UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DESENVOLVIMENTO DE UM DRONE DE BAIXO CUSTO

TRABALHO DE GRADUAÇÃO

Matheus Garay Trindade

Santa Maria, RS, Brasil

2016

DESENVOLVIMENTO DE UM DRONE DE BAIXO CUSTO

Matheus Garay Trindade

Trabalho de Graduação apresentado ao Bacharelado em Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Prof. Dr. Osmar Marchi dos Santos

Santa Maria, RS, Brasil

2016

Universidade Federal de Santa Maria
Centro de Tecnologia
Bacharelado em Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

DESENVOLVIMENTO DE UM DRONE DE BAIXO CUSTO

elaborado por
Matheus Garay Trindade

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Osmar Marchi dos Santos, Dr.
(Presidente/Orientador)

Andrei Piccinini Legg, Dr. (UFSM)

Daniel Fernando Tello Gamarra, Dr. (UFSM)

Santa Maria, 16 de Dezembro de 2016.

Aos meus pais, Alcino Francisco Magno Trindade e Rosângela Garay Trindade

“Yes, of course duct tape works in a near-vacuum. Duct tape works anywhere. Duct tape is magic and should be worshiped”

— ANDY WEIR, THE MARTIAN

RESUMO

Trabalho de Graduação
Bacharelado em Ciência da Computação
Universidade Federal de Santa Maria

DESENVOLVIMENTO DE UM DRONE DE BAIXO CUSTO
AUTOR: MATHEUS GARAY TRINDADE
ORIENTADOR: OSMAR MARCHI DOS SANTOS
Local da Defesa e Data: Santa Maria, 16 de Dezembro de 2016.

O objetivo deste trabalho consiste no desenvolvimento de um veículo aéreo não-tripulado (UAV - *Unmanned Aerial Vehicle*), chamado de drone neste trabalho. Do ponto de vista físico (hardware) do drone, um dos principais focos neste desenvolvimento é a obtenção de um drone de baixo de custo. Para isso, foram utilizados componentes e materiais de rápida aquisição e prototipação. Por exemplo, placas de desenvolvimento, assim como o uso de modelagem e impressão de componentes físicos através de impressora 3D. Além do desenvolvimento da parte física do drone, o trabalho também apresenta a implementação de uma interface programação para comunicação com a placa de controle utilizada na navegação. Isso possibilita mecanismos básicos para navegação autonoma, necessários para integrar outras aplicações e tecnologias ao drone, e.g., bibliotecas de processamento de imagens e de linguagem natural.

Palavras-chave: Drones. Robótica. VANTs.

ABSTRACT

Undergraduate Final Work
Bachelor in Computer Science
Federal University of Santa Maria

DEVELOPMENT OF A LOW-COST DRONE

AUTHOR: MATHEUS GARAY TRINDADE
ADVISOR: OSMAR MARCHI DOS SANTOS
Defense Place and Date: Santa Maria, December 16th, 2016.

The goal of this work consists in the development of a Unmanned Aerial Vehicle (UAV), reffered as drone in text. From the physical point of view (hardware) of the drone, one of the main focuses of this work is obtaining a low-cost drone. In order to do so, components and materials of rapid acquisition and prototyping were used. For example, development boards were used, as well as modelling and printing of physical components throught a 3D printer. Apart from the development of the physical part of the drone, this work also presents an implementation of a programming interface to communicate with the board used for navigation. This provides basic mechanisms for autonomous navigation, which are needed to allow for the integration of other applications and technologies to the drone, e.g., image processing and natural language proccessing libraries.

Keywords: Drones. Robotics, UAVs.

LISTA DE FIGURAS

Figura 2.1 – Drone sem receptor de rádio com uma plataforma embarcada de baixo custo 15
Figura 2.2 – Motor A2212 brushless (de (BRUSHLESS MOTOR PICTURE, 2016)) 15
Figura 2.3 – ESC 20A (de (ELECTRONIC SPEED CONTROLLER PICTURE, 2016)) .. 16
Figura 2.4 – CRIUS v2.0 board (de (CRIUS V2.0 PICTURE, 2016)) 16
Figura 2.5 – Remote Controller and receiver (de (RADIO CONTROLLER AND RECEIVER PICTURE, 2016)) 17
Figura 2.6 – 2200 mAh Li-Po battery (de (LI-PO BATTERY PICTURE, 2016)) 17
Figura 3.1 – *Frame* comercial 25
Figura 3.2 – Primeiro projeto de *frame* 26
Figura 3.3 – Braço do primeiro projeto de *frame*. Medidas representadas em milímetros . 26
Figura 3.4 – Suporte do motor do primeiro projeto de *frame*. Medidas representadas em milímetros 27
Figura 3.5 – Peça central superior do primeiro projeto de *frame*. Medidas representadas em milímetros 28
Figura 3.6 – Peça central inferior do primeiro projeto de *frame*. Medidas representadas em milímetros 28
Figura 3.7 – Braço do segundo projeto de *frame*. Medidas apresentadas em milímetros... 29
Figura 3.8 – Mecanismo de pouso do segundo projeto de *frame*. 29
Figura 3.9 – Peças do centro do segundo projeto de *frame*. Peça da esquerda é a superior e, a da direita, a inferior. Dimensões representadas em milímetros. O *software* usado para essa visualização foi o Meshmixer (MESHMIXER OFFICIAL WEBSITE, 2016). 30
Figura 3.10 – Parte inferior da proteção de hélice do segundo projeto de *frame*. Medidas apresentadas em milímetros. 30
Figura 3.11 – Parte inferior da proteção de hélice segundo projeto de *frame*. Medidas apresentadas em milímetros. 31
Figura 3.12 – Segundo projeto de drone 31
Figura 3.13 – Adaptador para prender os motores no braço de alumínio. Medidas representadas em milímetros 32
Figura 3.14 – Modelo final do *frame*. 32
Figura 4.1 – Arquitetura do sistema usando um Laptop se comunicando com o Middleware via TCP 39
Figura 4.2 – Xbox One® Controller (from (XBOX ONE CONTROLLER PICTURE, 2016)) 40
Figura 4.3 – GUI do Controle de Xbox One® 41
Figura 4.4 – GUI com tabela de entrada de programa de *throttle* 42

LISTA DE TABELAS

Tabela 4.1 – MultiWii Network API 35
Tabela 4.2 – API MultiWii Manipulation 36
Tabela 4.3 – Correspondência entre Valores de Controle e Sticks 40
Tabela 5.1 – Custo de material por drone, desconsiderando taxa de importação 45

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
DJI	*Da-Jiang Innovations Science and Technology Co. Ltd.*
ESC	Controlador Eletrônico de Velocidade
GUI	Interface Gráfica de Usuário
IMU	Unidade de Medição Inercial
LED	*Light Emiting Diode*
LQR	*Linear–quadratic regulator*
MSP	*MultiWii Serial Protocol*
PID	Proporcional-Integral-Derivativa
SSH	*Secure Shell*
USB	*Universal Serial Bus*
COTS	*Components Off-the-Shelf*
GPIO	Pinos Gerais de Entrada e Saída
VANT	Veículo Aéreo Não-Tripulado
ABS	Acrilonitrila butadieno estireno

SUMÁRIO

1 INTRODUÇÃO 12
1.1 Objetivo geral 12
1.2 Objetivos específicos 13
1.3 Organização 13
2 REFERENCIAL TEÓRICO 14
2.1 Anatomia de um drone 14
2.1.1 Motores e Hélices 14
2.1.2 ESCs 15
2.1.3 Unidade Controladora 15
2.1.4 Receptor de Rádio e Rádio Controle 16
2.1.5 Bateria 17
2.2 *Softwares* de Voo 17
2.2.1 Nomenclatura de Voo 18
2.2.2 Multiwii 19
2.2.2.1 MSP 21
2.2.2.1.1 Preâmbulo e Direção 21
2.2.2.1.2 Tamanho 21
2.2.2.1.3 Opcode 22
2.2.2.1.4 Dados 22
2.2.2.1.5 Checksum 22
2.2.3 ArduPilot 22
2.2.4 API para AR.Drone 2.0 23
3 USO E DESENVOLVIMENTO DE FRAMES 24
3.1 Frame Comercial 24
3.2 Primeiro Projeto de Frame 25
3.3 Segundo Projeto de Frame 28
3.4 Melhorias do Segundo Projeto de Frame 31
4 DESENVOLVIMENTO DE SOFTWARE 33
4.1 Teste Inicial do MSP 33
4.2 Implementação de um Middleware 34
4.2.1 *MultiWii Network* 34
4.2.2 MultiWii Protocol 35
4.2.3 MultiWii *Manipulation* 36
4.3 Testes no Frame Comercial 37
4.4 Primeiro Projeto de Frame 37
4.5 Segundo Projeto de Frame 38
4.6 Melhorias no Segundo Frame: Piloto Automático 41
4.6.1 Primeira Implementação 41
4.6.2 Resolvendo Problemas da Primeira Implementação e Controle de Altitude 43
5 CUSTOS 45
6 CONCLUSÕES 47
REFERÊNCIAS 49
ANEXOS 52

1 INTRODUÇÃO

A ideia de Veículos Aéreos Não-Tripulados (VANTs) não é recente. Eles já vêm sendo usados por décadas nas mais diferentes aplicações e nos mais diversos ambientes. Um bom exemplo é mostrado em (JET PILOTLESS DRONES COLLECT DATA ON RADIOLOGICAL HAZARDS IN ATOMIC CLOUDS, 1953), que apresenta o uso de um jato não-tripulado para coletar dados de uma nuvem atômica resultante de um teste nuclear já em 1953. Pilotar um jato através de uma nuvem atômica usando um piloto vivo não é uma tarefa que pode ser considerada, uma vez que seria proibitivamente perigoso para o piloto. Um experimento como esse demonstra o potencial de se usar VANTs.

Desde então, a tecnologia de drones evoluiu bastante. Eles estão cada vez mais presentes, podendo ser encontrados inclusive em lojas de brinquedos. Empresas dedicadas à produção de drones compõem hoje o mercado, focando principalmente no usuário sem um conhecimento tecnológico aprofundado. Da-Jiang Innovations Science and Technology Co. Ltd. (DJI) (DJI OFFICIAL WEBSITE, 2016), por exemplo, produz drones amplamente utilizados em filmagem e fotografia. Seus modelos mais recentes podem inclusive seguir uma pessoa (*tracking*) filmando-a. Além disso, cada vez mais surgem comunidades de *hobbystas*, como (ARDUPILOT OFFICIAL WEBSITE, 2016; MULTIWII OFFICIAL WEBSITE, 2016), onde membros compartilham suas experiências.

Para esse trabalho, optou-se por fazer drones do tipo quadcóptero. Esses drones são compostos por quatro motores dispostos em forma de 'X'. Esse design foi escolhido por ser o mais simples e mais recorrente. Ele também é uma alternativa barata, visto que existem multirotores com mais motores. No texto, onde se escreve drone sem se especificar que se fala do conceito geral, se faz referência ao produzido para o trabalho, um quadcóptero.

1.1 Objetivo geral

O objetivo deste trabalho consiste na modelagem e implementação de um veículo aéreo não-tripulado (VANT), i.e., drone, de baixo de custo, incluindo tanto sua infraestrutura física quanto de software. Para isso, deseja-se usar componentes fáceis de serem obtidos e materiais simples. Para facilitar o desenvolvimento, uma ferramenta que implementa algoritmos de controle será usada, adicionando-se sobre ela uma camada de software para prover uma interface de programação que possa ser facilmente integrada a outras tecnologias, e.g., bibliotecas de

processamento de imagens e de linguagem natural.

1.2 Objetivos específicos

O principal foco foi implementação de um drone autônomo, facilmente programável e de baixo custo. Para isso, necessitou-se: (i) entender o funcionamento dos componentes físicos que compõem um drone; (ii) projetar e construir um drone; (iii) testar os componentes; (iv) desenvolver uma ferramenta que possibilite o desenvolvimento de software para guiagem autônoma e integrável com tecnologias diversas; (v) fazer testes com o drone voando de forma autônoma. Ao fim, deseja-se possuir um drone bastante versátil, com uma interface simples de programação e que consiga executar os comandos que lhe são passados de forma correta.

1.3 Organização

Este trabalho está organizado da seguinte maneira. O Capítulo 2 apresenta a análise teórica necessária para o trabalho, descrevendo a estrutura básica de um drone e os *softwares* abertos de controle de voo mais relevantes. O Capítulo 3 descreve a evolução dos *frames* criados neste trabalho. O Capítulo 4 apresenta a evolução da infraestrutura de *software* desenvolvida em paralelo ao desenvolvimento de *frames*. O Capítulo 5 analisa o custo de produção da plataforma desenvolvida e a compara a alternativas existentes no mercado. O Capítulo 6, por fim, apresenta as discussões finais.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentadas as informações que foram necessárias para se dar início ao trabalho. Em suma, pode-se dividir o processo de criação de drones em dois grandes escopos: anatomia e algoritmos. Para facilitar o entendimento, estes assuntos serão abordados separadamente.

2.1 Anatomia de um drone

Para o começo da implementação do trabalho, foi feito um levantamento de quais são as partes necessárias para se montar um drone (MAHONY; KUMAR; CORKE, 2012). Essas informações foram buscadas em sites das ferramentas mais comumente utilizadas, como (MULTIWII OFFICIAL WEBSITE, 2016) e (ARDUPILOT OFFICIAL WEBSITE, 2016). Um drone simples é composto das seguintes partes:

- Corpo;
- Motores e hélices;
- Controladores Eletrônicos de Velocidade (ESCs);
- Unidade Controladora;
- Radio-controle e receptor de rádio;
- Bateria.

Podemos visualizar a maioria desses componentes na Figura 2.1.

2.1.1 Motores e Hélices

Motores e hélices são literalmente as entidades levantadoras de peso. Elas são responsáveis por gerar impulso no multirotor. Variando a potência imposta em cada motor, é possível projetar o multirotor para qualquer posição desejada. A forma que eles devem ser variados é melhor explicitada em 2.2.1. Um exemplo de motor é mostrado na Figura 2.2.

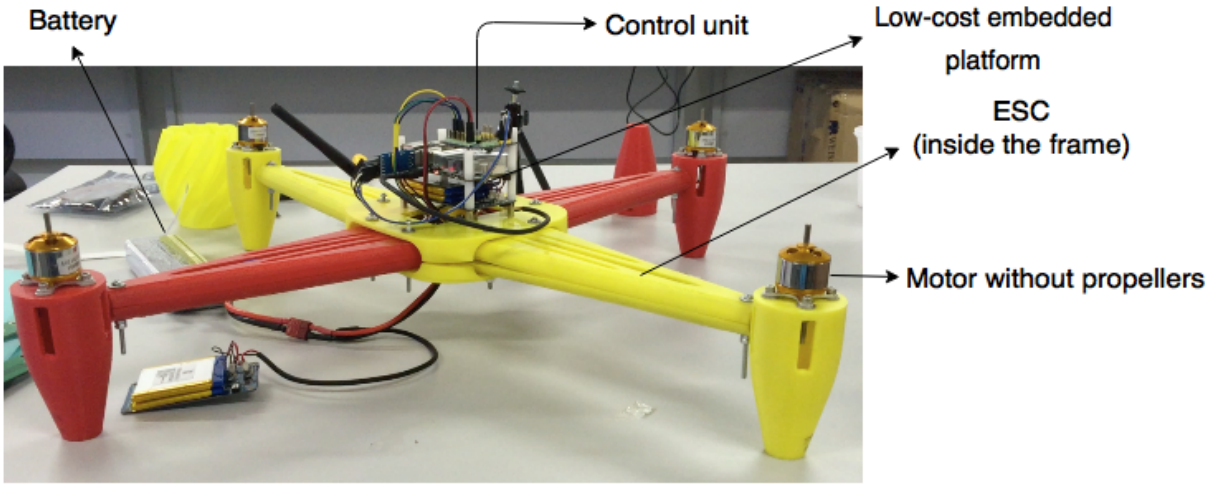

Figura 2.1 – Drone sem receptor de rádio com uma plataforma embarcada de baixo custo

Figura 2.2 – Motor A2212 brushless (de (BRUSHLESS MOTOR PICTURE, 2016))

2.1.2 ESCs

Controladores eletrônicos de velocidade (ESCs) são responsáveis por de fato interagirem com os motores para fazê-los girar na taxa desejada. Há um ESC por motor. Cada ESC recebe como entrada valor de aceleração e o converte para pulsos para motor. Um exemplo de ESC é mostrado na Figura 2.3.

2.1.3 Unidade Controladora

A principal tarefa da unidade controladora é receber valores de *pitch*, *roll*, *yaw* e *throttle* e converter para a quantia de potência que cada motor deve receber, repassando esses valores para os ESCs. Entretanto, essa não é uma tarefa tão simples. Multirotores são inerentemente instáveis e é trabalho da unidade controladora de compensar essas instabilidades. Normalmente,

Figura 2.3 – ESC 20A (de (ELECTRONIC SPEED CONTROLLER PICTURE, 2016))

a unidade controladora possui sua própria Unidade de Medição Inercial (IMU), uma unidade composta de diferentes sensores, como giroscópios, acelerômetros e barômetros. Esses valores são usados pelo *software* que executa na placa para corrigir problemas com a posição do multirotor. Na verdade, a unidade controladora recebe outros conjuntos de entrada que indicam como usar os dados de IMU. Esse processo é melhor detalhado em 2.2.1.

Uma unidade controladora é mostrada na Figura 2.4.

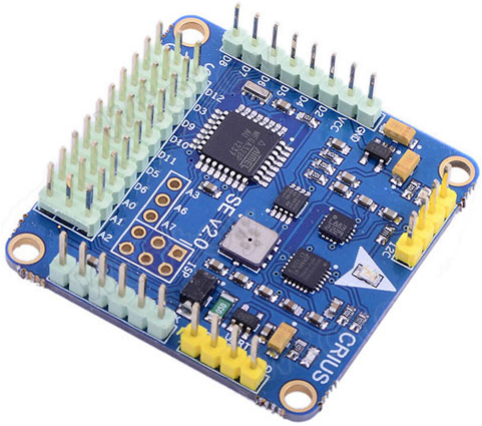

Figura 2.4 – CRIUS v2.0 board (de (CRIUS V2.0 PICTURE, 2016))

2.1.4 Receptor de Rádio e Rádio Controle

O rádio-receptor é conectado a unidade controladora para enviar a ela dados de *pitch*, *roll*, *yaw*, *throttle* e os canais auxiliares enviados pelo rádio controle. Esse é a forma mais recorrente de se controlar drones. A maioria dos *softwares* de controle são desenvolvidos já pensando em serem usados dessa forma, o que é um obstáculo para desenvolvedores que desejem construir um drone autônomo. Normalmente, não há mecanismos para enviar dados para o drone que não pelo rádio-controle. Exemplos de receptor de rádio e rádio-controle são mostrados na Figura 2.5.

Figura 2.5 – Remote Controller and receiver (de (RADIO CONTROLLER AND RECEIVER PICTURE, 2016))

2.1.5 Bateria

A bateria (Figura 2.6) é responsável por fornecer energia para tanto os motores quanto a unidade controladora. Multirotores usam bastante energia, puxando bastante corrente elétrica. Por isso, é necessário usar uma bateria potente. Um quadcóptero regular normalmente carrega baterias Li-Po 11.1V 5000mAh. Em condições normais, essa bateria é capaz de fornecer energia para 15-25 minutos de voo, dependendo de vários fatores, como peso, motores, hélices, etc...

Figura 2.6 – 2200 mAh Li-Po battery (de (LI-PO BATTERY PICTURE, 2016))

2.2 *Softwares* de Voo

Antes de se entrar em maiores de detalhes de *softwares* específicos, será apresentada a nomenclatura comum adotada por todos eles. Com relação a software de voo, incluindo controle de estabilidade, têm-se várias plataformas livres disponíveis, e.g., MultiWii (2016) e ArduPilot

(2016). Ambos serão analisados separadamente. Um terceiro exemplo, diferente desses dois também será discutido.

2.2.1 Nomenclatura de Voo

Em termos de nomenclatura, o básico que se deve ter em mente são os valores que as unidades de controle usam, normalmente recebidos pelo rádio-controle. Eles são apresentados abaixo e, após isso, são discutidos um a um. Vale ressaltar que eles sempre possuem seus valores em $[1000, 2000]$

- *Pitch*;
- *Roll*;
- *Yaw*;
- *Throttle*;
- *Aux1*;
- *Aux2*;
- *Aux3*;
- *Aux4*.

Para ilustrar o funcionamento desses valores será usado um quadcóptero em formato 'X', onde a frente está entre dois motores, mas o conceito é geral quando se fala em drones. *Pitch* consiste em fazer o drone se mover para a sua frente ou para trás. Para se projetar para a frente, i.e. $pitch > 1500$, basta reduzir a rotação dos motores da frente e aumentar a dos de trás. Isso criará uma inclinação no modelo, fazendo a sua força motriz, i.e., força gerada pelos motores, ficar inclinada, surgindo uma componente horizontal no plano de estabilidade do quadcóptero, i.e, plano do quadcóptero quando estável. Para se movimentar para trás, i.e, $pitch < 1500$, basta fazer o inverso. $Pitch = 1500$ faz o drone manter sua posição nesse eixo de movimento. *Roll* é similar, mas diz respeito a movimentos laterais. Para se mover para a direita, i.e., $roll > 1500$, reduz-se a intensidade dos motores da direta e aumenta-se dos da esquerda. Para se mover para a esquerda, i.e, $roll < 1500$, faz-se o inverso. Novamente, $roll = 1500$ mantém a inclinação do drone nesse eixo. *Yaw* refere-se à rotação do modelo em

torno de si. Para isso, eleva-se a rotação de motores opostos e reduz-se dos outros. Como os motores opostos giram na mesma rotação e apenas dois motores têm a mesma direção de giro, e.g. horário ou anti-horário, reduções dessa forma influenciam no momento angular resultante do modelo, tornado-o diferente de zero e causando uma rotação sobre seu próprio eixo. *Yaw* $<$ 1500 faz o drone girar no sentido anti-horário e *Yaw* $=$ 1500 faz o drone rotacionar no sentido horário. *Yaw* $>$ 1500 faz o drone manter sua rotação em zero. *Throttle* diz respeito aceleração geral dos motores. Quanto mais aceleração, mais força, e consequentemente, mais aumento de altitude. Aqui, não existe um valor padrão para se manter a altitude, variando de modelo para modelo. Com todos esses valores, cabe a unidade de controle interpretá-los e combiná-los para decidir qual será a aceleração efetiva a ser mandada para cada.

Além dos valores básicos, têm-se os 4 canais auxiliares. Eles são usados para controlar os chamados modos de voo. Nos rádios-controle, eles são atribuídos a chaves de três posições. Essas posições representam o canal em 1000, 1500 e 2000. Nos *softwares* de controle, pode-se atribuir um modo de voo para uma combinação dos canais auxiliares. Por exemplo, um modo pode ser representado por aux1 em 2000, aux2 em 1000 e aux3 em 1500. Em termos de modo de voo, serão apresentados apenas os principais: *Acro*, *Angle*, *Horizon* e *Altitude Hold*. Em *Acro*, o drone apenas compensa as diferenças detectadas pelo giroscópio. Em termos práticos, caso ele mude de orientação, ele tentará impedir que o movimento continue a ocorrer, mas quando ele parar, ele continuará na orientação que está, podendo ela ser inclinada. Já no *Angle*, o drone tenta permanecer paralelo ao solo, fazendo uso de um acelerômetro e do giroscópio. *Horizon* é uma mistura dos dois anteriores, agindo como *Angle* normalmente e, quando as mudanças de valores de controle são bruscas, age como *Acro*. Por fim, *Altitude Hold* faz o drone tentar manter a sua altitude, usando o acelerômetro, o giroscópio e um sensor de altitude, podendo ser um barômetro e/ou um GPS, usando algum algoritmo de controle, como uma malha Proporcional-Integral-Derivativa (PID) (ANG; CHONG; LI, 2005). É trabalho da unidade de controle monitorar os modos de voos e levá-los em conta ao calcular a aceleração a ser enviada para cada motor.

2.2.2 Multiwii

Como mencionado, o MultiWii é um software de controle de voo. Ele pode interpretar valores de entrada enviado pelo rádio controle e calcular a atuação que cada motor deve exercer para que o drone mova-se adequadamente. Em termos de plataformas de *hardware* necessárias

para rodá-lo, ele é bastante flexível. Ele é um projeto de Arduino (ARDUINO OFFICIAL WEBSITE, 2016) e pode rodar em qualquer placa que tenha suporte para o mesmo. Embora ele possa ser executado em placas genéricas, como o Arduino Uno, existem placas planejadas especialmente para drones. Um exemplo é a CRIUS v2.0, mostrada na Figura 2.4. Conectá-la aos ESCs é uma tarefa bastante simples, já que os seus Pinos Gerais de Entrada e Saída (GPIO) estão posicionados de acordo com conectores de ESCs. Em adição a isso, essas placas possuem uma IMU integrada. Em placas genéricas, sensores devem ser conectados manualmente.

A comunidade de uso desse software é grande e ativa. É possível encontrar vários tutorias sobre como fazer o *software* executar. Para isso, é necessário fazer algumas configurações no código. A *build* mais simples necessita que o usuário defina duas macros:

1. Qual placa será usada, e.g., a CRIUS v2.0;

2. A configuração do drone, e.g., octacóptero, quadcóptero em forma de X, entre outros.

Então basta gravar o *software* que o drone está quase pronto para o voo. ESCs ainda precisam ser calibrados. Isso pode ser feito via MultiWii também. Apenas descomentando uma linha do código, cria-se uma nova versão do MultiWii para calibrar os ESCs.

Em relação ao seu algoritmo de controle, MultiWii implementa uma malha PID. Esse algoritmo é responsável por compensar erros de posicionamento detectados ao se analisar os dados da IMU. Por exemplo, se um quadcóptero estiver desnivelado para um lado, a tendência é de ele movimentar para esse lado. O algoritmo identificará esse desnivelamento e compensará para evitar que se mantenha.

O MultiWii também possui uma interface de configuração, chamada de MultiWiiConf. É uma Interface Gráfica de Usuário (GUI) que permite que um usuário se conecte ao MultiWii a partir do computador usando um canal de comunicação serial, como Universal Serial Bus (USB) ou Bluetooth. Com ela, pode-se analisar e calibrar a IMU, alterar valores de PID e as atribuições de canais auxiliares.

MultiWii implementa algumas funcionalidades de segurança. Não é possível armar o drone, i.e., ligar seus motores, sem o valor de *throttle* estar abaixo de um valor mínimo. Isso é feito para que o drone não comece a voar sem que o piloto explicitamente queira. Além disso, a ferramente implementa uma rotina *failsafe*. Sempre que perde contato com o rádio por um período de tempo, ele automaticamente muda seu modo de voo para voo estável e aterrissa lentamente.

Embora o MultiWii seja feito para ser controlado por rádio controle, ele possui um mecanismo para acessar seus dados e o controlar via serial. Esse inclusive é o mecanismo usado pelo MultiWiiConf. É um protocolo conhecido por *MultiWii Serial Protocol* (MSP) (MULTIWII OFFICIAL WEBSITE, 2016) melhor descrito em 2.2.2.1. Embora esse mecanismo exista, não foi encontrado ferramenta que o implemente e permita controlar o drone a partir de uma porta serial, possivelmente de um computador de companhia. Pelo MultiWii ter sido a solução adotada e pela total relevância do MSP no trabalho, ele será descrito na sequência.

2.2.2.1 MSP

MSP, conforme mencionado em 2.2.2, é um protocolo para interagir com o *software* MultiWii através de um canal de comunicação serial, normalmente através de uma conexão cabeada. Ele descreve como devem ser feitas requisições à placa executando o MultiWii. Ele é um protocolo cliente-servidor, onde o MultiWii é um servidor respondendo às requisições. Os dados são enviados em pacotes da forma:

'$'	'M'	Dir	Size	Opcode	Data	Checksum

O MultiWii apenas transmite informação quando necessário. Se recebe um pedido que não necessita de resposta, ele não responderá. A seguir, serão apresentados os campos que formam a mensagem.

2.2.2.1.1 Preâmbulo e Direção

Para iniciar a mensagem, enviam-se os dois caracteres '$M' (ASCII). O próximo caractere especifica a direção na qual a mensagem está sendo transmitida. Se a mensagem está sendo enviada para o MultiWii, adiciona-se '>'. Se se está lendo da placa, o caractere '<' estará nessa posição. Ao monitorar um canal serial, um preâmbulo permite encontrar o começo de uma mensagem.

2.2.2.1.2 Tamanho

Em seguida, é enviado o número de bytes que irão compor a mensagem. Por exemplo, se o dado enviado são 8 valores básicos de controle, cada um representado com inteiros de 16 bits, a seção de tamanho possuirá o valor 32.

2.2.2.1.3 Opcode

Opcode é um byte contendo o código da operação a ser executada. Por exemplo, a operação para enviar valores de controle (conhecida por MSP_SET_RAW_RC) possui o opcode 200. MSP_RC, para recuperar os valores de controle na placa, possui Opcode 105. A lista completa das operações suportadas podem ser encontradas em (MULTIWII OFFICIAL WEBSITE, 2016).

2.2.2.1.4 Dados

Dados são as informações sendo trocadas. Cada operação necessitará de um número específico de bytes. Por exemplo, a operação MSP_SET_RAW_RC requer que 8 inteiros de 16-bit, representando *roll*, *pitch*, *yaw*, *throttle*, *aux1*, *aux2*, *aux3* e *aux4* (nessa ordem), sejam enviados. As informações de 16 bits devem ser serializadas na mensagem como 2 bytes, com o byte menos significante enviado antes. Os dados necessários para cada mensagem são descritos em (MULTIWII OFFICIAL WEBSITE, 2016).

2.2.2.1.5 Checksum

O checksum é calculado realizando a operação XOR entre tamanho, opcode e os bytes de dados. Essa informação é um excelente mecanismo para identificar distorção de dados na comunicação. Deve-se ter em mente que o canal será, de certa forma, próximo de motores elétricos. Esses motores geram interferência induzindo correntes elétricas no cabo de comunição serial.

2.2.3 ArduPilot

ArduPilot é um outro projeto de software controlador de drones. Além de multirotores, possui também suporte para aeronaves de asa fixa e veículos sobre roda. Em si, a proposta do ArduPilot é muito parecida com o MultiWii. Ele implementa o necessário para fazer um drone funcionar. Ele interpreta dados recebidos de um rádio controle e implementa um PID usando os dados de uma IMU. Também possui suporte a diferentes modos de voo. Entretanto, essa solução é um pouco restrita. Ela é dependente de uma família pequena de placas, não sendo tão flexível assim. Além disso, essas placas possuem um custo bastante superior em relação as placas suportadas pelo MultiWii.

Apesar de ser de mais alto custo, esta plataforma possui um pouco mais de suporte. Ela possui uma aplicação similar ao MultiWiiConf, mas com algumas funcionalidades a mais. A principal delas é possibilidade de planejar uma rota de voo. Além disso, o carregamento e configuração do código é feito diretamente por ela.

Com relação a controle usando um computador de companhia, essa é uma tarefa mais fácil nessa plataforma. Diferentemente do MultiWii, que implementa um protocolo próprio, o ArduPilot usa um protocolo bastante conhecido, o MavLink (MAVLINK OFFICIAL WEBSITE, 2016). Além disso, existe uma biblioteca de terceiros que implementa uma API de controle via computador de companhia. Essa ferramenta é o DroneKit (DRONEKIT OFFICIAL WEBSITE, 2016). Ela permite que, de forma bastante simples e explícita, se escreva *software* na linguagem de programação Python para controlar o drone.

2.2.4 API para AR.Drone 2.0

Em (LUGO; ZEIL, 2013), tem-se uma implementação de uma ferramenta que permite escrever software para guiagem autônoma de um quadcóptero. Diferentemente do MultiWii ou do ArduPilot, é usada para fornecer ao programador ferramentas para escrever *software* para navegação autônoma, e não para implementar a malha de controle, assemelhando-se ao DroneKit.

Desenvolvida com propósito educacional, essa *Application Programming Interface* (API) permite que alunos emitam comandos da mudança de altitude e orientação (e consequentemente posição) de um quadcóptero sem precisarem se preocupar com detalhes de baixo nível. Por ser escrito na linguagem de programação Java, permite a integração com bibliotecas externas. Por exemplo, um drone controlado por voz foi desenvolvido apenas integrando a API com uma biblioteca de processamento de linguagem natural. Porém, a solução não é genérica. Ela é específica para o drone AR.Drone 2.0, um produto comercial, não satisfazendo as necessidades de muitos desenvolvedores. Por exemplo, se um usuário desejar modelar e construir seu próprio drone, não será possível utilizar a API.

3 USO E DESENVOLVIMENTO DE FRAMES

Neste capítulo serão apresentados em detalhes os *frames* usados nos protótipos desenvolvidos. Entre eles, encontra-se um frame disponível comercialmente e os outros foram desenvolvidos no decorrer do trabalho. .

3.1 Frame Comercial

Devido a disponibilidade no laboratório, decidiu-se usar um frame comercial. Esse *frame* é bastante fácil de ser encontrado para compra. Esse *frame* foi comprado juntamente a motores, ESCs e hélices.

O *frame* é composto principalmente de dois materiais. Os braços são feitos de alumínio oco e bastante fino, sendo leves. As outras partes, como o centro, o suporte dos motores e os mecanismos de pouso são feitos de fibra carbono, outro material bastante leve. Em relação ao centro do *frame*, ele possui quatro furos para se parafusar elementos de *hardware*, como um computador de companhia ou uma unidade controladora. Esses elementos então podem ser facilmente empilhados. Os suportes dos motores também já possuíam os furos necessários para se parafusarem os motores. Os ESCs ficavam presos com fita isolante nos braços. Com relação ao mecanismo de pouso, ele era fabricado com fibra de carbono, sendo leve e resistente. Alem disso, ele ficava preso ao corpo por meio de um parafuso e uma mola. O parafuso servia de eixo para o mecanismo de pouso rotacionar, enquanto que a mola limitava esse giro. Essa combinação permitia que a mola amortecesse o impacto do pouso. As hélices também eram bastante robustas, sendo feitas de fibra de carbono. Esse *frame* é mostrado na Figura 3.1.

Apesar de aparentemente o *frame* ser bastante robusto, foram encontrados alguns problemas no seu uso. Primeiramente, ele é muito suscetível a vibrações. No primeiro teste, ao se ligarem os motores, era visível a vibração. Além disso, alguns parafusos se soltaram muito facilmente. Essa frame não chegou a voar devido a alguns problemas técnicos. Os motores e os ESCs foram conectados de forma errada, fazendo as hélices girarem no sentido oposto ao que deveriam.

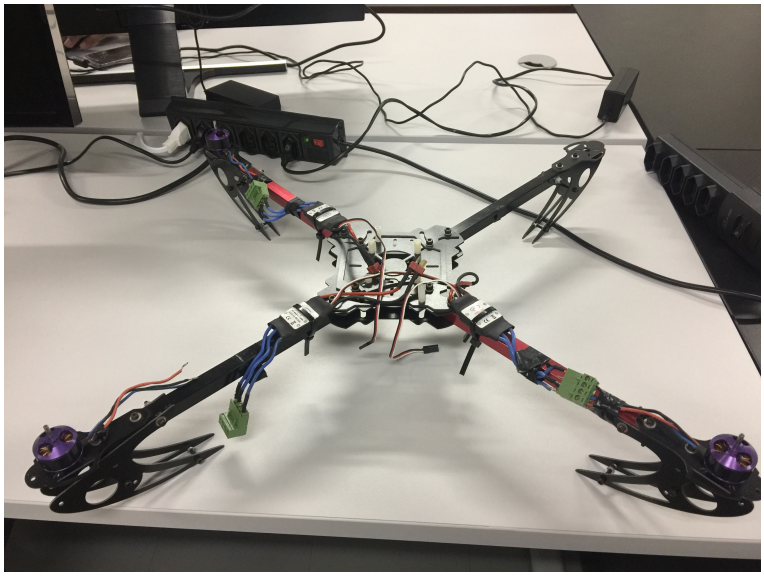

Figura 3.1 – *Frame* comercial

3.2 Primeiro Projeto de Frame

Para resolver os problemas com vibração, optou-se por modelar um *frame* para ser impresso em uma impressora 3D disponível no laboratório. Para o processo de modelagem, decidiu-se usar o *software* 123D Design (123D DESIGN OFFICIAL WEBSITE, 2016) por ser de fácil aprendizado. A maior parte das imagens de modelos apresentadas no trabalho foram capturadas nele, sendo que a única que não está indicada. A impressora, usada para esse e para os seguintes projetos, foi uma Hyrel modelo System 30M (HYREL OFFICIAL WEBSITE, 2016). Essa é uma impressora 3D que imprime usando diversos filamentos de plástico. Para esse projeto, optou-se por usar plástico Acrilonitrila-Butadieno-Estireno (ABS) por ser um plástico bastante resistente e pouco flexível, qualidades fundamentais para um drone. As peças foram impressas com densidade de 35 por cento. O *frame* pode ser visto na Figura 3.2.

Para maior segurança dos ESCs, idealizou-se que eles e os cabos fossem passado por dentro dos braços. Portanto, eles deveriam ser ocos. Eles também deveriam ser bastante compridos devido o tamanho das hélices (25,5cm de diâmetro). De forma a evitar que as hélices girassem sobre o centro, a peça foi projetada possuindo 21,7 cm de comprimento. Como a impressão é feita por camadas, imprimir uma peça como um braço oco seria um pouco complicado. Caso a peça fosse feita na horizontal, haveria diversas peças de suporte para que as camadas em cima do vão não caíssem. Em contrapartida, imprimir na vertical também não seria uma boa solução já que havia alguns vãos na estrutura que também necessitariam de suportes

Figura 3.2 – Primeiro projeto de *frame*

caso a peça fosse impressa nessa orientação. Para resolver esse problema, a peça foi dividida em duas idênticas. Essas peças possuíam um encaixe macho e um fêmea, que permitia que as duas se unissem para formar o braço. Essa peça é mostrada na Figura 3.3. Os furos na parte central são para diminuir a área de superfície, diminuindo a resistência do ar sobre o braço. Os dois furos na parte maior foram feitos para prender o braço na peça central, evitando que ela rotacionasse em qualquer direção. O furo na parte menor serve para parafusar o suporte do motor.

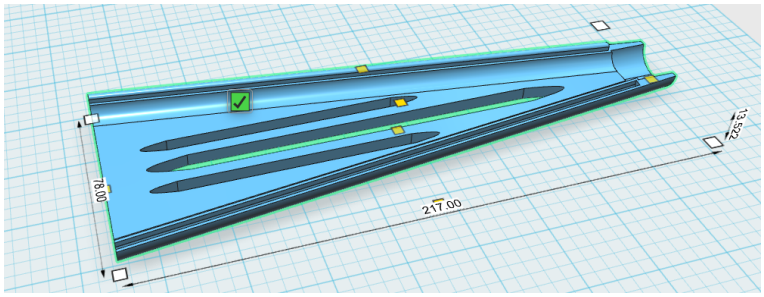

Figura 3.3 – Braço do primeiro projeto de *frame*. Medidas representadas em milímetros

A seguir, projetou-se o suporte do motor. A ideia era fazer uma peça simples de ser impressa e que permitisse facilmente montar os motores nela. Além disso, inspirado no *frame* comercial, essa peça possuiria também o equipamento de pouso, responsável por absorver o impacto do pouso e por fazer com que o corpo possuísse uma certa altura do chão, permitindo que a bateria fosse alocada em baixo. Essa peça é mostrada na Figura 3.4. Ela já possui os furos exatamente espaçados de acordo com o motor. Os espaços nas laterais são para permitir

que, durante a montagem do motor sobre a peça, se insira porcas nos parafusos do motor. Essa peça, para facilitar o processo de impressão, foi impressa com a sua parte de cima na bandeja de impressão. Ela e o braço eram unidos por um parafuso.

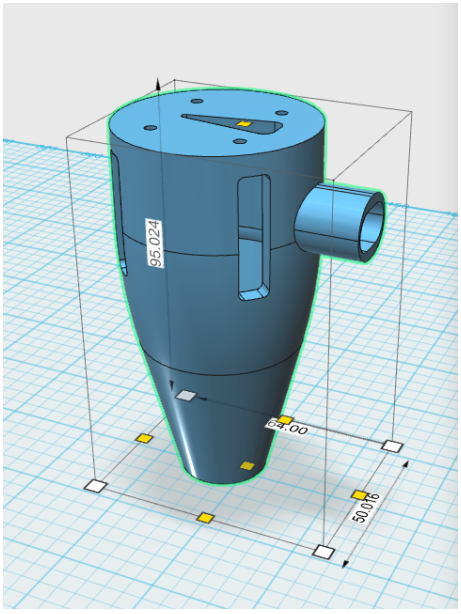

Figura 3.4 – Suporte do motor do primeiro projeto de *frame*. Medidas representadas em milímetros

O centro foi uma peça um pouco mais simples. Ele apenas deveria possuir os furos para parafusar acima dele elementos de *hardware*, i.e. um computador de companhia, um espaço interno para acomodar uma placa distribuidora de energia e se prender aos braços. Como ela deveria possuir elementos em seu interior, ela também foi separada em duas peças. A de cima, mostrada na Figura 3.5, possuía os furos para *hardware* e uma abertura central que permitia um fácil acesso a placa distribuidora de energia e passagem de cabos. A parte inferior, mostrada na Figura 3.6, possuía furos para parafusar a placa distribuidora de energia e um furo maior para permitir a passagem do cabo da bateria. Essas duas partes eram unidas pelos parafusos que as conectavam aos braços, dois em cada lado.

Logo na montagem, já se percebeu alguns problemas. O *frame* ficou demasiadamente difícil de montar. Os parafusos usados para prender os motores, por exemplo, ficaram de difícil acesso. Além disso, pequenos problemas, como desconexões de cabos, muitas vezes ocasionariam em se ter que desmontar o drone por completo. Assim já se sabia que o *frame* deveria ser redesenhado. Em adição a isso, o *frame* possuía uma fragilidade. Devido ao formato da peça do suporte do motor e a técnica de impressão, a parte que conectava essa peça ao braço poderia quebrar. Outro problema foi o fato do centro ser quadrado e desejar-se construir um quadcóp-

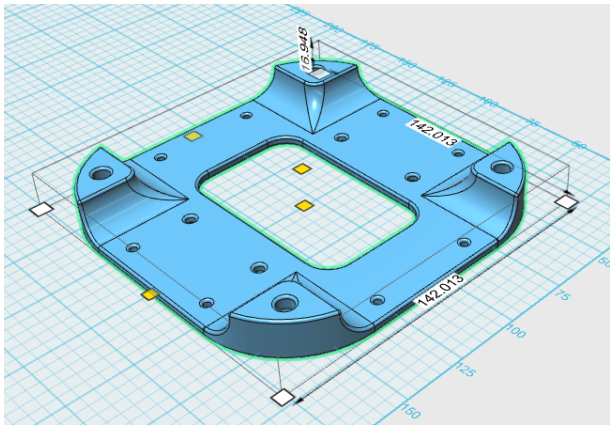

Figura 3.5 – Peça central superior do primeiro projeto de *frame*. Medidas representadas em milímetros

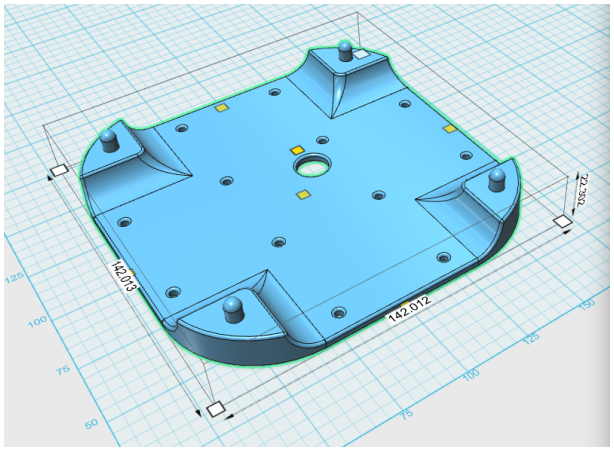

Figura 3.6 – Peça central inferior do primeiro projeto de *frame*. Medidas representadas em milímetros

tero no formato X. Ficou um pouco difícil posicionar a placa controladora com sua frente entre dois motores.

3.3 Segundo Projeto de Frame

Conforme dito, o outro *frame* acabou sendo difícil de ser montado. Por esse motivo, tornou-se muito difícil prestar manutenção, uma vez que consertar certos problemas requeriam a desmontagem e remontagem completa do drone. Somando-se isso aos problemas causados pelas partes frágeis, decidiu-se remodelar o *frame*.

O trabalho de modelagem nessa versão já foi orientado no intuito de corrigir os erros cometidos na modelagem da primeira versão. Para evitar que a desconexão de cabos no interior de um braço do quadcóptero acarretasse em uma desmontagem completa, o braço foi pensado para ser aberto. Além disso, era vazado para diminuir o arrasto. Os furos para prender essa peça

no corpo foram mudados de uma disposição horizontal para uma vertical. O motivo dessa mudança foi para, num futuro, trocar um dos parafusos por um prego. Esse prego, ao ser retirado, faria o parafuso remanescente servir de eixo para rotacionar os braço e deixar o drone mais enxuto para o transporte. A parte superior do suporte do motor do modelo antigo, onde existiam os furos para prender o motor, foi juntada a esse modelo, sendo agora o motor parafusado nele. Uma imagem dessa peça é mostrada na Figura 3.7.

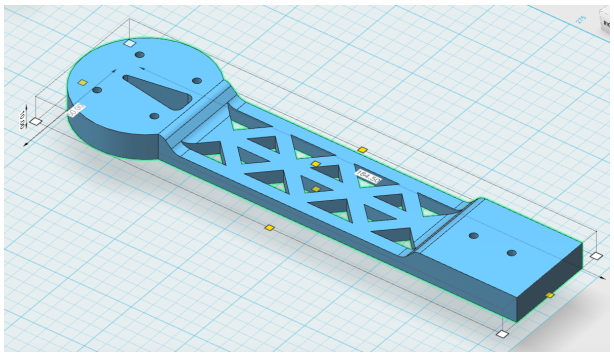

Figura 3.7 – Braço do segundo projeto de *frame*. Medidas apresentadas em milímetros

O mecanismo de pouso foi realocado das extremidades para o centro para resolver o problema dos parafusos de difícil acesso na peça antiga. Essa peça foi obtida em partir de um repositório livre e está disponível em (THINGVERSE OFFICIAL WEBSITE, 2016). O modelo dessa peça é mostrado na Figura 3.8. Ela apenas foi escalada para ficar proporcional ao tamanho do modelo desenvolvido. Essa peça inclusive facilitou a alocação da bateria, que poderia ficar presa nela com fita.

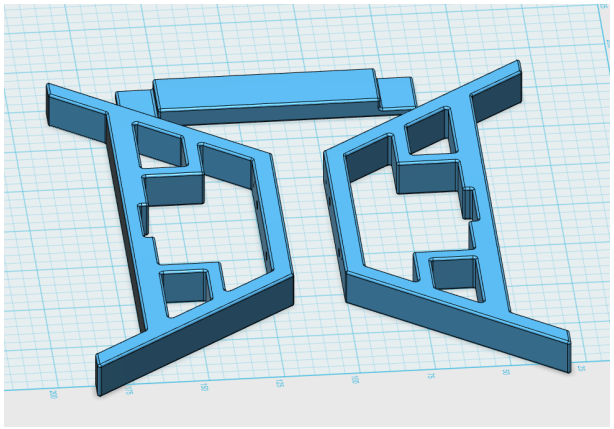

Figura 3.8 – Mecanismo de pouso do segundo projeto de *frame*.

Outra peça que sofreu bastante alteração foi o centro. Ele continuou sendo composto por duas peças para acomodar em seu interior uma placa de distribuição de energia. Entretanto,

para facilitar o posicionamento do *hardware* montado sobre ele, ele passou a ser octogonal. Dessa forma, usam-se quatro faces intercaladas para ligarem-se aos braços. Os furos no centro já são pensados de forma que o *hardware* ficasse com sua frente voltada para um dos lados que não possuísse braços conectados, i.e., entre dois braços. Os furos para conexão da bateria e para passagem de cabos para a parte superior foram mantidos similares ao do projeto antigo. O modelo dessa peça pode ser visto na Figura 3.9.

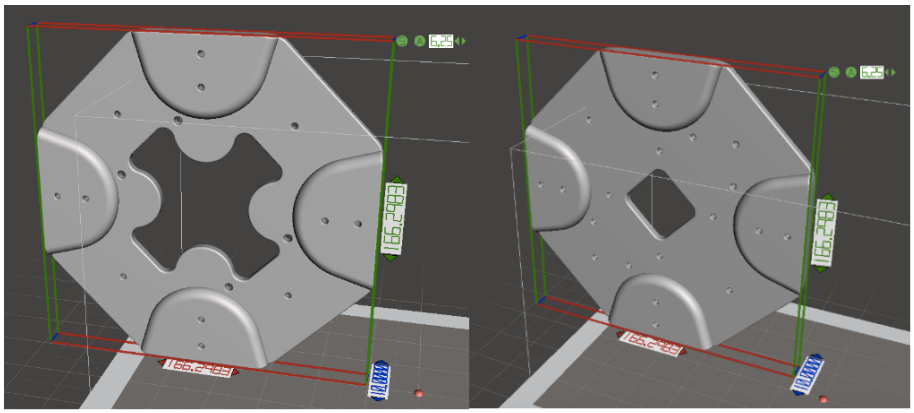

Figura 3.9 – Peças do centro do segundo projeto de *frame*. Peça da esquerda é a superior e, a da direita, a inferior. Dimensões representadas em milímetros. O *software* usado para essa visualização foi o Meshmixer (MESHMIXER OFFICIAL WEBSITE, 2016).

Para esse modelo, também foram modeladas proteção para as hélices. Elas ficaram presas nos mesmos parafusos que prenderam os motores aos braços. Devido ao formato, essa peça também foi impressa em duas partes. Elas são mostradas nas Figuras 3.10 e 3.11.

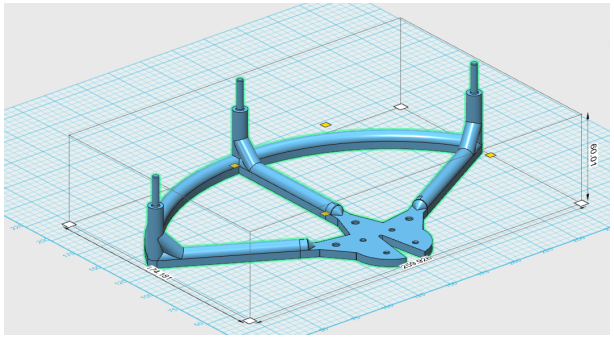

Figura 3.10 – Parte inferior da proteção de hélice do segundo projeto de *frame*. Medidas apresentadas em milímetros.

Essas peças foram então impressas. A montagem do drone foi muito mais simples se comparado ao modelo antigo, levando cerca de uma hora e meia. O modelo pronto pode ser visto na Figura 3.12.

O drone foi então levado para testes. Neles, percebeu-se um problema grave. Os braços

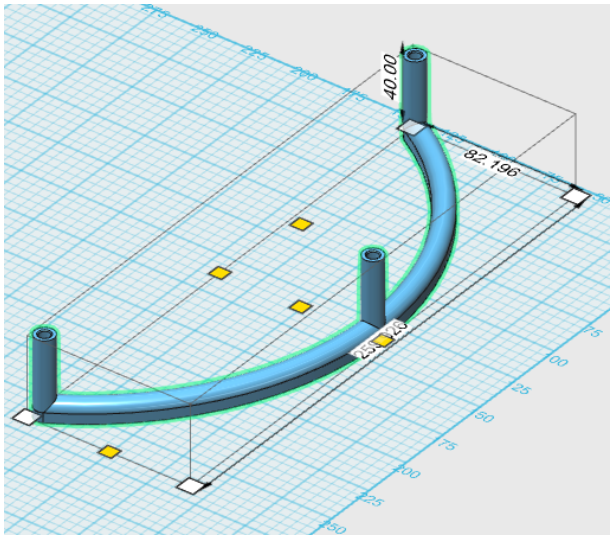

Figura 3.11 – Parte inferior da proteção de hélice segundo projeto de *frame*. Medidas apresentadas em milímetros.

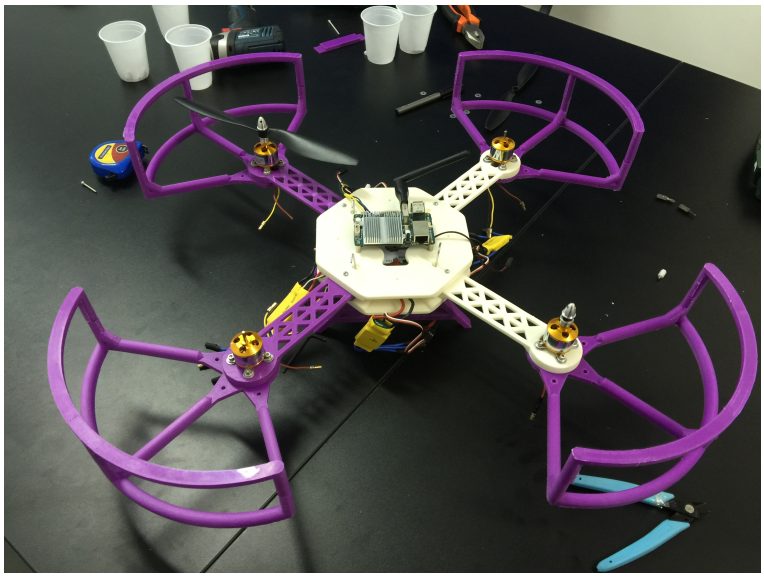

Figura 3.12 – Segundo projeto de drone

se curvaram com a força exercida pelo par motor-hélice, essa curvatura alterando o vetor de força criado pelos motores. Fora isso, o drone foi bastante robusto, inclusive resistindo a fortes quedas sem maiores danos. Alguns poucos danos foram corrigidos facilmente.

3.4 Melhorias do Segundo Projeto de Frame

Seguindo o quase sucesso do *frame* anterior, essas melhorias focaram principalmente na questão dos braços estarem se curvando. Para isso, a peça de plástico foi substituída por barras de alumínio. Com 30cm cada, elas receberam furos para serem presos no corpo conforme as

contrapartes de plástico. Para prender os motores nelas, foi feito um adaptador com os furos necessários para se prender o motor nele e com um furo para um parafuso prender essa peça no braço, que também recebeu um furo para acomodar o adaptador. Essa peça é mostrada na Figura 3.13. Para substituir a proteção das hélices, apenas posicionou-se esse adaptador não próximo da extremidade da barra. Dessa forma, as partes sobressalentes das barras impediam as hélices de baterem no chão.

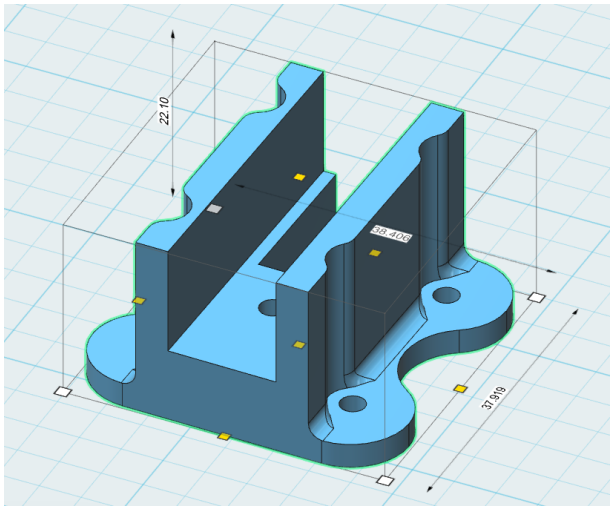

Figura 3.13 – Adaptador para prender os motores no braço de alumínio. Medidas representadas em milímetros

Em testes, esse frame apresentou bons resultados. Ele inclusive ficou mais robusto que o anterior, além de seus braços não se curvarem. Mesmo com algumas quedas um tanto quanto fortes, ele suportou muito bem, apenas com o mecanismo de pouso quebrando em alguns pontos fáceis de serem consertados. Uma imagem do modelo final é mostrado na Figura 3.14.

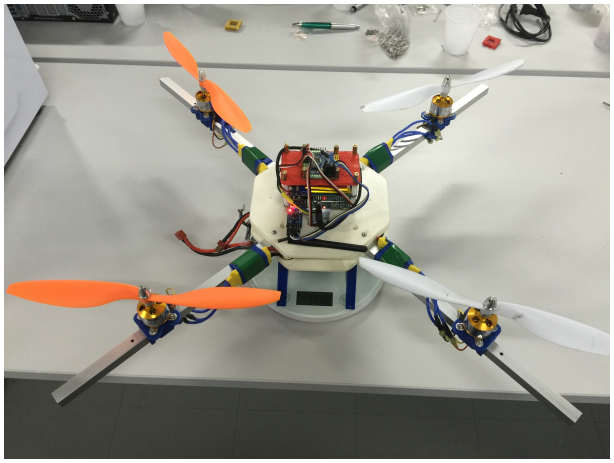

Figura 3.14 – Modelo final do *frame*.

4 DESENVOLVIMENTO DE SOFTWARE

A seguir, será apresentada a evolução do drone ilustrada usando os testes. O computador de testes utilizado possui a seguinte configuração: Laptop MacBook Air® 2015, executando OS X El Capitan, 8 GB de RAM, processador Intel Core i7 2,2 GHz e Intel HD Graphics 6000 1536 MB.

Note que o desenvolvimento de *software* foi feito concomitantemente ao desenvolvimento dos *frames*. Inclusive, muitas alterações nos *frames* foram feitas por causa de resultadas de testes de *software*. Por esse motivo, títulos das seções desses capítulos são similares aos do capítulo de desenvolvimento de *frames*. As seções sem referências a algum estágio de desenvolvimento de *frames* dizem respeito a etapas realizadas antes de se montar um quadcóptero. A partir disso, os títulos indicam onde, cronologicamente, as etapas de desenvolvimento de *software* se inserem.

4.1 Teste Inicial do MSP

O primeiro teste foi feito apenas para compreensão da forma de funcionamento do protocolo MSP. Para isso, escreveu-se mensagens MSP na porta serial entre um Laptop e a placa CRIUS v2.5. Para facilitar o trabalho com estruturas de baixo nível, como deslocamento de bits para montar a mensagem, optou-se por fazer esse trabalho inicial na linguagem de programação C. Pelo fato de as mensagens de leitura receberem respostas, elas foram as primeiras a serem experimentadas. Fez-se leituras como de valores de controle e dados da IMU. Os testes foram bem sucedidos, o que levou a um bom entendimento sobre o protocolo.

A partir disso, os processos de montar e enviar as mensagens foram encapsulados em um Middleware melhor descrito na sequência. Com ele, fez-se testes de leitura e de escrita. Para a escrita, como não há retorno de valores, adotou-se um outro método para testar a funcionalidade. Na placa CRIUS v2.5, há um *Light Emiting Diode* (LED) que acende para indicar se o MultiWii está enviando valores de aceleração para os motores, i.e., está com eles armados. Para armar os motores, deve-se enviar o valores de *Throttle* 1000 e *yaw* 2000 para o MultiWii. Ao passo que se fez isso, o LED acendeu, indicando que as escritas estavam funcionando.

4.2 Implementação de um Middleware

A ideia do Middleware é fornecer uma API para abstrair detalhes específicos do protocolo de comunicação MSP. Dessa forma, desejava-se produzir um Middleware para interagir com o drone de um simples maneira procedural. Ao invés de estar sempre lidando com detalhes do protocolo e leituras e escritas de portas seriais, foi desenvolvida uma ferramenta que permitia que a interação fosse feita em forma de chamadas de função e, caso a operação devesse retornar alguns valores, eles estariam em estruturas de dados bem definidas. O Middleware foi desenvolvido para ser executado em uma plataforma como a Raspberry Pi ou Odroid montado no drone, responsável por controlar o mesmo. Essas plataformas, nessa situação, são conhecidas como computador de companhia.

Em termos de linguagem de programação, foi escolhida a linguagem C devido a sua eficiência e porque, com ela, é mais fácil lidar com estruturas de baixo nível, conforme mencionado anteriormente. O Middleware segue uma estrutura modular, sendo dividido em três módulos mostrados na sequência. Vale notar que cada módulo consiste em um par de arquivos .c/.h. O código na íntegra pode ser visualizado no Anexo A.

- MultiWii Network;
- MultiWii Protocol;
- MultiWii Manipulation.

4.2.1 *MultiWii Network*

Esse módulo é dedicado somente para para fornecer um acesso simplificado ao canal de comunicação. É usado para abrir a porta serial e configurá-la. Embora algumas dessas configurações possam ser mudadas, a configuração padrão do MultiWii para comunição serial é:

- Baudrate de 115200;
- Paridade de bit desabilitada;
- Sem bit de parada;
- 8 bits de dados recebidos por vez.

Tabela 4.1 – MultiWii Network API

int init_serial_port()	Inicia a conexão
int send_data(char *data, int size)	Envia um pacote MSP pela conexão
received_frame_t get_data(int expected_size)	Recebe dados pela porta serial
void close_serial_port()	Fecha a conexão

Portanto, o Middleware deve ter a sua porta serial aberta com as mesmas opções. Além disso, as leituras possuem um tempo de expiração para caso um pacote tenha sido corrompido. Sem a presença de um tempo de expiração, o Middleware bloquearia indefinidamente, uma vez que o MultiWii não responde a requisições corrompidas.

Esse módulo não deve ser usado pelo programador de aplicações usando o Middleware. Ele apenas fornece um nível de abstração a ser usado internamente no Middleware. Em relação à interface, esse módulo possui funcionalidades bastante simples, mostradas na Tabela 4.1. Quando é necessário realizar uma escrita, ele simplesmente recebe um pacote MSP e envia pela porta serial. Leituras podem retornar ou um pacote de dados um erro em caso de *timeout*. Note que todas as funções na tabela têm seus nomes prefixados por "multiwii_", omitido apenas por brevidade.

Para definir o nome da porta serial em que se encontra o MultiWii, o autor do código deve alterar uma variável global em MultiWiiNetwork.cpp. Essa varável é usada por *init_serial_port* para abrir a conexão. As outras funções são bastante triviais e auto explicativas: *int send_data(char *data, int size)* envia um pacote de dados "data" de comprimento "size" usando a conexão aberta anteriormente; *received_frame_t get_data(int expected_size)* retorna um *received_frame_t*, uma estrutura definida nesse módulo que é apenas um vetor de caracteres com o seu tamanho; *void close_serial_port()* fecha a conexão anteriormente aberta. Note que é apenas possível abrir conexão com um MultiWii por vez, o que é uma limitação bastante razoável. As funções que retornam inteiros assim são apenas para retornam um número diferente de 0 caso bem sucedidas.

4.2.2 MultiWii Protocol

Esse módulo é responsável por adquirir informações e formatar de acordo com o formato MSP, descrito na Seção 2.2.2.1. Ele possui apenas um método genérico que pode gerar qualquer mensagem MSP. Ele recebe como parâmetro o Opcode, um vetor de bytes de dados e seu tamanho. Junto a isso, ele recebe um *buffer* para registrar a mensagem. O único processamento

Tabela 4.2 – API MultiWii Manipulation

int init_manipulation()	Inicializa a Manipulação do MultiWii
imu_t get_imu()	Lê os dados da IMU
control_t get_control()	Lê os dados de controle
void set_control (control_t ctrl)	Escreve os dados de controle no MultiWii
altitude_t get_altitude_data()	Lê os dados de altitude
motor_t get_motor_data()	Lê as informações de *throttle* de cada motor motor
void stop_manipulation()	Encerra o MultiWii Manipulation

necessário é *checksum*. Assim como o módulo descrito em 4.2.1, a sua interface não deve ser usada pelo usuário final.

Para facilitar o desenvolvimento, esse módulo define algumas macros para que o usuário faça referência aos Opcodes por nome ao invés de números.

4.2.3 MultiWii *Manipulation*

Esse é o módulo que é de fato exposto para o usuário final. Ele é construído sobre os outros dois módulos, possuindo uma interface simples. Por exemplo, caso o usuário deseje escrever dados de controle, ele apenas fará uma chamada de função, sem ter que se preocupar com o protocolo de comunicação ou com a tecnologia usada. Todos os detalhes do MSP estão escondidos do usuário, que pode focar simplesmente em produzir aplicações.

A versão atual do Middleware implementa as principais funções necessárias para voos autônomos. Além disso, ele define algumas estruturas para facilitar a manipulação de dados de voo, e.g., estruturas de dados para encapsular os 8 valores de controle. A interface atual MultiWii *Manipulation* é mostrada na Tabela 4.2.

Note a presença de várias estruturas de dados nessa interface. Elas foram criadas para tornar mais simples para o programador lidar com o Middleware. Por exemplo, a estrutura *control_t* guarda 8 valores inteiros: *pitch*, *roll*, *yaw*, *throttle* e *aux1* a *aux4*. *Motor_t* é composta por 4 inteiros, representando a aceleração exata sendo enviada para cada motor. Note que essa estrutura, por hora, só funciona para quadcópteros. *Imu_t* encapsula os dados da IMU, sendo composta por 9 inteiros, representando os 3 eixos do magnetômetro, do giroscópio e do acelerômetro. Por fim, *altitude_t* encapsula as informações de altitude do drone, retornados do controlador em 2 inteiros, um representando a estimativa de altitude e um a variação entre a última estimativa e a atual. Esses dois inteiros podem ser guardados na estrutura.

4.3 Testes no Frame Comercial

Aproveitando a disponibilidade do Middleware, decidiu-se testar os motores. Para isso, usou-se um frame comercial de drone disponível no laboratório, descrito em 3.1. Montou-se nele a placa CRIUS v2.5, uma placa distribuidora de energia, os ESCs e os motores. A bateria não ficou presa ao corpo, mas foi utilizada junto no teste. Primeiramente, calibrou-se os motores usando uma versão especial do MultiWii, como descrito em 2.2.2. Além disso, foi desenvolvido um *software* utilizando o Middleware para facilitar a manipulação dos valores de controle sendo enviados para o MultiWii. O software monitorava as teclas do teclado, onde algumas faziam os valores variarem de acordo com um offset pre-estabelecido, e.g. 50. Mais especificamente, "l" incrementava e "o" decrementava *throttle*, "e" incrementava e "q" decrementava *roll*, "d" incrementava e "q" decrementava *yaw* e "w" incrementava e "s" decrementava *pitch*. Os motores então foram ligados, mais uma vez mostrando o bom funcionamento da plataforma construída. Note que aqui, esse software rodava em um Laptop conectado a placa CRIUS v2.5 montada no *frame* comercial.

O próximo teste foi um teste simples de voo. Para isso, adicionou-se o computador de companhia no drone. O software utilizado no teste anterior agora estava executando no computador de companhia, um Odroid U3 montado fisicamente no drone. Para simular as teclas sendo apertadas em um teclado, o computador de companhia foi acessado via Secure Shell (SSH) pela rede Wi-Fi local. Em testes sem hélices no laboratório, funcionou muito bem. Um único problema encontrado foi o fato de que o Odroid estava com problemas intermitentes para abrir o seu serviço SSH. O drone então foi levado para o seu primeiro teste de voo. Infelizmente, alguns problemas ocorreram. O primeiro deles foi que os motores foram conectados de forma errada, o que fez que alguns girassem no sentido oposto ao que deveriam, obviamente impedindo que o drone levantasse voo. Outro problema foi a perda de conexão com o drone enquanto ele estava com os motores ligados, causado devido a problemas no serviço de SSH.

4.4 Primeiro Projeto de Frame

Antes de tentar voá-lo, fez-se um teste para examinar o funcionamento dos valores de controle, como *pitch* e *roll*. Para isso, construiu-se um estação similar a um pêndulo invertido, mas com menos liberdade de movimento, e o drone foi prendido no topo, agindo de acordo com os valores de controles enviados. Ele possuía alguns poucos centímetros para subir e poder

realizar as manobras. Com esse teste, foi possível perceber que o MultiWii estava respondendo corretamente aos valores de controle sendo passados. Aqui, o os valores de controle ainda eram alterados com o software sendo rodado no computador de companhia via SSH.

Após os testes anteriores, o drone foi levado para fazer o seu primeiro voo. Por receio de perder controle, o teste foi idealizado para que ele não subisse muito, ou seja, sem muito *throttle*. Para esse teste, o MultiWii estava configurado para usar o seu modo básico de voo, onde apenas usa giroscópio para se estabilizar. Esse modo foi o escolhido por ser recomendado para os primeiros voos, segundo (MULTIWII OFFICIAL WEBSITE, 2016). Nesse modo, ele apenas resiste a mudanças na sua orientação, mas não consegue permanecer paralelo ao solo por não fazer uso do acelerômetro. O drone levantou pouco, conforme era o desejado, mas logo de saída já ficou inclinado, deslocando-se para o lado, batendo no chão o suporte do motor. Devido a forma que caiu, o impacto quebrou esse suporte a ponto de ele se desprender do corpo e o motor ficar solto. Isso por sua vez levou à quebra a hélice. Apesar do problema, foi a primeira vez que se conseguiu fazer o drone sair do chão.

Para dar continuidade ao desenvolvimento, a hélice foi trocada e a peça quebrada foi consertada. Para o próximo teste, a ideia era ser um pouco menos conservador com o *throttle* e tentar corrigir manualmente qualquer desvio que surgisse, ainda usando o controle via SSH. No dia desse teste, ventava pouco, o que ajudou bastante o drone a manter a estabilidade sem o uso do acelerômetro. Em um primeiro momento, se colocou aceleração demais no drone, o que fez ele, em pouco tempo, atingir um altura de cerca de 6 metros. Com medo de perder controle e causar estragos, ele foi desligado, resultando em uma queda bastante forte. Dessa vez, não houve nenhum problema com o *frame*. Apesar da falha, o drone foi bastante estável no processo de subida. Ainda nesse dia, se repetiu o teste com valores mais conservadores de *throttle*. Apesar de não se ter conseguido estabilidade total, foi possível fazer o drone subir e descer.

4.5 Segundo Projeto de Frame

O software de controle foi evoluído. Como SSH estava se mostrando instável, decidiu-se por fazer o controle sem utilizá-lo, implementando um outro serviço para fazer a comunicação entre o Laptop e o computador de companhia. Para isso, estabeleceu-se um protocolo: o Laptop enviaria os 8 valores de controle em uma certa ordem via socket TCP, sendo interpretados nessa mesma ordem no computador de companhia. TCP foi escolhido para facilitar o desenvolvi-

mento, já que não seria necessário lidar com perdas de pacotes ao usar esse protocolo. A Figura 4.1 ilustra a arquitetura desse sistema.

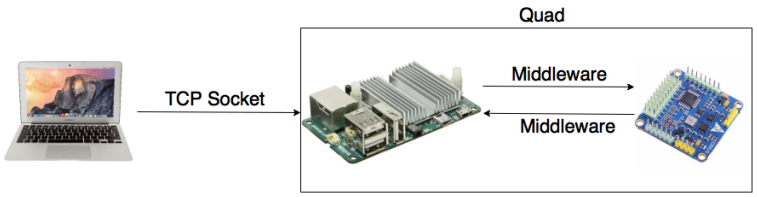

Figura 4.1 – Arquitetura do sistema usando um Laptop se comunicando com o Middleware via TCP

O uso desse novo protocolo de comunicação permitiu usar diferentes tecnologias no lado do Laptop. Um equipamento indisponível no laboratório era um rádio controle. Para simulá-lo, uma ideia foi a utilização de um controle de Xbox One®, conforme mostrado na Figura 4.2. Apesar de que o objetivo fosse de construir um drone autônomo, usar um controle nessa fase de desenvolvimento era relevante por ser uma interface simples que permitisse variar os valores de controle. Ressaltando, não era objetivo final usá-lo. Como o protocolo era baseado em TCP, bastava fazer uma aplicação no Laptop que interpretasse os dados do controle, convertesse-os para valores de controle e os enviasse para o computador de companhia. Isso foi então implementado usando uma ferramenta que permitiu realizar todas essas funções, além de permitir a criação de uma Graphical User Interface (GUI) de forma rápida. Unity (UNITY3D, 2016) é um motor gráfico idealizada para criação de jogos. Dito isso, como muitos jogos fazem uso de controle, ela integra nativamente essa tecnologia. Ainda por essa característica, ela possui várias ferramentas que auxiliam o programador a criar interfaces gráficas. Com relação a rede, Unity permite que o programador escreva *scripts* C# rodando sobre .NET, o que disponibiliza sockets TCP.

Em termos de algoritmo, o software funciona da seguinte forma. Os botões direcionais são monitorados. Cada um deles recebeu um modo de voo. Sempre que um deles é pressionado, os valores de controle *Aux* são alterados para refletir que o drone esteja no modo voo selecionado. Vale ressaltar que os modos de voo devem estar configurados de forma compatível com a forma que a interface está configurada. O botão menu, quando pressionado, faz o *software* tentar se conectador o computador de companhia. O funcionamento dos *sticks* analógicos é um pouco diferente. Cada *stick* possui dois valores, um representando o deslocamento no eixo horizontal e um no eixo vertical. Eles são representados como dois valores em ponto flutuante

Figura 4.2 – Xbox One® Controller (from (XBOX ONE CONTROLLER PICTURE, 2016))

Tabela 4.3 – Correspondência entre Valores de Controle e Sticks

Stick	Eixo	Valor de Controle
Esquerda	Vertical	Throttle
	Horizontal	Roll
Direita	Vertical	Pitch
	Horizontal	Yaw

no intervalo $[-1, 1]$, sendo 0 quando *stick* está no centro do eixo e os extremos quando ele está posicionado nos extremos. Como no total têm-se 4 valores, cada um deles é associado com um dos valores de controle *pitch*, *roll*, *yaw* e *throttle*. O cálculo do valor de controle a partir do valor do *stick* analógico é representado pela Equação 4.1.

$$ControlValue_i = 1500 + Axis_i * 500 \tag{4.1}$$

Onde $ControlValue_i$ representa o valor de controle $i \in \{pitch, roll, yaw, throttle\}$ e $Axis_i$ o eixo correspondente ao valor de controle i. A distribuição dos valores de controle e os eixos é mostrada na Tabela 4.3.

Para ajudar na visualização, a GUI, mostrada na Figura 4.3, possuía 8 caixas de texto, uma para cada valor de controle. Elas são atualizadas em tempo-real baseado diretamente nos valores de controles sendo transmitidos. Todas essas caixas são vermelhas enquanto o *software* não esta conectado com o Odroid e verde uma vez que a conexão é iniciada. Em adição a isso, existem mais dois campos de texto em que o usuário deve entrar com o IP do computador de companhia e a porta em que o serviço está rodando. Um conjunto de caixas de texto mostra os modos de voo disponíveis e o selecionado. Novamente, essas caixas de texto são dependentes

da configuração do MultiWii. Modos de voo podem ser alterados nas setas direcionais tanto do controle quanto do Laptop.

Figura 4.3 – GUI do Controle de Xbox One®

O teste usando essas ferramentas foi então realizado. O drone se saiu bem em algumas tentativas, mas ainda assim, ocorreram algumas quedas. Controlá-lo com controle é uma tarefa complexa para inexperientes. Muitas vezes, na iminência de perder o controle, o piloto não soube como reagir de forma coerente. Dessa forma, percebeu-se que havia muito espaço para falha humana. No intuito de remover essa variável, optou-se por desenvolver um piloto automático simples.

4.6 Melhorias no Segundo Frame: Piloto Automático

4.6.1 Primeira Implementação

Para o piloto automático, o teste seguiu a mesma ideia: fazer o drone alçar voo tentando encontrar um forma de equilibrá-lo no ar. Para isso, o controle de Xbox One foi removido e a interface do Laptop sofreu algumas pequenas mudanças. Fora isso, o teste usou a mesma infraestrutura básica. O software executado no computador de companhia foi o mesmo.

A GUI foi modificada para fornecer ao usuário uma tabela em que ele poderia entrar um *script* para que o drone seguisse. Essa nova versão da GUI é mostrada na Figura 4.4. Na esquerda, há um campo de texto acima de um botão. Nesse campo, o usuário deve inserir

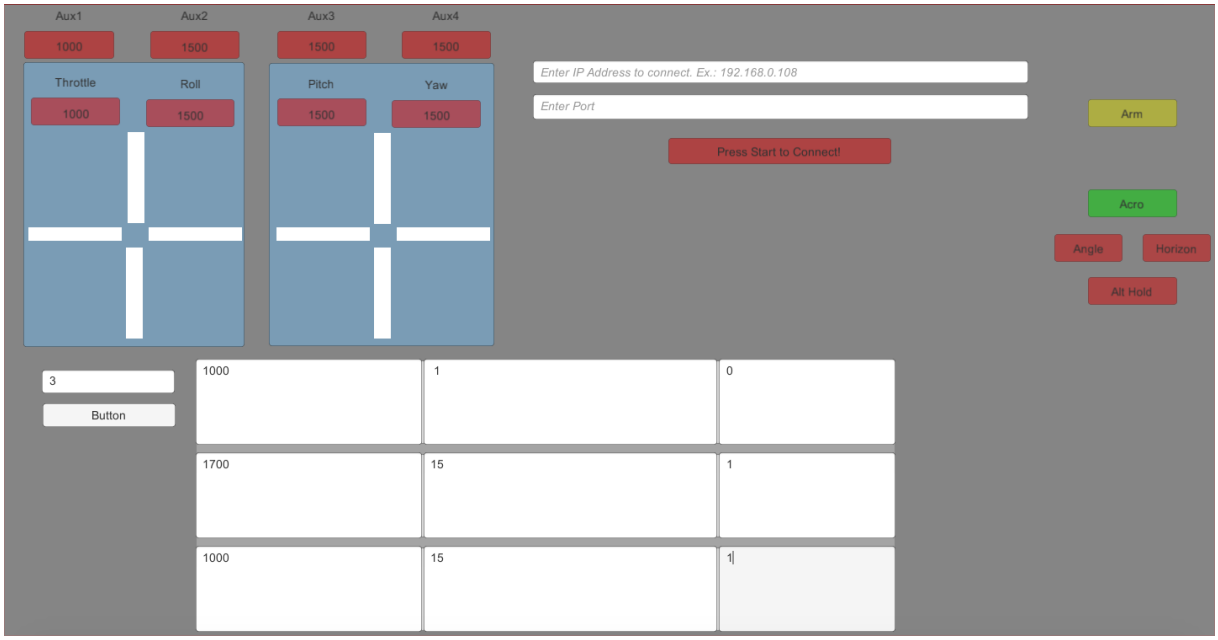

Figura 4.4 – GUI com tabela de entrada de programa de *throttle*

o número de linhas, $NumRows$, que deseja e então clicar no botão. Isso então gerará uma tabela o número de linhas inserido e três colunas. Cada linha corresponde a uma passo do algoritmo de entrada. A primeira coluna, $TargetThrottle$, corresponde ao valor de *throttle* que se deseja que o drone possua ao fim do passo. Esse valor será alcançado interpolando o valor de *throttle* no início do passo e $TargetThrottle$ em intervalos discretos. O número de intervalos, $NumberOfReps$, é dado pela segunda coluna. Durante cada intervalo, o *throttle* será constante. No fim do intervalo, o *throttle* será incrementado um valor $dThrottle$ dado pela Equação 4.2.

$$dThrottle = \frac{CurrentThrottle - TargetThrottle}{NumberOfReps} \tag{4.2}$$

A última coluna contém o valor $RateOfReps$. Ele representa a duração de cada intervalo do passo corrente. É possível então calcular o tempo do passo, $Time_{step[i]}$, através da Equação 4.3.

$$Time_{step[i]} = NumberOfReps * RateOfReps \tag{4.3}$$

i representa o i-ésimo passo, tal que:

$$1 \leqslant i \leqslant NumRows \tag{4.4}$$

A linha no topo corresponde ao passo em execução. Ao seu término, a linha correspondente é deletada da tabela.

Por questões de segurança, foi implementado um botão do pânico. Se, por alguma razão, o quadcóptero começasse a perder o controle, colocando em risco a si mesmo, elementos próximos, como janelas de carros, ou ainda mais importante, pessoas, era possível abortar a sequência automática e desligar o quadcóptero.

A evolução no teste usando essa técnica foi bastante considerável. O piloto automático também funcionou muito bem. Foi possível fazer testes bastante encorajadores, sendo possível fazer o quadcóptero levantar voo e pousar muito bem. Em termos de estabilidade no ar, ele pecou um pouco. Ainda era necessário comunicação constante entra o Laptop e o computador companhia, seguindo o protocolo descrito em 4.5. Devido a distância do roteador, essa comunicação tornava-se lenta, causando problemas com a comunicação e, em consequência, perdas de potência nos motores.

4.6.2 Resolvendo Problemas da Primeira Implementação e Controle de Altitude

Para resolver o problema da necessidade de um link de comunicação constante, o protocolo sofreu uma mudança. Ao invés de os passos serem executados no Laptop e os valores repassados para o computador de companhia, passou-se a transmitir os valores do algoritmo para o computador de companhia e ele ficou encarregado de fazer os cálculos localmente. O drone não mais perdeu potência nos testes, mas também não foi encontrado um valor constante de aceleração que fizesse o drone se manter estável.

Outro problema existente era o limite de 9 passos imposto pela interface em Unity, muitas vezes insuficiente para os testes desejados. Para resolver esse problema, fez-se um interpretador de arquivos no software do computador de companhia para ler os valores que seriam passados pela aplicação em Unity. O arquivo era composto de um inteiro i, indicando a quantidade de passos, seguido de i linhas com três valores, representado $TargetThrottle, NumberOfReps$ e $RateOfReps$, respectivamente, separados por um espaço em branco. O software do computador de companhia era então iniciado via SSH e recebia como parâmetro o arquivo de entrada.

Além dessas modificações, foi adicionada a possibilidade de se acionar o modo de Voo "Altitude Hold" do MultiWii durante o voo, pois começou-se a suspeitar que não seria possível manter a estabilidade com um valor constante de *throttle*. Para se fazer isso, adicionou-se uma *thread* extra no programa para monitorar a entrada do teclado. A *thread* principal e essa

thread dividiam um *mutex*. Quando a *thread* que monitora o teclado detecta que a tecla "H" havia sido pressionada, ela travava esse *mutex*, impedindo que o piloto automático continue a executar. Isso fazia com que o *throttle* ficasse travado no seu último valor, até ser desbloqueado. A *thread* bloqueante apenas liberava o *mutex* quando o usuário entrar com "H" novamente. Esse comportamento é interessante, pois segundo (MULTIWII OFFICIAL WEBSITE, 2016), o *throttle* não deve ser alterado uma vez que se inicie o modo "Altitude Hold".

O drone então foi levado para o seu último teste. O objetivo era avaliar a eficiência do modo "Altitude Hold". Para o teste, o drone seria levantado até atingir um bom nível de estabilidade usando "Horizon Mode" (paralelo ao chão) e o operador então mudaria para "Altitude Hold". Após varias baterias de testes, obtiveram-se resultados encorajadores. Apesar de certas variações, o modelo foi capaz de manter sua altitude com bastante estabilidade. Entretanto, o drone ainda apresentou um certo "*drift*" lateral. Esse comportamento era esperado, uma vez que "Altitude Hold" não corrige deslocamentos laterais que podem ser causados por diversos fatores, e.g. vento. Para antigir o nível inicial de estabilidade, para o modelo usado, usou-se um *throttle* em torno de 1620.

5 CUSTOS

Um grande objetivo do projeto consistia em manter o custo de fabricação do drone baixo. Para isso, buscou-se usar COTS importados da China. Os componentes usados e seus preços podem ser encontrados na Tabela 5.1. Note que o custo apresentado não leva em conta o imposto de importação. Esse imposto, segundo (Subsecretaria de Aduana e Relações Internacionais, 2016), é calculado da seguinte forma:

$$Imposto = CustoBruto * 0,60 \tag{5.1}$$

Logo, temos que, somando o preço bruto mais o preço com o imposto, o drone tem um custo total de R$1.464,27. O preço das ferramentas necessárias, e.g. impressora 3D, foi desconsiderado pois, dependendo da quantidade de drones produzidos, o preço delas pode se tornar irrisório se divido pelo número de unidades produzidas. Ou ainda, as peças podem ser fabricadas em empresas especializadas em impressão 3D a um custo mais elevado, mas ainda baixo.

Tabela 5.1 – Custo de material por drone, desconsiderando taxa de importação

Componente	Custo Unitário	Qtde. necessária	Custo total
Motor A2212 Brushless	R$61,19	4	R$244,76
ESC 20A	R$27,17	4	R$108,68
Hélice	R$9,12	4	R$36,49
CRIUS 2.5	R$58,45	1	R$58,45
Odroid U3	R$235,27	1	R$235,27
Bateria para Odroid U3	R$100,49	1	R$100,49
Bateria LiPo	R$88,84	1	R$88,84
Placa distribuidora de energia	R$13,87	1	R$13,87
1 Kg de Filamento plástico ABS	R$82,64	200g	R$16,52
Barra Alumínio	R$10,00 (metro)	1,20m	R$12,00
			R$915,37

Apesar de, em um primeiro momento, o valor parecer um pouco alto, é necessário uma análise mais a fundo da quantidade de tecnologia embarcada no modelo. Ele possui uma API de alto nível que permite facilmente a programação. Além disso, possui um grande poder computacional devido ao Odroid U3, sendo capaz inclusive de realizar processamento de imagem *onboard* caso adicionada uma câmera. Um drone similar da empresa DJI, o DJI Phantom 3, tem valor inicial de R$3.438,79 (DJI OFFICIAL WEBSITE, 2016). Obviamente, esse drone

tem consigo toda uma carga de anos de desenvolvimento. Ele possui um sistema de piloto automático, melhor estabilidade, processamento de imagem, aplicativo *mobile*, entre outras funcionalidades ainda não presentes no drone desenvolvido nesse trabalho. Entretanto, a maioria delas consiste em desenvolvimento de software, algo atingível com a infraestrutura desenvolvida. Processamento de imagem, por exemplo, pode ser realizado adicionando uma câmera do modelo desenvolvido e instalando uma biblioteca de processamento de imagem no computador de companhia. Além disso, esse drone não oferece uma API para programação *onboard*, apenas uma API para desenvolver aplicativos *mobile*, uma desvantagem em relação ao do trabalho. Um drone com API para processamento *onboard* da DJI pode chegar a cifras bastante altas, como R$22.734,62 (DJI Matrice 100).

Outro drone que vale ser comparado a infraestrutura desenvolvida é o Parrot AR.Drone 2.0. O drone possui um aplicativo mobile para navegação, câmera e oferece uma API para programação. Entretanto, não há informação a respeito do processador usado no modelo, mas é incapaz de realizar alto processamento. O seu preço gira em torno de R$1.838,69 segundo o site da Parrot. Note que, apesar do preço similar ao do desenvolvido, a Parrot fabrica em grandes quantidades, o que diminui bastante o custo dos componentes. Ou seja, se o drone desenvolvido nesse trabalho fosse produzido na escala do Parrot, seria mais barato. Claro, sempre deve ser levado em conta que, no drone do trabalho, se fala em custo de produção, sendo que no Parrot já está incluído o lucro. Outra diferença é o fato de o poder de processamento do drone desenvolvido nesse trabalho ser bastante superior ao do AR.Drone 2.0. Normalmente, processamento de imagem o AR.Drone é feito através de *stream* de vídeo para uma estação base que o processa e envia comandos. Na infraestrutura desenvolvida, isso teoricamente pode ser realizado sem a necessidade de uma estação base, que poderia ser um fator limitante uma vez que seria necessário um link constante de comunicação.

Assim, fica evidente que o drone desenvolvido é uma solução viável, apesar do mercado ser bastante concorrido. O drone apresenta como principal característica o seu baixo custo e alto poder de processamento em relação às alternativas. Apesar de não discutido, o drone desenvolvido também abre uma gama de possibilidade de customizações. O hardware pode ser constantemente atualizado. Inclusive, o poder computacional pode ser aumentado comprando-se computadores de companhia mais potentes a critério do usuário.

6 CONCLUSÕES

O trabalho de construção de um drone não é uma tarefa simples. O simples fato de fazer os componentes funcionar em conjunto é um processo trabalhoso, com uma curva de aprendizado acentuada mas altamente recompensante. Os resultados desse trabalho são bastante encorajadores, abrindo um leque de oportunidades de trabalhos futuros.

Este trabalho pode servir como um ponto de entrada para algum usuário que queira entender o funcionamento básico de um drone. Ele apresenta um dicionário de termos englobando os principais partes que compõem um drone, em especial quadcópteros.

Em termos de *frame*, chegou-se a uma solução barata e robusta. Conforme citado em 3.4, as avarias sofridas em testes que exigiram bastante do modelo foram pequenas. Além disso o modelo se comportou muito bem, com pouca vibração e poucas deformações. Outro ponto importante do *frame* é a sua facilidade de montagem e manutenção. Ele pode ser montado em cerca de uma hora e, no caso de alguma peça ser danificada, se for de plástico, pode ser impressa novamente, ou se for de alumínio, a peça pode ser rapidamente fabricada, necessitando somente de um corte e de algumas perfurações em uma barra de alumínio. O custo final do *frame* é bastante baixo também, custando menos de R$ 30,00. Por ser um *frame* na sua maior parte modelado, ele pode facilmente ser adaptados a diferentes necessidades, e.g. adição de câmeras.

Em termos de *software*, têm-se uma API que permite facilmente escrever rotinas de piloto automático por meio de um Middleware. Ele abstrai vários detalhes de baixo nível, fazendo com que o programador foque somente nas tarefas mais importantes do projeto do algoritmo de voo autônomo. Além disso, é bastante simples usar a API em conjunto com diversas outras tecnologias, como bibliotecas de processamento de imagem, por exemplo. A plataforma é basicamente um código-fonte na linguagem C rodando sobre uma sistema operacional Linux. Com isso, as possibilidades de criação ficam limitadas a criatividade do programador.

Os custos para a construção dessa plataforma também são bastante baixos. Por se fazer uso de COTS, as peças são bastante acessíveis no que tange preço e disponibilidade. Um pequeno incômodo apenas é a demora para receber os componentes importados, podendo muitas vezes chegar a dois ou três meses.

Conforme dito anteriormente, a infraestrutura construída abre espaço para uma diversidade de trabalhos futuros. Um sugestão seria fazer uma análise mais aprofundada da interação física do modelo com o ambiente. Por exemplo, poderia ser usada ferramentas de simulação,

como o Simulink do *software* MATLAB da companhia MathWorks (MATLAB, SITE OFICIAL, 2016). Com ele, seria possível trabalhar com simulações sobre um modelo computacional físico do drone. Isso facilitaria um pouco os testes, pois não seria necessário de fato levantar voo com o modelo, potencialmente colocando ele em risco. Apesar de robusto e fácil de montar, quedas fortes podem sim danificar bastante o modelo. Outro ponto importante dessa modelagem seria visualizar melhor as forças sendo exercidas pelo modelo.

Outro trabalho muito interessante de ser feito, que inclusive poderia ser feito utilizando simulações, seria conduzir uma investigação dos diferentes modelos de controle. O MultiWii faz uso de controle PID, mas poderia ser testado o uso de Linear–quadratic regulator (LQR) (ARGENTIM et al., 2013), por exemplo. Isso poderia inclusive levar a optar-se por outro modelo de controle mais interessante.

Uma possibilidade seria dar continuidade a implementação de um piloto automático. Um sugestão seria implementar uma navegação via *waypoints*, onde o drone recebe um conjunto de coordenadas de GPS compondo uma rota e navega entre esses *waypoint*. A funcionalidade não seria difícil de implementar na plataforma criada. Um módulo GPS poderia ser conectado ao computador de companhia. Esse então faria o cálculo de quais valores de controle deveriam ser enviados para placa controladora de modo a seguir a rota.

Uma vez com maior controle sobre o funcionamento de drones, pode-se começar a investigar a utilização de uma rede de drones em constante comunicação. Um exemplo dessa configuração é apresentada em (BRUST; STRIMBU, 2015), onde os autores trazem um modelo para "enxames" de drones. Eles estão em constante comunicação para que, em conjunto, resolvam um dado problema. Como a plataforma desenvolvida nesse trabalho pode integrar diversas tecnologias, poderia ser adicionado um rádio em cada drone. Definindo um protocolo básico, os drones teriam constante comunicação e poderiam decidir em conjunto como agir.

Conforme apresentado, as possibilidades são muitas. Esse trabalho cumpriu o seu objetivo de criar uma plataforma de baixo custo, simples de usar e bastante flexível. Apesar de ainda não ser perfeita, ele já esta dando bons resultados. Espera-se que no futuro ela seja expandida e traga resultados ainda mais animadores.

REFERÊNCIAS

123D Design Official Website. Disponível em <http://www.123dapp.com/design>. Acesso em 3 de dez. de 2016.

ANG, K. H.; CHONG, G.; LI, Y. PID control system analysis, design, and technology. **IEEE Transactions on Control Systems Technology**, [S.l.], v.13, n.4, p.559–576, Jul. 2005.

ARDUINO official website. Disponível em <https://www.arduino.cc>. Acesso em 18 de maio de 2016.

ARDUPILOT official website. Disponível em <http://ardupilot.org/ardupilot/index.html>. em Acessed on 18 de maio de 2016.

ARGENTIM, L. M. et al. PID, LQR and LQR-PID on a quadcopter platform. In: INTERNATIONAL CONFERENCE ON INFORMATICS, ELECTRONICS AND VISION (ICIEV), 2013. **Anais...** [S.l.: s.n.], 2013. p.1–6.

BRUSHLESS motor picture. Disponível em <https://goo.gl/dTUZCq>. Acesso em 18 de maio de 2016.

BRUST, M. R.; STRIMBU, B. M. A networked swarm model for UAV deployment in the assessment of forest environments. In: IEEE TENTH INTERNATIONAL CONFERENCE ON INTELLIGENT SENSORS, SENSOR NETWORKS AND INFORMATION PROCESSING (ISSNIP), 2015. **Anais...** [S.l.: s.n.], 2015. p.1–6.

CRIUS v2.0 picture. Disponível em <http://www.quadkopters.com/wp-content/uploads/2012/12/crius-mwc-1.jpg>. Acesso em 18 de maio de 2016.

DJI official website. Disponível em <http://www.dji.com>. Acesso em 18 de maio de 2016.

DRONEKIT Official Website. Disponível em <http://dronekit.io>. Acesso em 19 de out. de 2016.

ELECTRONIC Speed Controller picture. Disponível em <http://goo.gl/fqKObh>. Acesso em 18 de maio de 2016.

HYREL Official Website. Disponível em <http://www.hyrel3d.com>. Acesso em 3 de dez. de 2016.

Jet pilotless drones collect data on radiological hazards in atomic clouds. **Electrical Engineering**, [S.l.], v.72, n.6, p.567–570, jun. 1953.

LI-PO battery picture. Disponível em <http://goo.gl/p4lLRN>. Acesso em 18 de maio de 2016.

LUGO, J. J.; ZEIL, A. Framework for autonomous onboard navigation with the AR.Drone. In: INTERNATIONAL CONFERENCE ON UNMANNED AIRCRAFT SYSTEMS (ICUAS), 2013. **Anais...** [S.l.: s.n.], 2013. p.575–583.

MAHONY, R.; KUMAR, V.; CORKE, P. Multirotor Aerial Vehicles: modeling, estimation, and control of quadrotor. **IEEE Robotics Automation Magazine**, [S.l.], v.19, n.3, p.20–32, Set. 2012.

MATLAB, site oficial. Disponível em <https://www.mathworks.com/products/matlab.html>. Acesso em 03 de dez. de 2016.

MAVLINK Official Website. Disponível em <http://qgroundcontrol.org/mavlink/start>. Acesso em 19 de out. de 2016.

MESHMIXER Official Website. Disponível em <http://www.meshmixer.com>. Acesso em 3 de dez. de 2016.

MULTIWII official website. Disponível em <http://www.multiwii.com>. Acesso em 18 de maio de 2016.

RADIO controller and receiver picture. Disponível em <http://www.leetshop.com/Photos/RC-RCS-p148858-1.jpg>. Acesso em 18 de maio de 2016.

Subsecretaria de Aduana e Relações Internacionais. **Manual de Importações**. Disponível em <goo.gl/5my2Uc>. Acesso em 1 de dez. de 2016.

THINGVERSE Official Website. Disponível em <http://www.thingiverse.com>. Acesso em 3 de dez. de 2016.

UNITY3D. Disponível em <http://unity3d.com/>. Acesso em 18 de maio de 2016.

XBOX One controller picture. Disponível em <https://goo.gl/YKbzrc>. Acesso em 18 de maio de 2016.

ANEXOS

ANEXO A – Middleware Desenvolvido

Este anexo contém o código fonte do Middleware desenvolvido para programação de um Drone baseado em MultiWii, apresentado em 4.2. O capítulo contém maiores informações sobre a funcionalidade de cada módulo do Middleware.

A.1 MultiWiiNetwork

A.1.1 MultiWiiNetwork.h

```
/******************
*MultiWiiNetwork.h
******************/

#ifndef ____MULTIWIINETWORK_____
#define ____MULTIWIINETWORK_____


#include <stdint.h>
#include <stdlib.h>

#define MSP_SET_MOTOR 214
#define MSP_MOTOR 104
#define MSP_RC 105
#define MSP_SET_RAW_RC 200
#define MSP_RAW_IMU 102
#define MSP_ALTITUDE 109



//Basic Parameters to configure the MultiWii serial communication: Check
    your MultiWii Build for these pieces information
//Other than the g_porta, all the values used are the default ones from
    MultiWii

//Parametros basicos para configurar a comunicacao serial no MultiWii: Os
    valores usados, tirando g_porta, sao os
//valores padroes usados pelo MultiWii.


typedef struct {

    long unsigned int frame_size;
    unsigned char received_frame[500];

} received_frame_t;



//Initialise the connection
//Inicia a conexao
```

```
int multiwii_init_serial_port(void);

//Send the given data frame using the serial connection.
//Envia o quadro passado por parametro por porta serial.
int multiwii_send_data(unsigned char* data_frame, unsigned char data_size);

//Receives data from the serial port. Expected size is the maximum number
    of bytes it is supposed to read
//Recebe dados pela porta serial. Expected_size eh o numero maximo de bytes
     que devem ser lidos
received_frame_t multiwii_get_data(int expected_size);


//Close the connection
//Fecha a conexao
void multiwii_close_serial_port();



#endif // ____MULTIWIINETWORK_____
```

A.1.2 MultiWiiNetwork.cpp

```
/*********************
* MultiWiiNetwork.cpp
**********************/

#include "MultiWiiProtocol.h"
#include "MultiWiiNetwork.h"
#include <stdio.h>
#define FRAME_SIZE(x) (x+6)

#include <errno.h>
#include <iostream>
#include <cstdlib>
#include <stdint.h>
#include <time.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <pthread.h>

using namespace std;

#define BAUDRATE B115200                //B230400
//#define PERIOD 20000 // 20ms --> 50Hz
#define DEVICE "/dev/ttyUSB0"
#define FALSE 0
#define TRUE 1
```

```
// FILE DESCRIPTOR DA SERIAL
int fd;
uint16_t rc_signals[8] = { 1234 };
uint8_t rc_bytes[16] = { 0 };



void initPort()
{
    //Baseado em https://www.cmrr.umn.edu/~strupp/serial.html
    fd = open(DEVICE, O_RDWR | O_NOCTTY | O_NDELAY);
    if (fd == -1)
    {
        /*
         * Could not open the port.
         */

        perror("open_port: Unable to open /dev/ttyf1 - ");
        exit(1);
    }
    else
        fcntl(fd, F_SETFL, 0);


    struct termios options;

    /*
     * Get the current options for the port...
     */

    bzero(&options, sizeof(options));

    /*
     * Set the baud rates to 1500...
     */

    cfsetispeed(&options, B115200);
    cfsetospeed(&options, B115200);

    /*
     * Enable the receiver and set local mode...
     */

    options.c_cflag |= (CLOCAL | CREAD);
    options.c_cflag &= ~CSIZE; /* Mask the character size bits */
    options.c_cflag |= CS8;    /* Select 8 data bits */

    options.c_cflag &= ~PARENB;
    options.c_cflag &= ~CSTOPB;
    options.c_cflag &= ~CSIZE;
    options.c_cflag |= CS8;


    options.c_cc[VTIME] = 1;  // timeout after .1s that isn't working
    options.c_cc[VMIN] = 0;


    /*
```

```
     * Set the new options for the port...
     */

    tcsetattr(fd, TCSANOW, &options);

}


//Initialise the connection
//Inicia a conexao
int multiwii_init_serial_port(void)
{

    initPort();
    return 1;
}

//Send the given data frame using the serial connection.
//Envia o quadro passado por parametro por porta serial.
int multiwii_send_data(unsigned char* data_frame, unsigned char data_size)
{

    int res = (int)write(fd, data_frame, data_size);
    //tcdrain(fd);
    if(res < 0)
    {
        return 0;
    }

    return 1;
}

//Receives data from the serial port. Expected size is the maximum number
   of bytes it is supposed to read
//Recebe dados pela porta serial. Expected_size eh o numero maximo de bytes
    que devem ser lidos
received_frame_t multiwii_get_data(int expected_size)
{

    received_frame_t received_frame;
    received_frame.frame_size = read(fd, received_frame.received_frame,
       expected_size);
    if( (int) received_frame.frame_size < expected_size )
    {
        printf("Error on read! %s - %d %d\n", strerror(errno), (int)
           received_frame.frame_size, (int)expected_size);
        tcflush(fd, TCIFLUSH);
    }

    return received_frame;

}

//Close the connection
//Fecha a conexao
void multiwii_close_serial_port()
{
    close(fd);
```

```
}
```

A.2 MultiWiiProtocol

A.2.1 MultiWiiProtocol.h

```
#ifndef ____MULTIWIIPROTOCOL_____
#define ____MULTIWIIPROTOCOL_____


//MSP refers to the MultiWii Serial protocol. Please go to http://www.
    multiwii.com/wiki/index.php?title=Multiwii_Serial_Protocol
//for more information

//MSP se refere ao protocolo de comunicacao da plataforma MultiWii. Mais
    informacoes em
//http://www.multiwii.com/wiki/index.php?title=Multiwii_Serial_Protocol


//Macro to calculate the frame size when having x bytes
//Macro para calculo do tamanho do quadro quando enviamos x bytes
#define MULTIWII_FRAME_SIZE(x) (x+6)



//Generates a frame according to the MSP format. DATA SHOULD BE FREED
//Gera um quadro no formato aceito pelo MSP. CHAMAR FREE SOBRE O RETORNO
void get_msp(unsigned char opcode, unsigned char* data, unsigned char
    n_bytes, unsigned char* msp_message);




#endif
```

A.2.2 MultiWiiProtocol.cpp

```
#include "MultiWiiProtocol.h"
#include <stdlib.h>
#include <stdio.h>

//Generates a frame according to the MSP format. DATA SHOULD BE FREED
//Gera um quadro no formato aceito pelo MSP. CHAMAR FREE SOBRE O RETORNO

//Each [] is an octet
//Format -> ['$']['M']['<'][n_bytes][opcode]n_bytes*[data] [checksum]

void get_msp(unsigned char opcode, unsigned char* data, unsigned char
    n_bytes, unsigned char* msp_message)
{
    //Numero minimo de bytes em um frame
    int cont = 6;
    unsigned char checksum = 0;
```

```
    //Insere o cabecalho
    msp_message[0] = '$';
    msp_message[1] = 'M';
    msp_message[2] = '<';

    //Insere tamanho da mensagem
    msp_message[3] = (n_bytes&0xff);
    checksum ^= (n_bytes&0xff);

    //Inserw opcode
    msp_message[4] = opcode;
    checksum ^= opcode;
    //insere dados
    int i = 0;
    for(i = 0; i < n_bytes; i++)
    {
        msp_message[i+5] = (data[i]);
        checksum ^= data[i];
        cont++;
    }
    // insere o checksum
    msp_message[i+5] = checksum;

}
```

A.3 MultiWiiManipulation

A.3.1 MultiWiiManipulation.h

```
#ifndef ____MULTIWIIMANIPULATION_____
#define ____MULTIWIIMANIPULATION_____

#include <stdint.h>

//Basic data structure to encapsulate IMU (magnetometer, accelerometer and
    gyroscope) data
//Estrututa de dados para armazenar dados de IMU(acelerometro, giroscopio e
    magnetometro)
typedef struct {
    int gyro_x;
    int gyro_y;
    int gyro_z;
    int acc_x;
    int acc_y;
    int acc_z;
    int mag_x;
    int mag_y;
    int mag_z;
} imu_t;


//Basic data structure used to encapsulate control data
//Estrututa de dados para armazenar dados de controle(acelerometro,
    giroscopio e magnetometro)
typedef struct {
```

```
    uint16_t throttle;
    uint16_t pitch;
    uint16_t roll;
    uint16_t yaw;
    uint16_t aux1;
    uint16_t aux2;
    uint16_t aux3;
    uint16_t aux4;

} control_t;


//Data Structure to encapsulate the altitude information
//Estrutura de dados que encapsula os dados de altitude
typedef struct {

    int32_t est_alt;
    int16_t vario;

} altitude_t;


//Data structure that encapsulates the thrust values of each motor of a
    quadcopter
//Estrutura de dados que encapsula os valores de aceleracao dos motores de
    um quadcoptero
typedef struct{

    uint16_t front_right;
    uint16_t front_left;
    uint16_t rear_right;
    uint16_t rear_left;

} motor_t;



//Initialises the MultiWii manipulation. SHOULD ALWAYS BE CALLED BEFORE
    CALLING ANY OF THE OTHER FUNCTIONS
//Inicializa a manipulacao do MultiWii. DEVE SER CHAMADA ANTES DE CHAMAR
    QUALQUER OUTRA FUNCAO
int multiwii_init_manipulation(void);

//Get imu data
//Obtem dados dos sensores IMU
imu_t multiwii_get_imu(void);

//Get Control data
//Obtem dados de controle
control_t multiwii_get_control(void);

//Set control data on the FC
void multiwii_set_control(control_t control_data);

//Get altitude data
//Obtem informacoes de altitude
altitude_t multiwii_get_altitude_data(void);
```

```
//Get motor data
//Obtem informacoes dos motores
motor_t multiwii_get_motor_data(void);

//Closes the Middleware
//Finaliza o Middleware
int multiwii_stop_manipulation(void);

#endif
```

A.3.2 MultiWiiManipulation.cpp

```
#include "MultiWiiNetwork.h"
#include "MultiWiiManipulation.h"
#include "MultiWiiProtocol.h"

#include <stdio.h>

//Monta um inteiro de 16 bits a partir de 2 de 8, apesar de os parametros
   pedirem 16
uint16_t mount16(uint16_t a, uint16_t b){
    return (b<<8) + a;
}

//Monta um inteiro de 32 bits a partir de dois de 16
int32_t mount32(uint16_t a, uint16_t b){

    return (((int)b) << 16) + a;

}

//Transforma um inteiro para a sua notacao em complemento de 2
int complemento2(int a){
    return (a > 32768? a-65536 : a);
}


//Initialises the MultiWii manipulation. SHOULD ALWAYS BE CALLED BEFORE
   CALLING ANY OF THE OTHER FUNCTIONS
//Inicializa a manipulacao do MultiWii. DEVE SER CHAMADA ANTES DE CHAMAR
   QUALQUER OUTRA FUNCAO
int multiwii_init_manipulation(void){
    int success = multiwii_init_serial_port();
    return success;

}

//Get imu data
//Obtem dados dos sensores IMU
imu_t multiwii_get_imu(void){

    unsigned char msp_message[50];
    do{

        get_msp(MSP_RAW_IMU, NULL, 0, msp_message);

    }while(!multiwii_send_data(msp_message, MULTIWII_FRAME_SIZE(0)));
```

```
    //Espera-se receber 24 bytes de resposta
    int expected_receive_size = 24;
    received_frame_t received_frame = multiwii_get_data(
        expected_receive_size);



    //TODO: Validation of received frame -> Checksum!!!!!
    if(received_frame.frame_size < (uint16_t)expected_receive_size){
         return multiwii_get_imu();
    }

    imu_t ret;

    //Interpret the received frame
    ret.acc_x = complemento2(mount16(received_frame.received_frame[5],
        received_frame.received_frame[6]));
    ret.acc_y = complemento2(mount16(received_frame.received_frame[7],
        received_frame.received_frame[8]));
    ret.acc_z = complemento2(mount16(received_frame.received_frame[9],
        received_frame.received_frame[10]));
    ret.gyro_x = complemento2(mount16(received_frame.received_frame[11],
        received_frame.received_frame[12]))/8;
    ret.gyro_y = complemento2(mount16(received_frame.received_frame[13],
        received_frame.received_frame[14]))/8;
    ret.gyro_z = complemento2(mount16(received_frame.received_frame[15],
        received_frame.received_frame[16]))/8;
    ret.mag_x = complemento2(mount16(received_frame.received_frame[17],
        received_frame.received_frame[18]))/3;
    ret.mag_y = complemento2(mount16(received_frame.received_frame[19],
        received_frame.received_frame[20]))/3;
    ret.mag_z = complemento2(mount16(received_frame.received_frame[21],
        received_frame.received_frame[22]))/3;

    return ret;

}

//Get Control data
//Obtem dados de controle
control_t multiwii_get_control(void){

    //Makes the request for RC values
    unsigned char msp_message[50];
    do{

        get_msp(MSP_RC, NULL, 0, msp_message);

    }while(!multiwii_send_data(msp_message, MULTIWII_FRAME_SIZE(0)));


    int expected_receive_size = 22;
    received_frame_t received_frame = multiwii_get_data(
        expected_receive_size); //23?
```

```
    if(received_frame.frame_size < (uint16_t)expected_receive_size){
        return multiwii_get_control();
    }


    control_t ret;
    //Interpreta o frame recebido
    ret.roll = complemento2(mount16(received_frame.received_frame[5],
       received_frame.received_frame[6]));
    ret.pitch = complemento2(mount16(received_frame.received_frame[7],
       received_frame.received_frame[8]));
    ret.yaw = complemento2(mount16(received_frame.received_frame[9],
       received_frame.received_frame[10]));
    ret.throttle = complemento2(mount16(received_frame.received_frame[11],
       received_frame.received_frame[12]));
    ret.aux1 = complemento2(mount16(received_frame.received_frame[13],
       received_frame.received_frame[14]));
    ret.aux2 = complemento2(mount16(received_frame.received_frame[15],
       received_frame.received_frame[16]));
    ret.aux3 = complemento2(mount16(received_frame.received_frame[17],
       received_frame.received_frame[18]));
    ret.aux4 = complemento2(mount16(received_frame.received_frame[19],
       received_frame.received_frame[20]));

    return ret;
}

//Set control data on the FC
//Envia dados de Controle para o FC
void multiwii_set_control(control_t control_data){

    uint16_t serializedData[8];

    serializedData[0] = control_data.roll;
    serializedData[1] = control_data.pitch;
    serializedData[2] = control_data.yaw;
    serializedData[3] = control_data.throttle;
    serializedData[4] = control_data.aux1;
    serializedData[5] = control_data.aux2;
    serializedData[6] = control_data.aux3;
    serializedData[7] = control_data.aux4;

    //Converts to the order Multiwii wants to receive the 16 bit integers
    // Converte o inteiro para a forma que o MultiWii ira le-lo
    unsigned char sendBytes[16];
    for(int i = 0 ; i < 16 ; i+=2){
        sendBytes[i] = serializedData[i/2] &(0xff);
        sendBytes[i+1] = (serializedData[i/2]>>8) &(0xff);

    }
    unsigned char msp_message[50];

    get_msp(MSP_SET_RAW_RC, sendBytes, 16, msp_message);

    multiwii_send_data(msp_message, MULTIWII_FRAME_SIZE(16));
}
```

```
//Get altitude data
//Obtem informacoes de altitude
altitude_t multiwii_get_altitude_data(void){


    //Send request for imu data
    unsigned char msp_message[50];
    do{

        get_msp(MSP_ALTITUDE, NULL, 0, msp_message);

    }while(!multiwii_send_data(msp_message, MULTIWII_FRAME_SIZE(0)));

    int expected_receive_size = 12;


    received_frame_t received_frame = multiwii_get_data(
       expected_receive_size);

    //We are expecting 25 bytes of data (Multiwii sends a 0 before every
       frame)
    //TODO: Validation of received frame -> Checksum!!!!!
    if(received_frame.frame_size < (uint16_t)expected_receive_size){
        return multiwii_get_altitude_data();
    }


    altitude_t ret;

    //Interpret the received frame
    //Interpreta o frame recebido
    ret.est_alt = mount32(mount16(received_frame.received_frame[5],
       received_frame.received_frame[6]), mount16(received_frame.
       received_frame[7], received_frame.received_frame[8]));

    return ret;

}

//Get motor data
//Obtem informacoes dos motores
motor_t multiwii_get_motor_data(void){

    unsigned char msp_message[50];

    do{
        get_msp(MSP_MOTOR, NULL, 0, msp_message);
    }while(!multiwii_send_data(msp_message, MULTIWII_FRAME_SIZE(0)));

    int expected_receive_size = 22;
    received_frame_t received_frame = multiwii_get_data(
       expected_receive_size);


    if(received_frame.frame_size < (uint16_t)expected_receive_size){
        return multiwii_get_motor_data();
    }
```

```
    motor_t ret;
    //Interpreta o frame recebido
    ret.rear_right = mount16(received_frame.received_frame[5],
        received_frame.received_frame[6]);
    ret.front_right = mount16(received_frame.received_frame[7],
        received_frame.received_frame[8]);
    ret.rear_left  = mount16(received_frame.received_frame[9],
        received_frame.received_frame[10]);
    ret.front_left = mount16(received_frame.received_frame[11],
        received_frame.received_frame[12]);

    return ret;
}

//Closes the Middleware
//Finaliza o Middleware
int multiwii_stop_manipulation(void){

    multiwii_close_serial_port();
    return 1;

}
```

A.4 Exemplo de Uso

A seguir, é apresentado o código do último teste do drone, em que se testa o modo de voo Altitude Hold. Ele lê um programa de throttle de um arquivo e calcula os valores de *throttle* a serem enviados. É um bom exemplo de como se deve usar o Middleware. Note que, no código, ainda constam algumas funções para interação com a rede, não mais usada nesse teste. Elas foram mantidas aqui para caso no futuro, elas venham a ser usadas novamente.

```
//tutoriais comunicacao porta serial
//http://stackoverflow.com/questions/6068655/what-are-the-functions-i-need-
    use-for-serial-communication-in-vc


#include "MultiWiiManipulation.h"
#include <stdio.h>

#define MAX_THROTTLE 2000//1850
#define MAX 2000
#define MIN 1000
#define NOMINAL 1500

#define IMU_MODE 0
#define RC_MODE 1

#define TIME_SOBE_LENTAMENTE 5


#include <termios.h>
#include <fcntl.h>

#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <signal.h>
#include <sys/ioctl.h>
#include <sys/time.h>

#include <pthread.h>


void programa_sobe_lentamente();
void *multiwii_writer(void *args);
void *monitor_abort(void *args);
void *monitor_keyboard(void *args);
void trim_left(void *args);
void trim_right(void *args);
void trim_back(void *args);
void trim_front(void *args);


int sockfd;
int newsockfd;
int port;


typedef struct{
    control_t *send;
    control_t *send2;

}ponteiros_send;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t alt_hold_mutex = PTHREAD_MUTEX_INITIALIZER;


//Inicia um socket
int init_socket(int argc, char *argv[])
{
    int portno = port, clilen;
    struct sockaddr_in serv_addr, cli_addr;

    printf( "using port #%d\n", portno );

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        perror( "ERROR opening socket");
    bzero((char *) &serv_addr, sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons( portno );
    if (bind(sockfd, (struct sockaddr *) &serv_addr,
```

```
            sizeof(serv_addr)) < 0)
        perror(  "ERROR on binding" );
    listen(sockfd,5);
    clilen = sizeof(cli_addr);

    //--- infinite wait on a connection ---
    while ( 1 )
    {
        printf( "waiting for new client...\n" );
        if ( ( newsockfd = accept( sockfd, (struct sockaddr *) &cli_addr, (
            socklen_t*) &clilen) ) < 0 );
        else
            break;
    }
    return 0;
}



//Le um float da rede
float read_float(){

    float ret;
    int n;

    n = (int)recv(newsockfd,&ret,sizeof(ret),0);

    if (n <= 0)
    {
        printf("Erro lendo do socket!\n");
        exit(1);
    }
    return ret;


}

//Le um inteiro da rede
int read_int(){

    int ret;
    int n;


    n = (int)recv(newsockfd,&ret,sizeof(ret),0);


    if (n <= 0)
    {
        printf("Erro lendo do socket!\n");
        exit(1);
    }
    return ret;


}
```

```
//Desliga o quad. Ligado ao sinal ctrl+C
void closeEverything(int signal)
{

    control_t send;

    send.throttle = 1000;
    send.roll = 1500;
    send.pitch = 1500;
    send.yaw = 1000;
    send.aux1 = 1000;
    send.aux2 = 1500;
    send.aux3 = 1500;
    send.aux4 = 1500;


    for(int i = 0; i < 100 ; i++){
        multiwii_set_control(send);
    }

    multiwii_stop_manipulation();
    close(newsockfd);
    close(sockfd);
    printf("Bye\n");
    exit(1);

}




int main(int argc, char **argv)
{

    //Verifica se foi passado um arquivo de entrado
    if(argc < 2 )
    {
        printf("Uso ./leitor arquivo_entrada\n");
        exit(1);
    }

    printf("\n ----------------------------------------------------\n\n");
    printf("\nPrograma para controle de quadcoptero Multiwii por porta
       Serial");
    printf("\nDesenvolvido por Matheus Garay Trindade\n\n");

    //Liga o sinal ctrl+c com a funcao closeEverything
    signal(SIGINT, closeEverything);


    FILE *arquivo_entrada = fopen(argv[1], "r");

    //Verifica se o arquivo de entrada existe
    if(arquivo_entrada == NULL){
        printf("Arquivo %s nao encontrado\n", argv[1]);
        exit(1);
    }
    else{
```

```
        printf("Lendo %s\n", argv[1]);

    }


    /*******************/
    multiwii_init_manipulation();
    /*******************/


    //Duas variaveis de controle, uma para ser escrita na main
    //e a outra para a thread que interage com a placa escrever no FC/
    //Feito dessa forma para evitar que a thread escreve no FC valores
    // intermediarios ao se fazer um update dos mesmos
    control_t send, send2;
    send.throttle = 1500;
    send.pitch = 1500;
    send.roll = 1500;
    send.yaw = 1500;
    send.aux1 = 1500;
    send.aux2 = 1500;
    send.aux3 = 2000;
    send.aux4 = 1500;

    send2 = send;

    ponteiros_send ptr;
    ptr.send = &send;
    ptr.send2 = &send2;

    //Instancia thread escritora
    pthread_t thread, thread2;
    pthread_create(&thread, NULL, multiwii_writer, &send2);

    while(1)
    {
        //Fornece a opcao de fazer algums trimmings, caso necessario
        char option;
        fflush(stdin);

        printf("\n\nEntre com a opcao desejada\n");
        printf("Trim front: w\n");
        printf("Trim back: s\n");
        printf("Trim left: a\n");
        printf("Trim right: d\n");

        printf("\nEsperando tecla\n");
        scanf(" %c", &option);

        switch (option) {
            case 'w':
                trim_front(&ptr);
                continue;
                break;
            case 'a':
                trim_left(&ptr);
                continue;
                break;
```

```
            case 's':
                trim_back(&ptr);
                continue;
                break;
            case 'd':
                trim_right(&ptr);
                continue;
            case '\n':
                continue;
                break;
        }

        //Comeca a monitorar o teclado pela tecla h, para entrar em
           altitude hold
        pthread_create(&thread2, NULL, monitor_keyboard, &ptr);

        int number_of_iterations;

        fscanf(arquivo_entrada, "%d\n", &number_of_iterations);
        printf("Programa: ");

        printf("%d iteracoes\n", number_of_iterations);

        int target_throttle[number_of_iterations];
        int number_of_reps[number_of_iterations];
        float reps_duration[number_of_iterations];

        //Faz o parsing do arquivo
        for (int i = 0 ; i < number_of_iterations ; i++){

            fscanf(arquivo_entrada, "%d %d %f", target_throttle + i,
               number_of_reps + i, reps_duration + i);
            printf("%d, %d, %f\n",target_throttle[i], number_of_reps[i],
               reps_duration[i]);
        }

        //Horizon mode
        send.aux1 = 2000;
        send.aux2 = 2000;


        //Executa a rota
        for(int i = 0 ; i < number_of_iterations ; i++){
            float delta_throttle = (target_throttle[i] - send.throttle)/((
               float)number_of_reps[i]);
            printf("Iteracao %d\n", i);
            for(int j = 0 ; j < number_of_reps[i] ; j++){

                //Trava caso esteja em altidude hold
                pthread_mutex_lock(&alt_hold_mutex);
                send.throttle = (int) (send.throttle + delta_throttle);
                //Trava para que o escritor nao envie meios valores para o
                   FC
                pthread_mutex_lock(&mutex);
                send2 = send;
                pthread_mutex_unlock(&mutex);
                printf("\tRep: %d -> throttle = %d\n", j, send.throttle);
                pthread_mutex_unlock(&alt_hold_mutex);
```

```
                usleep(1000000 * reps_duration[i]);

            }

        }

        //Desliga as helices
        send.throttle = 1000;
        send.aux1 = 1500;
        pthread_mutex_lock(&mutex);
        send2 = send;
        pthread_mutex_unlock(&mutex);
        printf("Desligando\n");
        usleep(1000000);
        break;
    }

    multiwii_stop_manipulation();

    return 0;
}


// Continuamente escreve no controlador
void *multiwii_writer(void *args)
{
    control_t *send = (control_t *) args;
    control_t send2;

    while (true) {

        pthread_mutex_lock(&mutex);
        //Le o dado compartilhado com a outra thread em exclusao mutua
        send2 = *send;
        pthread_mutex_unlock(&mutex);


        multiwii_set_control(send2);
        usleep(2800);



        // Le os dados de controle
        //        control_t ctrl =
        //        multiwii_get_control();
        //        printf("Throttle -> %4d\t", ctrl.throttle);
        //        printf("Pitch    -> %4d\t", ctrl.pitch);
        //        printf("Roll     -> %4d\t", ctrl.roll);
        //        printf("Yaw      -> %4d\t", ctrl.yaw);
        //        printf("Aux4      -> %4d\n", send2.aux4);
        //        printf("\n");

    }
    return NULL;
}


/* this function is run by the second thread */
```

```
void *monitor_keyboard(void *args)
{
    ponteiros_send *ptr = (ponteiros_send*) args;

    control_t *send = ptr->send;
    control_t *send2 = ptr->send2;

    while (true) {
        char c;

        scanf(" %c", &c);

        if(c == 'h'){
            //Trava a thread main por meio do mutex. Entra em
            // Alt_hold mantendo os valores de controle
            pthread_mutex_lock(&alt_hold_mutex);
            printf("Entrando em Altitude Hold");
            send->aux2 = 1500;
            send->aux4 = 2000;
            pthread_mutex_lock(&mutex);
            *send2 = *send;
            pthread_mutex_unlock(&mutex);

            //Fica esperando novamente por h
            char c2;
            do{
                scanf("%c", &c2);
            }while(c2 != 'h');
            send->aux2 = 2000;
            send->aux4 = 1500;
            //Sai de alt hold
            pthread_mutex_lock(&mutex);
            *send2 = *send;
            pthread_mutex_unlock(&mutex);

            //Libera a thread main
            pthread_mutex_unlock(&alt_hold_mutex);
        }
    }


    return NULL;
}



//Faz trimming para a esquerda usando combos de valores de controle
void trim_left(void *args)
{
    ponteiros_send *ptr = (ponteiros_send*) args;

    control_t *send = ptr->send;
    control_t *send2 = ptr->send2;

    printf("Trimming left\n");

    send->throttle = 2000;
    send->roll = 1000;
```

```
    pthread_mutex_lock(&mutex);
    *send2 = *send;
    pthread_mutex_unlock(&mutex);

    printf("Hit a key to finish trim\n");
    char temp;
    scanf(" %c", &temp);
    fflush(stdin);

    send->throttle = 1000;
    send->roll = 1500;
    pthread_mutex_lock(&mutex);
    *send2 = *send;
    pthread_mutex_unlock(&mutex);



    printf("Trimmed left\n");

}



//Faz trimming para a direita usando combos de valores de controle
void trim_right(void *args)
{
    ponteiros_send *ptr = (ponteiros_send*) args;

    control_t *send = ptr->send;
    control_t *send2 = ptr->send2;

    printf("Trimming right\n");

    send->throttle = 2000;
    send->roll = 2000;
    pthread_mutex_lock(&mutex);
    *send2 = *send;
    pthread_mutex_unlock(&mutex);

    printf("Hit a key to finish trim\n");
    char temp;
    scanf(" %c", &temp);
    fflush(stdin);

    send->throttle = 1000;
    send->roll = 1500;
    pthread_mutex_lock(&mutex);
    *send2 = *send;
    pthread_mutex_unlock(&mutex);


    printf("Trimmed right\n");

}

//Faz trimming para a frente usando combos de valores de controle
void trim_front(void *args)
{
```

```
    ponteiros_send *ptr = (ponteiros_send*) args;

    control_t *send = ptr->send;
    control_t *send2 = ptr->send2;

    printf("Trimming front\n");

    send->throttle = 2000;
    send->pitch = 2000;
    pthread_mutex_lock(&mutex);
    *send2 = *send;
    pthread_mutex_unlock(&mutex);

    printf("Hit a key to finish trim\n");
    char temp;
    scanf(" %c", &temp);
    fflush(stdin);

    send->throttle = 2000;
    send->pitch = 1500;
    pthread_mutex_lock(&mutex);
    *send2 = *send;
    pthread_mutex_unlock(&mutex);


    printf("Trimmed front\n");

}


//Faz trimming para a tras usando combos de valores de controle
void trim_back(void *args)
{
    ponteiros_send *ptr = (ponteiros_send*) args;

    control_t *send = ptr->send;
    control_t *send2 = ptr->send2;



    send->throttle = 2000;
    send->pitch = 1000;
    pthread_mutex_lock(&mutex);
    *send2 = *send;
    pthread_mutex_unlock(&mutex);

    printf("Hit a key to finish trim\n");
    char temp;
    scanf(" %c", &temp);
    fflush(stdin);

    send->throttle = 2000;
    send->pitch = 1500;
    pthread_mutex_lock(&mutex);
    *send2 = *send;
    pthread_mutex_unlock(&mutex);
```

```
    printf("Trimmed back");

}
```

Um exemplo de arquivo de entrada é mostrado na sequência. O primeiro número i é o número de passos. As seguintes i linhas correspondem aos passos, cada um com $TargetThrottle$, $NumberOfReps$ e $RateOfReps$ do algoritmo descrito em 4.6.1.

```
2
1000 0 1
1500 1 1
```