Universidade de Brasília
Faculdade de Tecnologia

Estudo e controle de Drones na inspeção de linhas de transmissão

Douglas Mauricio Bispo de Souza

TRABALHO DE GRADUAÇÃO

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Brasília

2023

Universidade de Brasília
Faculdade de Tecnologia

Estudo e controle de Drones na inspeção de linhas de transmissão

Douglas Mauricio Bispo de Souza

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Orientador: Prof. Dr. Walter de Britto Vidal Filho

Brasília
2023

Souza, Douglas Mauricio Bispo de.
S729e Estudo e controle de Drones na inspeção de linhas de transmissão / Douglas Mauricio Bispo de Souza; orientador Walter de Britto Vidal Filho. -- Brasília, 2023.
132 p.

Trabalho de Graduação (Engenharia de Controle e Automação) -- Universidade de Brasília, 2023.

1. VANT. 2. inspeção de linha de transmissão elétrica. 3. comunicação com AR.Drone 2.0. I. Filho, Walter de Britto Vidal, orient. II. Título

Universidade de Brasília
Faculdade de Tecnologia

Estudo e controle de Drones na inspeção de linhas de transmissão

Douglas Mauricio Bispo de Souza

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Trabalho aprovado. Brasília, 15 de fevereiro de 2023:

Prof. Dr. Walter de Britto Vidal Filho, UnB/FT/ENM
Orientador

Prof. Dr. Jones Yudi Mori Alves da Silva, UnB/FT/ENM
Examinador Interno

Prof. Dr. Carlos Humberto Llanos Quintero, UnB/FT/ENM
Examinador Interno

Brasília
2023

Agradecimentos

Primeiramente a Deus, pelo dom da vida e por interceder por nós. Obrigado, meu senhor, por me dar condições de chegar até aqui, auxiliando em minhas decisões, me dando coragem para enfrentar os desafios, e paciência e força para completá-los. Que o senhor continue olhando por nós e nos possibilite caminhar sempre em frente. Aos meus pais, Julio Ferreira de Souza e Nadir alves Maurício, por todo amor e dedicação e paciência que tiveram ao longo de minha vida, e principalmente durante este trabalho, vocês são o meu melhor presente, tudo que sou hoje eu devo a vocês, saibam que tenho muito orgulho de ser seu filho e os amo muito. As minhas irmãs, Julia Mauricio Bispo de Souza e Luiza Mauricio Bispo de Souza, por todo apoio e carinho. Um agradecimento especial a minha namorada Camila Fideles da Silva, que me acompanhou durante todo o trabalho e me deu suporte emocional, sendo companheira, amiga, paciente e por me ajudar a levantar nos momentos difíceis e incentivar a continuar lutando, sem você jamais teria terminado esta etapa do projeto. Agradeço a todos aqueles que de maneira direta ou indireta me ajudaram a superar os obstáculos dessa jornada.

"O sucesso nasce do querer,
da determinação e persistência em
se chegar a um objetivo.
Mesmo não atingindo o alvo,
quem busca e vence obstáculos,
no mínimo fará coisas admiráveis."
(José de Alencar)

Resumo

As linhas de transmissão de energia são elementos de importância fundamental para levar a energia do ponto de geração para todas as cidades do país. Uma falha nesse sistema leva a paralisação das atividades nas cidades afetadas, causando um enorme prejuízo. As empresas de geração de energia fazem inspeções periódicas das linhas de transmissão de forma a prevenir falhas. Um meio muito empregado é a inspeção aérea usando helicópteros. Esta atividade apresenta custo elevado e riscos consideráveis. Uma forma de evitar o uso destas aeronaves é o emprego de drones. O presente trabalho realizou o estudo de inspeções de linhas de transmissão e o desenvolvimento de um sistema computacional para comunicação e comando de drones para automatizar o processo de inspeção, com uma interface web que permite a definição de rotas e de trajetórias previamente programadas para realização da atividade em questão, além do acompanhamento de VANT em tempo real e gestão das imagens coletadas geo-posicionadas.

Palavras-chave: VANT. inspeção de linha de transmissão elétrica. comunicação com AR.Drone 2.0.

Abstract

Power transmission lines are elements of fundamental importance to take energy from the point of generation to all cities in the country. A failure in this system leads to the stoppage of activities in the affected cities, causing enormous damage. Power generation companies make periodic inspections of transmission lines in order to prevent failures. A very used means is the aerial inspection using helicopters. This activity has a high cost and considerable risks. One way to avoid the use of these aircraft is the use of drones. The present work studies transmission line inspections and develops a computational system for communication and command of drones to automate the inspection process, with a web interface that allows the definition of previously programmed routes and trajectories to carry out the inspection. activity in question, in addition to monitoring the UAV in real time and managing the geo-positioned images collected.

Keywords: UAV. electrical transmission line inspection. communication with AR.Drone 2.0.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

Aneel	Agência Nacional de Energia Elétrica . . . 15
API	Application Programming Interface . . . 32
ASCII	American Standard Code for Information Interchange . . . 35
CEMIG	Companhia Energética de Minas Gerais . . . 16
CMOS	Complementary metal–oxide–semiconductor . . . 38
DDR3	Double Data Rate 3 Synchronous Dynamic . . . 61
DHCP	Dynamic Host Configuration Protocol . . . 35
EDP	Energias de Portugal . . . 16
EKF	Extended Kalman Filter . . . 37
ENEL	Entidade Nacional de Eletricidade . . . 16
FPV	First Person View . . . 54
GPS	global positioning system . . . 32
HTTP	HyperText Transfer Protocol . . . 60
IAD-AERO	Sistema Autônomo-Cooperativo de Planejamento e Execução de Inspeção de Ativos de Energia . . . 16
ID	IDentify . . . 58
IMU	inertial measurement unit . . . 32
IP	Internet Protocol . . . 32
JSON	JavaScript Object Notation . . . 61
KLT	Kanade-Lucas Tomasi Tracker . . . 32
LTs	Linhas de Transmissão . . . 15
MEMS	Micro-Electro-Mechanical Systems . . . 38
NPM	Node Package Manager . . . 32
ONS	Operadora Nacional do Sistema Elétrico . . . 15
OSRF	Open Source Robotics Foundation . . . 37
PTAM	Extended Parallel Tracking and Mappin . . . 37
PWM	Pulse-Width Modulation . . . 32

QVGA Quarter Video Graphics Array 38

RAM Random-Access Memory 61

ROS Robot Operating System 37

RPA Aeronaves Remotamente Pilotadas 22

SDK Software development kit 32

SURF Speed-Up Robust Features 32

TCP Transmission Control Protocol 32

UAV unmanned aerial vehicle 22

UDP User Datagram Protocol 32

UDP User Datagram Protocole 60

UnB Universidade de Brasília 22

VAT Aveículo aéreo não tripulado 22

Sumário

1 Introdução

1.1 contextualização

As linhas de transmissão de energia são elementos de importância fundamental para na sociedade atual, sendo responsável por levar energia do ponto de geração para todas as cidades do país. Segundo dados da ONS (Operadora Nacional do Sistema Elétrico), em 2021, o Brasil possuía quase 170 mil km de LTs (Linhas de Transmissão) com tensão maior ou igual a 230 kV, com projeção de em 2026 este valor chegue próximo a 202 mil km (ONS (2022)).

Uma falha nesse sistema leva a paralisação das atividades nas cidades e indústrias afetadas, causando um enorme prejuízo. Para garantir a qualidade e a confiabilidade desse sistema todas as atividades do setor elétrico são fiscalizadas e regulamentadas pela Aneel (Agência Nacional de Energia Elétrica) que estabelece de a cordo com a resolucao normativa nº 906 RESOLUÇÃO NORMATIVA Nº 853, DE 8 DE DEZEMBRO DE 2020 estabelece como plano mínimo de manutenção, inspeção de rotina com um período de, no mínimo, doze meses. Podendo as concessionárias criar um plano de inspeções e manutenções preventivas e preditivas de acordo com as características de suas estruturas e região de atuação respeitando a periodicidade mínima (Brasil (2020)).

A atividade de inspeção de linhas de transmissão, por padrão possui um elevado custo financeiro e de tempo, além riscos consideráveis, sendo a vida dos profissionais constantemente colocada em risco tanto que, inspetores de linhas de transmissão ocupam o nono lugar no terrível ranking das profissões mais perigosas dos EUA de acordo com Force (2017)

Devido a esse cenário, o uso de Drone em inspeções de LTs tem crescido bastante nos últimos anos, buscando principalmente o aumento da produtividade, por permitir um número maior de inspeções por conseguirem realizar o trabalho de forma muito mais rápida em razão de sua praticidade e operacionalidade, da qualidade, já que ele envia fotos e vídeos em ângulos privilegiados, da segurança, já que não há a presença física, apenas do equipamento. Além de reduzir os custos de serviços, tanto para inspeções de rotina quanto para levantamento de danos nas redes de energia após desastres naturais e melhorar a segurança dos trabalhadores envolvidos diretamente na inspeção.

Um exemplo disso é que em 2018, a empresa Energisa, que atua em oito estados do Brasil, concluiu um projeto piloto para usar os veículos aéreos não tripulados nas inspeções das suas redes e linhas de distribuição (PEDRO (2018)). A empresa EDP (Energias de Portugal) recebeu no dia 10 de junho de 2021, a certificação da ANAC para operar comercialmente o sistema SIAD-AERO (Sistema Autônomo-Cooperativo de Planejamento e Execução de

Inspeção de Ativos de Energia) que contempla o uso de drones como veículo para inspeção autônoma/assistida, substituindo, assim, a atividade humana e manual, a companhia estima monitorar de 30 a 40 mil quilômetros de redes e seus respectivos ativos de energia em um ano com o empregado do sistema (CANALENERGIA (2021)).

Outros projetos como o da empresa CEMIG (Companhia Energética de Minas Gerais) com objetivo de gerar inspeções autônomas por meio de drones com o desenvolvimento de um algoritmo de rotas inteligentes, que guiará o VANT na extensão das linhas e redes para coleta de imagens usando geoprocessamento e visão computacional e de um software, integrado ao Centro de Operações e ao VANT(30). E o da empresa ENEL(Entidade Nacional de Eletricidade), que utiliza drones e tecnologia Horus em inteligência artificial embarcada para inspeções em tempo real de linhas de transmissão,reduzindo o tempo de inspeção em 70%, demonstram que o mercado está bastante aquecido.

Com isso este trabalho propõe um sistema para inspeção de LTs com drone um drone comercial de baixo custo de forma autônoma, dispensando a necessidade de um piloto e automatizando ainda mais o processo .

1.2 Objetivo geral

Estudar a inspeção de torres de transmissão e propor um sistema computacional para comunicação e comando de Drones, que possam permitir o desenvolvimento de uma aplicação para inspeções das linhas de transmissão elétrica, de forma autônoma e mais eficiente se comparado a métodos tradicionais. Permitindo a definição de trajetórias e gestão das imagens coletadas geo-posicionadas.

1.3 Objetivos Específicos

Serão abordados os seguintes objetivos específicos:

- Identificar os elementos a serem verificados em uma inspeção de torre de transmissão elétrica.
- Desenvolver um software que permita o planejamento de rotas para inspeção, e envie comandos para um drone comercial em campo através de um computador.
- Estabelecer comunicação com um Drone comercial para a implementação de controle de trajetória
- Avaliar os requisitos necessários para realizar a inspeção

2 Fundamentação Teórica

2.1 Linhas de transmissão

A rede de energia elétrica é composta basicamente pelas atividades de geração, transmissão , distribuição e comercialização . Na parte de geração estão as usinas e as subestações elevadoras, que elevam o nível da tensão e abaixam o nível da corrente, com a intenção de facilitar o transporte da energia elétrica.

Esse transporte é realizado pela LTs, que se estendem por longas distâncias, conectando usinas geradoras aos grandes consumidores e para as empresas distribuidoras de energia, que são responsáveis por levar energia aos consumidores de menor porte.

Elas podem ser linhas de corrente alternada ou contínua, sendo que as de corrente alternada costumam ter como tensões de transmissão os valores de 230 kV, 345 kV, 440 kV, 500 kV e 750 kV. Enquanto as de corrente contínua possuem valores de 600 kV e 800 kV (ONS (2022)) Apesar das linhas de transmissão em corrente contínua apresentarem um custo menor se comparado às linhas em corrente alternada, as estações conversoras apresentam custo relativamente alto, elevando os gastos da transmissão em corrente contínua, se mostra vantajosas apenas em aplicações específicas, como na interligação de sistemas com frequências diferentes ou para transmissão de energia a distâncias acima de 600 km(Mattede (s.d.))

As LTs são compostas por:

1. Ferragens : manilhas, cavaletes ou mancas;
2. cabos condutores, revestidos por camadas isolantes;
3. Cadeia de isoladores, que podem ser de vidro ou porcelana, e circundam e sustentam os cabos, impedindo descargas elétricas durante o caminho, prevenindo acidentes e minimizando custos de perdas, por exemplo;
4. Pode conter amortecedores;
5. Espaçadores
6. Torres. Grandes estruturas metálicas
7. Cabos guarda (também conhecido como cabo para-raios), fixados no topo das torres para proteger a linha de descargas elétricas de raios

Essas componentes podem ser visualizados, assim como sua disposição no sistema, na Figura 1 e Figura 2

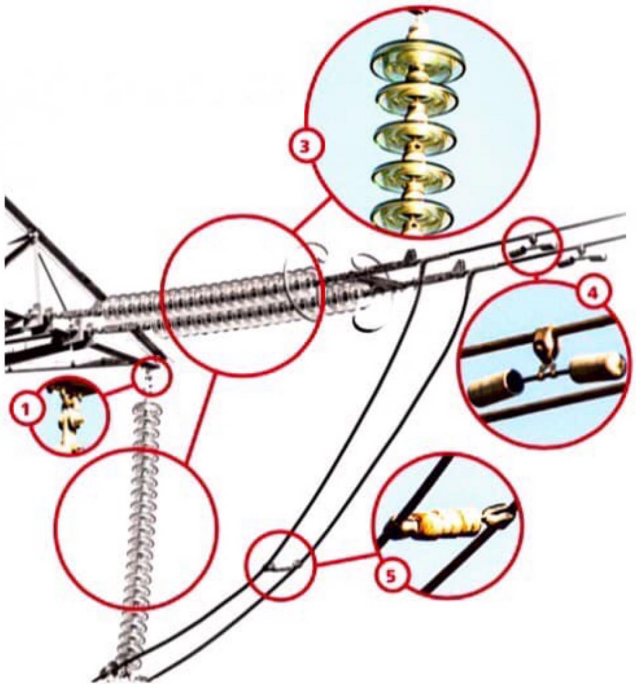


Figura 1 – Componentes de uma linha de transmissão

Fonte: Adaptado de BRAMETAL (2021)

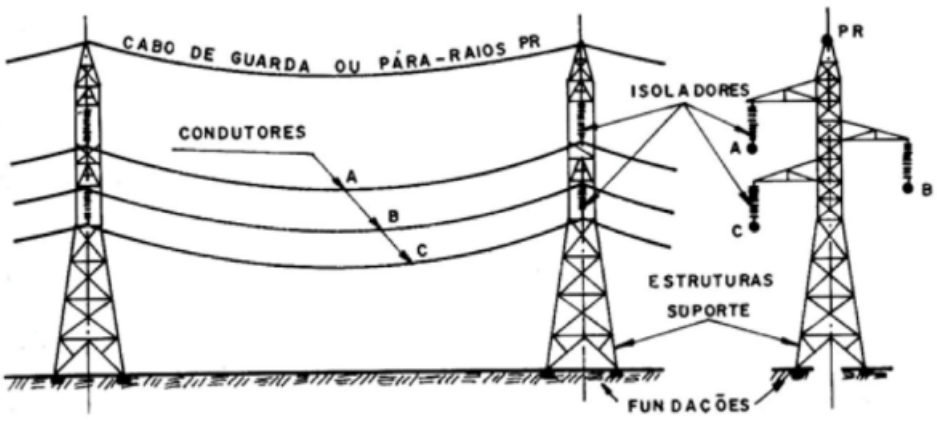


Figura 2 – Restante dos componentes de uma linha de transmissão

Fonte: Adaptado de Labegalini et al. (1992)

2.2 Inspeção de Linhas de transmissão

As inspeções serão responsáveis por detectar falhas ou defeitos nas linhas ou equipamentos, para que se possa efetuar as manutenções (preventivas, preditivas ou corretivas)

constatadas ou previstas pelas inspeções, de forma a evitar falhas maiores no sistema. Ao se realizar uma inspeção de rotina deseja-se verificar o estado geral da linha de transmissão, a integridade dos cabos condutores e pára-raios, a estabilidade das estruturas, a integridade das cadeias de isoladores, a situação dos acessos às estruturas, a proximidade da vegetação aos cabos e os casos de invasão de faixa de servidão, que é a faixa de terra necessária na construção, operação e manutenção das LTs.

As metodologias de manutenção que diversas instituições elétricas estão utilizando para manter sob controle os indicadores de desempenho do processo de transmissão, envolve três principais abordagens: patrulhas terrestres (Figura 3a), veículos aéreos tripulados (Figura 3b) e inspeções com escalada (Figura 3c).

(a) inspeção terrestres

Fonte: MP (s.d.)

(b) inspeção aérea

Fonte: VoeSP (s.d.)

(c) inspeção com escalada

Fonte: Alves (2018)

Figura 3 – Tipos de inspeções de LTs

Nas patrulhas terrestres, Figura 3a, a equipe vai de torre em torre, se utilizando de veículos terrestres (em alguns trechos pode ser necessário se fazer a pé devido ao terreno), realizando inspeção visual com a ajuda de binóculos e câmeras. Tal inspeção pode ser altamente precisa, mas apenas para as superfícies dos equipamentos de linha de energia que

podem ser bem vistas do solo, além de ser demorada.

As que utilizam veículos aéreos tripulados, geralmente utilizam helicópteros, como na Figura 3b, que pairam sobre as linhas de forma a se obter informações mais precisas. Os técnicos geralmente estão munidos de binóculos e câmeras, estas também podem estar instaladas na aeronave para filmar toda a LT. A vantagem desse método é a possibilidade de cobrir grandes distâncias em um único voo e acessibilidade a áreas de difícil acesso ou locomoção por terra. Porém, como já mencionado, esse método possui um alto custo e dependência de equipes treinadas para guiar a aeronave, além do grande risco envolvido pela necessidade de voar próximo da linha.

Inspeções com escalada geralmente ocorrem com dois técnicos escalando cada um um lado da torre para uma investigação mais minuciosa das torres e dos isoladores, como visto na Figura 3c. Entretanto, esse método traz grande risco a vida dos trabalhadores além de demandar muito mais tempo em relação às outras abordagens.

2.3 Drone

“Drone” é uma palavra da língua inglesa que pode ser traduzida para o português como zangão ou zumbido. É usada comercialmente para denominar de forma genérica veículos aéreos não tripulados. Quando usados para fins recreativos e são de pequeno porte são denominados, pela ANAC, como aeromodelos. Drones com fins não recreativos recebem o nome de Aeronaves Remotamente Pilotadas (RPA) (ANAC (s.d.)). O nome técnico de VANT (veículo aéreo não tripulado) ou como "UAV"(unmanned aerial vehicle) também é muito usado no meio acadêmico.

Tendo sua origem inspirado pela bomba V-1 (Figura 4), criada na Segunda Guerra Mundial, que voava em linha reta e em velocidade constante, as quais possibilitam o alcance do alvo a grandes distâncias. Os Drones surgiria por volta dos anos 60, popularizados nos anos 80, por efetuar ações perigosas, sem necessariamente colocar uma vida em risco em diante de um contexto militar(Buzzo (2015)).

Figura 4 – Gráfico produzido em Excel e salvo como PDF

Fonte: Wikipédia (s.d.)

Com o avanço da tecnologia se tornou possível a existência de diversos modelos desses veículos sendo eles classificados de forma mais abrangente como veículos de asa rotativa (helicópteros), multirotores (quadricópteros, octacópteros, hexacópteros) ou asa fixa (avião) (Neto (2017)).Cada tipo de Drone possui características distintas que devem ser consideradas para a escolha do modelo para a aplicação desejada, um exemplo desses modelos podem ser vistos na Figura 5 .

Figura 5 – Classificação do VANT de acordo modelo da asa.

Fonte: Adaptado de RANGEL (2019)

Apesar do intuito de sua origem, atualmente, eles são empregados em diversas áreas além de recreação e hobby como: monitoramento de fronteiras (Caiafa (2020)), auxílio em plantações (Emater (2020)), inspeção de linhas elétricas (Maia (2022)), combate a incêndio (Solution (2021)), cinegrafia, fotografia, entre outros. No futuro se espera poder usar esse equipamento para se fazer entrega de produtos como por exemplo indicado no pronunciamento da empresa norte-americana Amazon (Hawkins (2022)).

Para o desenvolvimento deste trabalho foi decidido utilizar um multirotor, mas especificamente um quadricóptero, devido a facilidade de operação, manobrabilidade, sendo esse capaz de mover-se em todas as direções e girar em torno do seu próprio eixo, além da sua possível operações em espaços limitados, sendo capazes de decolar e pousar em espaços muito menores, e capacidade de operar com baixas velocidades ou completamente parados no ar. Porém existe uma limitação significativa referente autonomia devido a utilização de baterias e o gasto de energia para sua sustentação.

2.4 Princípio de Funcionamento do Quadrimotor

O requisito principal para realizar uma decolagem em uma aeronave do tipo multicóptero é a geração de sustentação através dos motores rotativos. A sustentação é gerada pelo movimento rotacional do conjunto de hélices acoplado no eixo do motor, e este movimento imerso em um fluido compressível, como o ar, gera uma força de empuxo, utilizada para gerar sustentação. Além disso, ocorre a geração de uma força de arrasto contrária ao movimento da aeronave (Homa (2010)).

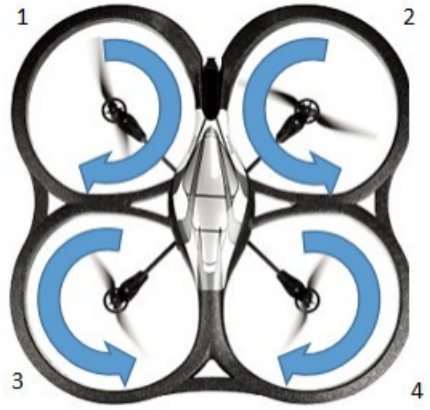


Figura 6 – Sentido de rotação dos motores

Fonte: Adaptado de Kaharovic (2019)

Devido a força de arrasto gerada pela hélice, o motor será submetido a um torque que fará sua base girar em sentido contrário à rotação da hélice, fazendo toda a estrutura da aeronave girar. Por isso, os motores giram em sentidos contrários aos pares e estão distribuídos simetricamente em torno do corpo do veículo, podendo estar em configuração de x ou +, para que ocorra o equilíbrio do momento,como mostrado na Figura 6, isso faz com que qualquer movimento no espaço tridimensional necessite da ação em conjunta de todos os motores, alterando os ângulo de arfagem (pitch), ângulo de rolagem (roll) e ângulo de guinada (yaw) para se manobras Drone. Tais ângulos são ilustrados na Figura 7.

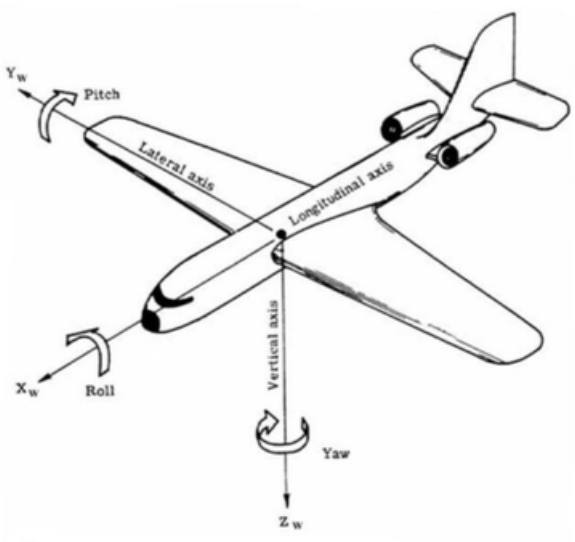


Figura 7 – Ângulos de pitch, roll e yaw

Fonte: Adaptado de Piskorski et al. (2012)

Dessa forma, para que se possa compreender seu funcionamento será ilustrado na Figura 8 os movimentos básicos de voo de um quadrirotor em configuração X, destacando quais motores deverão ter sua velocidade angular modificada para gerar força correspondente a cada movimento. A cor das setas dentro do VANT indica a intensidade de giro de cada rotor, as setas azuis que estão fora do corpo do Drone indicam o movimento que está sendo realizado. Para que o Drone realize o movimento de guinada (giro em torno do eixo z) no sentido anti-horário deve-se diminuir a velocidade dos motores 1 e 3 enquanto aumenta a velocidade dos motores 2 e 4, como mostra a Figura 8(a). O contrário ocorre quando se inverte a escolha dos motores, diminuindo a velocidade dos motores 2 e 4 enquanto aumenta a velocidade dos motores 1 e 3, isso faz o Drone realizar o movimento de guinada só que agora no sentido horário mostrado na Figura 8(b). Para fazer o Drone se movimentar para direita ou esquerda é necessário realizar a manobra de rolagem (giro em torno do eixo x), dessa forma, deve-se diminuir a velocidade dos motores 3 e 4 e aumentar a velocidade dos motores 2 e 1 fazendo o Drone realizar a rolagem no sentido horário mostrada na Figura 8(c), de maneira análoga, ao diminuir a velocidade dos motores 2 e 1 e aumentar a velocidade dos motores 3 e 4 ele realizará o movimento de rolagem no sentido anti-horário, conforme demonstrada na Figura 8(d). Para que o Drone decole ou aterrisse (movimento de subida e descida ao longo do eixo z) é preciso o acionamento de todos os motores simultaneamente, assim se aumentar igualmente a velocidade de todos os motores ele irá subir, da mesma forma ao se reduzir igualmente a velocidade de todos os motores o mesmo irá descer, como mostra a Figura 8(e) e Figura 8(f). Para fazer o Drone se deslocar para frente(sentido positivo do x) é necessário realizar o movimento de arfagem no sentido anti-horário (giro em torno do eixo y), para isso deve-se aumentar a velocidade dos motores 2 e 3 enquanto se diminui a velocidade dos

motores 1 e 4, Figura 8(g). Por fim, para que o Drone se mova para trás (sentido negativo do x) é realizado o movimento de arfagem no sentido horário, aumentando a velocidade dos motores 1 e 4 enquanto se diminui a velocidade dos motores 2 e 3, Figura 8(h).

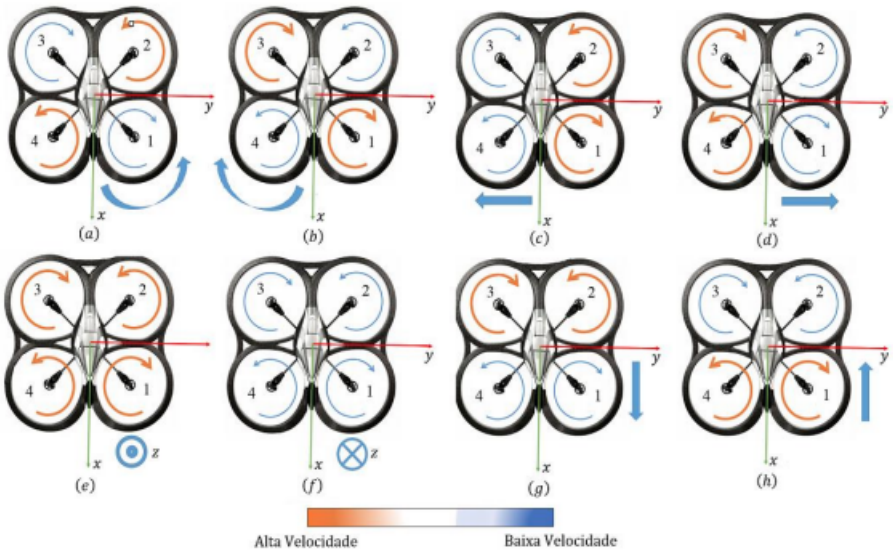


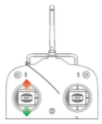
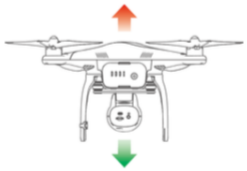
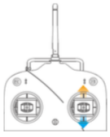
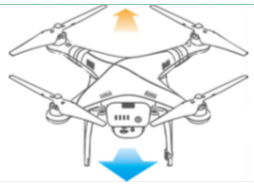
Figura 8 – Discrição dos movimentos basicos de voo de um quadrirotor em x

Fonte: Adaptado de Brandao (2013)

Geralmente, os Drones possuem mecanismos de navegação automática com a finalidade de auxiliar o operador do veículo a controlar seus movimentos, através de comandos simples e intuitivos originados em um rádio transmissor ou aparelho similar. Assim o piloto automático fica responsável por todos os comandos de baixo nível, no caso específico dos VANTs de pás rotativas, significa realizar procedimentos de decolagem, pouso, controle da atitude e dos angulos maximos de arfagem ,rolagem e guinada, controlando a velocidade das manobras. Usualmente há um parâmetro de configuração para velocidade angular ou angulação máxima que é atingida. gerando assim, uma padronização da pilotagem para todos os veículos multirrotores, conforme resumido na Tabela 1

Tabela 1 – Esquema geral de pilotagem de VANTs de pás rotativas equipados com piloto automático.

Fonte:Adaptado de Santana (2016)

Comando de Controle	VANT	Função Executada
		Propulsão vertical: utilizada para ganhar ou perder altitude, conforme indicação na imagem.O veículo mantém sua altitude se o joystick estiver no centro.
		Guinada: utilizada para controle de leme, ou orientação de guinada em sentido horário e anti-horário, conforme indicação da imagem. O veículo mantém sua orientação se o joystick estiver no centro.
		Rolagem: utilizada para controlar ângulo de rolagem. O veículo mantém esta orientção em 0° se o joystick estiver no centro. Esta inclinação ocasiona movimento lateral do veículo.
		Arfagem: utilizada para controlar ângulo de arfagem. O veículo mantém esta orientção em 0° se o joystick estiver no centro. Esta inclinação ocasiona movimento longitudinal do veículo.

3 Revisão Bibliográfica

Os Drones têm se tornado cada vez mais populares e presentes na nossa sociedade, devido ao avanço da tecnologia e o grande número de empresas produzindo tal equipamento, o que o torna mais acessível. Com isso é grande o número de pesquisas e trabalhos relacionados aos VANTs e suas aplicações. Este capítulo expõe trabalhos relacionados ao dispositivo e a inspeção de linhas de transmissão.

3.1 Na UnB

Somente na UnB (Universidade de Brasília) existem pelo menos 5 trabalhos relacionados ao desenvolvimento dos aeromodelos.

Os trabalhos vão desde a montagem de pequenos Drones quadrirrotores (Teixeira (2018)), passando pelo desenvolvimento de novas formas de manobrar um Drone comercial(Syma X5SW), como visto em Pantoja e Garcia (2017) e Silva e Libardi (2016), aprimoramento de controle do Drone comercial (AR.Drone 1.0) em Lima (2013) e desenvolvimento de plataformas para que o aparelho em questão possa realizar tarefas de forma autônoma (Mesquita (2015)).

Em Teixeira (2018) o quadrirrotor foi desenvolvido com o objetivo de realizar voos autônomos a partir de trajetórias pré-definidas no software Mission Planner. O trabalho foi feito a partir da placa controladora Ardupilot baseada em Arduino Mega 2560 conectada a controladores eletrônicos de velocidade, responsáveis por fazer a associação dos sinais PWM (Pulse-Width Modulation) e a potência elétrica que deve ser fornecida para cada motor, a sensores necessários para realização do voo autônomo como GPS, acelerômetro, giroscópio e bússola, além de um receptor radiofrequência para o controle manual do Drone, o esquemático de montagem do protótipo pode ser visto na Figura 10. Também foi utilizado um filtro de kalman para fazer a união dos dados de posição e velocidade oriundas do GPS e dos sensores para se estimar a posição e velocidade do VANT com maior precisão.

Figura 9 – Protótip desenvolvido por Teixeira (2018)

Fonte: Teixeira (2018)

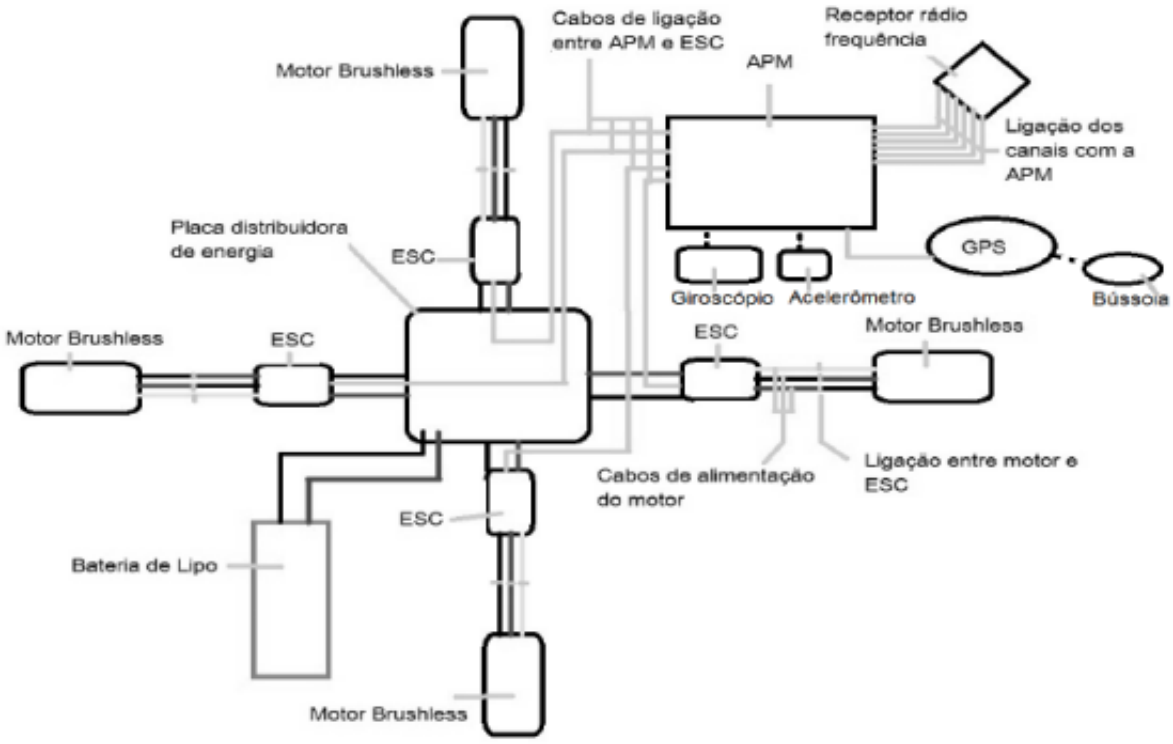
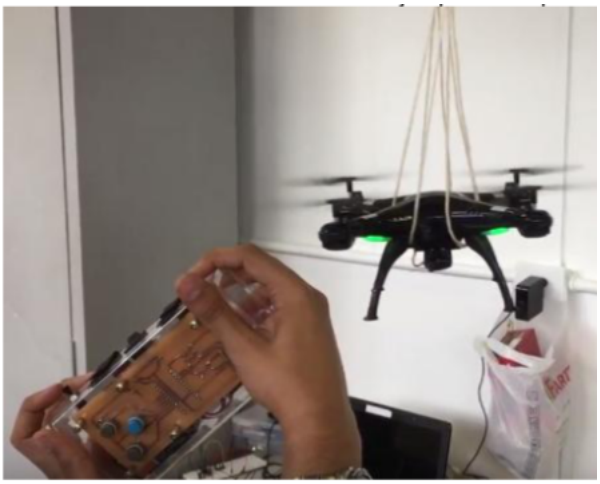


Figura 10 – Esquemático de montagem do protótip desenvolvido por Teixeira (2018)

Fonte: Teixeira (2018)

Tanto o trabalho Pantoja e Garcia (2017) quanto o Silva e Libardi (2016) utilizaram o drone comercial Syma X5SW, originalmente controlado por um controle de radiofrequência, para desenvolver um joystick sensível a rotações nas três dimensões. Esses movimentos foram traduzidos em informações que atuam no VANT buscando tornar a pilotagem do mesmo mais intuitiva para um piloto aprendiz. Ambos os trabalhos adaptaram a placa do joystick original para que fosse controlada por um Arduino, o qual recebe comandos via Bluetooth

do novo joystick, sendo que no trabalho Pantoja e Garcia (2017) esté se utiliza de uma IMU (MPU-6050), um Arduino para o processamento dos dados e um módulo de comunicação Bluetooth para se enviar sinais. Enquanto isso, o trabalho Silva e Libardi (2016) desenvolveu um aplicativo para a plataforma Android, utilizando o software Android Studio, que envia os sinais de controle para o arduino conectado a placa de controle original, possibilitando ao usuário decidir por controlar o veículo pela touchscreen ou utilizando o giroscópio do dispositivo. As imagens de ambos trabalhos podem ser visualizadas na Figura 11



(a) plataforma inercial e o dorne trabalho de Pantoja e Garcia (2017)

Fonte: Pantoja e Garcia (2017)

(b) Sistema final: plataforma de controle, aplicativo Android e o drone do trabalho de Silva e Libardi (2016).

Fonte: Silva e Libardi (2016)

Figura 11 – Imagens dos trabalhos de Pantoja e Garcia (2017) e Silva e Libardi (2016)

Já os outros trabalhos, se utilizam do Ar-Drone 1.0, onde testes de campo demonstram que o VANT apresenta instabilidades até mesmo em ambientes fechados com poucas perturbações (como o vento, por exemplo) devido ao controle embarcado não ser tão bem elaborado. Com isso, o trabalho Lima (2013) propõe um sistema de controle por realimentação, no qual os dados são obtidos através do processamento das imagens fornecidas pela câmera vertical do Drone, onde a malha de controle atua sobre o controle de estabilidade embarcado

já presente no Ar-Drone 1.0 para melhorar a confiabilidade do controle de rolagem e arfagem do modelo. O algoritmos de estimação de movimentos utilizado neste trabralho é mostrado na Figura 12

(a) LK-Piramidal FarneBack (Instante 1)

(b) LK-Piramidal FarneBack (Instante 2)

Figura 12 – Algoritmos de estimação de movimentos implementados no quadrirrotor do trabalho de Lima (2013)

Fonte: Lima (2013)

A malha de controle externa está presente em um computador que se conecta ao Drone através do wi-fi e se utiliza do software AutoPylot que pode ser rodado em alto nível com um script de Python, MATLAB, ou na linguagem C, para receber dados e enviar comandos (Lima (2013)). Esse software se utiliza da SDK fornecida pela empresa fabricante do modelo. Além da biblioteca open source de ferramentas e algoritmos de processamento de imagem OpenCV que também está disponível em C, C++ e Python .

Já o trabalho Mesquita (2015) tem como objetivo principal criar algoritmos de visão computacional com o objetivo de identificar um local para pouso de um quadrirrotor se utilizando também de um AR.Drone. Esse projeto se utilizou do AR DRONE SIMULINK DEVELOPMENT KIT, que consiste de blocos no SIMULINK que estabelecem a possibilidade de conexão em tempo real via Wi-Fi e simulação da dinâmica do VANT. Projetando controladores por alocação de pólos e zeros através do LGR (Lugar Geométrico das Raízes) e com resposta DeadBeat através de simulação no MATLAB. Para a tarefa de identificar um alvo através de imagens, foram elaborados dois códigos independentes, para ambientes escuros baseado na segmentação por cor, e outro para ambientes claros, baseado no reconhecimento de padrões através de extração e descrição de características com algoritmos SURF (Speed-Up Robust Features), juntamente com o KLT(Kanade-Lucas Tomasi Tracker) , capaz identificar e rastrear um objeto por um vídeo, tomando como referência a imagem do local de pouso, a Figura 13 mostra o local de pouso para ambos os ambientes.

Apesar de ter conseguido bons resultados em teste feito com os algoritmos de visão

computacional, eles não foram utilizados para controlar a rota de pouso do Ar.Drone, sendo a validação dos controladores projetados em relação a planta pura feita apenas em simulação no MATLAB.

(a) Local de pouso usado no trabalho de Mesquita (2015).

(b) Local de pouso no escuro identificado com marcação de seu centro usado no trabalho de Mesquita (2015).

Figura 13 – Imagens dos trabalhos de Pantoja e Garcia (2017)

Fonte: Mesquita (2015)

3.2 No âmbito nacional

Fora do âmbito da Universidade de Brasília existem diversos trabalhos relacionados também ao AR.Drone. Para este trabalho, destaca-se o trabalho Correa (2020), onde foi desenvolvido uma arquitetura de voo autônomo aplicada no modelo AR.Drone 2.0 devido a sua compatibilidade com o módulo GPS da própria Parrot, o qual não existe no modelo 1.0., que consistia em gerenciar percursos de rotas preestabelecidas, visando principalmente aplicabilidade de vigilância aérea em espaços externos, utilizando uma interface web para cadastrar novas bases e selecionar no mapa o percurso desejado para o drone realizar a vigilância do local . Além disso, oferece recursos para percorrer tais rotas de forma autônoma disponibilizando os dados nela registrados e em tempo real.

O desenvolvimento foi realizado utilizando o framework Node.js, que roda a linguagem JavaScript, integrado a bibliotecas NPM como node-ar-drone, ardrone-autonomy e Geolib. Para a seleção do percurso e coleta das coordenadas foi utilizada a biblioteca Leaflet e API do Google Maps.A interface web da aplicação é vista na Figura 14, onde a esquerda se tem a rota traçada pelo usuário e os botões de comando, e a direita a imagem da câmera frontal do drone em tempo real juntamente com dados do Drone

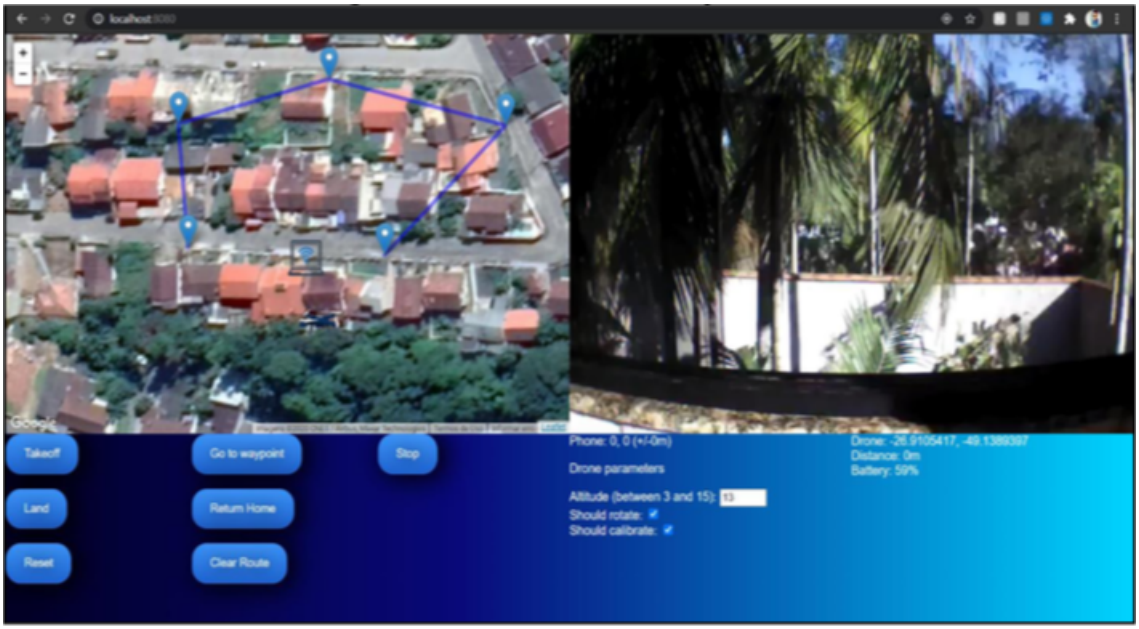


Figura 14 – Interface web da arquitetura do trabalho de Correa (2020).

Fonte: Correa (2020)

Não só drones são usados na inspeção de LT's, robôs que deslizam nos cabos também foram usados, com o que foi desenvolvido no trabalho Ventrella et al. (2003), que consiste em um prototipo de robô móvel, que se desloca ao longo do cabo guarda e é capaz de realizar inspeções visuais por câmeras em linhas de transmissão, proporcionando ao operador do robô uma visualização adequada tanto dos cabos guarda como dos cabos condutores.

O robo possui uma estrutura fisica capaz de traspor as torres, possuindo 3 rodas como sistema de tração, que basculam no momento em que for iniciado o procedimento de ultra-passagem de torres, e abaixo deste sistema de locomoção existe um fuso que é responsável pelo balanceamento das massas, também durante o procedimento de transposição de torres, essa estrutura é ilustrada na Figura 15.

Figura 15 – Foto do robô de inspeção de linhas de transmissão.

Fonte: Ventrella et al. (2003)

O robô é controlado por um pc104 situado dentro da caixa inferior que gerencia todos os acionamentos dos atuadores a partir de comandos recebidos de um no notebook em solo. Essa comunicação é feita através do protocolo TCP/IP, em uma rede local formada por um único par de rádios de comunicação, para envio de comandos e recebimento das imagens e dados dos sensores. Com o objetivo de aumentar o alcance do rádio, foi instalado no robô uma antena de 6dB e na base de operações uma antena de 18dB. Nos experimentos realizados em laboratório com a utilização de atenuadores foi possível estimar o alcance dos rádios de 5Km.

O sistema de visualização é composto por duas câmeras IP, uma fixa localizada sobre a roda da frente, e outra móvel fixada na caixa inferior com movimentos de tilt, pan e zoom de 16 vezes. A primeira câmara tem a função de fazer o monitoramento exclusivo do cabo guarda, enquanto a segunda, de alta resolução, destina-se à inspeção dos separadores existentes entre os cabos condutores de energia. Mas também não se descarta a utilização da mesma para observação dos próprios cabos condutores. A transmissão de dados é feita por meio de rede local sem fio, através de uma comunicação bidirecional, pois, além de enviar os sinais de vídeo ao operador, a câmera também recebe sinais de comando para a movimentação do mecanismo de tilt-pan e para utilização do zoom.

Nesta ferramenta cabe ao operador a função de continuamente observar as imagens capturadas a partir do computador em solo para determinar a posição de eventuais defeitos e solicitar as correspondentes operações de reparação dos cabos e/ou dos separadores.

4 Fundamentação Técnica

4.1 Descrição do drone utilizado

O uso de drones pela sociedade tem se tornado cada vez mais comum devido ao grande número de empresas comercializando tais produtos de diversos tamanhos e modelos, não sendo necessário a construção de um desses veículo para fins de pesquisa e teste, veículos comerciais são facilmente substituíveis além de apresentarem uma grande comunidade de usuários e pesquisadores, facilitando a reutilização de ferramentas, reposição de peças e desenvolvimento de novas aplicações.

O VANT adotado nesse trabalho foi o AR.Drone 2.0, da fabricante francesa Parrot e ilustrado na Figura 16, sendo este bastante conhecido no meio acadêmico graças ao seu baixo custo e a empresa disponibiliza o Source Code do AR.Drone bem como o SDK, Software Development Kit, ou simplesmente Kit de Desenvolvimento de Software, em português

Figura 16 – AR.Drone 2.0

Fonte: Santana (2016)

4.2 Composição do Drone

Ele possui quatro motores sem escova (brushless) com corrente trifásica que são usados para controlar as quatro hélices fixas, Cada motor é montado em pés de plástico resistentes carregando a placa de controle. Cada placa de controle usa seu próprio microcontrolador e sistema de segurança para desligar o motor no caso de um obstáculo entrar no

caminho da hélice para evitar choques repetidos. Juntos, os motores constituem o par de geradores de impulso variáveis (Santana (2016)).

Sua alimentação é feita por baterias LiPo carregadas de 1000mAh e 11,1V para voar. Durante o voo, a tensão da bateria diminui de carga total (12,5 Volts) para carga baixa (9 Volts). O AR.Drone 2.0 monitora a tensão da bateria e converte essa tensão em uma porcentagem de vida útil da bateria (100% se a bateria estiver cheia, 0% se a bateria estiver fraca). Quando o drone detecta uma baixa voltagem da bateria, ele primeiro envia uma mensagem de aviso ao usuário e, em seguida, aterrissa automaticamente. Se a tensão atingir um nível crítico, todo o sistema é desligado para evitar qualquer comportamento inesperado (Piskorski et al. (2012)). A capacidade da bateria possibilita um voo de 10 a 15 minutos de duração. A estrutura principal é fixada em cima de uma espuma, que filtra as vibrações dos motores. Duas proteções podem ser conectadas ao sistema. A primeira é uma capa removível que pode ser usado para evitar que as hélices sejam danificadas em caso de incidente, a segunda não fornece proteção (Santana (2016)).

Ele possui placas eletrônicas de gerenciamento de energia e rede de comunicação Wi-Fi, através da qual o AR.Drone recebe seus comandos de movimento. conectados ao computador de bordo com processador Cortex A8 de 1,00 GHz e 1GB de memória RAM, rodando um sistema operacional Linux, que também é responsável pela aquisição do fluxo de dados das duas câmeras sendo uma frontal e outra vertical. As quais são codificadas e transmitidas automaticamente para o dispositivo host (Santana (2016)).

A câmera frontal é um sensor CMOS com uma lente de ângulo de 90 graus, sendo uma câmera HD filmando a 720p em 30fps .A câmera voltada para baixo é uma câmera QVGA(320*240) 60fps. As imagens desta câmera serão aumentadas para 360p ou 720p para *streaming* de vídeo. Sendon assim, ambas as câmeras (com *upscaling* da câmera voltada para baixo) podem ser configuradas para transmitir imagens de 360p (640*360) ou 720p (1280*720), podendo a taxa de quadros do fluxo de vídeo ser ajustada entre 15 e 30 FPS. A câmera vertical também é usada nos algoritmos de navegação, para medir a velocidade do veículo. Esse algoritmo também se utiliza de sensores de movimento localizados abaixo do casco central, sendo esses uma unidade de medição inercial (IMU) miniaturizada de 9 DOF (3 a mais que a versao 1.0), baseada em MEMS (Micro-Electro-Mechanical Systems), com um acelerômetro de 3 eixos, um giroscópio de 3 eixos e um magnetômetro de 3 eixos adicionando nessa verssao 2.0. Além de um telémetro de ultrassom e um sensor de pressão para permitir medições de altitude em qualquer altura, possibilitando ao algoritmo estabilização automática de altitude e controle de velocidade vertical assistido (Piskorski et al. (2012)).

4.3 Comunicação

Uma das características levada em consideração para a escolha desse quadrirrotor é a sua facilidade de integração com outros dispositivos, bastando que o outro dispositivo suporte uma conexão wi-fi no modo ad-hoc. Uma vez que o AR-Drone cria uma rede wi-fi com um ESSID normalmente chamado de ardrone2_xxx e o auto atribui um endereço de IP ímpar e livre (normalmente 192.168.1.1). O drone também possui uma DHCP (Dynamic Host Configuration Protocol) responsável por fornecer um endereço IP ao dispositivo cliente, sendo esse IP o próprio endereço IP do drone mais um número entre 1 e 4.

Após ter se estabelecido a conexão o dispositivo pode se comunicar com o drone mediante o envio de requisições através dessa conexão, tendo um endereço de IP e porta definidos. A comunicação é feita por strings em ASCII, poden assim se obter os logs dos sensores, status da bateria e ainda dos dados das câmeras e tambem enviar comandos de takeoff (decolagem), landing(pouso), pitch(movimentação para frente ou para trás), roll(movimentação para o lado esquerdo ou direito), throttle(para cima ou para baixo) e yaw(movimentação sobre o eixo Z) por exemplo.

O controle, bem como a configuração do drone, são realizados mediante o envio de comandos AT na porta UDP, User Datagram Protocol, 5556. Os comandos podem ser enviados em uma base de tempo regular, geralmente 30 vezes por segundo. A lista de comandos disponíveis e sua sintaxe é discutida na tópico a seção 4.4. Informações sobre o drone como, posição, status, velocidade, velocidade de rotação dos motores, entre outros, conhecidos como navdata, são enviadas do drone ao cliente através de UDP pela porta 5554. Nesses navdata também estão inclusas informações de detecção de tags para games com o drone. O stream de vídeo é enviado pelo AR.Drone através da porta 5555. As imagens do vídeo podem ser decodificadas usando o codec incluso no SDK.

Um canal de comunicação, conhecido como control port, pode ser estabelecido na porta TCP, Transmission Control Protocol, 5559 para transferência de dados críticos, sendo utilizado para recuperar dados de configuração e ainda para reconhecer informações importantes como o envio de informações de configuração para o drone.

Originalmente, a pilotagem do AR.Drone é realizada com o auxilio do piloto automático, assim como outros modelos de VANT, entretanto, devido a sua comunicação por Wi-fi, o radio controle é substituído por um aplicativo para celular ou tablet, conforme ilustração da Figura 17. Assim, procedimentos cruciais de voo citados passam a ser comandados por botões e manetes virtuais.

Essa interface virtual é uma característica marcante dos VANTs da Parrot e representa uma imensa vantagem sobre outras plataformas similares no desenvolvimento de aplicativos computacionais para esses veículos. Tal argumento se deve ao fato do código fonte usado na construção do aplicativo de comandos do VANT ser liberado para consulta. Assim, é possível

entender facilmente os protocolos de comunicação usados para acessar as informações do piloto automático do veículo.

Figura 17 – Aplicativo AR.FreeFlight usado para controlar o AR.Drone

Fonte: Martin (2015)

É importante destacar que, os códigos usados pela Parrot no controle dos atuadores, no tratamento dos dados sensoriais, no gerenciamento da rede sem fio e em outras funções de baixo nível são indisponíveis pois são propriedade do fabricante.

4.4 Comandos AT

Os comandos AT são strings enviadas ao drone que são codificadas em caracteres ASCII de 8 bits com um delimitador <LF>, Line Feed. Ainda, cada comando é composto por três caracteres AT* seguidos pelo nome do comando e um sinal de igual, um número de sequência e, caso seja necessário, uma lista de argumentos separados por vírgula cujo significado dependerá do comando em questão. é possível enviar mais de um comando AT em pacotes UDP desde que sejam separados pelo delimitador <LF> mas um mesmo comando AT não poderá ser enviado em pacotes UDP separados. Como exemplo, temos:

AT*PCMD=21625,1,0,0,0,0<LF>AT*REF=21626,290717696<LF>

É de extrema importância que o tamanho total do comando não ultrapasse 1024 bytes, isto é, 1024 caracteres, caso contrário toda a linha de comando será rejeitada. Com a finalidade de evitar a execução de comandos antigos, um número de sequência é associado a cada comando AT de modo que o drone não execute nenhum comando cujo número de sequência seja menor que o último executado por ele. O comando de sequência é o primeiro número depois do igual e a cada novo comando devemos incrementá-lo. A Tabela 2 contem um resumo dos comnados AT

Tabela 2 – Resumo dos comnados AT.

Fonte:Póprio altor

Comado AT	Argumentos	Descrição
AT*REF	input	Takeoff/Landing/Comando de cessar envio de sinal de emergência
AT*PCMD	flag, roll(inclinação esquerda-direita), pitch(nclinação frontal e traseira), gaz (velocidade vertica), guinada(velocidade angular)	Movimenta o drone
AT*PCMD_MAG	flag, roll, pitch, gaz, guinada, psi(magneto psi), precisão psi	Mova o drone (com suporte Absolute Control)
AT*FTRIM	-	Define a referência para o plano horizontal (deve esta no chão)
AT*CONFIG	key, value	Configuração do AR.Done
AT*LED	animação, frequência, duração	Sequência de animação dos leds
AT*COMWDG	número do dispositivo	redefinir watchdog de comunicação

4.5 Integração Computacional do AR.Drone 2.0

A facilidade de integração com dispositivos digitais, utilizando para isso apenas uma conexão de rede sem fio wi-fi, se da devido base de desenvolvimento de softwares fornecido pela SDK oficial, que fornece uma biblioteca para a comunicação entre o quadrirrotor e o cliente, esta biblioteca é denominada ARdroneLib. Trata-se de um código fonte liberado pelo fabricante do VANT como exemplo de integração, que utiliza linguagem de programação C/C++. Originalmente preparados para funcionar em computadores com sistemas operacionais Linux demonstrando como executar a leitura dos dados sensoriais, das imagens das câmeras de vídeo e também do envio de comandos de movimento. O principal aspecto positivo dessa ferramenta é o acesso livre aos códigos, porém sua organização é de difícil compreensão.

Ferramentas computacionais surgiram a partir do SDK oficial, tornando-se especialmente úteis em pesquisas acadêmicas ao incorporar uma melhor organização dos códigos de exemplo e uma documentação adicional. Dentre elas, merecem destaque as ferramentas criadas sobre o Robot Operating System, conhecido como ROS (Quigley et al. (2009)), tais como o ardrone_autonomy(MONAJJEMI (2015)) e o tum_ardrone(Engel, Sturm e Cremers (2014)), que incorporam os códigos e protocolos de comunicação do AR.Drone a outras ferramentas do ROS.

O ROS é uma plataforma de software desenvolvido pela OSRF (Open Source Robotics Foundation). Funciona como uma plataforma (framework) para o desenvolvimento e uso de software voltados para a robótica. O maior pressuposto para a criação da ROS é o compartilhamento, o reuso e a manutenção de códigos, ferramentas, drivers, assim como bibliotecas que possam ser integradas para a simplificação no desenvolvimento de tarefas complexas e robustas na área da robótica, e que possam funcionar em uma grande variedade de ambientes e sistemas robóticos. O ROS foi desenvolvido como uma camada de comunicação em rede peer-to-peer (ponto a ponto) entre diferentes processos. Isso facilita o desenvolvimento e a integração de diferentes bibliotecas e aplicações. Também, devido a esse fato, várias aplicações voltadas para a área da robótica estão utilizando recursos oferecidos pela ROS, bons exemplos de aplicações utilizadas com VANTs são o PTAM(do inglês Parallel Tracking and MappingKlein e Murray (2007)) e o EKF(do inglês Extended Kalman Filter), ambos aplicados em Engel, Sturm e Cremers (2014).

Tais características fizeram com que o ROS ganhasse grande atenção da comunidade científica nos últimos anos, avançando consideravelmente rápido em seu desenvolvimento a partir das diversas contribuições de seus usuários. Em relação ao AR.Drone, os únicos aspectos negativos que podem ser apontados dessas ferramentas são sua dependência exclusiva do sistema operacional Linux. As ferramentas apresentam demonstrações de soluções de voo apenas de ambientes interiores, sem resultados expressivos para ambientes exteriores.

Uma ferramenta que também merece destaque é a node-ar-drone(GEISENDÖRFER (2014)), que serve como base para o desenvolvimento de diversas outras bibliotecas, construída sobre o Node.js, que é framework, orientado a evento e desenvolvida sobre a engine JavaScript V8, desenvolvida pelo Google para seu navegador WEB Google Chrome. O Node provê um ambiente de execução server-side que compila e executa JavaScript com muita rapidez. A grande velocidade de execução é devido ao fato de que a engine V8 compila JavaScript em código de máquina nativo ao invés de interpretá-lo ou executá-lo como bytecod (Aaron (2013)).

Node é um projeto open source e pode ser executado em MacOSX, Windows e Linux. E que juntamente com seu gerenciador de pacotes, o NPM (node package manager), possibilita a integração com diversos outros bibliotecas e frameworks JavaScript como as bibliotecas NPM leaflet e leaflet-google que permitem a integração API do Google Map, por exemplo.

5 Desenvolvimento da aplicação

A aplicação deve possuir o requisito de disponibilizar um sistema web para cadastro de rotas de torres a serem inspecionadas e cadastro de uma base (ponto de pouso e decolagem) para cada rota. Deve conter os tipos de torres cadastradas com seus respectivos planos de voos para a inspeção da mesma. Após o estabelecimento da rota, a interface web deve permitir que o usuário acompanhe as imagens do drone em tempo real, além de monitorar sua movimentação e conseguir da comando para o drone em casos de problemas, como interromper a inspeção, pousar, ou voltar para base.

Ela também deve capturar as imagens da inspeção e salvá-las com a indicação do geo posicionamento em que a imagem foi registrada.

5.1 Modo de uso

Antes de iniciar a aplicação, deve-se posicionar o drone na base de decolagem com a frente do drone e a orientação da TAG voltada para o norte pois essa orientação servirá de referência para toda a missão, uma vez que o magnetômetro do drone não é tão preciso para nos dar a direção do norte. Depois ligar o drone e conectar o computador ao a sua rede wifi.

Depois, ao iniciar a aplicação, se deve definir a posição da base da missão (ponto de início) informando um latitude e longitude ou definindo no mapa. Depois deve se localizar no mapa as torres na ordem em que se deseja inspecionar,clicando nelas no mapa e definindo seu tipo, definindo assim, rota da missão. Após a definição de todas as torres deve-se concluir a fase da criação de rota e se iniciar a fase de acompanhamento da missão. Onde a aplicação deve se conectar ao drone e iniciar a missão quando solicitado.

Após o início da missão o drone deve decolar e subir para a altura de cruzeiro (altura pré determinada que garanta que o drone voe por cima das torres sem risco de colisão) e seguir para a primeira torre sendo sua movimentação acompanhada tanto pela imagem da sua câmera, quanto pelo seu geo posicionamento no mapa. Ao chegar na torre, o drone deve confirmar sua posição, essa confirmação ocorrerá por meio de uma TAG do tipo roundel orientado preto e branco, mostrada na Figura 18, pois a aquisição do módulo GPS da Parrot que também poderia exercer essa função elevaria muito custo do projeto.

Figura 18 – TAG do tipo roundel orientado preto e branco

Fonte:Piskorski et al. (2012)

Após a confirmação, o drone deve iniciar sua rotina de inspeção de acordo com o tipo da torre. No final da rotina o drone deve voltar para a altura de cruzeiro e seguir para a próxima torre repetindo o processo. Ao realizar a inspeção de todas as torres selecionadas o drone deve voltar para base e finalizar a missão. A Figura 19 apresenta um fluxograma funcional para a aplicação

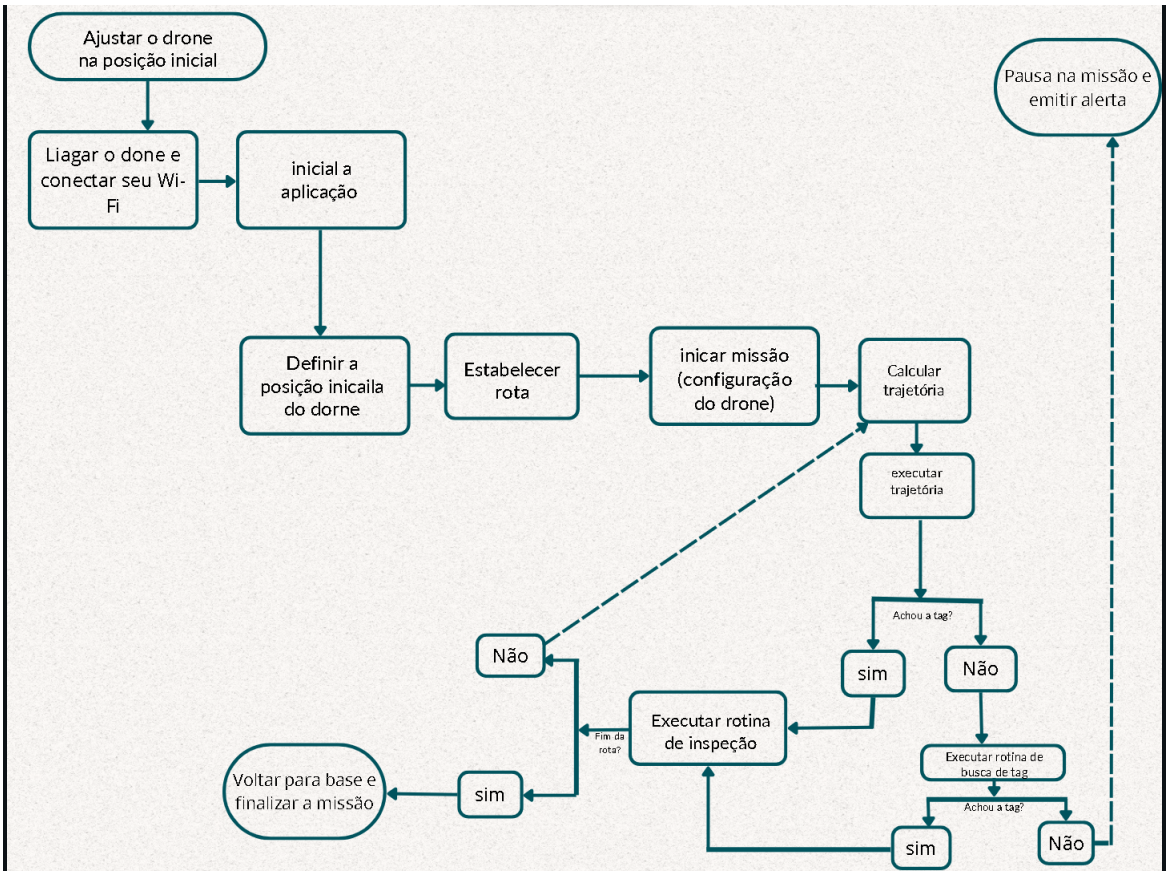


Figura 19 – fluxograma do funcionamento da aplicação

Fonte: Própio Autor

5.2 Implementação do software

Essa seção apresentará toda estrutura de desenvolvimento da arquitetura, contendo especificações dos algoritmos que processam e fazem a gestão dos eventos, alèm das ferramentas utilizadas na elaboração, assim como os cálculos matemáticos envolvidos para a definição dos pontos de latitude e longitude do mapa, tal como obter a distância precisa entre dois pontos distintos.

5.2.1 Ferramentas utilizadas

Para a implementação do software optou-se por utilizar o *Node.js* como plataforma de desenvolvimento da aplicação. Além dos benefícios citados na seção 4.5, ele é uma plataforma muito eficiente no consumo de recursos por ser uma arquitetura assíncrona orientada a eventos, possibilitando 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 os separa por requisição e de forma assíncrona retorna a promise da chamada.Esse modelo não bloqueante de tratar as requisições o torna excelente para o desenvolvimento de APIs consumindo pouquíssimo hardware.

Além disso, o fato do seu gerenciador de pacotes NPM conter não só a biblioteca node-ar-drone(GEISENDÖRFER (2014)) ja sitada na seção 4.5., ela ainda possui a biblioteca node-ardrone-autonomy (ESCHENAUER (2014)), que é uma biblioteca para voos autônomos desenvolvida especificamente para o Ar.Drone, construiída no topo da node-ar-drone. A Biblioteca geolib (BIEH (2020)) que fornecer operações geoespaciais básicas como cálculo de distância, direcao entre dois pontos entes outros calculos. E a biblioteca Leaflet (LIEDMAN (2021)) é a principal biblioteca JavaScript de código aberto para mapas interativos e funciona de maneira eficiente em todas as principais plataformas móveis e desktop.

5.2.1.1 Biblioteca *ar-drone*

Essa biblioteca conta com API de alto nível que abstrai as funcionalidades disponíveis na SDK (Piskorski et al. (2012)) permitindo enviar comandos escritos em JavaScript para serem interpretados pelo *framework Node.js* que envia e recebe pacotes através de conexão UDP (User Datagram Protocol).

A node-ar-drone possui, um módulo cliente que disponibiliza uma API de alto nível suportando a grande maioria das funções disponíveis para controlar o drone, de forma simples na usabilidade e interpretação lógica, usando desde acesso direto para enviar comandos UDP de baixo nível, através de métodos que enviam os dados usando a porta 5556, até a criar um pequeno programa autônomo que realiza comandos pré-programados via codificação JavaScript. Possui a possibilidade de acessar a câmera vertical e horizontal, salvando fotos e

vídeos ou até mesmo transmitir em tempo real usando HTTP (HyperText Transfer Protocol) web servidor. E também acesso aos sensores disponíveis no aparelho, bastando chamar métodos específicos, tal como obter altitude. Ela conta com um ampla quantidades de métodos disponíveis, e suas principais metodos são (informacao retirada de GEISENDÖRFER (2014)):

a) `client = ar-drone.createClient(opções);`:
   - retorna um novo objeto cliente, opções incluem ip (o IP do drone. O padrão é "192.169.1.1", 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(retornodechamada);`:
   - 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) `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;

m) `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;

n) `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;

o) `client.config(chave, valor, retorno de chamada);`:

- envia um comando de configuração para o drone.

5.2.1.2 Biblioteca *ardrone-autonomy*

Essa biblioteca foi construída por Laurent Eschenauer (ESCHENAUER, 2014) com base na node-ar-drone dando acesso a todos os métodos dela e a usando para integrar o drone com a arquitetura. Ficando para a biblioteca node-ardrone-autonomy a função de controlar a posição drone usando os métodos client.front(x), client.right(y), client.up(cz) e client.clockwise(cyaw) da biblioteca base, e desenvolver e estruturar as missões.

A posição do drone é estimada usando a velocidade em x e y fornecida pela hardware do Ar.drone através da navdata.demo que é calculada a partir dos dados oriundos de seus sensores inerciais. Sendo a direção positiva de x, a mesma em que a câmera frontal do drone aponta, e a direção positiva de y, a direta do drone. O plano x e y tido como referência para o posicionamento do drone no espaço se dá na posição inicial do drone, e não varia com os giros do drone em torno do eixo z. Porém é possível resetar esse plano com um comando a qualquer momento.

A node-ardrone-autonomy também se utiliza de projeção da câmera para estimar a posição de um objeto detectado pela câmera. Atualmente usado para estimar a posição da TAG baseado em sistemas de coordenadas do drone utilizando a câmera vertical. A estimação da posição dessa tag é utilizada como fonte de observação para um filtro Filtro de Kalman Estendido que é usado para fundir as estimativas de posicacao calculadas com a velocidade com a posicao do drone em realacao a TAG sendo essa considerada como zero do plano de coordenadas para corrigir a estimativa do estado atula drone. Isso faz com que, quando detectado a TAG, o sistema forneça uma estimativa de estado muito mais estável e utilizável.

Para que o controle da posição seja realizado de forma autônoma, também é implementado um controlador PID (Controlador proporcional integral derivativo) tendo como entrada a posição estimada pelo sistema, e como referência a posição declarada no plano de missão.

Ela conta com um ampla quantidades de métodos disponíveis, e suas principais metodos são (informacao retirada de ESCHENAUER (2014)):

a) `mission = ardrone-autonomy.createMission();`:
   - cria missão, disponibilizando acesso a todos métodos da API. Internamente esse método também executa o método ar-drone.createClient(opções). Concedendo acesso aos comandos da API node-ar-drone, para a configuração como por exemplo mission.client().config("general:navdata_demo", "FALSE") que executa client.config(chave, valor, retorno de chamada), ja sitando na subseção 5.2.1.1;

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 ocomportamento 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;

A utilização desta biblioteca apresentou excelentes resultados como pode ser observado no trabalho Correa (2020)

5.2.1.3 Biblioteca *geolib*

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 coordinatesálidos (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.

d) `computeDestinationPoint (ponto, distância, rumo, raio = raio da terra)`
   - Calcula o ponto de destino dado um ponto inicial, uma distância (em metros) e um rumo (em graus). Se nenhum raio for fornecido, o padrão é o raio terrestre médio de 6.371.000 metros. Atenção: esta fórmula não é 100% precisa (mas muito próxima).

5.2.1.4 Biblioteca *Leaflet*

Com apenas 38 Kilobyte (KB) de JavaScript (JS), a biblioteca Leaflet possui todos os recursos de mapeamento de forma off- line e pode ser estendida com muitos plugins, além de possui uma API documentada de código-fonte legível (LIEDMAN, 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], 22)`
   - 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,quanto maior o número, mais próximo da terra estará a imagem do satélite, sendo esso zoom máximo, podendo se utilizar de mais dois graus de zoom chegando a 24, porém sem nova renderização de imagem;

b) `map.addLayer(googleHybrid);`
   - é usado para usar o layout híbrido do google, onde a variável googleHybrid é definida como:

```
googleHybrid = L.tileLayer(
'http://{s}.google.com/vt/lyrs=s,h&x={x}&y={y}&z={z}',
{maxZoom: 22,
subdomains:['mt0','mt1','mt2','mt3']
});
```

c) `L.marker([latitude,longitude],ícone:objetoIcone).addTo(map)`
   - Esse método adiciona ao mapa já aberto o ícone desejado, na posição da latitudee longitude passadas no parâmetro.

5.2.2 Estruturação

Para criar e estruturar a aplicação foi usado exper generator, que é um *Node.js Framework*, usado para criar aplicativos *express.js* de maneira fácil e rápida, criando automaticamente a estrutura do aplicativo como mostrado na Figura 20.

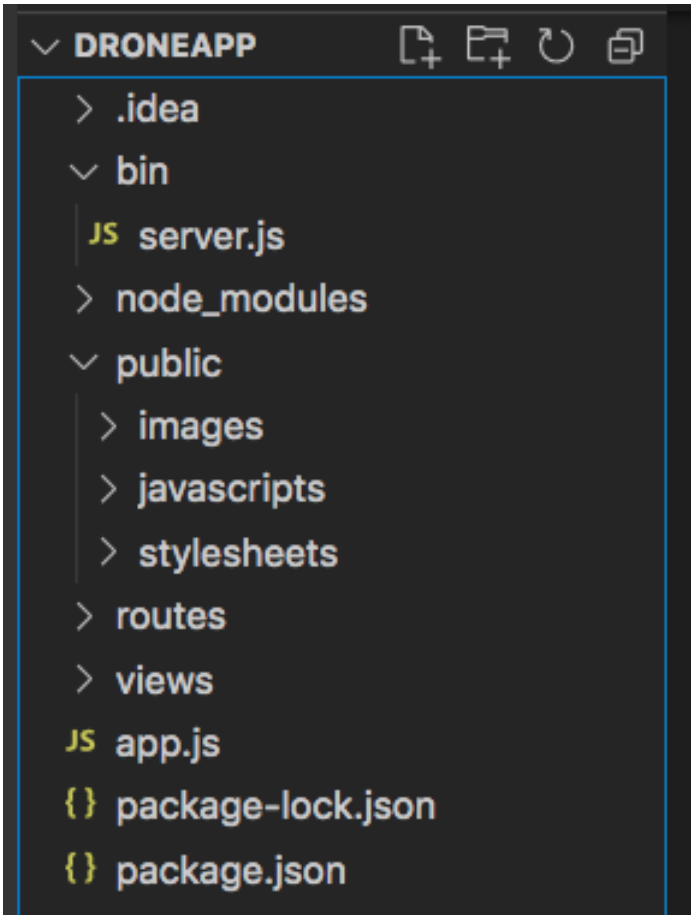


Figura 20 – Estrutura da implementação software

Fonte: Própio Autor

O *Express.js* é um Framework rápido e um dos mais utilizados em conjunto com o *Node.js*, facilitando no desenvolvimento de aplicações *back-end* e até, em conjunto com sistemas de templates, aplicações *full-stack*, uma vez que possui diversas características que facilitam o desenvolvimento de nossas aplicações. Dentre suas principais características, podemos citar

- Possuir um sistema de rotas completo;
- Possibilita o tratamento de exceções dentro da aplicação;
- Permite a integração de vários sistemas de templates que facilitam a criação de páginas web para suas aplicações;
- Gerência diferentes requisições HTTP com seus mais diversos verbos;
- Feito para a criação rápida de aplicações utilizando um conjunto pequeno de arquivos e pastas;

Dentro da estrutura mostrada na Figura 20, na pasta routes contém as rotas da aplicação, a pasta views contém os códigos das páginas web escrito em EJS (Embedded JavaScript templating) que é uma linguagem de modelagem simples que permite gerar marcação HTML com JavaScript simples. A pasta public contém todos os arquivos usados nas páginas web como os scripts em javascript a serem executados com as páginas, as imagens utilizadas e os códigos css que estiliza as páginas. Na node-modules estão as bibliotecas necessárias para o funcionamento da aplicação, essas estão listadas juntamente com o número das suas versões no arquivo *package.json* em que seu conteúdo está expresso no código abaixo, em que também se contém o caminho do arquivo que inicia a aplicação, sendo esse o arquivo server.js contido na pasta bin.

```
{
  "name": "droneapp",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/\emph{server.js}"
  },

  "dependencies": {
    "ar-drone": "^0.3.3",
    "ardrone-autonomy": "^0.1.2",
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "dronestream": "^1.1.1",
    "ejs": "^3.1.8",
    "express": "^4.18.2",
    "geolib": "^3.3.3",
    "http-errors": "~1.6.3",
    "jquery": "^3.6.1",
    "leaflet": "^1.9.2",
    "morgan": "~1.9.1",
    "socket.io": "^4.5.3"
  },

  "devDependencies": {
    "nodemon": "^2.0.20"
  }
}
```

No arquivo *server.js* foi desenvolvido o servidor da arquitetura utilizando o express (WILSON (2019)), que é configurado no arquivo App, e 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, alem de integrar o socket.io ao server HTTP, sendo esse 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*.

No mesmo arquivo também é implementada toda a conexão com o drone e sua configuração através da biblioteca ardrone_autonomy. Além disso, esse arquivo é responsável por armazenar os pontos das rotas em sequência junto com os respectivos tipos de torres pertencentes à rotam assim como os cálculos para definir o deslocamento que deve ser efetuado pelo drone traduzidos em coordenadas x e y, e os cálculos para tradução da movimentação do drone em latitude e longitude para que se possa plotar a movimentação no mapa utilizando a ajuda da biblioteca geolib.

As rotina de inspeção de cada tipo de torre também estão no arquivo server e *server.js* melhores explicadas na próxima seção.

Na pasta public também há 2 script em JavaScript, um para cada página da aplicação, que corresponde a cada uma das etapas do software (A primeira,definição de rota e tipos de torres. E a segunda, acompanhamento da inspeção). Ambos os scripts utilizam a biblioteca leaflet para a renderização do mapa, sendo a primeira página com o objetivos de obter as coordenadas das torres e visualizar uma prévia da rota e a segunda de informar o posicionamento do drone em tempo real além de guardar sua movimentação.

5.2.3 Rotinas de inspeção

Para o desenvolvimento desse projeto foram considerados dois tipos de torres, e foram criadas uma rotina de inspeção pré definida para cada uma. A torre do tipo "A", ilustrada na Figura 21, é uma torre de linhas de circuito duplo, e a torre tipo "B"(Figura 22) é uma torre de linha de transmissão de circuito simples encontradas no livro de FUCHS (1977).

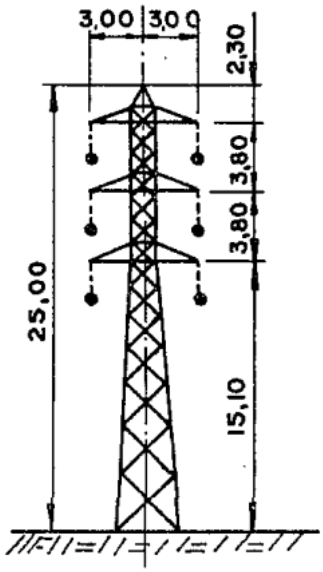


Figura 21 – Trorre de sircuito duplo (tipo *A*)

Fonte: FUCHS (1977)

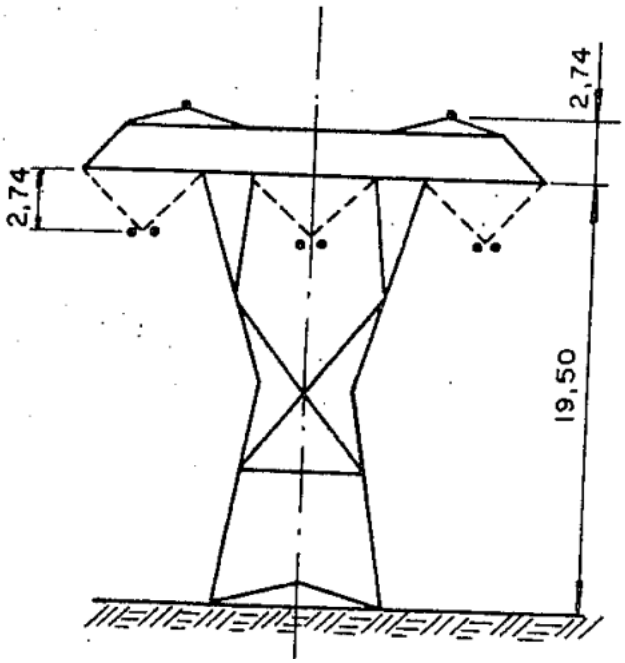


Figura 22 – Torre de circuito simples (tipo *B*)

Fonte: FUCHS (1977)

Para diminuir possíveis erros de aproximação nos cálculos para a movimentação do drone ou imprecisões na movimentação do mesmo deve-se colocar uma TAG no topo de cada torre em seu centro com sua orientação paralela aos cabos da torre. Isso para que no início de cada inspeção se garanta que o centro da torre seja a nova origem do plano cartesiano do drone, e que o eixo x estará sempre paralelo aos cabos da torre e o y positivo a direita.

O foco maior da inspeção está em principalmente verificar a integridadefísico das cadeias de isoladores, e em segundo plano a integridade da estrutura da torre, do cabo guarda e da ferragens quem conectam os cabos aos isoladores e esses a torre. E em terceiro plano, a situação dos acessos às estruturas, a proximidade da vegetação aos cabos e os casos de invasão de faixa de servidão.

Levando em conta as dimensões das torres mostradas na Figura 21 e Figura 22 foram traçadas rotas de inspeção para a torres do tipo A e B respectivamente. Para as do tipo A definiu-se 12 posições de inspeção, da onde se deve tirar as fotos,devido ao grande número de cadeias de isoladores, sabendo que os valores das movimentações são sempre dados em metros, os pontos são:(x, y, *altura*)

- (0,0,28) - foto tirada com a camera vertical
- (0,6,22) - drone deve girar -90 graus para fotografar com a câmera frontal
- (0,6,18)

- (0,6,14)
- (-6,6,10) - drone deve girar +45 graus para enquadrar a torre
- (-6,0,10) - se posiciona embaixo dos cabos e girar +45 graus para enquadrar a torre
- (-6,-6,14) - drone deve girar +45 graus para enquadrar a torre
- (0,-6,14) - drone deve girar +45 graus para enquadrar a torre
- (0,-6,18)
- (0,-6,22)
- (6,-6,28) - drone deve girar +45 graus para enquadrar a torre
- (6,0,28) - se posiciona em cima dos cabos e girar +45 graus para enquadrar a torre
- 0,0,28) - deve girar girando +180 graus para voltar para o estado do inicio da rotina de inspeção

Gerando a trajetória de inspeção da Figura 23 para as torres do tipo *A*

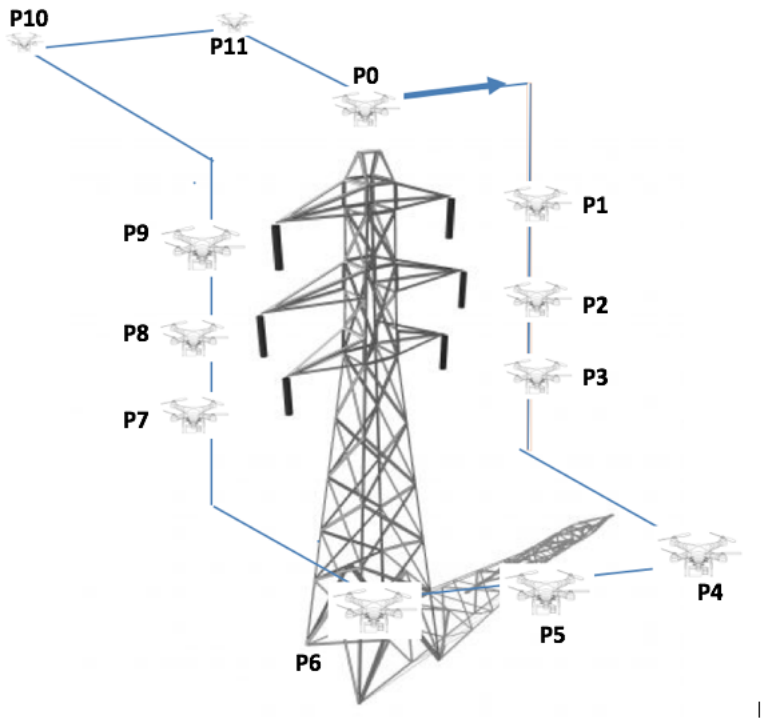


Figura 23 – Trajetória de inspeção para as torres do tipo *A*

Fonte: Próprio Autor

Para torres do tipo *B* definiu-se 9 pontos de interesse :

- (0,0,25) - foto tirada com a camera vertical
- (0,7,25) - foto tirada com a camera vertical
- (0,10,17) - drone deve girar -90 graus para fotografar com a câmera frontal
- (-10,10,10) - drone deve girar +45 graus para enquadrar a torre
- (-10,0,10) - se posiciona embaixo dos cabos e girar +45 graus para enquadrar
- (-10,-10,17) - drone deve girar +45 graus para enquadrar a torre
- (0,-10,17) - drone deve girar +45 graus para enquadrar a torre
- (0,-7,25) - foto tirada com a camera vertical
- (0,0,25) - drone deve girar -90 graus para voltar para o estado do inicio da rotina de inspeção

Gerando a trajetória de inspeção da Figura 24 para as torres do tipo *B*

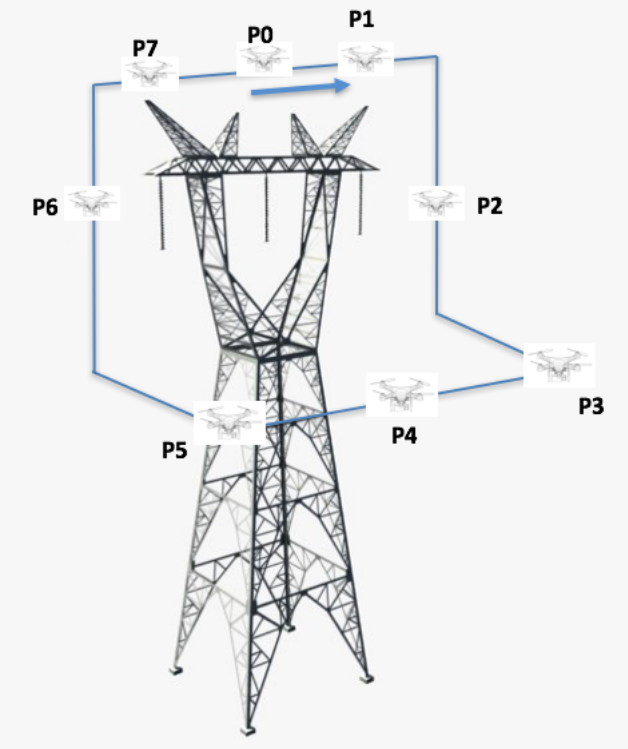


Figura 24 – Trajetória de inspeção para as torres do tipo *B*

Fonte: Próprio Autor

Ao final de cada inspeção o drone deve voltar para o ponto (0,0,30), sendo definido 30m como altura de cruzeiro, e finalizar a inspeção e se realinhar com o norte voltando com o seu plano cartesiano original com o x apontando para o norte, porém com a origem

exatamente sobre a torre. A TAG também ajudará nesse ponto para garantir que antes de ir para a próxima torre o drone esteja na posição esperada.

5.2.4 Implementação

O arquivo server levanta o servidor express, que abre a conexão para comunicação em tempo real utilizando o socket.io, a conexão se estabelece ao se iniciar a interface da arquitetura pelo navegador web. Além de estabelecer a primeira comunicação com o drone através da biblioteca dronestream , com o comando `require("dronestream").listen(server)`, que atua integrando a arquitetura a transmissão em tempo real do vídeo capturado pela câmera frontal do drone, essa técnica de visualização é chamada de First Person View, mais conhecida como voo em FPV, aonde é utilizado o protocolo de comunicação TCP para obter o vídeo do AR.Drone (WEISSHUHN (2013)).

Na página inicial, acessada pelo endereço http://localhost:3000, são emitidos os eventos gerados pelos clicks dos botões "definir"e "definir no mapa"com mostrados na Figura 25

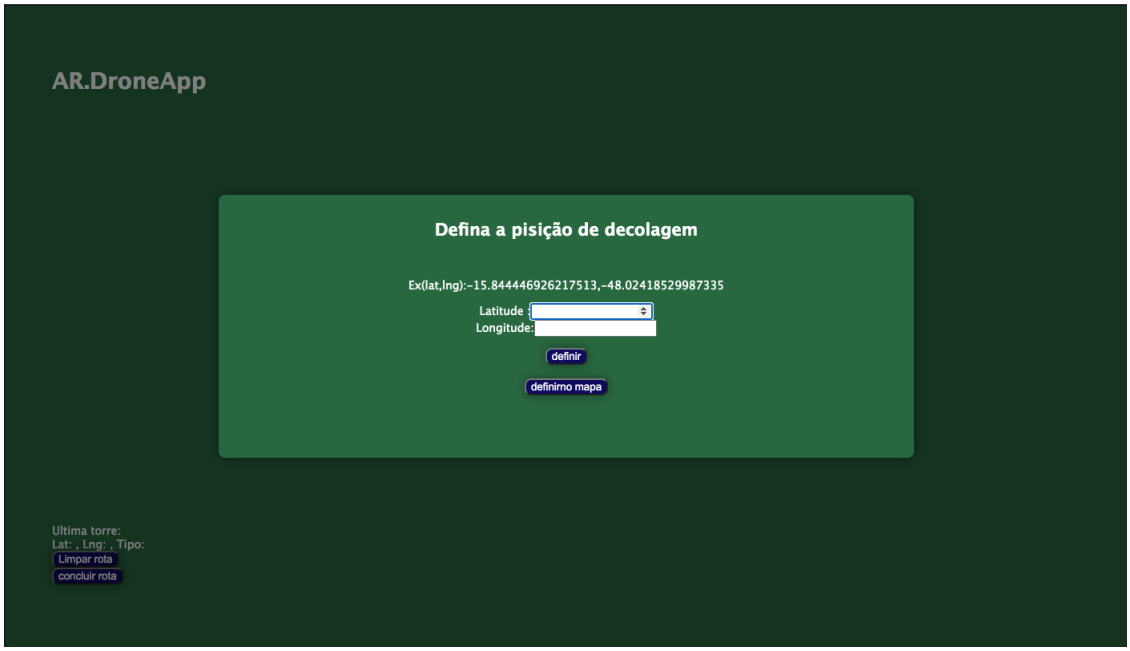


Figura 25 – Tela da página inicial da aplicação

Fonte: Própio Autor

Ambos os botões disparado um evento de abertura do mapa interativo sendo que o primeiro abre o mapa, tendo como centro a latitude e a longitude fornecida pelos campos disponíveis sendo nesse ponto também é estabelecido a base da missão indicado pelo ícone de heliporto no mapa mostrado na Figura 26. Ao clicar em definir no mapa o sistema tentar utilizar a localização fornecida pelo navegador como ponto inicial da abertura do mapa com um zoom setado em 22, caso o sistema não consiga obter essa informação através

do comando ele abrirá o mapa em uma coordenada pré estabelecida em um zoom de 18, mostrando uma grande área.

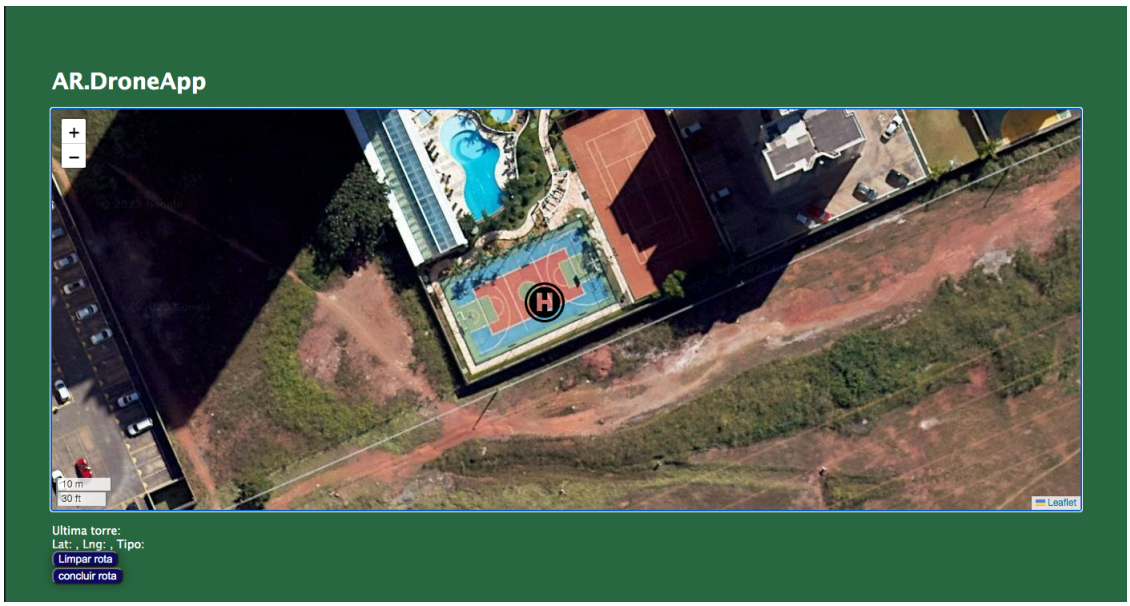


Figura 26 – Tela da página definição de rota

Fonte: Própio Autor

A abertura do mapa e suas intercessões são feitas usando a biblioteca leaflet, usando os métodos já citados da subseção 5.2.1.4, e para a obter a localização atual do computador com alta precisão através do método

- `L.marker([latitude,longitude],ícone:objetoIcone).addTo(map)`

A biblioteca foi utilizada também para processar cada clique no mapa. Aonde cada o primeiro click define a posição da base caso ela ainda não tenho sido definida pelo botão "definir", caso contrário, cada click no mapa é gerado visualmente um marcador na latitude e longitude, capturada no evento de click sobre o local no mapa esse marcadore são os marcadores são chamados waypoints e representam as torres selecionadas. Esse evento também fará com que abra um Pop-up para a definição do tipo de torre, e com a definição, a posição é armazenada juntamente com um tipo de torre em um array de torres assim como os dados dos waypoints em outro array. Conforme as torres vão sendo definidas é desenhado no mapa um traço que faz a ligação visual entre os pontos, formando uma rota conexa partindo da base esse traço tambem é feito pela biblioteca leaflet usando o array de waypoint, no qual é adicionado no seu inicio a posicao da base, e a seguinte estrutura

```
if (   rota == undefined) {
rota = L.polyline(waypoints, { color: 'blue' }).addTo(map);
   } else {
```

```
rota.setLatLngs(waypoints)
 }
```

Figura 27 – Tela da página definição de rota com a rota definida

Fonte: Própio Autor

Neste ponto há mais botões, um que limpa toda a rota, para que as torres sejam definidas novamente, e outro para concluir a definição da rota, o segundo emite um evento que é tratado pelo servidor armazenado posições das torres na ordem a serem inspecionadas e a posição da base atarves da ferramenta socket.io com o metodo `socket.emit("RotaConcluida", torres:torres, base: base)`.

Após a execução desse método a interface web é redirecionado para a segunda página (página de acompanhamento da inspeção). Após a página ser iniciada a interface se conecta com o servidor pelo socket.io e gera um evento para ser tratado pelo servidor, esse evento executa a rotina de inicialização do drone criando a variável mission como mostrado na subseção 5.2.1.2 e define as configurações da aplicação no dispositivo através do método `client.config(chave, valor, retorno de chamada);`, da biblioteca node-ar-drone acessado através da biblioteca ardrone-autonomy como já citado na mesma seção, como os dados que se deseja receber no navdata, o fluxo de vídeo de qual câmera se deseja receber, entre outros.

Nas configurações da navdata o comando `mission.client().config('general:navdata_demo', true);` define que o drone irá enviar um conjunto reduzido de dados de navegação para o cliente, esta configuração define que a taxa de atualização dos dados será de aproximadamente 15 vezes por segundo, além de permitir que o aplicativo solicite outros pacotes navdata através da chave `'general:navdata_options'`, como definido na SDK

Piskorski et al. (2012), a função `navdata_option_mask(c)` ajusta o valor a ser enviado para o VANT que definirá os pacotes solicitados, sendo esses definidos na variável navdata_options como pacotes de dados de demonstração (demo), detecção de tag, magnetômetro e do wi-fi. Tais definições são enviadas no comando mission.client().config('general:navdata_options', navdata_options);.

Configurando a chave `'video:video_channel'` como 0, é definido que o fluxo de imagem recebido será o da câmera frontal, e a chave `'detect:detect_type'` como 12 que a tag utilizada na detecção é do tipo roundel orientado preto e branco, utilizada no EKF da biblioteca ardrone-autonomy. A chave `'control:outdoor '` definida como verdadeiro configura o drone para voo externo, que de acordo com a SDK Piskorski et al. (2012) habilita o estimador de vento do ar.drone 2.0 para tentar fazer a compensação do mesmo em seu controle interno. E a chave `"control:flight_without_shell"` define para o controlador interno do drone com que tipo de casco o drone está, esta foi definida como falsa, uma vez que o drone está voando o casco de proteção para voos internos.

Esse evento gerado pela página 2 ao se iniciar também faz com que o servidor da aplicação gere o evento `socket.emit('carregarPag2',torres,base)`, que contém os dados necessários para a abertura do mapa interativo, usando a biblioteca leaflet assim como na página inicial, mas dessa vez colocando o ícone que representa a posição do drone na mesma posição da base, sem a linha azul ligando as torres, pois essa linha agora será usada para plotar a movimentação do drone no mapa.

Os demais eventos gerados pela interface web são os eventos gerados pelo click dos botões disponíveis.

a) botão stop
   - `socket.emit("stop")`
     - Esse método adiciona ao mapa já aberto o ícone desejado, na posição da latitudee longitude passadas no parâmetro.

b) botão pausar
   - `socket.emit("land")`
     - dispara evento para o servidor tratar a aterrissagem do drone, como explicado no método client.land(retornodechamada)

c) botão go to home
   - `socket.emit("goToHome")`
     - dispara eventos para o servidor iniciar a rotina para retornar a base

d) botão Iniciar Missão

- `socket.emit("iniciarMisaao")`

  - dispara eventos para o servidor iniciar a rotina que fará com que o drone execute a missão;

O Servidor da arquitetura também passa a emitir eventos para a interface cliente tratar são esses

a) `io.sockets.emit(‘‘droneData"data)`

  - dispara em tempo real de execução o evento para o cliente tratar atribuindo as informações na interface para o usuário visualizar, como posição do drone, percentual de bateria e altura em que o ele se encontra. quando a posição do drone variam, essa variação é desenhada no mapa como já citado nesta seção.

b) `io.sockets.emit(‘‘menssagen"str)`

  - dispara em tempo real de execução o evento para o cliente tratar mostrando ao usuário mensagens de status da missão.

No arquivo p2.ejs situado na pasta view,responsável pela geração de código HTML da página de acompanhamento, também fois criado uma div com IDentify (ID) igual a `"dronestream"` ,e no mesmo arquivo foi executado outro script também contido na biblioteca dronestream denominado *nodecopter-client.js*. Isso torna possivem a execucao do metodo `new NodecopterStream(document.getElementById("droneStream"))`, que passa a renderizar o buffer de video recebido do Ar.Drone na div assim que a pagina é aberta graças a conexão estabelecida feita pela a biblioteca dronestream. Gerando assim a tela da Figura 28

Quando o servidor da aplicação recebe o evento de iniciar a missão ele executa uma etapa de decolagem e depois de subida até a altura de cruzeiro (30 m), onde o seu posicionamento é corrigido com a ajuda da TAG posicionada na base, e se inicia os cálculos para a movimentação, onde fui usada a biblioteca geolib para abstrair fórmulas e cálculos matemáticos, os seguintes métodos foram implementados:

a) `getPreciseDistance (start, end [int precision])`

  - para se obter a distância que o drone deverá percorrer até chegar ao seu destino;

b) `getGreatCircleBearing(origin, destination)`

  - Para se obter o ângulo entre as duas coordenadas geográficas obtendo assim a direção

Figura 28 – Tela da página de acompanhamento

Fonte: Próprio Autor

Com a distância e a direção obtemos os componentes x e y do deslocamento do drone. com isso o servidor executa os seguinte métodos da biblioteca ardrone-autonomy :

```
mission.yam(ângulo)
mission.forward(distância)
mission.yam(0)
mission.go(x,y)
```

Onde o drone irá se virar de frente para a próximo ponto de destino e seguir em frente pela distância calculada, ao essa movimentam o drone volta a se alinhar com o norte e executa o comando go com os valores de x e y na tentativa de corrigir corrigir possíveis erros de angulação no primeiro comando. Em seguida o servidor deve executar a rotina de inspeção do tipo de torre.

Essa rotina consiste, independente do tipo de torre, em:

- Localizar a Tag para confirmar a posição;
- Registrar a orientação da TAG (a angulação necessária para se alinhar a direção da TAG, valor esse já fornecido pelo dados oriundo do sistema do drone ao detectar a TAG);
- Alinhar-se com a TAG e consequentemente com os cabos;
- Executar o comando mission.zero;
- Executar a sequência de inspeção que termina no mesmo ponto (sobre a TAG localizada no topo da torre;

- E por fim voltar a altura de cruzeiro;

Para que essa sequência funcionasse, foi necessário alterar a biblioteca ardrone autome de forma a conseguir habilitar e desabilitar o EKF, o deixando habilitado apenas após o comando para zerar os estados *x*, *y* e *yam* do drone,e o desabilita-lo após o fim da rotina de inspeção. Para que a referência do eixo de coordenada *x* e *y* não mude caso o drone passe por uma TAG no meio da trajetória,uma vez que haverá uma TAG em cima de cada torre e o sistema do drone não as diferenciam uma das outras, e para se possa ter o controle de quando se deve mudar a referência para TAG ou apenas obter a informações dela, como por exemplo, que torre foi encontrada ou a direção dos cabos.

Sempre que chega um novo conjunto de dados do drone a acionado um no servidor que atualiza a posição do drone em relação a sua latitude e longitude a partir da variação dos estados estimados de posição feitos pela biblioteca ardrone-autonomy. para traduzir essa movimentação, são feitos cálculos utilizando a novamente um método da biblioteca *Geolib*,

```
geolib.computeDestinationPoint({ latitude: lat_ponto_de_origem,
longitude:lng_ponto_de_origem},
deslocamento,   angulo);
```

Para as movimentações de um ponto a outro na rota de torres, os deslocamentos são calculados a partir da variação em *x* e em *y*, tendo como referência o ponto anterior, tirando a hipotenusa desses vetores e o ângulo sendo a rotação que o drone fez para se direcionar de frente para o próximo ponto da rota. Já para se obter a latitude e a longitude do drone enquanto ele está executando a rotina de inspeção, é necessário aplicar o rotacional em *z* do anglo que precisou-se rotacionar o drone para alinhá lo com os cabos nas variações de *x* e *y*, uma vez que para essas movimentos esse par de eixos está rotacionado em relação aos eixos *x* e *y* do sistema original, antes de aplicá-los no método para se obter as novas coordenadas geográficas.

6 Testes

6.1 Teste do Sistema

Para testar primeiramente a viabilidade do sistema de forma independente do controle do drone, foi implementado o sistema se abstraindo das bibliotecas ardorne-autonomy e dronestream. E foi criado uma rotina que a cada três segundos realiza a variação em x e y conforme a os cálculos de movimentação da do dronte de um ponto da rota para o outro. Também foi desenvolvida uma rotina que simula a movimentação do drone quando chegar na torre e realiza a inspeção. Podendo assim avaliar a exatidão dos cálculos e o fluxo do sistema. e a tradução da movimentação do drone em coordenadas geográficas Os códigos, exceto o do server.js, se mantiveram iguais, e o resultado pode ser visto na Figura 29. Onde os pontos vermelhos representam a posição que se foi tirada as fotos, e a esquerda da imagem pode se observar os últimos passos da inspeção que foram mostradas no campo de mensagem da aplicação. sendo que a rota traçada para o teste foi a mesma da mostrada na Figura 27.

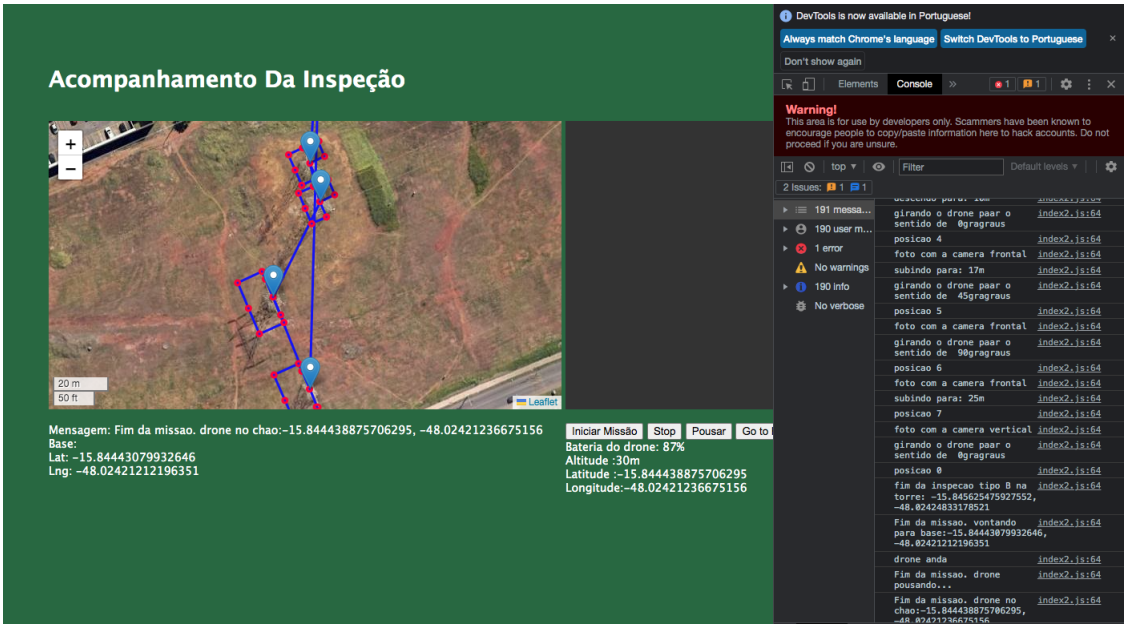


Figura 29 – Tela do resultado teste do sistema

Fonte: Próprio Autor

O que mostra que as estratégias empregadas na implantação do sistema entregam bons resultados para área de inspeção de linhas de transmissão com cálculos bastantes precisos, com erros que não ultrapassam 1 metro de distância entre o ponto marcado no mapa e os pontos atingidos pelos valores de movimentação. Os erros foram calculados usando novamente o método da biblioteca geolib e apresentados no final da execução do

teste mostrado na Figura 30, sendo de:

- 0.07 m para a primeira torre
- 0.23 m para a segunda torre
- 0.05 m para a terceira torre
- 0.48 m para a quarta torre
- 0.89 m para voltar para base

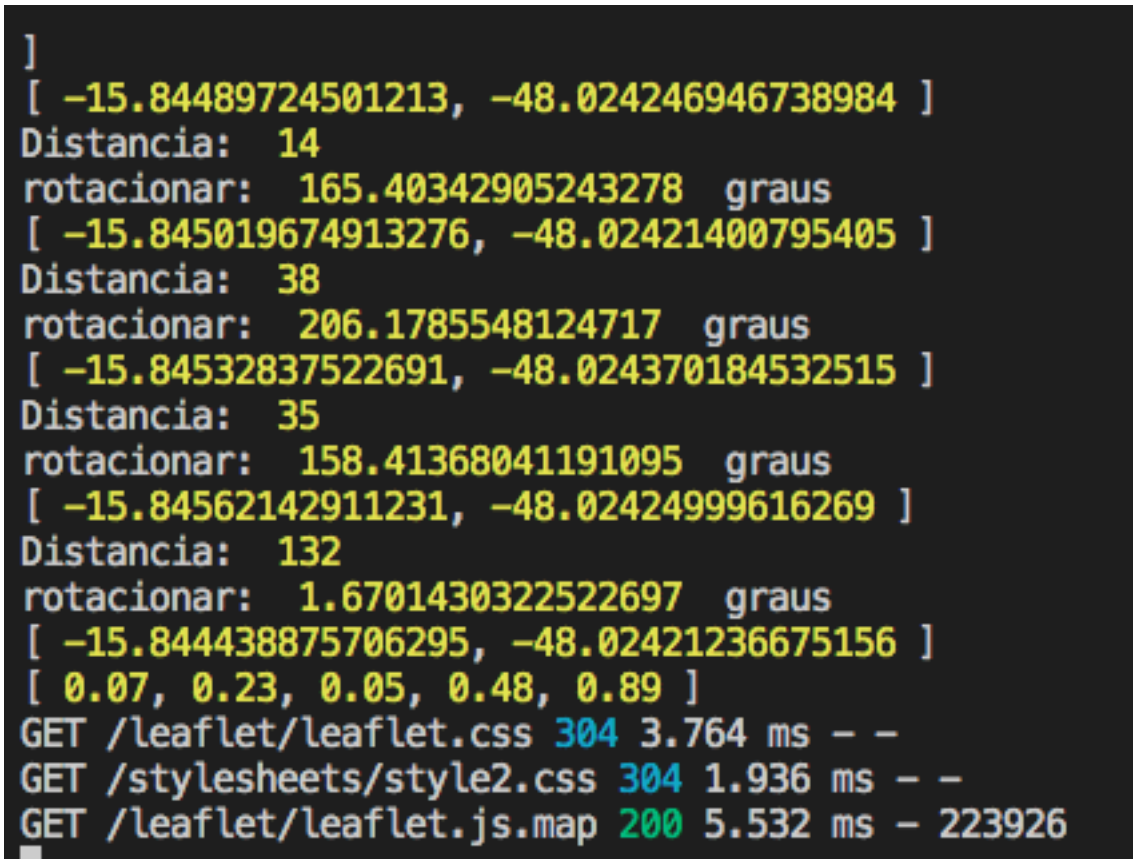


Figura 30 – erros de posicao calculados

Fonte: Próprio Autor

6.2 Teste de controle do drone

Nessa etapa foi desenvolvido sistema separado apenas com o objetivo de testar a controlabilidade do drone utilizando a biblioteca node ardrone-autonomy para controlar o drone e socket.io para criar eventos na interface web que fossem tratados pelo servidor com o objetivo de enviar comandos para o vant e também receber dados do mesmo assim como a aplicação principal. Também foi utilizada a biblioteca NPM Chart (2023) para plotar os

dados do drone usando a interface web. Com esse sistema foi possível testar a funcionalidade da detecção da TAG, os comandos de de troca de câmera, alternando entre as imagens ora da câmera frontal ora vertical durante o voo. Também foi possível avaliar as diferenças quando o drone está voando com a configuração de controle interno setada para voos em ambientes internos e externos. Verificou-se que a comunicação com o drone foi bem estabelecida, uma vez que a identificação da TAG e sua orientação foram bem sucedidas e assim como, a alternância entre as câmera, a obtenção dos dados e o comando de movimentos.

Figura 31 – Ambiente de teste do drone

Fonte: Próprio Autor

O que pode ser comprovado por testes de movimentação. Onde foram colocados marcadores no chão com uma distância de um metro entre eles para aferir a movimentação durante o voo conforme na Figura 32 e dado o comando para o drone decolar e ir dois metros para frente. onde observou se que ao executar a etapa de decolagem o drone não subia em linha reta. se deslocando sem muito padrão definido, por isso, após a decolagem passou a ser executado o comando *mission.go(0,0)* para o drone retorna a posição inicial e depois ir dois metros pra frente com o comando *mission.forward(2)*, planar e por fim posar para se observar o deslocamento. um exemplo deste teste pode ser visto na Figura 32.

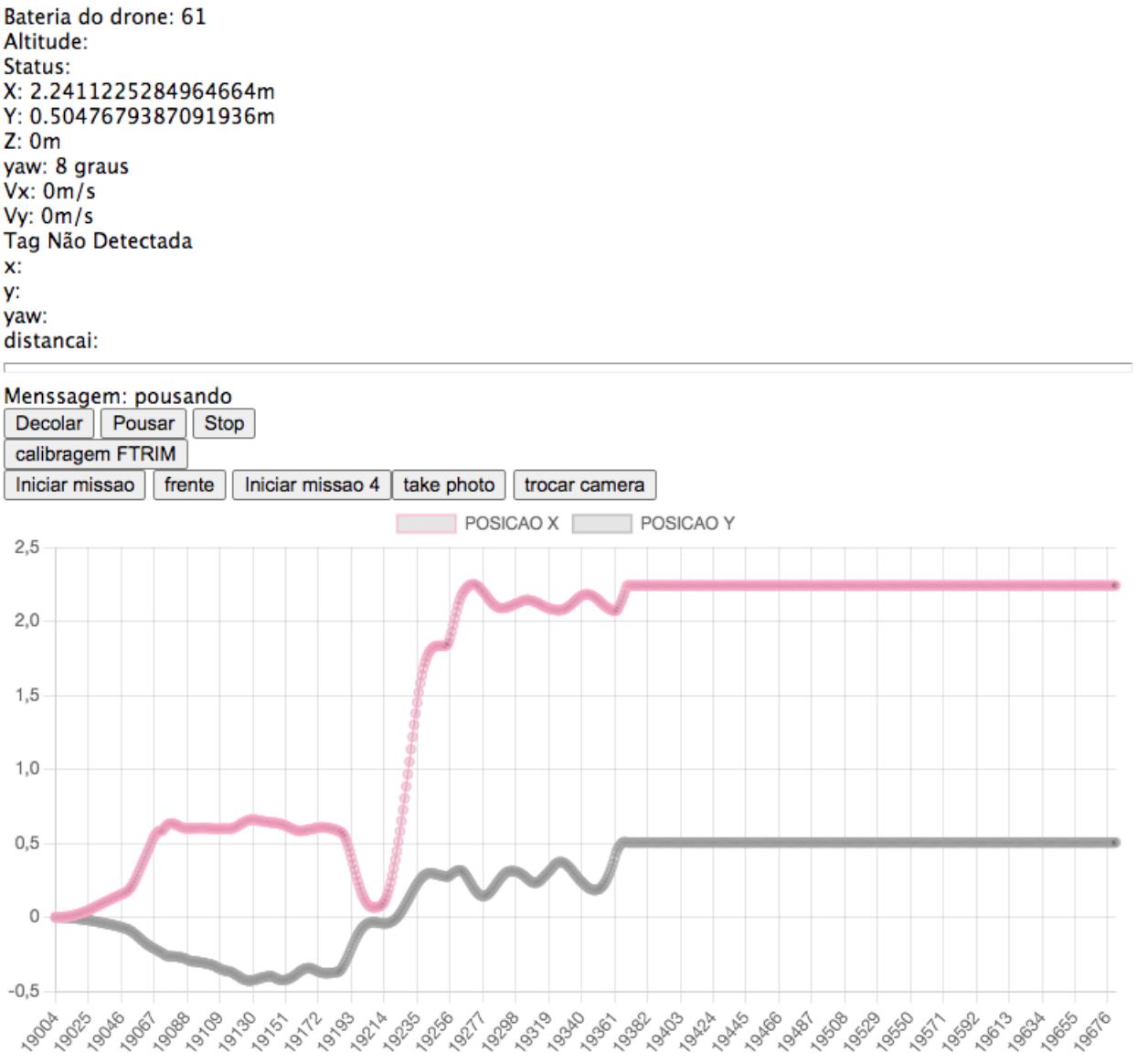


Figura 32 – Teste do drone com o comando mission.forward(2)

Fonte: Próprio Autor

Notou se que esse com esse comando o drone poderia apresentar variação de posição também em y caso na decolagem a angulação em yam fosse alterada, e que a movimentação pela comando *mission.go(x,y)* seria mais precisa uma vez que ela independe do ângulo de rotação em yam.

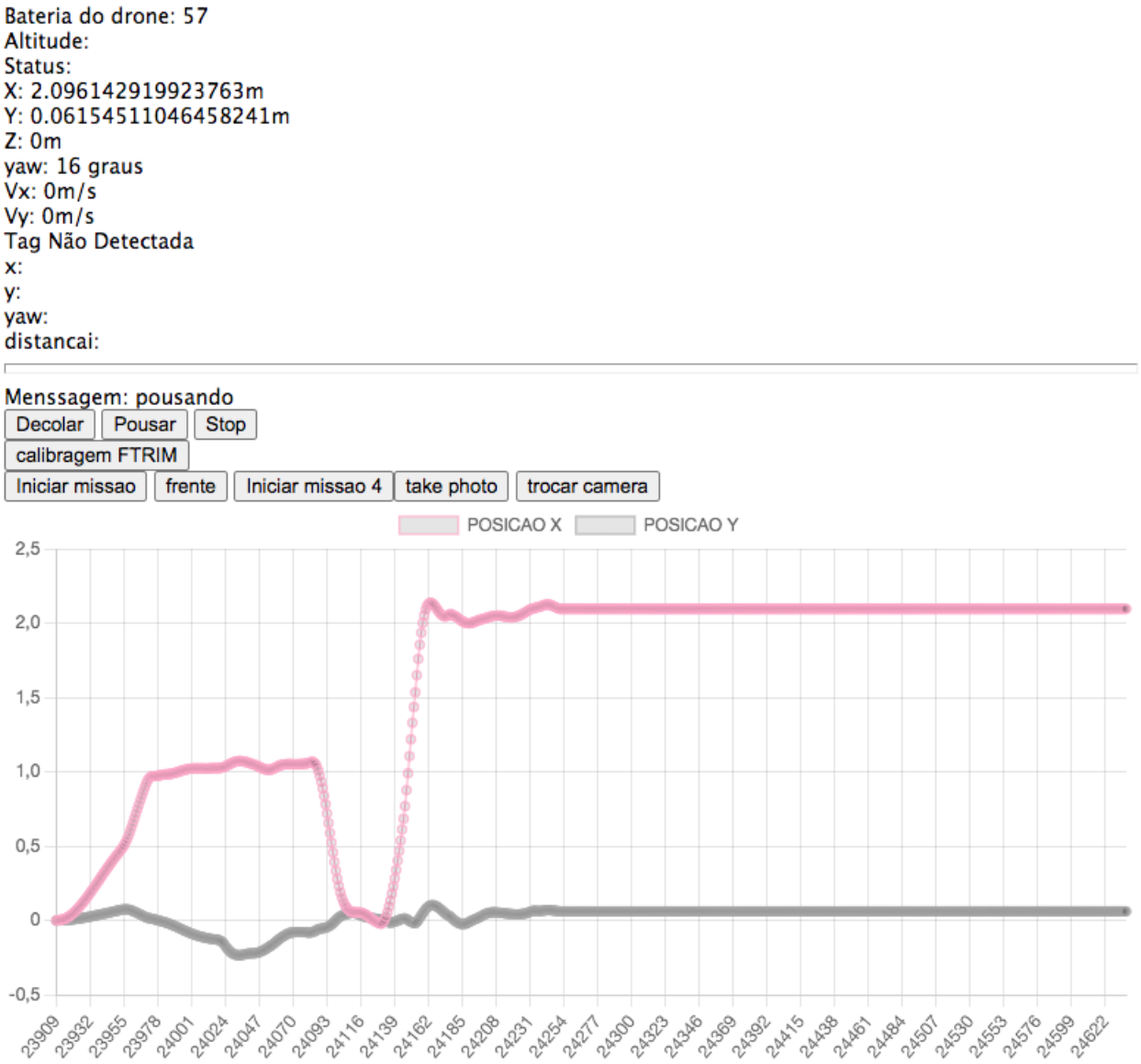


Figura 33 – Teste do drone com o comando *mission.go(2,0)*

Fonte: Próprio Autor

Notou se também que, sem a TAG presente no ponto de decolagem por muitas vezes a posição estimada pela biblioteca não correspondia com real, vista que os sensores inerciais do drone não detectaram pequenas movimentações do drone que ocorriam sem que houvesse inclinação no ângulo de guinada e arfagem, isso fazia com que por diversas vezes a biblioteca estima se que o drone estava na sua posição de decolagem ($x = 0$ e $y = 0$) porém na realidade o drone já tinha se movimentado cerca de um metro em alguma direção. Acumulando erros em todas as próximas movimentações.

O Ar.Drone 2.0 também só apresentou resultados significativos apenas em ambientes fechados uma vez que com a presença do vento essas movimentações não detectadas pelos sensores aumentavam tornando o controle muito menos eficiente mesmo com o modo de

voo externo habilitado. onde as guinadas eram mais abruptas e a movimentação menos precisa.

6.3 Teste de GPS

A maioria dos drone modernos tem como sua principal forma de definição de posicionamento o GPS (Global Positioning System), estes se conectam à satélite ao redor do planeta e com isso triangular sua posição em coordenadas geográficas. Eles chegam a se conectar com cerca de até 20 satélites, o que proporciona uma excelente precisão para suas movimentações em ambientes abertos. O Ar.drone 2.0 possui um módulo original da Parrot, que é conectado na USB da placa principal. Com isso o drone passa a receber e administrar coordenadas geográficas, tal como latitude e longitude. Porém esse módulo possui um custo bastante elevado e pouca disponibilidade de mercado. Sendo assim tentou-se adaptar um módulo GPS ao drone utilizando de um microcontrolador esp32,um módulo GPS GY-NEO6mV2, e para transmitir os dados para o computador um módulo Wireless Nrf24l01 de 2,4ghz. Para receber esses dados foram utilizados o módulo Nrf juntamente com uma placa Arduino uno conectada a porta serial do computador. Foi realizada a montagem no drone conforme a Figura 34 , se utilizando da sua porta USB apenas para alimentar o microcontrolador e os módulos. Os códigos para os dois microcontroladores foram implementados na IDE Arduino, por possuir diversas bibliotecas para o uso desses modelos. Mais um sistema foi implementado utilizando *node.js*, o *framework express* e as bibliotecas *leaflet* e *soket.io* para testar a integração dos dados oriundos do GPS com o sistema.

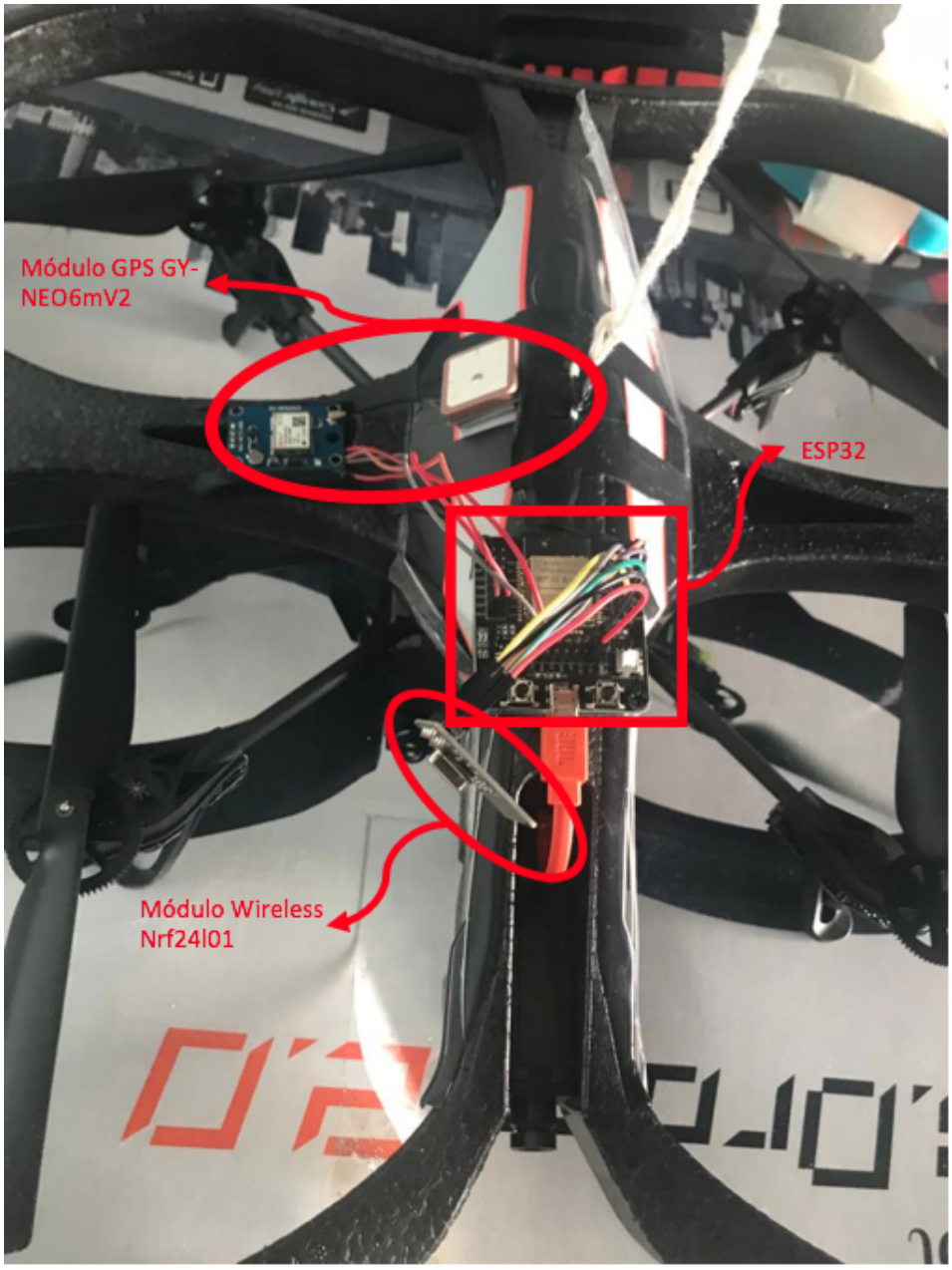


Figura 34 – Montagem do GPS no drone

Fonte: Próprio Autor

Os testes geraram as imagens da Figura 35, Figura 36 e Figura 37, onde se colocava o drone com o GPS em uma posição em que se conhecia suas coordenadas geográficas. Foram feitas 50 leituras válidas de latitude e longitude, sendo esses pontos interligados no mapa pela linha azul, e calculado a distância entre a média dessas leituras e o ponto real do drone, além de se plotar um círculo centrado no ponto medio e de raio igual a maior distância calculada entre cada medida e o ponto médio.

Bateria do drone: desconectado%
SATELITES: 8
Lat/Lon: -15.766824667, -47.870685167

Salvar Posicao

Posicao real (Lat/Lon):-15.766815400551996,-47.87063352763653
Status: distancia Calculanda
SATELITES:
DISTANCIA MEDIA COM 50 AMOSTRAS
Media da posicao (Lat/Lon):-15.766781656639996,-47.87062355000001 ------- Raio:13.985727867867892m
Distancia do ponto real:3.9170411045247113m

Figura 35 – Exempo de teste do GPS 1

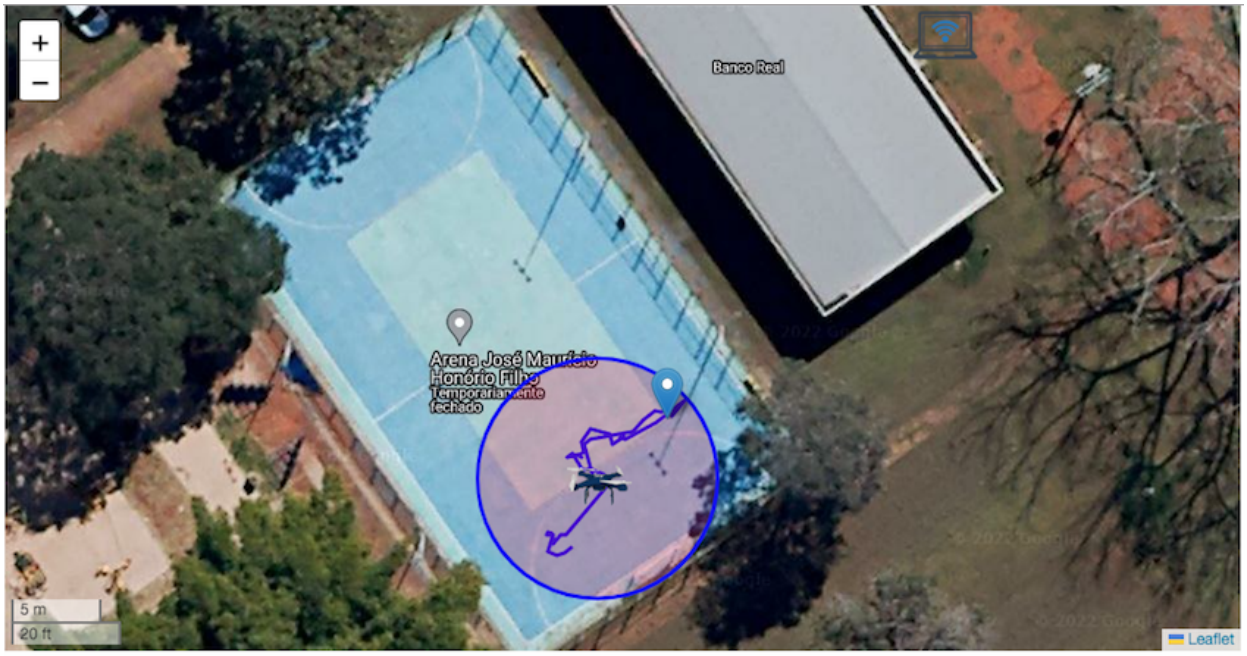
Fonte: Próprio Autor

A distância entre a média das medidas e o ponto foi denominada como "Distância do ponto Real", em que no teste da Figura 35 esse valor foi quase 4m, porém pode se notar pela medida do raio que dentre as 50 medições realizadas há um ponto que está a cerca de 14m da média calculada. Já na Figura 36 pode se perceber que a medidas foram mais precisas, com uma distância de 2m, porém com uma variação entre as medidas menor, cerca de 8m, mas que se fosse utilizado um número maior de medições esse valor seria maior uma vez que no momento do registro o GPS informava que o drone já estava fora do círculo.

Figura 36 – Exempo de teste do GPS 2

Fonte: Próprio Autor

Já no teste da Figura 37, apesar da variação ser a menor (6.5m) a distância da média ao ponto real chegou a mais de 5m, demonstrando a imprecisão dos dados obtidos por esse modelo de GPS. As quantidades de satélites conectados ao GPS mostrado na figura pelo campo "SATELITES"não condiz com a realidade dos testes, uma vez que esses valores eram bastantes inconstantes e mudaram durante as 50 medições. Tais características, somada à demora com que se tinha para que o módulo se conectasse a pelo menos 3 satélites, decidiu-se não utilizar o GPS disponível para fazer o controle da posição do drone, optando-se pelo uso dos sensores inerciais do drone.



Bateria do drone: desconectado%
SATELITES: 7
Lat/Lon: -15.766937, -47.870502667

Salvar Posicao

Posicao real (Lat/Lon):-15.766903729129558,-47.870468236505985
Status: distancia Calculanda
SATELITES:
DISTANCIA MEDIA COM 50 AMOSTRAS
Media da posicao (Lat/Lon):-15.766932903200004,-47.8705038467 ------- Raio:6.648982404011346m
Distancia do ponto real:5.12447189832133m

Figura 37 – Exempo de teste do GPS 3

Fonte: Próprio Autor

7 Considerações finais

7.1 Conclusão

Conclui-se com este trabalho que o sistema projetado para inspecionar torres de linhas de transmissão apresentou resultados bem satisfatórios quando executado com o script que simula a movimentação do drone. Sendo confiável afirmar que na prática seria alcançado o resultado esperado dentro dos quesitos de precisão nos cálculos geradores dos comandos para a movimentação de um drone, assim como o mapeamento de toda movimentação do drone no mapa. Sendo necessário apenas que haja alguma forma de identificar a direção em que os cabos estão passando pela torre, que neste trabalho foi feito através da TAG.

Também é possível a melhoria de sua arquitetura adicionando a possibilidade da criação de novas rotinas de inspeção de torres de forma personalizada, sendo estas criadas pelos usuários, além das já definidas previamente. Ainda há a possibilidade de implementação de salvar e executar rotas já configuradas no sistema.

Contudo , foi analisado que o modelo de drone adotado neste projeto não atendeu aos requisitos para realizar uma inspeção de torres de LTs com segurança devido a imprecisão dos seus movimentos e dos seus sensores internos. Pois, ele precisa visualizar a TAG ou algum outro ponto de referência a todo momento para que consiga apresentar resultados satisfatórios. Mesmo essa deficiência podendo ser sanada pelo uso do seu módulo GPS, que pelo trabalho de Correa (2020) possui um erro relacionado a posição de até 5 metros, sendo admissível para a movimentação de um ponto da rota até outra , mas não o suficiente para realizar as manobras de inspeção ao redor da torre com segurança onde se estima que o erro deveria ser de no máximo 2 a 3 metros. Ademais, existiria o problema dos motores desse modelo, por não serem tão potentes e capazes de resistir tão bem a rajada de vento, podem ser lançados contra as torres vindo a causar grandes prejuízos. Esses requisitos podem ser facilmente sanados por modelos de drones mais modernos.

Porém, mesmo este modelo da Parrot não servindo para inspeção em campo, ele serve para pesquisa, devido a sua SDK, a facilidade de conectividade com computadores, e as divisas bibliotecas já desenvolvidas para a comunicação e controle deste drone.

Referências

AARON, C. **Why you should learn node.js today**. Disponível em: https://blog.udemy.com/learn-node-js/ – Acesso em: 05 fevereiro. 2023. 2013. Citado na p. 38.

ALVES, P. M. **Trabalho Nas Alturas**. Disponível em: https://www.folhadelondrina.com.br/transmidia/trabalho-nas-alturas-2954823e.html – 20 de Setembro de 2022. Fev. 2018. Citado na p. 19.

ANAC. **Regras sobe drones**. Disponível em: https://www.cobra.org.br/documentos/arquivos/regras_drones.pdf – Acesso em 22 de agosto 2022. Citado na p. 20.

BIEH, M. **Geolib**. Dispoínvel em: https://www.npmjs.com/package/geolib – Acesso em: 05 fevereiro. 2023. 2020. Citado na p. 41.

BRAMETAL. **INFRAESTRUTURA DE TRANSMISSÕES ELÉTRICAS -Torres e estruturas metálicas em aço galvanizado BRAMETAL**. 2021. Citado na p. 18.

BRANDAO, A. S. **Projeto de controladores nao lineares para vôo autônomo de veıculos aéreos de pás rotativas**. 2013. Citado na p. 24.

BRASIL. **RESOLUÇÃO NORMATIVA ANEEL Nº 906, DE 8 DE DEZEMBRO DE 2020**. Anexo 3 - 8.Linhas de Transmissão. Ministério de Minas e Energia/Agência Nacional de Energia Elétrica/Diretoria/ANEEL. 16 dez. 2020. Disponível em: <https://www.in.gov.br/en/web/dou/-/resolucao-normativa-aneel-n-906-de-8-de-dezembro-de-2020-294354729>. Acesso em: 16 dez. 2020. Citado na p. 15.

BUZZO, L. **História dos Drones: do início aos dias de hoje**. Disponível em: https://odrones.com.br/historia-dos-drones/ – Acesso em 21 de agosto 2022. Abr. 2015. Citado na p. 20.

CAIAFA, R. **Exército Brasileiro usa drones para monitorar fronteira com o Uruguai**. Disponível em: https://www.infodefensa.com/texto-diario/mostrar/3124915/exercito-brasileiro-drones-monitorar-fronteira-com-uruguai – Acesso em 22 de agosto 2022. Out. 2020. Citado na p. 21.

CANALENERGIA. **EDP recebe certificação da ANAC para monitoramento de redes com uso de drones**. Disponível em: https://www.canalenergia.com.br/noticias/53176328/edp-recebe-certificacao-da-anac-para-monitoramento-de-redes-com-uso-de-drones – Acesso em 27 de Setembro 2022. 11 jun. 2021. Citado na p. 16.

CHART. **Chart.js**. Dispoínvel em: https://www.chartjs.org – Acesso em: 05 fevereiro. 2023. 2023. Citado na p. 62.

CORREA, D. F. **Drone autônomo: Vigilância aérea de espaços externos**. Trabalho de Conclusão de Curso (Bacharel em Sistemas de Informação) - Universidade Regional de Blumenau, Blumenau, 2020 .Disponível em http://dsc.inf.furb.br/arquivos/tccs/monografias/2020_1_diego-fachinello-correa_monografia.pdf. 2020. Citado nas pp. 30, 31, 45, 71.

EMATER. **Como o uso de drones ajuda produtores rurais a economizarem**. Disponível em: https://summitagro.estadao.com.br/tendencias-e-tecnologia/como-o-uso-de-drones-ajuda-produtores-rurais-a-economizarem/ – Acesso em 22 de agosto 2022. Mar. 2020. Citado na p. 21.

ENGEL, J.; STURM, J.; CREMERS, D. **Tum AR.Drone**. Disponível em: http://wiki.ros.org/tum_ardron – Acesso em: 03 fevereiro. 2023. 2014. Citado nas pp. 37, 38.

ESCHENAUER, L. **ardrone-autonomy**. Disponível em: https://www.npmjs.com/package/ardrone-autonomy – Acesso em: 05 fevereiro. 2023. 2014. Citado nas pp. 41, 44.

FORCE, P. **Inspeção com drones em Linhas de Transmissão: conheça as vantagens**. Disponível em: InspeÃğÃčocomdronesemLinhasdeTransmissÃčo:conheÃğaasvantagens – Acesso em 24 de agosto 2022. Jun. 2017. Citado na p. 15.

FUCHS, R. D. **Transmissão de Energia Elétrica - linhas aéreas volume 1**. 1977. Citado nas pp. 50, 51.

GEISENDÖRFER, F. **ar-drone**. Disponível em: https://www.npmjs.com/package/ar-drone – Acesso em: 05 fevereiro. 2023. 2014. Citado nas pp. 38, 41, 42.

HAWKINS, A. J. **Amazon's troubled drone delivery project is finally taking off**. Disponível em: https://www.theverge.com/2022/6/13/23165727/amazon-drone-delivery-pilot-lockeford-faa – Acesso em 22 de agosto 2022. Jun. 2022. Citado na p. 21.

HOMA, J. M. **Aerodinâmica e teoria de voo**. ASA, 2010. Citado na p. 22.

KAHAROVIC, N. **Programming the AR Drone 2.0 using JavaScript and Node.js - Part 1**. Disponível em: https://medium.com/maestral-solutions/programming-the-ar-drone-2-0-using-javascript-and-node-js-part-1-10bb946c60e5 – 20 de Setembro de 2022. Out. 2019. Citado na p. 22.

KLEIN, G.; MURRAY, D. **Parallel tracking and mapping for small AR workspaces**. IEEE. 2007. Citado na p. 38.

LABEGALINI, P. R.; LABEGALINI, J. A.; FUCHS, R. D.; ALMEIDA, M. T. de. **Projetos mecânicos das linhas aéreas de transmissão**. 2. ed.: Editora Blucher, 1992. Citado na p. 18.

LIEDMAN, P. e. a. **Leaflet**. Dispoínvel em: https://www.npmjs.com/package/leaflet – Acesso em: 05 fevereiro. 2023. 2021. Citado na p. 41.

LIMA, F. d. P. **Implementação de controle de rolagem e arfagem com realimentação visual aplicado a um quadrirrotor comercial**. 2013. Citado nas pp. 26, 28, 29.

MAIA, C. **Uso de drones facilita inspeção de linhas de transmissão e distribuição de energia em áreas de difícil acesso**. Disponível em: https://redepara.com.br/n/223261 – Acesso em 22 de agosto 2022. Jan. 2022. Citado na p. 21.

MARTIN, J. **Parrot AR.Drone 2.0 Elite Edition review**. Disponível em: https://www.techadvisor.com/article/715536/parrot-ar-drone-2-0-elite-edition-review.html – 20 de Setembro de 2022. Mar. 2015. Citado na p. 36.

MATTEDE, H. **O que são linhas de transmissão? Características e Curiosidades!** Disponível em: https://www.mundodaeletrica.com.br/o-que-sao-linhas-de-transmissao-caracteristicas-curiosidades/ – Acesso em 27 de Setembro 2022. Citado na p. 17.

MESQUITA, E. d. M. **Desenvolvimento de algoritmos via visão computacional para identificação de local e posterior controle de pouso de um quadrirrotor comercial**. 2015. Citado nas pp. 26, 29, 30.

MONAJJEMI, M. **AL. et. AR.Drone Autonomy: ROS driver for the AR.Drone quadrotor**. Disponível em: http://ardrone-autonomy.readthedocs.org – Acesso em: 03 fevereiro. 2023. Abr. 2015. Citado na p. 37.

MP, T. E. **Equipe de Inspeção em 230kV atuando em Conceição de Mato Dentro - MG**. Disponível em: https://www.folhadelondrina.com.br/transmidia/trabalho-nas-alturas-2954823e.html – 20 de Setembro de 2022. Citado na p. 19.

NETO, S. P. **Afinal, Quais São as Diferenças Entre Drone, Aeromodelo, VANT e RPA?** Disponível em: https://www.austertecnologia.com/single-post/2017/01/31/afinal-quais-sÃčo-as-diferenÃğas-entre-drone-aeromodelo-vant-e-rpa – Acesso em 24 de agosto 2022. Jan. 2017. Citado na p. 21.

ONS. **O sistema em numeros**. Disponível em: http://www.ons.org.br/paginas/sobre-o-sin/o-sistema-em-numeros – Acesso em 27 de Setembro 2022. 2022. Citado nas pp. 15, 17.

PANTOJA, C. R.; GARCIA, P. V. d. A. **Sistema de controle de VANT com emprego de plataforma inercial**. 2017. Citado nas pp. 26–28, 30.

PEDRO, A. T. **Uso de drones melhora inspeções da Energisa**. Disponível em: https://www.canalenergia.com.br/noticias/53058475/uso-de-drones-melhora-inspecoes-da-energisa – Acesso em 27 de Setembro 2022. 18 abr. 2018. Citado na p. 15.

PISKORSKI, S.; BRULEZ, N.; ELINE, P.; D'HAEYER, F. **Ar. drone developer guide**. 2012. Citado nas pp. 23, 34, 40, 41, 57.

QUIGLEY, M.; CONLEY, K.; GERKEY, B.; FAUST, J.; FOOTE, T.; LEIBS, J.; WHEELER, R.; NG, A. Y. et al. **ROS: an open-source Robot Operating System**. Kobe, Japan. 2009. Citado na p. 37.

RANGEL, S. **Drones: a tecnologia disruptiva das aeronaves remotamente pilotadas**. 2019. Citado na p. 21.

SANTANA, L. V. **Sistemas de Navegação e Controle para Veıculos Aéreos Não Tripulados e suas Aplicações**. Tese de Doutorado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Espírito San. 2016. Citado nas pp. 25, 33, 34.

SILVA, B. C. d.; LIBARDI, P. O. **Sistema de controle de um VANT**. 2016. Citado nas pp. 26–28.

SOLUTION, D. **Uso dos drones no combate a incêndios**. Disponível em: https://www.dronessolution.com.br/post/uso-dos-drones-no-combate-a-incendios – Acesso em 22 de agosto 2022. Nov. 2021. Citado na p. 21.

TEIXEIRA, B. F. F. **Controle para automatização de um quadricóptero**. 2018. Citado nas pp. 26, 27.

VENTRELLA, A.; MOSCATO, L.; SOUZA, A. de; SANTOS, M. dos; VITAL, W. **Robô móvel aplicado à inspeção em linhas de transmissão**. 2003. Citado nas pp. 31, 32.

VOESP, H. **SERVIÇOS AÉREOS ESPECIACIALIZADOS**. Disponível em: https://www.passeiohelicopterosp.com.br/helicoptero-reportagem-radio-e-televisao – 20 de Setembro de 2022. Citado na p. 19.

WEISSHUHN, B. e. a. **Dronestream**. Dispoínvel em: https://www.npmjs.com/package/dronestream – Acesso em: 06 fevereiro. 2023. 2013. Citado na p. 54.

WIKIPÉDIA. **V-1**. Disponível em: pt.wikipedia.org/wiki/V-1 – Acesso em 22 de agosto 2022. Citado na p. 21.

WILSON, D. e. a. **Express**. Dispoínvel em: https://www.npmjs.com/package/express – Acesso em: 06 fevereiro. 2023. 2019. Citado na p. 49.

Apêndices

APÊNDICE A – Códigos

A.1 Codigos da Aplicacao

Código A.1 – app.js

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/jquery',express.static(path.join(__dirname,
    'node_modules/jquery/dist')));
app.use('/leaflet',express.static(path.join(__dirname,
    'node_modules/leaflet/dist')));
app.use('/dronestream',express.static(path.join(__dirname,
    'node_modules/dronestream/dist')));

app.use('/', indexRouter);
app.use('/users', usersRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err :
     {};
```

```
39
40    // render the error page
41    res.status(err.status || 500);
42    res.render('error');
43  });
44
45  module.exports = app;
```

Código A.2 – pagina1.js

```
1
2   var baseIcon = L.icon({
3       iconUrl: '../images/helioporto.png',
4       iconSize:     [50, 50],
5       iconAnchor:   [25, 25]
6   });
7
8   var map,
9       escala,
10      rota;
11
12  var base = undefined,
13      baseMarkers
14  var waypointMarkers = [];
15  var torres = [];
16  var torre = {lat:undefined,lng:undefined,tipo:undefined};
17  var waypoints = [];
18
19
20
21  var inicio = document.getElementById("inicio");
22  inicio.showModal()
23
24  var setBase = document.getElementById("setBase");
25  var setBaseMap = document.getElementById("setBaseMap");
26
27
28  setBase.onclick = function (){
29      base = [document.getElementById("latBase").value,
30              document.getElementById("lngBase").value]
31      escala = 20
32      initMap1(base[0],base[1])
33      waypoints.push(base)
34      baseMarkers = L.marker(base, { icon: baseIcon }).addTo(map)
35      inicio.close()
36  }
37
38  setBaseMap.onclick = function (){
39      escala = 18
40      navigator.geolocation.getCurrentPosition(
41          initMap0, defaultMap,{ enableHighAccuracy: true })
42      inicio.close()
```

```
43
44 }
45 function initMap0(position) {
46     map = L.map('map').setView([position.coords.latitude,
47         position.coords.longitude],escala);
48
49     var googleHybrid =
          L.tileLayer('http://{s}.google.com/vt/lyrs=s,h&x={x}&y={y}&z={z}',{
50     maxZoom: 22,
51     subdomains:['mt0','mt1','mt2','mt3']
52     });
53     map.addLayer(googleHybrid);
54
55
56     scala = L.control.scale().addTo(map)
57
58     //laptop = L.marker([Lat,Lng], { icon: laptopIcon }).addTo(map)
59
60     map.on('click', function (e) {
61         if(base == undefined){
62             baseMarkers = L.marker(e.latlng, { icon: baseIcon
                  }).addTo(map)
63             base = [e.latlng.lat,e.latlng.lng]
64             waypoints.push(base)
65         }
66         else{torre.lat = e.latlng.lat
67             torre.lng = e.latlng.lng
68             waypointMarkers.push(L.marker(e.latlng).addTo(map))
69             popup.showModal()
70
71             $('#waypoint .lat').text(torre.lat)
72             $('#waypoint .lon').text(torre.lng)
73             $('#waypoint .tipo').text(" ")
74         }
75     });
76 }
77 function initMap1(Lat,Lng) {
78     map = L.map('map').setView([Lat,Lng],escala);
79
80     var googleHybrid =
          L.tileLayer('http://{s}.google.com/vt/lyrs=s,h&x={x}&y={y}&z={z}',{
81     maxZoom: 22,
82     subdomains:['mt0','mt1','mt2','mt3']
83     });
84     map.addLayer(googleHybrid);
85
86
87     scala = L.control.scale().addTo(map)
88
89     //laptop = L.marker([Lat,Lng], { icon: laptopIcon }).addTo(map)
90
91     map.on('click', function (e) {
```

```
92          if(base == undefined){
93              baseMarkers = L.marker(e.latlng, { icon: baseIcon
                    }).addTo(map)
94              base = [e.latlng.lat,e.latlng.lng]
95              waypoints.push(base)
96          }
97          else{torre.lat = e.latlng.lat
98              torre.lng = e.latlng.lng
99              waypointMarkers.push(L.marker(e.latlng).addTo(map))
100             popup.showModal()
101
102             $('#waypoint .lat').text(torre.lat)
103             $('#waypoint .lon').text(torre.lng)
104             $('#waypoint .tipo').text(" ")
105         }
106     });
107 }
108 function defaultMap(err){
109     console.log("falha ao iniciar o mapa" + err)
110     escala = 12
111     initMap1(-15.844446926217513,-48.02418529987335)
112 }
113
114 function clearWaypoints() {
115     waypoints = []
116     waypoints.push(base)
117     map.removeLayer(rota)
118     rota = undefined
119     $.each(waypointMarkers, function (i, m) { map.removeLayer(m) })
120 }
121
122 function setCurrentTarget(lat, lon) {
123     targetLat = lat
124     targetLon = lon
125     socket.emit('prossima torre', { lat: targetLat, lon: targetLon
            })
126 }
127
128 var clear = document.getElementById('limpar');
129 var concluir = document.getElementById('concluir');
130
131 var popup = document.getElementById("popup");
132 var tipoA = document.getElementById("tipoA");
133 var tipoB = document.getElementById("tipoB");
134 var cancelar = document.getElementById("cancelar");
135
136 clear.onclick = function (){
137     console.log("limpar")
138     clearWaypoints()
139
140 }
141
```

```
142 concluir.onclick = function (){
143     console.log("rota concluida")
144     socket.emit("RotaConcluida",{torres:torres, base: base})
145     window.location.href="p2"
146 }
147 
148 tipoA.onclick = function (){
149     console.log("Tipo A")
150     torre.tipo = 'A'
151     torres.push([torre.lat,torre.lng,torre.tipo])
152     waypoints.push([torre.lat,torre.lng])
153     if (   rota == undefined) {
154         rota = L.polyline(waypoints, { color: 'blue' }).addTo(map);
155     } else {
156         rota.setLatLngs(waypoints)
157     }
158     popup.close()
159     $('#waypoint .tipo').text(torre.tipo)
160 }
161 
162 tipoB.onclick = function (){
163     console.log("Tipo B")
164     torre.tipo = 'B'
165     torres.push([torre.lat,torre.lng,torre.tipo])
166     waypoints.push([torre.lat,torre.lng])
167     if (   rota == undefined) {
168         rota = L.polyline(waypoints, { color: 'blue' }).addTo(map);
169     } else {
170         rota.setLatLngs(waypoints)
171     }
172     popup.close()
173     $('#waypoint .tipo').text(torre.tipo)
174 }
175 
176 cancelar.onclick = function (){
177     console.log("cancelar")
178     popup.close()
179     map.removeLayer(waypointMarkers[waypointMarkers.length - 1])
180     $('#waypoint .lat').text(torres[torres.length-1][0])
181     $('#waypoint .lon').text(torres[torres.length-1][1])
182     $('#waypoint .tipo').text(torres[torres.length-1][2])
183 }
184 
185 var socket = io.connect('/');
186 
187 socket.on('connect', function () {
188 
189 })
```

Código A.3 – pagina2.js

```
1 var baseIcon = L.icon({
```

```
    iconUrl: '../images/helioporto.png',
    iconSize:     [50, 50],
    iconAnchor:   [25, 25]
});
var laptopIcon = L.icon({
    iconUrl: '../images/laptop.png',
    iconSize:     [50, 50],
    iconAnchor:   [25, 25]
});

function foto(centro){
    var circle
    circle = L.circle(centro, {
    color: '#f03',
    fillColor: '#f03',
    fillOpacity: 0.2,
    radius: 1
    })
    circle.addTo(map);
}

var drone = {
    status: undefined,
    posi: undefined,
    icon: L.icon({
        iconUrl: '../images/drone.gif',
        iconSize:     [50, 50],
        iconAnchor:   [25, 25]
    }),
    trajetoria:[],
    trajetoriaMarkers:undefined
}

var socket = io.connect('/');

socket.emit("pag2")

socket.on('connect', function () {
    socket.on("carregarPag2",(data)=>{
        base = data.base
        torres = data.torres
        console.log(torres)
        initMap(base)
        $('#inspecao .baseLat').text(base[0])
        $('#inspecao .baseLng').text(base[1])

    })
    socket.on("droneData",(data)=>{
        drone.posi = data.drone.posi
        drone.trajetoria.push(drone.posi)
        drone.posMake.setLatLng(drone.posi)
        if (   drone.trajetoriaMarkers == undefined) {
```

```
54              drone.trajetoriaMarkers = L.polyline(drone.trajetoria,
                    { color: 'blue' }).addTo(map);
55          } else {
56              drone.trajetoriaMarkers.setLatLngs(drone.trajetoria)
57          }
58          $('#drone .bateria').text(data.drone.bateria)
59          $('#drone .altitude').text(data.drone.altitude)
60          $('#drone .lat').text(data.drone.posi[0])
61          $('#drone .lng').text(data.drone.posi[1])
62      })
63      socket.on("menssagem",(data)=>{
64          console.log(data.str)
65          $('#inspecao .str').text(data.str)
66      })
67      socket.on("foto",(data)=>{
68          foto(data.centro)
69      })
70  })
71
72
73  var map,
74      rota;
75
76  var base,
77      baseMarkers
78  var waypointMarkers = [];
79  var torres = [];
80
81
82
83
84
85
86
87  function initMap(base) {
88      map = L.map('map').setView(base,18);
89
90      var googleHybrid =
            L.tileLayer('http://{s}.google.com/vt/lyrs=s,h&x={x}&y={y}&z={z}',{
91      maxZoom: 22,
92      subdomains:['mt0','mt1','mt2','mt3']
93      });
94      map.addLayer(googleHybrid);
95
96
97      L.control.scale().addTo(map)
98
99      baseMarkers = L.marker(base, { icon: baseIcon }).addTo(map)
100     drone.posi = base
101     drone.posMake = L.marker(drone.posi, { icon:
            drone.icon}).addTo(map)
102     try {
```

```
103         torres.forEach(function(torre){
104             console.log(torre[0],torre[1])
105             waypointMarkers.push(L.marker([torre[0],torre[1]]).addTo(map))
106         })
107     } catch (error) {
108 
109             console.log(torres[0],torres[1])
110             waypointMarkers.push(L.marker([torres[0],torres[1]]).addTo(map))
111 
112     }
113 
114 }
115 
116 new NodecopterStream(document.getElementById("droneStream"));
117 
118 
119 var iniciarMissao= document.getElementById("iniciarMissao");
120 
121 iniciarMissao.onclick = function(){
122     drone.trajetoria.push(drone.posi)
123     socket.emit("iniciarMissao")
124 }
```

Código A.4 – pagina1.ejs

```
1  <!DOCTYPE html>
2  <html lang="pt-br">
3    <head>
4      <title><%= title %></title>
5      <meta charset="UTF-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width,
          initial-scale=1.0">
8 
9      <link rel="stylesheet" href="/leaflet/leaflet.css"/>
10     <link rel='stylesheet' href='/stylesheets/style.css' />
11   </head>
12   <body>
13     <h1><%= title %></h1>
14     <dialog id = inicio>
15 
16       <h2>Defina a pisição de decolagem</h2><br />
17       <p>Ex(lat,lng):-15.844446926217513,-48.02418529987335</p>
18       <p>Latitude :<input type="number" id="latBase"  /><br />
19          Longitude:<input type="number" id="lngBase"  /></p>
20       <button id='setBase'>definir</button><br /><br />
21       <button id='setBaseMap'>definirno mapa</button>
22     </dialog>
23     <div id="map"></div>
24     <dialog id = popup>
25       <h2>Defina o tipo de inspeção</h2> <br/>
26       <button id='tipoA'>Torre tipo A</button>
```

```
      <button id='tipoB'>Torre tipo B</button>
      <button id='cancelar'>cancelar</button>
    </dialog>
    <div id="popup"></div>
    <div id="waypoint">
      <br />
      Ultima torre:<br /> Lat: <span class='lat'></span>,
      Lng: <span class='lon'></span>,
      Tipo: <span class='tipo'></span>
      <br />
    </div>
    <button id='limpar'>Limpar rota</button><br />
    <button id='concluir'>concluir rota</button><br />

    <script src="/socket.io/socket.io.js"></script>
    <script src="/leaflet/leaflet.js"></script>
    <script src="/jquery/jquery.min.js"></script>
    <script src="/javascripts/index.js"></script>
  </body>
</html>
```

Código A.5 – pagina2.ejs

```
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <title><%= title %></title>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
       initial-scale=1.0">

    <link rel="stylesheet" href="/leaflet/leaflet.css"/>
    <link rel='stylesheet' href='/stylesheets/style2.css' />
  </head>
  <style>
    .gridconteiner{
      display: grid;
      grid-template-columns: 640px 640px;
      grid-template-rows: 360px 360px;
      column-gap: 5px;

    }

  </style>
  <body>

    <h1>Acompanhamento Da Inspeção</h1>
    <br/>
    <div class="gridconteiner">
      <div id="map" class="map"></div>
      <div id="droneStream" class="droneStream"></div>
```

```
      <div id="inspecao" class="inspecao" >
        <br/>
        Mensagem:
        <span class="str"></span>
        <br/>
        Base:
        <br/>
        Lat:
        <span class="baseLat"></span>
        <br/>
        Lng:
        <span class="baseLng"></span>
      </div>
      <div id="drone" class="drone">
        <br/>
        <button id="iniciarMissao">Iniciar Missão</button>
        <button id="stop">Stop</button>
        <button id="goToHome">Pousar</button>
        <button id="goToHome">Go to Home</button>
        <br/>
        Bateria do drone:
        <span class='bateria'>0</span>%
        <br/>
        Altitude :<span class='altitude'></span>m
        <br/>
        Latitude :<span class='lat'></span>
        </br>
        Longitude:<span class='lng'></span>

      </div>
    </div>









    <script src="/dronestream/nodecopter-client.js"></script>
    <script src="/socket.io/socket.io.js"></script>
    <script src="/leaflet/leaflet.js"></script>
    <script src="/jquery/jquery.min.js"></script>
    <script src="/javascripts/index2.js"></script>
  </body>
</html>
```

Código A.6 – package.json

```
{
  "name": "droneapp",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/server.js"
```

```
  },
  "dependencies": {
    "ar-drone": "^0.3.3",
    "ardrone-autonomy": "^0.1.2",
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "dronestream": "^1.1.1",
    "ejs": "^3.1.8",
    "express": "^4.18.2",
    "geolib": "^3.3.3",
    "http-errors": "~1.6.3",
    "jquery": "^3.6.1",
    "leaflet": "^1.9.2",
    "morgan": "~1.9.1",
    "socket.io": "^4.5.3"
  },
  "devDependencies": {
    "nodemon": "^2.0.20"
  }
}
```

A.2 Teste do sistema

Código A.7 – sesver.js

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */
var app = require('../app');
var debug = require('debug')('droneapp:server');
var http = require('http');
/**
 * Get port from environment and store in Express.
 */
var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);
/**
 * Normalize a port into a number, string, or false.
 */
function normalizePort(val) {
  var port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
```

```
26      // port number
27      return port;
28    }
29
30    return false;
31  }
32  /**
33   * Event listener for HTTP server "error" event.
34   */
35  function onError(error) {
36    if (error.syscall !== 'listen') {
37      throw error;
38    }
39
40    var bind = typeof port === 'string'
41      ? 'Pipe ' + port
42      : 'Port ' + port;
43
44    // handle specific listen errors with friendly messages
45    switch (error.code) {
46      case 'EACCES':
47        console.error(bind + ' requires elevated privileges');
48        process.exit(1);
49        break;
50      case 'EADDRINUSE':
51        console.error(bind + ' is already in use');
52        process.exit(1);
53        break;
54      default:
55        throw error;
56    }
57  }
58  /**
59   * Event listener for HTTP server "listening" event.
60   */
61  function onListening() {
62    var addr = server.address();
63    var bind = typeof addr === 'string'
64      ? 'pipe ' + addr
65      : 'port ' + addr.port;
66    debug('Listening on ' + bind);
67  }
68
69  //-----------------------------------------------------------
70  /**
71   * Create HTTP server.
72   */
73
74  var server = http.createServer(app);
75
76  const { Server } = require("socket.io");
77  const io = new Server(server);
```

```
78
79  var dronestream = require("dronestream/index");
80  var geolib = require("geolib");
81  const { send } = require('process');
82
83  /**
84   * Listen on provided port, on all network interfaces.
85   */
86  //dronestream.listen(server);
87  server.listen(port);
88  server.on('error', onError);
89  server.on('listening', onListening);
90
91  var drone={
92    bateria:87,
93    conect: false,
94    posi:[],
95    states:{x:0,y:0,yaw:0},
96    altitude:0,
97    insp:undefined,
98    camera : 0
99  }
100 var E =[]
101 var torres,
102   torre,
103   base,
104   flag = 6,
105   flagFim = false,
106   tag=true
107 io.sockets.on('connection', function (socket) {
108   console.log('connection');
109   socket.on("RotaConcluida",(data)=> {
110     torres = data.torres
111     base = data.base
112     console.log("dados rcebidos")
113     console.log(base)
114     console.log(torres)
115
116   })
117   socket.on("pag2",()=> {
118     io.sockets.emit("carregarPag2",{torres,base})
119     iniciaDrone()
120   })
121
122   socket.on("iniciarMissao",()=> {
123     iniciaMissao()
124   })
125
126
127   setInterval(function () {
128     if(flag==0){
129         io.sockets.emit("menssagem",{str:"drone anda"})
```

```
130         drone.states.x += dx
131         drone.states.y += dy
132         var loc =
               calculaLatLog(torre[0],torre[1],drone.states.x,drone.states.y)
133         drone.posi =[loc.latitude,loc.longitude]
134         io.sockets.emit("droneData",{drone})
135         console.log(drone.posi)
136         try {
137           E.push(geolib.getPreciseDistance({ latitude:
                 drone.posi[0], longitude: drone.posi[1] },
138             { latitude: torres[0][0], longitude:  torres[0][1]},
                   accuracy = 0.01))
139         } catch (error) {
140           E.push(geolib.getPreciseDistance({ latitude:
                 drone.posi[0], longitude: drone.posi[1] },
141             { latitude: base[0], longitude:  base[1]}, accuracy =
                   0.01))
142         }
143
144         flag++
145     }else{
146       if(flag==1){
147         if(!flagFim){
148           torreAtingida()
149           flag++
150         }
151         else{
152           flag=3
153         }
154       } else{
155         if(flag==2){
156           iniciarInspecao(drone.insp)
157           flag++
158         } else{
159           if(flag==3){
160             if(!flagFim){
161               proximaTorre()
162             }
163             else{
164               console.log(E)
165               pouso()
166               flag++
167             }
168           }
169         }
170       }
171     }
172
173     /*if(drone.conect){
174       io.sockets.emit('droneData', { drone })
175
176
```

```
177     } else{
178       io.sockets.emit('desconect')
179       drone.conect = true
180     }*/
181   }, 3000)
182 })
183 
184 
185 var dx
186 var dy
187 
188 function iniciaMissao(){
189   drone.altitude = 30
190   torre = base
191   setTrajectory(torres[0])
192   dx = displacementVector[0]
193   dy = displacementVector[1]
194   io.sockets.emit("menssagem",{str:"missao iniciada "})
195   flag = 0
196   console.log(base)
197   console.log(torres)
198 }
199 
200 
201 function iniciaDrone(){
202   drone.posi = base
203 }
204 
205 
206 
207 
208 
209 function setTrajectory(destino){
210    // Calcula distancia para a proxima torre
211   distance = geolib.getPreciseDistance({ latitude: drone.posi[0],
       longitude: drone.posi[1] },
212   { latitude: destino[0], longitude:  destino[1]}, accuracy = 1);
213   console.log("Distancia: ",distance)
214 
215   // direcao da proxima torre
216   direction =  geolib.getGreatCircleBearing({ latitude:
       drone.posi[0], longitude: drone.posi[1] },
217     { latitude: destino[0], longitude:  destino[1]});
218   //console.log("direcao: ",direction)
219 
220   yawAdjustment = direction
221 
222   direction = direction*Math.PI/180
223   //console.log(direction)
224   // Encontra os componentes X e Y do vetor deslocamento
225   displacementVector = [Math.cos(direction) * distance,
       Math.sin(direction) * distance];
```

```
226   // Calcula o ângulo necessário para girar para que o drone fique
          voltado para a proxima torre
227   //console.log("X,Y : ",displacementVector)
228
229
230   console.log("rotacionar: ",yawAdjustment, " graus")
231 }
232 function calculaLatLog(lat0,lng0,x,y){
233   var displacement = Math.hypot(x,y);
234   //console.log(torres[0][0],torres[0][1])
235   //console.log(yawAdjustment)
236   var loc =geolib.computeDestinationPoint(
237     { latitude: lat0, longitude: lng0 },
238     displacement,
239     yawAdjustment);
240
241     return(loc)
242 }
243  function torreAtingida(){
244   io.sockets.emit("menssagem",{str:"torre atingida:
          "+torres[0][0]+", "+torres[0][1]})
245   torre = torres.shift()
246   //drone.posi=[torre[0],torre[1]]
247   drone.insp=torre[2]
248   //console.log(torre)
249
250  }
251
252  function iniciarInspecao(tipo){
253     io.sockets.emit("menssagem",{str:"buscando tag"})
254     if(tag){
255       io.sockets.emit("menssagem",{str:"tag localizada"})
256       drone.posi=[torre[0],torre[1]]
257       io.sockets.emit("droneData",{drone})
258       if(tipo=='A'){
259         inspecaoA()
260       }
261       if(tipo=='B'){
262         inspecaoB()
263       }
264     }
265     else{
266       io.sockets.emit("menssagem",{str:"tag não localizada"})
267     }
268  }
269
270  function proximaTorre(){
271     if(torres.length>0){
272       io.sockets.emit("menssagem",{str:"proxima torra: "
             +torres[0][0]+", "+torres[0][1]})
273       drone.altitude = 30
274       drone.states={x:0,y:0,yaw:0},
```

```
275         setTrajectory(torres[0])
276         dx = displacementVector[0]
277         dy = displacementVector[1]
278         flag = 0
279       }
280       else{
281         io.sockets.emit("menssagem",{str:"Fim da missao. vontando
              para base:" +base[0]+", "+base[1]})
282         drone.altitude = 30
283         drone.states={x:0,y:0,yaw:0},
284         setTrajectory(base)
285         flagFim =true
286         dx = displacementVector[0]
287         dy = displacementVector[1]
288         flag = 0
289       }
290  }
291
292 function go(u,v){
293   theta = 66*Math.PI/180;
294
295   // diamertoda terra = 12742km = d
296   // c=(d*pi)/360 -> *1000 passando para metros
297   const c = 111194.926644559
298
299
300   var Dx=Math.cos(theta)*u - Math.sin(theta)*v
301   var Dy=Math.sin(theta)*u + Math.cos(theta)*v
302
303   drone.posi[0]=torre[0]+(Dx/c)
304   drone.posi[1]=torre[1]+(Dy/c)
305
306   io.sockets.emit("droneData",{drone})
307 }
308 function yaw(ang){
309   io.sockets.emit("menssagem",{str:"girando o drone paar o sentido
        de  " + ang +"gragraus"})
310 }
311
312
313 function foto(centro){
314   io.sockets.emit("foto",{centro:centro})
315 }
316
317 function inspecaoA(){
318   io.sockets.emit("menssagem",{str:"iniciando inspecao tipo A na
        torre: " +torre[0]+", "+torre[1]})
319   drone.altitude = 28
320   io.sockets.emit("menssagem",{str:"decendo para:
        "+drone.altitude+"m"})
321   io.sockets.emit("menssagem",{str:"posicao 0"})
322   io.sockets.emit("menssagem",{str:"foto com a camera vertical"})
```

```
323   foto(drone.posi)
324
325   go(0,6)
326   drone.altitude = 22
327   io.sockets.emit("menssagem",{str:"descendo para:
         "+drone.altitude+"m"})
328   yaw(-90)
329   io.sockets.emit("menssagem",{str:"posicao 1 "})
330   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
331   foto(drone.posi)
332
333   drone.altitude = 18
334   io.sockets.emit("menssagem",{str:"descendo para:
         "+drone.altitude+"m"})
335   io.sockets.emit("menssagem",{str:"posicao 2 "})
336   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
337   foto(drone.posi)
338
339   drone.altitude = 14
340   io.sockets.emit("menssagem",{str:"descendo para:
         "+drone.altitude+"m"})
341   io.sockets.emit("menssagem",{str:"posicao 3 "})
342   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
343   foto(drone.posi)
344
345   go(-6,6)
346   yaw(-45)
347   io.sockets.emit("menssagem",{str:"posicao 4 "})
348   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
349   foto(drone.posi)
350
351   drone.altitude = 10
352   io.sockets.emit("menssagem",{str:"descendo para:
         "+drone.altitude+"m"})
353   go(-6,0)
354   yaw(0)
355   io.sockets.emit("menssagem",{str:"posicao 5 "})
356   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
357   foto(drone.posi)
358
359   go(-6,-6)
360   yaw(45)
361   io.sockets.emit("menssagem",{str:"posicao 6"})
362   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
363   foto(drone.posi)
364
365   go(0,-6)
366   yaw(90)
367   io.sockets.emit("menssagem",{str:"posicao 7"})
368   drone.altitude = 13
369   io.sockets.emit("menssagem",{str:"subindo para:
         "+drone.altitude+"m"})
```

```
370   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
371   foto(drone.posi)
372
373   drone.altitude = 17
374   io.sockets.emit("menssagem",{str:"subindo para:
          "+drone.altitude+"m"})
375   io.sockets.emit("menssagem",{str:"posicao 8"})
376   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
377   foto(drone.posi)
378
379   drone.altitude = 21
380   io.sockets.emit("menssagem",{str:"subindo para:
          "+drone.altitude+"m"})
381   io.sockets.emit("menssagem",{str:"posicao 9"})
382   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
383   foto(drone.posi)
384
385   go(6,-6)
386   yaw(135)
387   io.sockets.emit("menssagem",{str:"posicao 10"})
388   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
389   foto(drone.posi)
390   drone.altitude = 28
391   io.sockets.emit("menssagem",{str:"subindo para:
          "+drone.altitude+"m"})
392   io.sockets.emit("menssagem",{str:"posicao 11"})
393   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
394   foto(drone.posi)
395
396   go(6,0)
397   yaw(180)
398   io.sockets.emit("menssagem",{str:"posicao 12"})
399   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
400   foto(drone.posi)
401
402   go(0,0)
403   yaw(0)
404   io.sockets.emit("menssagem",{str:"posicao 0"})
405   io.sockets.emit("menssagem",{str:"fim da inspecao tipo A na
          torre: " +torre[0]+", "+torre[1]})
406 }
407
408 function inspecaoB(){
409   io.sockets.emit("menssagem",{str:"iniciando inspecao tipo B na
          torre: " +torre[0]+", "+torre[1]})
410   drone.altitude = 25
411   io.sockets.emit("menssagem",{str:"decendo para:
          "+drone.altitude+"m"})
412   io.sockets.emit("menssagem",{str:"posicao 0"})
413   io.sockets.emit("menssagem",{str:"foto com a camera vertical"})
414   foto(drone.posi)
415
```

```
416   go(0,7)
417   io.sockets.emit("menssagem",{str:"posicao 1"})
418   io.sockets.emit("menssagem",{str:"foto com a camera vertical"})
419   foto(drone.posi)
420
421   go(0,10)
422   drone.altitude = 17
423   io.sockets.emit("menssagem",{str:"descendo para:
          "+drone.altitude+"m"})
424   yaw(-90)
425   io.sockets.emit("menssagem",{str:"posicao 2 "})
426   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
427   foto(drone.posi)
428
429   go(-10,10)
430   yaw(-45)
431   io.sockets.emit("menssagem",{str:"posicao 3"})
432   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
433   foto(drone.posi)
434
435   drone.altitude = 10
436   io.sockets.emit("menssagem",{str:"descendo para:
          "+drone.altitude+"m"})
437   go(-10,0)
438   yaw(0)
439   io.sockets.emit("menssagem",{str:"posicao 4 "})
440   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
441   foto(drone.posi)
442
443   go(-10,-10)
444   drone.altitude = 17
445   io.sockets.emit("menssagem",{str:"subindo para:
          "+drone.altitude+"m"})
446   yaw(45)
447   io.sockets.emit("menssagem",{str:"posicao 5 "})
448   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
449   foto(drone.posi)
450
451   go(0,-10)
452   yaw(90)
453   io.sockets.emit("menssagem",{str:"posicao 6 "})
454   io.sockets.emit("menssagem",{str:"foto com a camera frontal"})
455   foto(drone.posi)
456   drone.altitude = 25
457   io.sockets.emit("menssagem",{str:"subindo para:
          "+drone.altitude+"m"})
458
459   go(0,-7)
460   io.sockets.emit("menssagem",{str:"posicao 7 "})
461   io.sockets.emit("menssagem",{str:"foto com a camera vertical"})
462   foto(drone.posi)
463
```

```
464   go(0,0)
465   yaw(0)
466   io.sockets.emit("menssagem",{str:"posicao 0"})
467   io.sockets.emit("menssagem",{str:"fim da inspecao tipo B na
          torre: " +torre[0]+", "+torre[1]})
468  }
469
470
471  function pouso(){
472   io.sockets.emit("menssagem",{str:"Fim da missao. drone
          pousando..."})
473   drone.altitude=0
474   io.sockets.emit("menssagem",{str:"Fim da missao. drone no chao:"
          +drone.posi[0]+", "+drone.posi[1]})
475  }
```

A.3 Teste do Drone

Código A.8 – app.js

```
1  var createError = require('http-errors');
2  var express = require('express');
3  var path = require('path');
4  var cookieParser = require('cookie-parser');
5  var logger = require('morgan');
6
7  var indexRouter = require('./routes/index');
8  var usersRouter = require('./routes/users');
9
10 var app = express();
11
12 // view engine setup
13 app.set('views', path.join(__dirname, 'views'));
14 app.set('view engine', 'ejs');
15
16 app.use(logger('dev'));
17 app.use(express.json());
18 app.use(express.urlencoded({ extended: false }));
19 app.use(cookieParser());
20 app.use(express.static(path.join(__dirname, 'public')));
21
22 app.use('/jquery',express.static(path.join(__dirname,
       'node_modules/jquery/dist')));
23 app.use('/dronestream',express.static(path.join(__dirname,
       'node_modules/dronestream/dist')));
24 app.use('/chart',express.static(path.join(__dirname,
       'node_modules/chart.js/dist')));
25 app.use('/google-charts',express.static(path.join(__dirname,
       'node_modules/google-charts/dist')));
26
```

```
app.use('/', indexRouter);
app.use('/users', usersRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err :
     {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

Código A.9 – pagina1.js

```
var drone  = {
    vetSN:[],
    vetx: [],
    vety:[],
    vetz:[],
    vetyaw:[],
    vetVx:[],
    vetVy:[],
    vetAut: [],
    vetTrans:{x:[],y:[]},

    vetAccelerometers: {x: [],y: [],z: []},
    VetGyroscopes: {x: [],y: [],z: []}}
var voando = false
var socket = io.connect('/');

new NodecopterStream(document.getElementById("droneStream"));

socket.on('connect', function () {
    socket.on('droneData', function (data) {

        $('#drone .bateria').text(data.drone.bateria)
        /*
        $('#drone .altitude').text(data.drone.altitude)
        $('#drone .rotX').text(data.drone.rotation.roll)
        $('#drone .rotY').text(data.drone.rotation.pitch)
        $('#drone .rotZ').text(data.drone.rotation.yaw)
        $('#drone .x').text(data.drone.translation.x)
```

```
29          $('#drone .y').text(data.drone.translation.y)
30          $('#drone .z').text(data.drone.translation.z)
31          */
32
33          $('#state .x').text(data.drone.state.x)
34          $('#state .y').text(data.drone.state.y)
35          $('#state .z').text(data.drone.state.z)
36          $('#state .yaw').text((data.drone.state.yaw * 180 /
               Math.PI).toFixed(0))
37          $('#state .pitch').text((data.drone.state.pitch * 180 /
               Math.PI).toFixed(0))
38          $('#state .roll').text((data.drone.state.roll * 180 /
               Math.PI).toFixed(0))
39          $('#state .vx').text(data.drone.state.vx)
40          $('#state .vy').text(data.drone.state.vy)
41          $('#state .ax').text(data.drone.state.ax)
42          $('#state .ay').text(data.drone.state.ay)
43
44
45          if(voando){
46            drone.vetSN.push(data.drone.sequenceNumber)
47            drone.vetx.push(data.drone.state.x)
48            drone.vety.push(data.drone.state.y)
49            drone.vetz.push(data.drone.state.z)
50            drone.vetyaw.push((data.drone.state.yaw * 180 /
                 Math.PI).toFixed(0))
51            drone.vetVx.push(data.drone.state.vx)
52            drone.vetVy.push(data.drone.state.vy)
53            drone.vetAut.push(data.drone.altitude)
54            drone.vetTrans.x.push(data.drone.translation.x)
55            drone.vetTrans.y.push(data.drone.translation.y)
56            //drone.vetAccelerometers.x.push(data.drone.accelerometers.x)
57            //drone.vetAccelerometers.y.push(data.drone.accelerometers.y)
58            drone.vetAccelerometers.x.push(data.drone.state.ax)
59            drone.vetAccelerometers.y.push(data.drone.state.ay)
60          }
61
62
63
64
65          if(data.drone.tag.d > 0){
66            console.log("tag detect")
67              $('#tag .detect').text("Detectada")
68              $('#tag .x').text(data.drone.tag.x)
69              $('#tag .y').text(data.drone.tag.y)
70              $('#tag .yaw').text(data.drone.tag.yaw)
71              $('#tag .dist').text(data.drone.tag.dist)
72          }
73          else{
74              $('#tag .detect').text("Não Detectada")
75              $('#tag .x').text("")
76              $('#tag .y').text("")
```

```
77                $('#tag .yaw').text("")
78                $('#tag .dist').text("")
79            }
80
81
82        })
83        socket.on('desconect', function () {
84            $('#drone .bateria').text("desconectado")
85        })
86
87        socket.on('voando', function (data) {
88          voando = data.status
89        })
90
91        socket.on('menssagem', function (data) {
92          $('#menssagem .str').text(data.str)
93          console.log(data.str)
94      })
95  });
96
97
98
99  var takeoff = document.getElementById('takeoff');
100 var land = document.getElementById('land');
101 var sos = document.getElementById('stop');
102 var calibragem = document.getElementById('calibragem');
103 var IM = document.getElementById('missao');
104 var IM2 = document.getElementById('frente');
105 //var IM3 = document.getElementById('missao3');
106 var statusVoando = document.getElementById('statusVoando');
107 var foto = document.getElementById('photo');
108 var TC = document.getElementById('TC');
109 var dados = document.getElementById('dados');
110
111
112 takeoff.onclick = function(){
113     console.log('takeoff')
114     socket.emit('takeoff')
115 }
116 land.onclick = function() {
117     console.log('land')
118     socket.emit('land')
119 };
120
121 sos.onclick = function() {
122     console.log('stop')
123     socket.emit('stop')
124 };
125
126 calibragem.onclick = function() {
127     console.log('calibragem')
128     socket.emit('calibragem')
```

```
};

IM.onclick = function() {
    console.log('missao')
    socket.emit('missao')
};

IM2.onclick = function() {
  console.log('frente')
  socket.emit('frente')
};
/*
IM3.onclick = function() {
  console.log('missao3')
  socket.emit('missao3')
};
*/
statusVoando.onclick = function() {
  console.log('statusVoando')
  voando = !voando
};

foto.onclick = function() {
  console.log('foto')
  socket.emit('foto')
};

TC.onclick = function() {
  console.log('trocar Camera')
  socket.emit('trocarCamera')
};
dados.onclick = async function() {
  console.log("grafico")
  //const image = await GoogleChartsNode.render(drawChart, {
    //width: 400,
    //height: 300,
  //});
  //google.charts.setOnLoadCallback(drawChart);
  chart1.update();
  chart2.update();
  chart3.update();
};


const ctx1 = document.getElementById('chart1');
const ctx2 = document.getElementById('chart2');
const ctx3 = document.getElementById('chart3');
const ctx4 = document.getElementById('chart4');
//pisicao x e y
var chart1 = new Chart(ctx1,{
    type: "line",
    data: {
```

```
        labels: drone.vetSN,
        datasets: [{
          label: 'POSICAO X',
          data: drone.vetx,
          borderWidth: 1,
          fill: false,
          borderColor:"red"
        },
        {
          label: 'POSICAO Y',
          data: drone.vety,
          borderWidth: 1,
          fill: false,
          borderColor:"blue"
      },]
      },
      options: {
        scales: {
          y: {
            beginAtZero: true
          },
          x:{
            beginAtZero: true
          }
        }
      }
})



//pisicao x por t
var chart2 = new Chart(ctx2,{
    type: "line",
    data: {
        labels: drone.vetSN,
        datasets: [{
          label: 'traslacao x',
          data: drone.vetTrans.x,
          borderWidth: 1,
          fill: false,
          borderColor:"red"
        },
        {
          label: 'traslacao y',
          data: drone.vetTrans.y,
          borderWidth: 1,
          fill: false,
          borderColor:"blue"
        }]
      },
      options: {
        scales: {
```

```
            y: {
              beginAtZero: true
            },
            x:{
              beginAtZero: true
            }
          }
        }
})


var chart3 = new Chart(ctx3,{
  type: "line",
  data: {
      labels: drone.vetSN,
      datasets: [{
        label: 'Aceleracao x',
        data: drone.vetAccelerometers.x,
        borderWidth: 1,
        fill: false,
        borderColor:"red"
      },
      {
        label: 'Aceleracao y',
        data: drone.vetAccelerometers.y,
        borderWidth: 1,
        fill: false,
        borderColor:"blue"
      }]
    },
    options: {
      scales: {
        y: {
          beginAtZero: true
        },
        x:{
          beginAtZero: true
        }
      }
    }
})

var chart4 = new Chart(ctx4,{
  type: "line",
  data: {
      labels: drone.vetSN,
      datasets: [{
        label: 'angulo yam',
        data: drone.vetyaw,
        borderWidth: 1,
        fill: false
      }]
```

```
    },
    options: {
      scales: {
        y: {
          beginAtZero: true
        }
      }
    }
})

/*
google.charts.load('current', {packages: ['corechart', 'line']});


function drawChart() {
  var data = new google.visualization.DataTable();
  data.addColumn('number', 'X');
  data.addColumn('number', 'Posicao X');
  data.addColumn('number', 'Posicao Y');

  var vet = [],
  aux = [];

  for(i=0;i>drone.vetSN.length;i++){
    aux = [i,drone.vetx[i],drone.vety[i]]
    vet.push(aux)
  }
  data.addRows(vet)
  var options = {
    vAxis: {
      title: 'metros'
    },
    colors: ['#AB0D06', '#007329'],
  };

  var chart = new google.visualization.LineChart(ctx1);
  chart.draw(data, options);
  }
*/
```

Código A.10 – pagina1.ejs

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width,
     initial-scale=1.0">
  <title>AR.Drone2.0</title>
  <link rel='stylesheet' href='/stylesheets/style.css' />

```

```
</head>
<body>
  <h3>Teste do GPS</h3>
  <div id="droneStream" style="width: 640px; height: 360px">
     </div>

  <div id="drone">
    <br />
    Bateria do drone:
    <span class='bateria'></span>
    <br />
    Altitude:
    <!--span class='altitude'></span>m
    <br />
    Rotacao:
    <br />
    eixo longitud.(x):
    <span class='rotX'></span>
    <br />
    eixo lateral(y)  :
    <span class='rotY'></span>
    <br />
    eixo vertical(z) :
    <span class='rotZ'></span>
    <br />
    Traslacao:
    <br />
    x:
    <span class='x'></span>m
    <br />
    y:
    <span class='y'></span>m
    <br />
    Z:
    <span class='z'></span>m
    <br />
    <hr width = 100% size = 7>

  </div-->
  <div id="state">
    Status:
    <br />
    X:
    <span class='x'></span>m
    <br />
    Y:
    <span class='y'></span>m
    <br />
    Z:
    <span class='z'></span>m
    <br />
    yaw:
```

```
61      <span class='yaw'></span> graus /
62      pitch:
63      <span class='pitch'></span> graus /
64      roll:
65      <span class='roll'></span> graus /
66      <br />
67      Vx:
68      <span class='vx'></span> m/s
69      <br />
70      Ax:
71      <span class='ax'></span> m/s2
72      <br />
73      Vy:
74      <span class='vy'></span>m/s
75      <br />
76      Ay:
77      <span class='ay'></span> m/ss2
78      <br />
79    </div>
80    <div id="tag">
81      Tag
82      <span class="detect"></span>
83      <br />
84      x:
85      <span class="x"></span>
86      <br />
87      y:
88      <span class="y"></span>
89      <br />
90      yaw:
91      <span class="yaw"></span>
92      <br />
93      distancai:
94      <span class="dist"></span>
95      <br />
96    </div>
97
98
99
100
101   <hr width = 100% size = 7>
102   <div id="menssagem">
103     Menssagem:
104     <span class='str'></span>
105   </div>
106   <button id='takeoff'>Decolar</button>
107   <button id='land'>Pousar</button>
108   <button id='stop'>Stop</button><br />
109   <button id='calibragem'>calibragem FTRIM</button>
110   <br />
111   <button id='statusVoando'>trocar status voando</button-->
112   <br />
```

```
113   <button id='missao'>Iniciar missao </button>
114   <button id='frente'>frente</button>
115   <!--button id='missao3'>Iniciar missao 3</button-->
116   <button id='photo'>take photo</button>
117   <button id='TC'>trocar camera</button>
118
119
120   <div>
121     <canvas id="chart1"></canvas>
122   </div>
123   <div>
124     <canvas id="chart2"></canvas>
125   </div>
126   <div>
127     <canvas id="chart3"></canvas>
128   </div>
129   <div>
130     <canvas id="chart4"></canvas>
131   </div>
132   <br />
133   <button id='dados'>obter dados</button>
134
135
136
137
138   <!--script type="text/javascript"
        src="https://www.gstatic.com/charts/loader.js"></script-->
139
140   <!--script src="/google-charts/dist/googleCharts.js"></script-->
141   <script src="/chart/chart.umd.js"></script>
142   <script src="/socket.io/socket.io.js"></script>
143   <script src="/dronestream/nodecopter-client.js"></script>
144   <script src="/jquery/jquery.min.js"></script>
145   <script src="/javascripts/index.js"></script>
146 </body>
147 </html>
```

Código A.11 – package.json

```
1  {
2    "name": "testdrone",
3    "version": "0.0.0",
4    "private": true,
5    "scripts": {
6      "start": "nodemon ./bin/www"
7    },
8    "dependencies": {
9      "ar-drone": "^0.3.3",
10     "ardrone-autonomy": "^0.1.2",
11     "chart.js": "^4.1.2",
12     "cookie-parser": "~1.4.4",
13     "debug": "~2.6.9",
```

```
    "dronestream": "^1.1.1",
    "ejs": "~2.6.1",
    "express": "~4.16.1",
    "ffmpeg": "^0.0.4",
    "google-charts-node": "^1.1.0",
    "http-errors": "~1.6.3",
    "jquery": "^3.6.1",
    "morgan": "~1.9.1",
    "nodecopter-stream": "github:bkw/nodecopter-stream",
    "nodemon": "^2.0.20",
    "socket.io": "^4.5.3"
  },
  "devDependencies": {
    "google-charts": "^2.0.0"
  }
}
```

Código A.12 – server.js

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('testdrone:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);



/**
 * Listen on provided port, on all network interfaces.
 */

 const { Server } = require("socket.io");
 const io = new Server(server);

 var dronestream = require("dronestream/index");
```

```
34 dronestream.listen(server);
35
36 server.listen(port);
37 server.on('error', onError);
38 server.on('listening', onListening);
39
40 var drone = {
41   conect: false,
42   bateria: 0,
43   tag: {d:0},
44   translation: { x:0,y:0},
45   accelerometers: {x: 0,y: 0,z: 0},
46   gyroscopes: {x: 0,y: 0,z: 0}
47 }
48
49
50 var autonomy = require("./index.js");
51 //var autonomy = require("ardrone-autonomy")
52 var mission = autonomy.createMission();
53
54 configuraDrone()
55 mission.client().on('navdata', lerNavData);
56
57
58 io.sockets.on('connection', function (socket) {
59   console.log('connection');
60
61   socket.on('foto', async function (data) {
62     await takePhoto().then(() => {
63       console.log("Picture taken")
64     });
65   });
66   socket.on('takeoff',function(){
67
68     io.emit('voando', {status: true})
69     mission.client().takeoff()
70   })
71
72   socket.on('land', function () {
73     console.log('land')
74     mission.client().land()
75     io.emit('voando', {status: false})
76   });
77
78   socket.on('stop', function () {
79     console.log('stop')
80     //mission.client().stop()
81     mission.control().disable();
82     mission.client().land();
83     io.emit('voando', {status: false})
84   });
85
```

```
86  socket.on('calibragem', function () {
87    console.log('calibragem FTRIM')
88    mission.client().calibrate('1');
89  })
90
91
92  socket.on('trocarCamera', function () {
93    console.log('trocar Camera')
94    if(camera==1){
95      camera = 0
96      mission.client().config('video:video_channel', camera);
97      console.log("Camera de baixo")
98    }
99    else{if(camera==0){
100     camera = 1
101     mission.client().config('video:video_channel', camera);
102     console.log("Camera frontal")
103   }}
104 })
105
106 socket.on('missao', function () {
107   // Plan mission
108   retangulo()
109   mission.run(function (err, result) {
110     if (err) {
111       io.emit('menssagem', {str: "Oops, something bad happened:
            %s" + err.message})
112       mission.client().stop();
113       mission.client().land();
114       io.emit('voando', {status: false})
115     } else {
116       io.emit('menssagem', {str: "fim da missao"})
117     }
118   });
119   io.emit('voando', {status: false})
120 })
121
122 socket.on('frente', function () {
123   // Plan mission
124   frente()
125   mission.run(function (err, result) {
126     if (err) {
127       io.emit('menssagem', {str: "Oops, something bad happened:
            %s" + err.message})
128       mission.client().stop();
129       mission.client().land();
130       io.emit('voando', {status: false})
131     } else {
132       io.emit('menssagem', {str: "pousando"})
133       //mission.client().stop();
134       mission.client().land();
135       io.emit('voando', {status: false})
```

```
136         }
137       });
138     })
139
140
141 });
142
143
144
145
146
147
148
149
150
151 // ------------------ DRONE
        ----------------------------------------------------
152
153 var camera = 0
154 function configuraDrone(){
155   const arDroneConstants = require('ar-drone/lib/constants')
156   mission.client().config('general:navdata_demo', true);
157
158   function navdata_option_mask(c) {
159     return 1 << c;
160   }
161
162   // Da SDK.
163   let navdata_options = (
164     navdata_option_mask(arDroneConstants.options.DEMO)
165     | navdata_option_mask(arDroneConstants.options.VISION_DETECT)
166     | navdata_option_mask(arDroneConstants.options.MAGNETO)
167     | navdata_option_mask(arDroneConstants.options.PHYS_MEASURES)
168   );
169
170   // Conecta e configura o drone
171
172
173   mission.client().config('general:navdata_options',
        navdata_options);
174   mission.client().config('video:video_channel', camera);
175   mission.client().config('detect:detect_type', 12);
176   mission.client().config("control:flight_without_shell", false);
177   mission.client().config("control:outdoor", false);
178
179   //mission.client().calibrate('1');
180 }
181
182 const fs = require("fs");
183 let takePhoto = async function (data) {
184
185   setTimeout(function () {
```

```
186     mission.client().getPngStream().once('data', function (data) {
187       var fileName = 'public/images/tcc.png';
188       // Save the picture
189       fs.writeFile(fileName, data, function (err) {
190         if (err) {
191           return console.log(err)
192         } else {
193           return console.log("Saved!")
194         }
195       });
196     });
197   }, 4000);
198 };
199
200
201
202  var flagAc = 0
203  var ofsetAc = {x: 0,y:0}
204 async function lerNavData(navdata) {
205
206   if(navdata != null && navdata.demo != null) {
207     conectDrone = true;
208     drone.sequenceNumber = navdata.sequenceNumber;
209     drone.bateria = navdata.demo.batteryPercentage;
210     drone.altitude = navdata.demo.altitudeMeters;
211     drone.translation.x = -navdata.demo.drone.camera.translation.x;
212     drone.translation.y = -navdata.demo.drone.camera.translation.y;
213
214
215     drone.mag = navdata.magneto
216     drone.state = mission._control._state
217     drone.tag.d = 0
218     if (navdata.visionDetect && navdata.visionDetect.nbDetected >
          0) {
219
220
221       drone.tag.d = navdata.visionDetect.nbDetected
222       drone.tag.x = navdata.visionDetect.xc[0]
223       drone.tag.y = navdata.visionDetect.yc[0]
224       drone.tag.largura = navdata.visionDetect.width[0]
225       drone.tag.altura = navdata.visionDetect.height[0]
226       drone.tag.yaw = navdata.visionDetect.orientationAngle[0]
227       drone.tag.dist = navdata.visionDetect.dist[0]
228
229
230
231       let aulx = p2m(drone.tag.x  + drone.tag.largura/2,
             drone.tag.y  + drone.tag.altura/2, drone.tag.dist);
232
233       // We convert this to the controller coordinate system
234       drone.tag.x =   -1 * aulx.y
235       drone.tag.y = aulx.x
```

```
236     }
237
238   }
239
240   io.emit('droneData', { drone })
241
242 }
243 p2m = function(x, y, altitude) {
244
245   // AR Drone 2.0 Bottom Camera Intrinsic Matrix
246   //
      https://github.com/tum-vision/ardrone_autonomy/blob/master/calibrations
247   let K_BOTTOM = $M([ [686.994766, 0, 329.323208],
248                       [0, 688.195055, 159.323007],
249                       [0, 0, 1]]);
250   let invK = K_BOTTOM.inverse();
251   // From the SDK Documentation:
252   // X and Y coordinates of detected tag or oriented roundel #i
      inside the picture,
253   // with (0; 0) being the top-left corner, and (1000; 1000) the
      right-bottom corner regardless
254   // the picture resolution or the source camera.
255   //
256   // But our camera intrinsic is built for 640 x 360 pixel grid,
      so we must do some mapping.
257   let xratio = 640 / 1000;
258   let yratio = 360 / 1000;
259
260   // Perform a simple back projection, we assume the drone is flat
      (no roll/pitch)
261   // for the moment. We ignore the drone translation and yaw since
      we want X,Y in the
262   // drone coordinate system.
263   let p = $V([x * xratio, y * yratio, 1]);
264   let P = invK.multiply(p).multiply(altitude);
265
266   // X,Y are expressed in meters, in the drone coordinate system.
267   // Which is:
268   //           <--- front-facing camera
269   //         |
270   //        / \------- X
271   //        \_/
272   //         |
273   //         |
274   //         Y
275   return {x: P.e(1), y: P.e(2)};
276 }
277 //-----------------------------------------------------------------------------
278 /**
279  * Normalize a port into a number, string, or false.
280  */
281
```

```
function normalizePort(val) {
  var port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

/**
 * Event listener for HTTP server "error" event.
 */

function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
  }

  var bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port;

  // handle specific listen errors with friendly messages
  switch (error.code) {
    case 'EACCES':
      console.error(bind + ' requires elevated privileges');
      process.exit(1);
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
}

/**
 * Event listener for HTTP server "listening" event.
 */

function onListening() {
  var addr = server.address();
  var bind = typeof addr === 'string'
    ? 'pipe ' + addr
```

```
334       : 'port ' + addr.port;
335   debug('Listening on ' + bind);
336 }
337
338 function retangulo() {
339   io.emit('menssagem', {str: "retangulo 1"})
340   mission.zero()
341       .taskSync(function (){
342         io.emit('voando', {status: true})
343       })
344       .takeoff()
345       .taskSync(function (){
346         io.emit('menssagem', {str: "drone no ar "})
347         //mission.client().calibrate('0')
348       })
349       .altitude(3)
350       .go({x:0, y:0,yaw:0})
351       .hover(3000)
352       .taskSync(function (){
353         io.emit('menssagem', {str: "iniciando inspecao "})
354       })
355       .go({x:0, y:2})//.right(2)//.go({x:2, y:2})
356       .hover(2000)
357       .yaw(-90)
358       .hover(500)
359       .altitude(2)
360       .hover(2000)
361       .taskSync(function (){
362         io.emit('menssagem', {str: "posicao x:0 y:2 "})
363       })
364       .hover(1000)
365       .go({x:-2, y:2})
366       .hover(500)
367       .yaw(-45)
368       .hover(2000)
369       .taskSync(function (){
370         io.emit('menssagem', {str: "posicao x:-2 y:2 "})
371       })
372       .hover(1000)
373       .yaw(0)
374       .hover(500)
375       .go({x:-2, y:0})
376       .hover(2000)
377       .taskSync(function (){
378         io.emit('menssagem', {str: "posicao x:-2 y:0 "})
379       })
380       .hover(1000)
381       .go({x:-2, y:-2})
382       .hover(500)
383       .yaw(45)
384       .hover(2000)
385       .taskSync(function (){
```

```
386           io.emit('menssagem', {str: "posicao x:-2 y:-2 "})
387         })
388         .hover(1000)
389         .yaw(90)
390         .go({x:0, y:-2})
391         .hover(2000)
392         .taskSync(function (){
393           io.emit('menssagem', {str: "posicao x:0 y:-2 "})
394         })
395         .hover(1000)
396         .altitude(3)
397         .go({x:0, y:0})
398         .hover(500)
399         .yaw(0)
400         .hover(2000)
401         .taskSync(function (){
402           io.emit('menssagem', {str: "fim da inspecao"})
403         })
404         .hover(1000)
405         .land()
406 }
407 function retangulo2() {
408   io.emit('menssagem', {str: "retangulo 2"})
409   mission.zero()
410         .taskSync(function (){
411           io.emit('voando', {status: true})
412         })
413         .takeoff()
414         .taskSync(function (){
415           io.emit('menssagem', {str: "drone no ar "})
416           mission.client().calibrate('0')
417         })
418         .altitude(3)
419         .go({x:0, y:0,yaw:0})
420         .hover(3000)
421         .taskSync(function (){
422           io.emit('menssagem', {str: "iniciando inspecao "})
423         })
424         .right(2)//.go({x:2, y:2})
425         .hover(2000)
426         .yaw(-90)
427         .hover(500)
428         .altitude(2)
429         .hover(2000)
430         .taskSync(function (){
431           io.emit('menssagem', {str: "posicao x:0 y:2 "})
432         })
433         .hover(1000)
434         .left(2)
435         .hover(500)
436         .yaw(-45)
437         .hover(2000)
```

```
438         .taskSync(function (){
439           io.emit('menssagem', {str: "posicao x:-2 y:2 "})
440         })
441         .hover(1000)
442         .yaw(0)
443         .hover(500)
444         .left(2)
445         .hover(2000)
446         .taskSync(function (){
447           io.emit('menssagem', {str: "posicao x:-2 y:0 "})
448         })
449         .hover(1000)
450         .left(2)
451         .hover(500)
452         .yaw(45)
453         .hover(2000)
454         .taskSync(function (){
455           io.emit('menssagem', {str: "posicao x:-2 y:-2 "})
456         })
457         .hover(1000)
458         .yaw(90)
459         .left(2)
460         .hover(2000)
461         .taskSync(function (){
462           io.emit('menssagem', {str: "posicao x:0 y:-2 "})
463         })
464         .hover(1000)
465         .altitude(3)
466         .go({x:0, y:0})
467         .hover(500)
468         .yaw(0)
469         .hover(2000)
470         .taskSync(function (){
471           io.emit('menssagem', {str: "fim da inspecao"})
472         })
473         .hover(1000)
474         .land()
475 }
476 function frente(){
477   io.emit('menssagem', {str: "frnte 1"})
478   mission.zero()
479         .taskSync(function (){
480           io.emit('voando', {status: true})
481         })
482         .takeoff()
483         .taskSync(function (){
484           io.emit('menssagem', {str: "drone no ar "})
485         })
486         .altitude(2)
487         .taskSync(function (){
488           io.emit('menssagem', {str: "drone indo para [0,0] "})
489         })
```

```
490         .go({x:0, y:0})
491         .hover(3000)
492         .taskSync(function (){
493           io.emit('menssagem', {str: "drone na posicao [0,0] "})
494         })
495
496         .taskSync(function (){
497           io.emit('menssagem', {str: "drone indo para 2m a frente"})
498         })
499         .go({x:2, y:0})//.forward(2)
500         .hover(3000)
501         .taskSync(function (){
502           io.emit('menssagem', {str: "drone na posicao [2,0] "})
503         })
504 }
```

A.4 Teste do GPS

A.4.1 Codigos dos microcontroladores

Código A.13 – EmissorGPS.ino

```
1                                                   #include <SPI.h>
2  #include "printf.h"
3  #include "RF24.h"
4
5  #include <TinyGPSPlus.h>
6  #include <TinyGPS++.h>
7
8
9
10 #define RXD2 12
11 #define TXD2 14
12 HardwareSerial neogps(1);
13
14 TinyGPSPlus gps;
15 bool flag;
16 unsigned long tempo,t,t0,t1;
17
18 RF24 radio(4, 5);  // using pin 7 for the CE pin, and pin 8 for
       the CSN pin
19 uint8_t address[6] = "1Node";
20 #define SIZE 32
21 String str;
22 char buffer[SIZE + 1];
23
24 void getGPS();
25 void trasmitir();
26
27 void setup() {
```

```
28    buffer[SIZE] = 0;  // add a NULL terminating character (for easy
          printing)
29
30    Serial.begin(115200);
31
32    neogps.begin(9600, SERIAL_8N1, RXD2, TXD2);
33    flag = true;
34    Serial.println("INICIANDO");
35
36    // initialize the transceiver on the SPI bus
37    while (!radio.begin()) {
38      Serial.println(F("radio hardware is not responding!!"));
39      delay(1000);
40    }
41    Serial.println(F("Iniciando recepitor"));
42
43    radio.setPALevel(RF24_PA_MAX);  // RF24_PA_MAX is default.
44    radio.setPayloadSize(SIZE);  // default value is the maximum 32
          bytes
45
46     // set the TX address of the RX node into the TX pipe
47    radio.openWritingPipe(address);  // always uses pipe 0
48
49    radio.stopListening();  // put radio in TX mode
50
51
52
53    t0 = millis();
54    t1 = millis();
55    tempo = millis();
56  }
57
58  void loop() {
59    while (neogps.available())
60    {
61      if (gps.encode(neogps.read()))
62      {
63        if(gps.satellites.value() > 0){
64          getGPS();
65
66        }
67        else{
68          if(millis()-t0 > 1000){
69            t0 = millis();
70            str = "buscando satelites...";
71            trasmitir();
72          }
73        }
74        t1 = millis();
75      }
76    }
77
```

```
78    if (millis() - t1  > 5000)
79    {
80      str= "Nenhum GPS detectado: verifique a fiação.";
81      trasmitir();
82      while(true);
83    }
84  }
85
86  void getGPS(){
87    if (gps.location.isUpdated())
88    {
89     //String gps_speed = String(gps.speed.kmph());
90      if(flag){
91        t = millis() - tempo;
92        //Serial.println(t);
93      }
94      flag = false;
95      str = (String(gps.satellites.value())+";"
96      +String(gps.location.lat(),7)+";"+
97      String(gps.location.lng(),7)+";"+
98      String(t/1000));
99      trasmitir();
100   }
101   else
102   {
103     str = "dados do GPS invalidos";
104   }
105   //trasmitir();
106 }
107
108 void trasmitir(){
109     radio.flush_tx();
110     str.toCharArray(buffer, SIZE);
111     unsigned long start_timer = micros();  // start the timer
112     if (!radio.writeFast(&buffer, SIZE)) {
113         Serial.println("Erro ao enviar!");
114         radio.reUseTX();
115
116       } else {
117         Serial.println(buffer);
118       }
119
120 }
```

Código A.14 – Recepitor.ino

```
1  #include <SPI.h>
2  #include "printf.h"
3  #include "RF24.h"
4
5  RF24 radio(7, 8);  // using pin 7 for the CE pin, and pin 8 for
       the CSN pin
```

```
6  uint8_t address[6] = "1Node";
7  #define SIZE 32             // this is the maximum for this
       example. (minimum is 1)
8  char buffer[SIZE + 1];      // for the RX node
9  uint8_t counter = 0;        // for counting the number of received
       payloads
10
11
12 void setup() {
13
14   buffer[SIZE] = 0;  // add a NULL terminating character (for easy
       printing)
15
16   Serial.begin(115200);
17
18   // initialize the transceiver on the SPI bus
19   while (!radio.begin()) {
20     Serial.println(F("hardware de rádio não está respondendo!!"));
21     delay(1000);
22   }
23
24   Serial.println(F("Iniciando recepitor"));
25
26   radio.setPALevel(RF24_PA_MAX);  // RF24_PA_MAX is default.
27   radio.setPayloadSize(SIZE);  // default value is the maximum 32
       bytes
28
29   // set the RX address of the TX node into a RX pipe
30   radio.openReadingPipe(1, address);  // using pipe 1
31
32   radio.startListening();  // put radio in RX mode
33
34 }
35
36 void loop() {
37   // This device is a RX node
38
39     if (radio.available()) {      // is there a payload?
40       radio.read(&buffer, SIZE);  // fetch payload from FIFO
41
42       Serial.println(buffer);  // print the payload's value
43       //Serial.print(F(" - "));
44       //Serial.println(counter++);  // print the received counter
45     }
46
47
48 }
```

A.4.2 Codigos na aplição do teste de tragetoria

Código A.15 – app.js

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/jquery',express.static(path.join(__dirname,
    'node_modules/jquery/dist')));
app.use('/leaflet',express.static(path.join(__dirname,
    'node_modules/leaflet/dist')));
app.use('/googlemutant',express.static(path.join(__dirname,
    'node_modules/leaflet.gridlayer.googlemutant/dist')));

app.use('/', indexRouter);
app.use('/users', usersRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err :
     {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

Código A.16 – pagina1.js

```
1
2  var baseIcon = L.icon({
3      iconUrl: '../images/helioporto.png',
4      iconSize:     [50, 50],
5      iconAnchor:   [25, 25]
6  });
7
8  var map,
9      escala,
10     rota;
11
12 var base = undefined,
13     baseMarkers
14 var waypointMarkers = [];
15 var torres = [];
16 var torre = {lat:undefined,lng:undefined,tipo:undefined};
17 var waypoints = [];
18
19
20
21 var inicio = document.getElementById("inicio");
22 inicio.showModal()
23
24 var setBase = document.getElementById("setBase");
25 var setBaseMap = document.getElementById("setBaseMap");
26
27
28 setBase.onclick = function (){
29     base = [document.getElementById("latBase").value,
30             document.getElementById("lngBase").value]
31     escala = 20
32     initMap1(base[0],base[1])
33     waypoints.push(base)
34     baseMarkers = L.marker(base, { icon: baseIcon }).addTo(map)
35     inicio.close()
36 }
37
38 setBaseMap.onclick = function (){
39     escala = 18
40     navigator.geolocation.getCurrentPosition(
41         initMap0, defaultMap,{ enableHighAccuracy: true })
42     inicio.close()
43
44 }
45 function initMap0(position) {
46     map = L.map('map').setView([position.coords.latitude,
47         position.coords.longitude],escala);
48
49     var googleHybrid =
       L.tileLayer('http://{s}.google.com/vt/lyrs=s,h&x={x}&y={y}&z={z}',{
```

```
50      maxZoom: 22,
51      subdomains:['mt0','mt1','mt2','mt3']
52      });
53      map.addLayer(googleHybrid);
54
55
56      scala = L.control.scale().addTo(map)
57
58      //laptop = L.marker([Lat,Lng], { icon: laptopIcon }).addTo(map)
59
60      map.on('click', function (e) {
61          if(base == undefined){
62              baseMarkers = L.marker(e.latlng, { icon: baseIcon
                    }).addTo(map)
63              base = [e.latlng.lat,e.latlng.lng]
64              waypoints.push(base)
65          }
66          else{torre.lat = e.latlng.lat
67              torre.lng = e.latlng.lng
68              waypointMarkers.push(L.marker(e.latlng).addTo(map))
69              popup.showModal()
70
71              $('#waypoint .lat').text(torre.lat)
72              $('#waypoint .lon').text(torre.lng)
73              $('#waypoint .tipo').text(" ")
74          }
75      });
76  }
77  function initMap1(Lat,Lng) {
78      map = L.map('map').setView([Lat,Lng],escala);
79
80      var googleHybrid =
            L.tileLayer('http://{s}.google.com/vt/lyrs=s,h&x={x}&y={y}&z={z}',{
81      maxZoom: 22,
82      subdomains:['mt0','mt1','mt2','mt3']
83      });
84      map.addLayer(googleHybrid);
85
86
87      scala = L.control.scale().addTo(map)
88
89      //laptop = L.marker([Lat,Lng], { icon: laptopIcon }).addTo(map)
90
91      map.on('click', function (e) {
92          if(base == undefined){
93              baseMarkers = L.marker(e.latlng, { icon: baseIcon
                    }).addTo(map)
94              base = [e.latlng.lat,e.latlng.lng]
95              waypoints.push(base)
96          }
97          else{torre.lat = e.latlng.lat
98              torre.lng = e.latlng.lng
```

```
99              waypointMarkers.push(L.marker(e.latlng).addTo(map))
100             popup.showModal()
101
102             $('#waypoint .lat').text(torre.lat)
103             $('#waypoint .lon').text(torre.lng)
104             $('#waypoint .tipo').text(" ")
105         }
106     });
107 }
108 function defaultMap(err){
109     console.log("falha ao iniciar o mapa" + err)
110     escala = 12
111     initMap1(-15.844446926217513,-48.02418529987335)
112 }
113
114 function clearWaypoints() {
115     waypoints = []
116     waypoints.push(base)
117     map.removeLayer(rota)
118     rota = undefined
119     $.each(waypointMarkers, function (i, m) { map.removeLayer(m) })
120 }
121
122 function setCurrentTarget(lat, lon) {
123     targetLat = lat
124     targetLon = lon
125     socket.emit('prossima torre', { lat: targetLat, lon: targetLon
        })
126 }
127
128 var clear = document.getElementById('limpar');
129 var concluir = document.getElementById('concluir');
130
131 var popup = document.getElementById("popup");
132 var tipoA = document.getElementById("tipoA");
133 var tipoB = document.getElementById("tipoB");
134 var cancelar = document.getElementById("cancelar");
135
136 clear.onclick = function (){
137     console.log("limpar")
138     clearWaypoints()
139
140 }
141
142 concluir.onclick = function (){
143     console.log("rota concluida")
144     socket.emit("RotaConcluida",{torres:torres, base: base})
145     window.location.href="p2"
146 }
147
148 tipoA.onclick = function (){
149     console.log("Tipo A")
```

```
150     torre.tipo = 'A'
151     torres.push([torre.lat,torre.lng,torre.tipo])
152     waypoints.push([torre.lat,torre.lng])
153     if (   rota == undefined) {
154         rota = L.polyline(waypoints, { color: 'blue' }).addTo(map);
155     } else {
156         rota.setLatLngs(waypoints)
157     }
158     popup.close()
159     $('#waypoint .tipo').text(torre.tipo)
160 }
161
162 tipoB.onclick = function (){
163     console.log("Tipo B")
164     torre.tipo = 'B'
165     torres.push([torre.lat,torre.lng,torre.tipo])
166     waypoints.push([torre.lat,torre.lng])
167     if (   rota == undefined) {
168         rota = L.polyline(waypoints, { color: 'blue' }).addTo(map);
169     } else {
170         rota.setLatLngs(waypoints)
171     }
172     popup.close()
173     $('#waypoint .tipo').text(torre.tipo)
174 }
175
176 cancelar.onclick = function (){
177     console.log("cancelar")
178     popup.close()
179     map.removeLayer(waypointMarkers[waypointMarkers.length - 1])
180     $('#waypoint .lat').text(torres[torres.length-1][0])
181     $('#waypoint .lon').text(torres[torres.length-1][1])
182     $('#waypoint .tipo').text(torres[torres.length-1][2])
183 }
184
185 var socket = io.connect('/');
186
187 socket.on('connect', function () {
188
189 })
```

Código A.17 – pagina1.ejs

```
1 <!DOCTYPE html>
2 <html lang="pt-br">
3   <head>
4     <title><%= title %></title>
5     <meta charset="UTF-8">
6     <meta http-equiv="X-UA-Compatible" content="IE=edge">
7     <meta name="viewport" content="width=device-width,
        initial-scale=1.0">
8
```

```
    <link rel="stylesheet" href="/leaflet/leaflet.css"/>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <dialog id = inicio>

      <h2>Defina a pisição de decolagem</h2><br />
      <p>Ex(lat,lng):-15.844446926217513,-48.02418529987335</p>
      <p>Latitude :<input type="number" id="latBase"  /><br />
         Longitude:<input type="number" id="lngBase"  /></p>
      <button id='setBase'>definir</button><br /><br />
      <button id='setBaseMap'>definirno mapa</button>
    </dialog>
    <div id="map"></div>
    <dialog id = popup>
      <h2>Defina o tipo de inspeção</h2> <br/>
      <button id='tipoA'>Torre tipo A</button>
      <button id='tipoB'>Torre tipo B</button>
      <button id='cancelar'>cancelar</button>
    </dialog>
    <div id="popup"></div>
    <div id="waypoint">
      <br />
      Ultima torre:<br /> Lat: <span class='lat'></span>,
      Lng: <span class='lon'></span>,
      Tipo: <span class='tipo'></span>
      <br />
    </div>
    <button id='limpar'>Limpar rota</button><br />
    <button id='concluir'>concluir rota</button><br />

    <script src="/socket.io/socket.io.js"></script>
    <script src="/leaflet/leaflet.js"></script>
    <script src="/jquery/jquery.min.js"></script>
    <script src="/javascripts/index.js"></script>
  </body>
</html>
```

Código A.18 – package.json

```
{
  "name": "testgps",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "nodemon ./bin/www"
  },
  "dependencies": {
    "ar-drone": "^0.3.3",
    "ardrone-autonomy": "^0.1.2",
    "cookie-parser": "~1.4.4",
```

```
    "debug": "~2.6.9",
    "ejs": "~2.6.1",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "jquery": "^3.6.1",
    "leaflet": "^1.9.2",
    "leaflet.gridlayer.googlemutant": "^0.13.5",
    "morgan": "~1.9.1",
    "nodemon": "^2.0.20",
    "serialport": "^10.4.0",
    "socket.io": "^4.5.3"
  }
}
```

Código A.19 – server.js

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('testgps:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);

const { Server } = require("socket.io");
const io = new Server(server);

/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);

io.sockets.on('connection', function (socket) {
  console.log('connection');
```

```
37
38    setInterval(function () {
39      if(conectDrone){
40        io.sockets.emit('droneData', { drone })
41      } else{
42        io.sockets.emit('desconect')
43      }
44    }, 500)
45  });
46
47  //_______________ ARDUINO _______________________________
48
49  const { SerialPort } = require('serialport')
50  const { ReadlineParser } = require('@serialport/parser-readline')
51  const porta = new SerialPort({ path: "/dev/cu.wchusbserialfd120",
      baudRate: 115200 })
52  //const porta = new SerialPort({ path: "/dev/cu.usbmodemFA130",
      baudRate: 115200 })
53
54  const parser = porta.pipe(new ReadlineParser({ delimiter: '\r\n'
      }))
55
56  parser.on("open", function() {
57    console.log("conectado a porta serial");
58  });
59
60  var conectDrone = false;
61  var drone = new Object;
62  drone.bateria = 0;
63  var SAT, lat, lng;
64  var vetP=[];
65  parser.on('data', function(data) {
66    try {
67      let s = data.split(";");
68      SAT = parseInt(s[0]);
69      lat = parseFloat(s[1].replace(',', '.'));
70      lng = parseFloat(s[2].replace(',', '.'));
71      let tempo = s[3];
72      //console.log(lat +','+lng)
73
74      let a = 10
75      vetP.push([lat,lng])
76      if(vetP.length > a-1){
77        let posMA=[0,0]
78        let posMH=[0,0]
79
80        vetP.forEach(function(p){
81          posMA[0] += p[0]
82          posMA[1] += p[1]
83
84          posMH[0] += 1/p[0]
85          posMH[1] += 1/p[1]
```

```
86
87          })
88          drone.lat = posMA[0]/a
89          drone.lng = posMA[1]/a
90
91          drone.latH = a/posMA[0]
92          drone.lngH = a/posMA[1]
93
94          vetP=[]
95          io.sockets.emit('posi', {sat: SAT, lat: drone.lat, lng:
               drone.lng, t:tempo,
96             lat1: drone.latH, lng1: drone.lngH})
97        }
98      }
99      catch (e) {
100       io.sockets.emit('falhaGPS', {string:data})
101     }
102 })
103
104 //---------------------------------
105
106
107 var autonomy = require('ardrone-autonomy');
108 var mission = autonomy.createMission();
109
110 configuraDrone()
111 mission.client().on('navdata', lerNavData);
112
113 function configuraDrone(){
114   const arDroneConstants = require('ar-drone/lib/constants')
115   mission.client().config('general:navdata_demo', true);
116
117   function navdata_option_mask(c) {
118     return 1 << c;
119   }
120
121   // Da SDK.
122   let navdata_options = (
123     navdata_option_mask(arDroneConstants.options.DEMO)
124     | navdata_option_mask(arDroneConstants.options.VISION_DETECT)
125     | navdata_option_mask(arDroneConstants.options.MAGNETO)
126     | navdata_option_mask(arDroneConstants.options.WIFI)
127   );
128
129   // Conecta e configura o drone
130
131   mission.client().config('general:navdata_options',
        navdata_options);
132   mission.client().config('video:video_channel', 0);
133   mission.client().config('detect:detect_type', 12);
134   mission.client().calibrate('1');
135 }
```

```
async function lerNavData(navdata) {
  if(navdata != null && navdata.demo != null) {
    conectDrone = true;
    drone.bateria = navdata.demo.batteryPercentage;
    drone.altitude = navdata.demo.altitudeMeters;
  }
}









//--------------------------------------------------------------------------
/**
 * Normalize a port into a number, string, or false.
 */

function normalizePort(val) {
  var port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

/**
 * Event listener for HTTP server "error" event.
 */

function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
  }

  var bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port;
```

```
188
189   // handle specific listen errors with friendly messages
190   switch (error.code) {
191     case 'EACCES':
192       console.error(bind + ' requires elevated privileges');
193       process.exit(1);
194       break;
195     case 'EADDRINUSE':
196       console.error(bind + ' is already in use');
197       process.exit(1);
198       break;
199     default:
200       throw error;
201   }
202 }
203
204 /**
205  * Event listener for HTTP server "listening" event.
206  */
207
208 function onListening() {
209   var addr = server.address();
210   var bind = typeof addr === 'string'
211     ? 'pipe ' + addr
212     : 'port ' + addr.port;
213   debug('Listening on ' + bind);
214 }
```