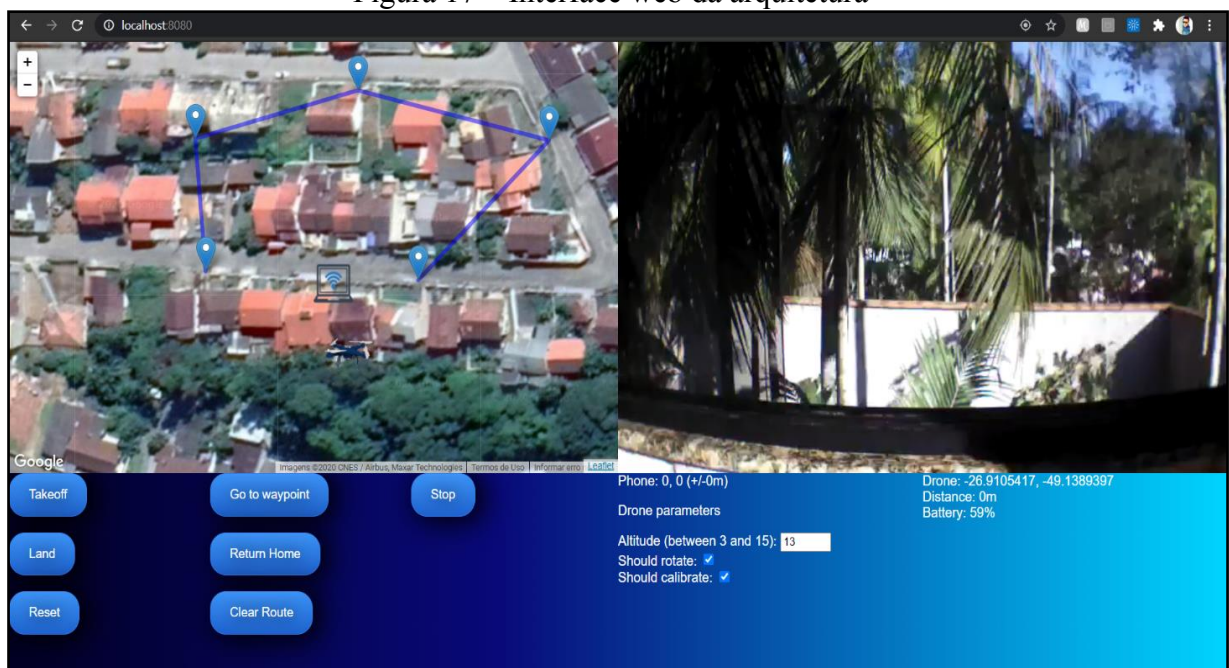
- caso o computador estiver conectado em alguma rede WI-FI com internet, ao conectar na rede do drone, o aparelho ficará sem conexão com internet, nesse caso recomenda-se:
- utilizar uma conexão via cabo plugado quando disponível, ou USB com uma mini placa WI-FI *off board*, além dessas opções também existe o modem USB que utiliza um chip de qualquer operado com dados moveis, ao plugar gera uma rede de conexão à internet.

Após estabelecida conexão com a rede WI-FI do drone será necessário iniciar o servidor node e com isso a própria arquitetura, contudo suponha-se que o projeto e todas as bibliotecas necessárias, inclusive o próprio Node estejam instalados no computador.

### 3.4.2 Tela principal de operacionalidade da arquitetura

A tela principal de operacionalidade da arquitetura, demonstrada na Figura 17 é o centro de comando e pilotagem do AR.Drone. Por questões de usabilidade foi optado por desenvolver uma única tela, disponibilizando ao usuário toda gestão da arquitetura em apenas um monitor.

Figura 17 – Interface web da arquitetura



Fonte: Elaborado pelo autor (2020).

### 3.4.3 Tela de operacionalidade da arquitetura

A interface web é subdividida em duas seções, a primeira parte localizada na esquerda é demonstrada na Figura 18, refere-se a tela de operacionalidade da arquitetura. Nessa parte o mapa e botões estão disponíveis para interação, o mapa exibe a localização atual do computador e do drone, basta selecionar os *waypoints* para compor o trajeto desejado.

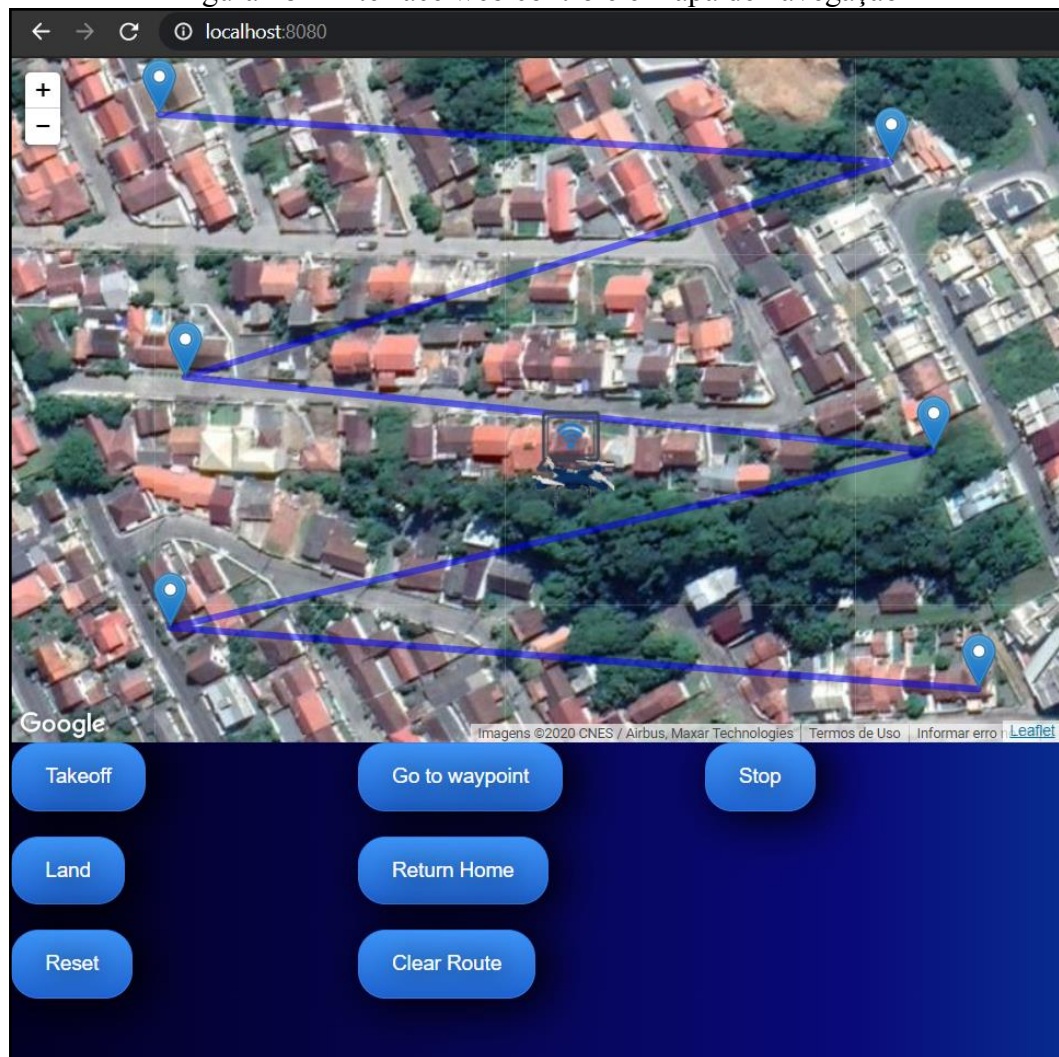
A parte do mapa superior da tela contém o mapa interativo, as formas de interação disponíveis, são:

- a) um `click` (selecionar ponto da rota no local);
- b) no canto superior esquerdo é possível clicar em menos e mais, aumentar e diminuir `zoom` respectivamente;
- c) o `zoom` poderá ser aumentado ou diminuído através do `scroll` do mouse ou pelo `touchpad` do notebook, quando disponíveis;
- d) clicar e segurar com movimento faz com que mova o mapa para outras localidades.

A parte inferior da tela contém os botões clicáveis, executam as ações atribuídas pela arquitetura, segue abaixo a tradução de cada botão:

- a) `takeoff` (decolar drone);
- b) `land` (pousar drone);
- c) `reset` (resetar desabilitando o modo de emergência);
- d) `go to waypoint` (ir para o próximo ponto);
- e) `return home` (retorna ao local inicial da decolagem);
- f) `clear route` (limpar a rota selecionada no mapa);
- g) `stop` (parar drone).

Figura 18 – Interface web controle e mapa de navegação



Fonte: Elaborado pelo autor (2020).

#### 3.4.4 Tela analitica da arquitetura

A segunda parte da interface está localizada ao lado direito da tela, assim como é demonstrada na Figura 19. Essa parte contém o vídeo em tempo real no formato FPV, os campos de parâmetros e os dados de navegação enviados pelo AR.Drone, disponíveis para visualização e análise.

Na parte superior da tela exibe-se o vídeo capturado pelo drone, através disso é possível acompanhar todo trajeto e ver o que o drone está vendo, tomar as devidas ações conforme as normas de vigilância, aplicadas ao local do voo.

Na parte inferior esquerda da tela localizam-se os parâmetros opcionais de configuração da arquitetura, sendo estes:

- a) altitude between 3 and 15 (selecionar altitude em metros, mínimo 3 e máximo 15);

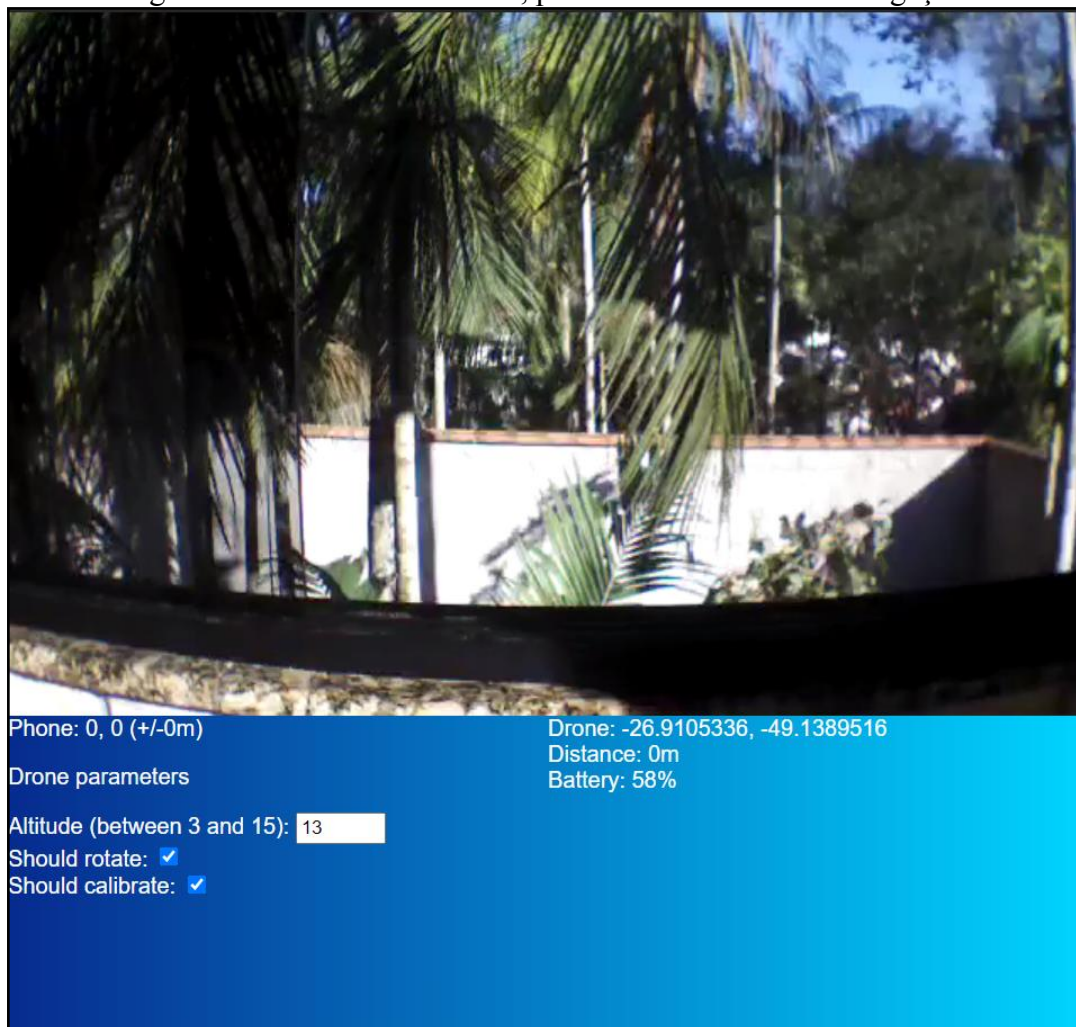


- b) `should rotate` (deve rotacionar a câmera na direção do próximo *waypoint*);
- c) `should calibrate` (deve calibrar o magnetômetro, essa ação faz com que o AR.Drone rotacione 360 graus).

Na parte inferior direita da tela localiza-se informações de navegação obtidas do AR.Drone 2.0, sendo estas:

- a) `drone` (latitude e longitude atualizadas em tempo real);
- b) `distance` (distância em metros até o próximo *waypoint*);
- c) `battery` (porcentagem atual da carga de bateria do AR.Drone).

Figura 19 – Interface web FPV, parâmetros e dados de navegação



Fonte: Elaborado pelo autor (2020).

### 3.5 RESULTADOS E DISCUSSÕES

Os testes de funcionamento da arquitetura foram realizados durante todo o ciclo de desenvolvimento, seguindo o conceito de melhoria contínua, ciclo que permaneceu no trabalho. Para realizar a coleta de resultados, foram realizados 3 cenários de teste, cada qual



implementado com suas particularidades. Essa seção apresenta os resultados e discussões, customizações do AR.Drone 2.0 e os cenários de testes de 1 a 3, exemplificados com dados reais, coletados durante o voo em trajetos distintos.

Os cenários de teste de 1 a 3 foram executados no Vale Auto Shopping em Blumenau, Santa Catarina, a escolha do local deu-se pela necessidade de ser possuir uma área aberta e ampla, sem obstruções físicas que impactassem nos voos de teste.

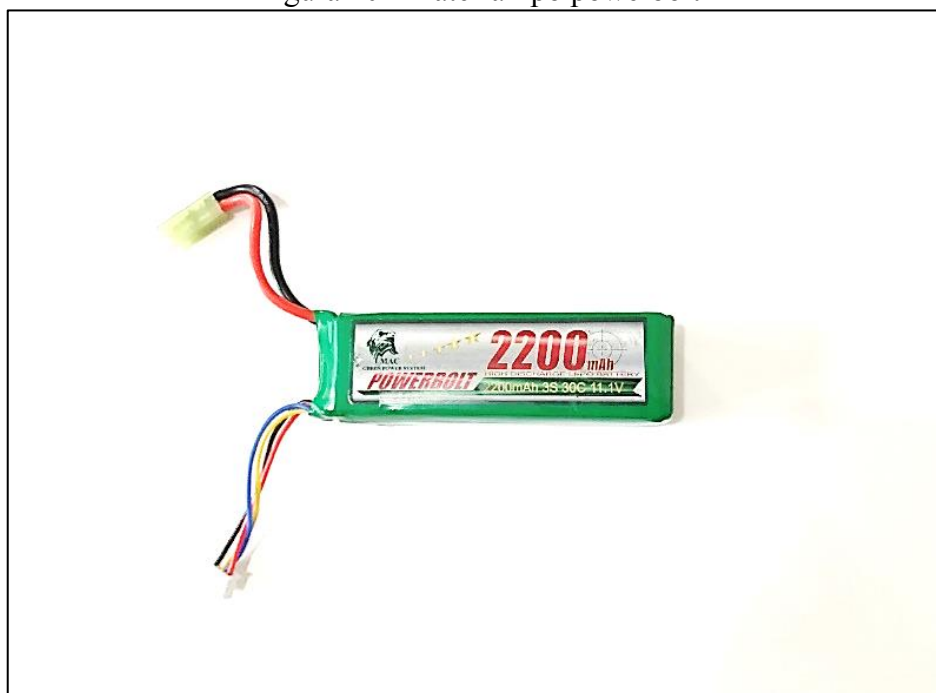
### 3.5.1 Customização AR.Drone 2.0

Para o projeto virar realidade foi necessário adquirir um drone modelo AR.Drone 2.0 com módulo de GPS vendido separadamente, bateria com maior duração comparada a original, além disso foi necessário customizar o casco externo.

A escolha de utilizar esse modelo no trabalho deu-se pela compatibilidade de obter informações de latitude e longitude, providas pelo módulo de GPS compatível apenas com o modelo AR.Drone 2.0, pois em sua primeira versão não existe a mesma compatibilidade de hardware e SDK (PISKORSKI, 2012).

A troca da bateria original por uma versão estendida deu-se pelo motivo de aumentar a duração do voo em até 5 minutos, proporcionando a execução de pelo menos 2 trajetos com 5 *waypoints* cadastrados no mapa. O modelo da bateria Figura 20 conta com 2200 MiliAmpere-Hora (MAH) de capacidade da carga, cerca de 40% maior que a capacidade do modelo original.

Figura 20 – Bateria lipo powerbolt



Fonte: Elaborado pelo autor (2020).

O casco externo foi customizado para comportar a nova bateria, medindo 40% maior que a original não coube na estrutura. Como pode ser visto na Figura 21, foi necessário criar um compartimento para encaixar internamente a bateria na estrutura do casco, assegurando o balanceamento uniformemente do peso. Para chegar a essa analogia, foi estudado o funcionamento geral de balanceamento ao rotacionar os motores do AR.Drone 2.0.

Figura 21 – Casco externo customizado parte inferior



Fonte: Elaborado pelo autor (2020).

Nos primeiros voos de teste com o módulo GPS foi identificado uma margem de erro de até 10 metros na precisão das coordenadas obtidas pelo AR.Drone. Esse problema ocorre devido ao módulo por padrão ficar escondido internamente no casco do drone, obstruindo a precisão dos sinais emitidos pelos satélites.

Para corrigir o problema e obter maior precisão das coordenadas, foi preciso customizar novamente o casco externo, criando-se uma abertura na parte superior, que coubesse o módulo encaixado de forma segura e livre de obstruções. Na Figura 22 é possível visualizar como ficou a estrutura de encaixe.

Figura 22 – Casco externo customizado parte superior



Fonte: Elaborado pelo autor (2020).

A Tabela 1 compara a diferença de peso e autonomia entre os dois modelos de bateria, levando em consideração o módulo de GPS plugado no AR.Drone.

Tabela 1 - Diferenças bateria original e paralela

Modelo da bateria	Diferença de peso em gramas	Autonomia média de voo em minutos
Original	135	8
Paralela	172	15

Fonte: Elaborado pelo autor (2020).

A Tabela 2 comparada a precisão do módulo de GPS e quantidade de satélites conectados, antes e após aplicar as customizações supracitadas.

Tabela 2 – Diferenças casco original e customizado

Casco externo	Quantidade de satélites conectados	Margem de erro na precisão em metros
Original	7	6
Customizado	13	2

Fonte: Elaborado pelo autor (2020).

### 3.5.2 Cenário de teste 1

No cenário de teste 1 foi selecionado um trajeto de apenas um *waypoint*, a distância foi calculada pela arquitetura antes de executar a missão. No mapa foi selecionado um ponto bem

próximo e visível ao drone, no máximo em linha reta possível, sem o parâmetro de virar em direção ao *waypoint*, parâmetro de calibrar selecionado, 10 metros de altitude iniciais.

O drone foi posicionado no chão com a câmera frontal em direção ao ponto no mapa, o trajeto foi executado dessa forma por 5 vezes consecutivas. Ao final de cada trajeto quando o drone alcançou o ponto, foi disparado o evento de retornar para a base inicial. Na Tabela 3 é demonstrada a distância calculada em cada execução, coordenadas e margem de erro na precisão ao chegar no *waypoint* destino, também ao retornar a base.

Tabela 3 – Resultados da execução do cenário 1

Execução	Trajeto origem e destino	Distância em metros arredondado	Coordenadas de latitude e longitude	Margem de erro na precisão em metros
1	<i>Waypoint</i> destino	36	-26.864890, -49.085248	4
	Base origem	34	-26.865172, -49.084912	2
2	<i>Waypoint</i> destino	35	-26.864890, -49.085248	3
	Base origem	30	-26.865211, -49.084852	1
3	<i>Waypoint</i> destino	36	-26.864890, -49.085248	4
	Base origem	37	-26.865222, -49.084869	3
4	<i>Waypoint</i> destino	38	-26.864890, -49.085248	2
	Base origem	40	-26.865190, -49.084869	5
5	<i>Waypoint</i> destino	31	-26.864890, -49.085248	4
	Base origem	32	-26.865192, -49.084934	3

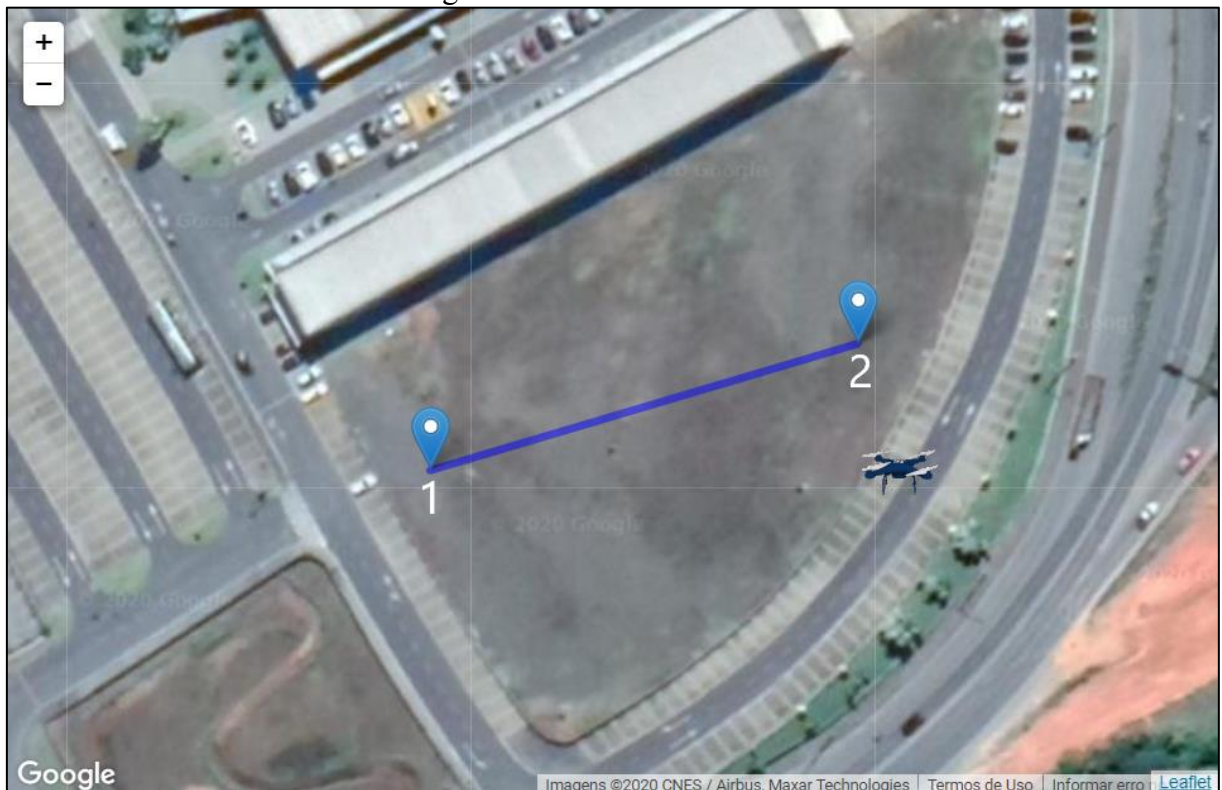
Fonte: Elaborado pelo autor (2020).

### 3.5.3 Cenário de teste 2

No cenário de teste 2 foi selecionado um trajeto composto por 2 *waypoints*, o primeiro à esquerda com relação a posição atual do drone, o segundo a direita, assim como demonstrado na Figura 23. A missão com parâmetro ativos de virar em direção ao *waypoint*, parâmetro de calibrar não selecionado e 12 metros de altitude iniciais.

No teste o drone foi posicionado com a câmera frontal virada para o lado direito em relação ao primeiro *waypoint*, ao executar a missão o drone decolou atingindo a altitude desejada, girando em sentido horário na direção do ponto destino, performando com sucesso a etapa.

Figura 23 – Cenário de teste 2



Fonte: Elaborado pelo autor (2020).

A Tabela 4 demonstra a distância calculada pela arquitetura, da base origem até o ponto 1 e 2 consecutivamente, além da margem de erro na precisão das coordenadas.

Nesse teste em específico na segunda execução a bateria do drone estava abaixo de 40% da autonomia total. Ao finalizar a execução até o ponto 1 do percurso, a arquitetura executou o evento da carga com porcentagem em aproximadamente 35%, realizando de forma autônoma o retorno a base, nesse cenário houve uma margem de erro de aproximadamente 2 metros até base origem.

Ao retornar e pousar no local, foi tentado realizar uma nova missão pelo trajeto anterior, porém a arquitetura não permitiu que o AR.Drone decolasse, validando a porcentagem da carga da bateria. Evitou-se assim um possível comprometimento do voo, com o descarregamento acelerado da bateria, ocasionando a queda do drone.



Tabela 4 – Resultados da execução do cenário 2

Execução	Trajeto origem e destino	Distância em metros arredondado	Coordenadas de latitude e longitude	Margem de erro na precisão em metros
1	Waypoint 2	42	-26.864874, -49.084896	2
	Waypoint 1	58	-26.865025, -49.085599	3
	Base	23	-26.865173, -49.084920	2
2	Waypoint 1	62	-26.865025, -49.085599	4
	Base	57	-26.865173, -49.084920	5

Fonte: Elaborado pelo autor (2020).

### 3.5.4 Cenário de teste 3

No cenário de teste 3 foi selecionado um trajeto de 5 *waypoints*, dispersos porém próximos no formato de um retângulo, basicamente simulando o percurso que faria para a realizar a vigilância autônoma de um espaço aberto, como exemplo o bloco S da FURB, assim como exemplificado na Figura 24.

Figura 24 – Cenário de teste 3



Fonte: Elaborado pelo autor (2020).

A Tabela 5 demonstra a distância calculada pela arquitetura entre os *waypoints* selecionados no mapa, também a distância da base até o ponto 1 e do ponto 4 até a base, considerando o retorno, além da margem de erro na precisão das coordenadas. Esse cenário por se tratar do principal com relação ao objetivo do trabalho, foi executado até esgotar a carga total

da bateria, em média cada percurso levou em torno de 3 minutos para ser totalmente percorrido, com velocidade média de 2 metros por segundo, totalizando em 3 execuções.

Em cada uma das 3 execuções foi possível observar que a arquitetura calculou o giro do drone, com relação ao próximo *waypoint* e margem de erro de 2 a 7 graus. Isso fez com que cada percurso sobrevoado não ocorresse da mesma forma que o seu antecessor.

Porém foi possível analisar que não houve impacto significativo na experiência do usuário em questão, pois foi possível acompanhar todo o trajeto pelo vídeo em FPV através da interface web, isso pelo fato do hardware do AR.Drone possuir uma lente da câmera frontal com ângulo de visão em 90 graus.

Tabela 5 – Resultados da execução do cenário 3

Trajetória origem e destino	Distância em metros arredondado	Coordenadas de latitude e longitude	Margem de erro na precisão em metros
<i>Waypoint 4</i>	11	-26.864992, -49.085825	3
<i>Waypoint 3</i>	54	-26.864743, -49.085964	1
<i>Waypoint 2</i>	8	-26.864274, -49.084982	2
<i>Waypoint 1</i>	14	-26.864508, -49.084856	1
Base origem	19	-26.864742, -49.085269	4
Execução 2	-	-	-
<i>Waypoint 4</i>	11	-26.864992, -49.085825	2
<i>Waypoint 3</i>	48	-26.864743, -49.085964	3
<i>Waypoint 2</i>	6	-26.864274, -49.084982	4
<i>Waypoint 1</i>	15	-26.864508, -49.084856	2
Base origem	17	-26.864756, -49.085298	1
Execução 3	-	-	-
<i>Waypoint 4</i>	9	-26.864992, -49.085825	5
<i>Waypoint 3</i>	45	-26.864743, -49.085964	3
<i>Waypoint 2</i>	7	-26.864274, -49.084982	2
<i>Waypoint 1</i>	16	-26.864508, -49.084856	4
Base origem	15	-26.864779, -49.085305	1

Fonte: Elaborado pelo autor (2020).

### 3.5.5 Comparativo entre os trabalhos correlatos

Nas informações do Quadro 9 é possível observar que o trabalho proposto consegue destacar-se na questão de disponibilizar a vigilância autônoma de espaços externos, adaptando o cadastro de rotas do trabalho de Rocha (2016). Com base e rotas cadastradas previamente pelo usuário, juntamente pela interface onde é possível visualizar o vídeo em tempo real sendo

registrado. Através dessa vigilância acreditasse ser possível identificar atividades não permitidas ou formas de violência físicas nos espaços verificados.

O trabalho correlato de Vanz (2015) se destaca ao disponibilizar o simulador de drone que simultaneamente controla o modelo físico. O trabalho correlato de Rocha (2016) se destaca com a disponibilidade de um aplicativo mobile que traça rotas para facilitar o deslocamento de estudantes na universidade. Por fim, o trabalho correlato do Borrow (2014) se destaca no reconhecimento de objetos em tempo real, além de percorrer rotas em espaços internos desconhecidos.

Quadro 9 - Comparativo entre os trabalhos correlatos

Características	Vanz (2015)	Rocha (2016)	Borrow (2014)
geolocalização	Não	Sim	Não
simulador	Sim	Não	Não
sistema Web	Sim	Sim	Sim
app mobile	Não	Sim	Não
cadastro de Rotas	Não	Sim	Não
reconhecimento de objetos	Não	Não	Sim
autônomo	Não	Não	Sim

Fonte: elaborado pelo autor (2020).



## 4 CONCLUSÕES

O objetivo principal do trabalho foi o desenvolvimento de uma arquitetura que permitisse controlar o AR.Drone 2.0 da fabricante Parrot de forma autônoma, utilizando uma interface web para cadastrar uma base e selecionar no mapa o percurso desejado para o drone realizar a vigilância do local.

Para tornar isso possível trazendo-o a realidade foi desenvolvida uma arquitetura em Node.js integrada a bibliotecas NPM, além de customizar conforme necessidade o quadricóptero, tornando possível selecionar o trajeto desejado e consequentemente o cadastro da rota origem. Disponibilizando acesso ao mapa atualizado no formato de visualização por satélite, além da visualização por vídeo em tempo real do percurso sobrevoado. Este objetivo foi atendido.

O cenário de aplicabilidade do estudo realizado foi identificar possíveis ameaças à segurança de locais externos, tal como universidades, escolas, empreendimentos, bancos etc. A arquitetura disponibiliza esse recurso de vigilância aérea, sem necessidade manual de pilotagem do AR.Drone.

Referente a implementação pode-se afirmar que foi uma pesquisa inovadora com aplicabilidade no mundo real, desafiante de certo modo, conseguir integrar toda essa gama de bibliotecas NPM disponíveis. Estudar toda parte do hardware, SDK e APIs, além de implementar e testar incansavelmente até atingir o resultado esperado de voo autônomo.

No desenvolvimento as tecnologias web agregaram significativamente ao trabalho. Foi possível construir uma arquitetura simples e leve de ser executada, não necessitando de um hardware poderoso ou instalações de softwares pesados.

A usabilidade da interface gerada pela arquitetura foi desenvolvida com inspiração no conceito amigável, utilizando técnicas de UX/UI, numa tela todo controle e gestão do voo. Opções disponíveis através de um click, sem obrigatoriedade de configurar parâmetros, monitoramento do drone através de um GIF com hélices giratórias e atualização em tempo real de respectiva localidade no mapa.

Por fim, conclui-se que a escolha e aplicabilidade das diversas bibliotecas NPM presentes nesse trabalho mostraram-se devidamente apropriadas. Integradas numa única arquitetura, abstraindo milhares de linha de código e muitas horas de desenvolvimento, isso pensando se caso fosse necessário implementá-las. Principalmente a comunicação e troca de comandos entre arquitetura e AR.Drone, além dos cálculos de longitude e latitude para obter distância precisa, *bearing* e funções matemáticas.

#### 4.1 EXTENSÕES

Essa seção elenca futuras extensões ao trabalho, sendo elas:

- a) implementar base de dados utilizando Mongo *Data Base* (DB) banco de dados não relacional para armazenar dados de navegação;
- b) implementar o método disponível da API do Google Maps de disponibilização do mapa offline, baixando e salvando a rota no dispositivo;
- c) estender a arquitetura para também executar num aplicativo mobile, tornando-a multiplataforma;
- d) integrar a arquitetura e hardware do AR.Drone 2.0 com Arduino para executar um script de voo autônomo, rodando diretamente no drone, sem a necessidade de possuir uma conexão WI-FI ativa ao computador;
- e) estender o trabalho aplicando conceitos de inteligência artificial, utilizando outras bibliotecas NPM que estão disponíveis;
- f) implementar reconhecimento de imagens, fazendo com que o AR.Drone choque-se em objetos durante a execução do trajeto;
- g) estender o trabalho adaptando arquitetura e hardware para o mundo agrícola, desenvolvendo uma solução para o Agro 4.0, tal como mapeamento de plantações, ou aplicação de pesticidas;
- h) estender o hardware da arquitetura, substituindo o módulo de GPS original da Parrot, integrar um GPS de precisão de até 5cm da Ublox.

## REFERÊNCIAS

- AGUIAR, João Filipe Vieira. **Transferência de energia sem fios para carregamento de baterias**. 2013. 112 f. Dissertação (Mestrado integrado em Engenharia Eletrônica) - Curso de Engenharia Eletrônica Industrial e Computadores, Universidade do Minho, Portugal, Braga.
- ARRACHEQUESNE, Damien et al. **Socket.io**. 2020. Disponível em: <https://www.npmjs.com/package/socket.io>. Acesso em: 16 mai. 2020.
- BARROW, Erik. **Autonomous Navigation and Search in an Indoor Environment Using AR.Drone**. 2014. 118 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Centro de Ciências Exatas e Naturais, Coventry University, Coventry, UK.
- BIEH, Manuel. **Geolib**. 2020. Disponível em: <https://www.npmjs.com/package/geolib>. Acesso em: 08 jun. 2020.
- BRUNO, Fernanda. **Contramanual para câmeras inteligentes: vigilância, tecnologia e percepção**. *Galáxia*, n. 24, p. 47-63, 2012.
- CARTER, Carl. **Great circle distances**: Computing the distance between two points on the surface of the Earth. SiRF White Paper, n. 01, p. 02-05, 2002.
- DE FARIA, Rodrigo Ribeiro; COSTA, Marledo Egidio. **A inserção dos veículos aéreos não tripuláveis (drones) como tecnologia de monitoramento no combate ao dano ambiental**. *Revista Ordem Pública*, v. 8, n. 1, p. 81-103, 2015.
- ESCHENAUER, Laurent. **ardrone-autonomy**. 2014. Disponível em: <https://www.npmjs.com/package/ardrone-autonomy>. Acesso em: 03 abr. 2020.
- ESTÊVÃO, Tiago Vaz. **O Novo Paradigma da Vigilância na Sociedade Contemporânea-"Who Watches the Watchers"**. *OBServatorio (OBS)*, v. 8, n. 2, p. 155-169, 2014.
- GEISENDÖRFER, Felix et al. **ar-drone**. 2014. Disponível em: <https://www.npmjs.com/package/ar-drone>. Acesso em: 03 abr. 2020.
- HAHN, Evan. **Express in Action**: Writing, building, and testing Node.js applications. *Manning Publications*, v. 10, n. 1, p. 11-21, 2016.
- LIEDMAN, Per et al. **Leaflet**. 2020. Disponível em: <https://www.npmjs.com/package/leaflet>. Acesso em: 11 mai. 2020.
- GOOGLE, Maps JavaScript API. **Google**. 2020. Disponível em: <https://developers.google.com/maps/documentation/javascript/tutorial>. Acesso em: 12 jun. 2020.
- MONKEN, Maurício; BARCELLOS, Christovam. **Vigilância em saúde e território utilizado: possibilidades teóricas e metodológicas**. *Cadernos de Saúde Pública*, v. 21, p. 898-906, 2005.
- PISKORSKI, Stephane et al. **AR.Drone developer guide**. 2012. Disponível em: <https://forum.developer.parrot.com/t/ar-drone-2-0-sdk-for-android-on-windows/465>. Acesso em: 03 mar. 2020.
- RAHN, Rafael Ronaldo. **Estabelecimento de rotas para AR.Drone utilizando Delphi 10 Seattle**. 2016. 73 f. TCC (Graduação) - Curso de Ciência da Computação, Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.
- ROCHA, Marcus Otávio. **Furb-móble**: Sistema Móvel Multiplataforma para Navegação Em Rotas Internas. 2016. 61 f. TCC (Graduação) - Curso de Ciência da Computação, Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SANTANA, Lucas Vago; BRANDAO, Alexandre Santos; SARCINELLI FILHO, Mario. **Sistema para estimação e controle da posição 3D de um quadrimotor em ambientes internos**. 2016. 61 f. TCC (Pós-Graduação) - Curso de engenharia elétrica, Centro de Centro Tecnológico, Universidade Federal do Espírito Santo.

SCHAEFER (Blumenau). Assessora de Comunicação (Ed.). **Superávit da Oktoberfest é usado para compra de drone para a PM de Blumenau**. 2018. Disponível em: <http://oktoberfestblumenau.com.br/noticias/superavit-da-oktoberfest-e-usado-para-compra-de-drone-para-a-pm-de-blumenau/>. Acesso em: 13 abr. 2020.

SHIRATSUCHI, L. S. O avanço dos drones. **Embrapa Agrossilvipastoril-Artigo de divulgação na mídia (INFOTECA-E)**, v. 2, n. 3, p. 45-47, 2014.

TOMAN, David (Ed.). **Parrot AR.Drone 2.0 Review**. 2017. Disponível em: <https://www.ign.com/articles/2017/04/17/parrot-ar-drone-20-review>. Acesso em: 15 abr. 2020.

VANZ, José Guilherme. **VISEDU-DRONE: Módulo de Integração com Robot Operating System**. 2015. 89 f. TCC (Graduação) - Curso de Ciência da Computação, Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

WEISSHUHN, Bernhard et al. **Dronestream**. 2013. Disponível em: <https://www.npmjs.com/package/dronestream>. Acesso em: 15 abr. 2020.

WILSON, Douglas et al. **Express**. 2019. Disponível em: <https://www.npmjs.com/package/express>. Acesso em: 16 mai. 2020.