Universidade Federal Rural de Pernambuco - UFRPE
Departamento de Estatística e Informática - DEINFO
Programa de Pós-Graduação em Informática Aplicada - PPGIA

HealthDrones - Navegação de VANTs Autônomos Baseada em Autômatos Celulares

Paulo César Florentino Marques

Recife

Agosto de 2017

Paulo César Florentino Marques

HealthDrones - Navegação de VANTs Autônomos Baseada em Autômatos Celulares

Orientador: Prof. Dr. Jones Oliveira de Albuquerque
Coorientador: Prof. Dr. Hernande Pereira Silva

Dissertação de mestrado apresentada ao Curso de Pós-Graduação em Informática Aplicada da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do grau de Mestre em Ciências da Computação.

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES

Recife

Agosto de 2017

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema Integrado de Bibliotecas da UFRPE
Nome da Biblioteca, Recife-PE, Brasil

L732p Marques, Paulo César Florentino
HealthDrones - Navegação de VANTs autônomos baseada em autômatos celulares / Paulo César Florentino Marques. – 2017.
80 f. : il.

Orientador: Jones Oliveira de Albuquerque.
Coorientador: Hernande Pereira Silva.
Dissertação (Mestrado) – Universidade Federal Rural de Pernambuco, Programa de Pós-Graduação em Informática Aplicada, Recife, BR-PE, 2017.
Inclui referências, anexo(s) e apêndice(s).

1. VANT 2. Drone 3. Monitoramento ambiental 4. Autômatos I. Albuquerque, Jones Oliveira de, orient. II. Silva, Hernande Pereira, coorient. III. Título

CDD 004

Universidade Federal Rural de Pernambuco

Departamento de Estatística e Informática

Programa de Pós-Graduação em Informática Aplicada

HealthDrones - Navegação de VANTs Autônomos Baseada em Autômatos Celulares

Paulo César Florentino Marques

Dissertação julgada adequada para obtenção do título de Mestre em Ciências da Computação, defendida e aprovada por unanimidade em 09/08/2017 pela Banca Examinadora.

Banca Examinadora:

Prof. Dr. Jones Oliveira de Albuquerque
(Orientador)
Universidade Federal Rural de Pernambuco

Prof. Dr. Alexandre Cabral Mota
Universidade Federal de Pernambuco

Prof. Dr. Carlos Henrique M. Castelletti
Universidade Federal de Pernambuco

Prof. Dr. Cícero Garrozi
Universidade Federal Rural de Pernambuco

Agradecimentos

Antes de qualquer coisa, não posso deixar de agradecer por toda a base moral, educação e todo amor proveniente de minhas Irmãs e Mãe, que sempre me apoiaram, mesmo sem entender o que tanto eu estudava durante noites e mais noites.

Agradeço a meu orientador, Prof. Jones Albuquerque, pela paciência de sempre e por todo entusiasmo e conhecimento que tentou me passar a cada conversa. Também a todo o grupo HealthDrones e LIKA. Aproveito para agradecer também a todos os que fazem parte do DEINFO, professores, funcionários e alunos. Também ao GEOSERE na pessoa do Prof. Hernande, sempre disposto a auxiliar no que fosse necessário. Agradeço também a CAPES e INES pelo apoio financeiro no projeto.

Por fim, gostaria de agradecer aos meus amigos de curso, sempre incentivando uns aos outros. Em ordem alfabética: Amarildo, Ameliara, Arthur, Carlos (Filhinho), Edgard, Elton, Hallyson, Joel, Josival, Micaias, Márcio, Pedro e William. Aos amigos pessoais que me incentivam e inspiram sempre: Jacqueline, Ramires, Ingredy, Élida, Danielle, Marcelo (Pulga), Danilo, Leandro, Marcílio, Rostan, Diego (Thuran), Fabrício e Lucas. Enfim, a todos os que de forma direta ou indireta contribuíram. Também agradeço pelos momentos de descontração e grandes amizades que fiz, ao fazer parte do grupo Legs, e que vou levar para toda minha vida, Michele, Renata, Marina, Rayssa, Abelardo, Davy e Leguito. Enfim, a todos que fazem parte dessa equipe que é praticamente uma família.

Um especialista é alguém que sabe quais os piores erros que podem ser cometidos na sua área e os evita.

Werner Heisenberg

Resumo

Instituído pela Lei 9985/2000, o Sistema Nacional de Unidades de Conservação regulamenta a gestão e fiscalização de recursos naturais em áreas de preservação ambiental. Porém, a grande quantidade de Unidades de Conservação em diferentes localidades e com diferentes extensões territoriais torna frágil sua fiscalização quanto aos crimes ambientais. Dessa forma a utilização de tecnologias capazes de auxiliar no processo de fiscalização é de grande contribuição. Neste trabalho investigam-se meios de auxilio nesse processo por meio de uma análise detalhada de veículos aéreos não tripulados autônomos, sendo nosso objetivo a definição e teste de modelo computacional capaz de realizar voos e manobras destinados a vistoria de uma área qualquer. Foi analisada a possibilidade de utilização desses veículos de forma autônoma e controlada. Foram realizados experimentos com a finalidade de medir a precisão de manobras de voo e estes se mostraram compatíveis com o que o fabricante dispõe a entregar no que diz respeito a software de controle do veículo. A critério de comparação, foram realizadas manobras iguais, tanto utilizando o software de controle remoto do fabricante quanto com scripts desenvolvidos utilizando API Node.js. Ambas as soluções produziram manobras que se mantiveram equivalentes em média com um intervalo de confiança de 95%. Além disso foi implementado um ambiente de simulação e um autômato celular capaz de gerar caminhos e executar as ações simuladas no veículo. Os scripts gerados pelo autômato celular tem o mesmo padrão de desenvolvimento do script que foi desenvolvido para os experimentos. Logo, o autômato celular foi capaz de gerar scripts de manobras equivalentes as realizadas através do Software do fabricante. Além disso, são apontadas soluções para tornar o veículo analisado autônomo, assim como, problemas encontrados no drone.

Palavras-chave: VANT, Drone, Monitoramento ambiental, Autômatos Celulares, Navegação Autônoma.

Abstract

Instituted by Law 9985/2000, the National System of Conservation Units regulates management and oversight of natural resources in environmental preservation areas. However, the large amount of Conservation Units in different locations with a wide range of territorial extension weakens its oversight regarding enviromental crimes. As such, using techonologies capable of assisting in the process of inspection can be of great help. In this work we investigate ways to assist this process by means of a detailed analysis of autonomous unmanned aerial vehicles (UAVs). This work's objective is to define and validate a computational model capable of performing manouvers and flights to inspect a certain area. The possibility of using these UAVs in an autonomous and controlled manner was analyzed. Experiments were performed to measure automated flight manouvers accuracy, and the results obtained were compatible with what the manufacturer atests to deliver when it comes to the remote control software of the UAV. The same maneuvers were tested using the maker's remote control software and the automated scripts developed using a Node.js API, with mostly identical average results, with a confidence interval of 95%. Besides, a simulation environment was implemented along with a cellular automaton capable of generating pathways and executing the vehicle's simulated actions. The scripts generated by the automaton follow the same developing patterns of the scripts in the first experiments, showing that it is statistically compatible with the maker's software. Finally, solutions to make the studied UAV more autonomous for environmental preservation areas inspection are proposed, and problems with selected UAV are pointed. .

Keywords: UAV, Drone, Environmental monitoring, Celullar Automata, Environmental monitoring.

Sumário

Lista de Abreviaturas

ABS	*Acrylonitrile Butadiene Styrene*
API	*Application Programming Interface*
CA	*Cellular Automata*
CIPOMA	*Companhia Independente de Policiamento do Meio Ambiente*
CPRH	*Agência Estadual de Meio Ambiente*
DAO	*Data Access Object*
EPP	*Expanded Polyolefin*
GNSS	*Global Navigation Satellite Systems*
GPRS	*General Packet Radio Services*
GPS	*Global Positioning System*
GSD	*Ground Sample Distance*
MIMO	*Multiple Input/Multiple Output*
MVC	*Model, View, Controller*
SDK	*Software Development Kit*
SNUC	*Sistema Nacional de Unidades de Conservação*
UC	*Unidade de Conservação*
VANT	*Veículo Aéreo Não Tripulado*
VO	*Value Object*

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

Neste capítulo é apresentada a motivação deste projeto, analisando os principais pontos no que se refere à navegação de robôs móveis e sua aplicação no monitoramento ambiental. Na Seção 1.1 expõe-se brevemente uma introdução ao assunto, apresentando conceitos sobre unidades de conservação e a necessidade do monitoramento nessas áreas. Na Seção 1.2 é apresentada a delimitação do Tema. Na Seção 1.3 é abordado o problema de pesquisa. As Seções 1.4, 1.5 e 1.6 apresentam, respectivamente, o objetivo deste trabalho, justificativa e descrição do trabalho.

1.1 Introdução

O SNUC – Sistema Nacional de Unidades de Conservação, instituído pela Lei 9985/2000, regulamenta critérios para criação, implantação e gestão das 2071 Unidades de Conservação ambiental (UCs) no Brasil, sendo elas de proteção integral (650) e de uso sustentável (1421). Essa Lei conceitua UCs como sendo um espaço territorial provedor de recursos ambientais, incluindo águas jurisdicionais, com características naturais relevantes, legalmente instituídos pelo Poder Público, com objetivos de conservação e limites definidos. Sob essas UCs se aplicam garantias adequadas de proteção previstas na Lei supracitada (MACIEL, 2012). Os recursos naturais dessas áreas estão sob o domínio da União, dos Estados, dos Municípios, do Distrito Federal ou das entidades da administração indireta.

O Manejo Florestal sustentável, ou seja, administração da floresta para a obtenção de benefícios econômicos, sociais e ambientais, respeitando-se os mecanismos de sustentação do ecossistema objeto do manejo e considerando-se, cumulativa ou alternativamente, a utilização de múltiplas espécies madeireiras, de múltiplos produtos e subprodutos não madeireiros, bem como a utilização de outros bens e serviços de natureza florestal, é realizado sob três principais formas de operação de Florestas Públicas:

1. Concessão florestal: acordo financeiro, feita pelo poder concedente, Ministério do Meio Ambiente (no âmbito Federal), que concede direito de praticar manejo florestal sustentável para exploração de produtos e serviços numa unidade de manejo, tal consentimento se dá, na maioria das vezes, mediante licitação;

2. Diretamente pelo Poder Público: esse tipo de operação está descrita no artigo 17 da Lei 9985/2000. Determina a faculdade ao Poder Público de exercer diretamente a gestão sobre as florestas, definindo o manejo e as atividades de exploração de serviços florestais;

3. Destinação às comunidades locais: são identificadas as florestas públicas que estejam ocupadas ou utilizadas por comunidades locais para sua destinação de forma não onerosa, através de concessão de uso, concedida pelo Poder Público.

Cada um desses tipos de operações nas UCs deve ter monitoramento relacionado à utilização dos recursos ambientais providos pela unidade. Esse monitoramento é normalmente realizado por agentes de fiscalização ambiental e as polícias de competência da UC em questão. No entanto, os limites das Unidades de Conservação são, em sua maioria, de difícil acesso, muitas vezes sem nenhum tipo de comunicação efetiva e dificilmente são monitorados como deveriam ser, assim como a qualidade dos recursos existentes na área.

Dessa forma, pensando na delimitação dos limites e monitoramento dessas áreas, este trabalho visa o desenvolvimento e aplicação de modelos computacionais para navegação de robôs autônomos que serão utilizados no monitoramento de unidades de conservação percorrendo os limites da reserva sem a utilização de qualquer tipo de comunicação via satélite (GPS – Global Positioning System) ou rede de comunicação móvel. Para isso, serão aplicados métodos baseados em Autômatos Celulares (CA – Cellular Automata) na geração de rotas, as quais irão tornar o robô autônomo. Como validação foi implementada a geração de rotas aleatórias e executadas em ambiente controlado para calibração e análise de precisão de movimentos do VANT (Veículo Aéreo Não Tripulado) utilizado, aqui chamado simplesmente de drone.

1.2 Delimitação do Tema

Os temas que serão abordados neste trabalho será caminho aleatório na geração de rotas de navegação para robôs móveis aéreos. Para isso será proposta a utilização de CA na determinação de modelos computacionais capazes de simular a movimentação de robôs por uma rota bem delimitada em uma imagem aérea da área que será monitorada.

1.3 Problema de Pesquisa

Conhecendo as dificuldades existentes no monitoramento de áreas de conservação ambiental, tais como o deslocamento em áreas de difícil acesso e também dificuldades que englobam a falta de conexão com sistemas de posicionamento (GPS), o desafio desse projeto será investigar meios de monitorar áreas utilizando drones comerciais que sejam capazes de executar rotas autônomas geradas por modelos computacionais baseados em autômatos celulares. Modelos desse tipo são utilizados para estudo de propagação de doenças. Neste trabalho será considerado um modelo no qual o robô se locomova obedecendo regras de um CA no qual cada célula será um espaço percorrido de acordo com a escala da imagens de entrada.

Dessa forma, este trabalho tenta responder as seguintes perguntas de pesquisa:

- como tornar *drones* comerciais em agentes autônomos no monitoramento ambiental?
- Como o modelos computacionais baseados em autômatos celulares podem gerar rotas autônomas para os drones?
- Essas rotas podem ser executadas sem o uso de GPS ou GPRS?

1.4 Objetivos

Com este trabalho busca-se apresentar a definição de uma aplicação prática de modelos computacionais que possam ser utilizados no monitoramento ambiental de Unidades de Conservação. Estão descritos nas próximas subseções o objetivo geral e os específicos para definir esses modelos.

1.4.1 Objetivo Geral

É objetivo desta dissertação definir e testar modelo computacional para a geração de rotas de navegação de *drones* autônomos sem a necessidade de uso de GPS ou GPRS.

1.4.2 Objetivos Específicos

Para alcançar o objetivo geral, têm-se os seguintes objetivos específicos:

1. Desenvolver sistema de gerenciamento de simulações e missões;
2. Desenvolver autômato celular para navegação de robô autônomo;
3. Desenvolver templates para gerar programas de rotas autônomas;
4. Desenvolver o módulo de geração de código;
5. Promover experimento visando precisão de movimentos comparando software de controle do fabricante e códigos gerados.

1.5 Justificativa

É cada vez mais comum o uso de drones para realizar tarefas rotineiras ou em locais de difícil acesso, como exemplo de aplicações é visto a área de vigilância, vistoria em plantações e regiões de fronteiras. O uso massivo desses veículos se dá devido ao baixo custo e minimização de riscos de operações com veículos aéreos (Stochero, 2013).

Este projeto parte de uma problemática real a qual visa a melhoria no monitoramento e gestão de Unidades de Conservação. Com a aplicação pretende-se disponibilizar um sistema capaz de melhorar a gestão de recursos naturais dessas áreas, através do monitoramento por

meio de agentes autônomos de vigilância, fazendo uso de veículos aéreos não tripulados de uso comercial.

Além disso, será desenvolvido um método de navegação autônomo no qual pretende-se dispensar o uso de GPS ou qualquer outro tipo de comunicação relacionado a posicionamento do agente inteligente que esteja realizando o monitoramento. Isto se torna relevante pelo fato das Unidades de Conservação serem, na sua grande maioria, localizadas em zonas inóspitas e de difícil acesso. Assim é importante a utilização de agentes inteligentes autônomos que possuam algum tipo de localização independente de sua comunicação com sistema externo.

1.6 Descrição do Trabalho

Para deixar clara a concretização dos objetivos desta Dissertação, nesta seção são resumidas as características dos seus desenvolvimentos. Para o desenvolvimento do sistema de gerenciamento de simulações e missões dos veículos foi utilizado linguagem de programação PHP para backend (toda parte do servidor) e tecnologias de desenvolvimento WEB (HTML, CSS e JavaScript) para desenvolvimento do frontend.

Para a geração de código (a serem executados nos veículos) foi feito uso da linguagem de programação Python. Os códigos gerados a partir da simulação são enviados para a execução no veículo, caso o computador esteja conectado ao *drone*.

Por fim, para a validação foi realizada uma sequência de experimentos (simulação de execução em ambientes distintos) e analisados estatisticamente os dados de cada experimento.

1.7 Organização do Trabalho

Esta dissertação está organizada da seguinte forma:

- Capítulo 1 - aborda-se a motivação que inspirou a realização desta pesquisa, a delimitação do tema abordado, o problema de pesquisa, o objetivo geral e os objetivos específicos, uma justificativa, a descrição sobre o desenvolvimento dos objetivos, organização desta dissertação e uma descrição sobre os capítulos;
- Capítulo 2 - apresenta-se a fundamentação teórica, ou seja todas as teorias e fundamentos que este trabalho se baseia e que foram necessárias para a realização do mesmo;
- Capítulo 3 - trata-se de métodos utilizados no trabalho e trabalhos relacionados;
- Capítulo 4 - apresenta-se a implementação do sistema como um todo e resultados obtidos em experimentos;
- Capítulo 5 - trata-se de uma conclusão e proposta para trabalhos futuros; e
- Ao final são apresentadas as referências bibliográficas, seguidas de apêndice e anexos que complementam a dissertação.

Capítulo 2

Fundamentação Teórica

Neste capítulo é apresentada uma explanação sobre os conceitos utilizados como fundamentação teórica para a aplicação. Devido à natureza multidisciplinar deste trabalho é necessário definir os conceitos utilizados como base. O capítulo está dividido em quatro seções: a primeira apresenta conceitos sobre Robótica e Navegação de robôs de modo geral, a segunda uma analogia de grafos e sua utilização na navegação, em seguida conceitos sobre Autômatos Celulares e sua utilização em navegação e por fim, uma discussão sobre Drones e suas aplicações.

2.1 Navegação e suas Aplicações na Robótica

2.1.1 Robótica Móvel

A robótica é uma linha de pesquisa multidisciplinar. Os estudos relacionados à locomoção de robôs ou o desenvolvimento de robôs capazes de se locomover em qualquer meio, é denominada robótica móvel. Os meios nos quais os robôs podem se locomover são: solo, água, ar dentre outros.

2.1.2 Navegação em Robótica

Navegação consiste em todo o processo que envolve o deslocamento de um robô móvel em um ambiente de forma adequada, ou seja, é a navegação que tem a responsabilidade de direcionar para onde, quando e com qual intensidade o robô deve se locomover no ambiente, fazendo com que evite colisões e se mantenha em uma trajetória adequada de acordo com a tarefa a ser realizada.

Três processos são fundamentais para a navegação, são eles: localização, planejamento de trajetórias e aquisição de dados relativos ao ambiente para reconstrução posterior (Meyer e Filliat, 2003). A localização refere-se à percepção do veículo com relação a sua posição no mapa do ambiente considerando algum referencial; planejamento de trajetórias é a antecipação do caminho que o veículo irá percorrer, esse trajeto é relativo à posição inicial do veículo; e, por fim, a aquisição de dados do ambiente realizada durante a trajetória

percorrida para posterior reconstrução virtual.

Para que esses processos fundamentais aconteçam de forma adequada é necessário que o robô realize outras diversas tarefas definidas onde há diferentes etapas que compõem a navegação de robôs. Barrera (2010) define que essas atividades estão inter-relacionadas e são compreendidas por:

- **Percepção:** capacidade de coletar dados do ambiente por meio de sensores e obter informações a partir dos dados;
- **Exploração:** de acordo com a tarefa a ser realizada, tomando como base a percepção, o veículo é orientado para onde irá seguir;
- **Mapeamento:** com dados coletados de sensores, esta atividade envolve a representação espacial do ambiente o qual o robô móvel está inserido e que o mesmo tem percepção;
- **Localização:** tarefa na qual se estima a posição do veículo de acordo com o sistema de referencia;
- **Planejamento de Trajetória:** melhor trajetória possível para a tarefa que deseja realizar;
- **Execução da Trajetória:** atuação dentro do ambiente que fazem com que o veículo mude de posição e siga a trajetória da tarefa anterior.

2.1.3 Dificuldades da Navegação de Veículos Autônomos

Através da navegação e das tarefas que lhe cabem é que o robô (software do robô) terá a rotina de execução. Isso executado de acordo com a aplicação no ambiente em que seja designado a realizar a tarefa. Existem problemas complexos na navegação e que merecem atenção durante a implementação de sistemas relacionados a esse tema. Cerqueira (2012) apresenta vários desses problemas relacionados à navegação de robôs móveis. São eles:

Modificação no Ambiente:

A variação de ambiente é um problema caótico, mesmo que exista a possibilidade de caracterizar a movimentação de determinados objetos. Isso não é regra para o ambiente real. Em se tratando de um ambiente aéreo, por exemplo, não se pode prever se haverá colisões com aves em voo, ou qualquer outro objeto que esteja suspenso no ar. Isto é imprevisível.

Os sistemas de navegação devem estar preparados para esses tipos de acontecimentos. No entanto, para drones, o que se pode fazer é criar sistemas estabilizadores que façam com que o veículo trate as colisões, ou qualquer outro tipo de forças externas que tentem tirá-lo de sua trajetória (tais como ventos em direções diferentes do movimento). Estes podem ser tratados como sendo acontecimentos previstos e mantenham sua posição - (de acordo com a intensidade da força recebida o veículo pode ter reações que pode ocasionar até o desligamento dos atuadores, em casos de emergência). Alguns dos fatores podem ser previstos, tais como direção, sentido e velocidade de ventos, além de condições meteorológicas que podem afetar o ambiente e na dinâmica dos movimentos do drone.

Representação do Ambiente:

É indispensável que exista uma representação do ambiente no qual o veículo irá executar uma tarefa. No entanto, a atualização constante da representação desse ambiente depende significativamente de alguns fatores críticos no processo:

- **Sensores:** a utilização de sensores adequados ao tipo de tarefa a ser realizada é de extrema importância. Sensores que são capazes de obter dados como cor, distância ou outros, podem não ser eficientes para identificar obstáculos, por exemplo. Esse fator gera incerteza quanto a validade dos dados coletados;
- **Localização (Erro):** a localização do veículo é de fundamental relevância na navegação do mesmo, no entanto sistemas inerciais, ou odometria, acumulam erro ao longo do tempo. Quanto mais tempo o veículo se locomove, mais erro associado à sua posição existe e, consequentemente, torna-se cada vez mais imprecisa a sua localização;
- **Dinâmica do ambiente:** em ambiente real é difícil prever o que irá acontecer no momento da execução da tarefa, essa incerteza faz com que seja muito difícil a construção de modelos exatos com prevenção de eventos;
- **Tomada de decisão:** em tempo de execução da tarefa pode acontecer eventos imprevisíveis que exijam que seja processada uma grande quantidade de dados em alta velocidade. Isso exige processamento de dados e caso esse processamento seja embarcado, como na maioria das vezes, está passível de erros e limitações.

Levando em consideração esses fatores críticos, o ambiente pode ser representado por meio de dois modelos básicos: mapeamento métrico e mapeamento topológico (Meyer e Filliat , 2003). O mapeamento métrico apresenta o ambiente por meio de matriz de ocupação multidimensional: divide a área total em unidades menores e uniformes, da mesma maneira que são construídos os mundos celulares dos autômatos celulares, como apresentado na Figura 2.1. Cada unidade - também chamada de célula - possui uma probabilidade estimada que indica a possibilidade do espaço estar ocupado.

Existe, ainda, o mapeamento topológico, que é a representação do ambiente como uma coleção de pontos de referência. Esses pontos são conectados entre si. Tanto os pontos quantos as conexões podem ser modelados como distribuições de probabilidade para estimativa de posição do veículo. Para este trabalho foi utilizado o mapeamento métrico, ou seja, é dividido em células os espaços em que é possível navegar.

2.2 Navegação em Mapa Métrico Análoga com Grafos

2.2.1 Mudança de Estado da Células como uma Transição Fásica

A Teoria dos Grafos surgiu em 1735, conceituado por Leonhard Euler (1707-1783) em um trabalho publicado em 1736 na revista Comentarii Academiae Scientiarum Imperialis Petropolitanae, no volume 8 (Simões-Pereira, 2013). Conceitualmente, um Grafo é um sistema

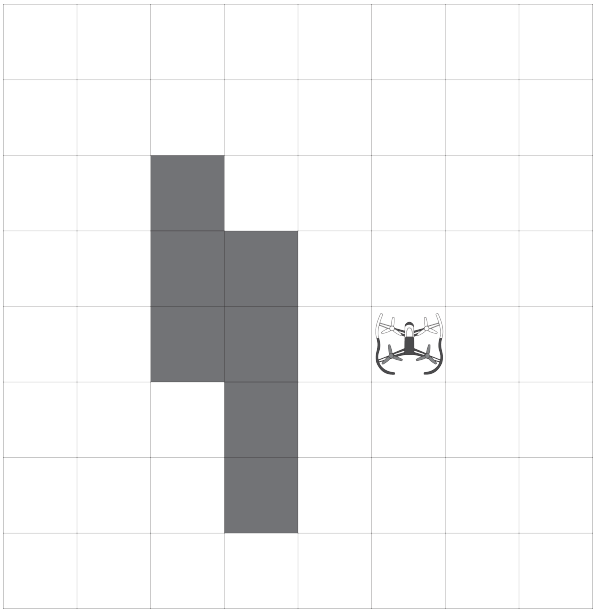

Figura 2.1: *Representação de divisão de mapa de ambiente, divisão por células; células escuras representa bloqueio, ou célula impedida de navegação - Fonte: elaborado pelo autor*

formado por um conjunto de pontos, denominados vértices, representados por V e um conjunto de arestas, que conecta os vértices do sistema, representado por E. Assim, um Grafo pode ser representado por uma estrutura como exposto na representação matemática:

$$G = (V, E)$$
$$V = \{1, 2, 3, 4\}, E = \{(1, 4), (2, 3), (2, 4), (3, 4)\}$$

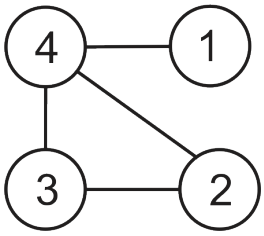

Figura 2.2: *Representação de um Grafo - Fonte: elaborado pelo autor*

Grafos podem ser direcionados, chamados de **Digrafos**. Estes possuem a mesma estrutura que o Grafo, no entanto o conjunto de arestas é um conjunto de pares ordenados, ou seja, mostra as direções de vértice origem e vértice destino.

$$D = (V, A)$$
$$V = \{1, 2, 3, 4\}, A = \{(1, 4), (2, 4), (2, 3), (3, 2), (4, 2), (4, 3)\}$$

Os Grafos, ou Digrafos, podem ser finitos ou infinitos. Para estruturas finitas significa que ambos os conjuntos, vértices e arestas, são conjuntos finitos. Para Grafos, ou Digrafos infinitos possuem pelo menos um dos conjuntos de vértices e arestas infinito (Simões-Pereira, 2013).

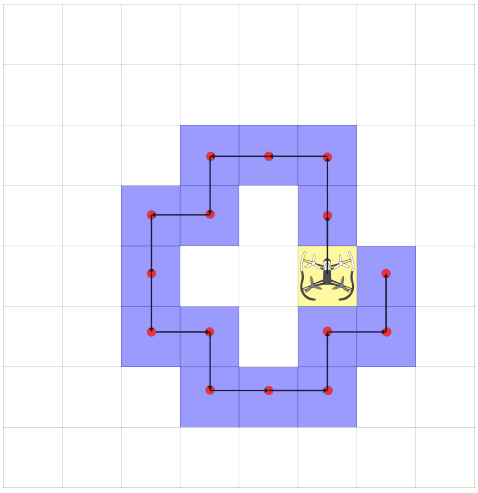

Figura 2.3: *Representação gráfica de um quadriculado no qual mostra parte finita de um grafo infinito. As setas indicam uma possível rota tendo origem na célula em destaque amarela. Estão representados 15 passos, onde cada passo representa a transição entre uma célula e outra. Os vértices estão representados pelo ponto vermelho e é o meio de cada célula - Fonte: elaborado pelo autor*

Matematicamente, um quadriculado é expresso como um Grafo, onde vértices em $\mathbb{Z}^2$ são formados por pares ordenados (i, j). A conexão entre uma célula (i, j) com qualquer outra se dá através de sua vizinhança com:

$$(i + 1, j), (i, j + 1), (i - 1, j), (i, j - 1)$$

Os algoritmos que realizam percursos em Grafos são apresentados neste trabalho como uma alternativa possível de implementação de regras. Dentre os métodos, é descrito, a seguir, o Caminho Hamiltoniano.

2.2.2 Caminho Hamiltoniano

Percurso, ou Caminho Hamiltoniano, é o trajeto que tem como objetivo passar por todos os vértices de um grafo sem visitar um vértice mais de uma vez. Caso haja ciclos (ciclo Hamiltoniano, no qual o trajeto é realizado em todos os vértices e volta ao ponto de origem) no grafo e o Caminho Hamiltoniano ainda seja possível esse grafo é chamado de Grafo Hamiltoniano.

O problema de determinar um Caminho Hamiltoniano para um grafo é um problema NP-Completo, ou seja, é bem provável que não exista algoritmo em tempo polinomial para o problema. No entanto, existem formas de otimizar e tratar o problema. Na Figura 2.4 é demonstrada a representação do Caminho Hamiltoniano.

A Teoria dos Grafos se faz importante para este trabalho para a determinação de regras da movimentação na simulação e na aplicação do sistema. Considerando como um ponto do grafo como sendo uma célula do mapeamento do ambiente, ou seja, a transição de uma célula para outra pode ser considerada como sendo a transição de um ponto a outro em um grafo.

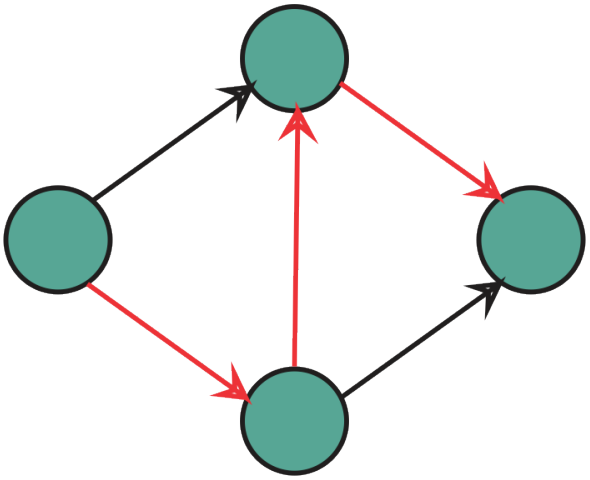

Figura 2.4: *Representação do grafo com o Caminho Hamiltoniano em vermelho - Fonte: elaborado pelo autor*

2.3 Conceitos de Autômatos Celulares

John Von Neumann, fazendo um elo entre biologia e a Teoria dos Autômatos, apresenta o conceito de autômato celular (NEUMANN e A.W., 1966). O conceito surgiu quando ele apresentava a seguinte questão: "Que tipo de organização lógica é suficiente para um autômato ser capaz de reproduzir a si próprio?". Com isso era estudado o fenômeno biológico da auto-reprodução.

Além do conceito explicito em NEUMANN e A.W. (1966), Wolfram (1994) apresenta uma variação desse conceito. Wolfram expõe Autômatos Celulares, que são modelos matemáticos simples, construídos a partir de um "mundo celular", também chamado de malha, ou reticulado, de células iguais entre si e discretas. Cada uma dessas células tem um estado representado por um valor de um conjunto finito. Os estados das células, ou seja, seus valores, são atualizados em passos discretos seguindo regras matemáticas que definem o valor de cada célula de acordo com os valores das células vizinhas.

Existem diversos tipos de vizinhanças, dentre elas as bidimensionais, como mostrado na Figura 2.5. Destas, destacam-se duas: a vizinhança de Von Neumann e a vizinhança de Moore. Esses tipos de vizinhanças se diferenciam pela quantidade de células consideradas vizinhas. Enquanto a vizinhança de Moore considera 8 células vizinhas, a de Von Neumann, 4 células vizinhas.

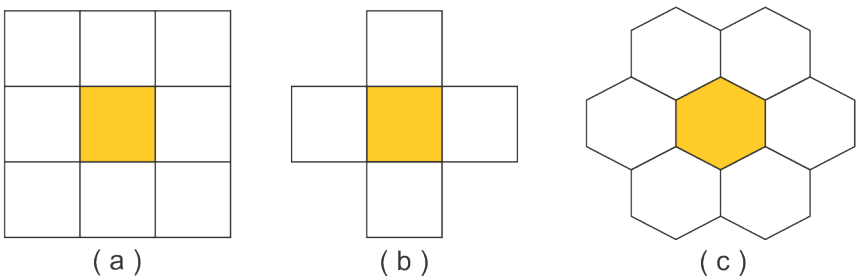

Figura 2.5: *Modelos de vizinhança. a) Representação da vizinhança de Moore, 8 células vizinhas; b) Representação da vizinhança de Von Neumann, 4 células vizinhas; c) Representação da vizinhança hexagonal, com 6 células vizinhas - Fonte: elaborado pelo autor*

Wolfram (1994) afirma que os Autômatos Celulares podem ser definidos como modelos matemáticos discretos baseados em equações diferenciais parciais que são muitas vezes utilizadas para simular sistemas naturais. Pelo fato desses autômatos celulares serem discretos, pode ser realizada uma analogia com computadores digitais, onde os autômatos celulares podem ser vistos como computadores de processamento paralelo. Segundo Ilachinski (2001), os CA-*Cellular Automata* possuem características genéricas que são:

- **Células discretas:** qualquer sistema CA consiste em uma, duas ou três dimensões;
- **Homogeneidade:** todas as células são iguais;
- **Estados discretos:** cada célula assume somente um estado de um conjunto finito de estados discretos;
- **Interações locais:** cada célula interage somente com sua vizinhança local;
- **Dinâmica discreta:** a atualização do sistema se dá em tempos discretos de acordo com as regras de transição de estado para cada célula e sua vizinhança.

Um exemplo de autômato celular unidimensional é um vetor de células as quais assumem valores binários, ou seja, 0 ou 1, onde a_i^t é uma célula na posição i, no tempo t. O estado dessa célula pode ser definido por uma regra simples a cada passo de tempo, chamada de regra F:

$$a_i^{(t)} = F(a_{i-r}^{(t-1)}, a_{i-r+1}^{(t-1)}, \cdots, a_{i+r-1}^{(t-1)}, a_{i+r}^{(t-1)})$$

Cada célula pode assumir um dos possíveis k valores de estado, ou seja, $a_i^{(t)} \in \{0, 1, 2, ..., k\}$. A regra F define um valor para o próximo passo sendo atribuído $k^{(2r+1)}$ possibilidades de regras onde r define a vizinhança e k a quantidade de estados. Assim para $r = 1$ e $k = 2$ (a pode ser 0 ou 1), por exemplo, teremos $2^3 = 8$ possibilidades. Como exemplo prático temos a Figura 2.6 que mostra a evolução para 5 passos.

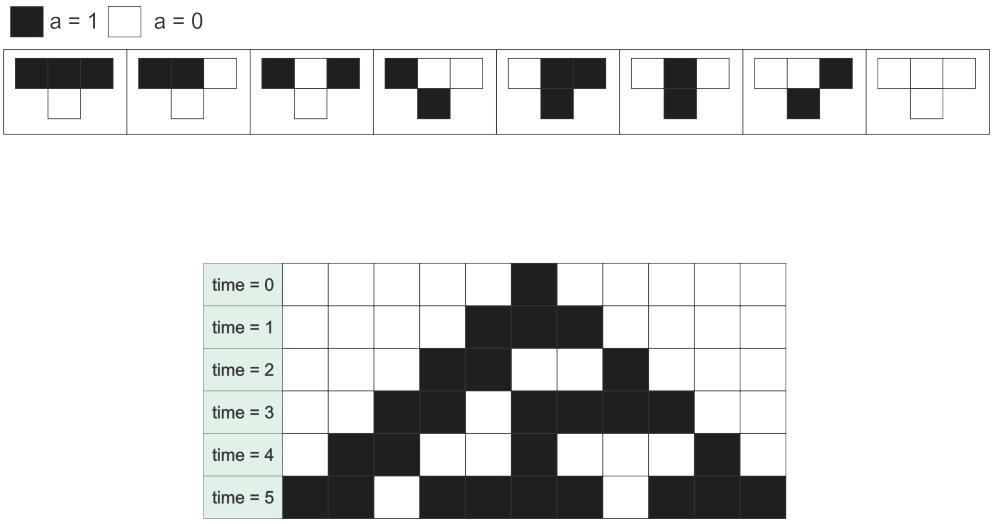

Figura 2.6: *Representação de um autômato celular unidimensional - Fonte: adaptado de (Ilachinski, 2001)*

Para este trabalho a teoria dos autômatos celulares servirá de base para simular e gerar a trajetória de navegação do agente autônomo (Drone) o qual será aplicado o modelo proposto.

2.4 Conceitos sobre Drones e suas Aplicações

É cada vez mais comum o uso de drones para realizar tarefas rotineiras ou de locais de difícil acesso. Como exemplo de aplicações, é visto na área de vigilância, em vistoria em plantações e regiões de fronteiras. O uso massivo desses veículos se dá devido ao baixo custo e minimização de riscos de operações com veículos aéreos (Stochero, 2013).

São diversas as aplicações de drones na atualidade. Microdrones (2017) apresentam alguns cenários em que são elencados tarefas que os drones podem executar, são elas:

- Aplicações de Âmbito Civil:
  - Segurança;
  - Monitoramento costeiro e de fronteiras;
  - Incêndio florestal;
  - Monitoramento de desastres naturais;
  - Monitoramento de trafego;
  - Operações anti-terrorismo;
  - Agricultura - monitoramento de plantações;
  - Monitoramento de eventos desportivos e sociais.
- Pesquisa Científica:
  - Pesquisas Arqueológicas;
  - Monitoramento Ambiental;
  - Pesquisas Oceanográficas;
  - Pesquisas Meteorológicas;
  - Monitoramento de Ambientes contaminados;
  - Monitoramento em Ambientes de difícil acesso;
  - Contagem de Animais e aplicações em biologia;
  - Monitoramento de crescimento florestal;
  - Monitoramento de pragas em plantações;
- Logística:
  - Transporte de cargas;
  - Entrega de encomendas;

Contudo, duas categorias de drones se destacam: os veículos de asas fixas e os de asas rotativas (hélices, em geral são usadas quatro e são também chamados de quadrotores). Cada categoria se adequa a uma determinada função, caso haja uma necessidade maior de estabilidade de voo, os drones de asas rotativas são os mais indicados. Para este trabalho foi utilizado um quadrotor, uma vez que necessita de uma boa estabilidade de voo para a obtenção de imagens nítidas o suficiente para a identificação dos elementos presentes nela e que atendesse a necessidade de monitoramento ambiental.

O primeiro quadrotor registrado na história foi o proposto pela Força Aérea Americana em 1920, proposto por Étienne Oehmichen, onde o seu segundo projeto apresentou melhor estabilidade. Com quatro eixos dispostos em estrutura cruzada com hélices de duas pás cada e controladas por um único motor. Foi o projeto que apresentou melhor estabilidade na época (Alvané, 2014). Na Figura 2.7 vemos a estrutura do Oehmichen n.2.

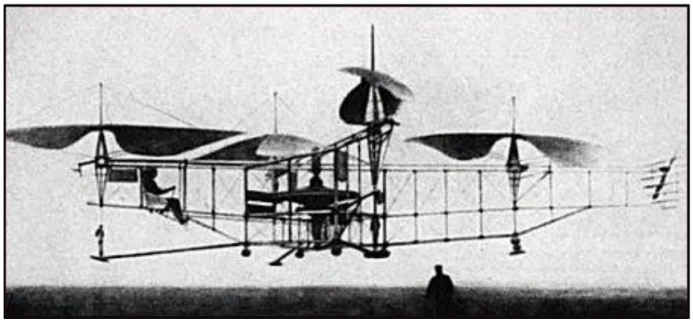

Figura 2.7: *Estrutura do Oehmichen n.2. Fonte: (Alvané, 2014)*

Atualmente os drones com asas rotativas são geralmente de pequeno porte com um motor para cada hélice e uma estrutura cruzada, onde cada motor é fixado em uma das extremidades da estrutura. É equipado com computador embarcado, responsável pelo controle do giro dos motores, bem como a execução dos algoritmos que estabilizam o voo, leitura de sensores, e aquisição e processamento de dados provenientes do ambiente em que estão sobrevoando.

2.4.1 Estrutura de Drones de Asas Rotativas

Existem diversas configurações de Drones. Com quatro ou mais motores, eles têm assumido uma das seguintes configurações como as apresentadas na Figura 2.8. Para cada atividade a ser realizada pode haver uma ou mais configurações adequadas para a tarefa. A configuração será relativa de acordo com a carga que se deseja carregar, o ambiente em que será realizado os voos, a precisão de movimentos desejada e a velocidade com que se deseja realizar os movimentos. Todos esses fatores influenciam na escolha da configuração desejada para o drone.

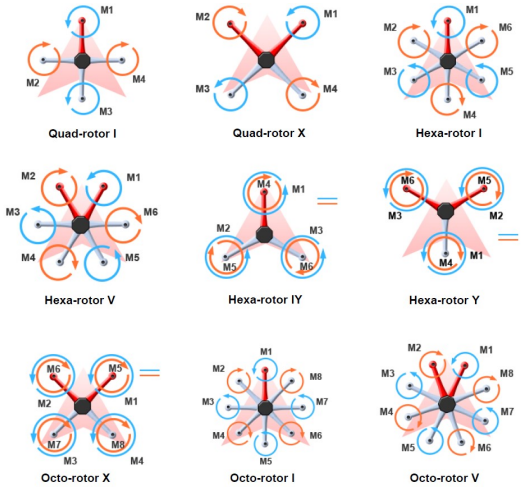

Figura 2.8: *Ilustrações de configurações de Drones de 3 ou mais motores - Fonte: https://blogdecis.wordpress.com/2014/02/28/un-dia-heliceando/*

Seguindo essas configurações de estruturas, nos últimos anos surgiram diversos drones comerciais capazes de realizar as mais diversificadas tarefas. Um desses drones, de quatro hélices, foi escolhido para este trabalho, o Parot Bebop Drone (Figura 2.9), com a configuração Quad-rotor X - de acordo com a Figura 2.8. A escolha de uso desse modelo para o projeto foi devido a seu baixo custo, quando comparado com outros do mesmo porte, e a suas funcionalidades, principalmente a disponibilidade de um SDK OpenSource, uma vez que o intuito era torná-lo autônomo. Essa característica era imprescindível para o projeto.

Figura 2.9: *Bebop Drone Parot - Fonte: https://www.parrot.com/fr/drones/parrot-bebop-drone*

Capítulo 3

Trabalhos Relacionados e seus Métodos

Neste capítulo são descritos os métodos utilizados no planejamento de trajetórias baseado em autômatos celulares. O capítulo está dividido em oito seções, distribuídas da seguinte forma: na primeira seção é apresentado uma visão geral dos trabalhos relacionados. Nas seções seguintes, cada uma, é apresentado um artigo ou método de abordagem da literatura para o problema.

3.1 Trabalhos Relacionados

O planejamento de trajetória baseado em autômatos celulares vem sendo estudado desde 1995 com o trabalho de Shu e Buxton (1995). Nos anos seguintes outros trabalhos surgiram, tais como Marchese (1996), Tzionas *et al.* (1997), Behring *et al.* (2000), Marchese (2002), Marchese (2008), Tavakoli *et al.* (2008), Marchese (2011) e Ferreira (2014) com um dos trabalhos mais recentes. Dentre os trabalhos analisados existem abordagens distintas, todas destinam-se a robôs móveis e somente algumas realizaram experimentos práticos. Foram analisados dez trabalhos, nove deles, com abordagens destinadas a implementação baseadas em autômatos celulares e um trabalho relacionado que também é utilizado como base para implementação da execução de rotas que foi o trabalho de Silva (2016).

Em seguida estão dispostas as análises dos trabalhos organizadas pelos diferentes métodos. Em seguida é realizada uma discussão com a finalidade de apresentar o método utilizado neste trabalho.

3.2 Parallel path planning on the distributed array processor (Shu e Buxton, 1995)

O primeiro trabalho relacionado a planejamento de trajetórias foi o trabalho de Shu e Buxton (1995). Este trabalho utiliza uma abordagem baseada em autômatos celulares e difusão por pontos de força com a finalidade de planejar o caminho mais curto sem colisões, partindo do ponto de origem para um ponto objetivo.

Shu e Buxton (1995) apresenta como sua principal contribuição a discretização do ambiente em que o robô pretende navegar. Em sua abordagem, o ambiente é dividido em células que são representadas pela posição de um *array* binário de dimensão relacionada à quantidade de graus de liberdade utilizada. Em tal *array*, cada posição é uma célula onde são atribuídos os valores 0 (zero) para células livres e 1 (um) para posições que estão ocupadas (obstáculos). O método de propagação é baseado em força de difusão, isto é, adotando a vizinhança de Von Neumann. Nele são analisadas as células vizinhas que estão livres, essas têm força de propagação. Uma célula obstruída não possui força em nenhuma direção.

As regras de propagação do autômato são baseadas nas forças atribuídas na movimentação para uma direção da seguinte forma: uma célula tem força de difusão igual a 1 em uma direção se na direção analisada estiver livre, ou se naquela direção no raio 2 a força for igual a 1, ou ainda, se as vizinhas nas direções não contrárias da vizinha tiverem força. Por exemplo: uma célula tem força para Norte quando:

- a força para a célula ao Norte for 1; ou,
- a força para a vizinhança de raio 2 ao Norte for 1; ou,
- a força das células a Nordeste e a Noroeste forem iguais a 1.

O término acontece em dois casos: i) em caso de recorrência no ponto de partida; e, ii) em caso de não existir mais espaço de propagação. O artigo conclui que o método é efetivo e eficiente através de simulações e não demonstra em robôs reais.

3.3 Modelos Baseados em Camadas (Marchese, 1996),(Marchese, 2002),(Marchese, 2008),(Marchese, 2011)

Marchese (1996) apresenta um método de planejamento de trajetória baseado em autômatos celulares utilizando múltiplas camadas. Assim, para a representação, são utilizadas várias camadas e o resultado é obtido com a execução encadeada em 5 etapas. Além da principal característica, que é a utilização de camadas, os robôs utilizados apresentam limitações em sua movimentação.

O trabalho aborda simulação em um ambiente plano e com obstáculos conhecidos anteriormente. O robô que se movimenta no ambiente possui a restrição de movimentação somente para a frente, assim são necessárias algumas informações a respeito dele para que seja definido uma orientação, ou seja, necessita de informações de um ângulo θ indicando sua direção relacionada a posição inicial.

A arquitetura baseada em camadas utiliza uma matriz para descrição de cada variável. Na definição da célula, são levadas em consideração 5 (cinco) camadas, ou seja, cinco variáveis definem a célula e cada camada é executada separadamente. São as camadas:

- Camada de Obstáculos: descreve se as células contém, ou não, obstáculo;
- Camada da Posição inicial: determina a posição inicial, ou seja qual a célula em que se encontra inicialmente o robô e qual a sua direção;

- Camada da Posição Objetivo: camada que descreve qual célula é o objetivo do robô e em qual direção o mesmo pode alcança-la;
- Camada de Atração para Objetivo: camada que contem a distância e uma das 8 direções até o objetivo levando em consideração a vizinhança de Moore; e,
- Camada de geração do Caminho: camada de saída do algoritmo, nela é definido um autômato não determinístico para cada célula;

A execução do algoritmo acontece em etapas e seguindo uma camada de cada vez, respectivamente: obstáculos, objetivo, posição inicial, atração para objetivo e camada de geração do caminho. Cada fase acontece de forma independente e encadeada, onde a principal fase é a geração do caminho.

A fase de atração para o objetivo acontece de forma a encontrar o caminho mais curto e não determinístico entre a célula de início e o objetivo. São atribuídos valores a cada célula de acordo com a distância entre cada uma delas e o objetivo. Partindo do objetivo é estabelecido a cada célula uma unidade de distância a cada passo até o objetivo, esse processo se repete até que seja atribuído esse valor na célula de início.

Nas fases seguinte e última, geração de caminho, são analisadas os caminhos de acordo com o cálculo de distância de cada célula e direção que o robô se encontra. São analisados os caminhos possíveis e em seguida escolhido um de menor distância, ou seja, a atração mais forte. Em caso de empate, é realizada uma escolha não-determinística.

Todos os resultados apresentados em (Marchese, 1996) são realizados em simulações e não foram realizados experimentos em robôs físicos.

Marchese (2002), mesmo autor do trabalho anterior, apresenta uma expansão da pesquisa, na qual é mantida a arquitetura em camadas. As 5 camadas são mantidas. O robô continua com três graus de liberdade (x, y, θ), sua posição no plano (x, y) e o ângulo de direção θ. A grande distinção entre os trabalhos acontece na fase de atração para o objetivo. Essa fase foi atualizada e incluída pesos a ela, o vetor de peso w tem a finalidade de ser utilizada para suprir a limitação do trabalho anterior, que seria o robô realizar a movimentação somente para frente. Agora, com a utilização desse vetor, é possível a utilização de robôs que se movimentam na direção escolhida. O vetor peso contém os valores de ir para: (frente, diagonal frente, mudança de direção, parada, rotação, trás, diagonal para trás). Quanto menor for o peso, mais fácil será de realizar o movimento. Assim, Marchese (2002) apresenta resultados em simulações e também acrescenta mais de um objetivo, contudo, sem o uso de robôs reais.

Marchese continua os trabalhos e publica em 2008 uma nova expansão para o projeto (Marchese, 2008). Nesse trabalho, o objetivo foi utilizar a arquitetura de múltiplas camadas com a aplicação de vários robôs no mesmo ambiente. É utilizado um robô como coordenador de movimentos, com a finalidade de não haver colisões entre os agentes no ambiente. Para isso, além da adaptação do robô coordenador, foi necessário realizar alguns ajustes nos atributos de algumas camadas. Duas das camadas são consideradas principais: a camada de obstáculos, agora com 3 (três) dimensões (x, y, t) onde t é tempo; e a camada de atração para o objetivo, agora com 5 dimensões, (x, y, θ) para descrever o espaço de configurações, além da dimensão tempo e robô.

O robô coordenador considera cada outro robô como sendo um obstáculo móvel e coor-

dena os movimentos de todos para que não haja colisões. A trajetória de cada robô é definida a partir de uma prioridade estabelecida para cada robô e é escolhido o caminho relativo para cada um deles. O trabalho também não realiza experimentos com robôs físicos.

Por fim, Marchese (2011) publica outro trabalho em 2011. Neste trabalho também trata o caminho para múltiplos robôs heterogêneos interagindo no mesmo ambiente. Da mesma forma que em (Marchese, 2008), os robôs possuem prioridades e consideram os demais robôs como obstáculos móveis. O principal objetivo foi melhor detalhar os processos e apresentar a prova da dualidade do problema de planejamento de rota. Assim, os resultados de calcular o trajeto do início ao objetivo será o mesmo de calcular o oposto, ou seja, partir do objetivo para o início. O trabalho também não apresenta realizações de experimentos com robôs reais.

3.4 Collision-free path planning for a diamond-shaped robot using two-dimensional cellular automata (Tzionas *et al.*, 1997)

Tzionas *et al.* (1997) apresenta uma abordagem distinta das vistas até agora mas também utilizando autômatos celulares no planejamento de trajetórias. Uma característica interessante deste método é que não se procura o menor caminho, o objetivo aqui foi encontrar o caminho mais distante dos obstáculos do ambiente.

A arquitetura do autômato celular utilizada é um reticulado bidimensional com vizinhança de Von Neumann. O algoritmo executa em duas fases, a primeira consiste na determinação de bordas e obstáculos no ambiente através da utilização de imagem binarizada a partir do ambiente discretizado. Os obstáculos são representados em qualquer formato. O robô utilizado tem formato de diamante e o algoritmo baseia-se na restrição do espaço livre no diagrama de Voronoi, determinado ao longo do tempo de evolução do autômato celular.

O Diagrama de Voronoi consiste em uma região $V(p)$ dos pontos do plano que estão mais próximos de p do que qualquer outro ponto do conjunto S de n pontos no ambiente, onde p é um par ordenado do tipo (x, y); Dessa forma, é realizado o diagrama de Voronoi para os pontos que são obstáculos no ambiente. O caminho pelo qual o robô passará será a borda do diagrama de Voronoi bastando apenas verificar se o tamanho do robô mais 1/2 de sua diagonal é menor que a célula do caminho. Ou seja, se é possível passar por aquela célula. Caso afirmativo, o robô percorrerá o caminho mais distante dos obstáculos do ambiente até o objetivo. Contudo, os autores não realizaram experimentos com robôs físicos, somente simulações.

3.5 An Algorithm for Robot Path Planning with Cellular Automata (Behring *et al.*, 2000)

Behring *et al.* (2000), em seu estudo sobre o planejamento de trajetórias para robôs móveis utilizando autômatos celulares, apresenta uma maneira simples de tratar a geração de rotas resultando no caminho mais curto e com uma abordagem simples. Neste trabalho os autores desprezam as condições físicas de dinâmica e cinemática, ou seja, considera-se que o robô

executará os movimentos perfeitos com precisão ótima.

O trabalho consiste na execução algumas etapas, sendo a primeira a abstração do ambiente em um espaço celular onde é dividido em células assumindo quatro possíveis estados: obstáculo, livre, início e objetivo. Uma aplicação de CA é o alargamento dos obstáculos com a finalidade de evitar colisões, uma vez que para a modelagem é desprezado a dinâmica e cinemática durante a execução. É utilizado a vizinhança de Moore e a regra é descrita em: se uma célula está no estado livre e possui pelo menos um dos seus vizinhos um obstáculo, então ela passará a ser obstáculo; em caso negativo ela permanecerá no estado em que se encontrava no ultimo passo de tempo. No trabalho foi executado 4 (quatro) passos desse apresentando um alargamento de 4 células.

O passo seguinte da execução é a aplicação de um CA na determinação da distância entre o objetivo e o ponto inicial. A regra de transição consiste na verificação da vizinhança com relação ao calculo de distância, ou seja, se a célula está livre e um vizinho já possui distância a distância dessa célula será a calculada para o seu vizinho mais 1. Caso contrário, mantém o estado da célula. A condição de parada é que seja calculado o valor da distância na célula de início, ou que não haja mais células para calcular a distância. Isso fará com que seja calculado o valor da distância entre o objetivo e o ponto inicial. A figura 3.1 mostra esse processo.

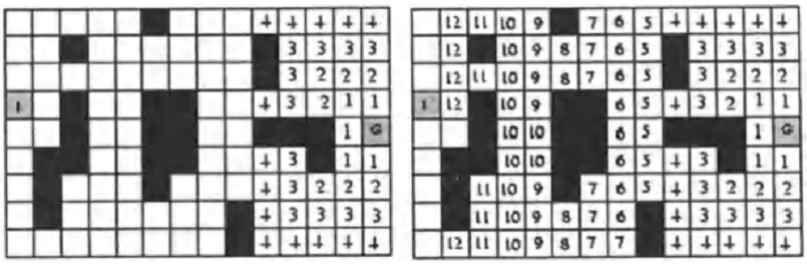

Figura 3.1: *Resultados do processo de determinação da distância com início no objetivo e com espalhamento até o ponto inicial - Fonte: Behring et al. (2000)*

O passo final é a escolha de um dos caminhos possíveis, o robô escolherá um dos caminhos que seja o menor entre eles, ao passo em que na execução irá diminuindo uma unidade de distância a cada passo em direção ao objetivo.

O experimento foi implementado em um robô real em uma mesa de testes seguindo especificações da RoboCup.

3.6 A Cellular Automata Based Algorithm for Path Planning in Multi-Agent Systems with A Common Goal (Tavakoli *et al.*, 2008)

Em 2008, Tavakoli *et al.* (2008), publica um trabalho baseado em (Behring *et al.*, 2000), no qual expande seu modelo apresentando a mesma abordagem no que se refere ao ambiente discretizado, regras e autômatos celulares. Nesse trabalho, Tavakoli *et al.* (2008) apresenta duas contribuições: a inserção de mais de um robô navegando no mesmo ambiente com um

objetivo em comum, e a utilização de custos de movimentação, uma vez que a movimentação na diagonal é mais custosa que movimentos em linha reta.

O controle dos multi-agentes pode ser realizado de duas formas: centralizada ou distribuída. Para o trabalho foi utilizado o controle centralizado, o qual apenas um controle é levado em consideração e especifica o caminho para todos os agentes móveis do ambiente.

3.7 Modelos Baseados em Autômatos Celulares para o Planejamento de Caminhos em Robôs Autônomos (Ferreira, 2014)

O trabalho de dissertação de Ferreira (2014) apresenta uma abordagem também baseada em autômatos celulares na qual, não diferente dos demais, pretende-se alcançar um objetivo. Inicialmente o trabalho é uma implementação baseada no trabalho de Behring *et al.* (2000), o qual foi simulado utilizando software de simulação de robótica, identificado problemas considerados críticos e, posteriormente, foi apresentado melhorias para os problemas identificados no método.

O primeiro problema encontrado, diz respeito a precisão na execução de movimentos do robô. Uma vez que o modelo implementado despreza a dinâmica do robô e considera que ele executa movimentos perfeitos. Durante a execução de movimentos não é assim que acontece. A dinâmica é considerada no ambiente de simulação e/ou ambiente real e faz com que o robô fique distante do objetivo ao final da execução da trajetória calculada, quanto maior a trajetória, maior será o erro.

A solução apresentada para esse problema foi a realização de um novo cálculo de trajetória a cada n passos, ou seja, conforme a execução do trajeto é realizado um novo cálculo de rota a cada n passos realizados. Com isso o problema da precisão foi reduzido e passou a ser considerado aceitável. Como mostra a figura 3.2

(a) Modelo original com um obstáculo.

(b) Modelo modificado com um obstáculo.

Figura 3.2: *Resultados da solução para o primeiro problema encontrado na implementação do planejamento de trajetória de (Behring et al., 2000) - Fonte: Ferreira (2014)*

O segundo problema encontrado no modelo implementado acontece quando o ponto inicial encontra-se dentro de uma área de obstáculo. Isso pode acontecer no processo de alargamento dos obstáculos, nesses casos o algoritmo não consegue realizar nenhuma trajetória, mesmo que em ambiente real exista, pois o cálculo da distância não atinge o células de obstáculos.

A solução para esse problema foi não trivial, pois existia várias possibilidades de acontecimentos do problema com várias configurações. A solução adotada consiste na implementação de outros 2 CA para a resolução do problema. O primeiro tem como objetivo tirar o robô da área de alargamento através do menor caminho possível, enquanto que o segundo classifica as áreas de alargamento de modo que o robô possa realizar sua trajetória pelo caminho mais distante possível do obstáculo real.

O trabalho apresenta resultados satisfatórios e realiza simulações em ambiente virtual, bem como experimentos utilizando um robô real, o e-puck.

3.8 Uma Plataforma de Software para Controle de Veículos Não-Tripulados e Planejamento de Trajetórias (Silva, 2016)

O trabalho de Silva (2016) realiza uma avaliação de plataformas de software capazes de controlar drones de pequeno porte com descrição de qualidade quanto ao controle de estabilidade, viabilidade e funcionalidades existentes nessas plataformas.

Foi implementado rotas para quadrirotores de pequeno porte, especificamente o Parrot AR.Drone 2.0. Foram analisadas as APIs AR.DroneFreeFlight, Node.js e YADrone. Para ambos foram analisados realizando testes de voo e suas possibilidades de implementação.

AR.DroneFreeFlight é uma plataforma proprietária, da mesma empresa fabricante do Drone, que dispõe de todas as possibilidades de movimentos que o drone pode realizar, no entanto o controle é fixo e não se pode realizar uma trajetória pre-programada e executada por um computador. O Node.js é um framework que disponibiliza todo o ferramental de acesso ao Drone no qual pode ser realizados movimentos com uma determinada precisão, mostrou-se o melhor controle dentre os estudados. O YADrone é uma biblioteca Java a qual faz uso do SDK do AR.Drone para a implementação de ações diretas no drone.

Dentre as plataformas analisadas Silva (2016) mostra que a maior precisão, resposta e controle se dá melhor com a utilização do NodeJS.

3.9 Discussão sobre abordagens relacionadas

A pesquisa aqui relacionada buscou trabalhos relacionados a autômatos celulares e a geração de rotas ou navegação de robôs móveis. O que foi levado em consideração na seleção deste trabalho está relacionado aos métodos utilizados e execução.

Dos trabalhos analisados nesta seção, nenhum deles abordou a temática relacionada a Drones aéreos. Todos utilizavam, mesmo em simulações, robôs terrestres para a execução do caminho gerado. Somente dois trabalhos que utilizam autômatos realizaram experimentos com robôs físicos. Todos os trabalhos realizaram simulações.

Capítulo 4

Modelo Desenvolvido e Métodos

Neste capítulo apresentam-se os resultados obtidos com o desenvolvimento do modelo computacional de navegação autônoma. Na Seção 4.1 é apresentado o Sistema desenvolvido, bem como os resultados de testes realizados. Na seção 4.2 apresenta-se resultados relativos aos cálculos de distância (escala das imagens e tamanho das células). A Seção 4.3 refere-se a resultados na geração de rotas a partir da dinâmica e estudo dos movimentos do veículo. Por fim, a Seção 4.4 apresenta-se resultados relacionados aos experimentos realizados

4.1 Sistema Desenvolvido e Resultados

O sistema implementado é parte de uma idealização maior, onde é projetado um modelo capaz de organizar a navegação de um robô aéreo não tripulado. Além do sistema de navegação, pretende-se que o modelo seja capaz de informar a localização do robô em tempo real, sem o uso de qualquer tipo de comunicação externa, tais como GPS. Este trabalho visa a realização de rotas autônomas para drones baseando-se em autômatos celulares. Para isso o modelo está dividido em três partes a serem discutidas: definição de passos: discretização de movimentos; manipulação de plataforma do drone; sistema de geração de códigos baseado em Autômato Celular; e, execução de scripts gerados na plataforma. O sistema como um todo funcionará da seguinte forma:

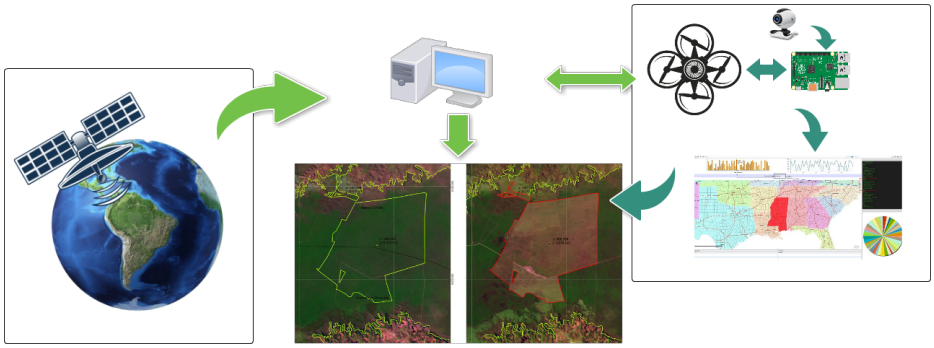

Figura 4.1: *Representação do Sistema completo - Fonte: elaborado pelo autor*

A Figura 4.1 mostra a composição de um sistema maior o qual esta Dissertação estuda parte dele, ou seja, meios de navegação autônoma de drones que possam utilizar desse sistema

para realizar monitoramento de áreas de preservação ambiental. O fluxo principal do sistema acontece a partir da inserção de uma imagem aérea, bem como seus parâmetros de obtenção da imagem. Através imagem do ambiente em que deseja monitorar, a partir dos parâmetros passados ao sistema, será calculado o tamanho da célula e rotas são geradas junto com arquivos de programação do drone para a execução autônoma, a frequência com que será realizados os voos é determinada pelo usuário. Um dos algoritmos utilizados para gerar rotas autônomas foi a realização de um "passeio aleatório". O motivo pelo qual foi utilizado esse método é a possibilidade de modelagem e simulação dos movimentos por meio de processos estocásticos.

4.1.1 Passeio Aleatório realizado pelo sistema

O passeio aleatório consiste em passos discretos de forma aleatória. Considere um caminhante que parte de uma origem e se desloca ao longo de um plano bidimensional. Cada passo, em intervalo de tempo T, o caminhante (drone) dará um passo de comprimento h para norte, sul, leste ou oeste. Supondo que a evolução do caminhante no plano $\mathbb{R}^2$ está descrita em coordenadas que assumem valores inteiros e que o tempo é discreto, ou seja as trajetórias podem ser modeladas por:

$$n \to X_n, n \in \{0, 1, 2, \cdots\}, X_n \in S,$$

onde S é o espaço bidimensional

$$S \equiv \mathbb{Z}^d := \{x = (x_1, ..., x_d) : x_i \in \mathbb{Z}\}$$

Dessa forma, a cada passo, o caminhante, ou drone, se desloca da célula onde se encontra para uma das $2d$ vizinhas a ele, com probabilidade $1/2d$. Ou seja, considerando $v_1, ..., v_d$ são vetores da base canônica de $\mathbb{R}^d$, temos que:

$$P(x \to x \pm v_j) = \frac{1}{2d}, \qquad \forall_i = 1, ..., d$$

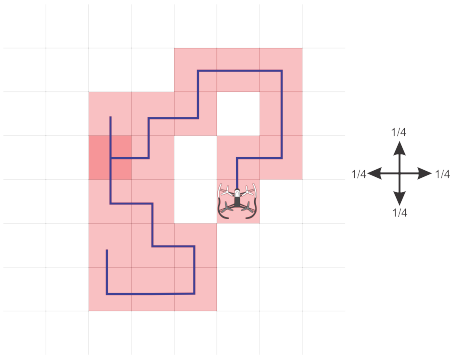

Figura 4.2: *Passeio Aleatório simétrico em dimensão $d = 2$, representação da movimentação do Drone. Quanto maior a quantidade de passos mais a trajetória se assemelha ao movimento Browniano - Fonte: elaborado pelo autor*

Levando em consideração que o drone sempre começa na origem no tempo $n = 0$, $X_0 = 0$

e que a cada unidade de tempo ele se desloca de uma célula para outra vizinha a ele com probabilidade de transição descrita anteriormente. Nessas condições se descreve um processo estocástico no conjunto S, também chamada de passeio aleatório simples simétrico.

4.1.2 Plataforma utilizada: Beebop Parrot

O Bebop Drone possui várias funcionalidades que permite o seu uso em diversos ambientes e para várias aplicações, em pesquisa, desenvolvimento e operacionalidade de algumas atividades das relacionadas na Seção 2.4.

As especificações do Parrot Bebop Drone, de acordo com o fabricante são (Parrot, 2017):

Figura 4.3: *Especificações do Bebop Drone - Fonte: http://global.parrot.com/au/products/bebop-drone/*

- **Conectividade**: WiFi com tecnologia MIMO (*Multiple Input/Multiple Output*) dual-band com possibilidade de utilização para 2.4 e 5 GHz; Com potência de envio de dados de 21dBm; e alcance de 250 metros; (item 7 da Figura 4.3)
- **Hardware e Estrutura**:
  1. 04 (quatro) Brushless (motores); (item 3 da Figura 4.3)
  2. Estrutura em ABS com reforço de 15% de fibra de vidro; (item 4 da Figura 4.3)
  3. Proteção em EPP de alta resistência para voos indoor; (item 9 da Figura 4.3)
  4. Hélices em policarbonato com três pás; (item 5 da Figura 4.3)
  5. Sistema anti-vibração; (item 10 da Figura 4.3)
- **Velocidade**: 13 m/s;
- **Câmera**: Câmera com lente "Fisheye"180º 1/2,2", 6 elementos óticos e sensor de 14 Mega pixels; (item 2 da Figura 4.3) Definição de vídeos: 1920x1080p (30fps); Definição de foto: 4096x3072px; Codec de vídeo: H264; Formato de imagem: JPEG, RAW, DNG; Memória interna: 8GB;

- **Bateria**: Lithium Pelymer 1200 mAh;
- **Processador**: Parrot P7 dual-core CPU Cortex 9 (item 1 da Figura 4.3) Quad Core GPU; Sistema Operacional Linux;
- **Sensores**: Magnetômetro 3-eixos ; giroscópio 3-eixos; acelerômetro 3-eixos (item 6 da Figura 4.3); Sensor de fluxo ótico: estabilização vertical da câmera (item 8 da Figura 4.3); Sensor de Ultrassom; Sensor de pressão;
- **Geo-localização (GNSS)**: GNSS (GPS + GLONASS) (item 6 da Figura 4.3);
- **Dimensões de peso**: 28x32x3.6cm sem proteção; 33x38x3.6cm com proteção; Peso: 400g com bateria;

Uma característica fundamental do Bebop Drone para a implementação deste trabalho é a disponibilidade de um SDK (*Software Development Kit*) OpenSource, esse fato, possibilida a implementação de sistemas de controle computacional, tornando possível a tarefa de automação dos movimentos.

4.1.3 Experimento de Voo e Análise de Movimentos Controlados em Ambiente Externo

Para a utilização da plataforma foi necessário realizar testes e experimentos controlados antes de definir passos autônomos para o sistema. Foram realizados 04 (quatro) voos experimentais, analisando dados de sensores que o veículo dispõe como recurso com a finalidade de entender e analisar a precisão de movimentos e definição de dinâmica cinética para diferentes ambientes, bem como interferências em movimento. Um desses voos foi realizado com a utilização de API programável, com intuito de testar voo autônomo.

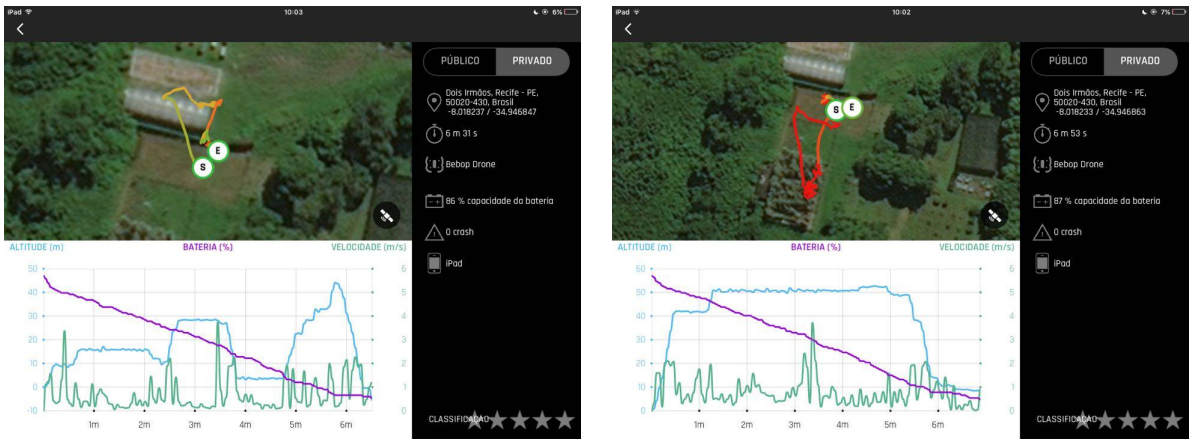

Figura 4.4: *Experimentos realizados para a definição de precisão de movimentos do Drone utilizado neste trabalho. Nos gráficos, canto inferior esquerdo de cada imagem, estão são apresentados os dados de velocidade, altitude e decaimento da carga de bateria. No mapa, canto superior esquerdo de cada imagem, são apresentados a trajetória realizada pelo drone, na qual ele parte do ponto "S"e retorna no ponto "E". - Fonte: PrintScreen do Aplicativo de controle do Drone*

Voo	Altitude Máxima	Tempo de voo	Descarga de Bateria
01	45m	6m 31s	86%
02	52m	6m 53s	87%

Tabela 4.1: *Dados de voos dos experimentos*

O local escolhido para a realização dos experimentos foi uma área livre onde possui uma área de matagal e áreas de plantação, também possui uma estação meteorológica no local a qual foi possível coletar informações ambientais no dia do experimento e a partir daí ter uma analise mais detalhada dos movimentos do drone.

Como pode ser visto na trajetória que o drone realiza (Figura **??**), possui uma inclinação para noroeste no momento em que se realizava a movimentação para frente, esse desvio de rota foi explicado quando analisado os dados ambientais, o vento foi o motivo pelo qual o deslocamento da movimentação descrita foi devido fatores externos.

Data	Temperatura	Umidade do Ar		Vel. Vento	Direção do Vento
10/01/2017	29,12 °C	57,40%	88,30%	3,68m/s	279,6º - Noroeste

Tabela 4.2: *Alguns dados meteorológicos ambientais do local do experimento. Fonte: GEOSERE/UFRPE*

Além dos movimentos analisados no experimento, também foram geradas imagens com a finalidade de análise da qualidade de vídeos e fotos gerados pelo equipamento. Abaixo seguem algumas imagens geradas durante o experimento:

Figura 4.5: *Imagens geradas pelo Drone durante os voos de experimento - Fonte: elaborado pelo autor*

4.1.4 Experimento de Voo e Análise de Uso para Monitoramento Ambiental

A principal aplicação que fará uso desse trabalho é o monitoramento ambiental. Assim é importante que o equipamento seja capaz de gerar informações relevantes com essa finalidade. Com a finalidade de teste da plataforma escolhida para este trabalho, no dia 13 de setembro de 2016 foram realizados sobrevoos de reconhecimento em uma área de preservação ambiental, a Unidade de Conservação da Mata do Zumbi, localizada na cidade do Cabo de Santo Agostinho, estado de Pernambuco (Figura 4.6). Na ocasião, Agentes da Agência Estadual de Meio Ambiente (CPRH) e policiais da Companhia Independente de Policiamento do Meio Ambiente (CIPOMA) também estiveram no local para vistoria e levantamento da degradação ambiental do local.

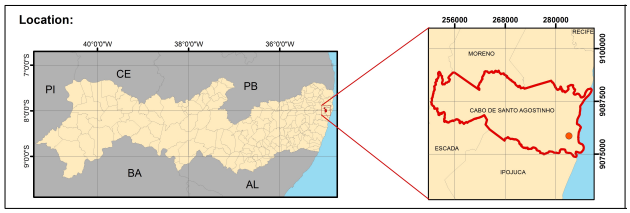

Figura 4.6: *Área estudada - S 8º19'30", W34º58'12" - Fonte: GEOSERE/UFRPE - Leite et al. (2017)*

Para a realização dos voos foi utilizada a aplicação disponibilizada pelo fabricante para controle do Drone. O aplicativo FreeFlight Pro é um aplicativo da Parrot SA, disponibilizado para o controle de drones Parrot. Com ele foi realizado o controle de movimentação do drone em todo o trajeto. Os voos tiveram como objetivo a realização de tomadas de fotos e vídeos para avaliar a viabilidade de uso do drone para essa finalidade. Junto com as fotos, foram registrados pontos de desmatamento e construções ilegais. Com a utilização de software Google Earth e ArcGis, foram mapeados esses pontos e dispostos em mapa com escala de 1:1500 utilizando WGS1984 como norma.

O sobrevoo aconteceu durante a ação de vistoria do local pelos agentes da Agência Estadual de Meio Ambiente (CPRH) e Companhia Independente de Policiamento do Meio Ambiente (CIPOMA), também com apoio também da Polícia Militar e teve 06 (seis) pontos de desmatamento e construções ilegais mapeados, como mostra a Figura 4.7. As imagens dispostas para cada ponto foram capturadas pelo drone em operação controlada.

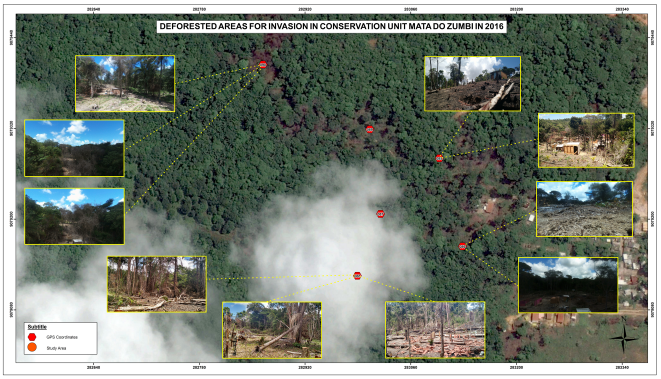

Figura 4.7: *Seis pontos georeferênciados e identificados desmatamento e construções irregulares na área, as imagens em destaque foram capturadas pelo drone em sobrevoo - Fonte: GEOSERE/UFRPE - Leite et al. (2017)*

Os gráficos a seguir foram gerados pelo aplicativo de controle do Drone, o FreeFlight, e mostram os dados de movimentação, altitude e descarga de bateria do drone nos voos realizados:

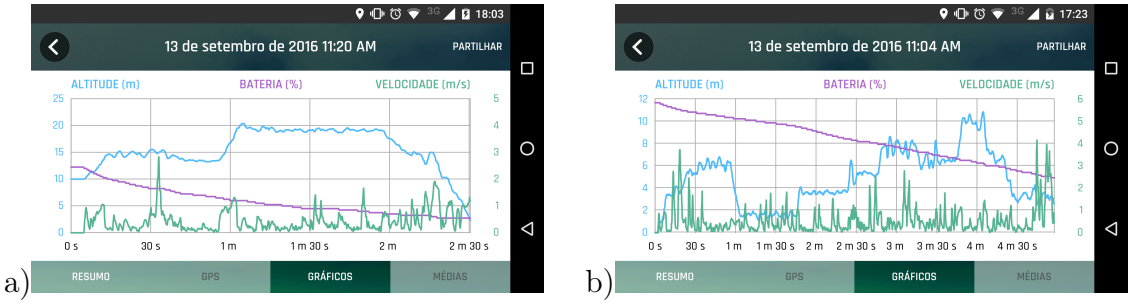

Figura 4.8: *Dados de voos, descarga de bateria, aceleração e altitude - Fonte: Gerado pelo FreeFlight*

Como pode ser visto na Figura 4.8, o primeiro voo (Fig.4.8a) teve duração de 2 minutos e 30 segundos, atingiu 21m de altitude e velocidade máxima em deslocamento de 3m/s, o consumo de bateria foi de 38%. Já o outro gráfico (Fig.4.8b) o voo teve duração de 4 minutos e 57 segundos, descarga de 45% da bateria, atingiu 11 metros de altura e um deslocamento de aproximadamente 150 metros, durante o voo a velocidade máxima alcançada foi de 5 m/s.

Como resultado dessa experimentação dos recursos do Drone, foi publicado um resumo no InterForensics Conferência Internacional de Ciências Forenses ((Leite *et al.*, 2017)).

4.1.5 Experimento de Voo e Analise de Movimentos Autônomos em Ambiente Externo

Considerando o objetivo desse trabalho, também foi analisado uma API para programação de voos autônomos utilizando o Bebop Drone. A plataforma de programação utilizada para executar a programação foi o Node.js. Esta tecnologia fornece um framework para entrada e saída assíncrona que possibilita programação simples e padrões de programação orientada a eventos.

Figura 4.9: *Local de realização do experimento de plataforma de programação de drones. Cada ponto demarcado na imagem é um vértice do trajeto - Fonte: Google Maps - adaptado pelo autor*

O procedimento do experimento foi a realização de sobrevoo autônomo em uma área de aproximadamente 50m com rota definida na programação. A rota executada pelo drone consistia em 5 movimentos e teve como finalidade analisar e observar o comportamento do drone com relação a comunicação com computador que estivesse processando os comandos, estabilidade de voo, captura de fotos e vídeo automática e definição de tamanho de passo para execução de movimentos.

A execução do teste, para ser bem sucedido, o drone deveria executar os comandos programados no script de execução e para isso foi configurado para utilizar 50% da potência dos motores e a cada passo dado o equipamento realizava a captura de uma foto do ambiente. A movimentação que ele deveria realizar foi respectivamente: decolar, realizar um deslocamento para a direita por 10 segundos, subir por 10 segundos, se deslocar para frente por 10 segundos, se deslocar para a esquerda por 10 segundos, se deslocar para trás por 10 segundos e, por fim, pousar. Teoricamente, o drone voltaria para posição inicial realizando um movimento de "quadrado", assumindo uma visão 2D, superior ao movimento, conforme apresentado na Figura 4.9 por pontos.

A execução dos scripts de controle do Bebop acontecem de forma assíncrona e por socket de comunicação de rede Wifi estabelecida pelo próprio equipamento. Assim, foram enviados os comandos de acordo com a leitura da execução do processo node.js. Era esperado que a execução acontecesse por completa, uma vez que já havia sido testada em uma escala de tempo menor. No entanto, não foi o que aconteceu. Durante a execução do script, houve uma falha no processo e aconteceu uma perda de comunicação juntamente com o controle da execução dos comandos enviados para o drone. O drone continuou a execução do último movimento realizado indefinidamente, ou seja, ele manteve-se em voo para frente até colidir com uma árvore, só assim parou a execução do script.

```
1  function Comandos(){
2      d.takeOff(); //Decolagem
3      setTimeout(function () {
4          d.stop(); //Parada
5          d.takePicture(); //Captura de imagem
6          d.on('PositionChanged', function(data) {
7              console.log('Imagem Capturada em: ' + data);
8              console.log(data);
9          });
10         d.forward(50); //'Passo' para frente a 50% da
               potencia
11     },2000);
12 }
```

Código 4.1: *Exemplo de Código executado no Drone*

No bloco de código 4.1 é apresentado um exemplo do que é executado para a realização de movimentos. A distância percorrida, altitude e inclinação dependem de quanto tempo o comando permanecerá em execução, além disso eles são realizados de forma assíncrona e concorrente, assim como todo processo no Node.js. Dessa forma, o que ocasionou a falha de comunicação e perda do controle do drone foi a concorrência de mais de um processo sendo executado ao mesmo tempo e, assim, a execução perde o tempo de envio de cada comando resultando em erro de execução e consequentemente quebra na execução do processo. Como pode ser visto no código, toda função de execução de um passo acontece em um tempo determinado, na linha 11, pode ser visto o tempo (2000 milissegundos) que será executada

a função anônima (iniciada na linha 3) que irá parar o drone, capturar uma imagem, e por fim realizar um passo para frente. Caso o processo seja parado, por *exception*, *timeout*, ou erro de execução, antes da execução de algum dos passos, o drone permanecerá no último comando processado. Portanto é necessário que haja um tratamento de erro no processo, bem como, redundância em códigos de segurança, para que o veículo não seja perdido.

Figura 4.10: *Imagens capturadas pelo Drone durante execução de experimento autônomo - Fonte: elaborado pelo autor*

Contudo, o drone foi recuperado posteriormente e pôde ser analisado o que havia acontecido. As imagens apresentadas na Figura 4.10 foram capturadas durante a execução do experimento. O vídeo gerado pelo drone pode ser visto no link: https://www.youtube.com/watch?v=Y0Vtqjn3oY4.

4.2 Imagem do Ambiente e Divisão do Espaço Celular

Para a determinação do espaço celular no ambiente em que irá executar e para que o autômato celular seja coerente com a realidade (em caso de inexistência de imagem aérea, imagem muito baixa, ou falta de parâmetros de configurações de imagem), será considerado que cada célula terá o tamanho do drone que estiver navegando no ambiente. As distâncias padrões mínimas serão estabelecidas mais adiante neste trabalho.

Caso exista uma imagem aérea como entrada para determinação do espaço celular, o cálculo para determinar o tamanho de cada célula será baseado no GSD (Ground Sample Distance), ou seja, a equivalência de cobertura de área no terreno para um pixel.

A determinação desse valor se dá através de um cálculo simples demostrado na Figura 4.12. Os parâmetros necessários para ter conhecimento da área de cobertura de cada pixel são: altitude em que foi capturada a imagem, tamanho do pixel no sensor e distância focal da câmera, com isso é possível determinar a área de cobertura da imagem como um todo e posteriormente sua correspondência de tamanho em terreno.

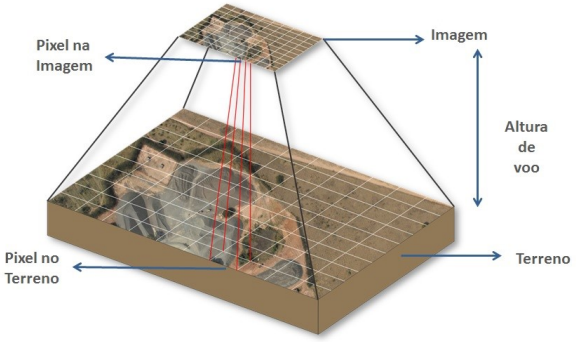

Figura 4.11: *Representação de pixel no terreno - Fonte: http://blog.droneng.com.br/planejamento-de-voo/*

Como pode ser visto na Figura 4.11, cada pixel na imagem corresponde a uma determinada área no terreno fotografado, mesmo existindo parâmetros fixos que são provenientes de cada câmera, o tamanho da área de cobertura depende diretamente da altitude com que foi fotografado.

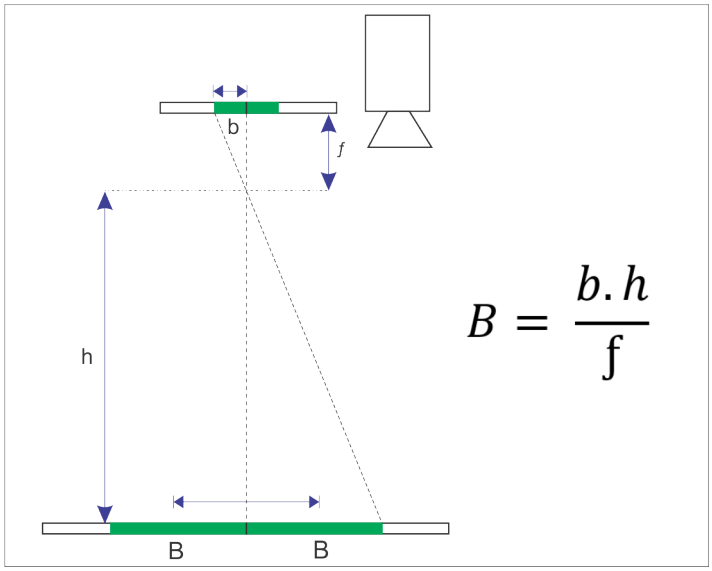

Figura 4.12: *Representação do cálculo realizado para obter o valor do GSD - Fonte: adaptado de http://blog.droneng.com.br/planejamento-de-voo/*

Assim, o tamanho da célula na representação do ambiente depende dos parâmetros da imagem a ser carregada. Para todos os casos é considerado que cada célula tem o tamanho do drone em estudo.

4.2.1 Definição de Passos: estudo de movimentação da plataforma

Como visto anteriormente nesta mesma seção, item sobre passeio aleatório (na subseção 4.1.1), nessa aplicação o modelo assume que os movimentos serão realizados de forma discretas, ou seja, é considerado tempo discreto e passos determinados de comprimento l para qualquer dos lados, e cada movimento representa a passagem de uma célula para outra vizinha. É admitido a vizinhança de Von Neumann para a locomoção do Drone, ou seja, cada

passo será realizado sempre para um dos quadro lados da vizinhança, para frente, esquerda, direita ou trás, ambos os lados com a mesma probabilidade de acontecer.

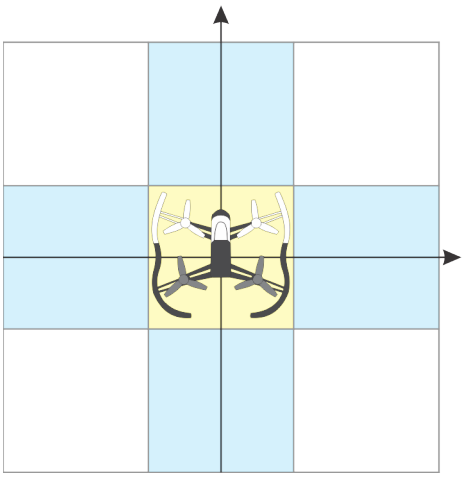

Figura 4.13: *Representação virtual do ambiente, dividido em células e coordenadas para localização de precisão de movimentos - Fonte: elaborado pelo autor*

Será considerado um passo toda transição do drone de uma célula para outra, assim o autômato celular irá processar a movimentação entre as células, assim montando um caminho durante a execução do autômato.

4.2.2 Experimento de Takeoff com Utilização de Aplicativo do Fabricante (FreeFlight)

Para definição de tamanho de passos, tempo de deslocamento e velocidade, foi montado um ambiente com a finalidade de definir e acompanhar a movimentação do drone em diferentes condições e uso de softwares distintos.

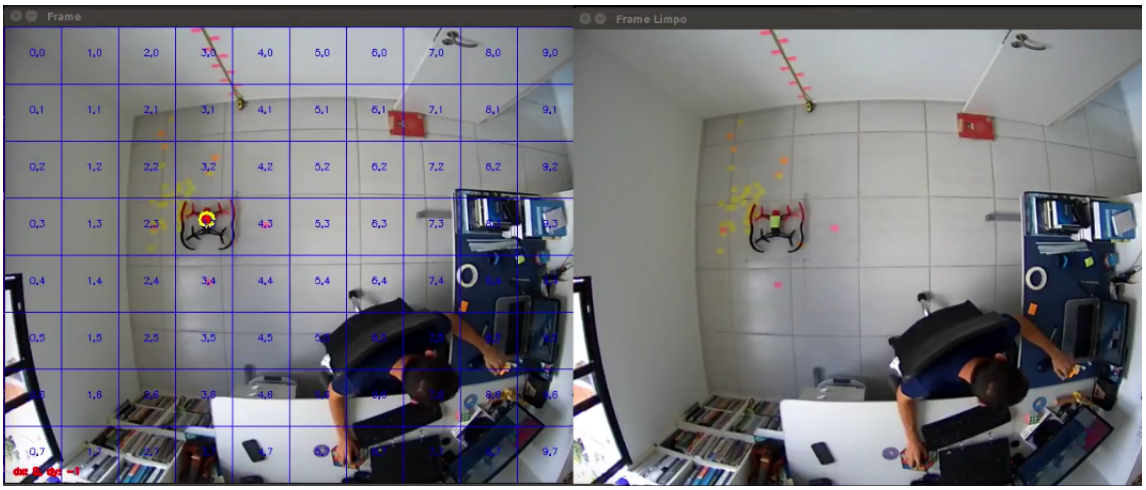

Figura 4.14: *Ambiente de realização dos experimentos - Fonte: elaborado pelo autor*

O primeiro ambiente estudado foi o mais controlado possível: um quarto com ausência de interferências externas (vento). Neste ambiente foi realizado o experimento de Takeoff: o experimento consistiu em realizar a decolagem do drone e mantê-lo pairando por 20 segundos,

em um ambiente controlado com a finalidade de observar o seu comportamento. Era esperado que o drone pousasse no mesmo ponto de decolagem, ou com um erro de no máximo 10cm; (vídeos dos experimentos). Foram realizados 30 (trinta) repetições e coletados os dados relativos aos voos conforme a Figura 4.15.

No ambiente de experimento foi instalado uma câmera no teto, objetivando acompanhar os experimentos, um software, também desenvolvido para essa finalidade, acompanha o drone e mapeia os movimentos para melhor caracterização e descrição dos movimentos.

Experimentos realizados geraram o seguinte gráfico com resultados:

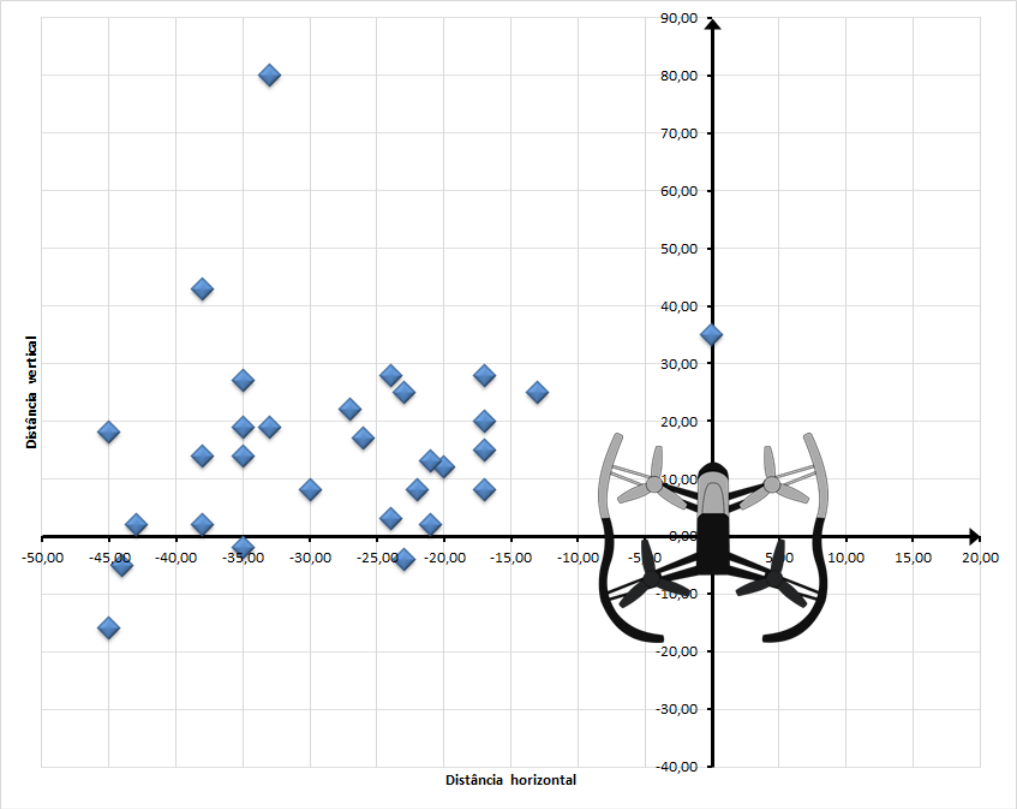

Figura 4.15: *Coordenadas de pousos realizados no Experimento de movimentação - Fonte: elaborado pelo autor*

Média da distâcia	Desvio Padrão
36,56 cm	12,17

Tabela 4.3: *Dados extraídos do Gráfico do experimento de Takeoff usando o FreeFlight*

Cada ponto no gráfico foi um pouso realizado. Com os dados foi calculada a distância do ponto de pouso para o ponto de decolagem, além de outros dados estatísticos, como apresentado na tabela 4.3.

Com o experimento ficou claro alguns parâmetros importantes para a geração de rotas automáticas. Tais parâmetros foram obtidos pela análise dos dados em gráficos, como os da figura 4.16, e também por meio de observações. Como pode ser visto na Figura 4.15 o drone tem uma tendência para o lado esquerdo, isto ocorre devido ao sistema de controle de estabilidade feito através de sensores inerciais. Com efeito os sensores inerciais tendem a manter o movimento ou não movimento do veículo com o objetivo de estabilizá-lo. No

entanto para que ocorra uma atuação de correção é necessário que aconteça uma ação. Como os testes foram realizados próximos a uma parede, o deslocamento de ar realizado pelo próprio drone em voo gerava uma força no sentido contrário à parede, com a correção do sensoriamento inercial o drone acabava com instabilidade e aterrissava próximo à parede.

Figura 4.16: *Gráfico e Configurações de exemplo de geração de dados em voos Link para todos os gráficos - Fonte: elaborado pelo autor*

Um outro parâmetro importante observado com os dados gerados em voo é o tempo de decolagem. Em todos os voos realizados, o tempo de decolagem ficou em média 4,016 segundos. Esse resultado é importante na implementação de rotas autônomas, pois devido aos processos serem assíncronos e concorrentes, é importante que o drone execute um movimento de cada vez. Ou seja, a partir do conhecimento do tempo de decolagem, os movimentos posteriores a decolagem devem ser executados semente após o tempo de decolagem.

4.2.3 Experimento de Takeoff com Utilização de API Node.js

Além do experimento realizado com o aplicativo do fabricante, também foi realizado o mesmo experimento por meio de controle distinto, ou seja, foi utilizado a API Node.js para enviar comandos de decolagem e aterrissagem. O experimento consiste sempre em decolar do mesmo ponto, pairar por 20 segundos e em seguida aterrissar. Era esperado que, como o ambiente é controlado e não há nenhuma força de deslocamento externo no drone, no entanto, como mencionado no experimento anterior o deslocamento de ar causado pela movimentação do drone, altera o ambiente causando uma força de atuação sobre ele, o que pode ocasionar um deslocamento para próximo das paredes.

O bloco de código 4.2 mostra o scrípt executado no experimento:

```
1  "use strict";
2  var bebop = require("node-bebop");
3  var drone = bebop.createClient();
4  drone.connect(function() {
5    drone.takeOff();
6    setTimeout(function() {
7      console.log("Landing...");
8      drone.land();
9    }, 25000);
10 });
```

Código 4.2: *Código executado no Drone como teste de precisão de decolagem*

O experimento foi executado 30 vezes e foram obtidos resultados dispostos no gráfico apresentado na Figura 4.18:

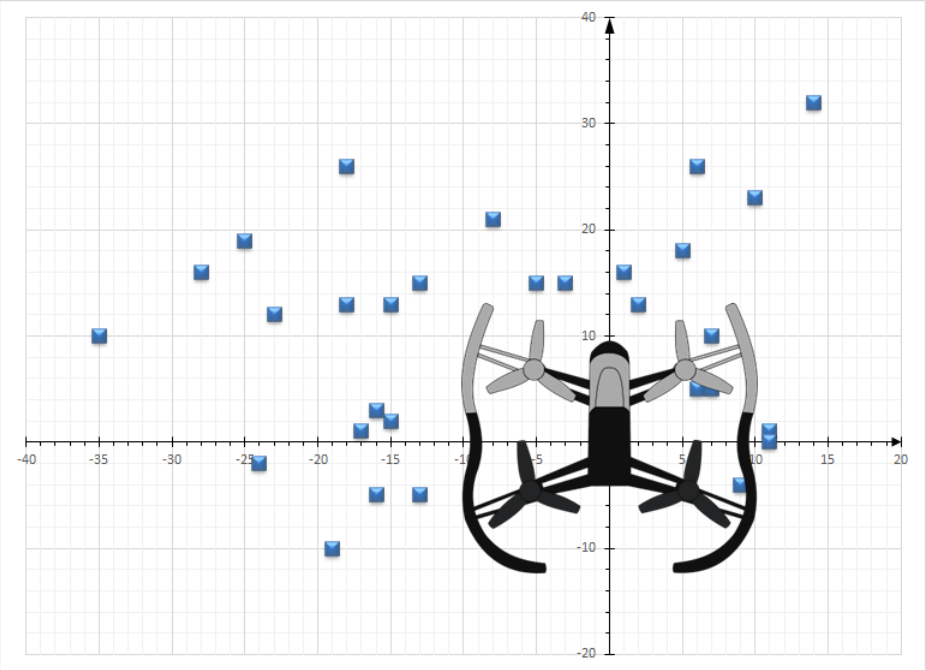

Figura 4.17: *Coordenadas de pousos realizados no Experimento Takeoff utilizando a API NodeJS - Fonte: elaborado pelo autor*

Média da distâcia	Desvio Padrão
19,73 cm	7,91

Tabela 4.4: *Dados extraídos do Gráfico*

Assim como no experimento anterior foram coletados dados referentes à distância entre o ponto de decolagem e o pouso. A média da distância e desvio padrão dos dados são apresentador na Tabela 4.4. Foram relacionadas as distância e calculado dados estatísticos para fins de comparação com o experimento anterior.

4.2.4 Análise de Dados e Resultados dos Experimentos

Ambos os experimentos descritos anteriormente (subseção 4.2.2 e subseção 4.2.3) foram realizados com o intuito de comparação entre as plataformas de softwares utilizados, com relação a controle e mobilidade do veículo. O primeiro teste objetiva uma comparação de precisão entre as plataformas de controle.

No experimento descrito na Subseção 4.2.2 foi realizado com software do fabricante, obteve uma média de distância entre o ponto de decolagem e o ponto de pouso de 36,56 cm, e um desvio padrão de 12,17 cm, conforme dados apresentados no Apêndice A. O outro experimento, descrito na Subseção 4.2.3, obteve uma média de distância de 19,73 cm entre o ponto de decolagem e o pouso, com desvio padrão de 7,91 cm, os dados também estão dispostos no Apêndice A.

Como demonstrado, foram tomados como base e parâmetro de configuração os dados relativos ao experimento com o software do fabricante, uma vez que foi desenvolvido para essa

finalidade e tem acesso a todos os recursos disponíveis pelo veículo, tende a ser mais preciso e mais completo. Assim, foi realizada uma comparação entre os dois experimentos. Tomando como base a média do experimento descrito na Seção 4.2.2 e levando em consideração um intervalo de confiança de 95%, foi aplicado o teste T-Student como método de comparação da média dos dois experimentos obtendo os seguintes valores: $\overline{x} = 36,56$ e $s = 148,32$. Como iremos adotar o intervalo de confiança de 95% temos $\alpha = 0,05$ e $n = 30$. Segundo a tabela de distribuição de T-Student temos que $t_{\alpha/2} = 2,045$. Com isso, iremos rejeitar H_0 se $t_{obs} < -2,045$ ou $t_{obs} > 2,045$ onde

$$t_{obs} = \frac{\overline{x} - \mu_0}{\frac{s}{\sqrt{n}}} \Rightarrow \frac{36,56 - 19,73}{\frac{148,32}{\sqrt{30}}}$$

Dessa forma, o valor de $t_{obs} = 0,6216$. Ou seja, como t_{obs} é maior que $-2,045$ e menor que $2,045$ não deve ser rejeitada a hipótese nula, ou seja a diferença entre as duas médias é irrelevante em um intervalo de confiança de 95%.

4.2.5 Experimento de Movimentação entre Células com Utilização de API Node.js

Uma vez analisada a precisão para ambas as plataformas de controle do drone, neste experimento pretende-se determinar o tempo e potência de movimento para o veículo realizar a transição de uma célula para outra, considerando a menor célula possível (do tamanho do drone).

O experimento teve intuito de configurar variáveis de tempo de execução t e potência dos motores p. Assim, foram realizados os seguintes passos nesse experimento: decolagem, movimentação para a direita por t segundos, parada, movimentação para esquerda por t segundos, parada e aterrissagem. O controle foi feito utilizando Node.js, como mostra o código de execução 4.3.

```
1  "use strict";
2  var bebop = require("node-bebop");
3  var power = 20;
4  var drone = bebop.createClient();
5  drone.connect(function() {
6    drone.takeOff();
7    setTimeout(function() {
8      drone.right(power);
9      console.log("Manobra para direita!");
10   }, 7000);
11   setTimeout(function() {
12     drone.stop();
13     console.log("Parada!");
14   }, 9000);
15   setTimeout(function() {
16     drone.left(power);
17     console.log("Manobra para esquerda!");
18   }, 11000);
19   setTimeout(function() {
```

```
20      drone.stop();
21      console.log("Parada!");
22    }, 13000);
23    setTimeout(function() {
24      drone.land();
25      console.log("Pouso!");
26    }, 15000);
27  });
```

Código 4.3: *Código executado no Drone para o experimento de transição entre células*

Além da análise de movimentos, tempo e potência utilizados para a transição de uma célula para outra, também era esperado que se mantivesse dentro da média de movimentação aceitável, uma vez que o experimento realiza o passo para a direita e em seguida retorna ao ponto de partida. Foram repetidos 30 movimentos com os resultados apresentados no gráfico 4.18.

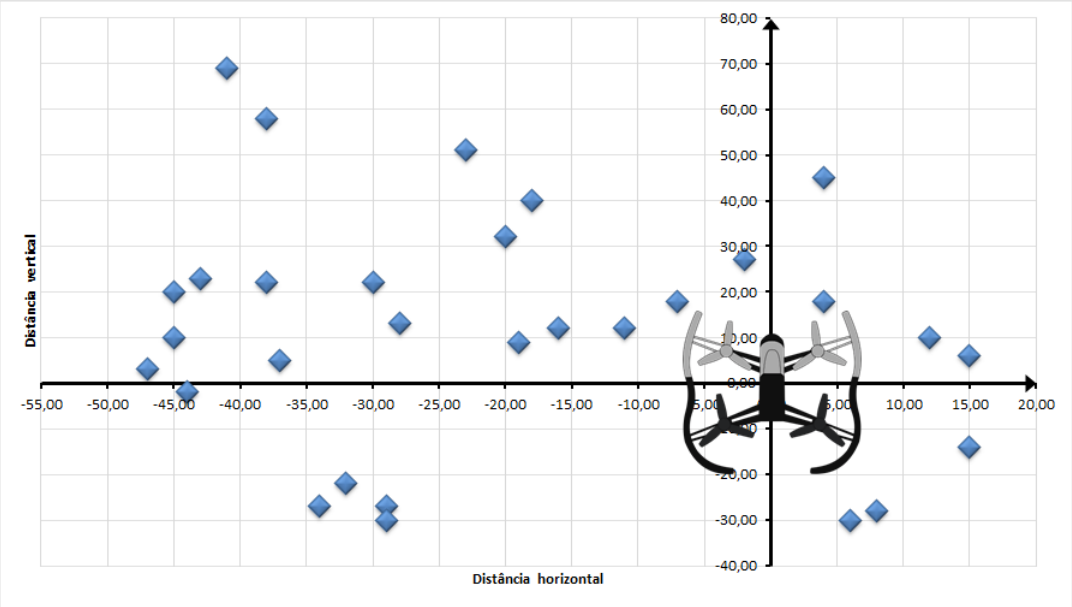

Figura 4.18: *Coordenadas de pousos realizados no Experimento de movimentação - Fonte: elaborado pelo autor*

Como pode ser visto no gráfico, cada ponto marcado no plano cartesiano corresponde a um voo realizado com o par ordenado no local da aterrissagem. Pode ser encontrado os vídeos da execução do experimento no link a seguir: vídeos de execução de movimentos.

Durante a execução do experimento, durante o ajuste do código a ser executado, houve perda de controle do drone (vídeo da perda de controle), após a perda houve uma tentativa de retomada que depois de algum esforço empregado foi bem sucedida, ou seja, foi retomado o controle do drone fazendo com que o mesmo aterrissasse de forma segura e controlada.

Como resultado deste experimento, o melhor controle para o ambiente descrito foi com potência de 20% e execução de movimentos com intervalo de 2 segundos entre eles com exceção da decolagem que foi considerado 7 segundos desde o acionamento dos motores, com isso o drone executou um passo de aproximadamente 93 cm, dessa forma, podemos estabelecer o menor tamanho de célula como sendo o tamanho do drone mais 1/2 de seu raio para cada lado, obtendo um valor de 76cm. Foi também realizado o teste de hipótese de

aceitação na média de distância do pouso e ponto de partida obtendo os seguintes valores: $\overline{x} = 36,56$ e $s = 148,32$. Como iremos adotar o intervalo de confiança de 95% termos $\alpha = 0,05$ e $n = 30$. Segundo a tabela de distribuição t de Student temos que $t_{\alpha/2} = 2,045$. Com isso, iremos rejeitar H_0 se $t_{obs} < -2,045$ ou $t_{obs} > 2,045$ onde

$$t_{obs} = \frac{\overline{x} - \mu_0}{\frac{s}{\sqrt{n}}} \Rightarrow \frac{36,56 - 38,87}{\frac{148,32}{\sqrt{30}}}$$

Dessa forma, o valor de $t_{obs} = -0,0850$. Ou seja, como t_{obs} é maior que $-2,045$ e menor que $2,045$ a hipótese nula é aceita, ou seja a diferença entre as duas médias é irrelevante em um intervalo de confiança de 95%.

4.3 Geração de Rotas Autônomas por meio de Autômato Celular

Após a realização de todos os experimentos práticos de conhecimento da plataforma que será controlada, definição de configurações do drone e especificações mínima para células, descreve-se aqui o sistema para a geração de rotas autônomas em veículos aéreos.

A aquisição de imagens aéreas será o *input* para o sistema de navegação por meio de um modelo baseado em autômatos celulares. Conforme indicado na Seção 4.2, o tamanho das células depende de parâmetros de configurações da imagem do sistema. É necessário que seja informada a distância focal e o tipo de sensor da câmera. No entanto é considerado inicialmente o tamanho mínimo para a célula, definida por meio de experimentos, ou seja $76cm^2$.

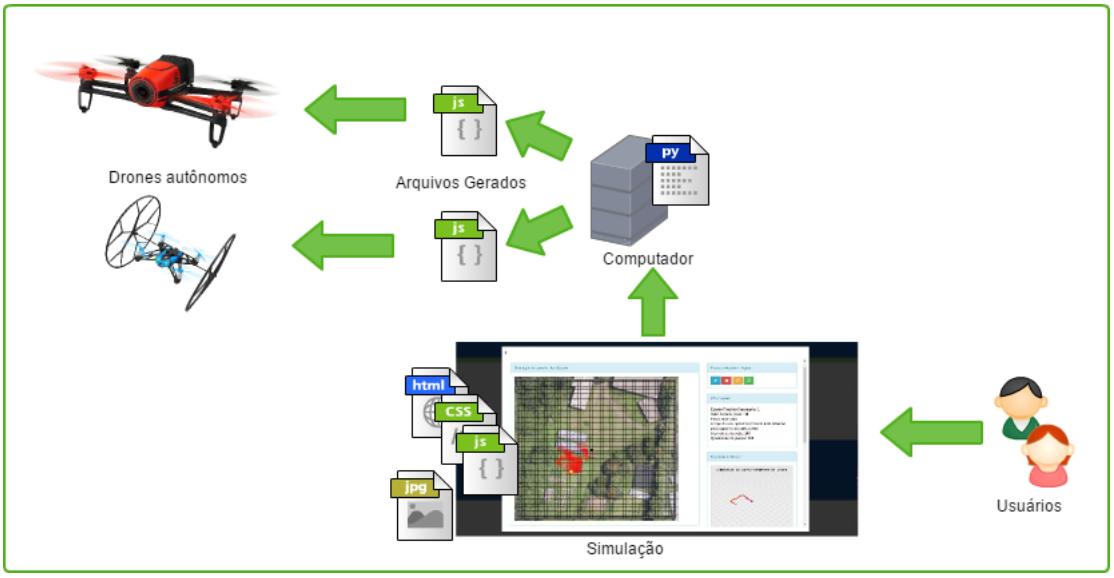

Figura 4.19: *Fluxo de funcionamento do sistema como um todo - Fonte: elaborado pelo autor*

A Figura 4.19 mostra como será o funcionamento do sistema por completo. Os usuários, conectados à rede do drone, acessam a página de simulação, após a realização da simulação o sistema gera a rota de acordo com a configuração desejada e posteriormente executa no veículo, fazendo com que ele se comporte conforme o que foi simulado e com raio de erro nos

movimentos de aproximadamente 19 cm para mais ou para menos. Observa-se que, devido a característica estocástica do processo, o erro é cumulativo, ou seja, quanto mais passos, mais tempo de voo o drone executa, maior será o erro associado aos movimentos.

Figura 4.20: *Imagem mostra uma simulação de 360 passos aleatórios em um ambiente - Fonte: elaborado pelo autor*

Na Figura 4.20 está a representação da simulação baseada em autômato celular. Na imagem, para cada posição da matriz é a representação de uma célula do autômato, cada célula representará uma área de acordo com a escala da imagem adquirida, nesse caso é aproximadamente $3m^2$. As células com marcação vermelha resultaram na passagem por aquela célula. A célula em preto trata-se da demarcação de onde partirá o veículo. O início da simulação leva em consideração somente um robô no ambiente, é admitido que o mesmo iniciará ao centro da imagem voltado para o norte, assim admitirá que os passos sejam executados como deveriam ser.

Implementação do sistema WEB

O sistema WEB integra um cadastro para drones com uma finalidade mais ampla, posteriormente pretende-se evoluir a aplicação para que ofereça suporte a outros drones além dos já existentes. Por isso sua arquitetura foi pensada para expansão de funcionalidades. Toda a arquitetura do sistema é composta por camadas como apresentado na Figura 4.21. No desenvolvimento da aplicação foi levado em consideração os seguintes padrões de projeto relacionados a seguir:

- MVC - um acrônimo para Model, View, Controller. O MVC faz com que a aplicação seja subdividido em uma aplicação em três partes, ou camadas: Model – camada em que envolve todas as entidades responsáveis pelo gerenciamento de dados da aplicação. View – está diretamente ligada ao usuário, responsável pela apresentação do sistema como um todo. Controller – é a interface de comunicação entre a view, comandando a Visão e o Modelo para se alterarem de forma apropriada (Minetto, 2007);

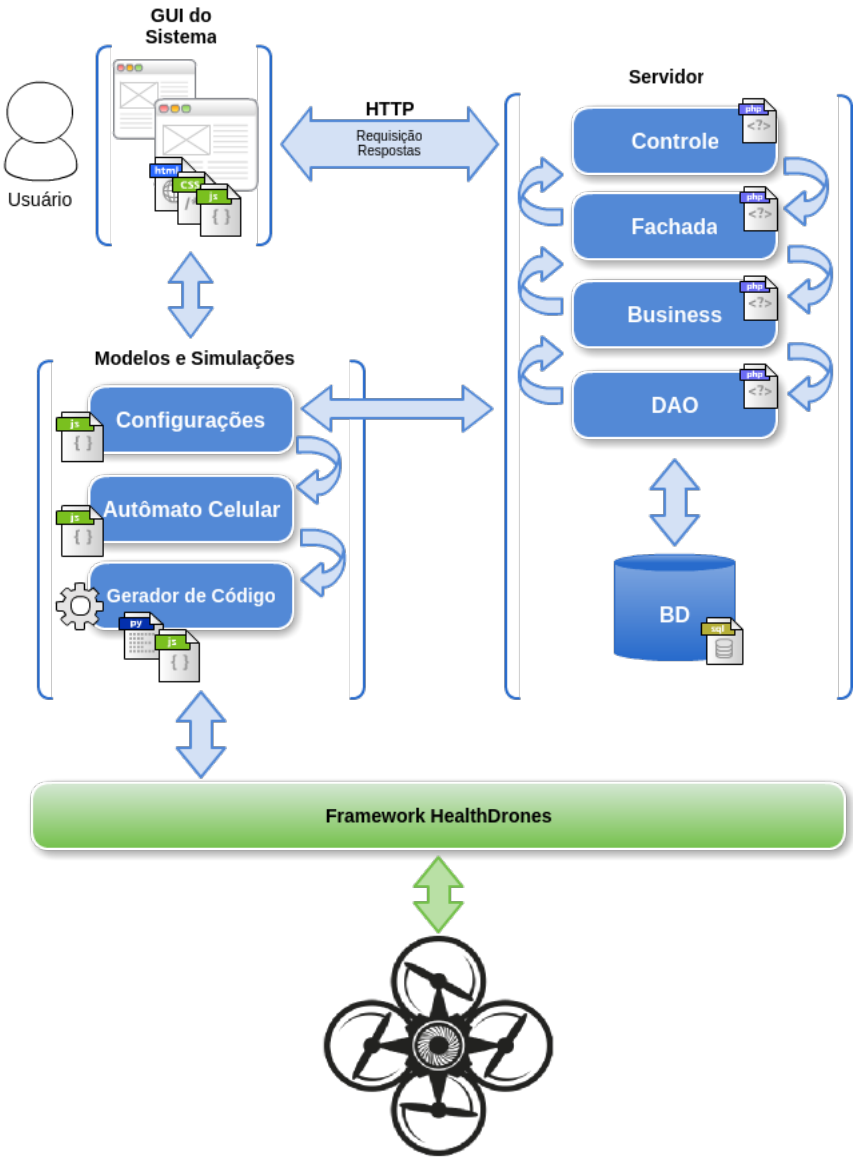

Figura 4.21: *Arquitetura do Sistema - Fonte: elaborado pelo autor*

- VO (*Value Object*) - um padrão que é utilizado como transferência agregada de dados entre diferentes entidados do sistema, como um meio de transporte e comunicação entre as classes DAO, Business e Fachada (Sampaio, 2007);
- DAO (*Data Access Object*) - padrão destina-se à o estabelecimento de objetos que trazem acesso aos dados que estão por trás de uma interface comum (Valdameri e Bachman, 2007);
- ObjectBusiness - implementado pelo sistema como meio de abstrair ou encapsulamento das regras de negócio e do gerenciamento da camada de persistência DAO (Sampaio, 2007);
- Fachada - realiza a interação de toda comunicação reunindo métodos que podem ser complexos de acessar a serviços (Sampaio, 2007).

O principal objetivo da aplicação web é a interface com o usuário. Porém o núcleo de geração de rotas é uma aplicação desenvolvida em python que será apresentada a seguir.

Autômato Celular e Geração de Scripts para o drone

O núcleo de geração de scripts é um programa escrito em python que gera e executa arquivos de movimentação dos drones em Node.js. A implementação do sistema é a parte principal do trabalho, é com uso dele que os drones irão executar as rotas geradas.

Para a implementação dele foi levado em consideração todos os experimentos realizados anteriormente, os arquivos gerados irão obedecer os parâmetros mínimos de execução do autômato. A implementação acontece obedecendo os elementos dos autômatos que são:

Células e definição de espaço

Como definido anteriormente na seção 4.2 cada célula é definido de acordo com a escala da imagem carregada, no entanto, de acordo com os experimentos foi definido um tamanho mínimo de célula para a execução real da movimentação do drone utilizado neste trabalha. O tamanho mínimo é de 76cm. Virtualmente cada célula terá seu tamanho em pixel, $10px^2$ cada célula. Foi esse valor considerado nos experimentos aqui realizados de geração de arquivos.

Espaço Celular ou grade de células

Dentro da Simulação ou no autômato celular, as células estão dispostas encadeadas em um plano bidimensional, ou seja é considerado que o drone irá se locomover em um plano bidimensional, como mostra a a figura 4.22

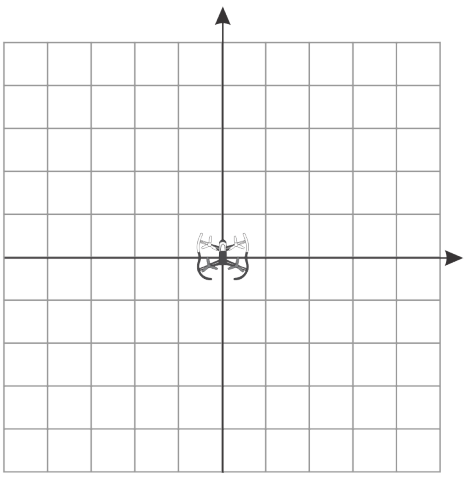

Figura 4.22: *Representação de uma grade com diversas células - Fonte: elaborado pelo autor*

Para a execução do trabalho foi considerada uma grade de 650x600 pixels, onde cada célula representa um espaço de $10px^2$. Ou seja, o espaço está dividido em 3900 células como apresentado na (Fig.: 4.24).

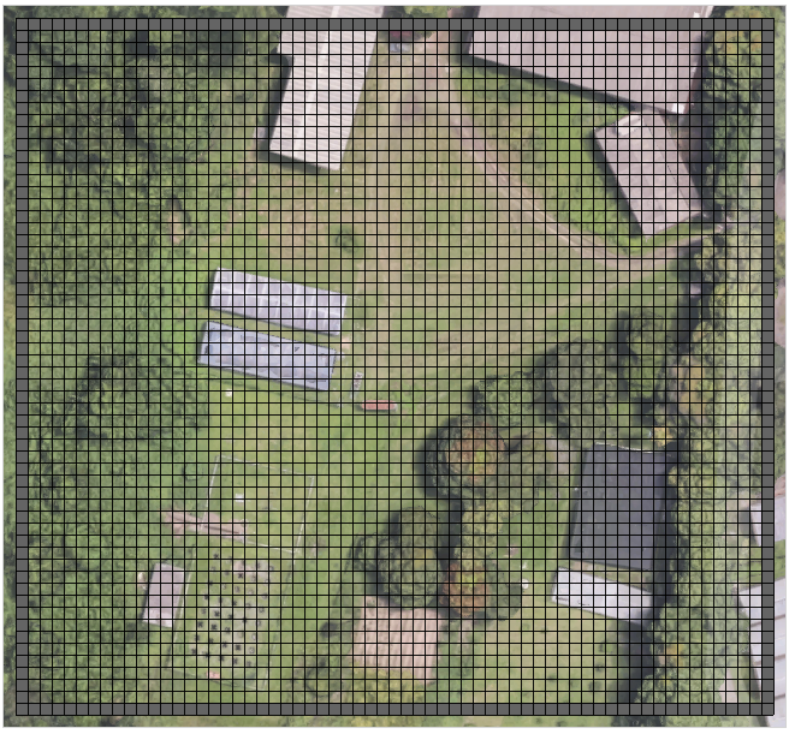

Figura 4.23: *Divisão do espaço celular com células de* $10px^2$ *- Fonte: elaborado pelo autor*

Vizinhança

Conforme mencionado anteriormente, o autômato segue a vizinhança de Von Neumann mostrado na figura 2.5(b). Essa vizinhança foi estabelecida devido ao controle de movimentação do drone, o qual se movimenta para ambos os lados. Porém, para uma movimentação em diagonal, se faz necessário a execução de mais de um comando de forma concorrente, o que demonstra instabilidade do framework utilizado. Por isso, é executado um comando de cada vez com a finalidade de minimizar os efeitos da concorrência dos processos assíncronos.

Bordas ou Limites do Espaço Celular

Como demonstrado em Pascoal (2005), existem vários tipos de bordas para autômatos celulares. O adotado para esta aplicação foi o de limites fixos, ou seja, é atribuído um estado para as células limites da grade, com a finalidade de limitar o espaço físico durante a evolução do autômato. Esse estado é o mesmo utilizado como obstáculos e é mantido a cada interação.

Estados

Os estados são atributos das células que constituem valores assumidos inicialmente e ao longo da evolução do autômato. Estes valores podem ser representados por cores, números, dentre outros tipos de representação. Inicialmente, os estados das células são pre-definidos antes da primeira interação. Assim, os estados são atualizados ao longo do tempo e das interações.

Para esse trabalho foi considerado um conjunto de 5 estados que a célula pode assumir. Sendo o conjunto de estados:

$$\sum = \{0, 1, 2, 3, 4\}$$

onde:

- **0:** estado no qual a célula está livre para navegação;
- **1:** estado em que a célula é borda do espaço celular ou é obstáculo;
- **2:** estado de uma célula já visitada;
- **3:** estado da célula ocupada pelo drone;
- **4:** estado do ponto de partida do drone.

Todos esses estados são representados de forma gráfica em cores distintas pelo simulador.

Regras de Transição

As regras de transições de estados para este trabalho são Regras Probabilísticas. Isto é, existe uma probabilidade de mudança de um estado para outro definida pela regra de um passeio aleatório como descrito na subseção 4.1.1.

Cada célula de estado diferente de 1 na vizinhança de uma célula no estado 3, tem probabilidade de 1/4 de ser ocupada pelo drone. A célula que já foi ocupada pelo drone passa a ter estado 2; As células do estado 1 não podem ser visitadas pelo drone. A cada interação, a célula que está ocupada pelo drone realiza um movimento aleatório, com probabilidade definida:

$$P(x \to x \pm v_j) = \frac{1}{2d}, \qquad \forall_i = 1, ..., d$$

Assim, a cada iteração, o drone assume uma nova posição aleatoriamente de acordo com a probabilidade e possibilidade de visita a uma determinada célula.

4.4 Execução da Geração de Arquivos

Foram executadas diversas simulações com diferentes quantidades de iterações. Em todas as execuções foram definidas os estados iniciais das bordas do espaço celular e o centro do espaço como sendo o ponto de partida do drone.

A execução da simulação gera arquivos de scripts para dois tipos de drone, o estudado aqui e um minidrone, da mesma empresa e que utiliza a mesma API para programação. Foram gerados arquivos para ambos os drones com a finalidade de mostrar a possibilidade de inclusão de outras plataformas de VANTs ao sistema.

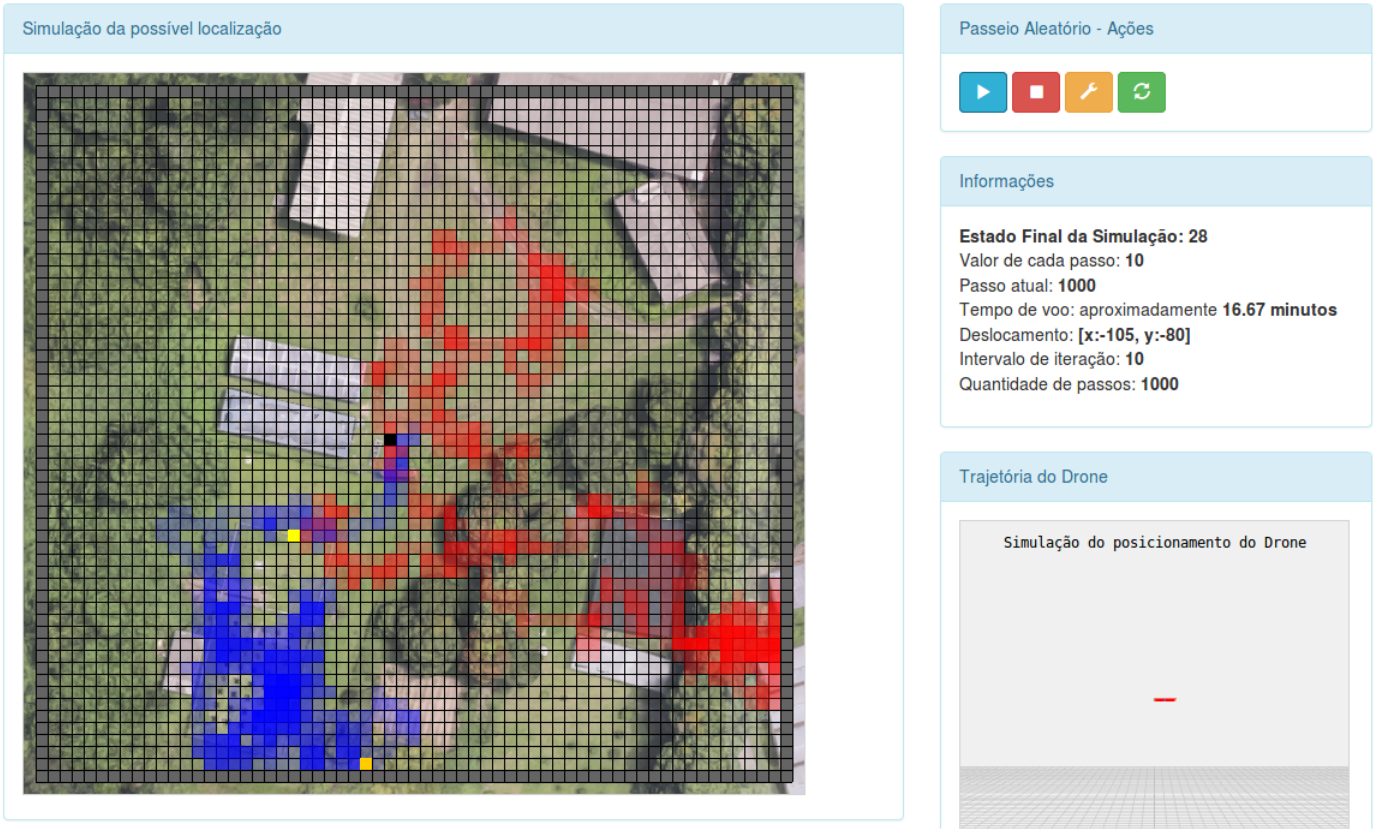

Figura 4.24: *Simulação de evolução do autômato celular para 1000 iterações; Cada cor (vermelha ou azul) representa simulações distintas, que mostra a aleatoriedade. A cada iteração, o drone se move para uma célula diferente - Fonte: elaborado pelo autor*

A execução física dos scripts gerados foi limitada a alguns passos devido a limitação de espaço físico para testes. A seguir, estão listados scripts para 4 iterações do drone Bebop analisado neste trabalho.

Código gerado para o Bebop Drone:

```
"use strict";
var bebop = require("node-bebop");
var power = 20;
var drone = bebop.createClient();
drone.connect(function() {
  drone.takeOff();
  setTimeout(function() {
    drone.left(power);
  }, 7000);
  setTimeout(function() {
    drone.stop();
  }, 9000);
  setTimeout(function() {
    drone.forward(power);
  }, 11000);
  setTimeout(function() {
    drone.stop();
  }, 13000);
```

```
19    setTimeout(function() {
20      drone.right();
21    }, 15000);
22    setTimeout(function() {
23      drone.stop();
24    }, 17000);
25    setTimeout(function() {
26      drone.right();
27    }, 19000);
28    setTimeout(function() {
29      drone.stop();
30    }, 21000);
31    setTimeout(function() {
32      drone.land();
33    }, 23000);
34  });
```

Código 4.4: *Código gerado para o Bebop Drone por 4 iterações*

Capítulo 5

Conclusão e Trabalhos Futuros

Neste capítulo apresentam-se as considerações finais sobre o trabalho desenvolvido nesta dissertação. Na Seção 5.1 apresentam-se as considerações finais e conclusões e na Seção 5.2 algumas propostas para trabalhos futuros.

5.1 Considerações Finais

Neste trabalho foi proposto o desenvolvimento de um modelo computacional baseado em autômatos celulares que fosse capaz de gerar rotas autônomas para VANTs. Era objetivo deste trabalho definir e testar o modelo computacional implementado para a geração de rotas de navegação desses *drones* autônomos que não tivesse a necessidade de utilização de sistemas externos de localização, tais como GPS ou GPRS. Para que fosse alcançado os objetivos desse trabalho, foi analisado uma plataforma comercial de drone, o Parrot Bebop Drone, com a finalidade de gerar arquivos de navegação para esse tipo de veículo.

O VANT analisado se mostrou com robustez suficiente para execução de movimentos programados como rotas. Tomando como base o software de controle do drone disponibilizado pelo fabricante, foi analisado e mostrado que o controle programado é compatível com o disponível para ele, onde nos experimentos se manteve na compatível com relação a precisão de movimentos, em todos os casos as comparação de movimentação entre o software do fabricante e scripts da API Node.js desenvolvidos foram aceitas com 95% de intervalo de confiança. No entanto apresenta instabilidades quando submetido a condições ambientais que possam prejudicar o movimento, tais como, ventos mais fortes que 3 m/s interferem significativamente nos movimentos, fazendo com que aumente o consumo de bateria e o mesmo seja desviado de sua rota programada.

Além dos testes comparativos de analise de viabilidade do uso do Bebop Drone, também foi desenvolvido um modelo computacional baseado em autômatos celulares para simular uma movimentação de possível vistoria. O autômato celular desenvolvido simula um passeio aleatório em um ambiente estruturado de tal forma que configure um autômato celular, ou seja, um espaço denominado grade, ou espaço celular, que é dividido em áreas menores, denominadas células, as quais mudam de estado a cada iteração do autômato. Com essa simulação é gerado uma rota no qual é transcrita para um arquivo de JavaScript possível de ser executado no veículo. Para o ambiente de simulação foi desenvolvido uma aplicação

web que apresenta uma interface com o usuário na qual é representado todo processo de caminhada aleatória do drone.

Assim, foram realizadas simulações com números reduzidos de iterações a fim de observar a qualidade do código gerado e a execução dele em um drone. Em teste se mostrou eficaz e eficiente dentro da média de movimentos apresentada nos resultados, no entanto quanto maior a quantidade de movimentos maior será o erro associado a precisão deles.

Assim, com esse trabalho foram realizados os seguintes objetivos:

- Sistema de gerenciamento de simulações e missões: sistema web de apresentação de simulações de rotas por meio de modelo baseado em autômatos celulares;
- Autômato celular para navegação de robô autônomo: foi definido um autômato celular com espaço definido, estados, regras de transição e iterações discretas para a movimentação do drone utilizado;
- Templates para gerar programas de rotas autônomas: arquivos padrões de scripts de movimentação foram associados ao resultado de iterações do autômato celular;
- Módulo de geração de código: geração de scripts para rotas autônomas são geradas nos padrões dos templates definidos e executados no drone analisado;
- Experimentos visando definição de precisão de movimentos comparando software de controle do fabricante e códigos gerados: foram realizados vários experimentos e comparados sua precisão, em todos os experimentos a média de precisão se mostrou compatível, foram comparadas e com um intervalo de confiança de 95% são estatisticamente iguais.

Por fim, também foram encontrados problemas durante a execução do trabalho, tais como:

- **Execução de processos assíncrona concorrente:** apesar de possibilitar que seja executado uma gama de processos concorrentemente, esse recurso faz com que comandos que sejam executados ao mesmo tempo mostrem instabilidade e possibilidade de erro de execução;
- **Estabilidade em voo:** o drone analisado apesar de robusto, apresenta fragilidade na estabilidade de voo em situações diversas, tais como lentidão no sistema de sensoriamento inercial ocasionando mudança de direção em movimentos e até queda do drone;
- **Limitações de hardware do drone:** apesar da analise desse trabalho ser voltada para software, uma limitação do drone analisado é a falta de sensores de obstáculos, apenas os sensores inerciais não são suficientes para manter o drone estável e em uma trajetória segura.

5.2 Propostas para Trabalhos Futuros

É proposta a continuidade deste projeto levando em consideração a resolução dos pontos fracos e problemas relatados nesse trabalho. A continuidade da pesquisa visa a criação de

uma plataforma de controle autônomo para drones, na qual será possível o controle e analise de dados em tempo de execução de missões. Duas linhas de melhorias podem ser adotadas para trabalhos futuros: a linha de controle de software e integração com IoT, e a linha de execução de movimentos em drones.

Assim é proposto os seguintes pontos como ampliação do trabalho:

- Geração de rotas em tempo de execução com atualização do espaço celular e desvio de obstáculos;
- Utilização de computador embarcado para controle local do drone; como proposta a utilização do Raspberry Pi;
- Implementação de protocolo de comunicação entre drones com a finalidade de desenvolvimento de um Swarm Autônomo;
- Implementação de Swarm de Drones Autônomos;
- Implementação de sistema de controle IoT de atualização em tempo de execução de missões de monitoramento;
- Verificação formal dos processo simulados para garantia de execução dos comandos.

Referências Bibliográficas

Alvané (2014) Tiago Alexandre Gonçalves Alvané. *Navegação de Drones via GPS (Drones Navigation through GPS)*. Tese de Doutorado, Universidade de Aveiro. Citado na pág. xii, 13

Barrera (2010) Alejandra Barrera. *Mobile Robots Navigation*. In-tech ed. Citado na pág. 6

Behring *et al.* (2000) C. Behring, M. Bracho, M. Castro e J. A. Moreno. An Algorithm for Robot Path Planning with Cellular Automata. *Proceedings of the Fourth International Conference on Cellular Automata for Research and Industry: Theoretical and Practical Issues on Cellular Automata*, páginas 11–19. doi: 10.1007/978-1-4471-0709-5. URL http://www.springerlink.com/index/10.1007/978-1-4471-0709-5. Citado na pág. ix, xii, xiii, 15, 18, 19, 20

Cerqueira (2012) Rômulo Guedes Cerqueira. *Navegação de Robôs Móveis Omnidirecionais: Uma abordagem baseada em Aprendizado Simbólico Automático*. Tese de Doutorado, Universidade Federal da Bahia. Citado na pág. 6

Ferreira (2014) Giordano Bruno Santos Ferreira. *Modelos Baseados em Autômatos Celulares para o Planejamento de Caminhos em Robôs Autônomos*. Tese de Doutorado. Citado na pág. ix, xiii, 15, 20

Ilachinski (2001) A Ilachinski. *Cellular Automata: A Discrete Universe*. London: World Scientific Publishing Co. Pte. Ltd. Citado na pág. xii, 11

Leite *et al.* (2017) L. S. Leite, P. C. F. Marques, N. R. F. Silva, N. C. L. Oliveira, C. H. M. Castelletti, H. P. Silva, J. O. Albuquerque e J. L. Lima-Filho. Environmental surveillance for criminal investigations by unmanned aerial vehicles (uavs). *InterForensics - Conferência Internacional de Ciências Forences*, página 162. URL http://interforensics.com/wp-content/uploads/2016/08/INTERFORENSICS_-_LIVRO_DE_RESUMOS.pdf?x33124. Citado na pág. xiii, 27, 28

MACIEL (2012) M. A. MACIEL. A compensação ambiental: instrumento para a implementação do sistema nacional de unidades de conservação. Dissertação de Mestrado, Centro Universitário de Brasília Programa de Mestrado em Direito. Brasilia. Citado na pág. 1

Marchese (1996) Fabio Marchese. Cellular automata in robot path planning. *Advanced Mobile Robot, 1996., Proceedings of ...*. URL http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=551890. Citado na pág. ix, 15, 16, 17

Marchese (2002) Fabio M. Marchese. A directional diffusion algorithm on cellular automata for robot path-planning. *Future Generation Computer Systems*, 18(7):983–994. ISSN 0167739X. doi: 10.1016/S0167-739X(02)00077-8. Citado na pág. ix, 15, 16, 17

Marchese (2008) Fabio M. Marchese. The motion coordination in the c-space-time with a multi-layered mrs architecture. páginas 2522–2527. ISSN 2161-8151. doi: 10.1109/ICAL.2008.4636593. Citado na pág. ix, 15, 16, 17, 18

Marchese (2011) Fabio M. Marchese. Time-invariant motion planner in discretized c-spacetime for mrs. *Multi-Robot Systems, Trends and Development, Dr Toshiyuki Yasuda (Ed.)*, páginas 307–324. doi: 10.5772/13388. Citado na pág. ix, 15, 16, 18

Meyer e Filliat (2003) Jean Arcady Meyer e David Filliat. Map-based navigation in mobile robots: II. A review of map-learning and path-planning strategies. *Cognitive Systems Research*, 4(4):283–317. ISSN 13890417. doi: 10.1016/S1389-0417(03)00007-X. Citado na pág. 5, 7

Microdrones (2017) Microdrones. Drone applications: Aerial photography, monitoring, search rescue etc, 2017. URL <https://www.microdrones.com/en/applications/>. Último acesso em 16/05/2017. Citado na pág. 12

Minetto (2007) E. L. Minetto. *Frameworks para desenvolvimento em PHP*. 1 ed. Citado na pág. 39

NEUMANN e A.W. (1966) J. Von NEUMANN e In: BURKS A.W. Theory of self-reproducing automata, 1966. ISSN 00200271. URL http://linkinghub.elsevier.com/retrieve/pii/0020027169900266. Citado na pág. 10

Parrot (2017) Parrot. Parrot bebop drone. lightweight yet robust quadricopter, 2017. URL <http://global.parrot.com/au/products/bebop-drone/>. Último acesso em 20/05/2017. Citado na pág. 24

Pascoal (2005) F. S. Pascoal. *Sociedade Artificial FIGHT4LIFE: Autômato Celular Modelando a Vida Artificial*. Instituto Nacional de Pesquisas Espaciais, 2005. URL http://abntex.codigolivre.org.br. Relatório Final de Projeto de Iniciação Científica (PBIC, CNPq, INPE). Citado na pág. 42

Sampaio (2007) C. Sampaio. *Guia do Java Enterprise Edittion 5: Desenvolvendo Aplicações Corporativas*. Citado na pág. 40

Shu e Buxton (1995) Chang Shu e Hilary Buxton. Parallel path planning on the distributed array processor. *Parallel Computing*, 21(11):1749–1767. ISSN 01678191. doi: 10.1016/0167-8191(96)80006-8. Citado na pág. ix, 15

Silva (2016) Alberto Rogério e Silva. *HEALTHDRONES - Uma Plataforma em Software para Controle de Veículos Não-Tripulados e Planejamento de Trajetórias Alberto Rogério e Silva*. Tese de Doutorado. Citado na pág. ix, 15, 21

Simões-Pereira (2013) J. M. S. Simões-Pereira. *Grafos e redes: teoria e algoritmos basicos*. Rio de Janeiro: Editora Interciência, 1. ed. Citado na pág. 7, 8

Stochero (2013) Tahiane Stochero. Polêmicos e revolucionários, mais de 200 'drones' voam no país sem regra, 2013. URL <http://g1.globo.com/brasil/noticia/2013/03/polemicos-e-revolucionarios-mais-de-200-drones-voam-no-brasil-sem-regra.html>. Último acesso em 16/05/2017. Citado na pág. 3, 12

Tavakoli *et al.* **(2008)** Yashar Tavakoli, H Haj Seyyed Javadi e Sepideh Adabi. A Cellular Automata Based Algorithm for Path Planning in Multi-Agent Systems with A Common Goal. 8(7):6–10. Citado na pág. ix, 15, 19

Tzionas *et al.* **(1997)** Panagiotis G. Tzionas, Adonios Thanailakis e Philippos G. Tsalides. Collision-free path planning for a diamond-shaped robot using two-dimensional cellular automata. *IEEE Transactions on Robotics and Automation*, 13(2):237–250. ISSN 1042296X. doi: 10.1109/70.563646. Citado na pág. ix, 15, 18

Valdameri e Bachman (2007) A. R. Valdameri e J. Bachman. Webmodeler: uma ferramenta case para modelagem de banco de dados relacional na web, 2007. URL http://www.inf.furb.br/seminco/2007/artigos/01_35559.pdf. Último acesso em 22/07/2017. Citado na pág. 40

Wolfram (1994) Stephen Wolfram. *Cellular automata and complexity: collected papers*. Westview Press. ISBN 0201627167. Citado na pág. 10, 11

APÊNDICE A - Dados dos Experimentos

Tabela de Dados do Experimento descrito na seção 4.2.2

Ponto	X	Y	Dist. Origem	Experimento
1	-45	-16	47.75	Tekeoff - Land
2	-20	12	23.32	Tekeoff - Land
3	-44	-5	44.28	Tekeoff - Land
4	-35	27	44.20	Tekeoff - Land
5	-22	8	23.40	Tekeoff - Land
6	-23	-4	23.34	Tekeoff - Land
7	-33	80	86.53	Tekeoff - Land
8	-35	14	37.69	Tekeoff - Land
9	-38	14	40.49	Tekeoff - Land
10	-45	18	48.46	Tekeoff - Land
11	-38	43	57.38	Tekeoff - Land
12	-26	17	31.06	Tekeoff - Land
13	-35	-2	35.05	Tekeoff - Land
14	-35	19	39.82	Tekeoff - Land
15	-33	19	38.08	Tekeoff - Land
16	-17	8	18.78	Tekeoff - Land
17	-24	3	24.18	Tekeoff - Land
18	-21	13	24.69	Tekeoff - Land
19	-27	22	34.82	Tekeoff - Land
20	-17	20	26.24	Tekeoff - Land
21	-24	28	36.87	Tekeoff - Land
22	-23	25	33.97	Tekeoff - Land
23	-17	28	32.75	Tekeoff - Land
24	0	35	35.00	Tekeoff - Land
25	-43	2	43.04	Tekeoff - Land
26	-38	2	38.05	Tekeoff - Land
27	-13	25	28.17	Tekeoff - Land
28	-17	15	22.67	Tekeoff - Land
29	-21	2	21.10	Tekeoff - Land
30	-33	0	33.00	Tekeoff - Land

Tabela 1: *Dados do Experimento Takeoff - FreeFlight*

Tabela de Dados do Experimento descrito na seção 4.2.3

Pto	X	Y	Dist.	Exp
1	-38	58	69.33	Right-Left-Land
2	-41	69	80.26	Right-Left-Land
3	4	45	45.17	Right-Left-Land
4	15	-14	20.51	Right-Left-Land
5	-43	23	48.76	Right-Left-Land
6	-20	32	37.73	Right-Left-Land
7	-37	5	37.33	Right-Left-Land
8	4	18	18.43	Right-Left-Land
9	-45	20	49.24	Right-Left-Land
10	-44	-2	44.04	Right-Left-Land
11	-11	12	16.27	Right-Left-Land
12	-29	-27	39.62	Right-Left-Land
13	-38	22	43.90	Right-Left-Land
14	-47	3	47.09	Right-Left-Land
15	-45	10	46.09	Right-Left-Land
16	8	-28	29.12	Right-Left-Land
17	-29	-30	41.72	Right-Left-Land
18	-32	-22	38.83	Right-Left-Land
19	12	10	15.62	Right-Left-Land
20	-34	-27	43.41	Right-Left-Land
21	-16	12	20.00	Right-Left-Land
22	6	-30	30.59	Right-Left-Land
23	-28	-13	30.87	Right-Left-Land
24	-7	18	19.31	Right-Left-Land
25	-19	9	21.02	Right-Left-Land
26	15	6	16.15	Right-Left-Land
27	-23	51	55.94	Right-Left-Land
28	-18	40	43.86	Right-Left-Land
29	-30	22	37.20	Right-Left-Land
30	-2	27	27.07	Right-Left-Land

Tabela 2: *Dados do Experimento Takeoff - NodeJS*

Tabela de Dados do Experimento descrito na seção 4.2.5

Pto	X	Y	Dist.	Exp
1	-15	13	19.84	Takeoff - NodeJS
2	-28	16	32.24	Takeoff - NodeJS
3	11	1	11.04	Takeoff - NodeJS
4	6	26	26.68	Takeoff - NodeJS
5	-24	-2	24.08	Takeoff - NodeJS
6	-23	12	25.94	Takeoff - NodeJS
7	6	5	7.81	Takeoff - NodeJS
8	-16	3	16.27	Takeoff - NodeJS
9	-35	10	36.40	Takeoff - NodeJS
10	11	0	11.00	Takeoff - NodeJS
11	2	13	13.15	Takeoff - NodeJS
12	-25	19	31.40	Takeoff - NodeJS
13	-15	2	15.13	Takeoff - NodeJS
14	10	23	25.07	Takeoff - NodeJS
15	7	5	8.60	Takeoff - NodeJS
16	-13	15	19.84	Takeoff - NodeJS
17	-18	26	31.62	Takeoff - NodeJS
18	-16	-5	16.76	Takeoff - NodeJS
19	9	-4	9.84	Takeoff - NodeJS
20	-5	15	15.81	Takeoff - NodeJS
21	-8	21	22.47	Takeoff - NodeJS
22	-3	15	15.29	Takeoff - NodeJS
23	14	32	34.92	Takeoff - NodeJS
24	-19	-10	21.47	Takeoff - NodeJS
25	-13	-5	13.92	Takeoff - NodeJS
26	-18	13	22.20	Takeoff - NodeJS
27	5	18	18.68	Takeoff - NodeJS
28	1	16	16.03	Takeoff - NodeJS
29	7	10	12.20	Takeoff - NodeJS
30	-17	1	17.02	Takeoff - NodeJS

Tabela 3: *Dados do Experimento Movimentação autônoma - NodeJS*

APÊNDICE B - Código do Gerador de Scripts

Código fonte do gerador de código, disponível para download no GitHub https://github.com/HealthDrones

```
1
2  # encoding: utf-8
3  import os
4
5  class geraArquivoRollingSpider(object):
6      """docstring for geraArquivoRollingSpider"""
7      def __init__(self, rota, nome):
8          super(geraArquivoRollingSpider, self).__init__()
9          self.base = ""
10         self.nomeArquivo = nome
11         self.inicioArquivo(rota)
12
13     def inicioArquivo(self, rota):
14         self.base = "'use strict';\n\n"
15         self.base += "var RollingSpider = require('rolling-
               spider');\n"
16         self.base += "var ACTIVE = true;\n"
17         self.base += "var STEPS = 10;\n\n"
18         self.base += "function cooldown() {\n"
19         self.base += "    ACTIVE = false;\n"
20         self.base += "    setTimeout(function () {\n"
21         self.base += "        ACTIVE = true;\n"
22         self.base += "    }, STEPS * 12);\n"
23         self.base += "}"
24         self.base += "if (process.env.UUID) {\n"
25         self.base += "   console.log('Procurando Drone ',
               process.env.UUID);\n"
26         self.base += "}\n"
27         self.base += "var d = new RollingSpider(process.env.
               UUID);\n"
28         self.base += "d.connect(function () {\n"
29         self.base += "  d.setup(function () {\n"
30         self.base += "    console.log('Configured for Rolling
               Spider! ', d.name);\n"
31         self.base += "    d.flatTrim();\n"
32         self.base += "    d.startPing();\n"
33         self.base += "    d.flatTrim();\n"
34         self.base += "    d.on('battery', function () {\n"
```

```
35        self.base += "      console.log('Porcentagem de
              Bateria: ' + d.status.battery + '%');\n"
36        self.base += "      d.signalStrength(function (err,
              val) {\n"
37        self.base += "        console.log('Sinal: ' + val + '
              dBm');\n"
38        self.base += "      });\n"
39        self.base += "    });\n"
40        self.base += "//    d.on('stateChange', function ()
              {\n"
41        self.base += "//      console.log(d.status.flying ?
              '-- flying' : '-- down');\n"
42        self.base += "//    })\n"
43        self.base += "    setTimeout(function () {\n"
44        self.base += "      console.log('ready for flight');\
              n"
45        self.base += "      ACTIVE = true;\n"
46        self.base += "    }, 1000);\n"
47        self.base += "    Comandos();\n"
48        self.base += "  });\n"
49        self.base += "});\n"
50        print "Executou o inicio do arquivo"
51        self.corpoDaRota(rota)
52        return self.base
53
54    def corpoDaRota(self, codigo):
55        self.base += "function Comandos(){\n"
56        self.base += "    d.takeOff(); //Decolagem \n"
57        self.base += "    setTimeout(function () {\n"
58        self.base += "    }, 1000);\n"
59        self.base += codigo
60        self.base += "    setTimeout(function () {\n"
61        #self.base += "      d.emergency();\n"
62        self.base += "      process.stdin.pause();\n"
63        self.base += "      process.exit();\n"
64        self.base += "    }, 30000);\n"
65        self.base += "}\n"
66        self.escreveArquivo()
67        return self.base
68
69    def escreveArquivo(self):
70        arquivo = open('ArquivosGerados/'+self.nomeArquivo+'
              _RollingSpider.js', 'w')
71        arquivo.writelines(self.base)
72        print "Gerou Arquivo!"
73        arquivo.close()
74        #self.executaArquivo()
75
76    def executaArquivo(self):
77        os.system("node ArquivosGerados/"+self.nomeArquivo+".
              js")
78        print "Executou o arquivo"
```

```
79
80  class geraArquivoBebop(object):
81      """docstring for geraArquivoBebop"""
82      def __init__(self, rota, nome):
83          super(geraArquivoBebop, self).__init__()
84          self.base = ""
85          self.nomeArquivo = nome
86          self.inicioArquivo(rota)
87
88      def inicioArquivo(self, rota):
89          self.base = "'use strict';\n\n"
90          self.base += "var bebop = require('node-bebop');\n"
91          self.base += "var ACTIVE = true;\n"
92          self.base += "var STEPS = 10;\n\n"
93          self.base += "var d = bebop.createClient();\n"
94          self.base += "d.connect(function () {\n"
95          self.base += "//Configura  es do GPS\n"
96          self.base += "//drone.GPSSettings.resetHome();\n"
97          self.base += "//drone.WifiSettings.outdoorSetting(1)
              ;\n"
98          self.base += "//drone.on('PositionChanged', function(
              data) {\n"
99          self.base += "//   console.log('Posi   o do Drone...
               Conex o do GPS: ' + data);\n"
100         self.base += "//});\n"
101         self.base += "//Configura a c mera para ficar
              voltada para baixo\n"
102         self.base += "//drone.Camera.orientation(0,-120);\n"
103         self.base += "Comandos();\n"
104         self.base += "});\n"
105         print "Executou o inicio do arquivo"
106         self.corpoDaRota(rota)
107         return self.base
108
109     def corpoDaRota(self, codigo):
110         self.base += "function Comandos(){\n"
111         self.base += "    d.takeOff(); //Decolagem \n"
112         self.base += codigo
113         self.base += "    setTimeout(function () {\n"
114         self.base += "        process.stdin.pause();\n"
115         self.base += "        process.exit();\n"
116         self.base += "    }, 30000);\n"
117         self.base += "}\n"
118         self.escreveArquivo()
119         return self.base
120
121     def escreveArquivo(self):
122         arquivo = open('ArquivosGerados/'+self.nomeArquivo+'
              _Bebop.js', 'w')
123         arquivo.writelines(self.base)
124         print "Gerou Arquivo!"
125         arquivo.close()
```

```
126         #self.executaArquivo()
127 
128     def executaArquivo(self):
129         os.system("node ArquivosGerados/"+self.nomeArquivo+".
              js")
130         print "Executou o arquivo"
```

Código 1: *Código Fonte do Núcleo de Geração de Arquios - arquivo: geraArquivo.py*

```
1  # encoding: utf-8
2  #importando bibliotecas necess rias ao programa
3  import numpy as np
4  from random import randint
5  import math as math
6  import matplotlib.pyplot as plt
7  import matplotlib.animation as animation
8  from geraArquivo import geraArquivoRollingSpider
9  from geraArquivo import geraArquivoBebop
10 
11 class AutomatoCelularPA(object):
12     """docstring for AutomatoCelularPA"""
13     def __init__(self, step, nome):
14         super(AutomatoCelularPA, self).__init__()
15         self.steps = ""
16         self.stepsBebop = ""
17         self.serie = []
18         self.caMonitoring(step, nome)
19 
20     def caMonitoring(self, steps, nome):
21         for i in range(0,steps):
22             passo = randint(1,4)
23             print "N mero aleat rio gerado: "+str(passo)
24             self.serie.append(passo)
25             if(passo == 1):
26                 comando = self.stepForward(i)+str((i+1)*2)+"
                      000);\n"
27                 self.steps += comando
28                 comando = self.stepForward_Bebop(i)+str((i+1)
                      *2)+"000);\n"
29                 self.stepsBebop += comando
30             elif(passo == 2):
31                 comando = self.stepRight(i)+str((i+1)*2)+"000)
                      ;\n"
32                 self.steps += comando
33                 comando = self.stepRight_Bebop(i)+str((i+1)*2)+
                      "000);\n"
34                 self.stepsBebop += comando
35             elif(passo == 3):
36                 comando = self.stepBackward(i)+str((i+1)*2)+"
                      000);\n"
37                 self.steps += comando
38                 comando = self.stepBackward_Bebop(i)+str((i+1)
                      *2)+"000);\n"
```

```
39              self.stepsBebop += comando
40          elif(passo == 4):
41              comando = self.stepLeft(i)+str((i+1)*2)+"000);\
                    n"
42              self.steps += comando
43              comando = self.stepLeft_Bebop(i)+str((i+1)*2)+"
                    000);\n"
44              self.stepsBebop += comando
45
46      self.steps += "     setTimeout(function () {\n"
47      self.steps += "       d.land(); //realizando pouso \n"
48      self.steps += "       //d.emergency(); //Desliga os
            motores \n"
49      self.steps += "     },"+str((steps+1)*2)+"000); \n"
50      self.stepsBebop += "     setTimeout(function () {\n"
51      self.stepsBebop += "       d.land(); //realizando
            pouso \n"
52      self.stepsBebop += "       //d.emergency(); //Desliga
            os motores \n"
53      self.stepsBebop += "     },"+str((steps+1)*2)+"000); \
            n"
54      arquivo = geraArquivoRollingSpider(self.steps, nome)
55      arquivoBebop = geraArquivoBebop(self.stepsBebop, nome
            )
56      print self.serie
57      return None
58
59  def stepRight(self, num):
60      retorno = "     setTimeout(function () {\n"
61      retorno += "        d.tiltRight({ steps: STEPS*2 });
            //numero do passo: "+str(num)+"\n"
62      retorno += "        console.log('Tilt Right');\n"
63      retorno += "     },"
64      return retorno
65
66  def stepLeft(self, num):
67      retorno = "     setTimeout(function () {\n"
68      retorno += "        d.tiltLeft({ steps: STEPS*2 }); //
            numero do passo: "+str(num)+"\n"
69      retorno += "        console.log('Tilt Left');\n"
70      retorno += "     },"
71      return retorno
72
73  def stepForward(self, num):
74      retorno = "     setTimeout(function () {\n"
75      retorno += "        d.forward({ steps: STEPS }); //
            numero do passo: "+str(num)+"\n"
76      retorno += "        console.log('Forward');\n"
77      retorno += "     },"
78      return retorno
79
80  def stepBackward(self, num):
```

```
81          retorno = "     setTimeout(function () {\n"
82          retorno += "        d.backward({ steps: STEPS }); //
              numero do passo: "+str(num)+"\n"
83          retorno += "        console.log('Backward');\n"
84          retorno += "    },"
85          return retorno
86
87      def stepRight_Bebop(self, num):
88          retorno = "     setTimeout(function () {\n"
89          retorno += "        d.stop();\n"
90          retorno += "        d.takePicture();\n"
91          retorno += "        d.on('PositionChanged', function(
              data) {\n"
92          retorno += "            console.log('Imagem Capturada
              em: ' + data);\n"
93          retorno += "            console.log(data);\n"
94          retorno += "        });\n"
95          retorno += "        d.right(STEPS); //numero do passo:
               "+str(num)+"\n"
96          retorno += "    },"
97          return retorno
98
99      def stepLeft_Bebop(self, num):
100         retorno = "     setTimeout(function () {\n"
101         retorno += "        d.stop();\n"
102         retorno += "        d.takePicture();\n"
103         retorno += "        d.on('PositionChanged', function(
              data) {\n"
104         retorno += "            console.log('Imagem Capturada
              em: ' + data);\n"
105         retorno += "            console.log(data);\n"
106         retorno += "        });\n"
107         retorno += "        d.left(STEPS); //numero do passo:
              "+str(num)+"\n"
108         retorno += "    },"
109         return retorno
110
111     def stepForward_Bebop(self, num):
112         retorno = "     setTimeout(function () {\n"
113         retorno += "        d.stop();\n"
114         retorno += "        d.takePicture();\n"
115         retorno += "        d.on('PositionChanged', function(
              data) {\n"
116         retorno += "            console.log('Imagem Capturada
              em: ' + data);\n"
117         retorno += "            console.log(data);\n"
118         retorno += "        });\n"
119         retorno += "        d.forward(STEPS); //numero do
              passo: "+str(num)+"\n"
120         retorno += "    },"
121         return retorno
122
```

```
123     def stepBackward_Bebop(self, num):
124         retorno = "     setTimeout(function () {\n"
125         retorno += "         d.stop();\n"
126         retorno += "         d.takePicture();\n"
127         retorno += "         d.on('PositionChanged', function(
              data) {\n"
128         retorno += "             console.log('Imagem Capturada
              em: ' + data);\n"
129         retorno += "             console.log(data);\n"
130         retorno += "         });\n"
131         retorno += "       d.backward(STEPS); //numero do
              passo: "+str(num)+"\n"
132         retorno += "     },"
133         return retorno
134 
135 if __name__ == '__main__':
136     aplicacao = AutomatoCelularPA(5, "Apresentacao")
```

Código 2: *Código Fonte do Núcleo de Geração de Arquivos - arquivo: CellularAutomata.py*

APÊNDICE C

ANEXO A - Tabela T

TABLE A-3 *t* Distribution: Critical *t* Values

	Area in One Tail				
	0.005	0.01	0.025	0.05	0.10
Degrees of Freedom	Area in Two Tails				
	0.01	0.02	0.05	0.10	0.20
1	**63.657**	31.821	**12.706**	6.314	3.078
2	**9.925**	6.965	**4.303**	2.920	1.886
3	**5.841**	4.541	**3.182**	2.353	1.638
4	**4.604**	3.747	**2.776**	2.132	1.533
5	**4.032**	3.365	**2.571**	2.015	1.476
6	**3.707**	3.143	**2.447**	1.943	1.440
7	**3.499**	2.998	**2.365**	1.895	1.415
8	**3.355**	2.896	**2.306**	1.860	1.397
9	**3.250**	2.821	**2.262**	1.833	1.383
10	**3.169**	2.764	**2.228**	1.812	1.372
11	**3.106**	2.718	**2.201**	1.796	1.363
12	**3.055**	2.681	**2.179**	1.782	1.356
13	**3.012**	2.650	**2.160**	1.771	1.350
14	**2.977**	2.624	**2.145**	1.761	1.345
15	**2.947**	2.602	**2.131**	1.753	1.341
16	**2.921**	2.583	**2.120**	1.746	1.337
17	**2.898**	2.567	**2.110**	1.740	1.333
18	**2.878**	2.552	**2.101**	1.734	1.330
19	**2.861**	2.539	**2.093**	1.729	1.328
20	**2.845**	2.528	**2.086**	1.725	1.325
21	**2.831**	2.518	**2.080**	1.721	1.323
22	**2.819**	2.508	**2.074**	1.717	1.321
23	**2.807**	2.500	**2.069**	1.714	1.319
24	**2.797**	2.492	**2.064**	1.711	1.318
25	**2.787**	2.485	**2.060**	1.708	1.316
26	**2.779**	2.479	**2.056**	1.706	1.315
27	**2.771**	2.473	**2.052**	1.703	1.314
28	**2.763**	2.467	**2.048**	1.701	1.313
29	**2.756**	2.462	**2.045**	1.699	1.311
30	**2.750**	2.457	**2.042**	1.697	1.310
31	**2.744**	2.453	**2.040**	1.696	1.309
32	**2.738**	2.449	**2.037**	1.694	1.309
34	**2.728**	2.441	**2.032**	1.691	1.307
36	**2.719**	2.434	**2.028**	1.688	1.306
38	**2.712**	2.429	**2.024**	1.686	1.304
40	**2.704**	2.423	**2.021**	1.684	1.303
45	**2.690**	2.412	**2.014**	1.679	1.301
50	**2.678**	2.403	**2.009**	1.676	1.299
55	**2.668**	2.396	**2.004**	1.673	1.297
60	**2.660**	2.390	**2.000**	1.671	1.296
65	**2.654**	2.385	**1.997**	1.669	1.295
70	**2.648**	2.381	**1.994**	1.667	1.294
75	**2.643**	2.377	**1.992**	1.665	1.293
80	**2.639**	2.374	**1.990**	1.664	1.292
90	**2.632**	2.368	**1.987**	1.662	1.291
100	**2.626**	2.364	**1.984**	1.660	1.290
200	**2.601**	2.345	**1.972**	1.653	1.286
300	**2.592**	2.339	**1.968**	1.650	1.284
400	**2.588**	2.336	**1.966**	1.649	1.284
500	**2.586**	2.334	**1.965**	1.648	1.283
750	**2.582**	2.331	**1.963**	1.647	1.283
1000	**2.581**	2.330	**1.962**	1.646	1.282
2000	**2.578**	2.328	**1.961**	1.646	1.282
Large	**2.576**	2.326	**1.960**	1.645	1.282