

Desenvolvimento de veículo aéreo não tripulado para análise de áreas de risco

José Roberto de Souza¹

1. Metrocamp – Grupo IbmeC, Engenharia de Computação, zehortigoza@gmail.com

Resumo – O objetivo desse trabalho é o desenvolvimento de um VANT (veículo aéreo não tripulado) para realizar análise de áreas de risco de maneira segura e efetiva. Para realizar essa tarefa o tipo de aeromodelo escolhido foi o quadricóptero, por este tipo oferecer uma boa estabilidade, manobrabilidade e capacidade de carga. Seu controle e estabilização são feitos utilizando controladores PID (*Proportional-Integral-Derivative*), que atuam na velocidade de rotação dos motores com base na orientação do quadricóptero fornecida por um acelerômetro e um giroscópio. Seguindo esta estratégia foi possível obter um aeromodelo de baixo custo, capaz de realizar análise de áreas de risco de maneira segura e simplificada.

Palavras-chave: quadricóptero, VANT, áreas de risco, PID, acelerômetro, giroscópio.

1. INTRODUÇÃO

Nas últimas décadas o uso de veículos aéreos não tripulados cresceu, devido aos avanços no desenvolvimento de processadores, microcontroladores e sensores de baixo custo e consumo energético. Outro grande motivo para esse crescimento é o uso desses veículos para tarefas que possam oferecer riscos à vida, tarefas como: coletar informações em terreno inimigo, construções em colapso e áreas com contaminação radioativa ou biológica [1].

Existem vários tipos de aeromodelos, mas o que se sobressai entre eles para uso não militar e pelo seu

custo benefício são os quadricópteros (Figura 1), veículos com pouso e decolagem vertical, altamente estáveis e manobráveis. Este artigo aborda o desenvolvimento de um quadricóptero de baixo custo para análise de áreas de riscos.



Fig. 1 – quadricóptero Phantom da DJI[2]
Fonte: <http://www.dji.com/>

2. METODOLOGIA

2.1. Dinâmica de voo do quadricóptero

O veículo consiste de uma *frame* (estrutura), com quatro braços simetricamente posicionados formando um quadrado e na extremidade de cada braço há um conjunto de motor e hélice.

Todo o conjunto de motores e hélices são responsáveis por fornecer sustentação, porém se todos os motores rodarem na mesma direção o torque produzido pelos motores faria o quadricóptero rotacionar sobre seu eixo. Para corrigir esse problema dois motores rodam no sentido horário e dois motores rodam no sentido anti-horário, anulando os torques. Dessa correção surge outro problema, dois motores forneceriam sustentação positiva e dois forneceriam sustentação negativa puxando o quadricóptero para o

chão. Para resolver esse outro problema são usados um par de hélice no sentido horário e outro par no sentido anti-horário, desse modo todos os motores fornecem sustentação positiva e o torque é anulado [3].

Existem dois formatos de *frame*, os em formato de X e os em formato de +, como visto na Figura 2. Os dois formatos trabalham quase que igualmente, só se diferenciam na equação que calcula a velocidade de rotação de cada motor a fim de obter os movimentos desejados.

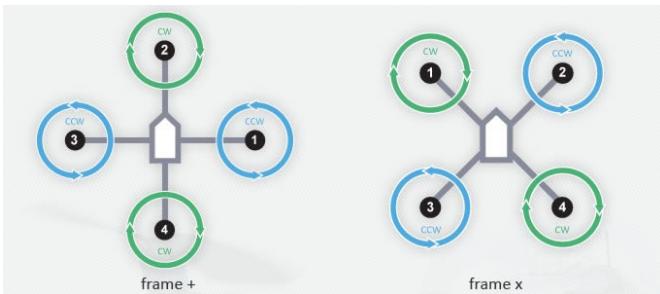


Fig. 2 – Tipos de *frames*
Fonte: <http://www.diydrones.com/>

Para esse projeto foi escolhido um *frame* em X, a Figura 3 ilustra como executar alguns movimentos com um quadricóptero em X.

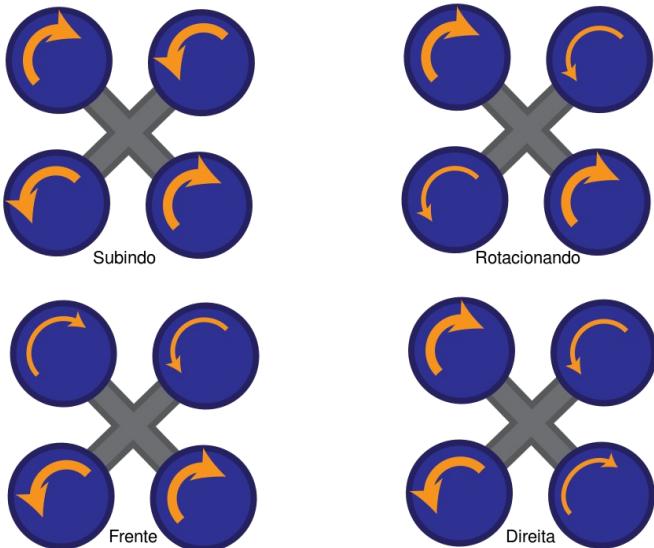


Fig. 3 – Movimentos quadricóptero
Setas finas indicam menor rotação, enquanto setas grossas indicam maior rotação
Fonte: <http://www.wikipedia.org/>

2.2 Hardware

Como descrito anteriormente são necessários quatro

hélices, duas no sentido horário e duas no sentido anti-horário, também quatro motores e o *frame*.

Existem dois tipos de projetos de quadricópteros o *slow flight* e *fast flight*. No *slow* como o nome sugere, esse tipo se movimenta mais lentamente, possui uma maior estabilização e maior capacidade de carga, enquanto o *fast* se movimenta mais rapidamente podendo executar manobras como piruetas na vertical e horizontal porém tem menor estabilização e capacidade de carga [3].

Visando o objetivo de analisar de áreas de risco o tipo *slow flight* possui as melhores características. Nesse tipo de quadricóptero os motores têm um menor RPM (rotações por minuto) porém as hélices possuem um maior comprimento e passo.

2.2.1 Motores

São os responsáveis por rotacionar as hélices a fim de obter sustentação, neste projeto foram utilizados motores de 1000KV ou seja, 1000 rpm/V este com capacidade de levantar 1Kg com hélices de 10 à 8 polegadas. Como o quadricóptero tem 4 motores a capacidade total é de 4Kg, porém é recomendado que o quadricóptero não pese mais que a metade da capacidade dos motores, ou seja, 2Kg.

Para um melhor desempenho e redução da complexidade da placa de controle fez-se a opção por motores sem escovas.

Motores com escovas são motores mais simples, onde um ímã é fixado nas extremidades do motor e quando aplicado uma corrente na bobina localizada no centro do motor este gera um campo magnético, a atração e repulsão dos campos magnéticos faz com que o motor gire. Aumentando a corrente aplicada na bobina o campo magnético aumenta, aumentando também a velocidade de rotação.

Já nos motores sem escovas o processo de rotação é um pouco mais complexo, em seu centro existe um ímã e várias bobinas estão localizadas nas extremidades, para rotacionar o eixo do motor a corrente deve ser aplicada em uma das bobinas rotacionando alguns graus, depois é aplicado corrente na bobina seguinte completando mais alguns graus da rotação, repetindo esse processo até acabar a rotação. A Figura 4 ilustra a disposição dos ímãs e bobinas em ambos os motores.

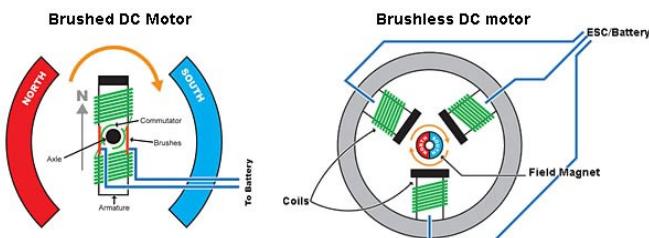


Fig. 4 – Comparação motor com escova e motor sem escova
Fonte: <http://www.thinkrc.com/>

Deste modo os motores sem escovas tem uma maior força, controle de velocidade de rotação e vida útil, porém para utilizá-los é necessário um circuito de controle de velocidade comumente chamado de ESC (*Electronic Speed Controller*) [4].

2.2.2 Hélice

Hélice faz o trabalho de dar a sustentação ao quadricóptero, quando o motor a rotaciona seu formato faz com que o ar seja puxado para dentro dela, fazendo com que a hélice obtenha sustentação “escalando” sobre esse ar que passa por ela [5].

Nesse projeto hélices de 10 polegadas de comprimento e 4,5 polegadas de passo foram utilizadas, além de serem duas hélices no sentido horário e duas no sentido anti-horário a fim de anular o torque dos motores.

2.2.3 ESC

O ESC (Figura 5) controla os acionamentos das bobinas dos motores, ajustando a tensão e o tempo entre os acionamentos a fim de chegar à velocidade desejada.

É informado ao ESC a velocidade desejada através de uma interface PWM (*Pulse-Width Modulation*), esta facilmente encontrada nas bibliotecas e no hardware dos microcontroladores.

Uma característica importante na escolha do ESC é que este deve suportar uma corrente maior ou igual à máxima corrente que o motor consome. Nesse projeto os motores consomem no máximo 21 amperes, então ESCs de 25 amperes foram adquiridos.



Fig. 5 – ESC de 10amperes
Fonte: <http://www.sussex-model-centre.co.uk/>

2.2.4 Frame (estrutura)

Dando continuidade aos componentes do projeto, um *frame* (Figura 6) no formato de X foi utilizado para unir os componentes, este com braços de polímeros de nylon e pratos de fibra de vidro, com 45 cm de ponta a ponta e 8 cm de altura sem contar os pés de pouso.



Fig. 6 – Frame em formato de X
A esfera indica qual é a frente do quadricóptero
Fonte: <http://www.hobbyking.com/>

2.2.5 Microcontrolador

Para interfacear com o ESC é necessário um microcontrolador, nesse projeto foi utilizado um *kit* de desenvolvimento da Texas Instruments chamado Stellaris® LM4F120 LaunchPad Evaluation Kit [6] (Figura 7). Este é equipado com um ARM® Cortex™-M4F [7], um microcontrolador de 32bits utilizando a arquitetura de Harvard e com unidade de ponto flutuante. Um ótimo microcontrolador com um bom poder computacional, se levando em conta seu consumo energético e preço.

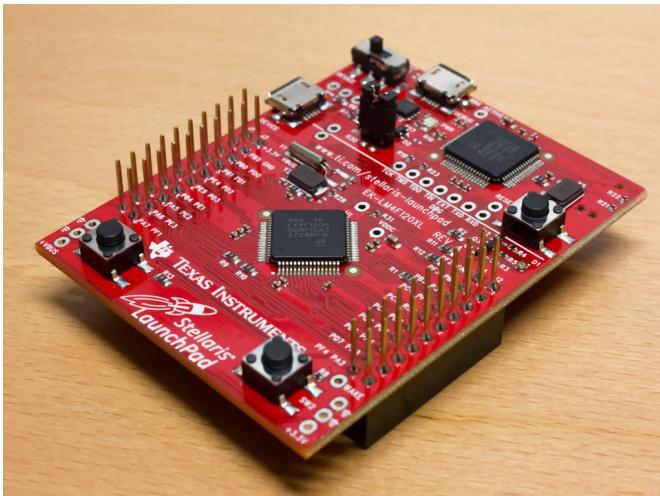


Fig. 7 – Kit de desenvolvimento Stellaris
Fonte: <http://www.kaibader.de/>

2.2.6 Acelerômetro e giroscópio

Outro componente extremamente necessário é o conjunto de acelerômetro e giroscópio, que fornecerão informações de força gravitacional e velocidade angular respectivamente, aplicadas nos três eixos do quadricóptero. Com essas informações o microcontrolador fará cálculos e enviará para os ESCs a velocidade que cada motor deve rotacionar a fim de manter o quadricóptero estabilizado.

Nesse projeto o MPU-6050™ [8] (Figura 8) da InvenSense foi utilizado, um circuito integrado onde existem um acelerômetro e um giroscópio conectados no mesmo barramento I²C.



Fig. 8 – MPU6050 numa placa de circuito impresso
Fonte: <http://www.amazon.com/>

2.2.7 Transceptor

Outro componente necessário é um rádio realizar a comunicação com o controle. Visando o custo, um transceptor utilizando o padrão IEEE 802.11 [9] foi escolhido, pois neste modo foi possível utilizar um *smartphone* como controle, não necessitando então a construção do hardware para o controle. Esse é um protocolo de comunicação com alta velocidade porém com curto alcance, no entanto o alcance pode ser facilmente estendido se distribuídos alguns *Access points*, aumentando o alcance da rede sem fio. O módulo RN-XV WiFly [10] da Roving Networks foi escolhido pela sua robustez e opções de configuração. Este se comunica com o microcontrolador utilizando o barramento UART (*Universal Asynchronous Receiver and Transmitter*).

2.2.8 Placa de circuito

E para conectar os quatro componentes anteriores uma placa de circuito é necessária, este fará as conexões entre o microcontrolador, MPU6050, transceptor e os ESCs.

Para prototipação geralmente uma *protoboard* é usada, até que tenha um esquema de conexões

funcional. Porém as conexões na *protoboard* correm o risco de se desfazerem, este problema é ainda agravado com o vento gerado pelas hélices do quadricóptero.

Por isso o correto seria desenhar uma PCB (*Printed Circuit Board*) e soldar os componentes na placa, mas essa é uma alternativa muito cara para pedidos com baixo número de placas, foram solicitados alguns orçamentos de uma placa 30 x 30 cm com 2 faces e o valor saiu por volta de 150 reais.

Outra opção a PCB é a placa perfurada, uma placa cheia de ilhas. Os componentes são soldados nas ilhas e conectados entre sim com fios ou com canais de estanho, se não muito distante as conexões. Essa placa tem um custo bem pequeno e fornece a liberdade de fazer alterações e correções o que uma PCB não fornece.

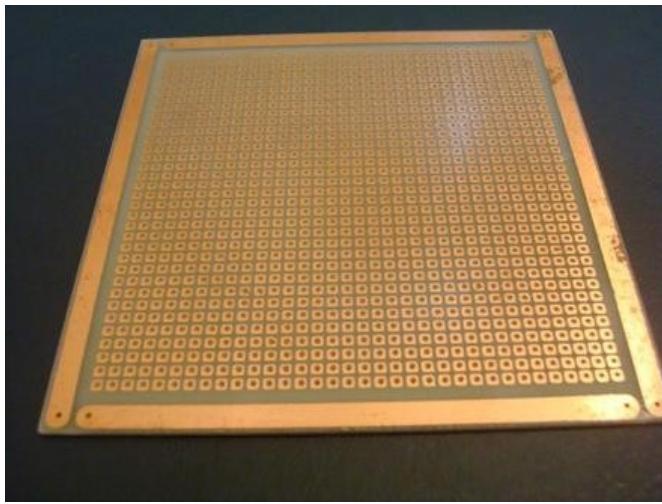


Fig. 9 – Placa perfurada

Fonte: <http://quebarato.com.br/>

Para fazer as conexões entre os componentes o *datasheet* é essencial. Esse documento descreve o que cada pino do componente pode fazer, como ativar e utilizar protocolos de comunicação implementados em hardware, qual é a tensão que o componente requer e etc.

Como dito anteriormente o MPU6050 fornece os dados do acelerômetro e giroscópio pelo protocolo I²C (*Inter-Integrated Circuit*), o transceptor utiliza o

protocolo UART e para controle dos ESCs é utilizando um sinal PWM (*Pulse-width modulation*), todos esses são suportados pelo microcontrolador do Stellaris.

Tabela 1 – Conexões dos componentes

Stellaris PB0	WiFly Dout (pino 2)
Stellaris PB1	WiFly Din (pino 3)
Stellaris +3.3V	WiFly VCC (pino 1)
Stellaris GND	WiFly GND (pino 10)
Stellaris PA6	MPU6050 SCL
Stellaris PA7	MPU6050 SDA
Stellaris +3.3V	MPU6050 VCC
Stellaris GND	MPU6050 GND
Stellaris GND	MPU6050 AD0
Stellaris PA5	MPU6050 <i>Interrupt</i>
Stellaris PB2	ESC 1
Stellaris PB7	ESC 2
Stellaris PB6	ESC 3
Stellaris PB4	ESC 4

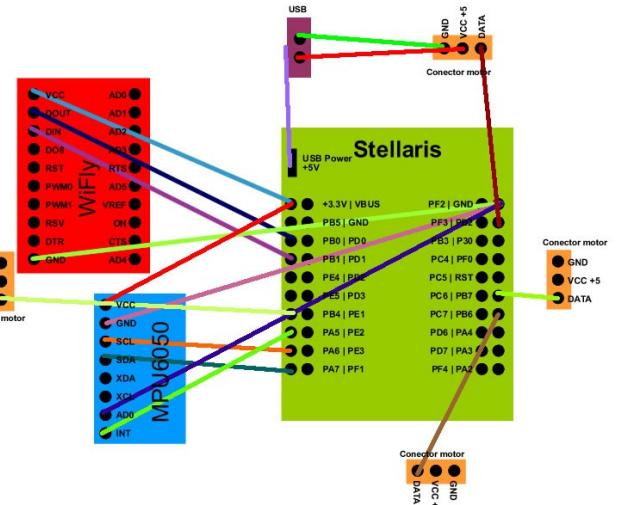


Fig. 10 – Diagrama de conexões

2.2.9 Controle

Como dito acima também é necessário um controle para enviar e receber informações e comandos para o quadricóptero. Visando redução de custo, nesse projeto um *smartphone* foi utilizado como controle. Esse *smartphone* possui o sistema operacional Android, abrindo uma grande gama de *smartphones* compatíveis com o projeto.

2.2.10 Bateria

E por último e não menos importante a bateria do quadricóptero. Esta alimentará os ESCs, que transformarão os 12v da bateria na tensão compatível com a velocidade de rotação enviada pelo microcontrolador. Os ESCs para aeromodelos possuem um recurso chamado BEC (*Battery Eliminator Circuit*) que é um circuito eliminador de bateria, este tem o objetivo de eliminar a necessidade de uma bateria secundária no aeromodelo para alimentar o microcontrolador e outros componentes que precisam de uma tensão bem menor, geralmente de 5v. Deste modo esta única bateria fornecerá tensão para todo o quadricóptero.

Para esse projeto uma bateria de 2,2 amperes de polímero de íons e lítio [11] foi escolhida, esta tem a capacidade de fornecer 30 amperes em um curto período de tempo, cerca de 15 minutos.

2.2.11 Sumário de componentes

Tabela 2 – Principais componentes

Peça	Modelo
Frame	Hobbyking SK450 Glass Fiber Quadcopter Frame 450 mm
ESC	Hobbyking SS Series 25-30A ESC
Motor sem escova	D2830-11 1000kv Brushless Motor
Hélice	10x4.5 SF Props 2pc Standard Rotation/2 pc RH Rotation
Bateria	Turnigy 2200mAh 3S 20C Lipo Pack
Giroscópio e acelerômetro	GY-521 6DOF MPU6050 Module
Transceptor Wi-Fi	RN-XV WiFi Module-Wire Antenna

2.3 Software

A Texas Instruments fornece uma IDE (*Integrated Development Environment*) para desenvolvimento com o Stellaris e toda a sua família de microcontroladores ARM Cortex-M.

Este chamado Code Composer Studio [12], com versões para Windows e Linux, porém a versão Linux não possui suporte ao JTAG (*Joint Test Action Group*) [13] incluso no *kit* de desenvolvimento do Stellaris.

A fim utilizar somente ferramentas *open source* para o desenvolvimento do projeto, optou-se por não utilizar o Code Composer Studio no Windows. Buscando por alternativas, foi encontrado um artigo [14] explicando como utilizar o GCC (*GNU Compiler Collection*) [15] e um *toolchain* [16] para compilar código para a arquitetura do ARM Cortex-M4F. Para edição de código, optou-se pelo Eclipse CDT [17]. Deste modo foi possível obter todo um ambiente de desenvolvimento utilizando somente software *open source*.

A linguagem escolhida foi C, por sua velocidade e robustez. Uma opção seria programar em Assembly mas utilizando essa linguagem aumentaria a complexidade do código e diminuiria a velocidade do desenvolvimento por outro lado o desempenho seria melhor. Outra opção seria utilizar C++, mas nesse caso, seria de maneira análoga, utilizar uma “bazuca pra matar uma formiga”. Teria um código mais estruturado e elegante, mas, por outro lado, perderia desempenho.

Outro componente de *software* utilizado no projeto foi a biblioteca StellarisWare® [18], fornecida pela Texas Instruments para facilitar uso dos recursos internos implementados em *hardware* como I²C, SPI, UART, PWM, CAN e outros nos microcontroladores ARM Cortex-M. A biblioteca é distribuída gratuitamente quando utilizada nos microcontroladores da Texas Instruments.

Nos tópicos a seguir serão fornecidos mais detalhes de como foi feita a interface com os componentes explicando também o funcionamento dos protocolos. Mas antes para fornecer uma visão geral do projeto e

como cada componente do software se integra, a Figura 11 apresenta o diagrama de módulos do projeto.

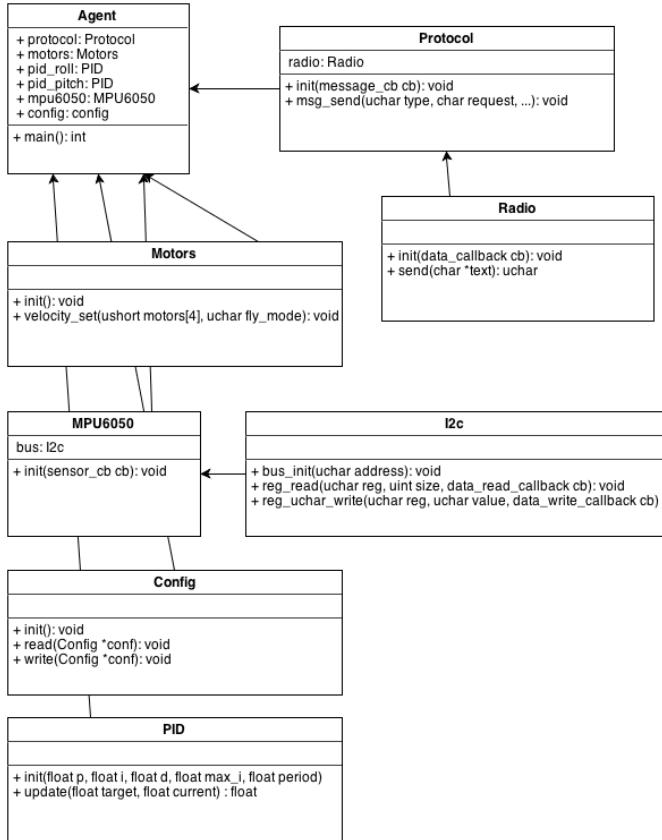


Fig. 11 – Diagrama de módulos

2.3.1 Interface microcontrolador-ESC

Como dito anteriormente o controle de velocidade do motor é feito pelo ESC, e para controlar a velocidade que o ESC fará cada motor rotacionar é necessário enviar a ele um sinal PWM.

PWM (*Pulse-With Modulation*) em tradução livre, modulação de largura de pulso, utilizado também em servos motores, onde alterando o sinal PWM o servo rotaciona para um certo ângulo.

Para controle de velocidade os ESCs requerem um pulso com tempo de 1ms a 2ms a cada 20ms, 1ms indicando a menor velocidade. Em alguns modelos de ESC não há rotação dos motores nesse pulso de 1ms, somente faz com que os ESCs fiquem armados e prontos para rotacionar os motores e 2ms sendo a maior velocidade de rotação. A Figura 12 representa 3 sinais PWM diferentes para controlar um ESC [19].

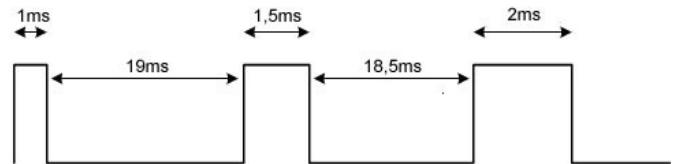


Fig. 12 – Exemplo de sinal PWM para controlar ESC

O microcontrolador do Stellaris possuí implementação em hardware para gerar PWM, porém sua configuração é complexa e não foi possível gerar o PWM que o ESC necessita então decidiu-se por implementar a geração de PWM por software.

O modulo de *agent* envia para o modulo de controle de motor um valor entre 1000 e 2000, esse valor nada mais é que o tempo do pulso em microsegundos.

A partir desse valor são calculados quantos *tricks* (ciclos) do processador são necessários para chegar a esse tempo. Os *tricks* são calculados baseando-se na frequência do microcontrolador. Nesse projeto foi configurado para o microcontrolador trabalhar a 16MHz, o que significa que para ter um pulso de 1ms temos o número de *tricks* igual a 16000.

Utilizando um *timer* (temporizador) implementado em *hardware* e o número de *tricks* a implementação do gerador de PWM foi simples. Necessitando só alterar o valor lógico dos pinos a cada interrupção do *timer*.

2.3.2 Interface microcontrolador-MPU6050

Obter a orientação do quadricóptero para realizar o controle de velocidade dos motores a fim de estabilizá-lo é a parte mais importante do projeto.

Para obter as informações de força gravitacional do acelerômetro e de velocidade angular do giroscópio é necessário comunicar com o MPU6050 utilizando o protocolo de comunicação I²C, este protocolo também implementado em *hardware* no microcontrolador do Stellaris.

O I²C (*Inter-Integrated Circuit*) é um protocolo de comunicação onde a comunicação entre os componentes é feita por meio de 2 fios: o SCL que

fornecer o *clock* para que todos os componentes compartilhem a mesma quantidade de ciclos por segundo e o SDA que é por onde é trocado mensagens entre os componentes. Outra vantagem do I²C é que ele permite que vários componentes estejam conectados no mesmo barramento. O *master*, o dono do barramento manda mensagens no barramento dizendo com qual dos componentes ele está enviando e de qual dos componentes ele quer uma resposta.

O I²C peca apenas na velocidade. Como toda a comunicação é feita por um único fio, ela tem uma velocidade bem inferior das encontradas no UART e SPI [20].

A troca de mensagens entre os componentes do barramento é feita da seguinte maneira, o *master* gera *clock* no SCL e no SDA ele envia o endereço do *slave* que ele quer comunicar, seguido de um bit indicando se o *master* vai escrever ou vai ler do barramento. Se ele for escrever, logo em seguida ele envia os bits da mensagem, a cada 8 bits o *slave* deve avisar o *master* que ele recebeu a mensagem enviando o valor lógico 0 no SDA. Da mesma forma acontece quando o *master* quer ler, após ele enviar o endereço do *slave* e o bit indicando que ele quer ler, o *slave* começa a enviar os bits da mensagem e a cada 8 bits o *master* deve enviar o valor lógico 0, avisando o *slave* que ele recebeu aqueles 8 bits.

Com um modulo I²C escrito para abstrair toda essa complexidade de envio e recebimento de mensagens, temos o modulo de controle do MPU6050, este tem como responsabilidade inicializar e configurar o MPU6050.

O MPU6050 também oferece um pino de interrupção, quando esse passa para o valor lógico 1, é necessário perguntar para o MPU6050 que tipo de interrupção ocorreu e tratá-la.

O MPU6050 fornece diversas opções de configuração, configurações estas como: a escala e a velocidade em que os valores do acelerômetro e giroscópio serão mensurados. Para esse projeto o MPU6050 foi configurado para que a cada 0,002 de segundo (500 Hz) os valores de acelerômetro e giroscópio sejam mensurados, estes na respectiva escala de -4 à +4 g e -500 à +500 °/s.

Para configurar o MPU6050, é necessário enviar algumas mensagens, todas estas mensagens estão documentadas no *datasheet* [21] do MPU6050.

2.3.3 Interface microcontrolador-transceptor

A interface de comunicação com o transceptor WiFly é feito utilizando o protocolo de comunicação UART (*Universal Asynchronous Receiver and Transmitter*). Este é um protocolo mais rápido que o I²C porém não fornece a possibilidade de ter vários equipamentos conectados no mesmo barramento.

Neste protocolo são utilizando dois fios, um para envio de dados e outro para recebimento. Deste modo o transmissor de um equipamento é conectado ao receptor do outro equipamento, possibilitando o envio e o recebimento de dados ao mesmo tempo. Como não há sincronismo com o *clock* que será usado na comunicação, é necessário que os dois equipamentos sejam configurados para trabalhar na mesma velocidade [22]. Nesse projeto a velocidade de comunicação definida foi 115200 bits por segundo.

O microcontrolador do Stellaris também possui implementação em *hardware* para protocolo de comunicação UART, sendo necessário somente configurar a velocidade da comunicação e adicionar as *callbacks* de envio e recebimento de dados.

O transceptor WiFly faz todo o duro trabalho de implementar a comunicação IEEE 802.11, conectar na rede Wi-Fi, enviar e receber pacotes TCP

(*Transmission Control Protocol*), gerenciar janelamento e reenvio de pacotes.

Por sua interface UART ele só envia e recebe o conteúdo dos pacotes TCP, facilitando e muito o projeto.

Exemplificando, para realizar o envio da *string* "Hello World", essa é colocada no *buffer* de transmissão UART do Stellaris. Este enviará essa *string* para o WiFly que criará um ou mais pacotes TCP e os enviará para o endereço IP (*Internet Protocol*) configurado nele. O mesmo acontece para o recebimento de mensagens, WiFly transmite pela UART somente o conteúdo do pacote TCP. A Figura 12 representa a comunicação entre os componentes.



Fig. 12 – Diagrama comunicação microcontrolador-transceptor

Para seu funcionamento somente é necessário realizar algumas configurações como: a velocidade de comunicação UART, o nome e a senha da rede Wi-Fi que o WiFly deve conectar e o endereço de IP para qual ele deve receber e enviar pacotes. No *datasheet* [23] é explicado como entrar em modo de configuração e configurá-lo.

2.3.4 Protocolo de comunicação

quadricóptero-controle

Utilizando o transceptor WiFly é possível realizar a comunicação entre o quadricóptero e o controle, porém é necessário definir como será essa comunicação, por isso foi criado um protocolo de comunicação entre os dois.

O formato das mensagens a serem trocadas seguem o seguinte padrão:

`^;<tipo>;<requisitando>;<parâmetro1>;<parâmetro2>;$`

"^": identifica o início de uma mensagem do protocolo

"\$": identifica o final da mensagem do protocolo

" ,": é o separador

<tipo>: um caractere identifica o tipo de mensagem. Exemplo: o caractere "m" identifica que a mensagem é do tipo de mensagem de movimento.

<requisitando>: um *boolean* identificando se esta é uma mensagem de requisição ou de resposta, exemplo o controle pedindo pro quadricóptero mover para esquerda ou uma mensagem de resposta quando o quadricóptero confirmando para o controle que ele recebeu a mensagem para mover o quadricóptero para a esquerda.

<parâmetros>: são os parâmetros da mensagem, cada tipo de mensagem tem parâmetros diferentes e em quantidades diferentes.

Tabela 3 – Tabela de mensagens e respostas

Caracter	Descrição	Requisição	Resposta
p	Pinga outro lado, para certificar que a comunicação está funcionando.	<code>^;p;1;<int número>;\$</code>	<code>^;p;0,<int número+1>;\$</code>
d	Mensagem de <i>debug</i> .	<code>^;d;1;<string mensagem>;\$</code>	<code>^;d;0;\$</code>
o	Habilita o envio da orientação do quadricóptero, deste modo a cada um período de tempo o quadricóptero envia sua orientação para o controle.	<code>^;o;1;<boolean 1=habilita envio ou 0=desabilita envio>;\$</code>	<code>^;o;0;<float roll>;<float pitch>;<float yaw>;\$</code>
e	Entra ou sai do modo de configuração dos ESCs.	<code>^;e;1;<boolean 1=entra ou 0=sai>;\$</code>	<code>^;e;0;\$</code>
h	Tempo de pulso a serem enviados aos ESCs no modo de configuração.	<code>^;h;1;<int esc da frente esquerda>;<int ESC da frente direita>;<int ESC de traz esquerda>;<int ESC de traz direito>;\$</code>	<code>^;h;0;\$</code>

f	Leitura de ganhos de PID.	$^{\wedge};f;1;\$$	$^{\wedge};f;0;<float>$ ganho proporcional>; $<float>$ ganho integral>;\$
w	Gravação dos ganhos de PID.	$^{\wedge};w;1;<float>$ ganho proporcional>; $<float>$ ganho integral>;\$	$^{\wedge};w;0;\$$
s	Arma motores.	$^{\wedge};s;1;\$$	$^{\wedge};s;0;\$$
m	Movimenta o quadricóptero.	$^{\wedge};m;1;<char>$ axis x, y, z ou rotação>;<int> velocidade>\$	$^{\wedge};m;0;\$$

Como visto é um protocolo simples de implementar e facilmente extensível, caso existam mais que 27 tipos de mensagens é possível utilizar uma das letras e usar ela como tipo de extensão, e nesse tipo extensão temos outro caractere como parâmetro para identificar mais 27 tipos de mensagem ou até mesmo usar um inteiro.

2.3.5 Interface de controle

Interface de controle é a interface que o operador do quadricóptero usará para controlar, mover e analisar as informações enviadas pelo quadricóptero.

Como dito anteriormente, essa interface será feita utilizando um *smartphone* Android para baratear o custo e reduzir complexidade do projeto. Utilizando um *smartphone* de 300 reais é completamente possível controlar o quadricóptero.

A interface foi implementada na forma de um aplicativo Android, utilizando a SDK (*Software Development Kit*) do Android e a linguagem de programação Java. Este aplicativo implementa todo o protocolo de comunicação quadricóptero-controle.

Esse artigo não abordará a implementação desse aplicativo, porém o código deste está também disponível para leitura e uso no GitHub [24].

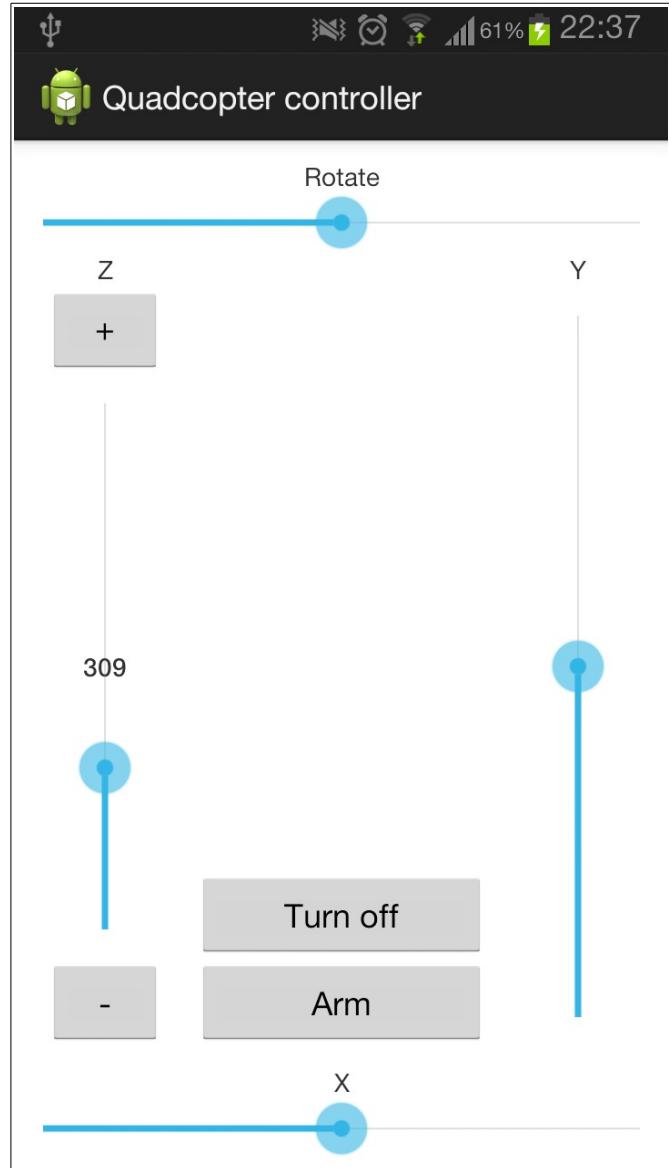


Fig. 13 – Interface do controle

A Figura 13 mostra a interface de controle do quadricóptero, nela o controle Z é utilizado para controlar a altura do quadricóptero, o controle Y para mover o quadricóptero para frente e para trás, o controle X para mover o quadricóptero para os lados direita e esquerda e por último o controle Rotate para rotacionar o quadricóptero sobre seu eixo. Também na mesma interface, o botão Arm para armar e deixar os ESCs prontos para voar e o botão Turn on/off, que liga e desliga os motores.

Para levantar voo é necessário seguir a seguinte sequência, *Arm->Turn on->*Aumentar gradativamente o controle Z até haver a decolagem.

A seguir a Figura 14 exibe a tela de *debug* da orientação do quadricóptero. Por ela é possível avaliar se os dados de orientação estão sendo coletados e calculados corretamente.

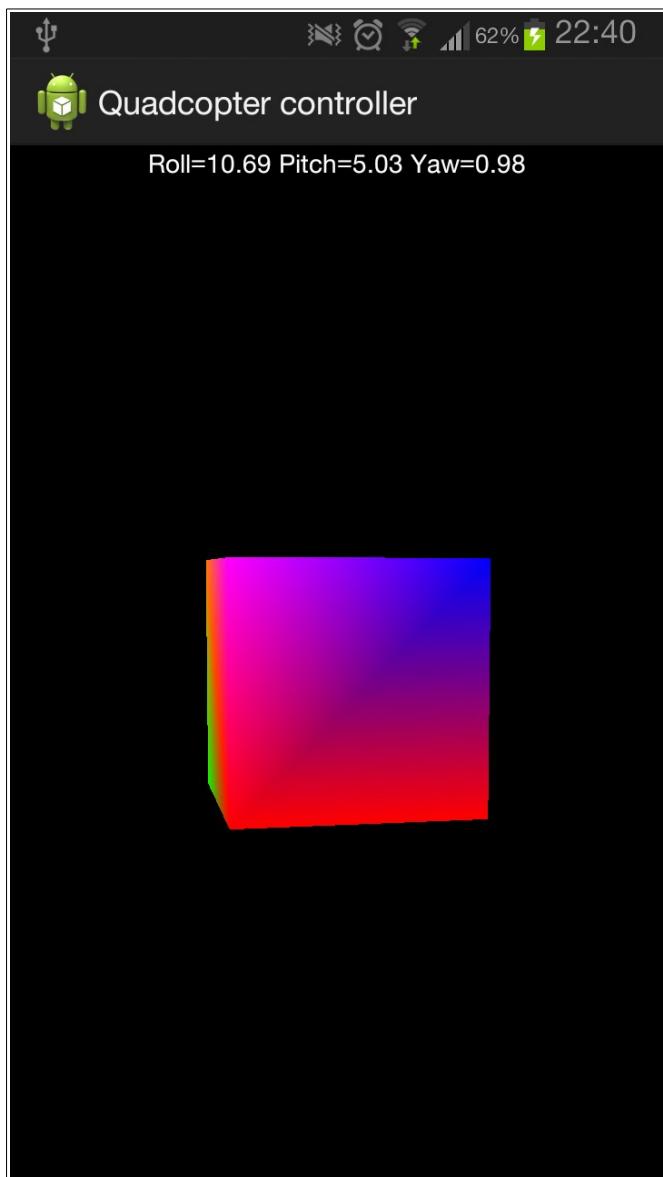


Fig. 14 – Tela para *debug* de orientação

Por fim, na Figura 15 tem se a tela para configurar os ganhos de PID do quadricóptero. No desenvolvimento esta funcionalidade foi muito utilizada para chegar em valores de ganhos que mantiveram o quadricóptero estável.

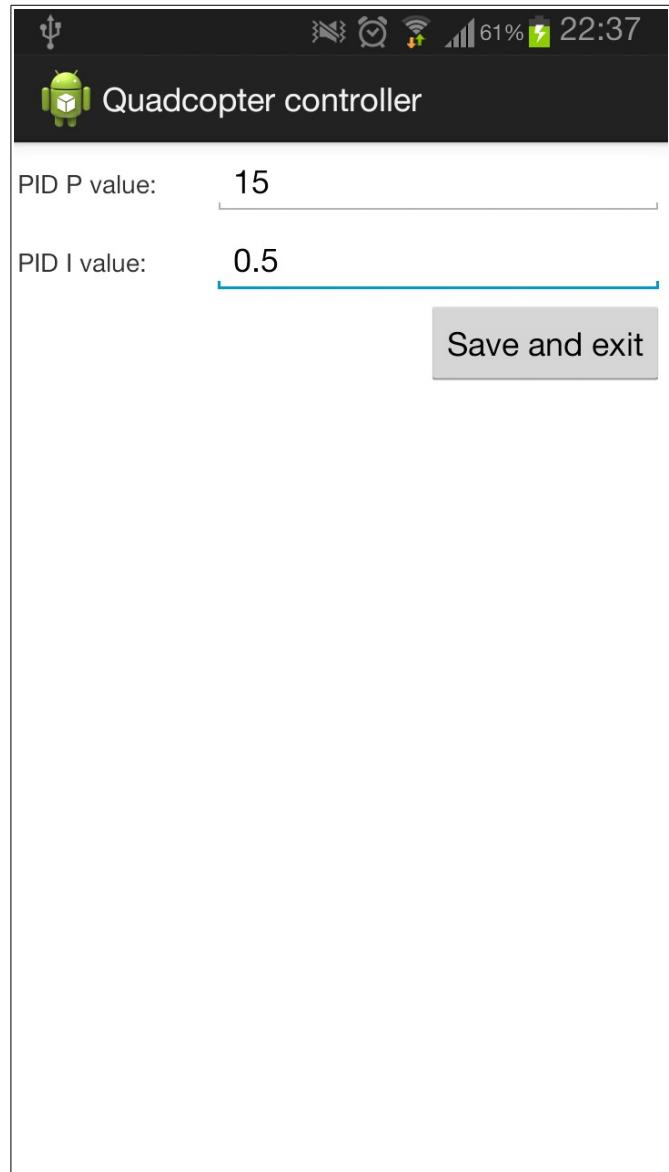


Fig. 15 – Tela para configuração dos ganhos do controlador PID

2.4 Estabilização e controle

Estabilização e controle é a parte mais importante e desafiadora deste projeto. Para manter o quadricóptero estável, os dados dos sensores de acelerômetro e giroscópio são usados para obter a orientação do quadricóptero. Caso esse esteja inclinando para algum lado a velocidade dos motores é alterada a fim de corrigir essa queda e estabilizar o quadricóptero.

O controle de movimentos nada mais é que alterar o ângulo que o quadricóptero deve manter. Quando é comandado para o quadricóptero inclinar para frente, esta alteração também o faz mover para frente. Isso vale

para movimentar o quadricóptero também para os lados sendo necessário alterar o ângulo de inclinação das laterais do quadricóptero. Um maior ângulo significa uma maior velocidade enquanto um ângulo negativo faz o quadricóptero mover para o outro lado.

Na Figura 16 é possível ver o fluxograma de estabilização e controle. Cada etapa será explicada nos tópicos a seguir.

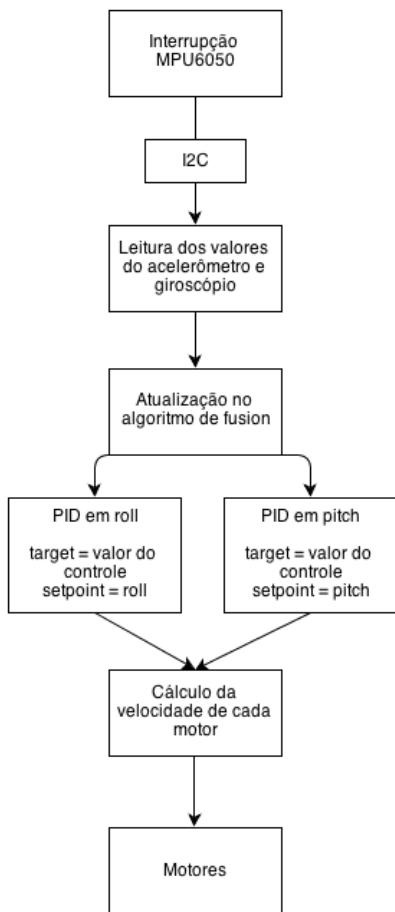


Fig. 16 – Fluxograma de estabilização e controle do quadricóptero

2.4.1 Orientação

O acelerômetro mede a força gravitacional aplicada sobre ele, deste modo quando o sensor está perfeitamente paralelo ao solo ele medirá:

$$X = 0 \text{ m/s}^2$$

$$Y = 0 \text{ m/s}^2$$

$$Z = 9,806 \text{ m/s}^2 = \text{aceleração gravitacional da Terra}$$

Rotacionando o acelerômetro esses valores mudam, com o ângulo rotacionado de cada coordenada e

utilizando trigonometria é possível calcular a força gravitacional aplicada em cada uma das coordenadas.

Já o giroscópio mede a velocidade angular, ou seja, a quantos metros por segundo o sensor está movendo em cada coordenada. Este é um valor instantâneo, assim que o sensor parar de mover a velocidade cai para zero.

Em fim o acelerômetro e o giroscópio somente conseguem medir movimentos aplicados em corpos rígidos [25]. Movimentos estes que foram descritos por Leonard Euler como os ângulos de Euler [26]. Estes ângulos definem três formas de movimento que podem descrever todos os movimentos usando as três coordenadas espaciais, evitando o uso de uma matriz com 9 cosenos diretores.

Derivado dos ângulos de Euler, os ângulos de Tait-Bryan [27], que nada mais são que os três ângulos de Euler aplicado em diferente ordem. Esse modelo de orientação é amplamente utilizado em aeromodelos, desde de aviões militares até helicópteros. Neste modelo temos três tipos de movimentos *roll*, *pitch* e *yaw*.

Roll é o movimento gerado rotacionando a frente do objeto no sentido horário ou anti-horário, como exibido na Figura 17.

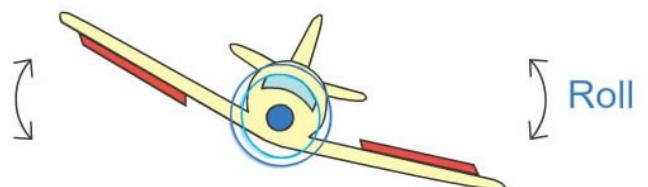


Fig. 17 – Avião movendo em *roll*
Fonte: <http://howthingsfly.si.edu/>

Pitch é o movimento gerado quando é inclinado a frente do objeto para cima ou para baixo, como visto na Figura 18.

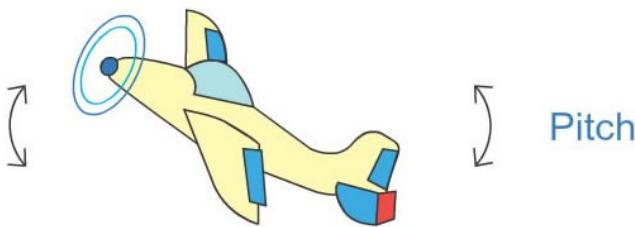


Fig. 18 – Avião movendo em *pitch*
Fonte: <http://howthingsfly.si.edu/>

E o movimento *yaw* é quando o aeromodelo é rotacionado pela sua superfície, como apresentado na figura 19.

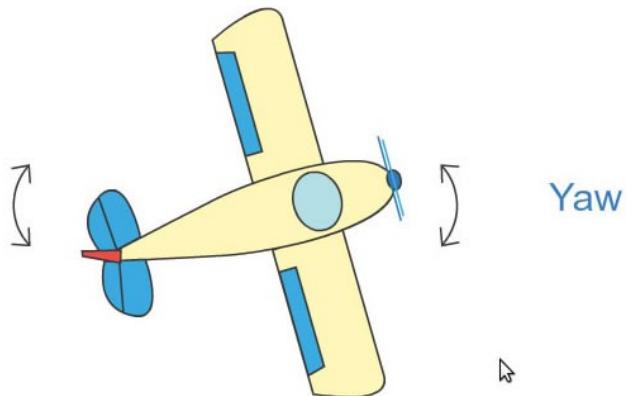


Fig. 19 – Avião movendo em *yaw*
Fonte: <http://howthingsfly.si.edu/>

É possível obter *roll*, *pitch* e *yaw* utilizando diversas equações e algoritmos, uma das mais simples é utilizando somente os valores do acelerômetro porém esse método tem um problema. Acelerômetros tendem a ser imprecisos quando recebem vibrações, vibrações que no quadricóptero vem dos motores

Outro método simples, é utilizando somente os valores de giroscópio, mas novamente temos um problema, giroscópios tendem a ter erros maiores conforme o tempo passa, neste modo após alguns minutos temos valores de *roll*, *pitch* e *yaw* inaceitáveis para estabilização do quadricóptero.

Qual é a solução? Juntar os valores de acelerômetro e sua precisão durante longo período de tempo e os valores do giroscópio e sua precisão sobre vibração.

Existe uma categoria de algoritmos chamados *fusion motion algorithms*, que fazem o trabalho de fundir os valores do acelerômetro e giroscópio e retornam valores precisos de *roll*, *pitch* e *yaw*. Entre eles temos o Kalman filter, Simple Simpson Integration, DCM filter, Complementary filter e o utilizado nesse projeto Madgwick gradient descent filter. A escolha se deve ao custo computacional e a precisão de cada um, há um vídeo [28] comparando três desses algoritmos.

Mas mesmo com um *fusion motion algorithm* não é possível obter um valor preciso de *yaw*. Para ter um valor preciso de *yaw* é necessário ter um magnetômetro, que mede a força dos campos magnéticos aplicados nas três coordenadas, com precisão para medir até a força do campo magnético dos polos da Terra. Somente com esses valores é possível obter valores de *yaw* precisos [29]. Neste projeto não existe um magnetômetro, mas isso não é um grande problema, claro que não será possível medir efetivamente se o quadricóptero está rotacionando em *yaw* na velocidade desejada, mas, mesmo assim, é possível ter um voo estável.

Este artigo não abordará como o Madgwick gradient descent filter funciona. Este é um algoritmo complexo desenvolvido como doutorado [29].

Sumarizando, com os valores dos dois sensores e o uso do algoritmo de Madgwick temos os valores de *roll* e *pitch* precisos.

2.4.2 Controlador PID

O controlador PID (*Proportional-Integral-Derivative*) [30] é um tipo de controlador de sistema, controladores de sistema tem a função de aplicar correções no sistema de modo a deixá-los estáveis. Controladores recebem a entrada desejada, o valor atual do sistema e através de modelagem matemática atuam no sistema.

Um controlador PID é formado por três subcontroladores, cada um contribuindo de um forma para a estabilização do sistema.

O primeiro subcontrolador é o proporcional, este multiplica o erro que é a diferença entre o valor desejado menos o valor atual, por um valor este chamado de ganho proporcional. Este controlador é o que mais contribui para o controle do sistema, o valor do ganho é escolhido com base na relação de controle e reação do sistema.

O próximo subcontrolador é o integral, este integra o erro, ou seja, ele soma erro a cada instante de tempo. O erro integrado é então multiplicado por outro valor este chamado de ganho integral. Esse controlador contribui com a correção de erros pequenos que acontecem por longos períodos de tempo, depois de certo tempo o erro integral será grande o suficiente para o controlador integral contribuir com uma correção para o sistema.

E por último o subcontrolador derivativo, este deriva o erro, ou seja, ele calcula a variação instantânea do erro. Esta variação é multiplicada novamente por outro valor este chamado ganho derivativo. Este controlador tem como objetivo diminuir a oscilação do sistema. Havendo uma grande variação de erro, este controlador contribuirá com um valor, que fará a oscilação diminuir a fim de contribuir com a estabilização do sistema.

Após o cálculo de cada um dos subcontroladores, seus resultados são somados resultando no valor de atuação do controlador PID, como mostrado na Figura 20.

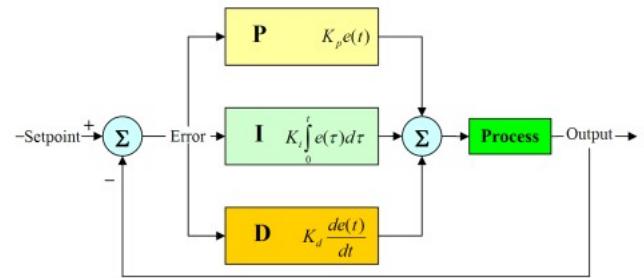


Fig. 20 – Controlador PID
Fonte: <http://radhesh.wordpress.com/>

A implementação de um controlador em *software* é uma tarefa trivial, porém a escolha dos ganhos não. Chegar em valores que deixem o sistema estável somente estimando valores é difícil. Mas existem alguns métodos para o cálculo dos ganhos baseados na dinâmica do sistema sem controlador. No entanto, não é possível utilizar esses métodos em um quadricóptero, pois o peso dele não está em perfeito equilíbrio, os motores não rodam na exata mesma velocidade e mesmo numa sala sem vento algum, o quadricóptero não conseguirá ter um voo estável, podendo oferecer perigo a pessoas em volta e riscos de quebra do aeromodelo se não utilizado nenhum tipo de controlador para estabilização, nos deixando somente com a opção de tentativa e erro.

Um controlador para *roll* e outro para *pitch* serão responsáveis por calcular a atuação necessário para manter o quadricóptero estável. Como não é possível obter um valor preciso de *yaw*, não há um controlador PID para *yaw*.

2.4.3 Cálculo de velocidade dos motores

As saídas dos PIDs fornecem o quanto devemos corrigir em *pitch* e *roll*, porém para efetivamente fazer a correção é necessário calcular a velocidade que cada motor deve rotacionar a fim de aplicar a correção. Como mostrado anteriormente, essa distribuição depende do formato do quadricóptero, nesse projeto o em X.

Para realizar um movimento positivo em *roll* é necessário desacelerar os motores 2 e 4 como mostrado na Figura 3 e acelerar os motores 1 e 3. Para realizar um movimento em positivo em *pitch* é preciso desacelerar os motores 1 e 2 e acelerar os motores 3 e 4. Para um movimento positivo em *yaw* os motores 1 e 4 são desacelerados e os motores 2 e 3 acelerados.

Transformando essa dinâmica de movimento em algoritmo:

```
fl = throttle - command_pitch + command_roll -  
command_yaw;  
fr = throttle - command_pitch - command_roll +  
command_yaw;  
bl = throttle + command_pitch + command_roll +  
command_yaw;  
br = throttle + command_pitch - command_roll -  
command_yaw;
```

Sendo o *throttle* a velocidade que enviamos do controle para configurar a altura do quadricóptero, *command_pitch* e *command_roll* as saídas dos PIDs e *command_yaw* o valor enviado pelo controle para *yaw*.

2.4.5 Análise de áreas de risco

Com um quadricóptero pronto e estável, é facilmente possível adicionar mais equipamentos nele para análise de áreas de risco. Entretanto não houve tempo hábil nesse projeto para um desenvolvimento nessa área. Porém, é possível utilizar equipamentos disponíveis no mercado para realizar essa tarefa, como adicionar uma câmera Go Pro [31], que faz gravação de imagens e vídeos em um número alto de quadros por segundo, sendo possível transmitir esses vídeos e imagens por Wi-Fi em tempo real, com o auxílio de um dispositivo com Wi-Fi e um navegador para acessar o *streaming* gerado pela câmera. Outra opção é utilizar um *smartphone* para realizar a filmagem, podendo utilizar o

software Skype [32] para transmissão de vídeos utilizando Wi-Fi ou 3G.

Outros tipos de sensores podem ser adicionados ao aeromodelo, como sensores de fumaça ou gás. Estes poderiam ser conectados aos pinos restantes do Stellaris e estendendo o protocolo de comunicação quadricóptero-controle para enviar essas informações para o controle.

3. RESULTADOS

O protótipo desenvolvido (Figura 21, Figura 22 e Figura 23), é capaz voar e de se autoestabilizar.



Fig. 21 – Protótipo de quadricóptero desenvolvido

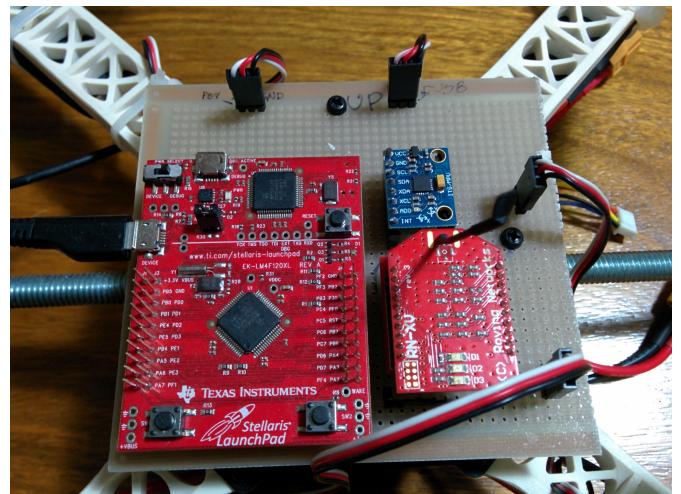


Fig. 22 – Placa de controle do protótipo

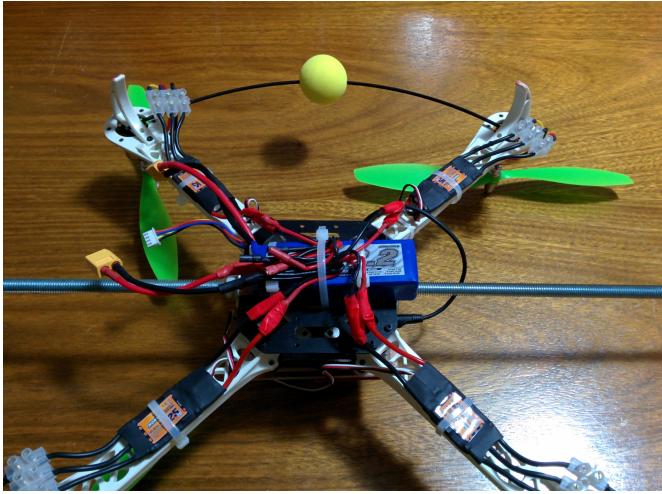


Fig. 23 – Lado inferior do protótipo

Os valores de ganho para os controladores PID que melhor mantiveram o quadricóptero estável foram:
 proporcional = 5
 integral = 0,01
 derivativo = 0

Como visto na Tabela 6, a cada segundo o controle recebe os valores de orientação do quadricóptero e o tamanho do pulso enviado aos ESCs de cada motor. No instante 130 o quadricóptero sofreu uma ação externa alterando bruscamente o valor de *roll*, onde este teve que realizar compensação para se estabilizar, o que foi realizado em menos de 4 segundos.

Tabela 4 – Estabilização em tempo real

T	Roll	Pitch	M1	M2	M3	M4
126	0.718	2.627	1396	1375	1475	1454
127	2.881	1.051	1453	1366	1484	1397
128	0.527	2.553	1395	1379	1471	1455
129	0.587	-1.131	1451	1433	1417	1399
130	-17.750	3.740	1103	1635	1215	1747
131	-10.980	5.120	1183	1513	1337	1667
132	3.600	2.930	1435	1327	1523	1415
133	-0.930	1.327	1391	1419	1431	1459
134	1.503	2.211	1414	1369	1481	1436
135	1.859	1.374	1432	1376	1474	1418
136	1.644	2.730	1408	1359	1491	1442
137	-1.593	0.002	1401	1449	1401	1449

4. CONCLUSÃO

Foi possível desenvolver um protótipo de um veículo aéreo não tripulado, manobrável, estável e de fácil controle, possuindo também capacidade para carregar equipamentos para análise de áreas de risco.

O desenvolvimento teve um baixo custo se comparado com alguns produtos comerciais disponíveis no mercado e com características similares. O mais conhecido de todos é o Parrot AR.Drone 2.0 [33] que custa 299,95 dólares, utiliza-se também *smartphone* para o controle do quadricóptero. A vantagem deste para o protótipo desenvolvido é que ele já possui uma câmera integrada por esse valor. Outro produto comercial é o DJI Phantom [2] que tem custo de 455,05 dólares, este controlado por um rádio controle incluso no preço do produto. A vantagem deste para o protótipo desenvolvido é que ele possui um receptor GPS (*Global Positioning System*), podendo utilizá-lo para fornecer informação de posicionamento e também para realizar pouso de emergência caso haja a perda de comunicação com o controle.

O protótipo desenvolvido teve custo de 200 dólares como mostrado na Tabela 7, e todos os componentes podem ser adquiridos através de duas lojas virtuais a HobbyKing e a SparkFun.

Tabela 5 – Custo do protótipo

Peça	QT.	Valor unitário	Valor total
Frame	1	\$17,49	\$17,49
ESC	4	\$5.99	\$23.96
Motor sem escova	4	\$9.04	\$36.16
Kit hélice CW e CCW	1	\$3.14	\$3.14
Bateria	1	\$16.99	\$16.99
Acelerômetro e giroscópio	1	\$9.8	\$9.8
Transceptor	1	\$34.95	\$34.95
Microcontrolador	1	\$7.99	\$7.99
Componentes diversos	1	\$50.00	\$50.00
Total			\$200,48

Todo o *software* do quadricóptero desenvolvido está disponível [34] [24] sobre a licença GPL3, e com a lista de equipamentos descrito nesse artigo é possível replicar o aeromodelo.

Algumas dificuldades encontradas foram como o *tunning* do PID, que causaram a destruição de várias hélices. Além disso, houve a alteração do microcontrolador no meio do projeto, inicialmente este seria desenvolvido usando um *kit* de desenvolvimento chamado MSP430 LaunchPad Value Line Development kit [35], que utiliza um microcontrolador da família MSP430, com 16 bits e um poder computacional muito inferior ao microcontrolador do Stellaris. Porém, o motivo da troca foi a falta de memória *flash* necessária para o código do quadricóptero.

Agradecimentos

Agradeço a todos os professores da Metrocamp por todo conhecimento que me foi ensinado, aos colegas de turma pela troca de conhecimento e amizade criada nesses 5 anos e aos meus amigos, familiares e namorada por todo suporte e apoio.

Referências

- [01] IDG Now. Anac e Força Aérea preparam regulamentação para uso de drones. Disponível em:
<http://idgnow.uol.com.br/ti-corporativa/2013/06/27/anac-e-forca-aerea-preparam-regulamentacao-para-uso-de-drones/>. Acesso em 09 de novembro de 2013
- [02] DJI. DJI Phantom Overview. Disponível em:
<http://www.dji.com/product/phantom/>. Acesso em: 09 de novembro de 2013
- [03] Vicente Martínez, "Modelling of the Flight Dynamics of a Quadrotor Helicopter". Disponível em:
http://prometheus4.com/share/quad-articles/martinez_2007.pdf. Acesso em 09 de novembro de 2013
- [04] Dynetic System. Brushless vs Brushed. Disponível em:
<http://www.dynetic.com/brushless%20vs%20brushed.htm>. Acesso em 09 de novembro de 2013
- [05] Mercury. How Propellers Work. Disponível em:
<http://www.mercurymarine.com/propellers/about/how-propellers-work/>. Acesso em 09 de novembro de 2013
- [06] Texas Instruments. Stellaris® LM4F120 LaunchPad Evaluation Board Overview. Disponível em:
<http://www.ti.com/tool/sw-ek-lm4f120xl/>. Acesso em: 09 de novembro de 2013
- [07] ARM. ARM Cortex-M4 Overview. Disponível em:
<http://arm.com/products/processors/cortex-m/cortex-m4-processor.php>. Acesso em 09 de novembro de 2013
- [08] Invensense. MPU6050 Overview. Disponível em:
<http://invensense.com/mems/gyro/mpu6050.html>. Acesso em 09 de novembro de 2013
- [09] IEEE. IEEE 802.11 standard. Disponível em:
<http://standards.ieee.org/about/get/802/802.11.html>. Acesso em: 09 de novembro de 2013
- [10] Roving Networks. RN-XV WiFly. Disponível em:
<http://rovingnetworks.com/products/RN171XV/>. Acesso em 09 de novembro de 2013
- [11] GlibTek. Lithium polymer battery packs. Disponível em:
<http://br.globtek.com/lithium-polymer-battery-packs.php>. Acesso em 09 de novembro de 2013
- [12] Texas Instruments. Code Composer Studio Overview. Disponível em: <http://www.ti.com/tool/ccstudio/>. Acesso em 09 de novembro de 2013
- [13] Texas Instruments. JTAG Tutorial. Disponível em:
http://www.ee.ic.ac.uk/pcheung/teaching/ee3_DSD/ti_jtag_seminar.pdf. Acesso em 09 de novembro de 2013
- [14] Doragasu. The complete tutorial for Stellaris LaunchPad development with GNU/Linux. Disponível em:
<http://kernelhacks.blogspot.com.br/2012/11/the-complete-tutorial-for-stellaris.html>. Acesso em 09 de novembro de 2013
- [15] GNU. GNU Compiler Collection. Disponível em:
<http://gcc.gnu.org/>. Acesso em 09 de novembro de 2013
- [16] Elinux. Toolchains. Disponível em:
<http://elinux.org/Toolchains>. Acesso em 09 de novembro de 2013
- [17] Eclipse. Eclipse CDT. Disponível em: <http://eclipse.org/cdt/>. Acesso em 09 de novembro de 2013
- [18] Texas Instruments. StellarisWare. Disponível em:
<http://www.ti.com/tool/sw-lm3s/>. Acesso em 09 de novembro de 2013
- [19] Lin Zhong. Guide to the Quadrotor Setup. Disponível em:
<http://www.ruf.rice.edu/~mobile/2010elec424/quadrotorsetup.htm>. Acesso em 09 de novembro de 2013
- [20] Robot Electronics. Using the I2C Bus. Disponível em:
http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html. Acesso em 09 de novembro de 2013

- [21] Invensense. MPU6050 datasheet. Disponível em:
<http://invensense.com/mems/gyro/documents/RM-MPU-6000A-00v4.2.pdf>. Acesso em 09 de novembro de 2013
- [22] The Linux Documentation Project. Serial Port Basics. Disponível em:
<http://www.tldp.org/HOWTO/Serial-HOWTO-4.html>. Acesso em 09 de novembro de 2013
- [23] Roving Networks. RN-XV WiFly datasheet. Disponível em:
<http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Wireless/WiFi/WiFiy-RN-XV-DS.pdf>. Acesso em 09 de novembro de 2013
- [24] José Roberto de Souza. Código fonte controle do quadricóptero. Disponível em:
<https://github.com/zehortigoza/quadcopter-controller/>. Acesso em 09 de novembro de 2013
- [25] Starlino. A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications. Disponível em:
http://www.starlino.com/imu_guide.html. Acesso em 09 de novembro de 2013
- [26] Wolfram, Euler Angles. Disponível em:
<http://mathworld.wolfram.com/EulerAngles.html>. Acesso em 09 de novembro de 2013
- [27] Vectoralgebra. Euler angles x Tait-Bryan angles. Disponível em: <http://vectoralgebra.info/euleranglesconventions.html>. Acesso em 09 de novembro de 2013
- [28] Mingyu Chen. Vídeo: Comparison of Methods for IMU Orientation Estimation. Disponível em:
<http://www.youtube.com/watch?v=V5p23OTXn7E>. Acesso em 09 de novembro de 2013
- [29] Sebastian O.H. Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. Disponível em:
http://www.x-io.co.uk/res/doc/madgwick_internal_report.pdf. Acesso em 09 de novembro de 2013
- [30] National Instruments. Explicando a Teoria PID. Disponível em:
<http://www.ni.com/white-paper/3782/pt/>. Acesso em 09 de novembro de 2013
- [31] Go Pro. Go Pro. Disponível em: <http://www.gopro.com>. Acesso em 09 de novembro de 2013
- [32] Skype. Skype. Disponível em: <http://www.skype.com>. Acesso em 09 de novembro de 2013
- [33] Parrot. Ardrone 2. Disponível em: <http://ardrone2.parrot.com/>. Acesso em 09 de novembro de 2013
- [34] José Roberto de Souza. Código fonte quadricóptero. Disponível em: <https://github.com/zehortigoza/stellaris-quad/>. Acesso em 09 de novembro de 2013
- [35] Texas Instruments. MSP430 LaunchPad Value Line Development kit overview. Disponível em:
<http://www.ti.com/tool/msp-exp430g2/>. Acesso em 09 de novembro de 2013