JOHNNATAN MESSIAS PEIXOTO AFONSO

Orientador: Ricardo Augusto Rabelo Oliveira

FRAMEWORK PARA SISTEMAS DE NAVEGAÇÃO DE VEÍCULOS AÉREOS NÃO TRIPULADOS

Ouro Preto

Dezembro de 2014

Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Biológicas
Bacharelado em Ciência da Computação

FRAMEWORK PARA SISTEMAS DE NAVEGAÇÃO DE VEÍCULOS AÉREOS NÃO TRIPULADOS

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Ouro Preto como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

JOHNNATAN MESSIAS PEIXOTO AFONSO

Ouro Preto
Dezembro de 2014

UNIVERSIDADE FEDERAL DE OURO PRETO

FOLHA DE APROVAÇÃO

Framework Para Sistemas de Navegação de Veículos Aéreos Não Tripulados

JOHNNATAN MESSIAS PEIXOTO AFONSO

Monografia defendida e aprovada pela banca examinadora constituída por:

Dr. Ricardo Augusto Rabelo Oliveira – Orientador
Universidade Federal de Ouro Preto

Msc. Marcelo Luiz Silva
Universidade Federal de Ouro Preto

Dr. Álvaro Guarda
Universidade Federal de Ouro Preto

Ouro Preto, Dezembro de 2014

Resumo

Tornar voos não tripulados autônomos sem dúvida capacitará novas oportunidades de desenvolvimento científico. Os drones podem ser utilizados em serviços militares como, por exemplo, em combates ou ainda bem como para missões de resgate, pesquisa aérea, supervisão e inspeção de um território, atraindo bastante atenção dos veículos de comunicação como, por exemplo, emissoras de televisão, rádio, jornais e internet. O objetivo desse projeto é saber se é possível tornar voos autônomos viáveis no drone AR.Drone 2.0 bem como a compreensão sobre o seu funcionamento. Para isso será necessário a implementação de um programa de controle para voos autônomos. Esta realização requer a aquisição de dados durante o voo, os quais são obtidos através de sensores que utilizam *Arduino*. A comunicação do *Arduino* com o drone é necessária para a inclusão de novos sensores e a utilização destes pelo Ar.Drone é realizada mediante o framework *Node.js*. Cada botão do controle remoto possui um comando específico, podendo ser com o objetivo de o próprio usuário criar missões ou até mesmo executar algumas missões anteriormente implementadas pelo desenvolvedor. Todos os testes foram executados no AR.Drone 2.0, utilizando o framework *Node.js*, sensores e um controle remoto. Mediante os experimentos e estudos apresentados tornou-se possível atingir o objetivo proposto, tornando viável a aplicação de voos autônomos no drone. Como resultado, para a realização de voos autônomos foi elaborado um framework onde o usuário poderá criar missões de voos autônomos para o drone executa-las. Esses comandos são enviados ao drone pelo usuário devido a utilização de um controle remoto. Esse controle remoto envia dados a um sensor conectado ao *Arduino* que processa os dados e em seguida é lido e interpretado pelo drone.

Palavras-chave: AR.Drone. Voo Autônomo. VANT's. Sensores. Quadricópteros. Arduino. Node.js.

Abstract

Become autonomous unmanned flights undoubtedly enable new opportunities for scientific development. The drones can be used in military services, for example, in combat or as well as for rescue missions, aerial survey, supervision and inspection of a territory, attracting significant attention from media outlets such as, for example, television stations, radio, newspapers and internet. The goal of this project is whether it is possible to make viable autonomous flights at AR.Drone 2.0 and the understanding of its operation. This will require the implementation of a control program for autonomous flights. This framework requires the acquisition of data during the flight, which are obtained using sensors which use *Arduino*. The *Arduino* communication with the drone is needed for the inclusion of new sensors and the use of the AR.Drone is performed by the framework *Node.js*. Each remote button has a specific command, and may be in order for the user to create own missions or even perform some missions previously implemented by the developer. All tests were run on the AR.Drone 2.0, using the *Node.js* framework, sensors and a remote control. Through the experiments and presented studies became possible to achieve the proposed objective, making possible the implementation of autonomous flights in drone. As a result, for the realization of autonomous flight we designed a framework where the user can create autonomous flight missions for the drone run them. These commands are sent to the drone by the user due to use of a remote control. This remote control sends data to a sensor connected to the *Arduino* that processes the data and then is read and interpreted by the drone.

Key words: AR.Drone. Autonomous Flights. UAV's. Sensors. Quadrotors. Arduino. Node.js.

Dedico este trabalho à minha família, namorada e amigos do intercâmbio pelo incentivo e apoio desde o início. Sempre encorajando-me a seguir sempre em frente de modo a alcançar todos os meus objetivos. Aos professores e orientadores por terem compartilhado todo o conhecimento possível para a minha formação em Ciência da Computação.

Agradecimentos

Em especial agradeço aos meus pais que sempre me apoiaram e incentivaram. Sem vocês esse sonho não seria possível. Meus irmãos o meu agradecimento pelo companheirismo e amizade. E a minha família agradeço a torcida e todo o carinho. Agradeço a minha namorada por todo o apoio em todos os momentos e por toda a força e motivação concebida a mim. Agradeço aos meus amigos do intercâmbio pelos quais compartilharam muitos momentos de felicidade comigo, tornando meu intercâmbio extremamente agradável e proveitoso. Aos orientadores por todo o apoio concebido, por compartilhar todo o conhecimento necessário para a minha formação e para a conclusão desse projeto. Agradeço em especial aos professores Ricardo Rabelo, Fabrício Benevenuto e David Menotti por todos os ensinamentos e apoio durante todo o meu período de graduação.

#ComputerScience, #UFOP, #CsFriends, #CsFamily, #Hungary, #VamoQueVamo

Sumário

Lista de Figuras

Lista de Tabelas

Universidade Federal de Ouro Preto – UFOP
Instituto de Ciências Exatas e Biológicas – ICEB
Departamento de Computação – DECOM
Curso de Bacharelado em Ciência da Computação
Disciplina de Monografia II (BCC391)

Formulário de avaliação – Monografia II

No dia 09 de dezembro de 2014, defendeu a sua monografia, parte 2, intitulada:

Framework para Sistemas de Navegação de Veículos Aéreos não Tripulados.

O aluno do curso de Ciência da Computação:

JOHNNATAN MESSIAS PEIXOTO AFONSO

Orientado pelo professor **Ricardo Augusto Rabelo Oliveira**, em sessão aberta a toda a comunidade, perante a banca de avaliação composta pelo orientador e pelos professores **Alvaro Guarda e Marcelo Luiz Silva**. A banca, após realizar suas considerações, decidiu pela seguinte nota, em uma escala de 0 a 100%:

100%

A banca de avaliação tem os seguintes comentários* (utilizar o verso e rubricar, se for necessário):

Ricardo Rabelo
(Orientador)

Álvaro Guarda
Membro 1

Marcelo Luiz Silva
Membro2

* **Caso a banca sugira alterações, a oficialização da nota ficará dependente da entrega** final, que deverá ser revisada pelo orientador.

Capítulo 1

Introdução

As aeronaves são cada vez mais utilizadas na sociedade, sejam mediante transporte de carga, de pessoas e até mesmo usos militares e resgate. Estes fatos evidenciam a utilidade das aeronaves para os seres humanos. De fato, é realmente, cada vez mais comum a interação humano-aeronave, entretanto os modelos convencionais deixam de lado voos que necessitem de manobras mais complexas como, por exemplo, voos realizados em locais com restrição de espaço, implicando em maior perícia para o voo. Suponha que ocorreu uma catástrofe natural, exigindo uma missão de resgate. A tarefa de localização dos sobreviventes é complexa e requer um mapeamento geográfico da região. Essa tarefa facilita a localização das vítimas e é realizada com sucesso por drones.

Devido as recentes melhorias na tecnologia hoje é possível criarmos aeronaves pequenas e não tripuladas conhecidas por VANT[1] ou do inglês *UAV*[2] mas popularmente conhecidos como drones. Podem ser controlados por um controle remoto, semi-autónomo ou autônomo, ou uma combinação dos dois. O drone é capaz de resolver tarefas específicas, como a vigilância do tráfego, vigilância meteorológica, transporte de carga específica ou resolver tarefas onde a presença de um piloto humano pode ser inviável ou perigosa. Em comparação com um veículo aéreo tripulado, este tem algumas vantagens como: custo reduzido, maior resistência, menor risco para as tripulações. Possuem as vantagens de poderem voar *indoor*, em baixas altitudes ou em territórios com obstáculos Morar e Nascu (2012). Os drones podem ser utilizados em serviços militares como, por exemplo, em combates ou ainda bem como para missões de resgate, pesquisa aérea, supervisão e inspeção de um território, atraindo bastante atenção dos veículos de comunicação como, por exemplo, emissoras de televisão, rádio, jornais e internet.

Neste projeto, será utilizado o AR.Drone 2.0, vide Figura 3.1, um helicóptero com quatro rotores construído pela empresa francesa Parrot apresentado ao mundo na *CES 2010*[3], uma feira de eletrônicos mundial. Nessa feira os fabricantes exibem seus produtos inovadores ao

[1] Veículo Aéreo Não Tripulado
[2] *Unmanned Aerial Vehicle*
[3] *Consumer Eletronics Show*

mundo. Em 2012, também na *CES*, a Parrot apresentou ao mundo a versão 2.0 do AR.Drone com inúmeras melhorias. Em consequência dele possuir baixo custo de aquisição houve um avanço significativo em pesquisas relacionadas aos drones. O AR.Drone 2.0, por se tratar de um drone com quatro rotores, possibilita a execução de manobras com alta estabilidade e maior aperfeiçoamento de voo estático, voo em que o veículo está parado no ar. Ainda, de modo que possa mover-se para locais em que há a necessidade de voos mais complexos. Sendo, portanto, considerado uma alternativa para a complexidade e altos custos relacionados aos padrões de helicópteros.

O objetivo desse projeto é saber se é possível tornar voos autônomos viáveis no drone AR.Drone 2.0 bem como a compreensão sobre o seu funcionamento. Para isso será necessário a implementação de um programa de controle para voos autônomos. Esta realização requer a aquisição de dados durante o voo, os quais são obtidos através de sensores que utilizam *Arduino*. Como resultado, para a realização de voos autônomos foi elaborado um framework onde o usuário poderá criar missões de voos autônomos para o drone executa-las. Esses comandos são enviados ao drone pelo usuário devido a utilização de um controle remoto. Esse controle remoto envia dados a um sensor conectado ao *Arduino* que processa os dados e em seguida é lido e interpretado pelo drone.

Este trabalho está dividido em 7 capítulos, sendo o primeiro a introdução. O capítulo 2, apresenta o estado da arte no uso de drones, com exemplos de trabalhos correlatos. No capítulo 3, são apresentadas as especificações do AR.Drone, contendo suas características bem como a descrição dos protocolos de comunicação e modelagem dinâmica do drone de modo a esclarecer a dinâmica de voo. No capítulo 4, há a descrição sobre os critérios de implementação dos softwares desenvolvidos ao decorrer do projeto como, por exemplo, a utilização da linguagem *Java*, o framework *Node.js* e as bibliotecas *Serial-port*, *AR-drone* e *Ardrone-Autonomy*. No capítulo 5, foi desenvolvido 5 cenários de estudo onde de fato houve as implementações dos softwares e a inclusão do *Arduino* ao drone com os sensores utilizados no projeto. Já no capítulo 6, após a fase de explicações e implementações, há os experimentos realizados onde o usuário poderá criar ou executar missões pré-programadas no próprio drone de forma autônoma. Para finalizar, no capítulo 7, há a conclusão do projeto.

Capítulo 2

Trabalhos Relacionados

O Berezny et al. (2012) apresenta uma combinação de hardware e software de maneira a tornar a autonomia aérea mais acessível, tanto em complexidade quanto em programação e em termos de custo. Utilizou o AR.Drone para os experimentos de maneira que o quadricóptero execute várias tarefas autônomas. Para a localização do drone construíram um algoritmo de localização baseado em técnicas de *SURF*[1] utilizando as imagens obtidas nas câmeras como os principais sensores. Para integrar o processamento visual com o controle do AR.Drone de forma eficiente utilizaram um middleware de robótica conhecido como *ROS*[2]. Por fim, conseguiram com que o drone faça voos autônomos em laboratório.

O Bristeau et al. (2011) propõe uma tecnologia de navegação e controle incorporado no AR.Drone. O sistema se baseia no estado-da-arte de sistema de navegação interior e assim combinando baixo custo de sensores inerciais, técnicas de visão computacional, sonar e contabilidade para os modelos de aerodinâmica. Porém, o principal problema é a exigência da estimativa de estado para incorporar vários sensores e tipos distintos. Entre eles, podemos citar os sensores inerciais e câmeras. A estimativa da arquitetura resultante é uma combinação complexa de vários princípios utilizados para determinar distintos horizontes temporais e defeitos de cada sensor. Como resultado, sabe-se que é um sistema sofisticado, entretanto a sua complexidade não é visível pelo usuário, enfatizando o papel do controle automático como uma ativação, mas não como uma tecnologia oculta. Os desenvolvimentos atuais estão focados em games para que seja possível usar esta plataforma em realidade aumentada interativa para *gameplays*.

O Morar e Nascu (2012) descreve o modelo de um processo simplificado para ser feito em condições de voo constante para drones, mais especificamente o AR.Drone. Como proposta, mostraram que pode ser encontrado no comércio um drone barato, podendo ser modelado para ser controlado a fim de realizar tarefas especiais. O objetivo desse processo é de encontrar um modelo simples tão bom quanto possível para o modelo físico do AR.Drone, para

[1] *Speeded Up Robust Features*
[2] *Robot Operating System*

isso, foi necessário descrever todas as especificações físicas do AR.Drone de modo que possam entender sobre as forças que interagem sobre ele no momento em que foi decolado bem como as especificações de softwares. Para implementar o modelo não-linear da aeronave foi desenvolvido um simulador. Através dessas descrições obtiveram uma modelagem dinâmica e matemática do drone, além, é claro, da cinética e a modelagem dos movimentos dos quatro motores. Como conclusão, perceberam que o modelo matemático não linear, implementado em *Simulink*[3], pode ser visto como estável e que os limites para os parâmetros, como ângulos de *Euler*, não foram alcançados, sendo necessário ajustar os parâmetros do modelo, em comparação com o modelo físico, para que o modelo se aproxime cada vez mais do mundo real. O modelo não-linear pode ser linearizado e as funções de transparência obtidas podem ser utilizadas na elaboração de algoritmos para controle do drone. Para os trabalhos futuros, serão desenvolvidos algoritmos de controle e monitoramento para que o quadricóptero possa acompanhar e identificar pequenos objetos.

Diferentemente dos trabalhos relacionados anteriormente citados, este projeto possui como principal contribuição a integração de demais sensores ao drone por meio do *Arduino*. Portanto, todos os novos sensores a serem integrados ao drone serão conectados ao *Arduino* e os dados sensoriais serão lidos pelo drone mediante uma porta serial. A inclusão de novos sensores ao drone possibilitará a obtenção de maiores informações sobre os dados de voo e por consequência torna-se bastante útil na resolução de tarefas específicas em que, para resolve-las, seria necessário a utilização de sensores específicos.

[3]http://www.mathworks.com/products/simulink/

Capítulo 3

Especificações do AR.Drone

O AR.Drone da Parrot possuiu quatro hélices e pode ser controlado por computadores, smartphones ou tablets. Ficou muito conhecido a partir do seu lançamento devido ser um drone de baixo custo e não letal. É utilizado tanto pela academia, em pesquisas científicas, quanto pelas crianças, passando-se, para elas, como um simples brinquedo que voa controlado pelo celular ou até mesmo pelos fotógrafos profissionais quando houver a necessidade de fotos em altura elevada. A Figura 3.1 evidencia as diferenças visuais dos drones apresentados na Tabela 3.1.

Ambos possuem características peculiares conforme percebe-se na Tabela 3.1.

Mediante a Tabela 3.1 nota-se que o AR.Drone 2.0 obteve um aprimoramento no hardware e nos sensores, aumentando o seu poder computacional.

- Câmera vertical: Útil para visualizar o solo e ainda, através de um algoritmo de comparação de imagens mede a velocidade do voo.
- Câmera horizontal: Útil para controlá-lo à distância bem como para utilização em técnicas de mapeamento e localização.

(a) AR.Drone 1.0 (b) AR.Drone 2.0

Figura 3.1: Comparação dos AR.Drones

Tabela 3.1: Tabela de Características do AR.Drone 1.0 e AR.Drone 2.0

Características	**AR.Drone 1.0**	**AR.Drone 2.0**
Câmera horizontal	640 x 480 (VGA)	1280 x 720 (720p HD)
Câmera vertical	176 x 144 (QCIF)	320 x 240 (QVGA))
Acelerômetro	3 eixos	3 eixos
Giroscópio	2 eixos 1 eixo yaw	3 eixos
Sensor de Pressão	Não	Sim
Ultrassom	Sim	Sim
Magnetômetro	Não	3 eixos
Absolute Control	Não	Sim
Kernel Linux	2.6.27	2.6.32
Processador	ARM9 32b 468MHz	ARM Cortex A8 32b 1GHz
RAM	128MB	1GB
WiFi	b/g	b/g/n

- Ultrassom: Possui um sensor de ultrassom para medir a altura do drone, isto é, a distância do drone em relação ao solo..
- Sensor de Pressão[1]: Ótimo para dar mais estabilidade durante os voos verticais.
- Giroscópio: Útil para obter a direção do drone.
- Acelerômetro: Para medir a força aplicada ao drone, informando a sua inclinação num determinado eixo.
- Magnetômetro: Uma bússola *3D*, útil no *Absolute Control*.
- Absolute Control[1]: Recurso interessante no AR.Drone 2.0 pois, através do magnetômetro o referencial para o controle do quadricóptero será o ponto de visão do usuário.

Além dos recursos oferecidos ao desenvolvedor, o drone conta com algumas vantagens relacionadas ao desenvolvimento de software para que seja possível controlá-lo:

- *Source Code* e SDK disponíveis: A *Parrot* disponibiliza o *Source Code* do AR.Drone bem como o *SDK*, *Software Development Kit*, ou simplesmente Kit de Desenvolvimento de Software, em português. Por sua vez facilita e muito o trabalho do desenvolvedor quando o assunto é implementação de algoritmos para controle do AR.Drone.
- Kernel Linux: Com kernel 2.6.27 e 2.6.32 nas versões 1.0 e 2.0, respectivamente, o AR.Drone tornou-se mais adequado aos desenvolvedores adeptos ao uso de software livres uma vez que estão sob licença *GPL*, General Public License.

[1]Recurso presente somente no AR.Drone 2.0

- Documentação do SDK: Possui documentação adequada sobre o *SDK*.
- Baixo custo: Pode-se considerar esse o fator decisivo para a utilização do AR.Drone em pesquisas acadêmicas devido ao seu custo relativamente baixo para um quadricóptero com o conjunto de sensores disponíveis no drone.

Claro que há desvantagens quando observamos que não há uma documentação adequada sobre a parte elétrica do hardware do drone e nem sobre o firmware.

3.1 Comunicação

No capítulo 3.2 está descrito a modelagem dinâmica do AR.Drone 2.0 e agora o foco será descrever como é realizada a comunicação de um computador ou dispositivo com o drone. Basicamente, o drone dispõe de uma conexão sem fio *802.11b* para enviar e receber strings em ASCII com a finalidade de comunicar-se com um dispositivo. Logo, mediante a essa comunicação, pode-se obter os logs dos sensores, status da bateria e ainda dos dados das câmeras e é claro, enviar comandos de *takeoff*(decolagem), *landing*(pouso), *pitch*(movimentação para frente ou para trás), *roll*(movimentação para o lado esquerdo ou direito), *throttle*(para cima ou para baixo) e *yaw*(movimentação sobre o eixo Z).

Conexão WiFi O AR.Drone 2.0 pode ser controlado por qualquer dispositivo que suporte *WiFi* em modo *ad-hoc*. Basicamente, a comunicação é realizada em quatro passos:

1. O drone cria uma rede WiFi para conexão com *ESSID*, *Extended Service Set Identification*, denominado *ardrone_xxx*, alocando um endereço de *IP*, *Internet Protocol*, ao drone.
2. Os dispositivos serão conectados ao drone mediante a conexão com esse *ESSID*.
3. O drone possui um *DHCP*, *Dynamic Host Configuration Protocol*, server para fornecer um endereço de *IP* válido ao dispositivo que se conectou ao AR.Drone 2.0.
4. O servidor *DHCP* do drone garante que a cada dispositivo será associado um único endereço *IP*.

A partir desse momento o dispositivo poderá se comunicar com o drone mediante o envio de requisições através dessa conexão, tendo um endereço de *IP* e porta definidos.

Comunicação entre o AR.Drone 2.0 e o dispositivo O controle, bem como a configuração do drone, são realizados mediante o envio de comandos *AT* na porta *UDP*, *User Datagram Protocol*, 5556. Os comandos podem ser enviados em uma base de tempo regular, geralmente 30 vezes por segundo. Informações e sintaxe dos comandos *AT* serão vistos no capítulo 3.1.1.

Informações sobre o drone como, por exemplo, posição, status, velocidade, velocidade de rotação dos motores, conhecidos como *navdata*, sendo, portanto, enviadas do drone ao cliente através de *UDP* pela porta 5554. Nesses *navdata* também estão inclusas informações de detecção de *tags* para games com o drone. O *stream* de vídeo é enviado pelo AR.Drone através da porta 5555. As imagens do vídeo podem ser decodificadas usando o *codec* incluso no *SDK*. Um canal de comunicação, conhecido como *control port*, pode ser estabelecido na porta *TCP*, *Transmission Control Protocol*, 5559 para transferência de dados críticos, sendo utilizado para recuperar dados de configuração e ainda para reconhecer informações importantes como o envio de informações de configuração para o drone.

3.1.1 Funções de controle do AR.Drone 2.0

Nesse capítulo tem-se os principais comandos para realizar o total controle do drone. Será informado os comandos *AT* correspondentes juntamente com as suas descrições e argumentos.

Modo de emergência Comando *AT* correspondente: **AT*REF**

Args: (int value: emergency flag)

Quando o drone está no estado normal, se enviado o comando com o parâmetro *value* igual a 1 o drone entrará em modo de envio de sinal de emergência, fazendo com que os motores parem e o drone caia. Agora, caso ele já esteja em modo de emergência bastará usarmos *value* igual a 0 para que ele tente voltar ao estado normal. Para encerrar o envio do sinal de emergência será preciso antes verificar se o estado do drone realmente foi alterado para o estado de emergência e, caso sim, basta fazer *value* igual a 0 para interromper o envio do estado de emergência.

Movimentos Comando *AT* correspondente: **AT*PCMD**

int flags: Habilita o controle do drone **Args:** (

- **float phi:** Ângulo direito e esquerdo ∈ [-1;1]
- **float theta:** Ângulo frontal e traseiro ∈ [-1;1]
- **float gaz:** Velocidade vertical ∈ [-1;1]
- **float yaw:** Velocidade angular ∈ [-1;1]

)

O drone, portanto, é controlado por quatro parâmetros:

1. **Ângulo direito e esquerdo:** Valores positivos representando o movimento para a direita e os negativos para a esquerda.

Tabela 3.2: Tabela de resumos dos comandos AT

Comando AT	Argumentos	Descrição
AT*REF	input	Takeoff/Landing/Comando de cessar envio de sinal de emergência
AT*PCMD	flag, roll, pitch, throttle e yaw	Movimenta o drone
AT*FTRIM	-	Configura o drone para se estabilizar no plano horizontal
AT*CONFIG	key, value	Configuração do AR.Drone
AT*LED	animação, frequência, duração	Sequência de animação dos leds
AT*ANIM	animação e duração	Sequência de animações de voo do drone

2. **Ângulo frontal e traseiro:** Valores positivos representando o movimento para trás e os negativos para a frente.

3. **Velocidade vertical:** Representando o movimento de subida com os valores positivos e os de descida com valores negativos.

4. **Velocidade angular em torno do eixo YAW:** Representa o movimento em torno do eixo Z, sendo, valores positivos representando o movimento para a esquerda e os negativos para a direita.

A função da *flag* é decidir se o drone estará em modo *hover* (piloto automático) ou se ele será controlado pelo usuário, sendo assim:

- Bit 0: drone em modo hover
- Bit 1: drone em modo controle

A Tabela 3.2 contém um resumo dos comandos *AT*.

3.1.2 Comandos AT

Os comandos *AT* são strings enviadas ao drone, conforme explicado no capítulo 3.1.

3.1.2.1 Sintaxe

As strings são codificadas em caracteres *ASCII* de 8 bits com um delimitador *<LF>, Line Feed*. Ainda, cada comando é composto por três caracteres **AT*** seguidos pelo nome do

comando e um sinal de igual, um número de sequência e, caso seja necessário, uma lista de argumentos separados por vírgula cujo significado dependerá do comando em questão. É possível enviar mais de um comando *AT* em pacotes *UDP* desde que sejam separados pelo delimitador *<LF>* mas um mesmo comando *AT* não poderá ser enviado em pacotes *UDP* separados. Como exemplo, temos:

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

É de extrema importância que o tamanho total do comando não ultrapasse 1024 bytes, isto é, 1024 caracteres, caso contrário toda a linha de comando será rejeitada. Para alterar o limite será necessário modificar o software interno do AR.Drone 2.0. Vale lembrar que comandos inválidos também serão rejeitados, entretanto é conveniente que o dispositivo envie comandos corretos em *UDP*.

3.1.3 Comandos sequenciais

Com a finalidade de evitar a execução de comandos antigos, um número de sequência é associado a cada comando *AT* de modo que o drone não execute nenhum comando cujo número de sequência seja menor que o último executado por ele. O comando de sequência é o primeiro número depois do igual e a cada novo comando devemos incrementá-lo. A cada nova conexão o número de sequência será retrocedido para o valor 1 no drone. Logo é importante que se envie um comando em no mínimo a cada 2 segundos a fim de não perder a conexão entre o dispositivo e o drone.

Portanto, deve-se seguir as seguintes regras:

- Sempre enviar o valor 1 como o primeiro número de sequência ao drone.
- O valor de sequencia deverá ser crescente, ou seja, incrementado a cada novo comando enviado ao AR.Drone 2.0.

3.1.3.1 Descrição dos comandos

A seguir será descrito cada comando aceito pelo AR.Drone de modo a obter total conhecimento e clareza no desenvolvimento de softwares de controle para o drone.

AT*REF Controla o comportamento básico do AR.Drone, isto é, os comandos de *take-off, landing, emergency, stop e reset.*

Sintaxe: AT*REF=%d,%d<LF>

- Argumento 1: O número de sequência
- Argumento 2: Um valor inteiro de 32b $[0..2^{32}-1]$

Exemplo: **AT***REF=1,290718208<**LF**>**AT***REF=2,290717696<**LF**>
Nesse exemplo, o primeiro comando aciona o *take-off* e o segundo o *landing*.

AT*PCMD Esse comando é responsável por fazer com que o drone se movimente.
Sintaxe: AT*PCMD=%d,%d,%d,%d,%d,%d<LF>

- Argumento 1: Número de sequência
- Argumento 2: Flag de controle do drone
- Argumento 3: Roll ∈ [-1;1]
- Argumento 4: Pitch ∈ [-1;1]
- Argumento 5: Throttle ∈ [-1;1]
- Argumento 6: Yaw ∈ [-1;1]

A função da *flag* é decidir se o drone estará em modo *hover* (piloto automático) ou se ele será controlado pelo usuário, sendo assim:

- Bit 0: drone em modo hover
- Bit 1: drone em modo controle

AT*FTRIM Este comando deve ser chamado toda vez que o drone entrar no estado de *take-off* para que ele possa se estabilizar em um plano horizontal automaticamente. Logo, ele ajustará automaticamente os movimentos de *pitch* e *roll* para que a estabilização esteja completa. Caso não utilize o *FTRIM* o drone poderá sofrer uma desestabilização no momento em que estiver decolando, não sabendo, portanto, a sua inclinação real.
Sintaxe: AT*FTRIM=%d<LF>

- Argumento 1: Número de sequência

AT*CONFIG Este comando é responsável por pré-ajustar as configurações padrões do drone como, por exemplo, o ajuste da altura máxima.
Sintaxe: AT*CONFIG=%d,%s,%s<LF>

- Argumento 1: Número de sequência
- Argumento 2: Nome da opção de configuração entre aspas duplas.
- Argumento 3: Valor da configuração, também entre aspas duplas

AT*LED Este comando é responsável por fazer com que os *LEDs*, *Light Emitting Diode*, ou Diodo Emissor de Luz em português, dos quatro motores pisquem em uma sequência definida.

Sintaxe: AT*LED=%d,%d,%d,%d<LF>

- Argumento 1: Número de sequência
- Argumento 2: Inteiro para iniciar a animação
- Argumento 3: float, definindo a frequência, em Hz, da animação
- Argumento 4: inteiro, duração total da animação, lembrando que a animação é executada da seguinte forma: duração x frequência da animação

AT*ANIM É responsável por executar movimentos pré-definidos.

Sintaxe: AT*ANIM=%d,%d,%d<LF>

- Argumento 1: Número de sequência
- Argumento 2: inteiro, definindo qual animação será executada
- Argumento 3: inteiro, definindo a duração total, em segundos, da animação

Demais informações podem ser encontradas na documentação do *SDK* da Parrot AR.Drone em Ar.Drone (2012).

3.2 Modelagem Dinâmica

Entender a modelagem dinâmica do AR.Drone 2.0 é essencial para a realização de qualquer experimento. Para um voo autônomo, por exemplo, é necessário pleno entendimento sobre as forças atuantes no drone uma vez que qualquer desequilíbrio discrepante tornará o voo inviável. Por esse motivo os desenvolvedores, ao implementar o sistema de estabilização, devem obter o entendimento sobre a modelagem dinâmica do drone. Na Figura 3.2 é apresentado o modelo simplificado do drone composto por quatro circunferências representando os rotores e os sentidos de rotação das hélices, as forças produzidas pelos rotores sendo as flechas perpendiculares a eles bem como os eixos X_B, Y_B, Z_B. Sobre o eixo X_B têm-se os rotores *Left* e *Right* girando com as hélices em sentido horário enquanto no eixo Y_B os rotores *Front* e *Rear* girando em sentido anti-horário de modo a eliminarem o efeito do torque produzidos pelos rotores.

O modelo dinâmico do drone é descrito, portanto, mediante as leis de Newton e Euler. O momento inercial é calculado assumindo que o quadricóptero pode possuir em seu centro uma esfera de raio r e massa M_0 suportando quatro outras massas, os motores no caso. Considerando que cada motor tem a massa m e estão anexados ao centro por um eixo de tamanho l. Este modelo pode ser visto na Figura 3.3.

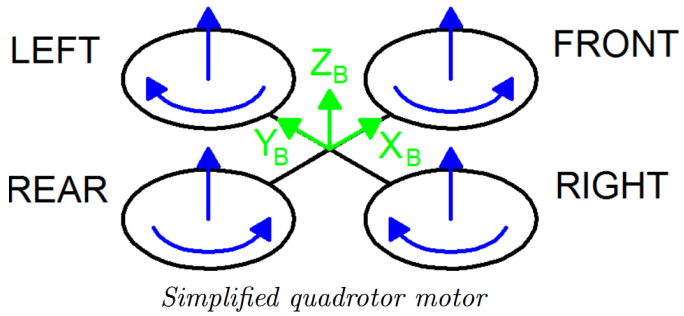

Figura 3.2: Modelo simplificado do Drone

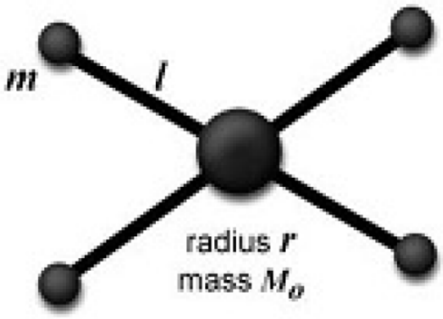

Figura 3.3: Modelo do centro de massas

3.2.1 Hover

Apresentado, na Figura 3.4, o modelo de movimento hover é o movimento responsável por manter o drone suspenso no ar, ou seja, modo estacionário denominado Hover. Para que esse movimento seja possível todas as hélices do drone devem girar com velocidade constante (Ω_h), fazendo com que as forças geradas pelos rotores sejam a mesma e sem variar a sua inclinação.

3.2.2 Throttle

Esse movimento é obtido mediante a variação de velocidade das hélices de modo que elas tenham a mesma velocidade, ou seja, incrementando ou decrementando a variável Ω_H com uma variável Δ_A, $(\Omega_H + \Delta_A)$. Isso faz com que o drone se desloque sobre o eixo Z_B verticalmente conforme observado na Figura 3.5.

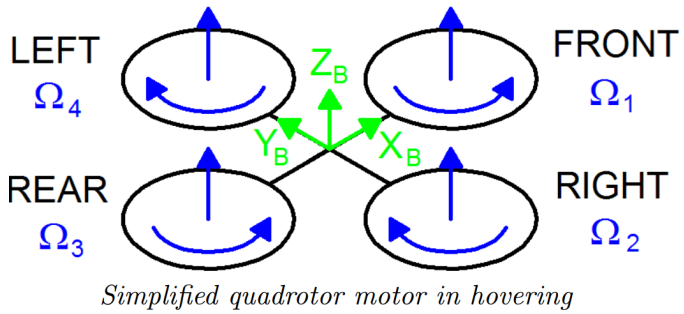

Figura 3.4: Modelo do movimento Hover

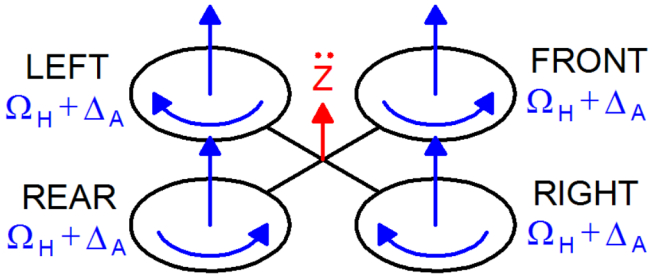

Figura 3.5: Modelo do movimento Throttle

3.2.3 Pitch

Para efetuar a inclinação do drone de modo que ele fique em posição de voo para frente ou para trás deve-se levar em consideração o modelo apresentado na Figura 3.6. Para isso, há a necessidade de modificar a velocidade de rotação das hélices *Rear* e *Front*, somando uma variável Δ_A em *Rear* $(\Omega_H + \Delta_A)$ e subtraindo uma variável Δ_B em *Front* $(\Omega_H - \Delta_B)$, ou seja, dos rotores situados ao longo do eixo X_B e mantendo os demais com a mesma velocidade (Ω_H). A partir desse modelo o drone será inclinado devido a um torque adquirido em relação ao eixo Y_B necessário para realizar tanto voo para frente quanto para trás. Para manter o impulso vertical inalterado são escolhidos valores positivos para as variáveis Δ_A e Δ_B, não sendo distantes demais. E, ainda, para valores pequenos de Δ_A tem-se que Δ_A será aproximado a Δ_B, isto é, $(\Delta_A \approx \Delta_B)$ Bresciani (2008).

3.2.4 Roll

Semelhante ao Pitch do capítulo 3.2.3, entretanto, nesse movimento, conforme descrito pela Figura 3.7, deve-se modificar a velocidade de rotação das hélices *Left* e *Right*, somando uma

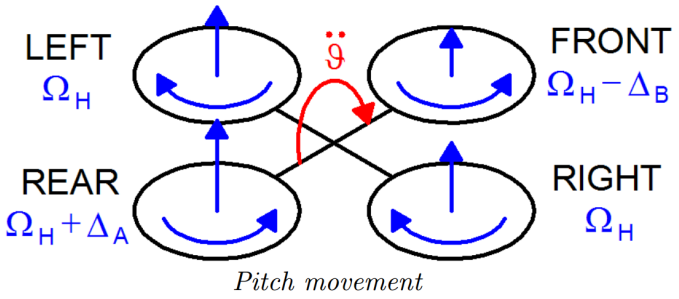

Figura 3.6: Modelo do movimento Pitch

variável Δ_A em *Left* $(\Omega_H + \Delta_A)$ e subtraindo uma variável Δ_B em *Right* $(\Omega_H - \Delta_B)$, ou seja, dos rotores situados ao longo do eixo Y_B e mantendo os demais com a mesma velocidade (Ω_H). A partir desse modelo o drone será inclinado devido a um torque adquirido em relação ao eixo X_B necessário para realizar tanto voo para a direita quanto para a esquerda.

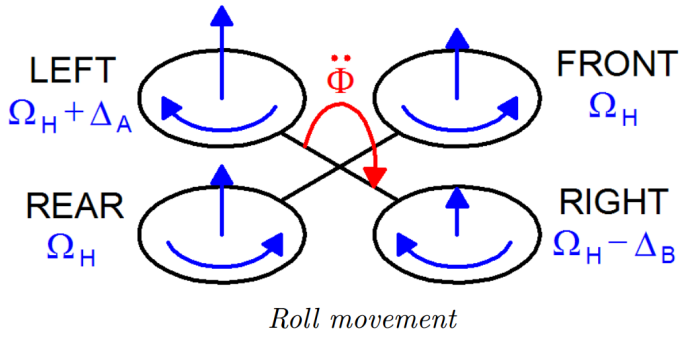

Figura 3.7: Modelo do movimento Roll

3.2.5 Yaw

Por fim, o movimento *Yaw* responsável pelo drone girar em torno do eixo Z_B devido ao torque gerado pelo rotores, onde altera-se as velocidades das quatro hélices conforme verificado na Figura 3.8.

Nesse modelo, aos rotores situados em torno do eixo Y_B, soma-se uma variável Δ_A, $(\Omega_H + \Delta_A)$ e subtrai-se uma variável Δ_B em torno do eixo X_B $(\Omega_H - \Delta_B)$.

A partir dos quatro movimentos descritos anteriormente pode-se derivar os outros dois movimentos possíveis para um corpo com seis graus de liberdade, isto é, ao realizar o movimento de *Pitch* juntamento com o movimento de *Throttle*, verifica-se que as forças produzidas pelos rotores poderão ser decompostas em componentes sobre o eixo X_B, para que se possa mudar o ângulo do sistema em torno do eixo Y_B. Claro, desde que a componente sobre o eixo

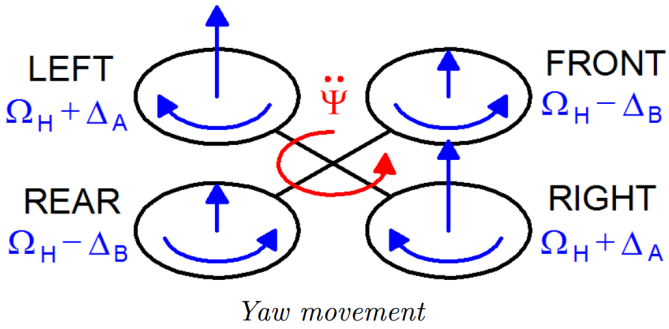

Figura 3.8: Modelo do movimento Yaw

Z_B seja igual a força da gravidade, tendo o único movimento em torno do eixo X_B. O mesmo ocorre ao realizar o *Roll*, mudando o ângulo em X_B, combinado ao movimento *Throttle*, para manter a componente da força sobre o eixo Z_B igual a força da gravidade, será produzido um movimento sobre o eixo Y_B.

3.2.6 Movimento Angular

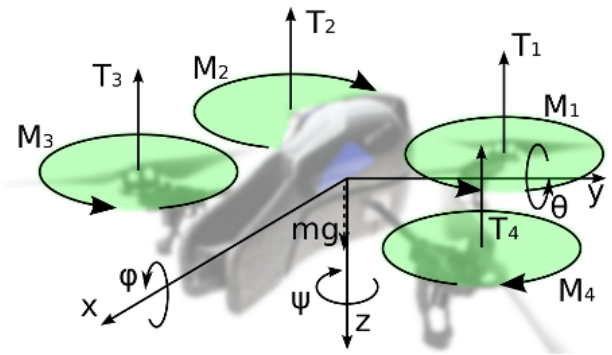

Figura 3.9: Modelo do movimento angular

Percebe-se, na Figura 3.9, que ao utilizar as associações de movimentos, combinado ao movimento *Yaw* do capítulo 3.2.5, o drone poderá ser movimentado em qualquer direção do espaço tridimensional. Logo, entender a inclinação do sistema é essencial para controlar, de fato, todos os movimentos do sistema Enomoto (2010).

Capítulo 4

Implementação de software

Sendo necessária a implementação de um software para realizar os testes de controle do AR. Drone 2.0 foi preciso, primeiramente, entender os protocolos de comunicação como é descrito no capítulo 3.1. Tendo como base o entendimento dos protocolos foi desenvolvido dois softwares para controlar o drone. O primeiro em Java e o segundo utilizando o framework *Node.js* que será explicado na secão 4.2.

4.1 Implementação em Java

Nessa etapa foi desenvolvida uma aplicação em *Java* utilizando a *API JavaDrone* JavaDrone (2014). Trata-se de uma *API* em *Java* que implementa os protocolos de comunicação do drone. Assim torna-se fácil a implementação de aplicações e controle em *Java* para o AR.Drone.

Para essa aplicação foi desenvolvida uma interface gráfica bem como uma classe *Controle.java* e *Sensor.java*:

- *Controle.java*: Classe responsável por gerir os controles de movimento do drone.
- *Sensor.java*: Classe responsável por obter e armazenar em um arquivo de texto os dados sensoriais coletados do drone.

Na Figura 4.1 temos a interface gráfica, elaborada de maneira bem simples para ser utilizada nos testes de voo da aplicação.

No Algoritmo 4.1, a classe *Controle.java*, para facilitar a utilização da *API JavaDrone*, foi implementado alguns métodos para abstrarir os parâmetros de movimentação passados como argumento no método *drone.move()*, onde:

- *hover()*: Comando responsável por estabilizar o drone no ar.
- *moveUp()*: Mover o drone para cima.

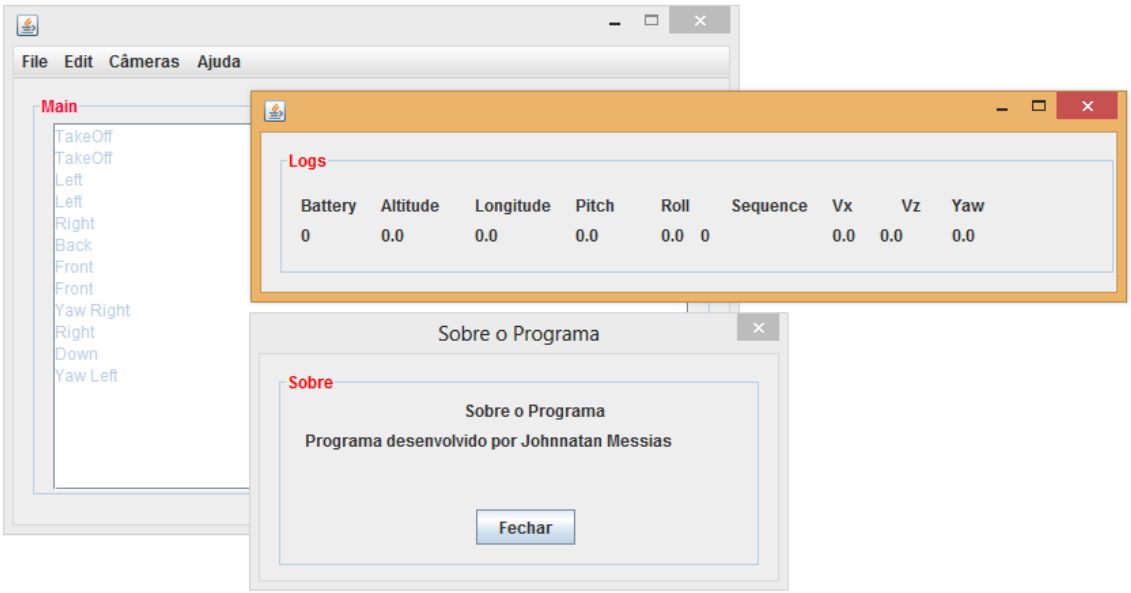

Figura 4.1: Interface Gráfica da Aplicação em Java

- *moveDown()*: Mover o drone para baixo.
- *moveFront()*: Mover o drone para frente.
- *moveBack()*: Mover o drone para trás.
- *moveRight()*: Mover o drone para a direita.
- *moveLeft()*: Mover o drone para a esquerda.
- *moveYawRight()*: Mover o drone, em relação ao *eixo Z*, para a direita.
- *moveYawLeft()*: Mover o drone, em relação ao *eixo Z*, para a esquerda.
- *takeOff()*: Aciona o modo decolagem do drone, fazendo-o voar.
- *land()*: Aciona o modo pouso.
- *disconnect()*: Necessário para fechar a conexão com o drone.

Através da classe *Sensor.java* pode-se efetuar a coleta dos dados sensoriais como pode-se observar no Algoritmo 4.2. A coleta é necessária para a realização de estudos da dinâmica de voo do drone como é observado no capítulo 5.1. Nesse experimento são coletados os seguintes dados: Altitude, Giroscópio e Acelerômetro.

```
public class Controle {

    public ARDrone drone;
    private Sensores sensores;
    private static String nameFile;

    public Controle() throws UnknownHostException, IOException {
        init();
    }

    private void init() throws UnknownHostException, IOException {
        drone = new ARDrone();
        drone.connect();
        drone.playAnimation(10, 10);
        sensores = new Sensores(nameFile);
        drone.addNavDataListener(sensores);
    }

    public static void setNameFile(String name) {
        nameFile = name;
    }

    public String getSensores() {
        return sensores.getData();
    }

    public void hover() throws IOException {
        drone.hover();
    }

    public void moveUp() throws IOException {
        drone.move(0f, 0f, 1f, 0f);
    }

    public void moveDown() throws IOException {
        drone.move(0f, 0f, -1f, 0f);
    }

    public void moveFront() throws IOException {
        drone.move(0f, -1f, 0f, 0f);
    }

    public void moveBack() throws IOException {
        drone.move(0f, 1f, 0f, 0f);
    }

    public void moveRight() throws IOException {
```

```
        drone.move(1f, 0f, 0f, 0f);
    }

    public void moveLeft() throws IOException {
        drone.move(-1f, 0f, 0f, 0f);
    }

    public void moveYawRight() throws IOException {
        drone.move(0f, 0f, 0f, -1f);
    }

    public void moveYawLeft() throws IOException {
        drone.move(0f, 0f, 0f, 1f);
    }

    public void takeOff() throws IOException {
        if (drone.isEmergencyMode()) {
            drone.clearEmergencySignal();
        }
        drone.trim();
        drone.takeOff();
    }

    public void land() throws IOException {
        drone.land();
    }

    public void disconnect() throws IOException {
        sensores.closeFile();
        drone.removeNavDataListener(sensores);
        drone.disconnect();
    }

    public ARDrone getDrone() {
        return drone;
    }
}
```

Algoritmo 4.1: Código Controle.java

```
public class Sensores implements NavDataListener {
    private String datas;
    private String printFile;
    private PrintWriter printWriter;

    public Sensores(String fileName) {
        try {
            printWriter = new PrintWriter(fileName);
```

```
            printWriter.write("Timestamp;Bateria;Altitude;Vy;"
                    + "Pitch;Roll;Sequence;Vx;Vz;Yaw" + "\n");
        } catch (FileNotFoundException ex) {
            Logger.getLogger(Sensores.class.getName()).log(Level.SEVERE, null,
                ex);
        }
    }

    @Override
    public void navDataReceived(NavData nd) {

        datas = "Bateria=" + nd.getBattery() + "%" + "\t"
        + "Altitude=" + nd.getAltitude() + "\t" + "Vy="
        + nd.getLongitude() + "\t" + "Pitch="
        + nd.getPitch() + "\t" + "Roll=" + nd.getRoll()
        + "\t" + "Sequence=" + nd.getSequence() + "\t"
        + "Vx=" + nd.getVx() + "\t" + "Vz=" + nd.getVz()
        + "\t" + "Yaw=" + nd.getYaw();

        printFile = getTimestamp() + ";" + nd.getBattery() + ";"
        + nd.getAltitude() + ";"
        + nd.getLongitude() + ";"
        + nd.getPitch() + ";" + nd.getRoll()
        + ";" + nd.getSequence() + ";"
        + nd.getVx() + ";" + nd.getVz()
        + ";" + nd.getYaw();
        printWriter.write(printFile + "\n");
        System.out.println(datas);
    }

    public PrintWriter getFile() {
        return printWriter;
    }

    private String getTimestamp() {
        Date date = new Date();
        String dateTimestamp = String.valueOf(date.getTime() / 1000);
        return dateTimestamp;
    }

    public void closeFile() {
        printWriter.close();
    }
}
```

Algoritmo 4.2: Código Sensor.java

4.2 Implementação em Node.js

Nessa etapa foi utilizado o framework *Node.js* Aaron (2014). Esse framework, orientado a evento, é uma plataforma desenvolvida sobre a *engine JavaScript V8*, desenvolvida pelo Google para seu navegador WEB Google Chrome. O *Node* provê um ambiente de execução *server-side* que complica e executa *JavaScript* com muita rapidez. A grande velocidade de execução é devido ao fato de que a *engine V8* compila *JavaScript* em código de máquina nativo ao invés de interpretá-lo ou executá-lo como *bytecode*. *Node* é um projeto *open source* e pode ser executado em Mac OSX, Windows e Linux.

4.2.1 Biblioteca Ardrone-Autonomy

O framework *Node* também provê a possiblidade de utilizar bibliotecas. *ArDrone-Autonomy* é uma biblioteca para voos autônomos desenvolvida especificamente para o Ar.Drone, construída no topo da biblioteca *Ar-Drone* para o *Node* Ardrone-Autonomy (2014). A partir dela pode-se planejar e executar missões autônomas, descrevendo o caminho, altitude e orientação que o drone deve seguir.

Características da Ardrone-Autonomy Essa biblioteca apresenta características importantes para a execução de voos autônomos como, por exemplo:

- Filtro de Kalman Extendido Macharet (2009): Utilização do Filtro para prover uma melhor estimativa de estado e utilizar detecção de *TAG* como o ponto e partida para o Filtro.
- Projeção da câmera: Para estimar a posição de um objeto detectado pela câmera. Atualmente usado para estimar a posição da *TAG* baseado em sistemas de coordenadas do drone. Utiliza a câmera vertical.
- Controlador de PID: Para controlar de maneira autônoma a posição do drone.
- Planejador de missões: Para preparar planos de voos e tarefas para serem executados.

No Algoritmo 4.3 podemos verificar um código de exemplo para exemplificar o funcionamento básico da biblioteca *Ardrone-Autonomy* para o framework *Node* onde:

- *mission.takeoff()*: Responsável por fazer o drone decolar.
- *mission.land()*: Responsável por acionar o pouso do drone.
- *mission.altitude(value)*: Responsável por definir a altitude do drone, em metros, em relação ao solo.

- *mission.forward(value)*: Movimenta o drone para a frente, em metros.
- *mission.right(value)*: Movimenta o drone para a direita, em metros.
- *mission.backward(value)*: Movimenta o drone para trás, em metros.
- *mission.left(value)*: Movimenta o drone para a esquerda.
- *mission.up*: Movimenta o drone para cima, em metros.
- *mission.down*: Movimenta o drone para baixo, em metros.
- *mission.hover(value)*: Aciona o modo de voo estacionário onde o drone fica o mais estável possível no ar, em milesegundos.
- *mission.run(callback)*: Responsável por executar a missão, tendo como resposta de chamada a *function(err, result)* que poderá ser chamada em caso de erro ao final da missão.
- *mission.cw(angle)*: Aciona o giro sobre o eixo Z do drone, ângulos em graus.
- *mission.log(path)*: Log da missão em arquivo *csv* formatado.
- *mission.wait(value)*: Faz com que o próximo comando seja executado após um tempo, em milisegundos, definido no parâmetro da função.
- *mission.go(position)*: O drone irá para uma posição dada antes de executar o próximo comando. A posição é da forma x: 0, y: 0, z: 0, yaw: 90.
- *mission.task(function(callback){...})*: Executa uma função fornecida antes de executar o próximo passo.
- *mission.taskSync(function)*: Adiciona uma função antes de executar o próximo passo, sendo uma tarefa.
- *mission.zero()*: Adiciona o estado inicial da missão, configurando o estado/orientação corrente como o estado base do Filtro de Kalman como, por exemplo, x: 0, y: 0, yaw: 0.

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

mission.takeoff()
       .zero()
       .altitude(1)
       .forward(2)
       .right(2)
       .backward(2)
```

```
        .left(2)
        .hover(1000)
        .land();

mission.run(function (err, result) {
    if (err) {
        console.trace("Oops, something bad happened: %s", err.message);
        mission.client().stop();
        mission.client().land();
    } else {
        console.log("Mission success!");
        process.exit(0);
    }
});
```

Algoritmo 4.3: Código de exemplo da Ardrone-Autonomy

Capítulo 5

Cenários de Pesquisa

Para alcançar o objetivo desse projeto, saber se é possível tornar voos autônomos viáveis no AR.Drone 2.0, o projeto foi dividido em 5 cenários, ou etapas. Em cada cenário é descrito as mudanças do projeto como, por exemplo, a mudança da linguagem *Java* para a utilização do framework *Node.js* e a integração do *Arduino* ao drone. Logo em cada novo cenário será apresentado melhorias e modificações no projeto. Os cenários são divididos em:

- Cenário 1: Desenvolvimento de uma aplicação em Java para enviar comandos ao AR.Drone. Nesse sentido pode-se enviar comandos necessários para a realização de voo bem como coletar os dados sensoriais do drone. Entretanto, após a realização desse experimento foi verificado que abordar voos autônomos com a linguagem *Java* seria mais trabalhoso uma vez que há ferramentas mais adequadas para esse tipo de tarefa. Logo, tornou-se necessário buscar uma nova solução, uma linguagem ou framework que proporcionasse maior abstração para a realização de voos autônomos. Será apresentado no cenário 2.

- Cenário 2: Nessa etapa foi utilizado um framework e reimplementada a aplicação para a realização de voo, entretanto em *JavaScript* mediante o framework *Node.js*. Devido a utilização desse framework foi possível utilizar bibliotecas como *AR-Drone, Ardrone-Autonomy*, para que fosse viável criar missões de voos autônomos, e *Serial-Port* e a comunicação direta do drone com o *Arduino*.

- Cenário 3: Anteriormente todos os algoritmos eram executados em computadores externos (laptop, desktop, celulares) e repassado somente os comandos *AT* ao drone, entretanto, para proporcionar maior nível de autonomia foi preciso verificar a possibilidade de o próprio drone executar os algoritmos desenvolvidos utilizando o seu próprio processador e memória principal. Isso foi possível nesse cenário.

- Cenrário 4: Foi preciso integrar o *Arduino* ao drone pois para a realização de voos autônomos torna-se necessário incluir novos sensores ao drone. A integração de sensores

é realizada mediante o *Arduino*. Portanto, todos os novos sensores a serem integrados ao drone serão conectados ao *Arduino* e os dados sensoriais serão lidos pelo drone mediante uma porta serial. Essa etapa é uma das principais contribuições desse projeto.

- Cenário 5: Cenário final onde é implementado um framework para enviar comandos ao drone. Esse framework permite a realização de voos autônomos onde o usuário poderá criar missões de voos autônomos, de forma simplificada, para o drone executa-las. Esses comandos são enviados ao drone pelo usuário devido a utilização de um controle remoto. Esse controle remoto envia dados a um sensor conectado ao *Arduino* que processa os dados e em seguida é lido e interpretado pelo drone.

5.1 Cenário 1

Inicialmente foi desenvolvida uma aplicação em Java utilizando a *API JavaDrone* para gerir os protocolos de comunicação do AR.Drone como descrito no capítulo 3.1. O foco principal desse cenário é realizar testes para saber se os protocolos de comunicação são coerentes. Assim, após a realização dos testes foi comprovada a coerêcia dos comandos. A aplicação pode enviar comandos necessários para a realização de voo bem como coletar dados sensoriais para tornar possível o estudo da dinâmica de voo do drone.

Como exemplo, foi elaborado um teste simples, Algoritmo 5.1, e plotado alguns gráficos da dinâmica de voo obtida pelos testes. Através da coleta dos dados dos sensores foi realizada as plotagens dos gráficos de *Altitude, Roll, Pitch, Vx, Vy, Vz e Yaw.*

Altitude: Basicamente mede o comportamento do estado *Throttle* uma vez que dependendo da variação da rotação de todas as hélices com a mesma velocidade, sendo um acréscimo ou decréscimo, o drone terá sua altura variada conforme verificamos no gráfico resultante do código descrito no Algoritmo 5.1. Veja Figura 5.1(a).

Roll: O gráfico abaixo representa o movimento responsável por fazer o AR.Drone 2.0 inclinar-se para direita ou esquerda. Sendo a parte positiva do gráfico representando o movimento para a direita e a negativa a esquerda. Veja Figura 5.1(b).

Pitch: Similares ao *Roll*, entretanto, esse movimento é responsável por fazer o drone ir para frente(parte positiva) ou para trás (parte negativa). Veja figura 5.2(a).

Yaw: Esse movimento é responsável por fazê-lo girar sobre o seu próprio eixo. Veja figura 5.2(b).

Após a realização desse experimento foi verificado que abordar voos autônomos com a linguagem *Java* seria mais trabalhoso uma vez que há ferramentas mais adequadas para esse tipo de tarefa. Logo, tornou-se necessário buscar uma nova solução, uma linguagem ou framework que proporcionasse maior abstração para a realização de voos autônomos. Será apresentado no cenário 2.

```
public class Ardrone {
```

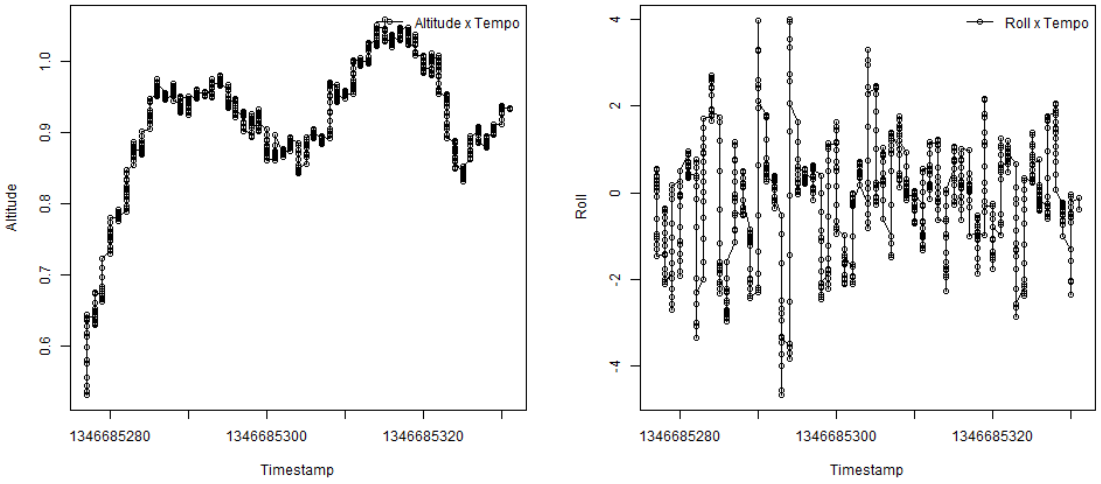

Figura 5.1: (a) Descricao de voo da Altitude (b) Descricao de voo Roll

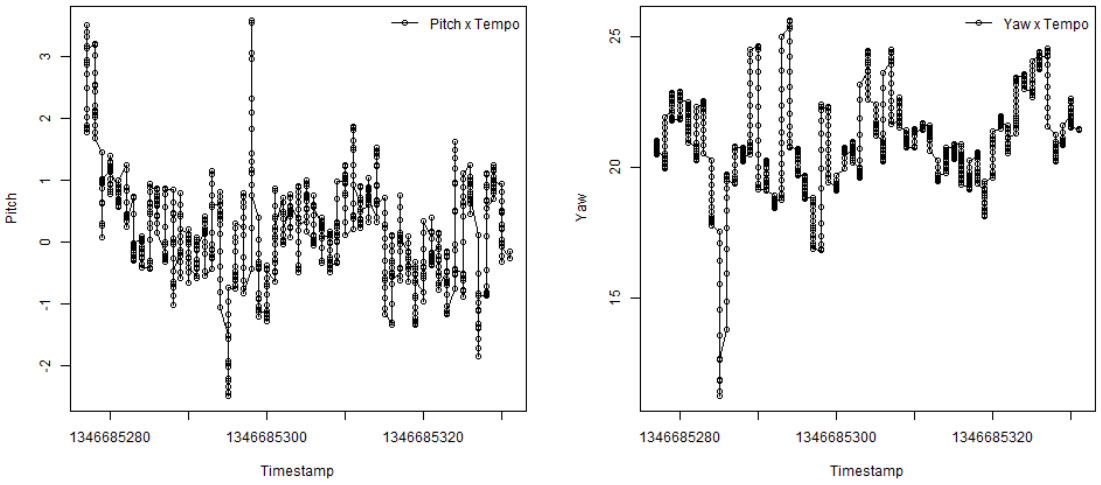

Figura 5.2: (a) Descricao de voo Pitch (b) Descricao de voo Yaw

```
    private static ARDrone drone;
    public static void main(String[] args) {
        try {
            drone = new ARDrone();

            System.out.println("Connect");
            drone.connect();

            System.out.println("clearEmergencySignal");
```

```
            drone.clearEmergencySignal();

            System.out.println("waitForReady");
            drone.waitForReady(3000);

            System.out.println("trim");
            drone.trim();

            Sensores sensores = new Sensores("sensorLog.txt");
            drone.addNavDataListener(sensores);

            System.out.println("Takeoff");
            drone.takeOff();

            Thread.sleep(50000);

            System.out.println("Land");
            drone.land();

            Thread.sleep(2000);
            System.out.println("Disconnect");
            sensores.closeFile();
            drone.disconnect();
        } catch (InterruptedException ex) {
            Logger.getLogger(Ardrone.class.getName()).log(Level.SEVERE, null,
                ex);
        } catch (IOException ex) {
            Logger.getLogger(Ardrone.class.getName()).log(Level.SEVERE, null,
                ex);
        }
    }
}
```

Algoritmo 5.1: Teste de voo com Java

5.2 Cenário 2

Esse cenário é bem similiar ao cenário 1, entretanto, agora, utiliza-se o framework *Node.js*. Ainda, pode-se utilizar a biblioteca *Ardrone-Autonomy* como descrito no capítulo 4.2. Essa biblioteca permite a elaboração de voos autônomos de maneira mais simples uma vez em que implementa características importantes como:

- Filtro de Kalman Extendido Macharet (2009): Utilização do Filtro para prover uma melhor estimativa de estado e utilizar detecção de *TAG* como o ponto e partida para o Filtro.

- Projeção da câmera: Para estimar a posição de um objeto detectado pela câmera. Atualmente usado para estimar a posição da *TAG* baseado em sistemas de coordenadas do drone. Utiliza a câmera vertical.
- Controlador de PID: Para controlar de maneira autônoma a posição do drone.
- Planejador de missões: Para preparar planos de voos e tarefas para serem executados.

Logo, nesse cenário, houve a necessidade de descartar o cenário 1 e agora utilizar o framework *Node.js*. Trata-se de um framework onde o desenvolvimento é realizado em *JavaScript*. Devido a utilização desse framework foi possível utilizar bibliotecas como *AR-Drone, Ardrone-Autonomy*, para que fosse viável criar missões de voos autônomos, e *Serial-Port* e a comunicação direta do drone com o *Arduino*.

5.3 Cenário 3

Até o cenário 2 todos os códigos eram executados por computadores externos como, por exemplo, desktops, laptops ou até mesmo celulares. Para deixar o drone mais flexível de modo que não dependa de um dispositivo externo tornou-se necessário viabilizar a execução dos códigos pelo próprio drone. Nesse sentido o drone se tornaria menos dependente de outros dispositivos. Para realizar essa etapa foi necessário verificar uma maneira de conectar-se ao drone via *WiFi* que ele mesmo disponibiliza. Através de acesso *TELNET* Postel e Reynolds (1983), via *IP: 192.168.1.1*, foi possível acessar o *Kernel Linux* do AR.Drone 2.0, como pode ser conferido na Figura 5.3. A partir do acesso ao *Kernel* pode-se ter acesso ao Sistema Operacional *Linux* embutido no AR.Drone 2.0, utilizando o processador e a memória principal do drone para a execução de programas. Após obter a conexão foi necessário realizar testes de execução de código em *JavaScript* no drone. Para isso foi fundamental seguir os passos:

1. Conectar o dispositivo (laptop, desktop ou celular) à rede *wireless* provida pelo próprio AR.Drone 2.0.
2. Preparar o ambiente para a execução dos códigos em *JavaScrip* pelo *Node.js*. Nesse caso, para execução interna no drone, foi preciso copiar para um pendrive o executável do framework *Node.js* para *Linux*.
3. Incluir as bibliotecas *ar-drone* e *ardrone-autonomy* para *Node.js* ao drone.
4. Transferir os códigos em *JavaScript* também para um pendrive.
5. Executar os códigos utilizando os arquivos e o node contidos no pendrive.

Para acessar arquivos contidos no pendrive pelo drone basta acessar o diretório */data/video/usb0/*. Na Figura 5.4 pode-se conferir a execução do Algoritmo 4.3 pelo próprio drone através do comando */data/video/usb0/./node testautonomy.js*. Onde o arquivo *node* é o executável responsável por rodar o framework *Node.js* e *testautonomy.js* o arquivo com os comandos em *JavaScript* para a execução.

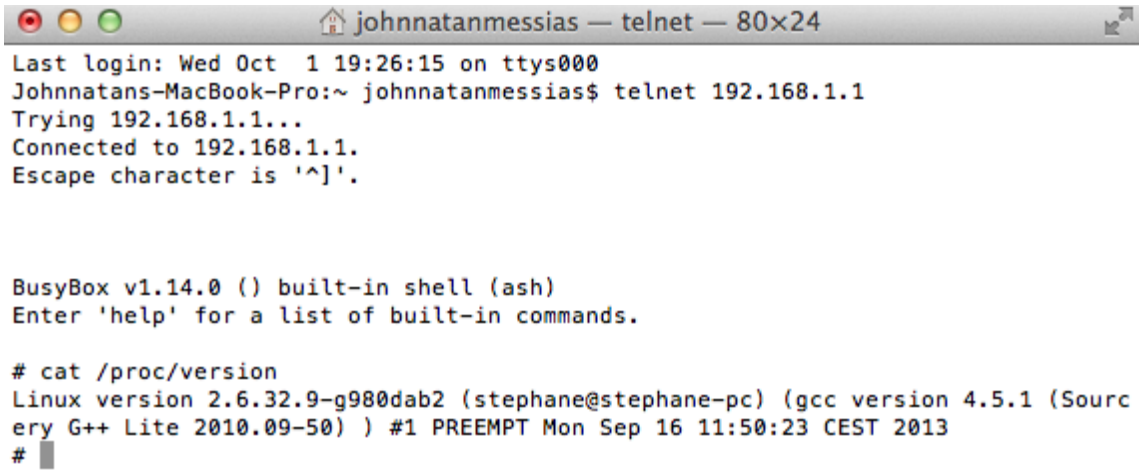

Figura 5.3: Teste de Conexão TELNET

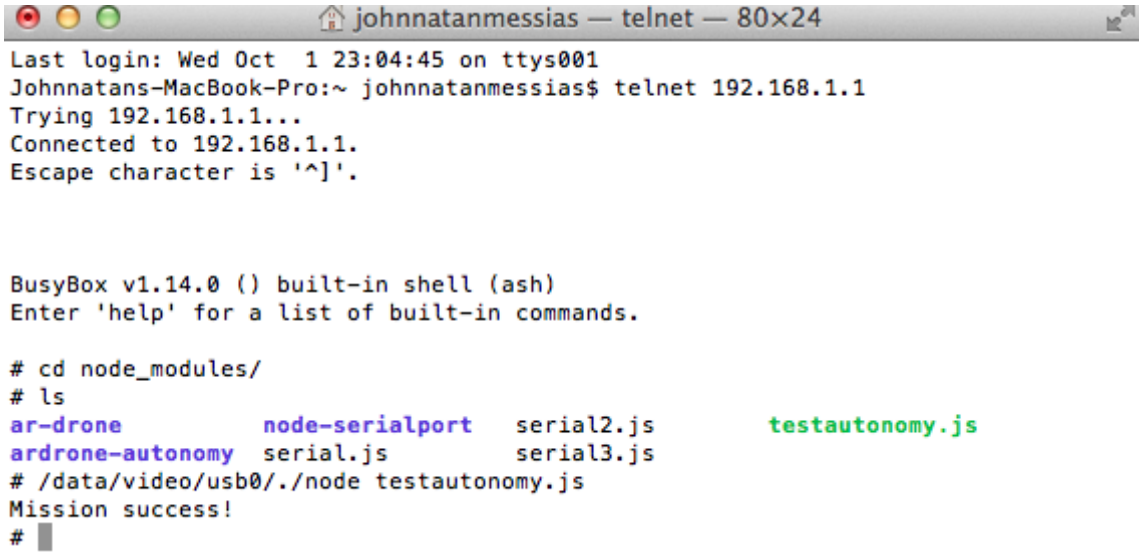

Figura 5.4: Teste de voo Testautonomy.js

5.4 Cenário 4

O AR.Drone 2.0 possui um conjunto de sensores disponíveis como é descrito no capítulo 3 porém torna-se necessário a inclusão de mais sensores a fim de coletar maiores informações sobre o ambiente. Para tornar voos autônomos viáveis é imprescindível a utilização de sensores e a escolha de quais sensores utilizar. Para o planejamento do voo dependerá do objetivo a ser conquistado. Pensando nisso torna-se necessária a inclusão de mais sensores ao drone. A estratégia utilizada foi viabilizar a inclusão do *Arduino* ao AR.Drone 2.0. Dessa maneira seria possível incluir sensores ao *Arduino* e obter os dados sensoriais pelo drone, realizando uma leitura dos mesmos pela porta *USB* contida na parte de baixo do AR.Drone 2.0. Essa etapa é uma das principais contribuições desse projeto.

Os passos essenciais para realizar esse cenário foram:

1. Identificar e entender as conexões *USB* da parte inferir do drone.
2. Criar um código simples para rodar no *Arduino*. Nesse caso foi a escrita na porta serial de uma string *Hello World*.
3. Incluir a biblioteca *node-serialport* Serial-port (2014) ao drone para a leitura dos dados enviados pelo *Arduino* ao drone mediante a porta serial.
4. Implementar o código de leitura em *JavaScript* para a leitura dos dados.
5. Conectar o *Arduino* ao drone.
6. Verificar a leitura, executando os códigos implementados.

A empresa Parrot, fabricante do AR.Drone 2.0, não fornece informações detalhadas sobre o *hardware* do drone, ainda, não fornece a descrição da porta *USB* o que levou a um trabalho de tentativa e erro. Após o entendimento da porta *USB* foi possível conectar o *Arduino* ao drone. A descrição da porta *USB* pode ser verificada na Figura 5.5 e o esquema de conexão na Figura 5.6. Vale lembrar que a porta *USB* do drone possui 4 (quatro) pinos, sendo:

- *Rx*: Para receber os dados enviados pela porta. Correspondente ao Pino 3 e deve ser conectado ao Pino *Tx* do *Arduino*.
- *Tx*: Para enviar os dados. Correspondente ao Pino 5 e deve ser conectado ao Pino *Rx* do *Arduino*.
- *Ground*: O aterramento. Correspondente ao Pino 8 e deve ser conectado ao Pino *Ground* do *Arduino*.
- *Vcc 5v*: Fonte de alimentação de corrente contínua com tensão de 5 volts. Correspondente ao Pino 9 e deve ser conectado ao Pino *Vin* do *Arduino*.

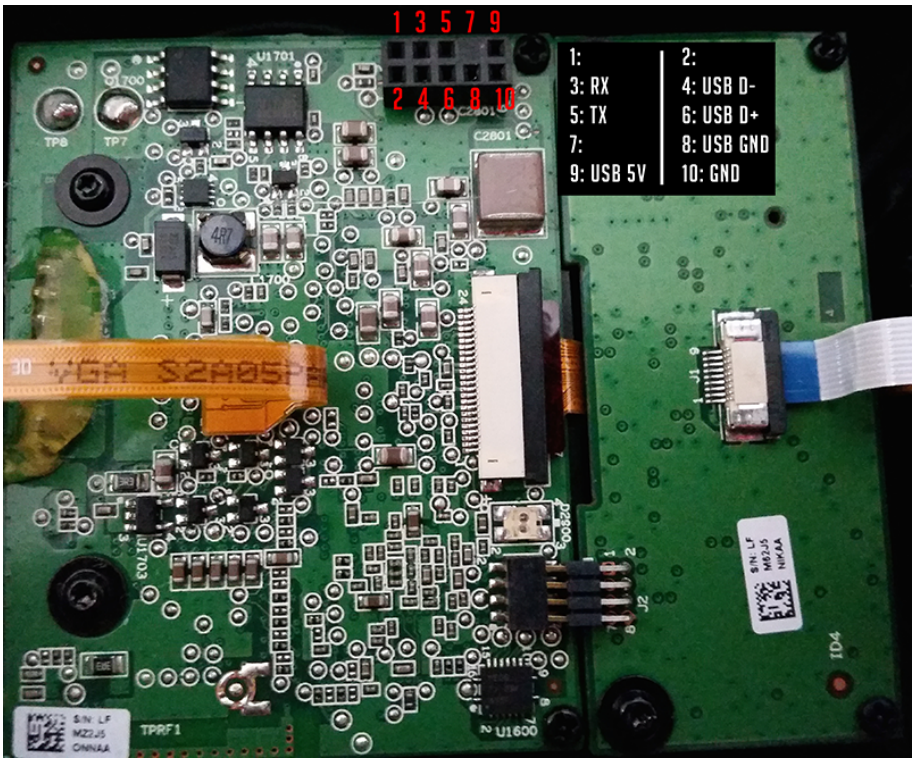

Figura 5.5: Conexão USB no AR.Drone 2.0

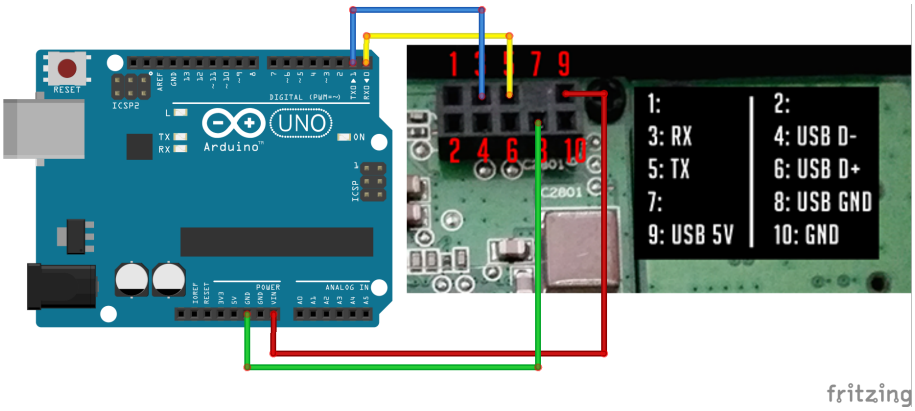

Figura 5.6: Esquema de Conexão do *Arduino* ao AR.Drone 2.0

Vale lembrar que o *Rx* do *Arduino* deve ser conectado ao *Tx* do drone e o *Tx* do *Arduino* ao *Rx* do drone uma vez que um pino envia e o outro deve receber. Ainda, o pino *Vcc* do drone deve ser conectado ao *Vin* do *Arduino* de modo que o *Vin* é a fonte de alimentação de corrente contínua do *Arduino*.

Após realizar todos os passos de preparação do ambiente de modo que o AR.Drone 2.0

possa se comunicar com o *Arduino* o código referente ao Algoritmo 5.2 de teste *Hello World* foi executado no *Arduino*. Após, foi lido pelo drone por meio de uma sequência de códigos em *JavaScript* utilizando a biblioteca *node-serialport* do framework *Node.js*. No Algoritmo 5.3 pode ser conferido o código responsável pela leitura da porta serial, */dev/ttyO3*, pelo AR.Drone 2.0. O resultado da leitura pelo drone por ser conferido na Figura 5.7.

Figura 5.7: Leitura de String pelo drone

Até este ponto mostra-se possível a comunicação do *Arduino* com o AR.Drone 2.0 uma vez que o *Arduino*escreve na porta serial de saída (*Tx*) a string *Hello World*. Assim o AR.Drone 2.0, mediante o Algoritmo 5.2 de leitura da porta serial recebe a string pela entrada serial (*Rx*), imprimindo-a no terminal de comando do drone.

```
void setup(){
  Serial.begin(9600);
}

void loop(){
  Serial.println("Hello World");
  delay(100);
}
```

Algoritmo 5.2: Código Hello World Arduino

```
var serialport = require('node-serialport')

var sp = new serialport.SerialPort("/dev/ttyO3", {
  parser: serialport.parsers.raw,
  baud: 9600
```

```
})

sp.on('data', function(chunk) {
  console.log(chunk.toString())
})
```

Algoritmo 5.3: Código para Leitura da Porta Serial

O próximo passo é incluir sensores no *Arduino* de modo que o drone possa receber os dados sensoriais enviados. Para exemplificar a inclusão de dados sensoriais foi incluído um sensor de temperatura, *LM35*, ao *Arduino*, conforme pode ser verificado o esquema de conexões na Figura 5.8. O código responsável pela leitura do sensor pelo *Arduino* é mostrado no Algoritmo 5.4. A variável *correctFactor* é o fator de correção calculado para a utilização do sensor de temperatura pelo drone uma vez em que há diferenças de voltagem na fonte de alimentação do drone e da porta *USB* de um computador convencional. Para calcular esse fator foi coletado uma amostra de 30 valores, isto é, resultados de temperatura do sensor utilizando como fonte de alimentação do *Arduino* a porta *USB* de um computador. Em seguida foi coletada a mesma quantidade de amostra com fonte de alimentação sendo o drone. Logo o fator de correção é a divisão da média aritmética dos resultados coletados pelo drone pela média coletada pelo computador. Na Figura 5.9 pode-se verificar a leitura do sensor de temperatura, em graus, conectado ao *Arduino*, pelo drone, executando o Algoritmo 5.3 pelo drone através do comando */data/video/usb0/./node serial3.js*.

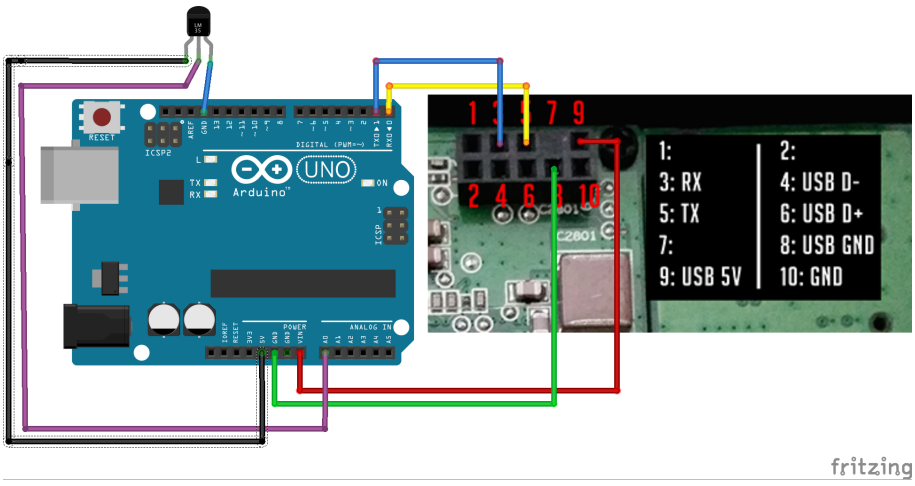

Figura 5.8: Esquema de Conexão do sensor de Temperatura ao *Arduino*

```
int analTempPin = A0;
float temp = 0;
float correctFactor = 0.768744354;
```

```
5  void setup(){
     Serial.begin(9600);
   }

   void loop(){
10   temp = analogRead(analTempPin);
     temp = ((5.0 * correcFactor) * temp * 100.0)/1024.0;
     Serial.println(temp);
     delay(200);
   }
```

Algoritmo 5.4: Código para Leitura do Sensor de Temperatura

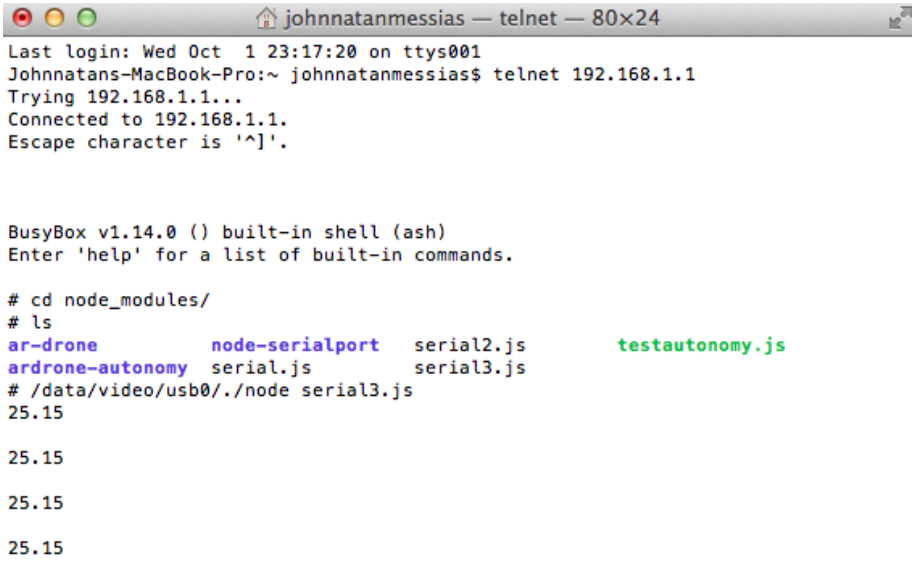

Figura 5.9: Leitura do Sensor de Temperatura pelo drone

5.5 Cenário 5

Tendo em vista os cenários anteriores, tornou-se necessária a utilização de sensores para uma aplicação real envolvendo o *Arduino* e o drone. Por esse motivo, foi implementado um sistema em *Arduino*, utilizando um sensor infravermelho, de modo a enviar comandos ao drone através de um controle remoto. A partir de um simples controle remoto, com os padrões de comunicação devidamente configurados, será possível enviar comandos de missão pré-programados ao drone bem como criar as próprias missões via controle remoto.

Para tornar viável esse sistema foi preciso criar o esquema de conexão do sensor de infravermelho como mostrado na Figura 5.10, onde há a inclusão de um *speaker* que emite um sinal sonoro tada vez em que recebe um sinal emitido pelo controle remoto e lido pelo sensor,

e ainda implementar a leitura do sensor bem como a interpretação dos dados recebidos por ele via *Arduino*. Após a interpretação dos dados o resultado é escrito na porta serial para ser lido pelo drone.

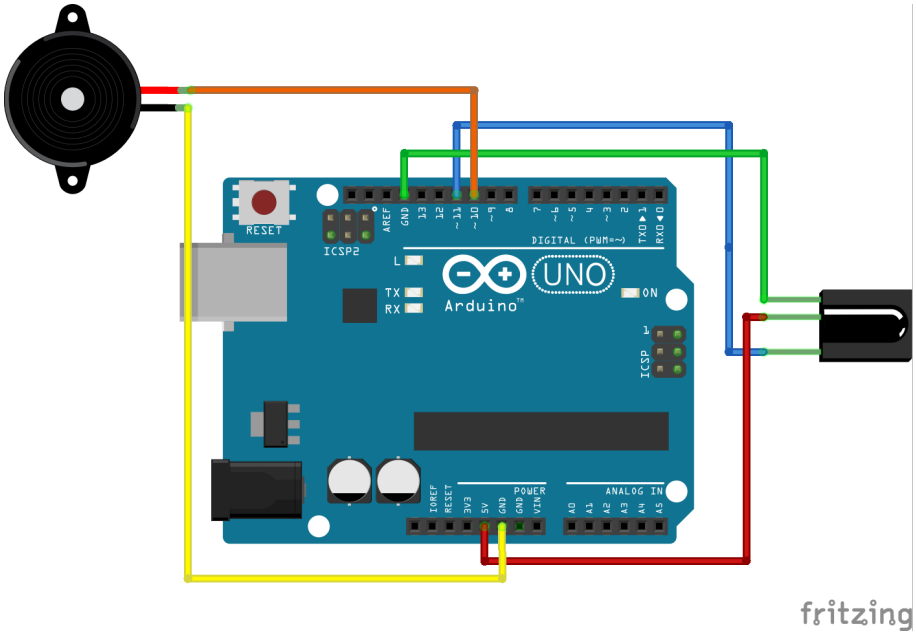

Figura 5.10: Esquema de Conexão do sensor de Infravermelho ao *Arduino*

De forma simplificada, os passos realizados nesse cenários são:

1. Enviar um sinal de controle para o sensor infravermelho.
2. Ler o dado sensorial pelo *Arduino*.
3. Interpretar o dado lido, traduzindo-o.
4. Escrever o dado traduzido na porta serial.
5. Ao receber o dado na porta serial o drone toma uma determinada ação.

Para a realização dos passos 1, 2 e 3 foi necessário implementar o Algoritmo 5.5 em *Arduino*. Nesse algoritmo foi utilizada a biblioteca *IRremote* Shirriff (2014). Após esses passos foi implementado o Algoritmo 5.6 responsável por ler os dados coletados pelo *Arduino* e prover as atitudes desejadas pelo drone. Os protocolos de comunicação adotados para gerenciar cada sinal do controle remoto foi aplicado ao controle remoto disponibilizado no *Kit Arduino* como pode ser observado na Figura 5.11. Cada botão do controle remoto foi traduzido de forma a obter uma comunicação com o drone, ou seja, cada botão realiza uma determinada ação, criando missões, no AR.Drone 2.0 como podemos verificar na Tabela 5.1. Cada missão criada

Tabela 5.1: Funções Associadas ao Controle Remoto

Botões	**Funções**	**Botões**	**Funções**
Power	TakeOff/Landing	Mode	-
Mute	Desconectar/Sair	Play/Pause	Subir/Descer
Voltar	Mover para Trás	Avançar	Mover para a Frente
Eq	Executar missão	Menos(-)	Mover Yaw para Esquerda
Mais (+)	Mover Yaw para Direita	0	Teste de comunicação
Loop	Decrementa Angle em 15	U/SD	Incrementa Angle em 15
1	Executa Missão Quadrática	2	Executa Missão Linear
3	Executa Missão Triangular	4	Executa Missão Circular
5	Mover para a esquerda	6	Move para a direita
7	Decrementa Value em 1	8	Incrementa Value em 1
9	Posiciona o drone no ponto de origem	-	-

será executada pela biblioteca *Ardrone-Autonomy* acionando o botão *EQ* do controle remoto. Assim o usuário terá a possibilidade de criar missões via código *JavaScript* ou simplesmente pelo próprio controle remoto.

O cenário 5, portanto, implementa os cenários 2, 3 e 4, diferenciando-se somente na abordagem de inclusão sensorial onde aqui utiliza-se um sensor de infravermelho para a comunicação via controle remoto.

```
#include <IRremote.h>

int receiverPin = 11;
int speakerPin = 10;
String value;
IRrecv irrecv(receiverPin);
decode_results results;

void setup(){
  Serial.begin(9600);
  pinMode(speakerPin, OUTPUT);
  irrecv.enableIRIn();
```

Figura 5.11: Controle Remoto Utilizado

```
}

void loop(){
  if(irrecv.decode(&results)){
    translateIR();
    irrecv.resume();
    digitalWrite(speakerPin, HIGH);
    delay(50);

  }else{
    digitalWrite(speakerPin, LOW);
  }
}

void translateIR(){
   switch(results.value){
     case 0xFFA25D:
       value = "POWER";
       break;
     case 0xFF629D:
       value = "MODE";
       break;
```

```
35        case 0xFFE21D:
            value = "MUTE";
            break;
          case 0xFF22DD:
            value = "PLAYPAUSE";
40          break;
          case 0xFF02FD:
            value = "BACK";
            break;
          case 0xFFC23D:
45          value = "FORWARD";
            break;
          case 0xFFE01F:
            value = "EQ";
            break;
50        case 0xFFA857:
            value = "-";
            break;
          case 0xFF906F:
            value = "+";
55          break;
          case 0xFF6897:
            value = "0";
            break;
          case 0xFF9867:
60          value = "LOOP";
            break;
          case 0xFFB04F:
            value = "USD";
            break;
65        case 0xFF30CF:
            value = "1";
            break;
          case 0xFF18E7:
            value = "2";
70          break;
          case 0xFF7A85:
            value = "3";
            break;
          case 0xFF10EF:
75          value = "4";
            break;
          case 0xFF38C7:
            value = "5";
            break;
80        case 0xFF5AA5:
            value = "6";
```

```
        break;
      case 0xFF42BD:
        value = "7";
        break;
      case 0xFF4AB5:
        value = "8";
        break;
      case 0xFF52AD:
        value = "9";
        break;
    case 0xFFFFFFFF:
      break;
    default:
        value = "ERROR";
   }
    Serial.println(value);
    delay(500);
}
```

Algoritmo 5.5: Código para Leitura do Sensor Infravermelho

```
var serialport = require('serialport')
var autonomy = require('ardrone-autonomy');
var mission = autonomy.createMission();
var content;
var altitude = 1;
var isInTheAir = false;
var value = 1;
var angle = 90;
var timeHover = 1000;
var sp = new serialport.SerialPort("/dev/ttyO3", {
  baudrate: 9600,
  parser: serialport.parsers.readline('\r\n')
});
console.log("Running");
sp.on('data', function(data) {
  content = data;
  console.log(content);
  execute();
})

function execute(){
    switch(content){
      case "MODE":
      break;
    case "POWER":
      if(!isInTheAir){
        console.log("TakeOff");
```

```
        mission.takeoff()
          .zero()
          .hover(timeHover)
          .altitude(value);
          isInTheAir = true;
      }else{
        console.log("Landing");
        mission.land();
        isInTheAir = false;
      }
      break;
    case "MUTE":
      console.log("Exit Program");
      process.exit(0);
      break;
    case "1":
      console.log("Execute Mission Square");
      createMissionSquare();
      break;
    case "2":
      console.log("Execute Mission Line");
      createMissionLine();
      break;
    case "3":
      console.log("Execute Mission Triangle");
      createMissionTriangle();
      break;
    case "4":
      console.log("Mission Circle");
      createMissionCircle();
      break;
    case "5":
      console.log("Move Left: " + value + "m");
      mission.left(value);
      break;
    case "6":
      console.log("Move Right: " + value + "m");
      mission.right(value);
      break;
    case "7":
      console.log("Value: " + (--value) + "m");
      break;
    case "8":
      console.log("Value: " + (++value) + "m");
      break;
    case "9":
      console.log("Going to the start point");
```

```
75          mission.go({x:0, y:0});
            break;
        case "0":
            console.log("Is Working Yeah");
            console.log("Value: " + value + "m");
80          console.log("Angle: " + angle + "Â°");
            break;
        case "PLAYPAUSE":
            console.log("Altitude: " + value + "m");
            mission.altitude(value);
85          break;
        case "+":
            console.log("Move Yaw right: " + angle + "Â°");
            mission.cw(angle);
            break;
90      case "-":
            console.log("Move Yaw left: " + angle + "Â°");
            mission.ccw(angle);
            break;
        case "BACK":
95          console.log("Move Back: " + value + "m");
            mission.backward(value);
            break;
        case "FORWARD":
            console.log("Move Front: " + value + "m");
100         mission.forward(value);
            break;
        case "LOOP":
            angle -= 15;
            console.log("Angle: " + angle + "Â°");
105         break;
        case "USD":
            angle += 15;
            console.log("Angle: " + angle + "Â°");
            break;
110     case "EQ":
            console.log("Run Mission");
            executeMission();
            break;
    }
115 }

    function createMissionSquare(){
      mission.takeoff()
       .zero()
120    .hover(500)
       .altitude(1)
```

```
    .forward(2)
    .cw(90)
    .forward(2)
    .cw(90)
    .forward(2)
    .cw(90)
    .go({x:0, y:0})
    .cw(90)
    .hover(1000)
    .land();
}

function createMissionTriangle(){
  mission.takeoff()
    .zero()
    .hover(500)
    .altitude(1)
    .ccw(30)
    .forward(2)
    .cw(120)
    .forward(2)
    .cw(120)
    .go({x:0, y:0})
    .hover(500)
    .land();
}

function createMissionLine(){
  mission.takeoff()
    .zero()
    .hover(500)
    .altitude(1)
    .forward(1)
    .go({x:0, y:0})
    .hover(1000)
    .land();
}

function createMissionCircle(){
  var angleRad = 0;
  var two_pi = Math.PI * 2;
  var radian = 2;
  mission.takeoff()
    .zero()
    .hover(500)
    .altitude(1)
    mission.go({x:radian, y:0});
```

```
  while(angleRad < two_pi){
    droneX = radian * Math.sin(angleRad);
    droneY = radian * Math.cos(angleRad);
    angleRad += 0.2;
    mission.go({x:droneX, y:droneY});
  }
  mission.go({x:0, y:0})
    .hover(1000)
    .land();
}

function degreeToRad(degree){
  return (degree * Math.PI) / 180;
}

function radToDegree(rad){
  return (180 * rad) / Math.PI;
}

function executeMission(){
     mission.run(function (err, result){
  if(err){
    console.trace("Error: %s",err.message);
    mission.client().stop();
    mission.client().land();
  }else{
    console.log("Mission success!");
    process.exit(0);
  }
});
}
```

Algoritmo 5.6: Código de Controle do Drone

Capítulo 6

Experimentos

Após a finalização das implementações dos cenários descritas no capítulo 5 será necessária a realização dos testes de voo com o drone utilizando o cenário 5 descrito no capítulo 5.5.

Para isso iremos considerar os seguintes experimentos:

1. Missão definida pelo usuário, sendo, portanto, criada mediante comandos pelo controle remoto.
2. Missão anteriormente definida, pelo desenvolvedor, via código *JavaScript*:
    - Movimento Linear: Planejamento de voo de modo que o drone vá para a frente e volte ao ponto de origem.
    - Movimento Quadrático: Planejamento de voo quadrático realizado pelo drone, retornando à origem ao término da missão.
    - Movimento Triangular: Planejamento de voo triangular realizado pelo drone, retornando à origem ao término da missão.
    - Movimento Circular: Planejamento de voo circular realizado pelo drone, retornando à origem ao término da missão.

Ambos os algoritmos responsáveis pelo planejamento dessas missões estão descritos no Algoritmo 5.6.

6.1 Movimento Definido pelo Usuário

Nesse tipo de movimento, o usuário necessita criar a missão de voo utilizando o controle remoto. Para isso será necessário seguir os comandos do controle remoto definidos na Tabela 5.1. Suponha que o usuário necessite realizar a seguinte missão:

1. Iniciar o processo de decolagem

2. Configurar a variável *value* em 2 metros
3. Posicionar a altura do drone em 2 metros
4. Avançar 2 metros para a frente
5. Configurar o ângulo de rotação para 45 graus
6. Aplicar uma rotação para a direita usando o ângulo configurado anteriormente
7. Configurar a altura do drone em 1 metro
8. Avançar para a esquerda 1 metros
9. Ir para o ponto de origem
10. Iniciar o processo de decolagem

Para criar essa missão, seguindo os comandos da Tabela 5.1, será necessário o usuário pressionar os botões do controle remoto na seguinte sequência: Botões *POWER, 8, PLAY/-PAUSE, AVANÇAR, LOOP, LOOP, LOOP, MAIS(+), 7, 5, 9, POWER, EQ*. Seguindo esses comandos o drone executará a missão descrita na Figura 6.1.

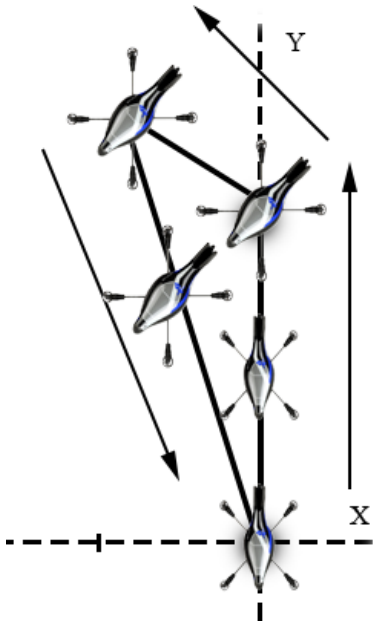

Figura 6.1: Descrição do Movimento Definido pelo Usuário

6.2 Movimento Linear

O Algoritmo 6.1 descreve a missão responsável por realizar o movimento linear com o drone. Trata-se de uma implementação em *JavaScript* onde o drone realiza a decolagem mediante o método *takeOff()* e em seguida utiliza o *zero()* para determinar o local de decolagem como o ponto de origem do drone. Em seguida, utiliza-se o *hover(500)*, voo estacionário, nesse caso por 0.5 segundos e a altitude em 1 metro através do método *altitude(1)*. Após o ponto de origem ser definido e o drone estar em modo de voo estacionário o próximo passo é utilizar o método *forward(1)* para obter o movimento para frente do drone em 1 metro. Para finalizar, o método *go(x:0, y:0)* é utilizado para que o drone volte ao local de origem, entre em modo de voo estacionário por 1 segundo e pouse. A Figura 6.2 descreve esse tipo de movimento bem como o ponto de origem e destino.

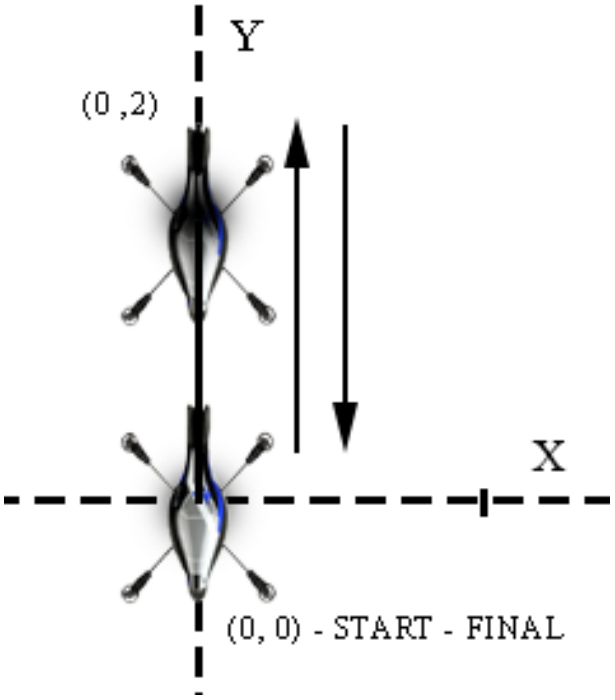

Figura 6.2: Descrição do Movimento Linear

```
function createMissionLine(){
  mission.takeoff()
    .zero()
    .hover(500)
    .altitude(1)
    .forward(1)
    .go({x:0, y:0})
    .hover(1000)
    .land();
```

```
10 }
```

Algoritmo 6.1: Algoritmo do Movimento Linear

6.3 Movimento Quadrático

Nesse movimento, realizado através da execução do Algoritmo 6.2, os passos iniciais são semelhantes ao anterior até o ponto em que é definida a altitude. Após esse passo é definido o movimento quadrático onde a missão será o drone ir para a frente por 2 metros, virar com angulo de 90 graus para a direita, andar mais 2 metros para a frente, virar para a direita em 90 graus, ir novamente para a frente por 2 metros virando ao final para a direita com angulo de 90 graus. Para finalizar usa-se o comando *go(x:0, y:0)* para que o drone volte ao local de origem e em seguida vira-se o drone em 90 graus para a direita seguido de um modo de voo estacionário e finaliza-se com o pouso. A Figura 6.3 descreve esse tipo de movimento bem como o ponto de origem e destino.

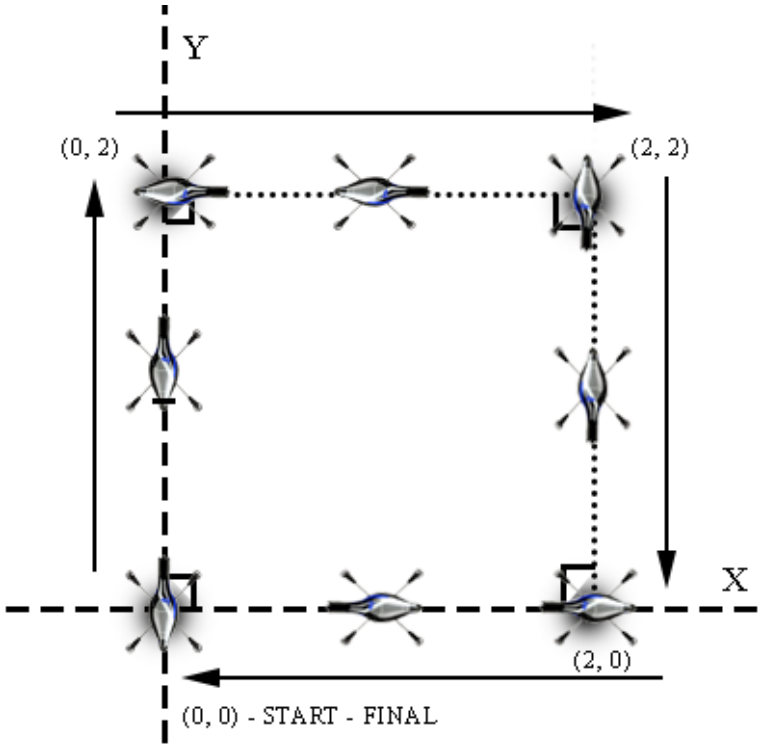

Figura 6.3: Descrição do Movimento Quadrático

```
function createMissionSquare(){
  mission.takeoff()
   .zero()
```

```
    .hover(500)
    .altitude(1)
    .forward(2)
    .cw(90)
    .forward(2)
    .cw(90)
    .forward(2)
    .cw(90)
    .go({x:0, y:0})
    .cw(90)
    .hover(1000)
    .land();
}
```

Algoritmo 6.2: Algoritmo do Movimento Quadrático

6.4 Movimento Triangular

Seguindo o Algoritmo 6.3, o drone percorrerá um caminho definido por um triângulo equilátero de lados 2 metros, isto é, onde os lados se interceptam formando um ângulo de 60 graus. Para isso, a orientação do drone foi elaborada de acordo com a Figura 6.4. Assim, ao decolar e deixar o drone em modo estacionário o drone sofre uma rotação *yaw*, sobre o eixo Z, para a esquerda em 30 graus, em seguida, o drone desloca-se para a frente por 2 mertros. Nesse ponto, como pode ser verificado na Figura 6.4 torna-se necessário também uma nova rotação mas com ângulo de 120 graus e para a direita, após o drone desloca-se mais 2 metros para a frente, aplicando mais uma rotação de 120 graus para a direita. Após esses passos o drone volta ao ponto de origem e pousa.

```
function createMissionTriangle(){
  mission.takeoff()
    .zero()
    .hover(500)
    .altitude(1)
    .ccw(30)
    .forward(2)
    .cw(120)
    .forward(2)
    .cw(120)
    .go({x:0, y:0})
    .hover(500)
    .land();
}
```

Algoritmo 6.3: Algoritmo do Movimento Triangular

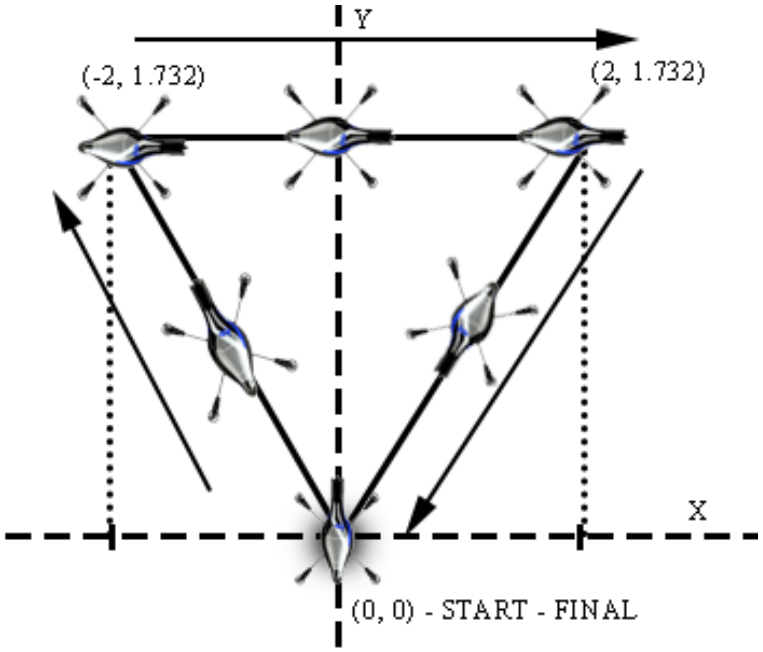

Figura 6.4: Descrição do Movimento Triangular

6.5 Movimento Circular

Para realizar com êxito o movimento circular, implementado no Algoritmo 6.4, será necessário definir um raio para a circunferência, nesse caso foi escolhido 2 metros. Após realizar a decolagem e os procedimentos de estabilidade do voo o drone será deslocado para a posição *X: 2, Y:0*. Vale lembrar que esse movimento circular será único, isto é, o drone fará o movimento circular somente uma vez, por esse motivo no loop responsável pelas coordenadas no espaço da circunferência definimos que sejam geradas as coordenadas até que o ângulo (em radianos) seja menor do que $2 * \pi$, ou seja, $2 * 3.14$, completando toda a circunferência. Nesse loop, calcula-se a posição em X pela fórmula $droneX = raio * \sin angulo$ e o Y definida por $droneY = raio * \cos angulo$. O ângulo será incrementado em 0.2 radianos, definindo a distância a cada passo da iteração que o drone se deslocará pelo método *go(x:droneX, y:droneY)*. Após esses passos o drone volta para a origem *go(X:0, Y:0)* e inicia-se o procedimento de pouso, concluindo a missão. A Figura 6.5 descreve esse tipo de movimento bem como o ponto de origem e destino.

```
function createMissionCircle(){
  var angleRad = 0;
  var two_pi = Math.PI * 2;
  var radian = 2;
  mission.takeoff()
    .zero()
```

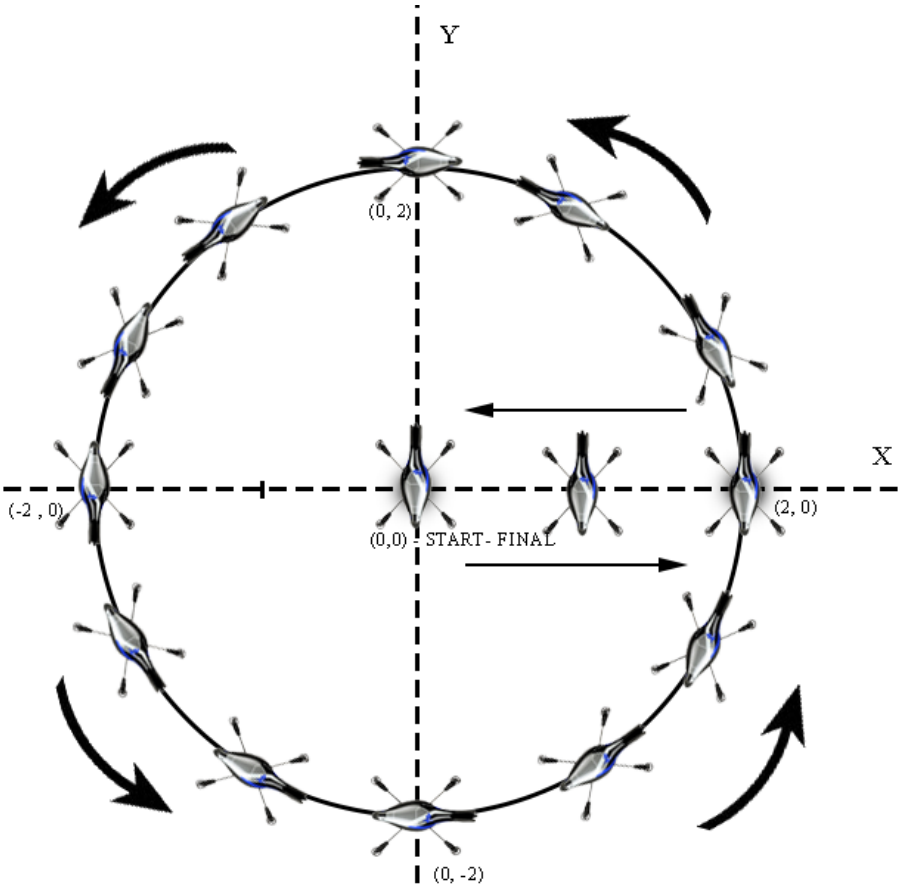

Figura 6.5: Descrição do Movimento Circular

```
    .hover(500)
    .altitude(1)
    mission.go({x:radian, y:0});
  while(angleRad < two_pi){
    droneX = radian * Math.sin(angleRad);
    droneY = radian * Math.cos(angleRad);
    angleRad += 0.2;
    mission.go({x:droneX, y:droneY});
  }
  mission.go({x:0, y:0})
    .hover(1000)
    .land();
}
```

Algoritmo 6.4: Algoritmo do Movimento Circular

Capítulo 7

Conclusões

Neste trabalho foi abordado os passos necessários para tornar voos autônomos possíveis utilizando o AR.Drone 2.0. Para isso, foi necessário introduzir os conceitos dos protocolos de comunição do drone bem como salientar sobre a dinâmica de voo do mesmo. Ambos são passos importantes para o desenvolvimento de programas de comunição com o drone fazendo com que ele possa voar.

O objetivo desse projeto é saber se é possível tornar voos autônomos viáveis no AR.Drone 2.0 bem como a compreensão sobre o seu funcionamento. Para que isso, foram considerados 5 cenários. O primeiro teve como objetivo implementar uma aplicação em *Java*, utilizando a *API JavaDrone*. Entretanto, após a realização desse experimento foi verificado que abordar voos autônomos com a linguagem *Java* seria mais trabalhoso uma vez que há ferramentas mais adequadas para esse tipo de tarefa. Logo, tornou-se necessário buscar uma nova solução, uma linguagem ou framework que proporcionasse maior abstração para a realização de voos autônomos. A nova solução é descrita pelo cenário 2. Nesse cenário, houve a necessidade de descartar o uso da Linguagem *Java* descrita pelo cenário 1 e agora utilizar o framework *Node.js*, trata-se de um framework onde o desenvolvimento é realizado em *JavaScript*. Devido a utilização desse framework foi possível utilizar bibliotecas como *AR-Drone, Ardrone-Autonomy*, para que fosse viável criar missões de voos autônomos, e *Serial-Port* e a comunicação direta do drone com o *Arduino*. Até o momento todos os códigos eram executados por computadores externos conectados via *WiFi* ao drone.

Para proporcionar maior nível de autonomia foi preciso verificar a possibilidade de o próprio drone executar os algoritmos desenvolvidos utilizando o seu próprio processador e memória principal. Através do cenário 3 essa etapa foi concluída, a partir de agora todos os algoritmos serão executados pelo próprio drone. No cenário 4, foi preciso integrar o *Arduino* ao drone pois para a realização de voos autônomos torna-se necessário incluir novos sensores ao drone. A integração de sensores é realizada mediante o *Arduino*. Portanto, todos os novos sensores a serem integrados ao drone serão conectados ao *Arduino* e os dados sensoriais serão lidos pelo drone mediante uma porta serial. Essa etapa é uma das principais contribuições desse projeto

uma vez que, agora, podemos integrar quaisquer sensores para *Arduino* ao drone. No cenário 5, houve a implementação das missões de voos autonômos com a utilização de um controle remoto e um sensor de infravermelho conectado ao Arduino.

Mediante os experimentos e estudos apresentados tornou-se possível atingir o objetivo proposto, tornando viável a aplicação de voos autônomos no drone. Como resultado, para a realização de voos autônomos foi elaborado um framework onde o usuário poderá criar missões de voos autônomos, de forma simplificada, para o drone executa-las. Esses comandos são enviados ao drone pelo usuário devido a utilização de um controle remoto. Esse controle remoto envia dados a um sensor conectado ao *Arduino* que processa os dados e em seguida é lido e interpretado pelo drone.

Vale lembrar que o AR.Drone, tanto a versão 1.0 quanto a 2.0, é um drone de baixo custo e possui *SDK* e os protocolos de comunicação disponíveis no site do fabricante o que favoreceu para o constante uso desse drone em estudos científicos. Por outro lado, para uso profissional ele não é indicado uma vez que é bastante instável se comparado a outros modelos disponíveis no mercado.

Como trabalho futuro podemos considerar a utilização das câmeras de forma a proporcionar um processamento de imagens e extração de características do ambiente. Existem técnicas relacionadas ao uso das câmeras como, por exemplo, o $SLAM$[1] que consiste em criar ou atualizar um mapa de um ambiente desconhecido enquanto, simultaneamente, realiza a localização do drone no ambiente. Ainda, também como trabalho interessante seria fazer com que dois ou mais drones se comunicassem de modo a compartilhar as missões delegadas a algum deles.

[1] Simultaneous Localization and Mapping

Apêndice

Todos os códigos implementados nesse projeto bem como as bibliotecas envolvidas podem ser baixados através desse link: `http://www.decom.ufop.br/imobilis/johnnatan/thesis/`.

Referências Bibliográficas

Aaron, C. (2014). Why you should learn node.js today. Acessado em 12 de outubro de 2014.

Ar.Drone, P. (2012). Ardrone open api platform. https://projects.ardrone.org/. Acessado em 20 de setembro de 2014.

Ardrone-Autonomy (2014). Ardrone autonomy api. https://github.com/eschnou/ardrone-autonomy. Acessado em 20 de setembro de 2014.

Berezny, N.; Greef, L.; Jensen, B.; Sheely, K.; Sok, M.; Lingenbrink, D. e Dodds, Z. (2012). Accessible aerial autonomy. *Technologies for Practical Robot Applications (TePRA), 2012 IEEE International Conference on*, pp. 53 – 58.

Bresciani, T. (2008). *Modelling, Identification and Control of a Quadrotor Helicopter*. PhD thesis, Department of Automatic Control, Lund University.

Bristeau, P.; Callou, F.; Vissière, D. e Petit, N. (2011). The navigation and control technology inside the ar.drone micro uav. *18th IFAC World Congress*, pp. 1477–1484.

Enomoto, J. L. F. (2010). Controle de inclinação utilizando lógica fuzzy. Technical report, Departamento de Engenharia Elétrica, UFPR.

JavaDrone (2014). Ar.drone java api. https://projects.ardrone.org/. Acessado em 10 de maio de 2013.

Macharet, D. G. (2009). *Localização e Mapeamento em Terrenos Irregulares Utilizando Robôs Móveis*. PhD thesis, Universidade Federal de Minas Gerais.

Morar, I. e Nascu, I. (2012). Model simplification of an unmanned aerial vehicle. *Automation Quality and Testing Robotics (AQTR), 2012 IEEE International Conference on*, pp. 591–596.

Postel, J. e Reynolds, J. (1983). Telnet protocol specification. internet std 8, rfc 854. Acessado em 15 de outubro de 2014.

Serial-port (2014). Node-serialport. https://github.com/voodootikigod/node-serialport. Acessado em 10 de outubro de 2014.

Shirriff (2014). Irremote library for the arduino. https://github.com/shirriff/Arduino-IRremote. Acessado em 01 de novembro de 2014.