UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ELIAS RICARDO VIEIRA
GUSTAVO HENRIQUE DA SILVA LOPES

APLICAÇÃO PARA AUTOMAÇÃO DE VÔO DE DRONE

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2016

ELIAS RICARDO VIEIRA
GUSTAVO HENRIQUE DA SILVA LOPES

APLICAÇÃO PARA AUTOMAÇÃO DE VÔO DE DRONE

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2016

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

APLICAÇÃO PARA AUTOMAÇÃO DE VÔO DE DRONE

por

ELIAS RICARDO VIEIRA
GUSTAVO HENRIQUE DA SILVA LOPES

Este trabalho de conclusão de curso foi apresentado no dia 21 de junho de 2016, como requisito parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, pela Universidade Tecnológica Federal do Paraná. O acadêmico foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho APROVADO.

Banca examinadora:

Profª. Drª. Beatriz Terezinha Borsoi
Orientadora

Profª. Dr. Kathya Silvia Collazos Linares

Prof. Me. Robison Cris Brito

Prof. Dr. Edilson Pontarolo
Coordenador do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Profª. Me. Soelaine Rodrigues Ascari
Responsável pela Atividade de Trabalho de Conclusão de Curso

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

RESUMO

VIEIRA, Elias Ricardo; LOPES, Gustavo Henrique da Silva. Aplicação para automação de voo de. 2016. 49f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

Veículos Aéreos não Tripulados (VANT) como os Drones, por exemplo, são ótimas ferramentas que podem ser adaptadas para usos especializados. Uma das aplicações dos Drones é se locomover para locais de difícil acesso. O custo dessa locomoção, se comparado a imagens de satélite ou veículos aéreos tripulados (avião, helicóptero), é muito inferior. O valor médio do Drone utilizado na realização desse projeto é, atualmente, em torno de R$1.500,00. A flexibilidade está no sentido que um Drone pode chegar a locais de difícil acesso e perigosos (como de incêndio). A utilização de um software com algoritmo que faça com que o Drone vá de um ponto a outro, torna o deslocamento, se comparada à manipulação de pontos por meio da utilização de controles remotos, mais simples. Isso porque não é necessário ter conhecimento em manipulação de controles remotos e nem mesmo na utilização do Drone, basta selecionar o destino o qual deseja enviar o Drone.

Palavras-chave: Automação de voo de quadricóptero. Drone. VANT.

ABSTRACT

VIEIRA, Elias Ricardo; LOPES, Gustavo Henrique da Silva. Aplicação para automação de vôo de drone. 2015. 49f. Monografia (Trabalho de Conclusão de Curso) - Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

Unmanned Aerial Vehicles (UAV), like Drones, as example, are great tools that can be adapted for specific uses. One of the uses of Drone is to reach places of hard acces on other ways, and get data from sensors on these places, like photos, temperature or humidity, through sensors and photo cameras. The cost to obtain this data, if compared to aerial images from satelites or manned aerial vehicles (planes, helicopter) it is drastically lower, faster and flexible. The flexibility is towards that a Drone can reach places of hard access and dangerous (like caves, submerged, fire places). The use of a software with specific algorithms, turns the displacement and data collect more effective, if compared to the route manipulation using remote controls. In addition to reading and storage of sensor's data, the Drone can also be used as transporting of light cargo. The developed system as result of this essay is not limited do manage only one Drone, neither the presence of a user is required to the be displaced from one place to another. As the system will be web, any person will have access to the platform, since this person have a device and internet connection.

Palavras-chave: Unmanned Arial Vehicles. Quadricopter. Drone.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGEM DE CÓDIGO

LISTA DE SIGLAS

API	*Application Programming Interface*
CSS	*Cascading Style Sheet*
HTML	*HyperText Markup Language*
JSON	*JavaScript Object Notation*
MVR	*Model-View-Route*
NPM	*Node Package Manager*
ORM	*Object-Relational Mapping*
REST	*Representational State Transfer*
SPA	*Single Page Application*
UAV	*Unmanned Arial Vehicles*
VANT	Veiculo Aéreo Não Tripulado

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta a introdução do trabalho que abrange as considerações inicias, os objetivos e a justificativa. O capítulo é finalizado com a apresentação do texto por meio da descrição sumária dos próximos capítulos.

1.1 CONSIDERAÇÕES INICIAIS

Drone, Quadricóptero, Veiculo Aéreo Não Tripulado (VANT) ou Veículo Aéreo Remotamente Pilotado (VARP), são algumas das variações de nomes para uma atividade que está crescendo no Brasil e no mundo. Eles eram originalmente um *hobby* ou passatempo, mas com a evolução e acessibilidade de novas tecnologias, o que era chamado Aeromodelismo, hoje é tratado com seriedade, pois suas aplicações vêm crescendo nas mais diversas aplicações: como auxílio em regate de pessoas, obtenção de imagens aéreas para mapeamento e gerenciamento agrícola, entre muitos outros. O uso de Drones nos diversos setores da indústria pode gerar um impacto positivo de quase US$ 14 bilhões em três anos e até 2025, estima-se a criação de 100 mil novos empregos (RONCOLATO, 2015).

Um Drone, para generalizar as diversas denominações atribuídas aos veículos aéreos não tripulados de pequeno porte, é um equipamento que precisa ser manuseado com cuidado. As suas hélices de carbono ou plástico, por exemplo, que girando a altas rotações pode ocasionar ferimentos sérios em caso de acidente por desprender-se, pelo veículo perder o equilíbrio ou colidir com as pessoas. Assim, esses equipamentos devem ser pilotados com cautela e utilizando aplicativos de controle confiáveis.

Quando devidamente programados e utilizados com conhecimento, a automação torna-os aplicáveis a usos de, por exemplo, como chegar do ponto A ao ponto B de forma com que, o deslocamento seja feito de maneira segura e independente.

Um dos objetivos no aprimoramento dos Drones, seja em termos de hardware ou de software que os controlam, é torná-los mais seguros e eficazes. E, assim, possam ser mais difundidos e utilizados no dia a dia e possam melhorar serviços, como por exemplo: entregas, auxílio no resgate de pessoas, monitoramento de ambientes e locais de difícil acesso, monitoramento de lavouras, filmagens áreas e muitos outros.

Para a realização deste projeto será desenvolvido uma aplicação que auxilie no controle de um Drone, denominado quadricóptero em decorrência de suas quatro hélices controladas por motores, e faze-lo ir de um ponto A até um ponto B, definidos pelo usuário do sistema por meio de uma interface *web*, de forma automatizada.

1.2 OBJETIVOS

A seguir são apresentados os objetivos pretendidos com a realização deste trabalho.

1.2.1 Objetivo Geral

Desenvolver uma aplicação que auxilie na automatização do vôo de um Drone, fazendo com que o mesmo se desloque de maneira autônoma.

1.2.2 Objetivos Específicos

- Implementar um servidor REST para enviar informações a interface *web* e enviar comandos remotamente ao Drone, para que ele possa ir em linha reta aproximadamente no ponto definido.
- A automação será realizada a partir do momento em que o usuário definir a missão. Após isso, o Drone irá decolar e avançar automaticamente.
- Implementar uma interface *web*, utilizando AngularJS, que deverá enviar os dados ao servidor via JSON, como por exemplo: coordenadas dos locais a serem cadastrados, distância a ser percorrida pelo Drone.

1.3 JUSTIFICATIVA

Existe uma grande demanda para o uso de Drones na área profissional como apontado por Roncolato (2015). Isso motivou o interesse em estudar nessa área e desenvolver aplicativos, ainda que simples, mas que possam contribuir para que Drones

sejam utilizados para ações rotineiras, como transporte de objetos, ou fotografias aéreas, sem a necessidade de alguém estar no controle.

Conforme no relato feito por Bastos (2015), os Drones apresentam uma grande versatilidade, por exemplo, em pulverização e vigilância, valendo, assim, o investimento. Essas aplicações podem multiplicar-se à medida que equipamentos e software mais eficientes sejam desenvolvidos. Além das aplicações agrícolas podem ser citadas muitas outras como o auxílio em resgate de pessoas e animais em situações de desastre ou perigo ou a simples entrega de um objeto.

Independentemente da aplicação, a eficiência de um Drone está associada à eficiência do seu hardware e de aplicativos de controles. Assim, algoritmos para prover o deslocamento seguro de um ponto A para um ponto B podem ser utilizados em qualquer aplicação porque o princípio básico desses equipamentos está no deslocamento. O uso de câmeras e outros dispositivos são utilizados para cada atividade específica, mas o deslocamento é a base para qualquer uso. Justificando, assim, o desenvolvimento de pesquisa e testes nessa área. Paw e Balas (2011) ressaltam que melhorias na modelagem, teste e no controle de voo para esses veículos auxiliarão a melhorar a sua confiabilidade e desempenho durante o voo autônomo.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está organizado em capítulos. Este é o primeiro capítulo é apresenta as considerações iniciais com o contexto do sistema a ser desenvolvido, os seus objetivos e a justificativa. O Capítulo 2 apresenta o referencial teórico centrado em comércio internacional. No Capítulo 3 estão as ferramentas e as tecnologias utilizada na modelagem do sistema e que serão utilizadas na implementação subsequente do sistema. No Capítulo 4 é apresentado o resultado da realização do trabalho, ou seja, a implementação do sistema. Por fim estão as considerações finais seguidas pelas referências

2 REFERENCIAL TEÓRICO

O referencial teórico do trabalho se refere aos Veículos Aéreos Não Tripulados e está centrado em Quadricópteros por ser o modelo de Drone utilizado na implementação realizada como resultado deste trabalho.

2.1 VEÍCULOS AÉREOS NÃO TRIPULADOS

Veículos Aéreos Não Tripulados (VANT), do inglês *Unmanned Arial Vehicles* (UAV), comumente conhecidos como Drones são sistemas aéreos que podem ser remotamente controlados com propósitos militares e civis (SOUTHWORTH, 2012). Drones são usualmente equipados com câmeras e podem contar com diversos outros dispositivos, como mísseis, quando vinculados a objetivos militares e bóias quando em aplicações de resgate na água, por exemplo.

Imagens aéreas têm sido usadas com objetivos militares desde longa data. Obter essas imagens têm alto custo e requer planos de voo e pilotos para voar por meio de espaços aéreos hostis (LUGO; ZELL, 2013). Esses autores ressaltam que hoje em dia qualquer cidadão pode enviar Drones e obter imagens e usá-las em aplicativos para juntá-las e criar um mapa referenciado com informações de determinados pontos.

Os VANT têm atraído interesse considerável devido a uma ampla variedade de aplicações (BENINCASA; CAMARGO; OKAMOTO JUNIOR, 2011). Gomes e Aquino (2015) ressaltam que é notável o crescimento do uso de Drones ou VANTs, principalmente no reconhecimento de áreas, na vigilância de construções como barragens, usinas, plantações e regiões como as de conflito, fronteiras e outras que requeiram monitoramento ou vigilância.

Um VANT pode assumir várias configurações em relação à disposição e quantidade dos seus rotores. Uma dessas configurações é o Quadricóptero (SANTOS; MORATA, 2010) ou quadrator. Na configuração Quadricóptero, quatro conjuntos motor/hélice de mesmas dimensões são fixados, cada um, em uma das extremidades de uma estrutura em forma de “X” ou “+”. A Figura 1 apresenta um Quadricóptero com estrutura em forma de ‘+’.

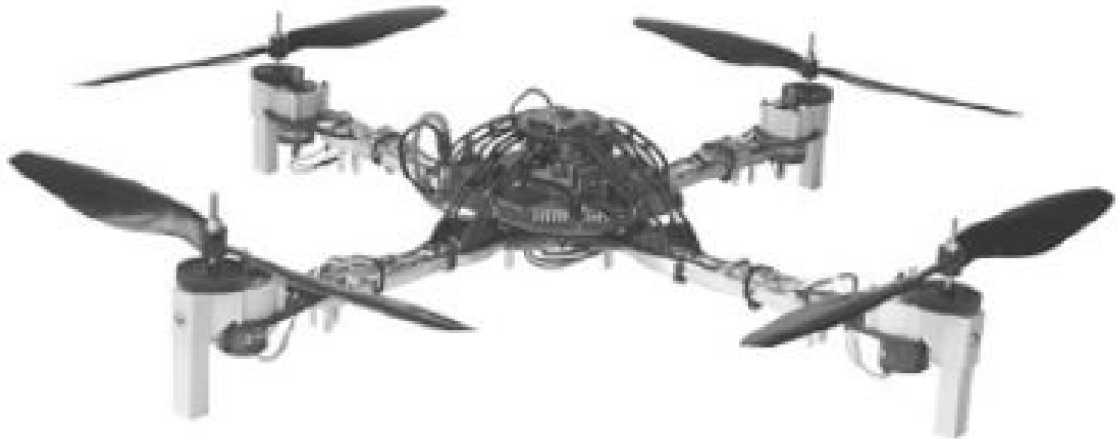
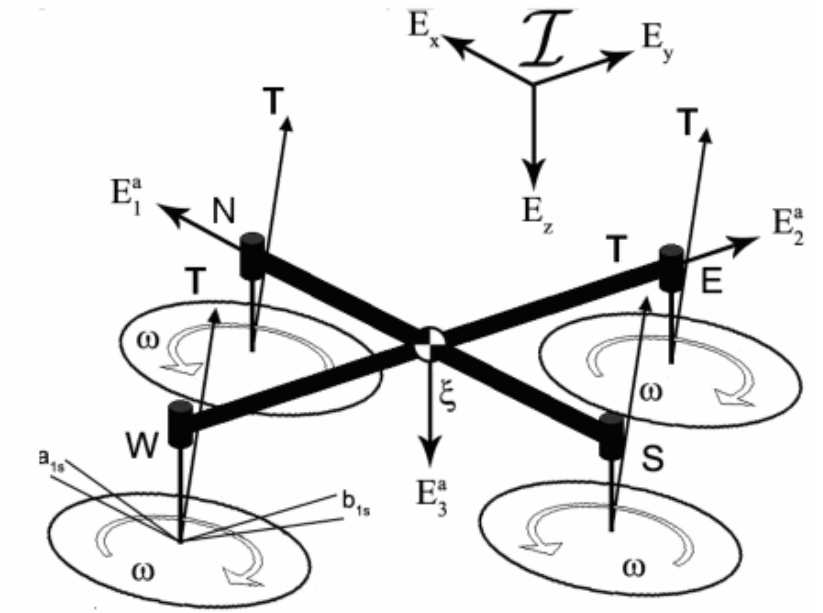


Figura 1 - Protótipo de um quadricoptero
Fonte: Güçlü (2012, p. 5).

Para Güçlu (2012), quadricoptero ou *quadrotor* é caracterizado por possuir quatro motores, tipicamente pequenos e de estrutura cruzada simples. Eles podem ser controlados autonomamente ou por controle remoto. Eles são mais usados em aplicações que necessitam de uma alta estabilidade de voo (GOMES; AQUINO, 2015).

A Figura 2 apresenta o funcionamento das hélices. A estrutura representada é em forma de 'X', mas o funcionamento das hélices não muda em relação a uma estrutura em forma de '+'.



Legenda:

E – eixo (x, y, z)
N, S, E, W – pontos cardeais.
T – direcionamento da força de torque

Figura 2 - Funcionamento das hélices de um quadricóptero
Fonte: Silva Filho, Rudiger e Nascimento (2011, p. 28).

O controle de movimento da aeronave pode ser realizado variando-se a velocidade relativa de cada rotor para alterar o empuxo e o torque produzido por cada um, como indicam as setas direcionadoras representadas na Figura 1. Sincronizar o controle desses dispositivos com sensores para gerar a estabilidade ao voo é o grande desafio nesse tipo de dispositivo (PANCERI et al., 2013).

Paw e Bala (2011) destacam que a abordagem tradicional usada para sintetizar, implementar e validar o sistema de controle de voo consome muito tempo e recursos. E, para esses autores, aplicar essa mesma técnica para pequenos VANT não é realístico. Para Alves (2012) um dos principais desafios do projeto de controladores de voo é a identificação da dinâmica de voo com fidelidade suficiente para ser usado em diferentes estágios de desenvolvimento do controlador. Esse autor ressalta que os modelos matemáticos utilizados são apenas uma aproximação da dinâmica do veículo. Eles são usados para descrever a complexa dinâmica de voos reais (PAW; BALAS, 2011). O desenvolvimento do controlador de voo deve ser abordado simultaneamente no contexto da modelagem dinâmica, controle e análise do modelo, simulação, projeto do controlador, implementação em tempo real e testes de voo (PAW; BALAS, 2011).

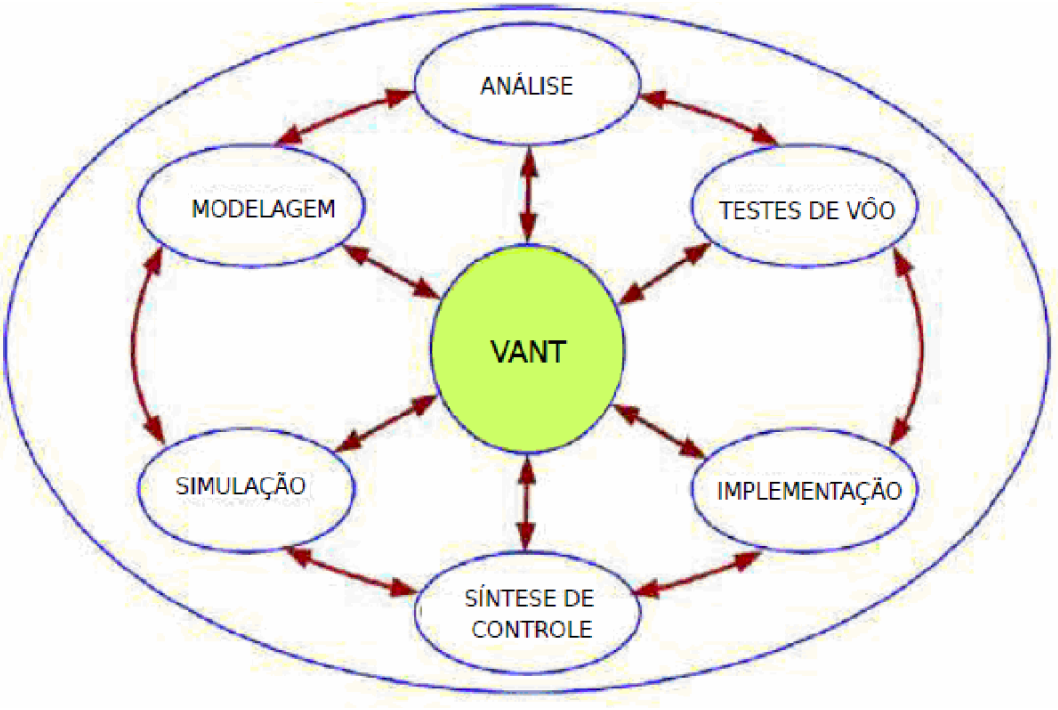


Figura 3 - Framework integrado para desenvolvimento de controle de voo
Fonte: traduzido de Paw e Balas (2011, p. 790).

As características não lineares e multivariáveis fazem com que Quadricópteros sejam difíceis de controlar (COZA; MACNAB, 2006). Técnicas tradicionais de controle lineares e não lineares têm sido aplicadas. Para desenvolver um controle confiável e assegurar a capacidade de um voo estável, torna-se importante o desenvolvimento de leis de controle simples e robusta (ADIGBLI et al., 2007). Alves (2012) faz um apanhado bastante extensor sobre soluções propostas que visam melhorar a estabilidade do voo.

3 MATERIAIS E MÉTODO

A seguir estão os materiais e o método utilizados para a modelagem e a implementação do sistema obtido resultado deste trabalho.

3.1 MATERIAIS

O Quadro 1 apresenta as tecnologias e as ferramentas utilizadas para o desenvolvimento do trabalho.

Ferramentas	Versão	Disponível em	Aplicação no projeto
Vertabelo		https://www.vertabelo.com	Modelagem de banco de dados
Cacoo		https://cacoo.com/	Desenvolvimento de diagramas
Visual Paradigm	Community Edition	https://www.visual-paradigm.com/	Desenvolvimento de diagramas
Angular.JS	1.5.3	https://angularjs.org/	Fremework para desenvolvimento cliente-side
JSON		http://www.json.org/json-pt.html	Formato de arquivo para transferência de dados entre servidor e cliente
Bower	1.3.9	https://bower.io/	Gerenciador de pacotes para frontend
Node.JS	4.4.1	https://nodejs.org/en/	Servidor
NPM	2.4.20	https://www.npmjs.com/	Gerenciador de bibliotecas para Node.JS
Express	4.13.1	http://expressjs.com/pt-br/	Framework web minimalista para Node.JS
SQLite 3	3.1.2	https://www.sqlite.org/	Bando de dados
Sequelize	3.20.0	https://sequelizejs.com/	Framework para persistência de dados
Gulp	3.9.0	http://gulpjs.com/	Automatizador de tarefas
ArDrone	2	http://www.parrot.com/usa/products/ardrone-2/	Drone
ArDrone-Autonomy	0.1.2	http://eschnou.github.io/ardrone-autonomy/	Biblioteca para auxiliar na automação do drone

Quadro 1 – Tecnologias e ferramentas utlizadas

A seguir são descritos os materiais utilizados para o desenvolvimento deste projeto e apresentados no Quadro 1:

a) Vertabelo

Vertabelo é um aplicativo *web* para modelagem de banco de dados. Além de gerar o *script* para o banco de dados, as informações podem ser compartilhadas e alteradas de forma colaborativa.

b) Cacoo

Para o desenvolvimento dos diagramas de caso de uso e diagramas de sequência, foi utilizado o aplicativo Cacoo, que assim como o Vertabelo, pode ser utilizado de forma colaborativa.

c) Visual Paradigm

Também para os diagramas de sequencia, foi utilizado o Visual Paradigm, devido ao conhecimento na ferramenta.

d) Angular.JS

Angular.JS é um *framework* MVC *client-side* que trabalha com tecnologias já estabelecidas: *HyperText Markup Language* (HTML), *Cascading Style Sheet* (CSS), JavaScript. O Foco de Angular.JS é *Single Page Application* (SPA) (ALMEIDA, 2011).

e) JSON

De acordo com o site oficial, *JavaScript Object Notation* (JSON) é uma formatação leve de troca de dados. Para seres humanos, é fácil de ler e escrever. Para maquinas é fácil gerar e interpretar.

f) Bower

Bower é um gerenciador de pacotes para *web*, voltados para *frontend*, que realiza grande parte das tarefas que seriam realizadas manualmente, como por exemplo, gerência das dependências da aplicação (ALMEIDA, 2011).

g) Node.JS

O Node.JS é altamente escalável e pode ser programando diretamente com diversos protocolos de rede e Internet, ou utilizar bibliotecas que acessam diversos recursos do sistema operacional. Para programar em Node.JS é utilizada a linguagem JavaScript. Sendo que suas principais características são: *Single-thread*, fazendo com que a aplicação tenha uma instância de uma *thread* principal por processo iniciado e *Event-Loop* que - como o Node.JS é orientado a eventos de entrada e saída, como por exemplo, conectar ao banco de dados - trata-se basicamente de um *loop* infinito, que a cada interação, verifica se algum evento da sua lista, foi executado (PEREIRA, 2016).

h) NPM

O Node Package Manager (NPM) é o gerenciador de pacotes do Node.JS, assim como por exemplo, o Maven, do Java. Ele se tornou o gerenciador padrão e foi integrado ao instalador do Node.JS desde a versão 0.6.X. Atualmente o site oficial do NPM (https://www.npmjs.com/) possui mais de 200.000 módulos, desenvolvidos por terceiros e por comunidades ativas.

i) Express

Express é um *framework* web para Node.JS, que foi fortemente inspirado no *framework* Sinatra da linguagem Ruby. O Express possibilita desenvolver uma aplicação *web* ou uma *Application Programming Interface (*API).

j) SQLite 3

O SQLite 3 é um banco de dados, com os dados salvos em um arquivo de extensão .sqlite. Sua sintaxe é muito semelhante aos outros bancos de dados relacionais, como Postgres e MySql.

k) Sequelize

Sequelize é um *framework Object-Relational Mapping* (ORM) para Node.JS. Ele utiliza o padrão *Promises,* que se trata de uma implementação semântica para tratamento de funções assíncronas.

l) Gulp

Gulp é uma ferramenta para automatizador de tarefas, que pode auxiliar na execução de tarefas repetitivas, como concatenação e testes.

m) ArDrone

O ArDrone, fabricado pela francesa Parrot, é um Drone com estrutura de fibra de carbono, pesando 380 gramas, contado com 4 motores de 14.5 W.

n) ArDrone-Autonomy

É uma biblioteca para Node.JS, que auxilia no voo autônomo do ArDrone. Com ela, é possível executar missões, descrevendo o caminho que o Drone percorrerá.

3.2 MÉTODO

A seguir estão descritas as etapas definidas para o desenvolvimento da aplicação e as principais atividades de cada uma dessas etapas.

Levantamento de requisitos

Para o levantamento de requisitos foram analisados diversos modelos de Drones que poderiam ser utilizados neste trabalho. Os principais elementos levados em conta para a escolha foram a disponibilidade do equipamento e documentação. Os Quadricópteros analisados foram o Ar Drone 2.0 da fabricante Parrot e o X550 da Xfly. O escolhido foi o Ar Drone 2.0 por existir uma API de código aberto disponibilizada para desenvolvedores. Partindo dessa escolha foram estudados os requisitos e levantado as

informações necessárias para a automação. Conclui-se, então, que seriam necessários um servidor *web* para armazenar as informações e fazer o processamento dos pontos e um cliente *web* para fazer o cadastro das informações prévias e enviar as informações como distancia que deve o Drone deve percorrer.

Implementação

Para o desenvolvimento do servidor foi escolhida a linguagem Node.JS como servidor e Angular.JS para o *frontend*, registrando os dados em um banco de dados SQLite. O principal motivo para a escolha dessas ferramentas, foi por que existe uma biblioteca para auxiliar na automação do voo do ArDrone disponível para Node.JS. Como Node.JS e Angular.JS tem grande compatibilidade com JSON e ambas utilizam JavaScript para desenvolvimento, foi criado um servidor que envia informações via JSON ao cliente, assim como o cliente, envia informações ao servidor utilizando JSON.

O cliente e os acesso às informações foram com as tecnologias HTML e Angular.JS, pois são linguagens amplamente utilizadas na internet e são altamente compatíveis umas com as outras.

Testes

Os testes realizados até o momento foram os de compatibilidade, armazenamento de dados e comunicação entre servidor e o Drone. Foram testadas as interfaces de captação e de armazenamento de dados por meio de testes informais e manuais, isto é, sem um plano de testes definido, além de serem verificados os dados obrigatórios e sua integridade. Foi, ainda, realizada análise dos cálculos referente à distância que o Drone percorrerá e também se o Drone realizou a missão definida.

Os testes realizados foram unitários, manuais e sem um plano de testes. Eles se voltaram para verificar a comunicação do servidor com o Drone visando definir a melhor forma de comunicação entre ambos. Nesses testes foram levadas em conta estabilidade da conexão, integridade dos dados e tempo de resposta.

4 RESULTADO

Este capítulo apresenta o resultado da realização deste trabalho. O resultado está centrado na modelagem e na implementação de funcionalidades básicas de cadastro para um sistema gerenciador de vôos de Drone.

4.1 ESCOPO DO SISTEMA

O sistema desenvolvido tem o objetivo auxiliar na automação de um vôo do Drone, enviando uma lista de operações, como: decolar e avançar com a respectiva distância que o Drone deve percorrer, feita a comunicação entre a interface com o usuário e o Drone por meio do Wi-Fi, fornecido pelo próprio Drone, para enviar esses comandos. O sistema é dividido em duas partes: servidor e *frontend.* O servidor é responsável pela comunicação com o Drone, persistência de dados e envio de informações. O *frontend* recebe as informações enviadas pelo servidor, calcula a distância em metros entre um ponto e outro e envia os dados para o servidor.

4.2 MODELAGEM DO SISTEMA

No Quadro 2 estão os requisitos funcionais identificados para o projeto desenvolvido.

Identificação	**Nome**	**Descrição**
RF01	Vôo	O usuário deverá definir o percurso a ser realizado pelo Drone. O percurso é definido por um ponto (localização) de origem e um ponto de destino.
RF02	Cadastros	Cadastro e consulta de informações sobre pontos e missões.
RF03	Localização atual	Verificação se a localização atual está próxima ao ponto de partida.

Quadro 2 - Requisitos funcionais

Os requisitos não funcionais são apresentados no Quadro 3.

Identificação	**Nome**	**Descrição**
RF01	Plataforma	Deve ser executado em um servidor web.
RF02	Altura	A altura que o Drone realizará o vôo deverá ser fixa.

Quadro 3 - Requisitos não funcionais

A Figura 4 apresenta as principais funções do sistema e os dois atores envolvidos. No diagrama os atores são: o Usuário que tem acesso à plataforma *web* e o Drone que realiza as ações. O ator Usuário deve cadastrar pontos e missões, depois disso deve definir os pontos na missão desejada e então escolher a missão que deseja executar, assim, o ator Drone poderá realizar a missão que contenham os pontos escolhidas.

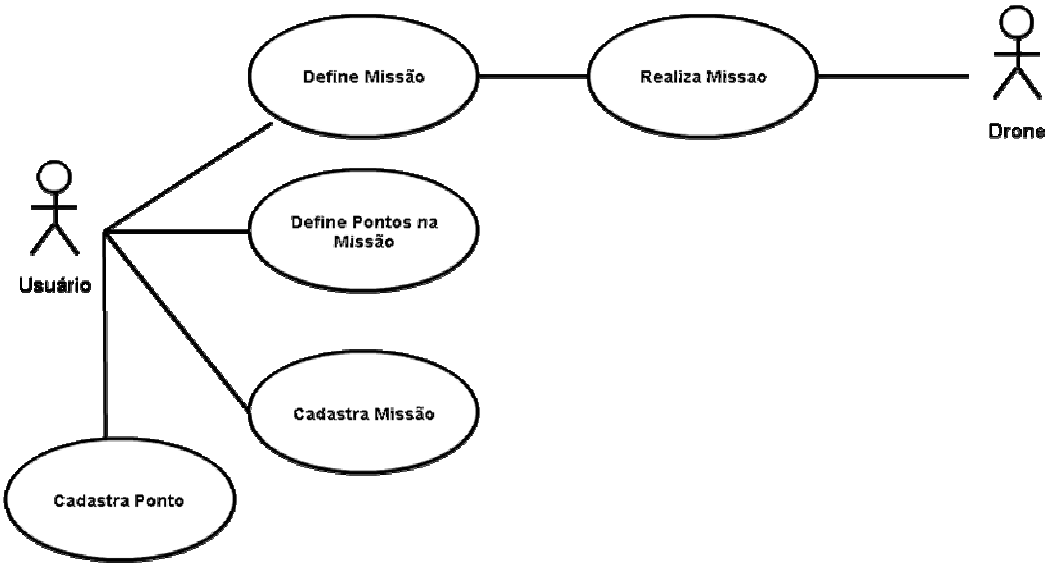


Figura 4 - Diagrama de casos de uso

A Figura 5 apresenta o diagrama de entidades e relacionamentos para o banco de dados da aplicação. O banco de dados é bastante simples com tabelas para armazenar dados dos pontos contendo latitude e longitude, dos pontos a serem percorridos em cada missão e missões que o Drone realizará. A missao_pontos é composta por duas localizações que definem os pontos de origem e de destino que são os pontos de deslocamento do Drone.

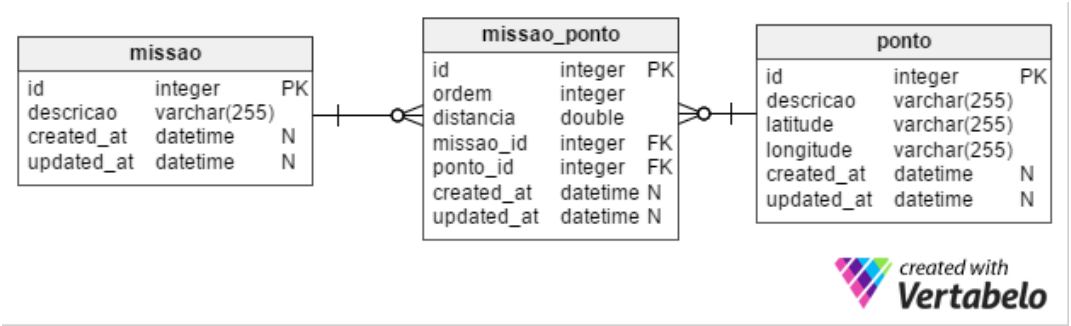


Figura 5 - Modelo lógico, representando relacionamento entre as entidades

Os campos da tabela missao_pontos estão presentes no Quadro 4. Um ponto é definido com a latitude e longitude. Na tabela missão_pontos, será marcada a distância entre o ponto A ao ponto B.

Campo	**Tipo**	**Nulo**	**Chave Primária**	**Chave Estrangeira**	**Observações**
id	Numérico	não	sim	não	
ordem	Numérico	não	não	não	
distancia	Numérico	não	não	não	
missao_id	Numérico	não	não	sim	Da tabela missao
ponto_id	Numérico	não	não	sim	Da tabela ponto
distancia	Numérico	não	não	não	
created_at	Data	não	não	não	
updated_at	Data	não	não	não	

Quadro 4 - Campos da tabela Missao_pontos

A tabela ponto (com os campos apresentados no Quadro 5) contém latitude e longitude, oriundos de um mapa digital. Nesses campos estarão as coordenadas geográficas dos lugares previamente cadastrados para o Drone percorrer.

Campo	Tipo	Nulo	Chave Primária	Chave Estrangeira	Observações
id	Numérico	não	sim	não	
nome	Texto	não	não	não	
latitude	Numérico	não	não	não	
longitude	Numérico	não	não	não	
created_at	Data	não	não	não	
updated_at	Data	não	não	não	

Quadro 5 - Campos da tabela localização

No Quadro 6 estão descritos os campos da tabela missao. A missão é construída a partir de 2 ou mais pontos, possuindo um campo descrição para facilitar a identificação do percurso a ser percorrido.

Campo	Tipo	Nulo	Chave Primária	Chave Estrangeira	Observações
id	Numérico	não	sim	não	
descricao	Numérico	não	não	não	
created_at	Data	não	não	não	
updated_at	Data	não	não	não	

Quadro 6 - Campos da tabela missao

Por meio de diagramas de sequência são identificados os padrões de interações entre os componentes do sistema. Na Figura 6 é apresentado como é definida a missão e os parâmetros necessários para o vôo.

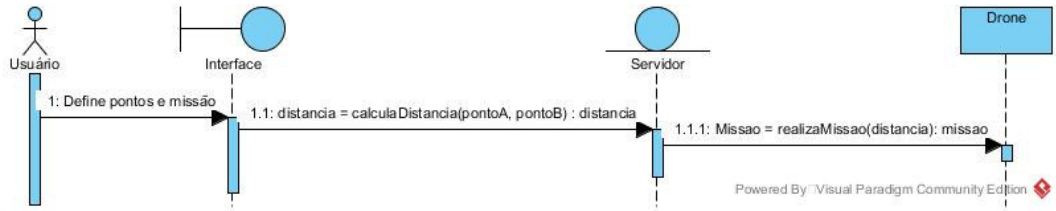


Figura 6 - Diagrama de sequência enviar uma missao para o drone

Na Figura 7 está o diagrama de sequência para o cadastro de pontos (um ponto geográfico) que é feito por meio da interface *web*. Um ponto é composta por duas coordenadas geográficas, que são latitude e longitude, e uma descrição para facilitar a identificação do ponto.

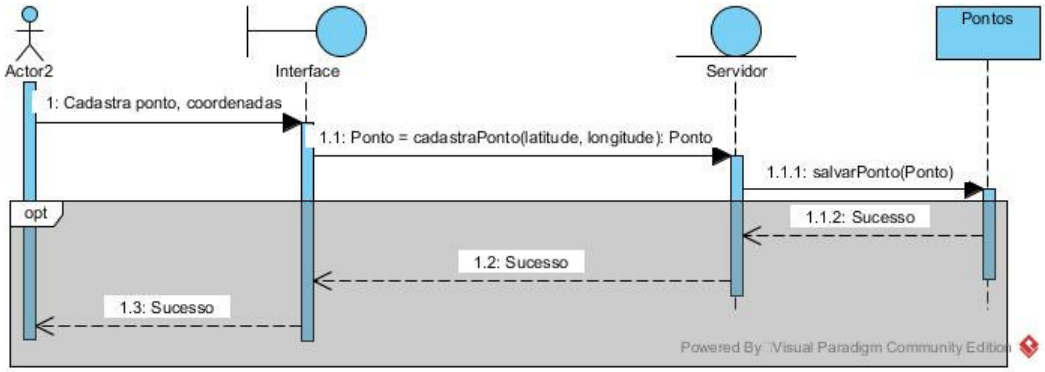


Figura 7 - Diagrama de sequência para cadastrar um ponto

4.3 APRESENTAÇÃO DO SISTEMA

O leiaute do sistema é composto por um menu principal, localizado na lateral da página, com um menu de navegação. A Figura 8 apresenta o menu principal. Ao acessar essa tela, estará disponível um mapa, que mostra por meio de marcadores, todas os pontos já cadastrados, que podem ser incluídas em uma missão além de mostrar a sua localização atual. Ao clicar sobre o mapa, será adicionado um novo marcador, sua latitude e longitude ficarão disponíveis nos respectivos campos. A partir disso, é possível incluir uma descrição para esse local, apertando no botão salvar. Esse local se torna um ponto que fica disponível para ser inserida em qualquer missão.

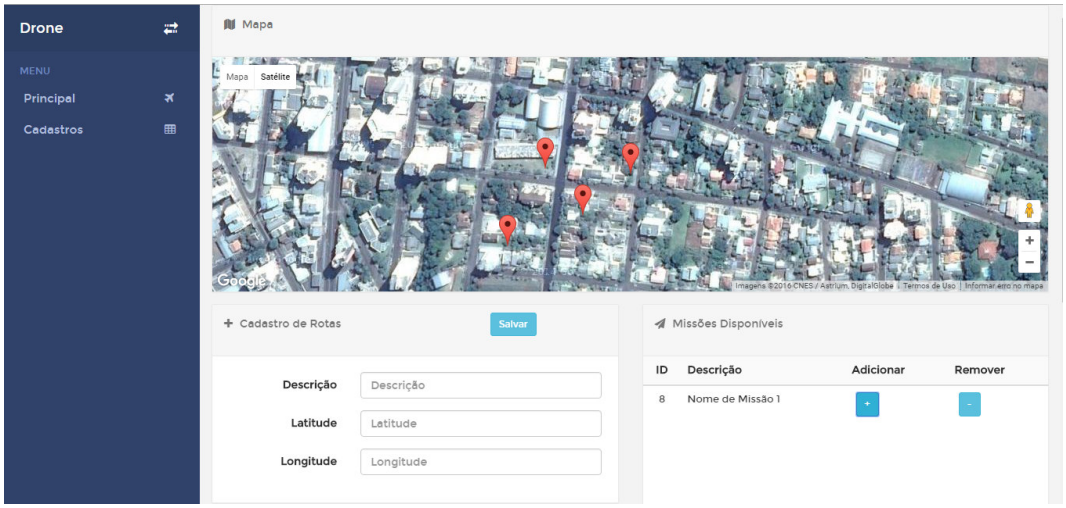


Figura 8 - Tela principal

Ao lado do cadastro de pontos, está presente uma lista com todas as missões disponíveis para serem executadas. Como mostra a Figura 9, ao clicar no botão adicionar, essa missão é detalhada e tem-se todas os pontos pertencentes a ela. Juntamente com a latitude, longitude de cada ponto e a distância em metros do próximo ponto, que é calculada pela fórmula descrita na Listagem 1. Clicando no botão viajar, as distâncias são enviadas ao servidor que envia esses comandos para o Drone.

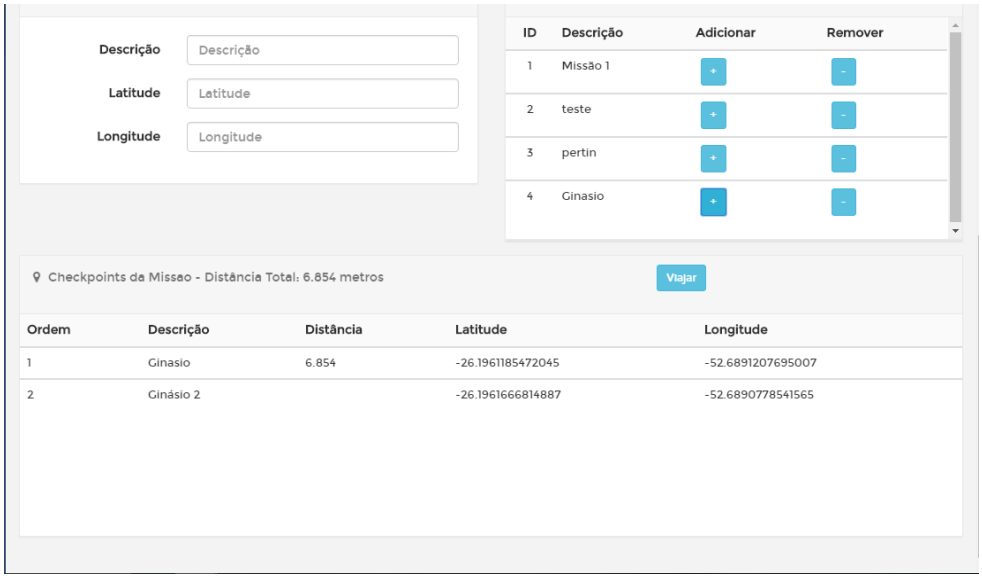


Figura 9 - Detalhando a missão

A fórmula utilizada para determinar a distância mais curta entre dois pontos A e B deve conter uma latitude e uma longitude para cada ponto. Essa fórmula pode haver um erro de 0,3%, especialmente nos extremos polares ou por longas distâncias. Para calcular

essa distância, admite-se que R é igual a 6371, que representa o raio da terra em quilômetros. Após isso, é calculada a diferença das latitudes. Para isso, a partir do valor da latitude do ponto A é subtraído o valor da latitude B e o resultado é multiplicado pelo valor de PI dividido por 180. O mesmo processo é realizado para a longitude, apenas substituindo os valores de latitude por longitude. Então, é realizada a divisão da diferença da latitude por 2. Desse valor é realizado o cálculo para obter o seno elevado à segunda potência. Esse mesmo processo é realizado para a diferença da longitude. A implementação dessa fórmula está na Listagem 1.

```
function getDistancia(lat1, lon1, lat2, lon2){
    var R = 6371; // Raio da terra em km
    var dLat = (lat2-lat1) * Math.PI / 180;  // Javascript functions
in radians
    var dLon = (lon2-lon1) * Math.PI / 180;
    var a = Math.sin(dLat/2) * Math.sin(dLat/2) +
Math.cos(toRadians(lat1)) *
Math.cos(toRadians(lat2)) * Math.sin(dLon/2) * Math.sin(dLon/2);
    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    var d = (R * c)  * 1000.0 //distancia em metros
        return d;
}
```

Listagem 1 – Fórmula em JavaScript para retornar a distancia em metros, entre duas coordenadas

O mesmo procedimento é utilizado para a latitude A e a latitude B, que antes de passarem por esses cálculos, seus valores devem ser convertidos de grau para radiano, utilizando a fórmula da Listagem 2.

```
function toRadians(degrees) {
    return degrees * Math.PI / 180;
};
```

Listagem 2 – Fórmula em JavaScript para transformar um valor em grau para radiano

Ao clicar no menu Cadastros, será redirecionado para a página de cadastro de missão, apresentada na Figura 10. Nela, no lado esquerdo da tela, é apresentada uma lista com todas os pontos disponíveis, para serem adicionados a uma nova missão. Clicando no botão com símbolo +, essa missão vai para a tabela de pontos na missão, que está ao lado direito da tela, já com a ordem que deve ser executada pelo Drone.

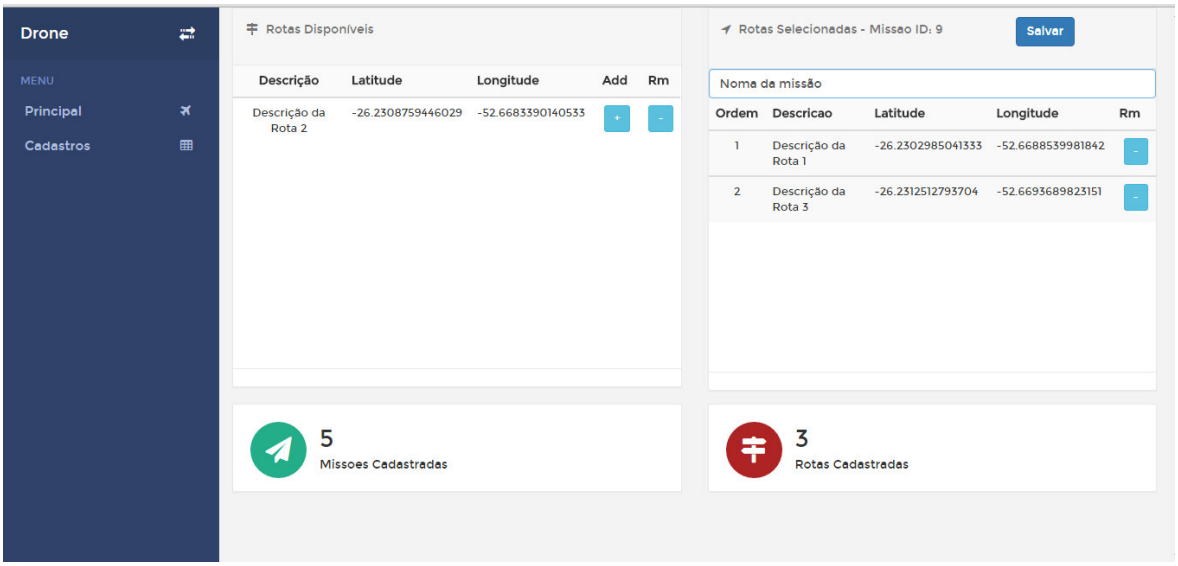


Figura 10 - Tela para cadastro de missão

Após a inclusão dos pontos e da descrição da missão, basta clicar no botão Salvar. Caso tudo ocorra corretamente será exibida uma mensagem que apresenta o resultado. Além disso, é possível redefinir a ordem de execução dos pontos, clicando no símbolo de -, presente na lista de cadastro de Missão. Com isso, o ponto selecionado voltará para a tabela de pontos disponíveis.

A biblioteca *ardrone-autonomy*, implementada no servidor Node.JS e executada em um *notebook*, faz o envio dos comandos necessários para que o drone realize a missão selecionada, exemplo: decolar, avançar e pousar. Esses comandos são enviados via Wi-Fi, fornecido pelo próprio drone no qual o notebook deve estar conectado. Após a conexão ser realizada, os comandos enviados são executados na mesma ordem em que eles foram implementados no código e após o envio, a biblioteca armazena os comandos na memória interna do drone. Fazendo, assim, com que não seja mais necessária a conexão com o *notebook*, a menos que seja necessário enviar o comando de emergência para pouso imediato.

4.4 IMPLEMENTAÇÃO DO SISTEMA

O leiaute do sistema é divido entre o menu contido na coluna esquerda estático e o conteúdo da página. Para simplificar o desenvolvimento, uma página modelo foi utilizada chamada “index.html”. Essa página contém os itens de navegação do sistema e também

faz a importação dos *scripts* JavaScript utilizados e arquivos CSS. Como o servidor foi construído em Node.JS e o *frontend* utilizando Angular.JS, as páginas foram feitas com HTML e a utilização das diretivas do Angular.JS. A Listagem 3 apresenta o código do index.html.

```
<!doctype html>
<html lang="pt-BR" ng-app="RDash">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="description" content="Simple Map">
  <meta name="keywords" content="ng-map,AngularJS,center">
  <meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
  <script
src="https://maps.google.com/maps/api/js?libraries=placeses,visualization
,drawing,geometry,places"></script>
      <title>Automação de Vôo</title>
  <!-- STYLES -->
  <link rel="stylesheet" href="lib/css/main.min.css"/>
  <!-- SCRIPTS -->
  <script src="lib/js/main.min.js"></script>
  <!-- Custom Scripts -->
  <script type="text/javascript" src="js/dashboard.min.js"></script>
</head>
<body ng-controller="MasterCtrl">
  <div id="page-wrapper" ng-class="{'open': toggle}" ng-cloak>

    <!-- Sidebar -->
    <div id="sidebar-wrapper">
      <ul class="sidebar">
        <li class="sidebar-main">
          <a ng-click="toggleSidebar()">
            Drone
            <span class="menu-icon glyphicon glyphicon-transfer"></span>
          </a>
        </li>
        <li class="sidebar-title"><span>MENU</span></li>
        <li class="sidebar-list">
          <a   href="#">Principal   <span   class="menu-icon   fa   fa-
plane"></span></a>
        </li>
        <li class="sidebar-list">
          <a href="#/statistics">Cadastros <span class="menu-icon fa fa-
table"></span></a>
        </li>
      </ul>
    </div>
    <!-- End Sidebar -->

    <div id="content-wrapper">
      <div class="page-content">
        <!-- Main Content -->
        <div ui-view></div>
      </div><!-- End Page Content -->
    </div><!-- End Content Wrapper -->
  </div><!-- End Page Wrapper -->
</body>
</html>
```

Listagem 3 – index.html

Como foi utilizado Angular.JS para desenvolver o *frontend*, é necessário definir quais módulos a aplicação irá utilizar. A Listagem 4 mostra como isso é realizado.

```
angular.module('RDash', ['ui.bootstrap', 'ui.router', 'ngCookies',
'ngResource', 'ngMap']);
```

Listagem 4 – module.js

A página principal será direcionada por meio do Angular.JS como mostra a Listagem 5. A partir disso, será feita a junção do index.html e da página dashboard.html.

```
'use strict';

angular.module('RDash').config(['$stateProvider',
'$urlRouterProvider',
    function($stateProvider, $urlRouterProvider) {
        $urlRouterProvider.otherwise('/');
        $stateProvider
            .state('index', {
                url: '/',
                templateUrl: 'templates/dashboard.html'
            })
            .state('statistics', {
                url: '/statistics',
                templateUrl: 'templates/statistics.html'
            });
    }
]);
```

Listagem 5 – routes.js

Com o procedimento citado anteriormente, toda requisição ao caminho "/" será direcionada a página dashboard.html, assim como, toda requisição do caminho "/statistics" será direcionada a pagina statistics.html. Esse redirecionamento é controlado pelo AngularJS no arquivo routes.js.

A Listagem 6 mostra a implementação da página dashboard.html.

```
<div class="row">
<div class="col-lg-12 col-xs-12">
<rd-widget>
    <rd-widget-header icon="fa-map" title="Mapa">
    </rd-widget-header>
    <rd-widget-body classes="map no-padding">

        <ng-map default-style="true" map-type-id="SATELLITE" zoom="17"
on-click="addMarker()"
center="[{{localAtual.latitude}},{{localAtual.longitude}}]">
        <div ng-repeat="rota in rotas">
        <marker         position="{{rota.latitude}},{{rota.longitude}}"
title="{{rota.descricao}}"></marker>
    </div>
        </ng-map>
    </rd-widget-body>
</rd-widget>
</div>
```

```
</div>
<div class="row">
<div class="col-lg-6 col-md-6 col-xs-12">
<rd-widget>
<form class="form-horizontal" role="form">
    <rd-widget-header icon="fa-plus" title="Cadastro de Pontos">
        <button       class="btn       btn-sm       btn-info"       ng-
click="saveCheckpoint()">Salvar</button>
    </rd-widget-header>
    <rd-widget-body classes="map">
            <div class="form-group">
                <uib-alert   type="success"   ng-if="msg">{{msg}}</uib-
alert>
                <label    for="descricao"    class="col-sm-4    control-
label">Descrição</label>
                <div class="col-sm-8">
                <input   type="text"   ng-model="checkpoint.descricao"
class="form-control" id="title" placeholder="Descrição" required>
                </div>
            </div>
            <div class="form-group">
                <label    for="latitude"    class="col-sm-4    control-
label">Latitude</label>
                <div class="col-sm-8">
                <input    type="text"    ng-model="checkpoint.latitude"
name="latitude"           class="form-control"           id="latitude"
placeholder="Latitude" required>
                </div>
            </div>
            <div class="form-group">
                <label    for="longitude"    class="col-sm-4    control-
label">Longitude</label>
                <div class="col-sm-8">
                <input   type="text"   ng-model="checkpoint.longitude"
class="form-control" id="longitude" placeholder="Longitude" required>
                </div>
            </div>
    </rd-widget-body>
    </form>
</rd-widget>
</div>
<div class="row">
<div class="col-lg-6 col-md-6 col-xs-12">
<rd-widget>
    <rd-widget-header       icon="fa-paper-plane"       title="Missões
Disponíveis">
    </rd-widget-header>
    <rd-widget-body classes="medium no-padding">
        <div class="table-responsive">
            <table class="table">
                <uib-alert              type="danger"              ng-
if="msgSucessoMissaoExcluida">{{msgSucessoMissaoExcluida}}</uib-alert>
                <thead>
                    <tr><th                                    class="text-
center">ID</th><th>Descrição</th><th>Adicionar</th><th>Remover</th></t
r>
                </thead>
                <tbody ng-repeat="missao in missoes">
                    <tr>
                        <td class="text-center">{{missao.id}}</td>
                        <td>{{missao.descricao}}</td>
```

```
                        <td>
                            <button class="btn btn-sm btn-info" ng-
click="selectMissao(missao.Missao_pontos)">+</button>
                        </td>
                        <td><button class="btn btn-sm btn-info" ng-
click="excluirMissao($index, missao.id)">-</button></td>
                    </tr>
                </tbody>
            </table>
        </div>
    </rd-widget-body>
<rd-widget>
</rd-widget></rd-widget></div>
<div class="col-lg-12">
<rd-widget>
    <rd-widget-header icon="fa fa-map-marker" title="Checkpoints da
Missao - Distância Total: {{distanciaTotal}} metros">
        <button class="btn btn-sm btn-info" ng-
click="">Viajar</button>
    </rd-widget-header>
    <rd-widget-body classes="medium no-padding">
        <div class="table-responsive">
            <uib-alert type="danger" ng-
if="missao.msg">{{missao.msg}}</uib-alert>
            <table class="table">
            <thead>

<tr><th>Ordem</th><th>Descrição</th><th>Distância</th><th>Latitude</th
><th>Longitude</th></tr>
            </thead>
            <tbody ng-repeat="rota in missaoSelecionada">
                <tr>
                <td>{{rota.ordem}}</td>
                <td>{{rota.Rota.descricao}}</td>
                <td>{{rota.distancia}}</td>
                <td>{{rota.Rota.latitude}}</td>
                <td>{{rota.Rota.longitude}}</td>
                </tr>
            </tbody>
            </table>
        </div>
    </rd-widget-body>
</rd-widget>
</div>
</div>
</div>
```

Listagem 6 – master-ctrl.js

Todas as variáveis e funcionalidades utilizadas nessas páginas estão implementadas no master-ctrl.js, que está descrito na Listagem 7. Nela, também é mostrado como o Angular.JS é utilizado para fazer requisições ao servidor, enviar e definir variáveis de escopo para serem utilizadas nas páginas HTML.

```
/**
 * Master Controller
 */
```

```
angular.module('RDash')
    .controller('MasterCtrl', ['$scope', '$cookieStore', '$resource',
MasterCtrl]);

function MasterCtrl($scope, $cookieStore, $resource) {

    /**
     * Sidebar Toggle & Cookie Control
     */
    var mobileView = 992;
    var checkpoints = $resource("http://localhost:3000/rotas");
    var missoesR = $resource("http://localhost:3000/missao");
    var missoesAll = $resource("http://localhost:3000/missao/all");
    var missaoRotas = $resource("http://localhost:3000/missaoRotas");
    var Missao = $resource('http://localhost:3000/missao/:id',
{id:'@id'});
    var Rota = $resource('http://localhost:3000/rotas/:id',
{id:'@id'});
    $scope.lastCheckpoint = {};
    var ultimoIDResource =
$resource('http://localhost:3000/missao/ultimoID');


    $scope.totalMissoes = 0;
    $scope.totalRotas = 0;
    $scope.localAtual = [];

    $scope.loadData = function () {

        $scope.missoes = missoesR.query();
        $scope.rotas = checkpoints.query();
        getPosicaoAtual();
        nextMissaoId = $scope.missoes.length;
        $scope.rotasDisponiveis = checkpoints.query();
        $scope.checkpointsSalvos = checkpoints.query();
        $scope.missoesSize = $scope.checkpointsSalvos.length;

        $scope.ultimoIDResource = ultimoIDResource.query();

        allMissions = missoesAll.query();

        missao = allMissions;
        missao.$promise.then(function(value) {

            $scope.totalMissoes = value.length; // Success!

        }, function(reason) {
            console.log(reason); // Error!
        });

        $scope.checkpointsSalvos.$promise.then(function(value) {

            $scope.totalRotas = value.length; // Success!

        }, function(reason) {
            console.log(reason); // Error!
        });
    };
    $scope.loadData();
```

```
    $scope.checkpoint = {};

    function getPosicaoAtual(){
        if (navigator.geolocation) {
          var timeoutVal = 10 * 1000 * 1000;
          navigator.geolocation.getCurrentPosition(
            displayPosition,
            displayError,
            { enableHighAccuracy: true, timeout: timeoutVal,
maximumAge: 0 }
          );
        }
        else {
          alert("Geolocation is not supported by this browser");
        }
    }
    function displayPosition(position) {
        $scope.localAtual.latitude = position.coords.latitude;
        $scope.localAtual.longitude = position.coords.longitude;
        $scope.rotas.push({latitude:$scope.localAtual.latitude,
longitude: $scope.localAtual.longitude, descricao:"Você está aqui
perto"});
    }

    function displayError(error) {
      var errors = {
        1: 'Permission denied',
        2: 'Position unavailable',
        3: 'Request timeout'
      };
      alert("Error: " + errors[error.code]);
    }

    $scope.saveCheckpoint = function() {

$resource("http://localhost:3000/rotas").save($scope.checkpoint,
function(result) {

            $scope.msg = "Registro inserido com sucesso!";
            $scope.loadData();

            markerAdicionado = false;
        });

    };
    $scope.missaoTemp = {};
    $scope.salvarMissao = function(missaoID) {
        missoesR.save({"id": missaoID,
                    "descricao": $scope.missaoTemp.descricao});

        for (i = 0; i < $scope.rotasSelecionadas.length; i++) {
            pontoUm = $scope.rotasSelecionadas[i];

            pontoDois = $scope.rotasSelecionadas[i+1];

            var distancia = null;
            if(i < $scope.rotasSelecionadas.length -1){
                distancia = getDistancia(pontoUm.latitude,
pontoUm.longitude, pontoDois.latitude, pontoDois.longitude);
                distancia = distancia.toFixed(3);
```

```
            }

            missaoRotas.save({
                "missao_id": missaoID,
                "ordem": $scope.rotasSelecionadas[i].ordem,
                "rota_id": $scope.rotasSelecionadas[i].rota_id,
                "distancia": distancia
            })
        }

        $scope.rotasSelecionadas = [];
        $scope.loadData();
        $scope.ultimoIDResource = ultimoIDResource.query();

        $scope.missoes = missoesR.query();

        $scope.msgSucessoMissaoSalva = "Missao " +
$scope.missaoTemp.descricao +" salva com sucesso!";
    };

    $scope.excluirMissao = function(idx, missaoId){
        Missao.delete({}, {'id':missaoId});
        $scope.msgSucessoMissaoExcluida = "Missao Removida com
Sucesso!";

        $scope.missaoSelecionada = [];
        $scope.missoes.splice(idx, 1);
    };
    $scope.distanciaTotal = 0;
    $scope.missaoSelecionada = [];
    $scope.selectMissao = function(missao) {
        $scope.missaoSelecionada = missao;
        $scope.distanciaTotal = 0;
        for (i = 0; i < missao.length; i++) {
            $scope.distanciaTotal = $scope.distanciaTotal +
missao[i].distancia;
            console.log(missao[i].distancia);
        }
    };

    var markerAdicionado = false;

    $scope.addMarker = function(event) {

        var ll = event.latLng;
        if(markerAdicionado){
            $scope.rotas.pop();
        }
        $scope.rotas.push({latitude:ll.lat(), longitude: ll.lng()});

        $scope.checkpoint = ({latitude:ll.lat(), longitude:
ll.lng()});
        markerAdicionado = true;
    };

    function removeItem(arr) {
        var what, a = arguments, L = a.length, ax;
        while (L > 1 && arr.length) {
            what = a[--L];
            while ((ax= arr.indexOf(what)) !== -1) {
```

```
                arr.splice(ax, 1);
            }
        }
        return arr;
    }
    $scope.rotasSelecionadas = new Array();
    var ordemNro = 0;

    $scope.adicionarRota = function(rotaId, descricao, lt, lg){
        limparMensagensDeAviso();
        ordemNro = $scope.rotasSelecionadas.length +1;
        $scope.rotasSelecionadas.push({
            ordem:ordemNro,
            missao_id: nextMissaoId,
            rota_id: rotaId,
            descricao: descricao,
            latitude: lt,
            longitude: lg
        });

        for(var i in $scope.rotasDisponiveis)
        {
             var id = $scope.rotasDisponiveis[i].id;
             if(id == rotaId){
                $scope.rotasDisponiveis.splice(i, 1);
                break;
             }

        };
    };

    function limparMensagensDeAviso(){
        $scope.msgSucessoMissaoSalva = null;
        $scope.msg = null;
        $scope.msgSucessoRotaExcluida= null;
        $scope.msgSucessoMissaoExcluida = null;
    };

    $scope.excluirRota = function(rotaId){
        limparMensagensDeAviso();
        var rotaExluir = $resource("http://localhost:3000/rotas/" +
rotaId);

        Rota.delete({}, {'id':rotaId});
        $scope.msgSucessoRotaExcluida = "Rota removida com sucesso!";
        $scope.loadData();

    };

    $scope.removerRota = function(ordem, rotaId){

        $scope.rotasSelecionadas.splice(ordem -1, 1);

        var rota = $resource("http://localhost:3000/rotas/" +
rotaId);
        ordemRota = 1;
        for(var i in $scope.rotasSelecionadas)
        {
            $scope.rotasSelecionadas[i].ordem = ordemRota;
            ordemRota ++;
```

```
        };

        $scope.rotasDisponiveis.push(rota.get());
    };

    $scope.getWidth = function() {
        return window.innerWidth;
    };

    $scope.$watch($scope.getWidth, function(newValue, oldValue) {
        if (newValue >= mobileView) {
            if (angular.isDefined($cookieStore.get('toggle'))) {
                $scope.toggle = ! $cookieStore.get('toggle') ? false
: true;
            } else {
                $scope.toggle = true;
            }
        } else {
            $scope.toggle = false;
        }
    });

    $scope.toggleSidebar = function() {
        $scope.toggle = !$scope.toggle;
        $cookieStore.put('toggle', $scope.toggle);
    };

    window.onresize = function() {
        $scope.$apply();
    };
}
```

Listagem 7 – master-ctrl.js

A implementação do módulo controlador da interface foi baseada no conceito de *Single Page Application* (SPA) e padrão de desenvolvimento *Representational State Transfer* (REST) utilizando JavaScript e Angular.JS. SPA consiste basicamente em equilibrar e dividir o processamento de dados entre o servidor e cliente. Dessa forma, o consumo de banda é menor pois são menos dados trafegando e a interação do usuário com o uso de aplicativo é melhorada, uma vez que não é necessário recarregar a página para navegar entre as páginas.

Na Listagem 7 é criado um módulo chamado RDash, e declarada uma função controladora chamada *MasterCtrl*, passando como parâmetros algumas dependências da página. Dentro dessa função, que controla a interface *web*, existem as variáveis globais e métodos, assim como a função *loadData()* que carrega da API os dados cadastrados no banco do servidor, ou funções que fazem a manipulação dos dados da interface antes de enviar requisições de salvar ou excluir para a API.

Também é implementada a função que busca a localização geográfica atual por meio da busca de coordenadas pela API de geolocalização da W3C, com o objetivo de

facilitar a navegação pelo mapa principal, inserindo um ponto e centralizando o mapa na posição atual do cliente.

O padrão de desenvolvimento REST, consiste em trabalhar na criação e manipulação de recursos. Esses recursos são entidades da aplicação para realizar um *Create-Read-Update-Delete* (CRUD).

A API desenvolvida para este projeto tem como principal recurso, a tarefa Missão, acessada pela url /missão. Nela serão vinculadas os pontos que pertenceram a essa Missão, fazendo com isso a utilização do padrão *Model-View-Route* (MVR). Na Listagem 8 é apresentado o arquivo que permite carregar e injetar as dependências que serão utilizadas em todo o projeto.

```
import express from "express";
import consign from "consign";

const app = express();

consign()
  .include("libs/config.js")
  .then("db.js")
  .then("libs/middlewares.js")
  .then("routes")
  .then("libs/boots.js")
  .into(app);
```

Listagem 8 – index.js

Nesse arquivo é realizada a importação dos módulos que auxiliam no carregamento e injeção das dependências, sendo que, primeiro é carregado o arquivo de configuração para acesso ao banco de dados (Listagem 9); em seguida o arquivo responsável por criar as entidades no banco de dados (Listagem 10); depois o arquivo que contem os *middlewares*, que são outras bibliotecas e aplicações utilizadas para auxiliar no desenvolvimento desta (Listagem 11) e a pasta que contém todos os pontos (Listagem 12) e, por último, o arquivo de inicialização da aplicação (Listagem 13). A ordem de inclusão dos registros, neste caso, faz diferença, pois para que um arquivo carregue, por exemplo, “db.js”, é necessário que o arquivo “config.js” já esteja carregado, pois é esse arquivo que contém o caminho do banco de dados e outros.

```
module.exports = {
    database: "drone",
    username: "",
    password: "",
    params: {
      dialect: "sqlite",
      storage: "drone.sqlite",
      define: {
```

```
        underscored: true
      }
    }
};
```

Listagem 9 – config.js

A Listagem 10 apresenta o arquivo em que a conexão com o banco de dados é feita, utilizando um arquivo JSON para informar o tipo do banco, o nome do banco, usuário e senha.

```
import fs from "fs";
import path from "path";
import Sequelize from "sequelize";

let db = null;

module.exports = app => {
  if(!db){
    const config = app.libs.config;
    const sequelize = new Sequelize(
      config.database,
      config.username,
      config.password,
      config.params
    );
    db = {
      sequelize,
      Sequelize,
      models: {}
    };
    const dir = path.join(__dirname, "models");
    fs.readdirSync(dir).forEach(file => {
      const modelDir = path.join(dir, file);
      const model = sequelize.import(modelDir);
      db.models[model.name] = model;
    });
    Object.keys(db.models).forEach(key => {
      db.models[key].associate(db.models);
    });
  }
  return db;
};
```

Listagem 10 – db.js

O arquivo em questão trata-se do documento responsável para vincular os arquivos de *model* (Listagem 11), em entidades para o banco de dados. Essa tarefa é realizada com o apoio do *framework* Sequelize.

```
import bodyParser from "body-parser";
module.exports = app => {
  app.set("port", 3000);
  app.use(bodyParser.json());
  app.use(function(req, res, next) {
```

```
    res.header("Access-Control-Allow-Origin", "*");
    res.header("Access-Control-Allow-Headers", "Origin, X-Requested-
With, Content-Type, Accept");
    res.header("Access-Control-Allow-Methods", "GET, POST, PUT,
DELETE");
    next();
  });
  app.use((req, res, next) => {
    delete req.body.id;
    next();
  })
}
```

Listagem 11 – middlewares.js

Na Listagem 11 são definidos os *middlewares* utilizados na aplicação, como por exemplo, *bodyParser* que é utilizado para fazer a conversão de objetos em JSON.

Como a aplicação desenvolvida contém relacionamento entre as tabelas, verificou a necessidade de utilizar uma ferramenta que auxiliasse nesse processo. Para isso o Sequelize foi escolhido. O Sequelize faz a leitura dos arquivos e os transforma em entidade para o banco de dados, cada uma com os relacionamentos já definidos. Na Listagem 12, é mostrado como é feito o *model*, utilizando as especificações do Sequelize.

```
module.exports = (sequelize, DataType) => {
  const Missao_rotas = sequelize.define("Missao_rotas", {
    id: {
      type: DataType.INTEGER,
      primaryKey: true,
      autoincrement: true
    },
    ordem: {
      type: DataType.INTEGER,
      allowNull: false,
      validate: {
        notEmpty: true
      }
    },
    distancia: {
      type: DataType.INTEGER,
      allowNull: true
    }
  },{
    classMethods:{
      associate: (models) => {
        Missao_rotas.belongsTo(models.Missao);
      },
      associate: (models) => {
        Missao_rotas.belongsTo(models.Rotas);
      }
    }
  });

  return Missao_rotas;
}
```

Listagem 12 – missao_pontos.js

A função sequelize.define, presente no início da Listagem 12, é responsável por criar ou alterar uma entidade no banco de dados, isso ocorre quando é feita a sincronização, configurada no arquivo "db.js" (Listagem 10). No modelo apresentado na Listagem 12, id é do tipo inteiro (DataType.INTEGER). Ele representa uma chave primária (primaryKey: true) e seu valor é autoincremental (autoIncrement: true) a cada novo registro. O campo ordem é semelhante ao id, pois também é inteiro. Nele foi incluído o atributo allowNull: *false* para não permitir valores nulos e também um campo validador, que verifica se o campo não está vazio (validate.notEmpty: true). Por último, o terceiro parâmetro permite incluir funções estáticas encapsuladas dentro do atributo classMethods. Nele foi criada a função associate(models), que vai permitir realizar uma associação entre os modelos. Neste caso, o relacionamento foi estabelecido por meio da função Missao_pontos.belongsTo(models.Missao) e por meio da função Missao_pontos.belongsTo(models.Pontos).

A Listagem 13 apresenta a configuração para inicialização do servidor, passando a porta que foi configurada no arquivo "middlewares.js".

```
module.exports = app => {
  app.db.sequelize.sync().done(() => {
    app.listen(app.get('port'), () => {
      console.log(`Drone API - porta ${app.get('port')}`);
    });
  });
}
```

Listagem 13 – boots.js

O servidor é iniciado pela função app.listen(). O método app.db.sequelize.sync() é utilizado para sincronizar o banco de dados com os modelos do servidor.

Como apresenta a Listagem 14, foram implementadas várias funções: funções para definir URLs ou *endpoints* para as funções de CRUD da classe Missão (/missao e /missao/:id); função para retorno do último id cadastrado (/missao/ultimoID); função para retornar todas as missões e suas respectivas missões pontos e pontos; função para enviar a missão ao Drone (/missao/viajar/voar); e função para enviar um comando de parada imediata ao Drone (/missao/viajar/parar).

```
var Sequelize = require('sequelize');
var autonomy = require('ardrone-autonomy');
var mission  = autonomy.createMission();

function executar(){
    mission.run(function (err, result) {
      if (err) {
```

```
          console.trace("Algo de errado, não está certo: %s",
err.message);
          mission.client().stop();
          mission.client().land();
      } else {
          console.log("Isso é tudo por hoje, pessoal!");
      }
  });
}

module.exports = app => {
  const Missao = app.db.models.Missao;
  const Missao_pontos = app.db.models.Missao_pontos;
  const Pontos = app.db.models.Pontos;
  var db = app.db.sequelize;
  var distancias = [];
  app.route("/missao")
    .get((req, res) => {
      Missao.findAll({
        include: [{
          model: Missao_pontos,
          where: { state: Sequelize.col('missao.id') },
          include: [{
            model: Pontos,
            where: { state: Sequelize.col('missao_pontos.ponto_id') }
          }],
        }]
      })
        .then(result => res.json(result))
        .catch(error => {
          res.status(412).json({msg: error.message});
        });
    })
    .post((req, res) =>{
      Missao.create(req.body)
        .then(result => res.json(result))
        .catch(error => {
          res.status(412).json({msg: error.message});
        });
    });
  app.route("/missao/all")
    .get((req, res) => {
      Missao.findAll()
        .then(result => res.json(result))
        .catch(error => {
          res.status(412).json({msg: error.message});
        });
    });
  app.route("/missao/ultimoID")
    .get((req, res) => {
      db.query("select id from Missaos order by id desc LIMIT 1", {
       type: Sequelize.QueryTypes.SELECT
      })
        .then(result => {
          res.json(result);
        })
    })
  app.route("/missao/viajar/parar")
    .get((req, res) => {
        console.log('Parada forçada.');
```

```
        mission.control().disable();
        mission.client().land();
        res.sendStatus(200);
    });
  app.route("/missao/viajar/:id")
    .get((req, res) => {
      Missao.findOne({
        include: [{
          model: Missao_pontos,
          where: { state: Sequelize.col('missao.id') },
          include: [{
            model: Pontos,
            where: { state: Sequelize.col('missao_pontos.ponto_id') }
          }],
        }],
        where: req.params
    })
      .then(result => {
        if(result){
            console.log(result.Missao_pontos[0].distancia);
mission.takeoff().altitude(1).forward(result.Missao_pontos[0].distanc
ia).land();
            executar();
          res.sendStatus(200);
        };
      })
    })
  app.route("/missao/:id")
    .get((req, res) => {
      Missao.findOne({
        include: [{
          model: Missao_pontos,
          where: { state: Sequelize.col('missao.id') },
          include: [{
            model: Pontos,
            where: { state: Sequelize.col('missao_pontos.ponto_id') }
          }],
        }],
        where: req.params
      })
        .then(result => {
          if(result){
          var latitudeAux;
          var longitudeAux;
          if (result.Missao_pontos.length <= 1){
            res.status(206).json({msg: 'Ops! parece que você não
pontos o suficiente'});
          } else{
            res.json(result);
          };
          } else{
            res.sendStatus(404);
          }
        })
        .catch(error => {
          res.status(412).json({msg: error.message});
        });
    })
    .put((req, res) => {
      Missao.update(req.body, {where: req.params})
```

```
        .then(result => res.sendStatus(204))
        .catch(error => {
          res.status(412).json({msg: error.message});
        });
    })
    .delete((req, res) =>{
      Missao.destroy({where: req.params})
        .then(result => res.sendStatus(204))
        .catch(error => {
          res.status(412).json({msg:error.message});
        });
    });
};
```

Listagem 14 – routes/missao.js

No início do arquivo (Listagem 14), são declaradas variáveis que serão utilizadas no restante do arquivo, var autonomy = require('ardrone-autonomy'), por exemplo, é a biblioteca responsável por auxiliar na comunicação e envio de comandos ao Drone. Após importar a biblioteca ardrone-autonomy é possível instanciar o objeto que faz os envios dos comandos ao Drone (var mission = autonomy.createMission();). A função "executar()" tem como finalidade iníciar o vôo utilizando o comando "mission.run". Então é implementado um *callback* para verificar se a missão foi ou não bem sucedida, ou seja, se o Drone conseguiu finalizar a missão enviada. A utilização dessa biblioteca juntamente com o Drone, necessita somente com que o Wi-Fi do computador em que o servidor está rodando esteja conectado ao Wi-Fi que o ArDrone fornece.

Para armazenar os caminhos que o Drone deve percorrer são utilizados os comandos fornecidos pela biblioteca ardrone-autonomy, como mostra a Listagem 15.

```
 mission.takeoff()
              .altitude(1)
              .forward(result.Missao_pontos[0].distancia)
              .land();
executar();
```

Listagem 15 – Método para armazenar as operações que o drone deve realizar

Primeiro adiciona-se o comando para o Drone decolar, com a função takeoff(), logo após, o comando que recebe a altitude média que o Drone deve permanecer, neste caso, foi definido 1 metro, então, com o comando forward(), definimos a distancia que o Drone deve se locomover para frente, o valor vem como parâmetro da url "/missao/viajar/:id" após isso, o comando land() faz com que o Drone pouse imediatamente. Ao entrar na função executar(), os comandos são enviados ao Drone, e executados cada um em sua ordem respectiva.

Por questões de segurança, foi implementado um método que faz o pouso imediato do Drone, independente de qual ação esteja sendo executada (Listagem 16).

```
app.route("/missao/viajar/parar")
    .get((req, res) => {
        console.log('Parada forçada.');
        mission.control().disable();
        mission.client().land();
        res.sendStatus(200);
    });
```

Listagem 16 – Método para pouso imediato do drone

Quando a url "/missao/viajar/parar" for acessada será enviado ao Drone, primeiro o comando para desabilitar os controles e parar as tarefas, logo após é enviado o comando para o Drone realizar o pouso.

Para que os dados sejam recebidos e recuperados do banco de dados é necessária a utilização dos métodos de uma API REST, ou seja: app.get(), normalmente utilizado para consulta e retorna algum conjunto de dados; app.post() que é semanticamente utilizado para cadastrar novos dados; app.put() que tem como função principal nas implementações REST de atualizador de alguns atributos; e app.delete() que pode ser utilizado para deletar dados.

Na primeira implementação, o método app.route("/missao").get() é utilizado para recuperar todas as missões cadastradas no banco de dados, com o objeto "Missao", instanciado logo acima, como um *model* do Sequelize (const Missao = app.db.models.Missao;) é possível utilizar as funções como FindAll, para retornar todos os registros dessa entidade no banco de dados, FindOne, para retornar um objeto, passado o identificador como parâmetro, Create, para criar um novo objeto, Update para atualizar e Delete para excluir. Nesse momento, foi utilizado a função FindAll apresentada na Listagem 17.

```
Missao.findAll({
        include: [{
          model: Missao_pontos,
          where: { state: Sequelize.col('missao.id') },
          include: [{
            model: Pontos,
            where: { state: Sequelize.col('missao_pontos.ponto_id') }
          }],
        }]
      })
```

Listagem 17 – Método para retornar todos os registros da tabela missão

Na função findAll() estão sendo vinculados todos os registros dependentes de todas as Missões, ou seja, é realizada uma busca de todos os registros no banco de dados, com todas as suas ligações. Esse procedimento é realizado passando o nome do *model* e depois sua cláusula de relacionamento.

O método para salvar está reaproveitando a url "/missao", mudando apenas o método de .get para .post (Listagem 18).

```
.post((req, res) =>{
      Missao.create(req.body)
        .then(result => res.json(result))
        .catch(error => {
          res.status(412).json({msg: error.message});
        });
    });
```

Listagem 18 – Método para registrar uma nova missão no banco de dados

Nesse método, o Sequelize já faz uma limpeza dos parâmetros que não fazem parte do modelo. Caso o req.body contenha diversos atributos que não foram definidos para o modelo, eles serão descartados na hora da inserção da tarefa. Após a tentativa de incluir o registro no banco de dados, o resultado é analisado, se for positivo, retorna o próprio objeto salvo com o status 200, se não, será retornado a mensagem de erro junto com o *status* 412.

5 CONCLUSÃO

Por ser um conteúdo diferente do abrangido na maior parte do curso, foi necessário realizar muito estudo para definir o escopo e o desenvolvimento do trabalho. Foram realizadas leituras de manuais, trabalhos já desenvolvidos com Drones e diversos livros das principais linguagens e *frameworks* utilizados no projeto, como Node.JS e Angular.JS.

A principal mudança no processo de modelagem e desenvolvimento do aplicativo foi em relação à utilização da linguagem Node.JS ao invés da linguagem Java. Isso ocorreu, devido à existência de bibliotecas para auxiliar no controle do ArDrone, causando assim, maior facilidade na implementação das rotinas. O uso de JavaScript como linguagem principal também foi fundamental para o desenvolvimento do trabalho, pois é uma linguagem que facilita o desenvolvimento ágil para *web*, mesmo havendo dificuldades durante o desenvolvimento da aplicação, a comunidade desta linguagem é grande e costuma compartilhar soluções.

No início do projeto, foi cogitada a utilização do microcomputador Raspberry PI, para que, por meio dele, fossem enviados comandos ao sistema operacional embarcado no Ardrone, utilizando a própria biblioteca *ardrone-autonomy*. com o auxílio de GPS. O objetivo era que o Raspberry PI recebesse as coordenadas por meio de um servidor REST, então o Raspberry PI seria responsável por receber a posição atual por meio do GPS integrado neste microcomputador, verificar o ponto recebido pelo servidor REST, comparar os pontos, rotacionar o drone, enviá-lo até o ponto e por fim repetir este processo até que a missão esteja finalizada, enviando os comandos ao drone utilizando a biblioteca especifica para isto. Contudo, o meio de conexão encontrado entre o drone e o Raspberry PI foi o protocolo de rede Telnet, que se mostrou muito instável, sendo que hora a conexão era estabelecida com sucesso, hora não. Devido a isso, os testes utilizando o microcomputador foram excluídos do propósito do trabalho.

Como foi utilizado georreferenciamento para calcular a distância a ser percorrida, o estudo sobre a geodésia também se fez necessário. Apesar do uso de um microcontrolador não ser necessário nesse momento, ainda existe a possibilidade de utilizá-lo em uma implementação futura. Também foi tentado utilizar a conexão via *telnet* com o linux embarcado, quando o drone estava conectado via wifi para uma possível utilização de sensores não nativos, porém, a conexão se mostrou muito instável. O uso da

bússola para melhor orientação e direcionamento também foi descartado, devido à documentação não mostrar como obter este valor,

O funcionamento foi um assunto muito discutido durante a elaboração do trabalho. A divisão da implementação das aplicações também foi bastante discutida. As variáveis que foram consideradas para cada operação, também foram estudadas para o melhor desempenho possível da aplicação. Durante os testes foram levados em conta os ambientes para realização do vôo, para garantir maior precisão, se fez necessário um ambiente indoor, sem interferência do vento. Para fazer o levantamento dos requisitos das operações foram levadas em conta as APIs de Drone e de mapas abertos já existentes.

REFERÊNCIAS

ADIGBLI, Patrick. **Nonlinear attitude and position control of a micro quadrotor using sliding mode and backstepping techniques**. 3rd US-European Competition and Workshop on Micro Air Vehicle Systems (MAV07) & European Micro Air Vehicle Conference and Flight Competition (EMAV2007), Citeseer, 2007, p. 1-9.

ALMEIDA, Flávio. MEAN Full Stack JavaScript para aplicações web com mongodb express angular e nodeJS. Casa do Código. SérieCAELUM. 2011.

ALVES, Ana Sophia Cavalcanti. **Estudo e aplicação de técnicas de controle embarcadas para estabilização de voo de quadricopteros**. Tese (doutorado) em Engenharia Elétrica, da Faculdade de Engenharia da Universidade Federal de Juiz de Fora (2012). Disponível em: <http://www.ufjf.br/ppee/files/2013/03/Tese_Ana_Sophia.pdf>. Acesso em: 08 ago. 2015.

BASTOS, Teresa Raquel. **15 usos de drones na agricultura e na pecuária.** Disponível em: <http://revistagloborural.globo.com/Noticias/Pesquisa-e-Tecnologia/noticia/2015/05/15-usos-de-drones-na-agricultura-e-na-pecuaria.html> Acesso em: 08 set. 2015

BENINCASA, Felipe; CAMARGO, Marco Antonio; OKAMOTO JUNIOR, Jun. **Desenvolvimento de um quadricóptero autônomo com controle dinâmico de estabilidade**. Disponível em: <http://www.pmr.poli.usp.br/sites/pmr.poli.usp.br.euniversidade.com.br/files/03.%20Artigo.pdf>. Acesso em: 20 out. 2015. 2011.

COZA, Cosmin; MACNAB, Chris. J. B. **A new robust adaptive-fuzzy control method applied to quadrotor helicopter stabilization**. In: Annual meeting of the North American Fuzzy Information Processing Society (NAFIPS 2006), IEEE, 2006, p. 454-458.

GOMES, Raif C.; AQUINO, Franscisco José A. de. **Simulação de voo vertical de um quadricoptero usando software livre.** 2015. p. 1-4. Disponível em: <http://www.infobrasil.inf.br/userfiles/OK-Simulacao-122309.pdf>. Acesso em: 23 out. 2015.

GÜÇLÜ, Anil. **Attitude and altitude control of an outdoor quadrotor**. Dissertação de mestrado, 2012.

LUGO, Jacobo Jiménez; ZELL, Andreas. **Framework for autonomous onboard navigation with the AR.Drone**. 2013 International Conference on Unmanned Aircraft Systems (ICUAS). May 28-31, 2013, Grand Hyatt Atlanta, Atlanta, GA, p. 575-583

PANCERI, João Antônio Campos; MAIA, Gustavo; AMARAL, Rogério Passos Pereira; CUADROS, Marco Antônio de Souza Leite**. Controle fuzzy aplicado a estabilização de um eixo de um quadricoptero com a utilização do Labview.** XLI Congresso Brasileiro de Educação em Engenharia. 2013, p. 1-11

PAW, Yew Chai; BALAS, Gary J. **Development and application of an integrated framework for small UAV flight control development**. Mechatronics 21 (2011) 789–802.

PEREIRA, Caio R. **Construindo APIs REST com Node.JS**. Casa do Código. SérieCAELUM. 2016.

RONCOLATO, Murilo. **Mercado de drones cresce sem lei no Brasil e indústria nacional fica para trás**. Diponível em: <http://blogs.estadao.com.br/link/mercado-de-drones-cresce-sem-lei-no-pais-e-industria-nacional-fica-para-tras/>. Acesso em: 10 set. 2015.

SANTOS, Matilde; LÓPEZ, Victoria; MORATA, Franciso. **Intelligent fuzzy controller of a quadrotor**. IEEE, 2010, p. 141-146.

SILVA FILHO, Gerson Luis; RUDIGER, Gustavo Teixeira; NASCIMENTO, Jonathan Pontes Morais do. **Quadricoptero**. Universidade Tecnológica Federal do Paraná Engenharia de Computação. Curitiba, 2011. Disponível em: <http://paginapessoal.utfpr.edu.br/msergio/portuguese/ensino-de-fisica/oficina-de-integracao-ii/oficina-de-integracao-ii/Monog-11-2-Quadricoptero.pdf>. Acesso em: 18 ou. 2015.

SOUTHWORTH, Matt. **Drones**. Friends Committee on National Legislation. 2012. Disponível em: <http:/ /www.fcnl.org>. Acesso em: 12 out. 2015.