



Universidade Federal da Bahia
Instituto de Matemática e Estatística

Departamento de Ciência da Computação

**SISTEMA DE ORIENTAÇÃO PARA DRONE
BASEADO EM GPS, IMU E BARÔMETRO**

Fábio Machado Costa

TRABALHO DE GRADUAÇÃO

Salvador
28 de fevereiro de 2018

FÁBIO MACHADO COSTA

**SISTEMA DE ORIENTAÇÃO PARA DRONE BASEADO EM GPS,
IMU E BARÔMETRO**

Este Trabalho de Graduação foi apresentado ao Departamento de Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Flávio Moraes de Assis Silva

Salvador

28 de fevereiro de 2018

Sistema de Bibliotecas - UFBA

Costa, Fábio Machado.

Sistema de orientação para drone baseado em GPS, IMU e barômetro /
Fábio Machado Costa – Salvador, 2018.
43p.: il.

Orientador: Prof. Dr. Flávio Moraes de Assis Silva.

Monografia (Graduação) – Universidade Federal da Bahia, Instituto de
Matemática e Estatística, 2018.

1. Drone. 2. Orientação 3D. 3. Evasão de Colisão. I. Silva, Flávio Mo-
rais de Assis. II. Universidade Federal da Bahia. Instituto de Matemática
e Estatística. III Título.

CDD – XXX.XX

CDU – XXX.XX.XXX

TERMO DE APROVAÇÃO

FÁBIO MACHADO COSTA

SISTEMA DE ORIENTAÇÃO PARA DRONE BASEADO EM GPS, IMU E BARÔMETRO

Este Trabalho de Graduação foi julgado adequado à obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Departamento de Ciência da Computação da Universidade Federal da Bahia.

Salvador, 28 de fevereiro de 2018

Prof. Dr. Flávio Moraes de Assis Silva
Orientador

Prof. Dr. Sérgio Gorender
Convidado 1

Prof. Dr. Alírio Santos de Sá
Convidado 2

Dedico esse trabalho à minha Mãe.

AGRADECIMENTOS

Sem dúvida, este espaço é o mais importante do trabalho. Seria impossível concluir essa caminhada sem o apoio das pessoas que estiveram comigo. Mas por onde começar? Aqui a ficha vai caindo que um ciclo de minha vida se fecha, mas a universidade me ensinou, por esses "longos" anos, que a nossa (trans)formação nem começa nela nem precisa terminar nela. Foi um ciclo cheio de desafios e sacrifícios, mas ao mesmo tempo, com muita aprendizagem, conquistas e encontros.

Primeiramente agradeço a Deus. Obrigado meu Pai por não ter deixado faltar amor, perseverança, paz e fé. Agradeço a minha mãe por todos momentos dedicados a mim, pelas palavras, pelo amor, pela amizade e julgo que até a própria vida. Não consigo encontrar palavras para demonstrar todo meu agradecimento. Te amo eternamente maezinha! Ao meu pai por me dar os melhores conselhos e por fazer eu aprender com os meus erros, tem sido o caminho que encontrei para viver de forma verdadeira e justa. Não posso esquecer de mencionar meu irmão e irmã que tanto amo. Falando em amores familiares, agradeço a meus tios, tias, avós, primos e amigos de Amargosa. Ao amor da minha vida, minha gratidão e alegria por estar ao meu lado em todos os momentos, obrigado por tanto amor, pela boa convivência e por me fazer completo. Também agradeço as novas amizades feitas em Salvador e na Austrália durante minha trajetória acadêmica, são muito especiais para mim.

Agradeço aos meus professores da Universidade Federal da Bahia por terem proporcionado uma visão mais abrangente e plural da área de computação. Agradeço especialmente ao Profo. Dr. Flávio Moraes de Assis Silva pela sabedoria e competência com que conduziu as orientações. Seu profissionalismo e competência são exemplos para minha trajetória. As supervisões foram fundamentais para que superasse as dificuldades comumente apresentadas.

"Twice the pride, double the fall."

— (Count Dooku - Star Wars)

RESUMO

À medida que os drones tornam-se mais acessíveis e fáceis de usar, novos campos de pesquisa surgem para descomplicar as atividades manuais feitas por pessoas. Um desses possíveis usos é na inspeção aérea. Porém, independente da atividade, pilotar um drone requer muito treinamento e também exige que o piloto tenha uma visão da localização do drone. Este trabalho apresenta uma proposta de sistema de orientação para uma aplicação que monitora uma área, sendo o voo completamente automatizado sem a necessidade de um piloto. Ao contrário de outros trabalhos baseados no modelo ar.drone 2.0, nossa abordagem dá ênfase no controle, desempenho e segurança na realização da inspeção aérea. Esta abordagem consiste no uso de coordenadas tridimensionais e de dados sensoriais para o deslocamento correto. Além do sensor ultra-sônico para desvio de obstáculo. Nossos experimentos mostram resultados significativos para a vistoria de uma determinada área. Os testes efetuados mostraram que o sistema de orientação viabiliza um voo autônomo com precisão e baseado nos conceitos de navegação. Além disso, a plataforma genérica utilizada no trabalho demonstrou ser bastante relevante, uma vez que possibilita a utilização do sistema de orientação em outros tipos de drone.

Palavras-chave: AR.Drone, Drone, Voo autônomo, IMU, Orientação 3D, Evasão de Colisão, Inspeção aérea.

ABSTRACT

As drones become more accessible and easier to use, new fields of research come up to facilitate the manual activities done by people. One of these possible uses is in aerial inspection. However, regardless of the activity, piloting a drone requires a lot of training and also requires the pilot to have a view of the drone's location. This paper presents a guidance system proposed for an application that monitors an area, and the fully automated flying without a pilot. Unlike other works based on the ar.drone 2.0 model, our approach focuses on control, performance and safety in carrying out the aerial inspection. This approach consists in the use of three-dimensional coordinates and sensory data for correct displacement. In addition to the ultrasonic sensor to avoid collisions. Our experiments show significant results for evaluation of a particular area. The tests performed showed that the guidance system enables an autonomous flight accurately and based on navigation concepts. In addition, the generic platform used in the work proved to be quite relevant, since it makes possible to use the guidance system in other types of drones.

Keywords: AR.Drone, Drone, Autonomous Flight, IMU, 3D Orientation, Collision Avoidance, Aerial Inspection

SUMÁRIO

Capítulo 1—Introdução	1
1.1 Motivação	2
1.2 Objetivo Geral	3
1.3 Objetivos Específicos	3
1.4 Organização do Trabalho	3
Capítulo 2—Arquitetura do Drone	4
2.1 Plataforma do AR.Drone 2.0	4
2.1.1 Controlador de Voo	4
2.1.2 Especificação de Hardware	6
2.1.2.1 Sensores	7
2.1.3 Software	9
2.2 Plataforma Genérica	10
2.2.1 Especificação de Hardware	10
2.2.2 Software	12
2.3 Integração	13
Capítulo 3—Trabalhos Relacionados	16
3.1 Desvio de Obstáculos com Sensor Ultrassônico	16
3.2 Trajetória Autônoma Utilizando o AR.Drone	17
Capítulo 4—Sistema de Orientação baseado em GPS, Giroscópio e Barômetro	20
4.1 O Problema	20
4.2 Escopo da Aplicação de Inspeção Aérea	21
4.3 Implementação do Sistema de Orientação	22
4.3.1 Modelagem	22
4.3.2 Aplicação Principal	23
4.3.3 Captura de Imagem	24
4.3.4 Controle de Velocidade	25
4.3.5 Módulo de Proteção contra Colisão	26
4.3.6 Módulo de Rota	28
4.4 Fórmulas de Navegação	30
4.4.1 Cálculo do Norte Verdadeiro	30
4.4.2 Cálculo do Ângulo de Rotação	32
4.4.3 Cálculo da Distância	34

SUMÁRIO	x
Capítulo 5—Resultados	36
5.1 Experimentos	36
Capítulo 6—Conclusão	39
6.1 Dificuldades Encontradas	40
6.2 Trabalhos Futuros	40

LISTA DE FIGURAS

2.1	Funcionamento dos rotores (Khan, M., 2014)	5
2.2	Quadros de referência e diagrama de forças do quadricóptero. (Sydney, N. et al, 2013)	5
2.3	Eixos de movimentos do quadricóptero.	6
2.4	AR.Drone 2.0 com casco indoor (PARROT, 2018)	7
2.5	Raspberry Pi Model 1 A+ (Raspberry, 2018)	11
2.6	Raspberry Pi 3 B (Raspberry, 2018)	11
2.7	Alimentação do Raspberry	14
2.8	Ligaçāo entre Raspberry e HC-SH04	15
3.1	(a) As quatro direções do drone; (b) Direção dos Sensores (Rambabu, R. et al, 2015)	17
3.2	Exemplos de marcadores (Velez, P. et al, 2015)	18
3.3	(a) Referência linear da trajetória; (b) Função polinomial cúbica; (c) Curva de Bézier (Velez, P. et al, 2015)	18
4.1	Escopo da Aplicação de Inspeção Aérea.	21
4.2	Esquema do sistema de orientação.	23
4.3	Trajetória do desvio de obstáculos.	26
4.4	Aplicativo <i>Compass</i> para referência do giroscópio.	32
4.5	Conceitos básicos de navegação.	33
5.1	Topologia dos experimentos.	36
5.2	(a) Trajetória linear; (b) Menor percurso percorrido; (c) Maior percurso percorrido.	37
5.3	Ferramenta GPS Visualizer	38

LISTA DE TABELAS

2.1	Sistemas Operacionais para o Raspberry PI	12
2.2	Ligações entre GPS e o Raspberry	14
3.1	Situação das limitações em relação ao sistema de localização	19
4.1	Comandos do controle manual	24

LISTA DE ABREVIATURAS E SIGLAS

<i>OpenCV</i>	Open Source Computer Vision Library
<i>PNG</i>	Portable Network Graphics
API	Application Programming Interface
AT	ATTENTION
fps	Frames per second
GPIO	General-purpose input/output
GPS	Global Positioning System
IMU	Inertial Measurement Unit
LaSiD	Laboratório de Sistemas Distribuídos
MEMS	micro electro-mechanical system
<i>PNG</i>	Portable Network Graphics
PP	Polipropileno
RAM	Random Access Memory
rpm	Rotação por minuto
SDK	Software Development Kit
UFBA	Universidade Federal da Bahia
USB	Universal Serial Bus
VATs	Veículos Autônomos não Tripulados
Wi-Fi	Wireless Fidelity

Capítulo

1

INTRODUÇÃO

Os *Veículos Autônomos não Tripulados* (VATs), popularmente conhecidos como drones, tornaram-se capazes de executar diversas atividades devido à evolução tecnológica dos sensores embarcados e dos próprios drones. A cada dia, os VATs provam ser muito úteis em setores como vigilância aérea, sensoriamento remoto, inspeções aéreas, operações de busca e salvamento, entre outros. Os voos acontecem de maneira manual ou autônoma. Voos controlados manualmente acarretam em custos operacionais, por isso, é mais favorável operá-los de forma autônoma. Porém, o voo autônomo apresenta desafios na navegação sobre terreno com elevações (orientação tridimensional) e cenários com obstáculos estáticos e dinâmicos. Dessa forma, os dois principais problemas no voo autônomo estão relacionados com a interação entre o drone e ambiente com o intuito de evitar colisões e como definir a localização e a orientação do drone. Nesse sentido, para a realização do voo autônomo é necessário o desenvolvimento do sistema de orientação que forneça um bom controle, desempenho, habilidade em desviar de obstáculos e segurança na realização de alguma atividade.

O projeto será testado e avaliado na atividade de inspeção aérea. Para isso, uma aplicação de inspeção aérea está sendo desenvolvida com o objetivo de realizar a vistoria de pequenas ou grandes áreas, bem como equipamentos e infraestruturas específicas (Cardoso, P., 2018). Este tipo de atividade sempre representou riscos para os trabalhadores e preocupação para as empresas. Porém a inspeção aérea com drones tem um potencial de ser mais confiável, rápida e econômica, além de reduzir o risco de acidentes de trabalho.

A realização dos voos autônomos ocorreram em um ambiente controlado (voos em ambiente fechados com proteção nas hélices) utilizando o *Ar.drone 2.0*, que é um drone de baixo custo produzido pela empresa francesa *Parrot SA*. Inicialmente, o objetivo foi avaliar as capacidades do drone na realização de voos autônomos utilizando o software *SDK 2.0 (Software Development Kit)* disponível pela própria empresa (PARROT, 2018). Os planos de voo foram criados usando a biblioteca chamada de *PS-Drone*, escrita em *Python* e disponibilizada em (GRAAFF, J., 2012), com o intuito de obter o acesso aos

sensores e movimentos do drone. Os resultados iniciais indicaram limitações no movimento, na segurança e na necessidade do desvio de obstáculo para aplicação de inspeção aérea.

A primeira limitação encontrada é na simplicidade dos movimentos implementados pela biblioteca *PS-Drone*. O plano de voo é pré-programado e baseado somente na escolha da direção, tempo de execução e velocidade do movimento. Assim, por exemplo, a ação do vento modifica o plano de voo sem que haja uma correção, provocando movimento incorreto. Além disso, percorrer uma área sem uma posição de referência exige cálculos específicos que o drone não é capaz de realizar sem ajuda de sensores. Por exemplo, calcular a distância do deslocamento até o início da área a ser inspecionada. Dessa forma, o sistema de orientação do drone precisa desempenhar movimentos complexos baseados em sensores de orientação, altitude e posição para uma navegação tridimensional.

Outro ponto crítico para a aplicação de inspeção aérea está relacionado com a falta de segurança do projeto *Ar.drone 2.0*. Segundo Pleban, J. et al (2014), o fato de o drone não possuir recursos de segurança torna-o alvo de ataques. Dessa forma, um potencial atacante poderia facilmente obter controle do drone de maneira a comprometer a missão. Para minimizar esse problema, é preciso fornecer um módulo que possibilite apoderar-se do drone durante a realização da inspeção aérea para obter o controle, realizar um pouso de emergência e evitar que outra pessoa tenha acesso ao drone.

Por fim, no momento da inspeção de uma determinada área, é importante que o drone tenha a capacidade de interagir com o ambiente de forma a evitar colisões. Para isso, é necessário inserir um módulo de evasão no sistema de orientação do drone, utilizando sensores de obstáculo que identifiquem objetos durante o percurso.

Este trabalho procura minimizar as limitações no movimento, segurança e evasão de obstáculo para que a aplicação de inspeção aérea tenha as funcionalidades básicas para um bom funcionamento.

1.1 MOTIVAÇÃO

Nas pesquisas que foram realizadas durante esta monografia, não foi encontrada nenhuma aplicação, solução ou proposta que utilizasse tecnologias de código aberto (*opensource*) e arquitetura genérica para inspeção aérea. Entretanto, o Laboratório de Sistemas Distribuídos (LaSiD)¹, do qual faço parte, desenvolveu um sistema de controle de rota baseado somente em dados do GPS, descrito na monografia (Martins, M., 2017). Porém, o sistema de controle de rota apresentou consideráveis limitações (detalhe na seção 3.1) para a aplicação de inspeção aérea.

O presente trabalho tem como motivações principais: (i) resolver as limitações do controle de rota baseado em GPS; (ii) construir componentes para uma primeira versão de uma aplicação de inspeção aérea.

¹laboratório de pesquisa do Departamento de Ciência da Computação da Universidade Federal da Bahia (UFBA)

1.2 OBJETIVO GERAL

Este trabalho tem como objetivo desenvolver um sistema de orientação para drone que possa operar de forma robusta e autônoma em ambientes reais, servindo como base para a aplicação de inspeção aérea. Um dos principais pré-requisitos para a aplicação de inspeção aérea é a capacidade de o drone conhecer sua localização e movimentos a serem realizados para concluir a missão de percorrer uma área. Uma vez que o drone esteja no percurso correto, o mesmo deve interagir com o ambiente para desviar de obstáculos. A aplicação de inspeção aérea consiste na integração de outros trabalhos feitos por pesquisadores do LaSiD.

1.3 OBJETIVOS ESPECÍFICOS

Desenvolver procedimentos a serem embarcados para que o drone possa:

- Realizar movimento com precisão nos 3 eixos;
- Melhorar o desempenho do percurso;
- Realizar pouso de emergência;
- Obter controle manual durante a missão;
- Desviar de obstáculo.

1.4 ORGANIZAÇÃO DO TRABALHO

Este trabalho está estruturado como se segue: No Capítulo 2 são apresentados os conceitos envolvendo a arquitetura utilizada e sobre demais conteúdos importantes para o entendimento do trabalho. Um levantamento das limitações existentes no sistema de rota baseado em GPS bem como propostas utilizadas em outras pesquisas serão apresentadas no Capítulo 3. Uma descrição do escopo e método utilizado nesse trabalho bem como do processo de desenvolvimento do sistema de orientação e desvio de obstáculo serão mostradas no Capítulo 4. O Capítulo 5 apresentará os resultados obtidos e uma avaliação sobre eles. A conclusão e direcionamento para trabalhos futuros estão no Capítulo 6.

Capítulo

2

ARQUITETURA DO DRONE

Nesta seção, os componentes físicos, o controle de voo e o software integrado do *AR.Drone 2.0* são avaliados, da mesma forma, é avaliada a proposta de plataforma genérica para acoplar outros sensores e controlar diferentes drones. Por fim, é descrito como é feita a integração das plataformas.

2.1 PLATAFORMA DO AR.DRONE 2.0

Como informado no capítulo anterior, o drone utilizado foi o *AR.Drone 2.0*, projetado pela empresa *Parrot SA*. As principais vantagens deste drone é o preço acessível e a quantidade de sensores embarcados. O *AR.Drone 2.0* é equipado com um sensor ultrassônico e uma *Inertial Measurement Unit* (IMU), detalhado na seção 2.1.2.1, que mede movimentos e acelerações ao longo de todos os eixos. Além disso, também é equipado com uma câmera frontal e outra câmera localizada abaixo do drone, que fornece imagens para a inspeção aérea. A comunicação é feita através de comandos AT, abreviatura de *ATTENTION*, utilizando conexão Wi-Fi. O controle do drone é feito através de uma *Application Programming Interface* (API), que permite não somente realizar o controle básico, mas também o desenvolvimento de aplicações inteligentes através de voos autônomos (PARROT, 2018).

2.1.1 Controlador de Voo

Entender o controle de voo do *AR.Drone 2.0* é fundamental para a realização de qualquer experimento. Para um voo autônomo, por exemplo, é necessário entender as forças atuantes no drone, uma vez que qualquer instabilidade tornará o voo impraticável. Nesse sentido, a estrutura mecânica do drone do tipo quadricóptero consiste em quatro rotores ligados a uma estrutura física. Cada par de rotores opostos (par 1, 4 e par 2, 3 da figura 2.1) gira na mesma direção. Um par está girando no sentido horário e o outro no sentido anti-horário (Khan, M., 2014).

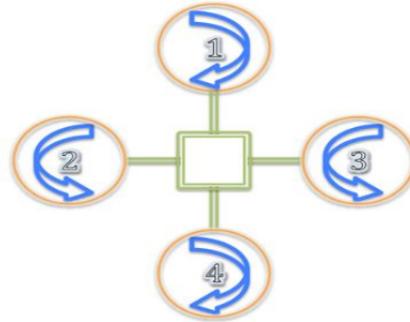


Figura 2.1 Funcionamento dos rotores (Khan, M., 2014)

As forças estão representadas no diagrama da figura 2.2 (Sydney, N. et al, 2013). Cada rotor produz um impulso T e torque τ em torno do seu centro de rotação, bem como uma força de arrasto D , que é oposta ao sentido do drone. Os impulsos T_1 , T_2 , T_3 e T_4 são forças que são geradas pela expulsão ou aceleração da massa do drone em uma direção. O torque τ é a tendência de uma força para girar um objeto em torno de um eixo. A força de arrasto D é a força que se opõe ao movimento do drone através do ar. Juntos, os rotores devem gerar impulso vertical suficiente para permanecer no ar, o que é indicado pela força de gravidade mg na direção W .

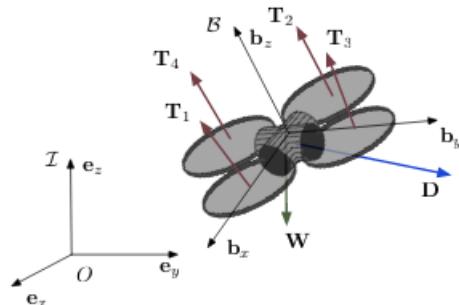


Figura 2.2 Quadros de referência e diagrama de forças do quadricóptero. (Sydney, N. et al, 2013)

Para realizar os movimentos, o drone depende do diferencial entre o torque e o impulso. Analisando a figura 2.3, temos que o *pitch*, *roll* e *yaw* são usados na dinâmica do voo para indicar os ângulos de rotação em três dimensões sobre o centro de massa do drone. O *pitch* é a rotação que efetua o movimento para frente e para trás. Semelhante ao *pitch*, o *roll* realiza os movimentos para direita e esquerda. Por fim, o movimento *yaw* é responsável por fazer o drone girar para esquerda e direita, sem necessidade de inclinação, pois uma variação da velocidade angular de cada par de rotor produz uma aceleração angular em relação ao eixo de *yaw*. Agora se o impulso total for mantido constante, a velocidade

vertical permanece inalterada, o que caracteriza um movimento denominado de *hover*, ou seja, o drone mantém-se em modo estacionário. O movimento na vertical, conhecido como *throttle*, é produzido aumentando ou diminuindo o impulso de cada motor pela mesma quantidade, de modo que o impulso total muda, mas a diferença de torque total permanece zero.

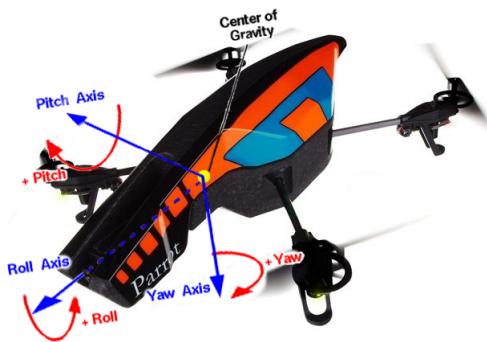


Figura 2.3 Eixos de movimentos do quadricóptero.

2.1.2 Especificação de Hardware

A estrutura do *AR.Drone* (Figura 2.4) é composta por tubo de fibra de carbono e plástico de alta resistência. Além disso, utilizou-se uma espuma de polipropileno (PP) na construção do casco de proteção, que é durável, leve e reciclável. Há dois tipos de cascos: o casco com proteção nas hélices (*indoor*) e o casco sem proteção nas hélices (*outdoor*). As hélices são alimentadas por quatro rotores sem escovas com potência de 15 Watts e girando a 35.000 rpm. A energia é fornecida por uma bateria de polímero de lítio com uma capacidade de 1000mAh, o que permite um tempo de vôo de aproximadamente 10 minutos.

O *AR.Drone* possui um computador interno com um processador ARM9 de 468MHz e 128MB de RAM, executando um sistema operacional Linux. Um conector USB está incluso para fins de atualização de software, gravação de imagens e para acoplamento de dispositivos (por exemplo, sensor GPS). Uma placa de rede sem fio 802.11g integrada oferece conectividade entre o drone e um dispositivo externo. Porém, não existe um controle remoto na aquisição do *AR.Drone*. Em vez disso, um dispositivo com WiFi pode ser usado para controlar o *AR.Drone*. Nesse sentido, é possível controlar o *AR.Drone* a partir de qualquer dispositivo que tenha suporte ao software *SDK 2.0* (PARROT, 2018). Mais informação pode ser encontrada na seção 2.1.3.



Figura 2.4 AR.Drone 2.0 com casco indoor (PARROT, 2018)

2.1.2.1 Sensores

Apesar de o *AR.Drone* ter sido projetado para ser um drone de baixo custo, a diversidade de sensores encontrada no drone permite realizar medição de vários aspectos do ambiente. A principal tarefa dos sensores é proporcionar, de forma automática, informações para que o controlador de voo realize uma boa estabilidade durante o voo. Os sensores encontrados no *AR.Drone* são: ultrassônico, barômetro e o pacote *Inertial Measurement Unit* (IMU), que é composto por acelerômetro, magnetômetro e um giroscópio.

Inertial Measurement Unit

A unidade de medição de inércia, traduzida do inglês *Inertial Measurement Unit* (IMU), fornece 6 (seis) medidas de graus sendo 2 (duas) medidas para cada eixo: *pitch*, *roll* e *yaw*. Essas medidas são usadas para a estabilidade automática e controle da direção. A IMU é um sistema microeletromecânico (traduzido do inglês *micro electro-mechanical system - MEMS*), projetado como pacotes de circuito integrado que contêm um acelerômetro, magnetômetro e um giroscópio (Son, Y. et al, 2015).

O IMU do *AR.Drone* utiliza o modelo de **acelerômetro** chamado de BMA150, fabricado pela Bosch Sensortec (Bosh, 2018). Segundo Cheeman, P. et al (2016), um acelerômetro produz uma força de aceleração baseada no fenômeno de que o peso (observado) de uma massa muda durante a aceleração. O acelerômetro tem três eixos perpendiculares e a medida da aceleração é feita na direção de cada eixo. Como cada ponto na superfície da Terra está acelerando em relação ao drone é preciso obter a aceleração em relação ao movimento da Terra. O deslocamento do drone em relação à gravidade deve ser subtraído e corrigido para os efeitos causados pela rotação da Terra em relação ao drone. Uma vez que o acelerômetro consegue medir a gravidade, ele pode ser usado como um sensor de inclinação.

O funcionamento do giroscópio parte da idéia básica de medir a velocidade angular em

graus por segundo, com base nos princípios do momento angular. Para estimar o ângulo absoluto Θ , o sinal de velocidade angular Ω precisa ser integrado em relação ao tempo. Normalmente, os giroscópios usam uma roda giratória. No entanto, dispositivos pequenos, como o *AR.Drone*, não podem ser beneficiados com tal roda giratória. Em vez disso, um pequeno giroscópio (MEMS) é utilizado nesses tipos de dispositivos. Basicamente, um giroscópio no MEMS consiste de um prato, chamada de massa de prova, que vibra (oscila), como no acelerômetro. Porém, quando o drone gira, o prato é deslocado nas direções x, y e z pelas forças de Coriolis (Domelen, D., 2018). Um processador detecta o deslocamento do prato através de placas de capacitores localizadas debaixo do prato, bem como capacitores localizados nas bordas da embalagem do MEMS.

O modelo de **giroscópio** adotado pelo *AR.Drone* foi o IDG-500 Dual-Axis (InvenSense, 2018), que tem duas configurações: para os movimentos de alta velocidade (alcance: $500^\circ/\text{s}$, sensibilidade: $2,0 \text{ mV}^\circ/\text{s}$) ou movimentos precisos de menor velocidade (alcance: $110^\circ/\text{s}$, sensibilidade: $9,1 \text{ mV}^\circ/\text{s}$). Para a orientação de *yaw*, a medição é feita com um giroscópio de alta precisão, o Epson Toyocom XV-3500CB (Sensing Device, 2018). Este sensor possui um alcance de $100^\circ/\text{s}$ e uma sensibilidade de $0.67\text{mV}^\circ/\text{s}$. Dessa forma, o *AR.Drone* possui um giroscópio com 2 (dois) eixos no *pitch* e *roll* e outro giroscópio somente para o eixo *yaw*.

O **magnetômetro** é um sensor usado para descobrir a direção ou a força do campo magnético. Dessa forma, existem dois tipos básicos de magnetômetro: os escalares, que medem a força do campo magnético; e os vetoriais, que medem a direção do campo magnético. No caso do *AR.Drone*, o magnetômetro é usado para descobrir a direção em que o drone está voando. Isso é possível porque a presença do campo magnético é percebida através da desordem da corrente elétrica, pois os elétrons da corrente se acumulam em um dos lados do material condutor. Assim, o movimento dos elétrons é usado para determinar a direção do campo magnético.

Ultra-som como altímetro

O sensor ultrassônico está ligado na parte inferior do *AR.Drone* e aponta para baixo para medir a distância ao chão. Um sinal de ondas ultrassônicas é transmitido para o chão e reflete o som de volta ao sensor. O sistema mede o tempo t para o eco retornar ao sensor e calcula a distância d ao alvo usando a velocidade do som:

$$d = \frac{c \cdot t}{2}$$

onde c é a velocidade do som no ar, $c \approx 343\text{m/s}$.

As medidas de alcance ultrassônico têm algumas desvantagens fundamentais que limitam a utilização. Essas desvantagens são inerentes ao princípio de um sensor ultrassônico e seus comprimentos de onda comumente usados. *Parrot SA* (2018) descreve algumas dessas desvantagens no contexto de evasão de obstáculos. Uma limitação é a pequena faixa em que o sensor pode operar. O sensor ultrassônico no AR.Drone tem uma faixa

efetiva de aproximadamente 20cm a 6m. Outra limitação é que o sensor ultrassônico não consegue obter informações direcionais precisas sobre objetos. O som se propaga de forma semelhante a um cone, onde os ângulos de abertura são geralmente entre 20 a 40 graus. Devido a esta propriedade, o sensor adquire regiões inteiras de profundidade constante em vez de pontos discretos de profundidade. Assim, o sensor ultrassônico só pode dizer que existe um objeto na distância medida em algum lugar dentro do cone. Além disso, o piso não é sempre perpendicular à orientação do sensor ultrassônico, o que pode influenciar nas medidas de altitude.

Barômetro como altímetro

O barômetro é um sensor usado para determinar altas altitudes. Próximo ao solo, o sensor ultrassônico funciona com mais precisão do que o barômetro. Nesse sentido, o barômetro do *AR.Drone* é projetado para trabalhar em conjunto com o sensor ultrassônico. Ou seja, quando o drone voa acima de 6 metros, o barômetro auxilia na medição da altitude, que pode atingir 50 metros de altura acima do solo.

Câmera

O *AR.Drone* está equipado com duas câmeras: uma câmera frontal e uma câmera inferior. Ambas as câmeras suportam transmissão de vídeo ao vivo a 15 quadros por segundo. A câmera frontal tem uma resolução de 720p 30fps e um campo de visão de 92°. A câmera inferior possui uma resolução de QVGA e um campo de visão de 64°. A frequência de vídeo desta câmera é de 60fps. Apesar da alta frequência, os quadros são transmitidos a 15 quadros por segundo. A câmera inferior desempenha um papel central na inteligência do *AR.Drone*, pois é usada para estabilização horizontal e para estimar a velocidade do drone.

2.1.3 Software

A *AR.Drone API* é um projeto implementado no *Software Development Kit* (SDK), ou simplesmente Kit de Desenvolvimento de Software, em português. A API é utilizada como referência para o desenvolvimento de aplicativos para o *AR.Drone*. O projeto foi desenvolvido para ser multiplataforma e tem uma excelente documentação.

A comunicação com o *AR.Drone* é feita através de quatro portas de comunicação:

1. **Porta 5554:** Informações sobre o drone como status, altitude e velocidade são consultadas através dessa comunicação. Esta informação é enviada pelo drone para o cliente com uma frequência de aproximadamente 30 (modo de demonstração) ou 200 vezes por segundo;
2. **Porta 5555:** O fluxo de vídeo é enviado pelo drone para o dispositivo cliente. As imagens desse fluxo de vídeo são decodificadas usando os decodificadores (*codecs*) incluídos no SDK;

3. **Porta 5556:** Controle e configuração do drone através dos comandos AT;
4. **Porta 5559:** Os dados críticos são transmitidos através de um canal denominado porta de controle. Este canal é usado para configurar ou recuperar dados de ajuste padrão do drone.

2.2 PLATAFORMA GENÉRICA

A plataforma genérica consiste em um conjunto de soluções de *hardware* e *software*, que permite o desenvolvimento das aplicações independente de um modelo específico de drone. Para desenvolver essa plataforma genérica buscou-se uma arquitetura que utilizasse padrões abertos, ou seja, que permitisse a implementação e o livre acesso. A primeira escolha foi utilizar um *Raspberry Pi*, projetado para ser um computador de pequeno porte, baixo custo, com habilidades de programação e integração de outros hardware (Raspberry, 2018). Em relação ao software, a escolha foi utilizar tecnologias de código aberto (*opensource*), devido à qualidade técnica do sistema GNU/Linux, além do grande número de desenvolvedores e documentações.

2.2.1 Especificação de Hardware

Os elementos de hardware utilizados na plataforma genérica e suas funcionalidades serão abordados a seguir.

Raspberry

O *Raspberry Pi* é uma pequena placa de computador, projetada no Reino Unido pela *Fundação Raspberry PI*, com o objetivo de estimular o ensino de informática nas escolas. Geralmente, os modelos medem 85 x 54 mm, exceto o modelo Zero. Apesar de existir uma grande variedade de modelos, foram avaliados somente dois modelos:

- **Raspberry Pi Model 1 A+**

Este modelo foi lançado em novembro de 2014 e é a evolução do modelo A. Com este projeto conseguiu-se reduzir o peso ao eliminar a porta *Ethernet* e ter apenas uma porta USB. Graças a isso também se reduziu significativamente o consumo de energia (cerca de 30%) em relação ao modelo B+, descrito em seguida. Ele também possui uma porta microSD e um GPIO de 40 pinos. Foram identificadas duas desvantagens: não tem suporte a rede sem fio e tem apenas 256MB de RAM.



Figura 2.5 Raspberry Pi Model 1 A+ (Raspberry, 2018)

- **Raspberry Pi 3 B**

Este modelo chegou ao mercado um ano após o Raspberry Pi 2. Esta é a evolução mais importante em termos de poder de processamento e possibilidade de conectividades, uma vez que incorpora placas Wi-Fi e Bluetooth. Possui um novo processador ARM Cortex-A53 de 64 bits com quatro núcleos operando a 1,2 GHz, que pode ser 10 vezes mais rápido do que o modelo 1.



Figura 2.6 Raspberry Pi 3 B (Raspberry, 2018)

O modelo *Raspberry Pi 3 B* foi escolhido devido à capacidade de processamento de imagem e vídeo, além de possuir conectividade sem fio já embarcada, uma vez que a aplicação de inspeção aérea necessita de tais características.

Módulo GPS GY-NEO6MV2

O *GPS GY-NEO6MV2* é um módulo simples e de fácil aplicação. O módulo utiliza comunicação serial, permitindo a utilização em diversos tipos de equipamentos. Além disso, o módulo possui antena embutida e a corrente de operação do conjunto é de apenas 45mA, tornando o módulo *GPS GY-NEO6MV2* uma opção econômica para projeto (UBLOX, 2018).

Sensor Ultrassônico HC-SR04

HC-SR04 é um módulo ultrassônico usado para medir a distância entre o sensor e algum objeto. Ele pode ser alimentado a partir de uma fonte de alimentação de 5V, o que é compatível com o *Raspberry*. ELEC (2018) descreve o módulo *HC-SR04* como sendo um sensor ultrassônico com precisão de 2cm a 400cm. Porém, os testes realizados identificaram um limite no alcance de, no máximo, 200cm.

2.2.2 Software

Sistema Operacional

Devido à popularidade do *Raspberry Pi*, não houve dificuldade em escolher um sistema operacional para o projeto. Existem diversos tipos de sistemas operacionais e os mesmos podem ser classificados de acordo com as necessidades. A seguir, os sistemas operacionais mais relevantes foram classificados em dois grupos: oficiais, o sistema operacional é disponibilizado no site do *Raspberry Pi*; não oficiais, é disponibilizado a partir do site da empresa do sistema operacional.

Tabela 2.1 Sistemas Operacionais para o Raspberry PI

Oficiais	Não Oficiais
Raspbian	ARCH Linux
Pidora	Occidentalis
OSMC	OpenSUSE
OPENELEC	Ubuntu 14.04
RISC OS	Windows 10

Para este projeto, foi usado o *Raspbian*, pois é gratuito e baseado no *Debian*, que é um sistema operacional seguro e especialmente conhecido pela gerenciamento fácil e rápido dos programas. Além disso, é considerado como o sistema operacional mais otimizado para o hardware do *Raspberry Pi*. Raspbian (2018) permite instalar mais de 35.000 pacotes que fornecem programas básicos e utilitários. Este sistema operacional foi concluído em junho de 2012. No entanto, seu desenvolvimento ainda está ativo com ênfase especial na melhoria da estabilidade e na atualização dos pacotes que já estão disponíveis. Vale salientar que a linguagem de desenvolvimento, *Python*, adotada para o projeto, já vem instalada no *Raspbian* (veja em 2.2.2).

Python

Para o projeto em questão, foi escolhida a linguagem *Python* por vários motivos. Primeiro, encontrou-se uma biblioteca chamada *PS-Drone* (detalhe em 2.2.2), responsável pela comunicação do *Raspberry* com o *AR.Drone*. Em segundo lugar, o número de bibliotecas disponíveis foi muito relevante para o desenvolvimento do projeto. Por fim, devido à familiaridade dos integrantes com a linguagem. Vale salientar que a linguagem adotada já vem instalada no *Raspberry*.

Python é uma linguagem de programação interpretada e de programação multiparadigma, uma vez que é capaz de suportar orientação a objetos, programação imperativa e programação funcional (Python, 2018).

PS-Drone

PS-Drone é uma biblioteca completa, escrita em *Python*, para controlar o *AR.Drone 2.0*. Essa biblioteca, foi projetada para oferecer um conjunto completo de movimentos e leitura de sensores do AR.Drone 2.0. GRAAFF, J. (2012) disponibiliza uma documentação contendo uma lista completa dos comandos e uma descrição detalhada dos dados dos sensores.

OpenCV

OpenCV é a principal biblioteca de código aberto (*opensource*) utilizada no âmbito acadêmico e comercial para processamento de imagem e aprendizagem de máquina em aplicações de tempo real. *OpenCV* é liberado sob a licença BSD e dispõe de interfaces *C++*, *C*, *Python* e *Java* e suporta diversos sistemas operacionais (OpenCV team, 2018). Possui vários algoritmos implementados que podem ser utilizados em várias atividades, como: montagem de mapas, detecção de movimentos, orientação robótica, inspeção de área e entre outras.

2.3 INTEGRAÇÃO

A integração do *AR.Drone* com o *Raspberry* é essencial para suprir as exigências da aplicação de inspeção aérea. Por meio da integração é possível controlar os movimentos do drone de maneira autônoma, realizar processamento de imagens, monitorar os sensores, implementar segurança nos voos, entre outras inúmeras atividades. Conforme ilustrada na figura 2.7, o *Raspberry* é alimentado através da porta USB disponibilizada no *AR.Drone* e a comunicação entre as plataformas é feita via WiFi.

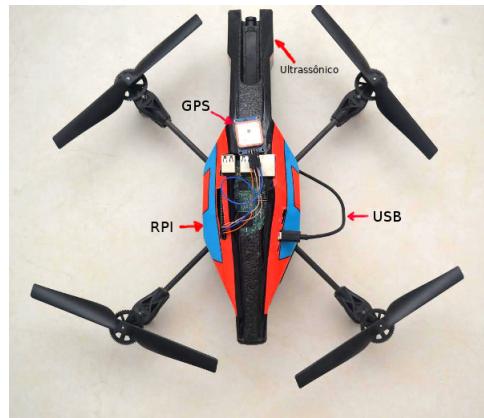


Figura 2.7 Alimentação do Raspberry

A comunicação entre Raspberry e o módulo GPS GY-NEO6MV2 é feita através da porta serial, pois a adoção desse tipo de comunicação apresenta mais economia de energia do que as portas USB. O módulo GPS GY-NEO6MV2 vem com 4 conexões: RX, TX, VCC e GND, que é bastante fácil de incorporar com o uso de *jumpers*. A tabela 2.2 mostra as ligações corretas entre os pinos do GPS com a interface GPIO do *Raspberry*.

Tabela 2.2 Ligações entre GPS e o Raspberry

Pinos do GPIO	Pinos do GPS
4 – 5v	VIN/VCC
6 – GND	GND
8 – TX	RX
10 – RX	TX

A integração entre *Raspberry* e Sensor Ultrassônico HC-SR04 é um pouco delicada. Este sensor produz uma corrente de 5 Volts que é incompatível com o *Raspberry* e, para não danificá-lo, é preciso converter a corrente de 5 Volts para 3.3 Volts através da utilização de resistores, conforme figura 2.8.

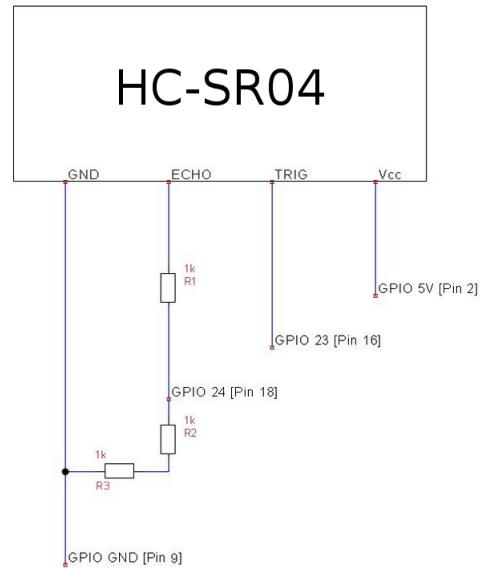


Figura 2.8 Ligação entre Raspberry e HC-SH04

TRABALHOS RELACIONADOS

Dois problemas importantes para um voo autônomo são a localização do drone e o mapeamento do ambiente. Esses problemas surgem quando o drone não tem conhecimento dos objetos e nem da sua posição no ambiente. Nesse sentido, vários pesquisadores estão se concentrando na busca de soluções que ofereçam um bom equilíbrio em termos de tamanho, precisão e consumo de energia, visto que os sensores de maior precisão não são atraentes para pequenos drones, pois são maiores e exigem um grande consumo de energia. Quase todas as publicações relacionadas a esta pesquisa descrevem o problema de localização e mapeamento do ambiente separadamente. Além disso, as publicações não utilizam uma arquitetura genérica conforme proposto neste trabalho. Uma exceção é a pesquisa de Martins, M. (2017), que apesar de utilizar o *Raspberry PI*, apresenta algumas limitações (detalhe na tabela 3.1) no âmbito da aplicação de inspeção aérea.

3.1 DESVIO DE OBSTÁCULOS COM SENSOR ULTRASSÔNICO

Vários trabalhos apresentam técnicas de desvio de obstáculos utilizando um sensor ultrassônico, por exemplo (Pratama, Y. et al, 2017) e (Rambabu, R. et al, 2015). Devido ao tamanho reduzido do sensor ultrassônico, a maioria das pesquisas utiliza vários sensores em conjunto para obter um melhor resultado. Estes sensores fornecem informações tanto para estabilidade na altitude como para detecção de obstáculos. Segundo o meu conhecimento, nenhuma publicação aborda o problema do desvio de obstáculo usando um único sensor ultrassônico. Geralmente, para obter um melhor desempenho utilizam-se vários sensores. Rambabu, R. et al (2015) define a motivação de utilizar vários sensores devido às limitações de cada tipo de sensor. Por exemplo, o sensor de infra-vermelho é influenciado pela intensidade da luz do ambiente e pela reflexão da superfície dos objetos medidos, enquanto que o sensor ultrassônico é afetado pelo tipo de material do objeto medido, pelo nível de desordem no ambiente, bem como pela interferência cruzada quando várias unidades de sensores são utilizadas no mesmo sentido.

A abordagem de Rambabu, R. et al (2015), por exemplo, utiliza um par de sensores (infra vermelho e ultrassônico) nos quatro lados do drone (conforme Figura 3.1). O

algoritmo de detecção de obstáculos monitora continuamente a presença de obstáculos nas direções citadas. Caso sejam detectados obstáculos dentro da distância de segurança configurada, uma resposta é produzida para evitar colisão.

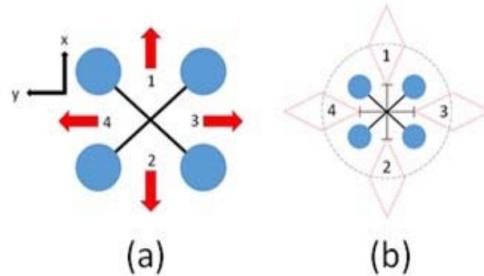


Figura 3.1 (a) As quatro direções do drone; (b) Direção dos Sensores (Rambabu, R. et al, 2015)

Já em (Pratama, Y. et al, 2017) é feita uma avaliação para identificar se a utilização do algoritmo de rede neural artificial irá influenciar no desempenho do robô. Nessa abordagem foram utilizados 3 (três) sensores ultrassônicos. O resultado obtido mostrou que, através do algoritmo de rede neural, o robô realizou desvios de obstáculos em menor tempo do que a mesma atividade sem o algoritmo de rede neural.

Como observado nas publicações anteriores, a quantidade de sensores possibilita a utilização de algoritmos mais robustos. Existem dois aspectos negativos no desenvolvimento do módulo de desvio de obstáculo para a aplicação de inspeção aérea. Primeiro, o projeto fez a aquisição de dois sensores ultrassônicos, sendo um para cada drone. Segundo, o modelo *HC-SR04* tem uma leitura imprecisa na detecção de obstáculo. Nesse sentido, diante desse cenário, foi utilizado um algoritmo simples conhecido como *First-choice Hill-climbing* (Mais detalhe na seção 4.3.5).

3.2 TRAJETÓRIA AUTÔNOMA UTILIZANDO O AR.DRONE

O número de pesquisas com base no *AR.Drone* é bastante expressivo. A seguir, algumas publicações relacionadas com este trabalho:

Rastreamento de trajetórias

Velez, P. et al (2015) propõem uma abordagem que faz uso de função polinomial cúbica e curva de *Bézier* para realizar trajetórias tridimensionais em tempo real. Para isso, a abordagem utiliza uma câmera e marcadores para determinar as posições, rastreando-as em tempo real. Inicialmente, o algoritmo busca identificar uma imagem quadrada. Encontrada a imagem, o padrão desenhado dentro do quadro é combinado com a base de dados fornecida pelo usuário (Ver Figura 3.2) disponibilizando assim, a direção da trajetória.



Figura 3.2 Exemplos de marcadores (Velez, P. et al, 2015)

Em seguida, estabeleceram uma trajetória de três segmentos em que o movimento ao longo de cada eixo foi traçado pela função polinomial cúbica e a curva de *Bézier* (Figura 3.3). O trabalho mostra que os erros de rastreamento usando a função polinomial cúbica foram de 3,73 metros (simulação) e de 5,13 metros (AR.Drone) em trajetórias de 60 metros. Nos testes utilizando a curva de *Bézier*, observaram-se erros de rastreamento de 2,8 metros (simulação) e 5,06 metros (AR.Drone) em trajetórias de 40 metros. Como resultado, Velez, P. et al (2015) descobriram que a curva de *Bézier* funcionou melhor do que a função polinomial cúbica, pois garantiu a continuidade total ao longo da trajetória.

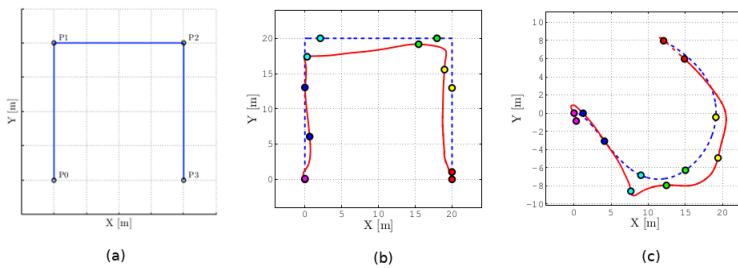


Figura 3.3 (a) Referência linear da trajetória; (b) Função polinomial cúbica; (c) Curva de *Bézier* (Velez, P. et al, 2015)

Sistema de controle de rota para voo autônomo

Nas pesquisas realizadas não foi encontrada nenhuma aplicação focada na inspeção aérea dada uma determinada área, utilizando tecnologias de código aberto e arquitetura genérica. Entretanto, o trabalho de Martins, M. (2017, p. 35-40) aborda a construção de um algoritmo de controle de rota para voo autônomo baseado em dados de GPS que serviu como base para o desenvolvimento do sistema de localização proposto neste trabalho. Segundo Martins, M. (2017), o controle de rota baseado em GPS alcançou o objetivo esperado permitindo que o *AR.Drone* voasse de forma autônoma de um ponto inicial até o destino. No entanto, devido à baixa precisão do módulo de GPS utilizado, surgiram limitações que impediram o ajuste automático de velocidade e o problema de fuga da rota.

A fim de avaliar a capacidade do algoritmo proposto em Martins, M. (2017), o projeto realizou alguns voos autônomos que constataram limitações críticas para a aplicação de

inspeção aérea. As limitações encontradas foram catalogadas na tabela 3.1 e usadas como referência para o desenvolvimento do sistema de orientação. Além disso, novas demandas surgiram durante a avaliação e serão abordadas na seção 4.

Tabela 3.1 Situação das limitações em relação ao sistema de localização

Limitações do controle de rota	Situação da limitação
Permite somente duas coordenadas	Restrição eliminada
Ajuste automático da velocidade	Restrição eliminada
Dispersão da trajetória	Restrição eliminada
Primeiro movimento desnecessário	Restrição eliminada
A trajetória é feita pausadamente	Restrição eliminada

Baseado nas pesquisas relacionadas, o trabalho propõe um novo método para fornecer um sistema de orientação com movimentos autônomos, contínuos e tridimensionais, sendo que as missões serão feitas com desempenho, segurança e habilidade em desviar de obstáculos (detalhe na seção 4), itens fundamentais para o desenvolvimento da aplicação de inspeção aérea.

SISTEMA DE ORIENTAÇÃO BASEADO EM GPS, GIROSCÓPIO E BARÔMETRO

Este capítulo apresenta um sistema de orientação para a aplicação de inspeção aérea baseado na posição, orientação e altura do drone. O drone escolhido foi o *AR.Drone 2.0* por existir uma API de código aberto disponibilizada para desenvolvedores. Entretanto, o sistema desenvolvido também pode ser utilizado com outros drones devido à arquitetura genérica e às tecnologias de código aberto (*opensources*) envolvidas. Conforme a seção 2.3, o *Raspberry PI*, além de ser um dos responsáveis pela compatibilidade com o drone, também é usado para acessar a API do *AR.Drone 2.0* e realizar os movimentos, a leitura dos sensores e o processamento das imagens, bem como para consultar a API do servidor à procura de uma missão disponível.

4.1 O PROBLEMA

Para controlar o drone de forma autônoma através do sistema de orientação, o drone precisa conhecer sua posição, velocidade e orientação em relação ao ambiente. Isso foi possível através da coleta de dados de alguns sensores. No entanto, os sensores tendem a ser imprecisos ou sensíveis a interferências. Por exemplo, mesmo usando dados fornecidos pelo GPS como posição, velocidade, altitude e norte verdadeiro, as limitações descritas na Tabela 3.1 foram encontradas. Uma maneira de minorar esse problema é usar um conjunto de dados sensoriais de forma coletiva para prever com mais precisão o estado atual. Assim, será feita uma análise da combinação dos dados como método para estimar com mais precisão o estado atual.

Vale salientar que, durante o desenvolvimento da aplicação de inspeção aérea, as seguintes características da versão anterior do sistema foram identificadas e eliminadas:

- Realização de movimentos em apenas dois planos (*front, back, left, right*);
- Realização de uma trajetória em velocidade constante;

- Não realização de pouso de emergência;
- Impossibilidade de obter o controle manual durante a missão;
- Incapacidade do drone em desviar de obstáculos;

4.2 ESCOPO DA APLICAÇÃO DE INSPEÇÃO AÉREA

O servidor executa uma aplicação chamada *Control Tower*, que é responsável pela troca de informações com o sistema de orientação, persistência de dados e montagem das imagens. Entretanto, a *Control Tower* ainda está em processo de desenvolvimento pelo LaSiD. A conexão sem fio do drone é utilizada para comunicação com o servidor e tem um alcance de aproximadamente 50m. Essa conexão não é segura, podendo ser alvo de ataques. O escopo da aplicação de inspeção aérea é apresentado na Figura 4.1.

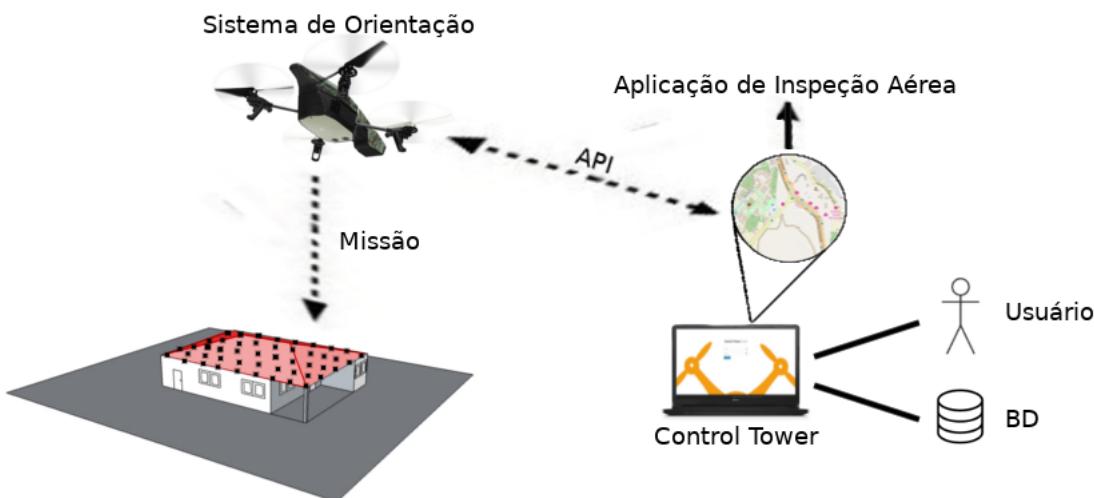


Figura 4.1 Escopo da Aplicação de Inspeção Aérea.

Segue uma descrição resumida do escopo da aplicação de inspeção aérea:

- **Control Tower:** É uma interface web projetada para gerenciar pilotos e drones, realizar o controle manual e acoplar várias aplicações. O *backend* da *Control Tower* está sendo desenvolvida usando o *framework Flask* (Flask, 2018) e o *frontend*, em *Bootstrap* (Bootstrap, 2018).
- **Aplicação de Inspeção Aérea:** É uma aplicação onde, a partir de um mapa, o usuário especifica uma determinada área. Em seguida, a aplicação gera um *grid* nessa área com coordenadas geográficas no centro de cada quadrado do *grid* (Cardoso, P., 2018). A aplicação de inspeção aérea será a primeira aplicação inserida na *Control Tower*.

- **API:** A aplicação de inspeção aérea mantém duas *API* para troca de informações. A primeira é utilizada pelo sistema de orientação para identificar se existe alguma missão ativa. Caso positivo, o sistema de orientação faz a leitura das coordenadas geográficas e inicia a missão. A segunda é usada para enviar as imagens para a aplicação de inspeção aérea.
- **Missão:** O drone, ao receber as coordenadas geográficas, deve, através do GPS, direcionar-se para a primeira coordenada gerada pela aplicação de inspeção aérea. Ao alcançar a primeira coordenada, o drone realiza o registro fotográfico e se desloca para a próxima coordenada. Ao percorrer todas as coordenadas geográficas, o drone retorna para o ponto de partida. Por fim, o drone irá fornecer as imagens para serem processadas na aplicação de inspeção aérea. A imagem gerada pelo drone é no formato PNG (*Portable Network Graphics*), com dimensões de 640 x 360 pixels.

4.3 IMPLEMENTAÇÃO DO SISTEMA DE ORIENTAÇÃO

Os dados do GPS, IMU e Barômetro foram combinados para estimar uma movimentação mais precisa e confiável para o drone. O resultado é uma orientação tridimensional, ou seja, inclui altitude, orientação e posição a partir de dados dos 3 (três) sensores. Através desta integração é possível realizar movimentos nos 3 (três) eixos de acordo com precisão dos sensores, visto que o GPS e o giroscópio informam a posição e a orientação do drone em duas dimensões. O barômetro é usado para controlar a altitude do drone. Logo, a junção dos dados sensoriais resultam em informações para os movimentos tridimensionais.

4.3.1 Modelagem

O sistema de orientação contém módulos para captura de imagens, controle de velocidade, controle de colisão e de rota. A Figura 4.2 exibe todos os módulos implementados para fornecer o suporte à aplicação de inspeção aérea. Os módulos foram implementados utilizando a linguagem *Python*. O nó cinza indica que a função é executada em um subsistema dedicado (*thread*).

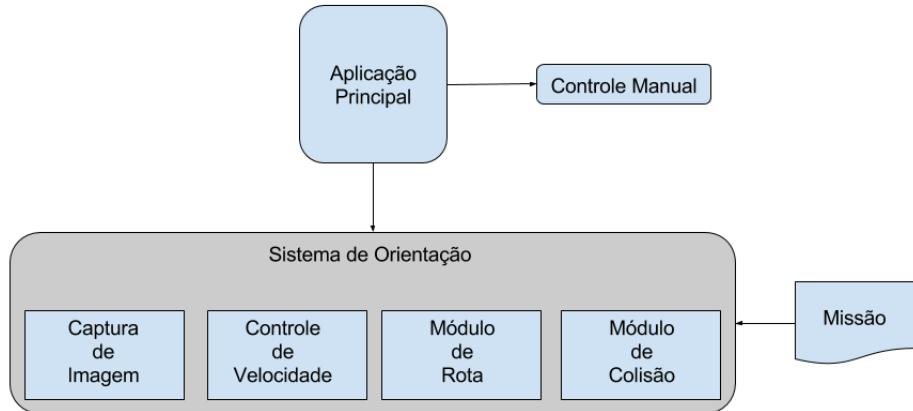


Figura 4.2 Esquema do sistema de orientação.

4.3.2 Aplicação Principal

A **aplicação principal** é um arquivo chamado de *main.py* que é executado no *Raspberry* e desenvolvido de maneira modular, para permitir a integração com diversos sistemas. Além disso, a **aplicação principal** é responsável pela comunicação com o drone através da biblioteca *ps-drone* e pela comunicação com o *Control Tower* para receber as informações da missão. O **controle manual** também foi inserido no *main.py* com o intuito de controlar o drone durante a missão. O motivo do desenvolvimento do **controle manual** está ligado à falta de segurança da comunicação criada pelo drone (Pleban, J. et al, 2014) e para evitar qualquer movimento não esperado. Somente é possível executar os comandos manuais devido à implementação do sistema de orientação como um sub-sistema (*thread*) da **aplicação principal**. A partir do *Control Tower*, um usuário do sistema poderá executar os comandos da Tabela 4.1 via teclado.

Tabela 4.1 Comandos do controle manual

Tecla	Função
<space>	decolagem e aterrissagem
<w>	mover para frente
<s>	mover para trás
<a>	mover para esquerda
<d>	mover para direita
<u>	mover para cima
<j>	mover para abaixo
<q>	girar para esquerda
<e>	girar para direita
<0>	parar qualquer movimento
<m>	iniciar a missão
<c>	cancelar a missão

4.3.3 Captura de Imagem

Este processo é responsável pelo registro fotográfico sempre que o drone alcança uma coordenada geográfica. Primeiramente, o drone é colocado em modo de configuração, com o intuito de ativar a câmera vertical, pois a câmera padrão é a horizontal. Em seguida, utiliza-se a biblioteca *OpenCV* para criar a conexão com a câmera do drone e realizar a captura da imagem em formato *PNG*. As imagens são salvas em um diretório específico do *Raspberry* e, somente no final da missão, enviadas para serem processadas na aplicação do *Control Tower*. A seguir é apresentado o funcionamento do registro fotográfico através do algoritmo 4.1:

Algoritmo 4.1 Captura de Imagem.

```
def takePicture():
    urlAccess = "tcp:192.168.1.1:5555"
    number = str(len(next(os.walk("./pictures"))[2])) + 1
    file_pic = "./pictures/picture"+number+".png"
    drone.setConfigAllID()
    drone.groundCam()
    cameraCapture = cv2.VideoCapture(urlAccess)
    success, frame = cameraCapture.read()
    cv2.imwrite(file_pic, frame)
    cameraCapture.release()
    drone.frontCam()
```

Sempre que o drone chega em algum destino, a função *takePicture* é chamada para realizar o registro fotográfico. A variável *urlAccess* contém as informações de acesso para o serviço de transmissão de vídeo do drone. A variável *number* é utilizada para diferenciar o nome dos arquivos de imagens, caso contrário, a cada destino a imagem tirada anteriormente seria substituída pela imagem atual. O valor da variável *number* é

atribuída com a quantidade de imagens da pasta *pictures*, acrescentando mais 1 (um). Nesse sentido, a variável *file_pic* corresponde ao nome do arquivo de imagem concatenada com a variável *number*. Em seguida, o drone é colocado em modo de configuração através do método *drone.setConfigAllID*. A partir do modo de configuração, o método *drone.groundCam* é chamado para selecionar a câmera localizada abaixo do drone. O método *cv2.videoCapture*, disponibilizado pela biblioteca *OpenCV*, é chamado passando como parâmetro a variável *urlAccess* para criar um objeto responsável por gerenciar o fluxo de vídeo. Logo após, o método *cameraCapture.read* realiza a captura da imagem e, o método *cv2.imwrite* é chamado para salvar a imagem no diretório *pictures*. Por fim, o método *cameraCapture.release* é chamado para desconectar a captura de vídeo. A câmera frontal é ativada novamente para que a *Control Tower* possa continuar a exibir o percurso.

4.3.4 Controle de Velocidade

Este módulo ajusta a velocidade do drone automaticamente de acordo com a distância entre a posição atual do drone e o próximo destino, com o objetivo de evitar que o drone passe pelo destino. Por exemplo, se a distância entre o drone e o destino for de 3 metros e o drone utilizasse sempre a velocidade máxima, o drone passaria pelo destino, consequentemente, diminuindo o desempenho na trajetória. Vale salientar que os movimentos do drone são baseados no valor da velocidade que varia entre 0.0 a 1.0, representando, respectivamente, 0% e 100% da velocidade máxima de 5m/s. Desta forma, foram definidos os seguintes valores:

- 100% da velocidade do drone para distâncias maiores que 100 metros.
- Para distância entre 10 e 100 metros, a velocidade é igual à distância dividida por 100.
- 10% da velocidade do drone para distâncias menores que 10 metros. Em ambientes abertos, o movimento do drone é fortemente influenciado pelo vento quando a velocidades é abaixo de 10%. Ou seja, 10% é a velocidade mínima em qualquer percurso.

O módulo de **controle de velocidade** tem como objetivo diminuir o tempo gasto no deslocamento entre as coordenadas e, consequentemente, melhorar o desempenho do percurso ao ponto de a velocidade não comprometer o reconhecimento da coordenada de destino. A seguir é apresentado um trecho do funcionamento do algoritmo:

Algoritmo 4.2 Controle da velocidade.

```
def getSpeed(distance):
    if distance > 100:
        speed = 1
    elif distance < 10:
        speed = 0.1
```

```

else:
    speed = distance/100
return speed

```

Em resumo, a função `getSpeed` recebe como parâmetro a distância entre o drone e o destino. Caso a distância possua um valor maior que 100 metros, significa que o movimento do drone será feito com velocidade máxima. Se a distância for menor que 10 metros, o drone utilizará 10% da velocidade, sendo essa a menor velocidade durante o percurso. Do contrário, a velocidade receberá o resultado da divisão da distância por 100, caracterizando um valor entre 100% e 10% da velocidade do drone.

4.3.5 Módulo de Proteção contra Colisão

O objetivo deste módulo é fazer com que o drone desvie dos obstáculos encontrados durante o voo. Basicamente, o módulo de colisão realiza o desvio através de uma trajetória perpendicular ao caminho atual (detalhe na figura 4.3), alterando o destino temporariamente durante 3 (três) segundos até que o caminho se torne livre novamente.

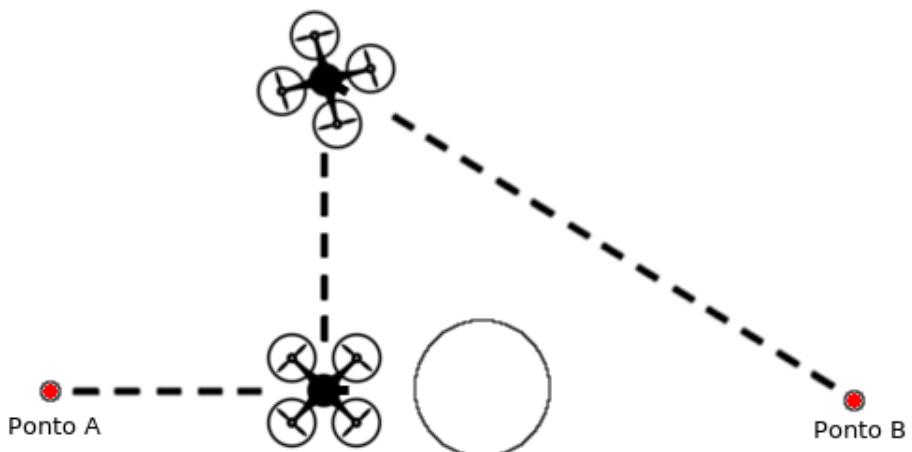


Figura 4.3 Trajetória do desvio de obstáculos.

Para que o drone evite os obstáculos por ele detectados, o módulo de colisão utiliza o algoritmo *First-choice Hill-climbing* (Russell, S. and Norvig, P., 2010). Este algoritmo é uma variação do *Hill-climbing* tradicional e difere pelo fato de escolher o primeiro melhor caminho encontrado. O funcionamento do algoritmo implementado no módulo de colisão consiste de um *loop* que continuamente verifica se há algum objeto dentro de uma distância inferior a 50 (cinquenta) centímetros. Caso encontre algum objeto, o drone para o movimento atual e verifica novamente para evitar um falso-positivo. Devido à precisão do sensor, o drone somente evita a colisão se a velocidade do drone for menor que 20%.

Se realmente for constatado algum obstáculo, o drone realiza um movimento aleatório de 90 graus para esquerda ou direita e em seguida segue em frente por 3 (três) segundos. Caso haja obstáculos, o drone continua girando 90 graus aleatoriamente em busca de uma direção sem objetos. Após 4 (quatro) tentativas consecutivas encontrando obstáculos, o que caracteriza um platô, segundo Russell, S. and Norvig, P. (2010), o drone realiza um movimento de subida durante 3 (três) segundos com o intuito de sobrevoar o obstáculo. Caso encontre obstáculos nesta nova altitude, o drone realiza um pouso de emergência.

Seguem os motivos da escolha do algoritmo *First-choice Hill-climbing*:

- usa pouca memória, visto que não armazena dados dos caminhos anteriores;
- frequentemente pode encontrar soluções razoáveis em grandes ou infinitos (contínuos) espaços de estados.
- escolhe sempre o primeiro melhor caminho para progredir rapidamente no percurso.

Antes de executar cada movimento do drone, a função *checkObstacle*, apresentada no algoritmo 4.3, é chamada para verificar se há algum objeto dentro de uma distância de 50 (cinquenta) centímetros a partir do drone. Caso encontre algum objeto, a distância do drone ao objeto é calculada novamente. Através das condições do *loop*, é verificado se a distância entre o objeto e o drone é menor ou igual a 50 (cinquenta) centímetros e, se a variável *STUCK* (inicializada com o valor zero e representa o platô) é menor ou igual às quatro tentativas de encontrar um objeto. Caso seja confirmada a presença de um obstáculo, a variável *direction* receberá, aleatoriamente, o valor 0 (zero) ou 1 (um). Caso a variável *direction* possua o valor igual a 1 (um), significa que o drone irá girar 90 (noventa) graus para esquerda, através do método *drone.turnAngle(-90,1,1)*. Mas se a variável *direction* possuir o valor igual a 0 (zero), o drone irá girar 90 (noventa) graus para a direita, através do método *drone.turnAngle(90,1,1)*. Após o drone realizar a rotação, uma nova distância é calculada. Caso a distância entre o objeto e o drone seja maior que 50 (cinquenta) centímetros, o drone irá mover-se para frente com 20% da velocidade máxima do drone, através do método *drone.moveForward(0.2)*, durante 3 segundos e depois ficará parado no ar até receber novas orientações para o destino. Do contrário, a variável *STUCK* será incrementada e verificada se possui valor maior que as quatro tentativas de encontrar um objeto. Caso positivo, o drone irá mover-se para cima com velocidade de 30% da máxima durante 3 (três) segundos, através do método *drone.moveUp(0.3)*, com o intuito de sobrevoar o obstáculo. Em seguida, a distância é calculada novamente. Caso a distância seja menor ou igual a 50 (cinquenta) centímetros, o drone realizará um pouso de emergência através do método *drone.land()*. Do contrário, o drone receberá novas orientações para o destino, fora da função *checkObstacle*.

Algoritmo 4.3 Desvio de obstáculo.

```
def checkObstacle():
    distance = sensor.get_distance()
    if distance <= 50:
        distance = sensor.get_distance()
```

```

STUCK = 0
while distance <= 50 and STUCK <= 4:
    direction = random.randint(0, 1)
    if direction == 1:
        drone.turnAngle(-90, 1, 1)
    elif direction == 0:
        drone.turnAngle(90, 1, 1)
    distance = sensor.get_distance()
    if distance > 50:
        drone.moveForward(0.2)
        time.sleep(3)
        drone.hover()
    return True
else:
    STUCK = STUCK + 1
    if STUCK > 4:
        drone.moveUp(0.3)
        time.sleep(3)
        drone.hover()
    distance = sensor.get_distance()
    if distance <= 50:
        drone.land()
return False

```

4.3.6 Módulo de Rota

Este é o principal módulo do sistema de orientação, pois minimiza a maioria das limitações do trabalho de Martins, M. (2017), tais como:

- Permite somente duas coordenadas;
- Dispersão da trajetória;
- A trajetória é feita pausadamente.

Além disso, resolve um dos objetivos específicos do projeto, **realizar movimento com precisão nos 3 eixos**.

Funcionamento

O módulo de rota é executado quando o operador pressiona a tecla $<m>$, conforme a Tabela 4.1. Em seguida, a **aplicação principal** cria uma *thread* passando como parâmetro o método *mission*, o objeto *drone* (criado pela biblioteca *ps-drone*) e uma lista encadeada contendo as coordenadas tridimensionais geradas pela *Control Tower*.

O método *mission* está implementado no arquivo chamado *orientation_system.py* e sua função é percorrer toda a lista de coordenadas geográficas. Para cada nó da lista é chamada a função *startRoute*, passando como parâmetro a coordenada geográfica (*pointB*) e a altitude desejável do ponto de destino. Além disso, existe uma variável global (*STOP-PER*) que é verificada para cancelar a missão, caso o operador pressione a tecla *<c>*. Após percorrer toda a lista de coordenadas geográficas, a função *startRoute* é chamada pela última vez, passando como parâmetro a coordenada geográfica que foi salva antes de iniciar a missão, com o intuito de retornar ao ponto de partida.

O algoritmo implementado na função *startRoute* (detalhe no algoritmo 4.4) consiste de um *loop* que somente é finalizado quando o drone chega em cada destino. O primeiro passo do *loop* é verificar se a autonomia da bateria está acima de 20%, através da função *checkBattery*. Caso esteja abaixo, o drone realiza um pouso de emergência. O segundo passo é verificar qual é a distância até o destino utilizando a função *checkDistancia* e passando como parâmetro o ponto de destino. O cálculo da distância é explicado na seção 4.4.3. A partir da distância, a velocidade do drone é configurada através da função *getSpeed*. Caso o drone esteja no raio de 2 (dois) metros de distância do ponto de destino, a função *takePicture* é acionada para fazer o registro fotográfico daquela área.

Na primeira interação do *loop*, os movimentos dependem da distância entre o drone e o destino. Se a distância for maior do que 2 (dois) metros, a função *getDirection* é utilizada para calcular o ângulo de rotação que o drone deve fazer para ficar de frente para o destino, dadas as coordenadas geográficas (leitura do GPS) e a orientação atual (leitura do giroscópio) do drone. O ângulo de retorno varia de 0 a 180 graus e o valor pode ser positivo ou negativo, indicando que o drone deve girar para direita ou esquerda, respectivamente. Em posse do ângulo de rotação, o método *turnAngle* da biblioteca *ps-drone* é chamado para executar o primeiro movimento de rotação do drone. O método *turnAngle* é executado apenas uma única vez a cada novo destino, pois esse método não foi projetado para ser usado com o drone em movimento. Logo em seguida, o método *moveAltitude* é chamado para realizar, ao mesmo tempo, os movimentos de subida (até atingir a altitude desejada) e de deslocamento para frente. A biblioteca *ps-drone* não implementa movimentos complexos, dessa forma, fez-se necessário adicionar 2 (dois) novos métodos na biblioteca: *moveAltitude* e *moveTurnAngle*.

Para as outras interações do *loop*, os movimentos são executados, basicamente, na mesma ordem conforme descrito acima. A diferença está em uma nova verificação de intervalo do ângulo. Em outras palavras, o drone continua movendo-se para frente enquanto o ângulo de rotação estiver dentro do intervalo de -10 a 10 graus, através da variável *LIMIT_ANGLE*, sendo esse valor configurável. O motivo dessa configuração é evitar movimentos desnecessários, consequentemente, menor consumo de bateria. Mas caso o ângulo seja maior do que o intervalo proposto, o método *moveTurnAngle* é chamado para realizar a rotação ao mesmo tempo em que o drone movimenta-se para frente. Dessa forma, o percurso somente terá pausa para o registro fotográfico.

O pouso emergencial pode ser feito manualmente através do módulo de **controle manual** e, de maneira automática, se a bateria estiver comprometida. Mas, além dessas duas formas, foram implementadas algumas condições (*exceptions*) no código do sistema de orientação com o intuito de realizar o pouso, caso ocorram os seguintes problemas:

perda de conexão entre o *Raspberry* e o GPS, perda de conexão entre o GPS e todos os satélites, falha no sistema de orientação e interrupção forçada por comando do operador.

Algoritmo 4.4 deslocamento do drone.

```

def startRoute(pointB, altitude):
    try:
        arrived = False
        initial = True
        while not arrived:
            checkBattery()
            distance = checkDistance(pointB)
            speed = getSpeed(distance)
            if distance <= 2:
                drone.stop()
                time.sleep(2)
                arrived = True
                takePicture()
                print "The coordinate has been reached..."
            else:
                # ----- Calculing direction -----
                direction = getDirection(pointB)
                if initial:
                    print "First moving to: ", direction
                    drone.turnAngle(direction,1)
                    initial = False
                    time.sleep(2)
                    drone.moveAltitude(altitude, speed)
                elif abs(direction) > LIMIT_ANGLE:
                    print "Direction out to: ", LIMIT_ANGLE
                    drone.moveTurnAngle(direction, speed)
                    drone.moveAltitude(altitude, speed)
                # Stop mission any time
                if STOPPER:
                    drone.stop()
                    arrived = True
                    time.sleep(3)
    ...

```

4.4 FÓRMULAS DE NAVEGAÇÃO

4.4.1 Cálculo do Norte Verdadeiro

Honeywell (2018) explica os conceitos básicos do campo magnético da Terra e como as leituras dos eixos Hx e Hy do magnetômetro podem ser usadas para definir o norte

magnético. Além disso, cada região do globo tem um ângulo de declinação do campo magnético que deve ser adicionado ou subtraído para determinar o norte verdadeiro. O norte magnético é a direção indicada pelo campo magnético da Terra. O norte verdadeiro, bastante usado em navegação, é a direção da superfície da terra até o norte geográfico. Desta forma, para calcular o norte verdadeiro é necessário usar o magnetômetro disponível no drone. A direção é definida em Honeywell (2018) conforme abaixo:

$$\text{let } \theta = \tan(x/y)$$

$$\text{direction} = \begin{cases} 90 - \frac{180}{\pi}\theta, & \text{if } y > 0 \\ 270 - \frac{180}{\pi}\theta, & \text{if } y < 0 \\ 180, & \text{if } y = 0 \wedge x < 0 \\ 0, & \text{if } y = 0 \wedge x \geq 0 \end{cases}$$

O resultado da função para calcular o norte verdadeiro em *Python* é a seguinte:

Algoritmo 4.5 Norte verdadeiro.

```
def trueNorth(x,y):
    return (180/math.pi * math.atan2(y, x))
```

Porém, mesmo quando o drone estava parado, foram identificados muitos ruídos na leitura dos dados do magnetômetro, causando movimentos inesperados. Outro problema encontrado foi que o sensor travava durante a execução da missão, informando o mesmo valor independentemente da direção do drone. Para contornar esse problema de forma a não comprometer o cronograma da pesquisa, foi decidido utilizar o aplicativo *Compass*¹ para obter o norte verdadeiro. Como resultado, a bateria deve ser conectada ao drone quando o mesmo estiver apontando para a direção do norte verdadeiro, calculado pelo aplicativo *Compass*. No momento em que o drone é ligado, o giroscópio define o ângulo 0 (zero) para a frente do drone. Dessa forma, o ângulo 0 (zero) do giroscópio torna-se a referência do norte verdadeiro para o cálculo de rotação da seção 4.4.2.

A figura 4.4 ilustra a abordagem descrita acima para contornar o problema com o magnetômetro do *AR.Drone 2.0*. Provavelmente, a abordagem ideal seria utilizar um novo sensor de magnetômetro no *Raspberry* e algum algoritmo de filtragem, conforme relatado em Kok, M. et al (2017). Contudo, esta abordagem ficará como trabalho futuro.

¹<https://play.google.com/store/apps/details?id=com.gn.android.compass>

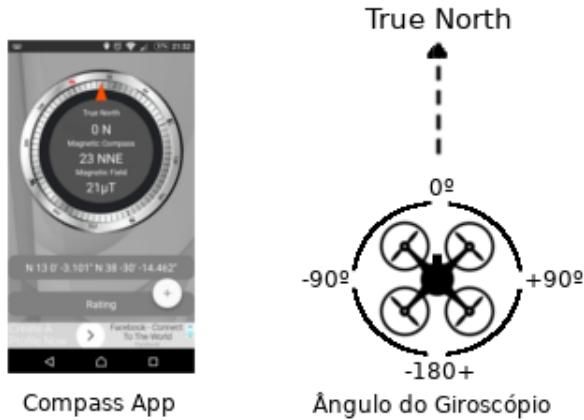


Figura 4.4 Aplicativo *Compass* para referência do giroscópio.

4.4.2 Cálculo do Ângulo de Rotação

Na navegação aérea, existem alguns conceitos básicos para entender e calcular o ângulo de rotação. Segundo Veness, C. (2018), o *course* é o caminho pretendido para chegar ao destino. Se não houver influência externa, basta seguir em linha reta até o destino. O *track* é o caminho real feito pelo drone. O *heading* é a direção para a qual o drone está apontando. O *bearing* é o ângulo medido no sentido horário do norte para o destino, tendo como vértice a posição do drone. Por fim, *relative bearing* é o ângulo medido no sentido horário do *heading* para o destino, tendo como vértice a posição do drone. Os ângulos são definidos em graus tendo o norte verdadeiro como referência. Apenas o ângulo do *relative bearing* tem como referência o *heading*. Por exemplo, da figura 4.5, o drone decolou do ponto azul e uma reta verde (*course*) foi traçada até o destino (ponto verde). Porém, devido à influência do vento, o drone realizou outro caminho (*track*), representado pela reta amarela. A cada três segundos, durante o percurso do *track*, o *relative bearing* é calculado e, caso seja maior do que 10 graus, o drone irá girar em direção ao destino utilizando o ângulo do *relative bearing*. Como o drone foi ligado apontando para o norte verdadeiro, o valor de 95 graus da reta azul (*heading*) pode ser lido pelo giroscópio. Neste exemplo, o ângulo formado entre as retas vermelhas (*bearing*) foi de 135 graus. Logo, o *relative bearing* (α) é calculado subtraindo o valor do *heading* pelo valor do *bearing* ($\alpha = -95 + 135$). Como resultado, o drone vai girar 40 graus para a direita, pois o valor é positivo e maior do que o limite de 10 graus. O algoritmo foi implementado na função *getDirection* mostrada no algoritmo 4.4.

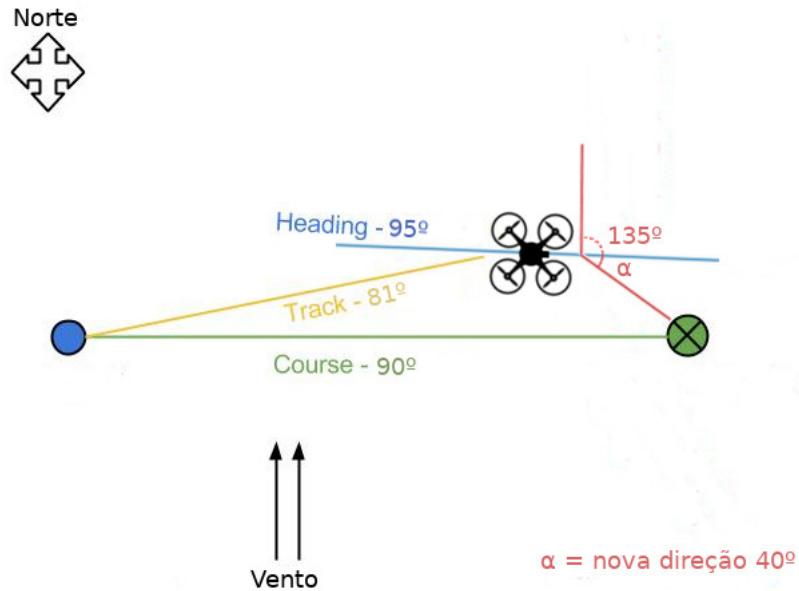


Figura 4.5 Conceitos básicos de navegação.

A seguir é apresentado o algoritmo 4.6, responsável pelo cálculo do *relative bearing*.

Algoritmo 4.6 Calculo do *relative bearing*.

```

while True:
    if (len(gpsc.satellites) > 0):
        heading = drone.NavData["demo"][2][2]
        coordinatesGPS = (gpsc.fix.latitude, gpsc.fix.longitude)
        lat1 = math.radians(coordinatesGPS[0])
        lat2 = math.radians(pointB[0])
        diffLong = math.radians(pointB[1] - coordinatesGPS[1])
        x = math.sin(diffLong) * math.cos(lat2)
        y = math.cos(lat1) * math.sin(lat2) - (math.sin(lat1)
            * math.cos(lat2) * math.cos(diffLong))
        bearing = math.atan2(x, y)
        bearing = math.degrees(bearing)
        if not math.isnan(bearing):
            relativebearing = round(-heading + bearing)
            if relativebearing > 180:
                relativebearing = relativebearing - 360
            if relativebearing < -180:
                relativebearing = relativebearing + 360
        return relativebearing

```

Em resumo, o algoritmo começa verificando, em um *loop*, se o GPS já conseguiu conexão com algum satélite. Assim que a conexão é estabelecida, o valor em graus do

eixo z do giroscópio é colocado na variável *heading*. A posição atual do drone, latitude e longitude, é inserida na tupla *coordinatesGPS*. Em seguida, ambas as latitudes de destino e da posição atual são inseridas, respectivamente, em *lat2* e *lat1*. A diferença entre as longitudes de destino e da posição atual é inserida na variável *diffLong* (Δlong). Para calcular o *bearing* (Θ) utiliza-se a seguinte fórmula trigonométrica:

$$\begin{aligned}\Theta &= \text{atan2}(x, y) \\ x &= \sin(\Delta\text{long}) \cdot \cos(\text{lat}2) \\ y &= \cos(\text{lat}1) \cdot \sin(\text{lat}2) - \sin(\text{lat}1) \cdot \cos(\text{lat}2) \cdot \cos(\Delta\text{long})\end{aligned}$$

Após calcular o *bearing* (Θ) que corresponde aos valores entre -180° e $+180^\circ$, o *relativebearing* é calculado multiplicando o *heading* por menos 1 (um) e somando pelo *bearing*, com o intuito de obter o menor ângulo de rotação. Porém, caso o *relativebearing* possua valor maior que 180 graus, o *relativebearing* será subtraído por 360, indicando que o menor ângulo deve ser negativo e o movimento para esquerda. Mas, caso o *relativebearing* possua valor menor que -180 graus, o *relativebearing* será somado por 360, indicando que o menor ângulo deve ser positivo e o movimento para direita.

4.4.3 Cálculo da Distância

Outro cálculo importante na navegação é saber qual é a distância do drone até o destino. Isso é possível através da fórmula de *haversine*. Essa equação calcula a distância entre duas coordenadas geográficas a partir da relação dos lados e dos ângulos de triângulo esférico que é a união de três segmentos de uma esfera. Baseado nessa distância, o sistema de orientação descobre se o drone já alcançou o destino para, assim, realizar o registro fotográfico. A fórmula de *haversine* é mostrada a seguir (Veness, C., 2018):

$$\begin{aligned}a &= \sin^2(\Delta\varphi/2) + \cos \varphi_1 \cdot \cos \varphi_2 \cdot \sin^2(\Delta\lambda/2) \\ c &= 2 \cdot \text{atan2}(\sqrt{a}, \sqrt{1-a}) \\ d &= R \cdot c\end{aligned}$$

Onde φ é a latitude das coordenadas geográficas, λ é a longitude, R é o raio da terra (raio médio = 6.371 km) e d é a distância entre as coordenadas em metros.

Contudo não foi necessário implementar a fórmula de *haversine*, pois existe uma biblioteca em *Python* que implementa a equação. Além disso, a biblioteca considera o raio da terra na posição correta das coordenadas, evitando erros no cálculo. Segue trecho da função *checkDistance* no sistema de orientação. No algoritmo 4.7, a função *checkDistance* recebe como parâmetro uma tupla contendo a latitude e a longitude do destino. Um *loop* é utilizado até que o GPS estabeleça conexão com algum satélite. Assim que a conexão é estabelecida, a posição atual do drone, latitude e longitude, é inserida na tupla *coordinatesGPS*. A função de *haversine* recebe como parâmetro as coordenadas geográficas de destino e da posição atual do drone, retornando a distância em quilômetros. Porém, durante a missão utiliza-se a distância em metros. Logo, a distância receberá o

retorno da função *haversine* multiplicado por 1000.

Algoritmo 4.7 Cálculo da distância.

```
def checkDistance(pointB):
    if (type(pointB) != tuple):
        raise TypeError("The argument must be a tuple")
    while True:
        if (len(gpsc.satellites) > 0):
            coordinatesGPS = (gpsc.fix.latitude, gpsc.fix.longitude)
            distance = round(haversine(coordinatesGPS, pointB) * 1000)
        return distance
    ...
```

Capítulo

5

RESULTADOS

O desempenho do sistema de orientação foi avaliado em dois experimentos, ambos em ambientes iguais e reais. No primeiro experimento, os movimentos do drone são manualmente manipulados com o objetivo de desprezar as influências externas (e.g., vento). E no segundo experimento, os movimentos são realizados pelo drone de forma autônoma. Além disso, os movimentos ao longo do percurso foram comparados com a pesquisa de Velez, P. et al (2015). Por fim, a aplicação de inspeção aérea foi colocada em prática.

5.1 EXPERIMENTOS

Nos experimentos conduzidos foi utilizada uma trajetória de 60 metros com três percursos em direções perpendiculares, conforme a Figura 5.1, por se tratar de um percurso tridimensional e ao mesmo tempo utilizada no trabalho acadêmico (Velez, P. et al, 2015). A trajetória é representada pelas distâncias nos eixos x e y em função da altura no eixo z . Cada percurso contém 20 metros de comprimento e a altura varia de 1 a 2 metros.

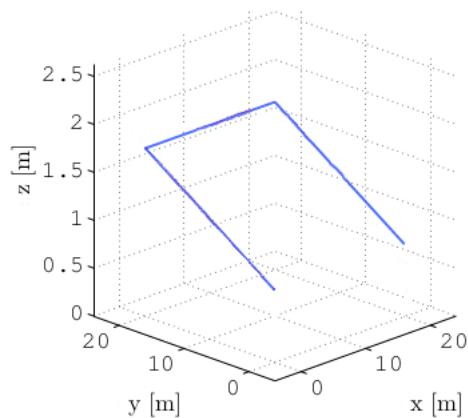


Figura 5.1 Topologia dos experimentos.

Voo Simulado

O primeiro experimento utilizou um modelo orientado a rastreamento. Segundo Jain, R. (1990), os experimentos baseados em rastreamento são bastante comuns nas análises de sistemas devido à credibilidade, pois utilizam-se dados de sistema real. O modelo orientado por rastreamento deve ser semelhante ao sistema que está sendo modelado. Dessa forma, para realizar os movimentos manipulados manualmente, foi criado o arquivo *simulator.py* que implementa todos os algoritmos do sistema de orientação de maneira a interagir a cada movimento. O primeiro passo na simulação foi monitorar o rastreamento. Após coletar os dados de 20 amostras, os mesmos foram inseridos na ferramenta SunEarthTools (2018) para comparar os cálculos de rotação e distância. Observou-se que tanto a distância como a rotação são calculadas corretamente. Na trajetória de 60 metros, o maior percurso foi feito em 68.72 metros e o menor percurso em 52.22 metros, conforme a Figura 5.2.

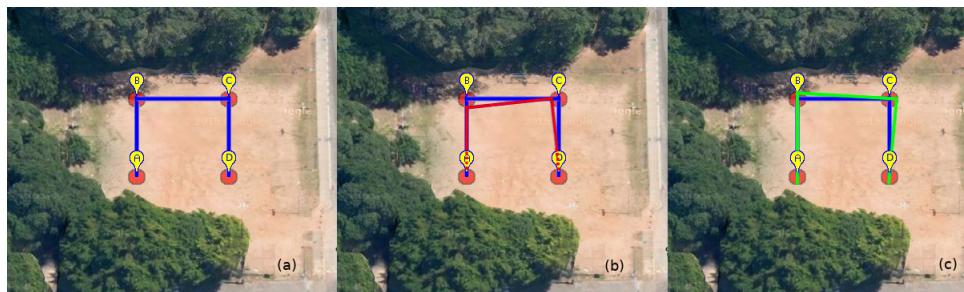


Figura 5.2 (a) Trajetória linear; (b) Menor percurso percorrido; (c) Maior percurso percorrido.

Os resultados dependem do ponto de partida do drone, visto que o destino é alcançado quando a posição do drone encontra-se dentro de uma circunferência de raio igual a dois metros, sendo o centro a coordenada geográfica de destino. Como o objetivo não é chegar exatamente na coordenada de destino, conclui-se que os algoritmos utilizados no sistema de orientação pode obter resultados tão bons quanto a pesquisa de Velez, P. et al (2015).

Voo Autônomo

No segundo experimento, foram realizados 20 voos autônomos. Durante as missões, o movimento na direção das coordenadas geográficas de destino teve 100% de acerto. O deslocamento sempre foi contínuo até o destino e a velocidade variando de acordo com a distância restante. Além disso, em todos os voos autônomos, o drone realizou o pouso de emergência quando a bateria estava comprometida. Porém, devido à imprecisão do módulo de GPS, o drone movimentou-se além do espaço físico permitido, inviabilizando o percurso da trajetória de 60 metros. Entretanto, após aumentar o raio da circunferência para 10 metros, foi possível realizar o deslocamento entre três coordenadas. A figura 5.3

mostra o caminho percorrido entre as coordenadas geográficas, mapeado pela ferramenta desenvolvida em Schneider, A. (2018).



Figura 5.3 Ferramenta GPS Visualizer

Capítulo

6

CONCLUSÃO

A proposta deste trabalho parte da necessidade de criação de um sistema de orientação para ser utilizada na aplicação de inspeção aérea, bem como em outras aplicações. Sabe-se que os drones vêm sendo utilizados em diversas atividades devido ao avanço da tecnologia, o que aumenta ainda mais a relevância deste trabalho. Dentre as limitações existentes no voo autônomo, destacam-se a segurança, evasão de obstáculos e orientação tridimensional.

A plataforma selecionada neste trabalho é o quadricóptero *AR.Drone 2.0*. Porém, um modelo de plataforma genérica é proposto para permitir a utilização do sistema de orientação a partir de outros drones. Este modelo consiste de um *Raspberry*, sensor ultrassônico e GPS. A plataforma genérica é capaz de executar algoritmos robustos e eficientes ao ponto de realizar voos autônomos mais elaborados, além do processamento de imagens.

Os experimentos mostraram que o sistema de orientação cumpriu os objetivos que a seguir se especificam novamente, contribuindo para a concretização da presente monografia:

- Realizar movimento com precisão nos 3 eixos;
- Melhorar o desempenho do percurso;
- Realizar pouso de emergência;
- Obter controle manual durante a missão;
- Desviar de obstáculo.

Após a conclusão deste trabalho é possível observar duas principais contribuições técnicas. A primeira consiste na plataforma genérica que permite a execução de algoritmos e voos autônomos sobre diferentes tipos de drones. A outra principal contribuição tem como base o desenvolvimento da primeira aplicação para drones e que apresenta grande utilidade na inspeção de qualquer caminho.

6.1 DIFICULDADES ENCONTRADAS

Os experimentos se mostraram um grande desafio devido à qualidade dos sensores e da autonomia do drone. A primeira limitação identificada foi a duração da bateria do drone. No máximo, a trajetória do experimento era feita três vezes com a mesma bateria. Em relação aos sensores, a maior limitação foi no módulo de GPS, pois a leitura das coordenadas não representou a real localização do drone. Por sua vez, a limitação de 2 metros de distância do sensor ultrassônico impossibilitou a identificação de obstáculos com velocidade acima de 20% da máxima.

6.2 TRABALHOS FUTUROS

Em termos de trabalho futuro, o documento final prevê algumas linhas de pesquisa que possam vir a ser alvo de interesse, sobretudo uma análise mais ampla do desempenho da bateria do drone, especialmente para diferentes valores de velocidade.

Em relação ao algoritmo para obter o norte verdadeiro, seria interessante utilizar um novo magnetômetro que permitisse prever a direção do campo magnético da Terra sem que haja travamentos. Ainda sobre o cálculo do norte verdadeiro, para evitar ruídos e outras incertezas, existe a possibilidade de utilizar algoritmos de filtragem para gerar resultados que busquem aproximar dos valores reais.

No que tange à evasão de obstáculos, seria importante adquirir um número maior de sensores de diferentes tipos, com melhor precisão, para cobrir todos os lados do drone. Assim, desenvolver um algoritmo mais eficiente na detecção de obstáculo.

Por fim, outra linha de pesquisa essencial ao sistema de orientação seria aperfeiçoar a localização do drone. Neste sentido, seria indispensável adquirir um módulo GPS com qualidade superior ao GPS atual. Outro tipo de abordagem seria utilizar a combinação do GPS com a Internet e redes de celulares para determinar a localização com mais precisão, uma vez que, com o uso somente do GPS, houve grande variabilidade nos resultados obtidos.

REFERÊNCIAS BIBLIOGRÁFICAS

- Bootstrap. *Bootstrap*. 2018. Último acesso em 13 de Fevereiro de 2018. Disponível em: <<https://getbootstrap.com>>.
- Bosh. *BMA150 Digital, triaxial acceleration sensor*. 2018. Último acesso em 11 de Fevereiro de 2018. Disponível em: <https://wiki.odroid.com/_media/en/universal/_motion/_joypad/bma150.pdf>.
- Cardoso, P. *Sistema de Definição e Acompanhamento de Missões para Aerofotogrametria utilizando Drones Autônomos*. 2018. 10-11 p.
- Cheeman, P. et al. *Development of a Control and Vision Interface for an AR.Drone*. MATEC Web of Conferences, 2016. 2 p. Disponível em: <https://www.matec-conferences.org/articles/matecconf/pdf/2016/19/matecconf_iccae2016_07002.pdf>.
- Domelen, D. *Getting Around The Coriolis Force*. Physics Education Research Group - Department of Physics - Ohio State University, 2018. Último acesso em 11 de Fevereiro de 2018. Disponível em: <<https://www.eyrie.org/~dvandom/Edu/newcor.html>>.
- ELEC. *Ultrasonic Ranging Module HC-SR04*. 2018. Último acesso em 03 de Janeiro de 2018. Disponível em: <<http://www.micropik.com/PDF/HCSR04.pdf>>.
- Flask. *Web development one drop at a time*. 2018. Último acesso em 13 de Fevereiro de 2018. Disponível em: <<http://flask.pocoo.org>>.
- GRAAFF, J. *A tutorial and a detailed documentation of PS-Drone*. 2012. Último acesso em 27 de Dezembro de 2017. Disponível em: <<http://www.playsheep.de/drone/downloads.html>>.
- Honeywell. *Compass Heading Using Magnetometers*. 2018. Último acesso em 26 de Janeiro de 2018. Disponível em: <<https://cdn-shop.adafruit.com/datasheets/AN203_Compass_Heading_Using_Magnetometers.pdf>>.
- InvenSense. *Integrated Dual-Axis Gyro*. 2018. Último acesso em 11 de Fevereiro de 2018. Disponível em: <<https://www.sparkfun.com/datasheets/Components/SMD/Datasheet_IDG500.pdf>>.
- Jain, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. [S.I.]: John Wiley Sons, 1990.

- Khan, M. Quadcopter Flight Dynamics. International Journal of Scientific & Technology Research, p. 2, 2014. Disponível em: <<http://www.ijstr.org/final-print/aug2014/Quadcopter-Flight-Dynamics.pdf>>.
- Kok, M. et al. *Using Inertial Sensors for Position and Orientation Estimation*. 2017. Disponível em: <<https://arxiv.org/pdf/1704.06053.pdf>>.
- Martins, M. *Sistema de Controle de Rota para Vôo Autônomo de Drones Baseado em Dados de GPS*. 2017. 35-40 p.
- OpenCV team. *Open Source Computer Vision Library*. 2018. Último acesso em 08 de Fevereiro de 2018. Disponível em: <<https://opencv.org/>>.
- PARROT. *Parrot for Developers*. 2018. Último acesso em 15 de Janeiro de 2018. Disponível em: <<http://developer.parrot.com/docs/SDK3>>.
- Pleban, J. et al. *Hacking and securing the AR.Drone 2.0 quadcopter: investigations for improving the security of a toy*. [S.l.]: Proceedings of the SPIE, Volume 9030, id. 90300L 12 pp., 2014.
- Pratama, Y. et al. *Analysis and Design of Obstacle Avoidance on Robot Detection of Pipe Cracked*. Jurnal Ilmiah Pendidikan Teknik Elektro, 2017. Disponível em: <<http://jurnal.untirta.ac.id/index.php/VOLT/article/viewFile/2044/2058>>.
- Python. *The Python Tutorial*. 2018. Último acesso em 03 de Janeiro de 2018. Disponível em: <<https://docs.python.org/3/tutorial/index.html>>.
- Rambabu, R. et al. *Relative Position-Based Collision Avoidance System for Swarming UAVS Using Multi-Sensor Fusion*. [S.l.]: ARPN Journal of Engineering and Applied Sciences, VOL. 10, NO. 21,, 2015.
- Raspberry. *The Official Raspberry PI Projects Book*. Raspberry Pi (Trading) Ltd., Mount Pleasant House, Cambridge, CB3 0, 2018. Último acesso em 30 de Dezembro de 2017. Disponível em: <https://www.raspberrypi.org/magpi-issues/Projects__Book__v1.pdf>.
- Raspbian. *Welcome to Raspbian*. 2018. Último acesso em 03 de Janeiro de 2018. Disponível em: <<https://www.raspbian.org>>.
- Russell, S. and Norvig, P. *Artificial Intelligence A Modern Approach*. [S.l.]: Prentice-Hall, 3nd Ed, 2010. 158-198 p.
- Schneider, A. *GPS Visualizer*. 2018. Último acesso em 27 de Fevereiro de 2018. Disponível em: <<http://www.gpsvisualizer.com>>.
- Sensing Device. *Sensor Ultra Miniature Size Vibration Gyro Sensor*. 2018. Último acesso em 11 de Fevereiro de 2018. Disponível em: <<http://www.mouser.com/ds/2/137/1335447-283493.pdf>>.

- Son, Y. et al. *Rocking Drones with Intentional Sound Noise on Gyroscopic Sensors*. 24th USENIX Security Symposium, 2015. 881 p. Disponível em: <<https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-son.pdf>>.
- SunEarthTools. *Sun Earth Tools*. 2018. Último acesso em 27 de Fevereiro de 2018. Disponível em: <<https://www.sunearthtools.com/pt/tools/distance.php>>.
- Sydney, N. et al. Dynamic Control of Autonomous Quadrotor Flight in an Estimated Wind Field. IEEE, p. 4, 2013. Disponível em: <<https://pdfs.semanticscholar.org/fcdc/2324f80d3db06a41c519a29a6172176fd816.pdf>>.
- UBLOX. *NEO-6 u-blox 6 GPS Modules*. 2018. Último acesso em 03 de Janeiro de 2018. Disponível em: <[https://www.u-blox.com/sites/default/files/products/documents/NEO-6__DataSheet__\(GPS.G6-HW-09005\).pdf](https://www.u-blox.com/sites/default/files/products/documents/NEO-6__DataSheet__(GPS.G6-HW-09005).pdf)>.
- Velez, P. et al. *Trajectory generation and tracking using the AR.Drone 2.0 quadcopter UAV*. [S.1.]: IEEE, 2015.
- Veness, C. *Calculate distance, bearing and more between Latitude/Longitude points - Movable Type Scripts*. 2018. Último acesso em 26 de Janeiro de 2018. Disponível em: <<http://www.movable-type.co.uk/scripts/latlong.html>>.