UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
ENGENHARIA ELETRÔNICA

UMBERTO XAVIER DA SILVA NETO

DESENVOLVIMENTO DE UM SISTEMA DE CONTROLE PARA QUADRIRROTORES

TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO
2019

UMBERTO XAVIER DA SILVA NETO

DESENVOLVIMENTO DE UM SISTEMA DE CONTROLE PARA QUADRIRROTORES

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina TCC2, do curso de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para a obtenção do título de Bacharel.

Orientador: Prof. Dr. Márcio Aurélio Furtado Montezuma.

CORNÉLIO PROCÓPIO
2019

Universidade Tecnológica Federal do Paraná
Campus Cornélio Procópio
Departamento Acadêmico de Elétrica
Curso de Engenharia Eletrônica

FOLHA DE APROVAÇÃO

Umberto Xavier da Silva Neto

Desenvolvimento de um sistema de controle para Quadrirrotores

Trabalho de conclusão de curso apresentado às 16:00hs do dia 26/06/2019 como requisito parcial para a obtenção do título de Engenheiro Eletrônico no programa de Graduação em Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná. O candidato foi arguido pela Banca Avaliadora composta pelos professores abaixo assinados. Após deliberação, a Banca Avaliadora considerou o trabalho aprovado.

_____

Prof(a). Dr(a). Marcio Aurelio Furtado Montezuma - Presidente (Orientador)

_____

Prof(a). Dr(a). Alessandro do Nascimeto Vargas - (Coorientador)

_____

Prof(a). Dr(a). Kleber Romero Felizardo - (Membro)

_____

Prof(a). Dr(a). Luis Fernando Caparroz Duarte - (Membro)

A folha de aprovação assinada encontra-se na coordenação do curso.

Dedico este trabalho à minha família e amigos.

AGRADECIMENTOS

Agradeço ao meu orientador Prof. Dr. Márcio Aurélio Furtado Montezuma pela oportunidade de pesquisar e aprender ao seu lado.

Aos meus colegas do laboratório LaSisC.

Agradeço também a minha família, pilar fundamental de toda minha formação acadêmica.

“I used to be an adventurer like you. Then i took an arrow in the Knee”

Guarda de Whiterun

RESUMO

SILVA NETO, U. X. **Desenvolvimento de Um Sistema de Controle para Quadrirrotores.** 2018. 282 f. Trabalho de Conclusão de Curso (Graduação) – Engenharia Eletrônica. Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2018.

A fim de ambientar-se entre os sistemas quadrirrotores e, principalmente, entre as controladoras OSPs, foram construídos dois sistemas quadrirrotores com orientações diferentes. Para a confecção foram utilizados frames já disponíveis no mercado, e toda adaptação necessária foi realizada com a produção em medida de peças através do sistema de controle numérico computadorizado. Tendo a estrutura pronta, foi realizada a instalação dos componentes eletrônicos, acomodando todo o sistema embarcado. Foram utilizadas duas controladoras OSPs diferentes em cada sistema, a *MultiWii Pro*, contendo um *ATMega2560* e a *CC3D OpenPilot*, contendo um *STM32.* Sendo a *MultiWii Pro* uma controladora com características favoráveis, esta foi a selecionada para realizar o desbravamento de seu firmware. Tendo a necessidade de calibração dos controles clássicos, geralmente em cascata e com um sistema de aumento de estabilidade, embarcados nessa controladora, foi desenvolvida uma estrutura de calibração, onde juntamente com uma comunicação com o software *SIMULINK* através do protocolo serial identificado, foram analisadas as respostas na busca de um resultado ótimo entre saída e referência dos controles.

Palavras-chave: Quadrirrotores. Controladoras. Controle Clássico. Open-Source.

ABSTRACT

SILVA NETO, U. X. **Development of a Control System for Quadrotors.** 2018. 282 p. Course Completion Work (University graduate) – Eletronic Engineering. Federal Techonological University of Paraná. Cornélio Procópio, 2018.

In order to interface between the Quadrotors systems and the OSP controllers, two Quadrotors systems with different orientations were built. In the manufacture were used frames available on the market, and the necessary adaptation was performed with the parts production through the computer numeric control. Having the structure done, the installation of electronic components was realized, placing the entire embedded system. Two OSPs controllers were used in each system, The *MultiWii Pro*, with an *ATMega2560* embedded, and the *CC3D OpenPilot*, with a *STM32* embedded. With the *MultiWii Pro* being a controller with favorable features, this was the one selected to perform the pathfinder of its firmware. The need of calibrate the classic controls, usually in feedback configuration and with the stability augmentation system, led to the development of a calibration structure, where along the serial communication protocol identified, were used to analyze answers in search of a better result between setpoint and output.

Keywords: Quadrotors. Control Boards. Classic Control. Open-Source.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ADCs	Analog to Digital Converter
CNC	Computer Numeric Control
CI	Circuito Integrado
CISC	Complex Instruction Set Computer
CPU	Central Process Unit
DAC	Digital to Analog Converter
DOF	Degrees Of Freedom
EEPROM	Eletrically-Erasable Programmable Read-Only Memory
ESC	Eletronic Speed Control
GCS	Ground Control Software
GPL	General Public License
HTA	Heavier Than Air
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
JTAG	Joint Test Action Gorup
LASISC	Laboratório de Sistemas Automatizados e Controle
LTA	Lighter Than Air
LCD	Liquid Crystal Display
NASA	National Aeronautics and Space Administration
OSPs	Open Source Projects
P	Proporcional
PD	Proporcional Derivativo
PID	Proporcional Integral Derivativo
PWM	Pulse Width Modulation
RISC	Reduced Instruction Set Computer
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
UAVs	Unmanned Aerial Vehicle
USAF	United States Air Force
USB	Universal Serial Bus
VANTs	Veículos Aéreos Não Tripulados
VTOL	Vertical Take Off and Landing

LISTA DE SÍMBOLOS

ψ	Ângulo de guinada
θ	Ângulo de arfagem
φ	Ângulo de rolagem

SUMÁRIO

1 INTRODUÇÃO

Os veículos aéreos não tripulados (VANTs, do inglês Unmanned Aerial Vehicles, UAVs) são veículos aéreos sem a necessidade de um corpo humano embarcado para operação. (BORSOI et al., 2012). Nos dias de hoje, sua operação pode ser feita de maneira remota, apenas diminuindo a intervenção humana com a utilização de um rádio, ou completamente autônoma, através de um sistema de navegação e um plano de voo. (LISBOA, 2014).

O desenvolvimento tecnológico vem adicionando possibilidades de aplicações, antes muito complexas, em veículos aéreos não tripulados, mais especificamente, possibilidades provenientes do surgimento e melhoria gradual de dispositivos eletrônicos como processadores e sensores de aplicação inercial compactos. (PAULA, 2012). As aplicações originam-se de uma ampla variação de áreas de atuação, como cartografia, manutenção de infraestruturas, segurança pública, intervenção em ambientes hostis e agricultura de precisão. (BORSOI et al., 2012).

Dentre as vantagens em relação ás aeronaves tripuladas, citam-se: maior manobrabilidade, permitindo forças gravitacionais não suportáveis pelo corpo humano; voos em locais de difícil acessibilidade, como em locais hostis; modelos que permitem longas jornadas e altitudes de voo (Global Hawk[1] produzido pela NASA); anulação de riscos fornecidos por tripulantes; custo de produção e manutenção menores. (LISBOA, 2014; PAULA, 2012).

Um exemplo emblemático de VANT produzido no final do século XX no setor militar é o MQ-1 Predator[2], ilustrado na Figura 1. Fabricado pela General Atomics, esse VANT foi equipado pela Força Aérea dos Estados Unidos (USAF) para exercer funções de reconhecimentos e ataques de bombardeamento aéreo. Ele é provido das duas formas de operação, a remota e a autônoma. (LISBOA, 2014; PAULA, 2012).

[1] airforce-technology, “Global Hawk High-Altitude, Long-Endurance,” [Online]. Disponível: http://www.airforce-technology.com/projects/rq4-global-hawk-uav/. [Acesso em Outubro 2018].

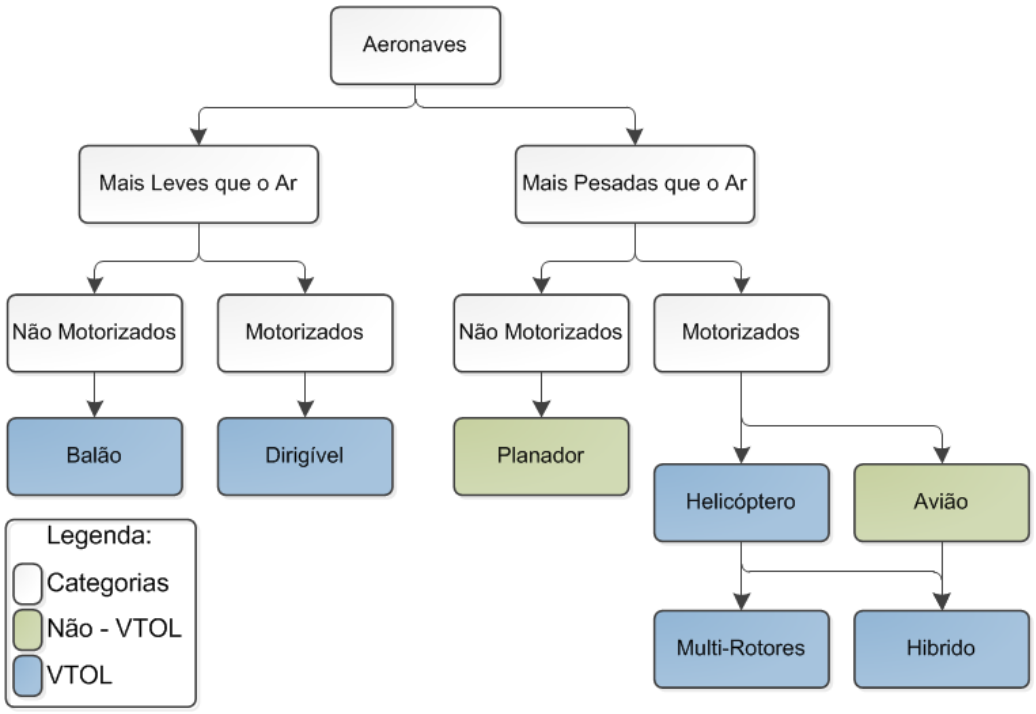
[2] airforce-technology, “Predator RQ-1 / MQ-1 / MQ-9,” [Online]. Disponível: http://www.airforce-technology.com/projects/predator/. [Acesso em Outubro 2018].

Figura 1 - MQ-1 Predator

Fonte: General Atomics (2016).

Segundo (BOUABDALLAH, 2007), as aeronaves podem ser divididas em duas categorias principais: mais leves que o ar (LTA, Lighter Than Air) e mais pesadas que o ar (HTA, Heavier Than Air). A Figura 2 faz a classificação das aeronaves em relação ao princípio de voo e ao modo de propulsão, variando entre não motorizados e motorizados.

Figura 2 - Classificação geral das aeronaves



Fonte: Paula (2012, p. 5).

Aviões possuem uma boa velocidade de cruzeiro e uma excelente autonomia, tendo sua utilização, por exemplo, em vistorias de grandes áreas. Porém, por possuírem propulsores com hélices na vertical, necessitam de uma velocidade mínima de sustentação produzida pela diferença de pressão sob suas asas. Portanto, além de não ser possível atingir velocidades pequenas para vistorias mais detalhadas, são necessárias áreas de decolagem e pouso, limitando o alcance de sua operação. (COSTA, 2012).

Entrando na categoria de veículos de pouso e decolagem vertical (do inglês Vertical Take Off and Landing, VTOL), pode-se citar primeiramente os balões e dirigíveis autônomos, que ainda possuem uma excelente autonomia e conseguem manter velocidades mínimas, sendo aplicável em vistorias mais detalhadas. O grande problema é, por serem LTA e por possuírem grande volume, a susceptibilidade às irregularidades climáticas. (COSTA, 2012; PAULA, 2012).

Ainda na categoria VTOL, têm-se as aeronaves que pairam no ar e, por isso, são aplicáveis em vistorias mais detalhadas. Os helicópteros e multi-rotores fornecem grande manobrabilidade e precisão, deixando de lado a autonomia e dando maior foco à precisão. (COSTA, 2012). A diferença entre helicópteros e multi-rotores é, principalmente, a ausência do rotor de cauda no segundo, dependendo apenas das velocidades individuais das hélices para sua locomoção, não havendo a necessidade de alterar o ângulo de ataque das hélices, o que ocorre nos helicópteros. (BRAGA E SANTANA, 2008; PAULA, 2012).

Dentro da categoria de multi-rotores, destacam-se os quadrirrotores, onde seu controle é objetivo de estudo deste trabalho. Os quadrirrotores tem sua propulsão feita por quatro rotores de empuxo vertical dispostos nas extremidades de uma estrutura em forma de cruz. (PAULA, 2012). Sua dinâmica e conceitos gerais de funcionamento terão um capítulo reservado neste trabalho.

Devido ao número de rotores, os quadrirrotores possuem um comportamento de voo bem variado, podendo se manter em posição estática com grande estabilidade e voar para qualquer direção independendo de sua posição. Esse comportamento acaba tornando o controle desses quadrirrotores bem complexo, justamente por serem sistemas com muitas variáveis e com uma dinâmica complexa e fortemente acoplada. (BORSOI et al., 2012).

Como a estabilidade e movimentos dos quadrirrotores são controlados por meio de modificações na força dos quatro atuadores através da leitura e

processamento dos dados fornecidos pelos sensores inerciais, que exercem papel de uma unidade de medida inercial (do inglês Inertial Measurement Unit, IMU), aplica-se um controle de atitude. Tal controle consiste num conjunto de controles capaz de manter uma estimativa dos ângulos de rolagem, arfagem e guinada enquanto o quadrirrotor se movimenta. (BORSOI et al., 2012).

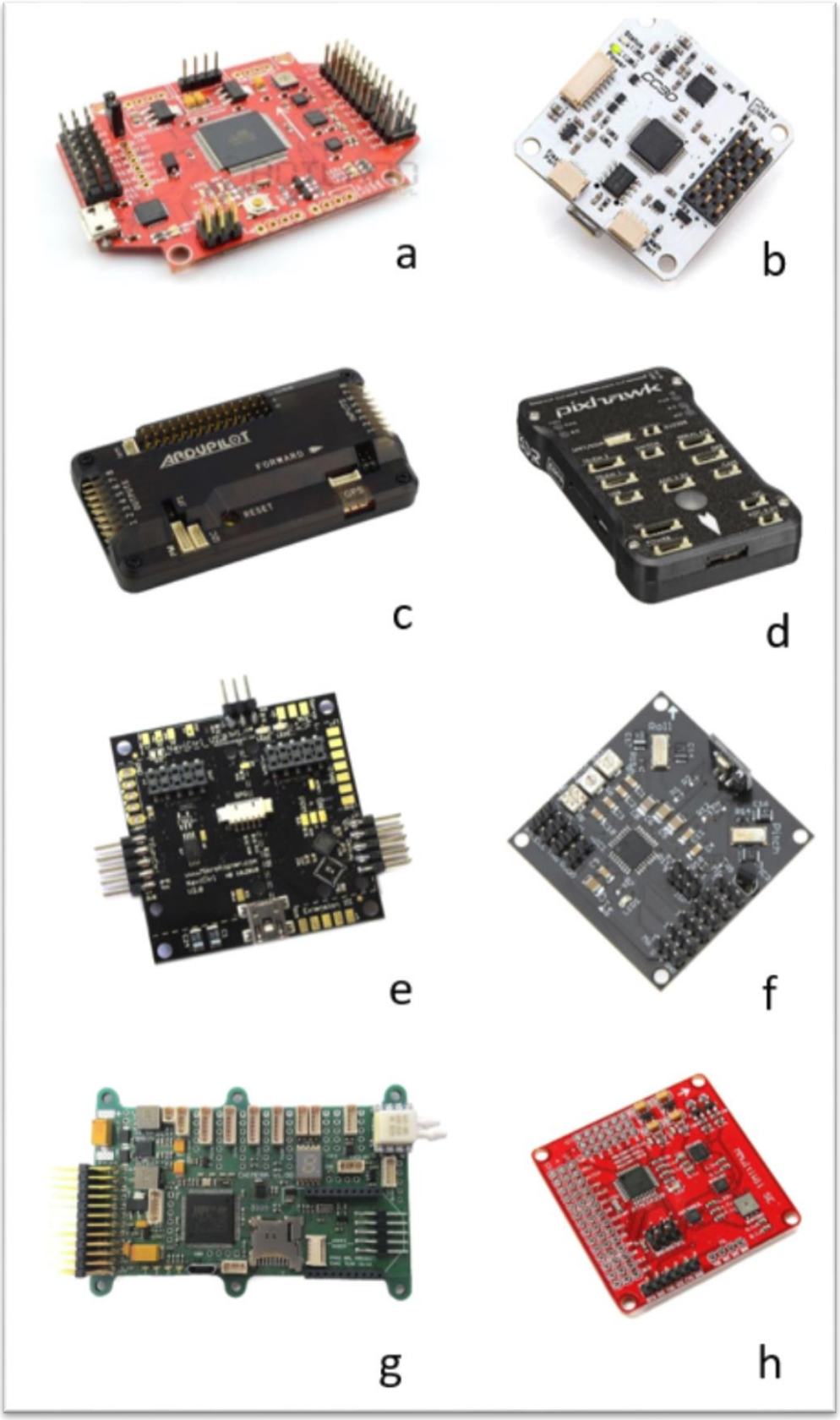
O controle de atitude pode ser aplicado em técnicas clássicas de controle, como Proporcional (P), Proporcional Derivativo (PD) e Proporcional Integral Derivativo (PID), sendo estas dependentes de ajustes em seus ganhos. (BORSOI et al., 2012). Estes controles são comumente encontrados em sistemas embarcados nos projetos de código fonte aberto (do inglês Open Source Projects, OSPs), aparecendo em várias configurações diferentes.

OSPs de Quadrirrotores fazem a utilização de sites de hospedagem comunitária, como Google Code e Github, para a criação e divulgação de códigos, plantas e esquemáticos. Essas criações são disponibilizadas como licença pública geral (do inglês General Public License, GPL), abrindo oportunidade para desenvolvedores independentes de todo o mundo. Este método acaba permitindo um desenvolvimento muito rápido, pois toda uma comunidade estaria engajada testando novas características e retornando respostas sobre várias condições e configurações, tornando o firmware mais robusto. (KIM et al., 2012).

OSPs permitem que pesquisadores repliquem e ampliem os resultados de outros e providenciem uma base para comparação entre várias abordagens. (KIM et al., 2012). Analisar e compreender essas abordagens, tanto em nível de hardware quanto de software, é estar ciente de métodos utilizados em nível comercial.

A Figura 3 ilustra algumas das placas de controle dos OSPs de quadrirrotores disponíveis no mercado.

Figura 3 - OSPs de quadrirrotores. (a)*MultiWii Pro*; (b)*OpenPilot CC3D*; (c)*Ardupilot*; (d)*Pixhawk;* (e)*Mikrokopter*; (f)*KKmulticopter*; (g)*Paparazzi;* (h)*Muliwii SE*



Fonte: Autoria Própria.

1.1 Objetivos

Nesta seção estão descritos o objetivo geral e os objetivos específicos.

1.1.1 Objetivo geral

O objetivo deste trabalho de conclusão de curso é estudar os controles e abordagens presentes para o desenvolvimento das controladoras de quadrirrotores que fornecem projetos de código fonte aberto. Analisa-se por fim, a relação entre referência e saída das malhas de controle identificadas, buscando um resultado ótimo.

1.1.2 Objetivos específicos

- Construção de duas estruturas de quadrirrotores;
- Estudo, compreensão e instalação dos componentes eletrônicos presentes em um quadrirrotor;
- Estudo e apresentação detalhada do firmware modificado presente em uma das placas controladoras de código fonte aberto (*MultiWii Pro*);
- Reconhecimento e análise das malhas de controle presentes na *MultiWii Pro*;
- Desenvolvimento de uma plataforma de sintonização de ganhos de controle para um sistema quadrirrotor;
- Reconstrução do esquemático da placa MultiWii Pro.

2 O QUADRIRROTOR

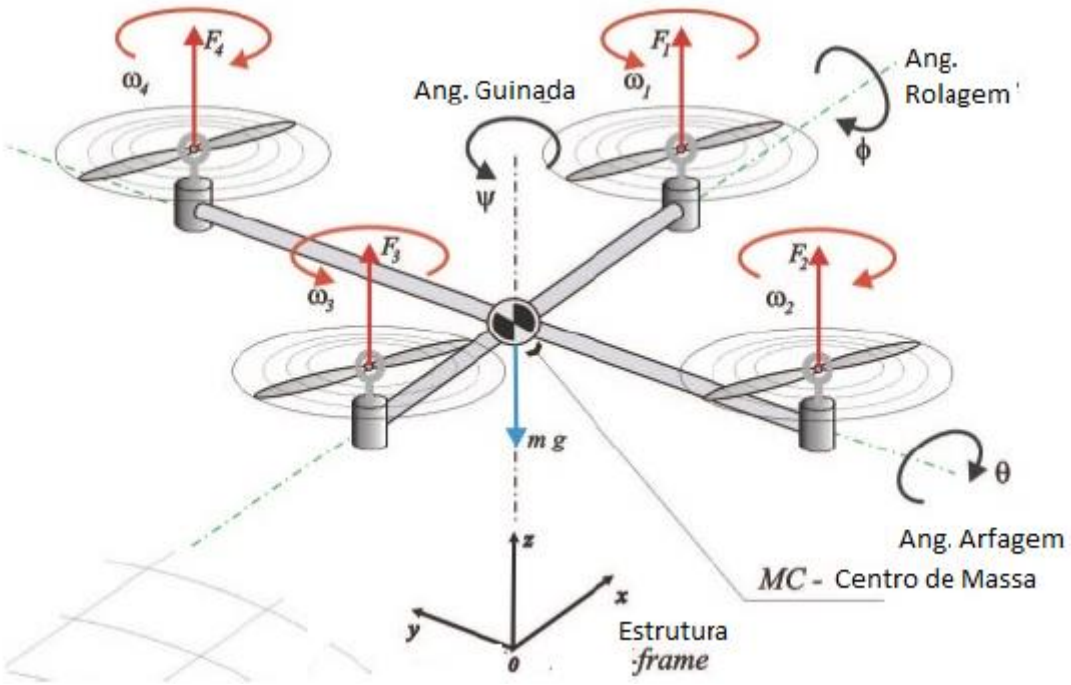
Conforme análise da Figura 2, o quadrirrotor é uma aeronave HTA, motorizada, com derivação direta dos helicópteros por possuir propulsão com rotores de empuxo vertical, possui características de decolar e aterrissar na vertical (VTOL) e oferece a possibilidade da realização de voos pairados. Sua estrutura tem formato simétrico em forma de cruz, onde cada extremidade abriga um rotor, totalizando quatro, e em seu centro localizam-se os componentes da eletrônica embarcada responsáveis pelo controle remoto e autônomo. (PAULA, 2012). A Figura 4 apresenta um tipo de quadrirrotor comumente utilizado em meio acadêmico.

Figura 4 - Quadrirrotor em desenvolvimento. Um sistema anti-colisão foi instalado através de sensores sonar

Fonte: DIY DRONES (2014).

A Figura 5 introduz a análise dos conceitos gerais de funcionamento de um quadrirrotor. Esta figura trata-se de um modelo simplificado, desconsiderando efeitos particulares de elementos do sistema e perturbações externas.

Figura 5 - Modelo de um quadrirrotor



Fonte*:* BORSOI et al (2012).

O princípio básico de funcionamento de um quadrirrotor é o sentido de rotação contrária entre hélices adjacentes. (BRAGA e SANTANA, 2008). Na Figura 5, observa-se que os rotores 1 e 3 giram em sentido anti-horário, enquanto os rotores 2 e 4 giram em sentido horário, cancelando os efeitos giroscópios ocasionados pelos torques dos rotores (É por conta dos efeitos giroscópios que os helicópteros possuem um rotor de cauda). Por conta dessa configuração, as hélices que giram no sentido anti-horário possuem seu passo invertido para que a direção do empuxo ($\vec{F}_n$) se iguale em todos os rotores do sistema. (PAULA, 2012).

Um quadrirrotor possui seis graus de liberdade (do inglês Degrees Of Freedom, DOF), havendo seis movimentos, 3 de rotação e 3 de translação. Como há a disposição de 4 rotores, tem-se um sistema sub atuado, possibilitando o controle direto de apenas 4 movimentos: altitude e atitude (rolagem (φ), arfagem (θ) e guinada (ψ)). Os outros movimentos translacionais em relação ao eixo x e y são dependentes dos controláveis. (PAULA, 2012).

A Tabela 1 demonstra as variações de velocidade das hélices da Figura 5 para cada movimento desejado, onde Ω_c é a velocidade angular constante para gerar empuxo vertical suficiente contra a aceleração gravitacional, e Ω_v é a velocidade angular variável.

Tabela 1 - Variação de velocidade angular para movimentos em uma orientação em cruz (+).

Movimento	Ω_1	Ω_2	Ω_3	Ω_4
Pairado	Ω_c	Ω_c	Ω_c	Ω_c
Altitude	$\Omega_c \pm \Omega_v$	$\Omega_c \pm \Omega_v$	$\Omega_c \pm \Omega_v$	$\Omega_c \pm \Omega_v$
Arfagem	$\Omega_c \pm \Omega_v$	Ω_c	$\Omega_c \mp \Omega_v$	Ω_c
Rolagem	Ω_c	$\Omega_c \pm \Omega_v$	Ω_c	$\Omega_c \mp \Omega_v$
Guinada	$\Omega_c \pm \Omega_v$	$\Omega_c \mp \Omega_v$	$\Omega_c \pm \Omega_v$	$\Omega_c \mp \Omega_v$

Fonte: Paula (2012).

Em voo pairado, todas as hélices precisam manter as mesmas velocidades. A soma dos empuxos deve ser o suficiente para se igualar à força da gravidade. (PAULA, 2012).

Para ganho ou perda de altitude, todas as hélices aumentam ou diminuem suas velocidades na mesma proporção. (PAULA, 2012; BRAGA e SANTANA, 2008).

O movimento de arfagem é feito no sentido positivo quando a hélice 3 aumenta sua velocidade e a 1 diminui, no sentido negativo quando a hélice 1 aumenta sua velocidade e a 3 diminui. As velocidades das hélices 2 e 4 são aumentadas na mesma proporção para compensar a perda de empuxo vertical na angulação. (BRAGA E SANTANA, 2008; PAULA, 2012).

O movimento de rolagem é feito de maneira análoga ao de arfagem, utilizando as hélices 2 e 4 para variação de angulação e as hélices 1 e 3 para compensar a perda de empuxo vertical. (BRAGA E SANTANA, 2008; PAULA, 2012).

Têm-se o movimento de guinada através do desbalanceamento do torque giroscópio, ou seja, quando a velocidade do par de hélices que giram na mesma direção é aumentada e do outro par, diminuída. O quadrirrotor gira em torno do eixo Z no sentido contrário ao par de hélices de maior rotação. (BRAGA E SANTANA, 2008; PAULA, 2012).

Há também estruturas com orientação em X, mudando a relação vista na Tabela 1, onde os movimentos de rolagem e arfagem são feitos quando um par de hélices aumenta sua velocidade, e o outro par diminui, mudando a relação dos pares para cada movimento respectivo.

3 SISTEMAS MICROCONTROLADOS

Na atualidade, os microcontroladores estão presentes na maioria dos dispositivos eletrônicos, desde aqueles do nosso dia a dia, até nos sistemas industriais mais complexos. Essa grande concentração no mercado se deve a sua facilidade de permitir o desenvolvimento de inúmeros projetos, abrangendo várias classes de aplicação, devido ao fato de conter grande número de funcionalidades (periféricos) em um único componente. (LIMA e VILLAÇA, 2012, p. 2).

Antes de descrever mais a fundo sobre os microcontroladores e suas funcionalidades, é necessário discutir um pouco sobre os microprocessadores.

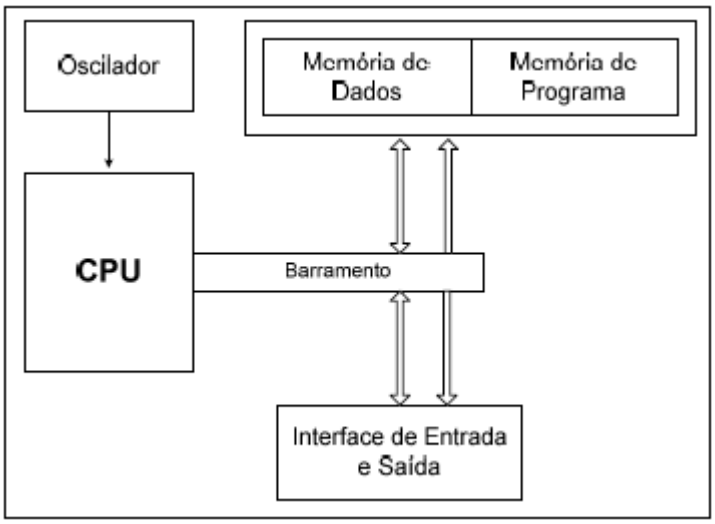
3.1 Microprocessadores

Um microprocessador é um CI (circuito integrado) com inúmeras portas de operação lógica, organizadas com o intuito de realizar operações digitais, lógicas e aritméticas. Instruções presentes em sua memória programável ditam sua operação, sendo essas obedecidas por um sinal periódico, chamado de CLOCK. (LIMA e VILLAÇA, 2012, p. 2).

A estrutura básica de um sistema microprocessado é apresentada na Figura 6. Esta é composta por um oscilador, que gera o sinal de CLOCK, a CPU, que é responsável pelas operações supracitadas, a memória de programa, que armazena o código de programa a ser executado, e a memória de dados, onde estão armazenadas informações para o trabalho da CPU, informações estas temporárias. Há junto uma interface de entrada e saída, onde o microprocessador pode fazer a recepção de dados externos e transferência de dados internos. (LIMA e VILLAÇA, 2012, p. 2).

As arquiteturas de barramento, Von-Neumann e Harvard, e o conjunto de instruções, CISC e RISC, não serão discutidos, apenas deve-se ter em mente, que as informações dentro do microprocessador são transportadas por barramentos, e que o emprego de um dos conjuntos de instruções varia entre os processadores disponíveis no mercado.

Figura 6 - Estrutura de um sistema microprocessado



Fonte: Lima e Villaça (2012, p.4).

3.2 Microcontroladores

A necessidade de citar os microprocessadores subsiste no fato de que o microcontrolador é, basicamente, um microprocessador com algumas funcionalidades (periféricos) embutidas. Essas funcionalidades podem ser observadas na Figura 7.

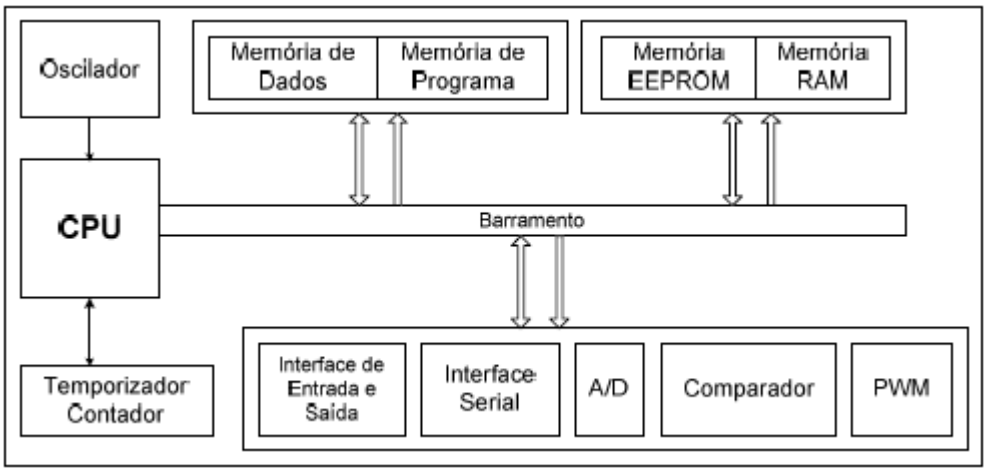
De acordo com LIMA e VILLAÇA (2012, p. 9):

> Dentre as funcionalidades encontradas nos microcontroladores, pode-se citar: gerador interno independente de clock (não necessita de cristal ou componentes externos); memória SRAM[3], EEPROM[4], e flash[5]; conversores analógicos-digitais (ADCs), conversores digitais-analógicos (DACs); vários temporizadores/contadores; comparadores analógicos; saídas PWM; diferentes tipos de interface de comunicação, incluindo USB, USART, I2C, CAN, SPI, JTAG, Ethernet; relógio de tempo real; circuitos para gerenciamento de energia no chip; circuitos para controle de inicialização (reset); alguns tipos de sensores; interface para LCD; e outros periféricos de acordo com o fabricante.

Algumas destas funcionalidades citadas serão discutidas no decorrer deste trabalho, conforme forem apresentadas partes pertinentes e que façam a aplicação destas.

[3] Memória volátil para o armazenamento de dados que não precisam ficar retidos após a desenergização do circuito.
[4] Memória regravável eletricamente para o armazenamento dos dados.
[5] Memória regravável eletricamente para o armazenamento do programa.

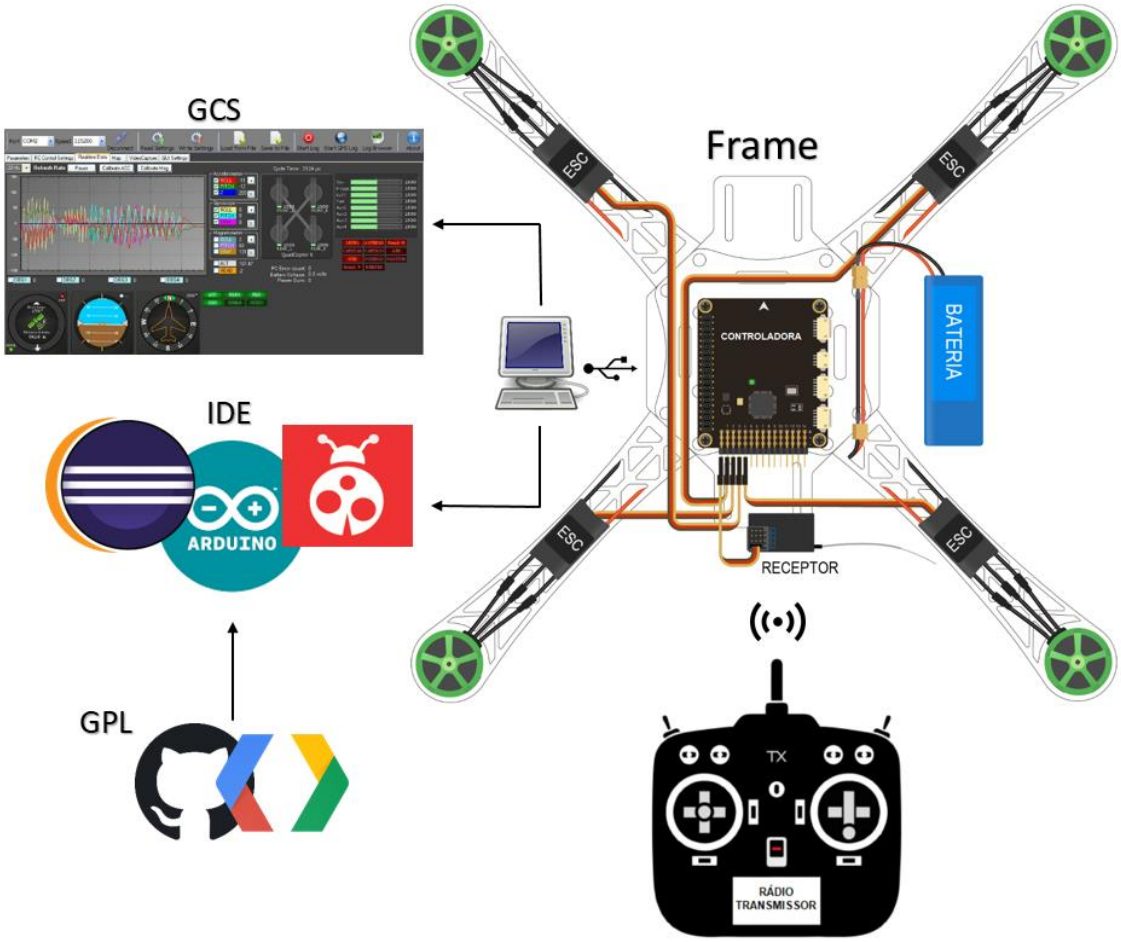
Figura 7 - Estrutura de um microcontrolador



Fonte: Lima e Villaça (2012, p.10).

4. DISPOSITIVOS DE UM QUADRIRROTOR

Figura 8 - Sistema geral de um quadrirrotor com controladora OSPs



Fonte: Autoria Própria.

A Figura 8 demonstra um sistema geral simplificado de um quadrirrotor com uma placa controladora OSPs aplicada. Os principais componentes deste sistema são:

- Motor Brushless;
- Hélice;
- Eletronic Speed Controller (ESC);
- Sistema Rádio Controle;
- Bateria;
- Controladora.

4.1 Motor Brushless (Sem escova)

Os motores síncronos de imã permanente (do inglês Permanent-Magnet Synchronous Motor, PMSC), ou seja, que não possuem escorregamento como os motores de indução, são classificados em motores Brushless AC (BACM) e motores Brushless DC (BDCM). Em ambos, o enrolamento de armadura polifásico (sendo o de três fases o mais frequente) está localizado nas ranhuras do estator e o rotor com imã permanente serve como sistema de excitação. A diferença entre os dois tipos está na forma de onda da corrente aplicada ao enrolamento de armadura e a distribuição do fluxo magnético gerado. (MOURA, 2010; TRINDADE, 2009).

Sendo mais frequente o uso do BDCM nos quadrirrotores, a discussão será restringida a apenas esta classificação, que possui como vantagens em relação aos motores CC convencionais e motores de indução monofásicos as seguintes características:

- Razão maior entre torque e tamanho do motor;
- Peso menor;
- Melhor relação Torque x Velocidade;
- Melhor resposta dinâmica;
- Maior eficiência;
- Menor ruído;
- Grandes faixas de velocidade.

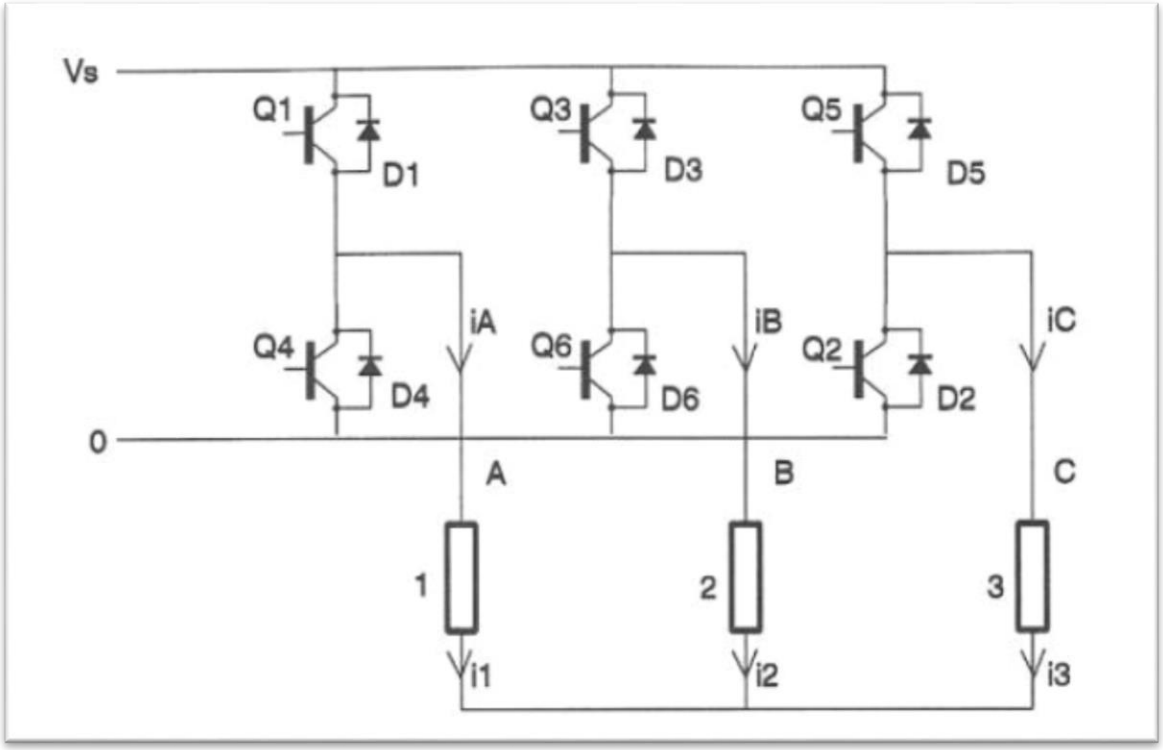
A maior desvantagem/dificuldade dos BDCM, inclusive em relação aos BACM e não só entre os motores CC convencionais e motores de indução monofásicos, está na comutação eletrônica necessária, que por um lado não possui o desgaste frequente presente nos comutadores mecânicos, porém por outro pode-se elevar a complexidade e valor do sistema devido a presença do inversor eletrônico e aos sensores de posição. (MOURA, 2010; TRINDADE, 2009).

A comutação deve ocorrer no instante em que a posição angular do rotor é tal que o fluxo magnético do rotor se alinha ao fluxo do estator. Os BDCM utilizam a realimentação direta da posição angular do rotor, através de sensores de posição do tipo Hall, de modo que a corrente de armadura seja comutada entre as fases do motor em sincronismo com a posição do rotor. Os comutadores eletrônicos utilizados nesse

processo podem ser constituídos por transistores bipolares de porta isolada, transistores de efeito de campo ou tiristores em comutação forçada. (TRINDADE, 2009).

Considerando um BDCM com três fases, conectado tanto em delta (Δ), quanto estrela (Y), utiliza-se como inversor uma ponte inversora trifásica com seis transistores acionados como chaves e seis diodos de roda livre para proteger as chaves na abertura. (TRINDADE, 2009). A Figura 9 demonstra o circuito de ponte inversora trifásica para uma ligação estrela.

Figura 9 - Ponte Inversora Trifásica com ligação em estrela.



Fonte: HENDERSHOT; MILLER (1994).

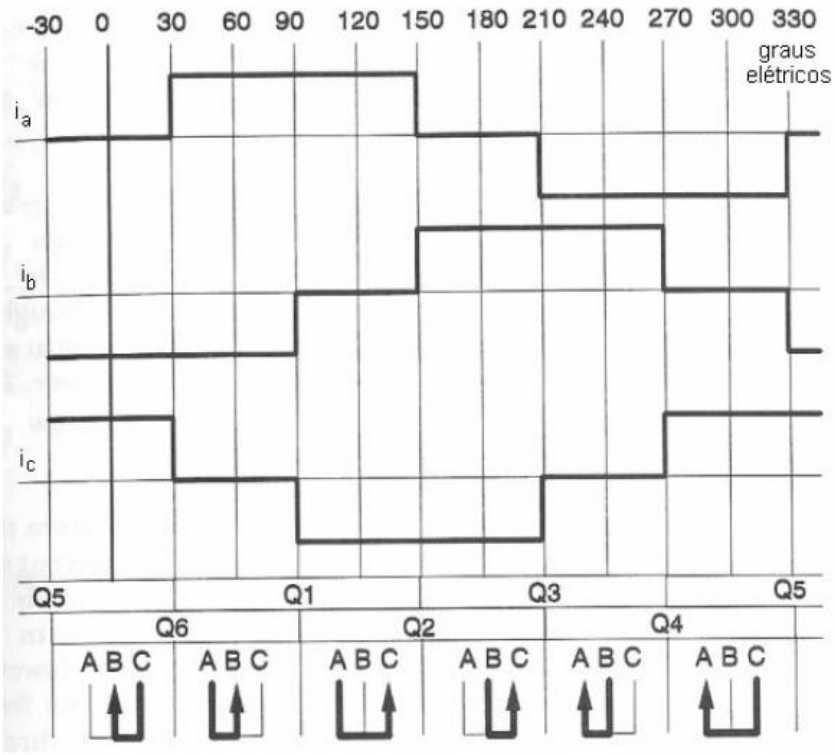
Em cada comutação, alterna-se quais das fases é conectada ao terminal positivo da fonte, terminal negativo da fonte e em aberto (alta impedância). Isso ocorre a cada 60˚ elétricos, devido à informação realimentada pelos sensores Hall. A partir do momento em que é fechada, a chave permanece neste estado por 120˚ elétricos. (MOURA, 2010; TRINDADE, 2009).

O número de pares de polos do rotor determina a quantidade de ciclos elétricos necessários para se completar uma rotação inteira do motor. (MOURA, 2010). Um motor de apenas um par de polos necessita de 6 passos de 60˚ elétricos

para completar um ciclo elétrico e uma rotação, já um motor de dois pares de polos necessita de 12 passos de 60˚ elétricos para completar dois ciclos elétricos e uma rotação.

A Figura 10 ilustra um ciclo elétrico com as formas de onda das correntes de cada fase, o período de condução de cada chave da Figura 9 e sentido da corrente nas bobinas do motor.

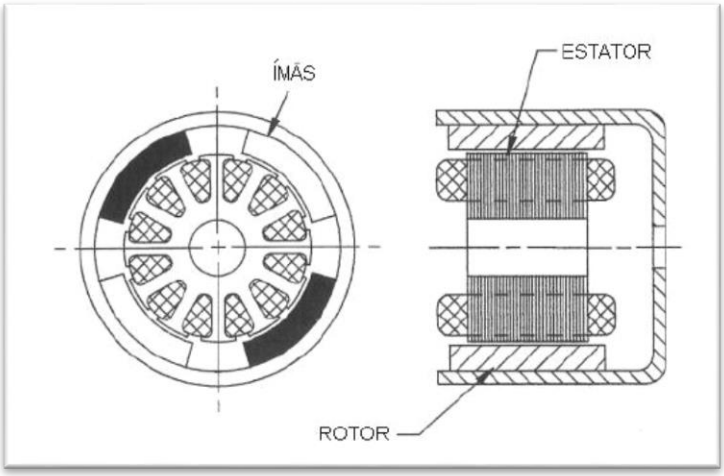
Figura 10 – Correntes em cada fase na comutação eletrônica.



Fonte: HENDERSHOT; MILLER (1994).

Dentro da classificação dos BDCM, há diversas configurações, dependendo do número de fases, ranhuras, polos, e principalmente, de acordo com o tipo de rotor. Quando se fala em quadrirrotores, procuram-se motores de velocidades altas e constantes, portanto o motor de rotor externo deve ser o escolhido. (TRINDADE, 2009). A Figura 11 ilustra a configuração de um BDCM com rotor externo.

Figura 11 – BDCM de rotor externo.



Fonte: HENDERSHOT; MILLER (1994).

Motores com rotor externo geralmente utilizam outra técnica para sensoriar a posição angular do rotor. Estes possuem a informação de velocidade e direção de rotação através da tensão gerada pela força contraeletromotriz.

A constante de RPM KV é expressa em RPM/Volt, e fornece a força contraeletromotriz gerada no motor a certa velocidade. Ou seja, a constante informa quantos volts o motor gera quando este rotaciona. Quanto mais rápido ele rotacionar, maior essa tensão gerada. Quando se aplica tensão no motor, este aumenta sua velocidade até que a tensão gerada pelo próprio motor se iguale a tensão aplicada. Pode-se equacionar a força contraeletromotriz medida em duas fases a partir da equação [1]. (D'AVILA et al., 2011).

$$E_f = \frac{n}{k_n} \qquad [1]$$

Onde, n é a velocidade de rotação do motor em rpm e k_n é a constante da velocidade, expressa em rpm/V.

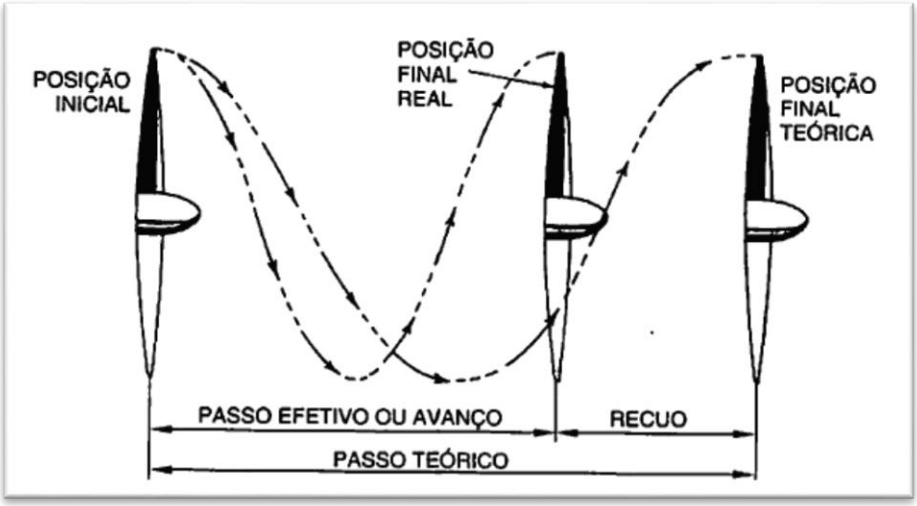
Quando não há carga, a velocidade rotacional é igual à esperada para uma certa tensão aplicada, porém quando se começa a aplicar carga, a velocidade rotacional irá cair proporcionalmente.

4.2 Hélice

São os elementos do quadrirrotor, que, quando juntos aos motores geram o empuxo para criar sustentação e, portanto, realizar movimentos de atitude no quadrirrotor.

Devido às angulações presentes nas pás da hélice, esta deveria avançar uma determinada distância a cada rotação completa (passo teórico), porém, como o ar é um meio viscoso, a distância percorrida é menor (passo efetivo). (HOMA, 2010). A Figura 12 mostra o recuo entre o passo teórico e o passo efetivo.

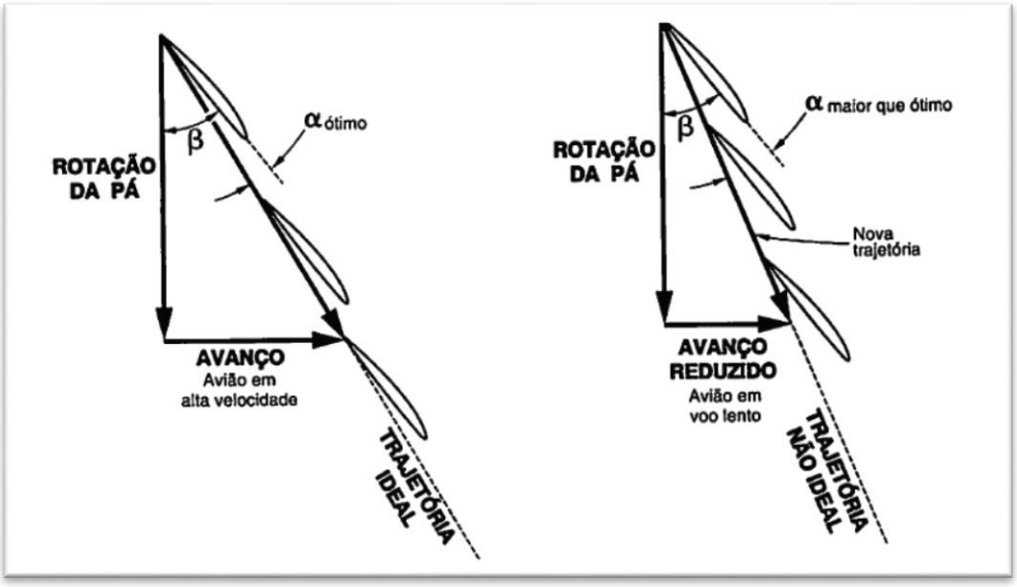
Figura 12 – Passo teórico e passo efetivo.



Fonte: HOMA (2010).

As hélices dos quadrirrotores possuem passo fixo, portanto estas possuem máximo rendimento somente nas condições de velocidade e rotação para que a mesmas foram confeccionadas. Quando se necessita de boa eficiência em voo lento, deve-se diminuir o ângulo de passo da hélice (β), sacrificando a velocidade máxima do sistema. Para entender melhor essa afirmação, a Figura 13 ilustra uma mesma hélice em duas situações diferentes, uma em voo de alta velocidade (avanço alto) e outra em voo com velocidade reduzida (avanço reduzido). O ângulo de ataque (α) ótimo é alcançado em alta velocidade de voo, necessitando diminuir β para melhor rendimento em velocidade reduzida de voo. (HOMA, 2010).

Figura 13 – Ângulo de ataque de uma hélice para velocidades distintas.



Fonte: HOMA (2010).

As hélices são as causadoras de cargas no motor, quanto maior a velocidade, maior a carga criada.

4.3 Eletronic Speed Controller (ESC)

O controlador eletrônico de velocidade (do inglês Eletronic Speed Controller, ESC) é um circuito eletrônico no qual o propósito é variar a velocidade de um motor elétrico e sua direção de rotação. Este varia a taxa de chaveamento da rede de transistores de efeito de campo (FETs), ou seja, este é o circuito responsável pela comutação eletrônica dos motores, apresentada na Figura 9.

O barulho característico dos BDCM é resultado da alta velocidade de comutação desses FETs, bem perceptível em baixas rotações. Estes FETs permitem um controle de velocidade do motor mais preciso, suave e eficiente do que outros controles de velocidade mecânicos mais antigos.

Muitos ESCs mais modernos possuem um circuito eliminador de bateria (do inglês Battery Eliminator Circuit, BEC) incorporado para regular uma tensão estável destinada a alimentação de receptores e servos motores. Isto remove a necessidade de carregar uma bateria extra no modelo quadrirrotor. Estes BECs podem ser lineares,

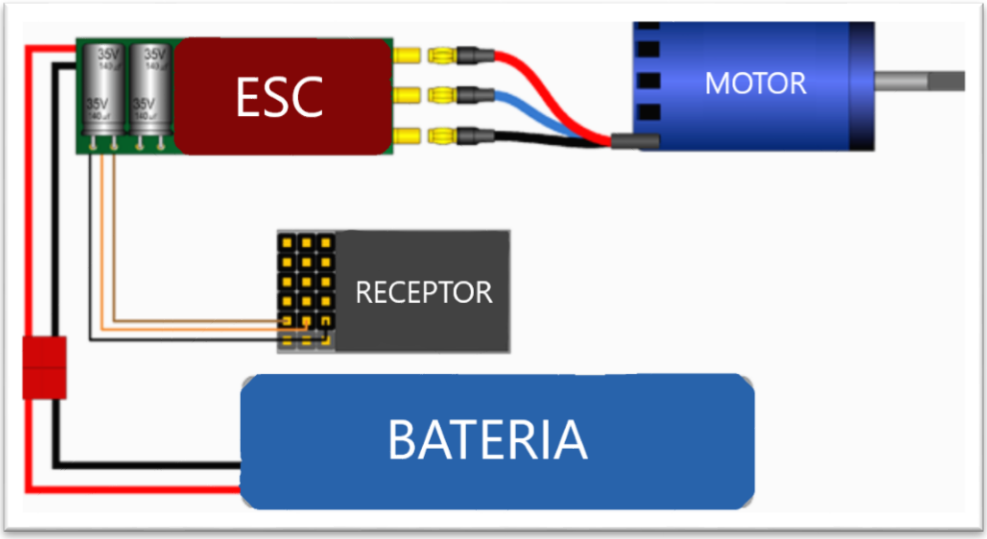
possuindo apenas um resistor divisor de tensão, ou chaveados, reduzindo a tensão nominal para a tensão média de chaveamento.

A ação de aumentar ou diminuir a velocidade do motor é entendida pelo ESC através de um pulso PWM. O pulso varia linearmente de 1ms a 2ms, sendo 1ms para o motor em estado estacionário e 2ms para velocidade máxima.

Após ler o pulso PWN e interpretar a amplitude de velocidade de rotação desejada, o ESC, alimentado por uma fonte de corrente contínua, cria 3 fases de corrente alternada, respeitando a sequência de chaveamento do circuito de ponte inversora trifásica. Em um determinado passo de um ciclo elétrico, duas fases estarão conectadas, e a fase remanescente gerará uma tensão proporcional a velocidade em que o motor está girando, tensão esta proveniente da força contraeletromotriz. Essa tensão é utilizada para sensoriar a velocidade e sentido de rotação do motor e, portanto, saber quando realizar a comutação das chaves.

A Figura 14 demonstra as ligações entre bateria, receptor, ESC e motor.

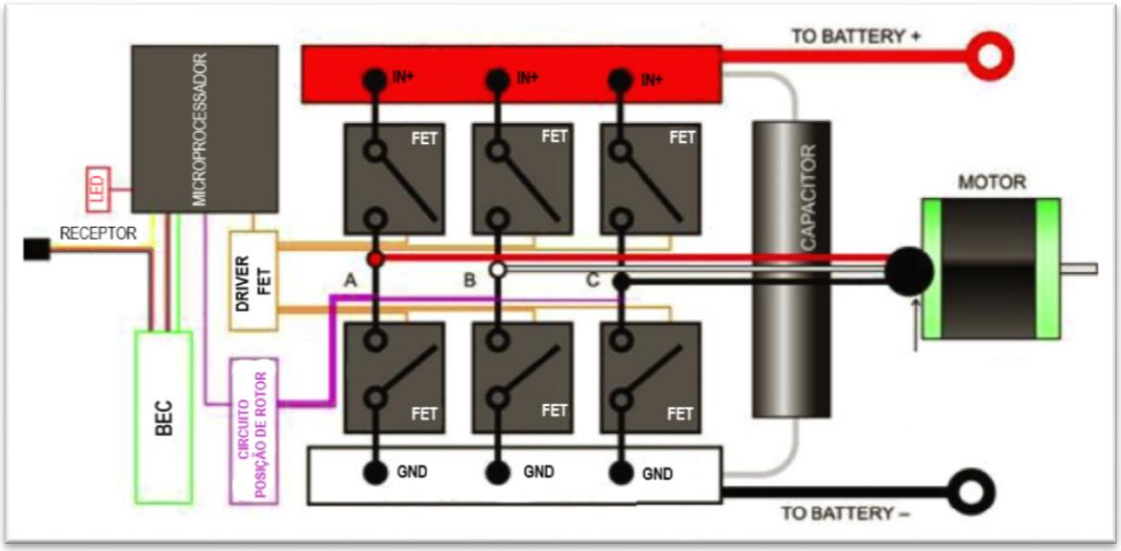
Figura 14 – Ligações entre bateria, receptor, ESC e motor.



Fonte: Adaptado de HOBBY KING (2018).

A Figura 15 demonstra o interior simplificado de um ESC.

Figura 15 – Interior simplificado de um ESC.



Fonte: Adaptado de HOBBY KING (2018).

4.4 Sistema Rádio Controle

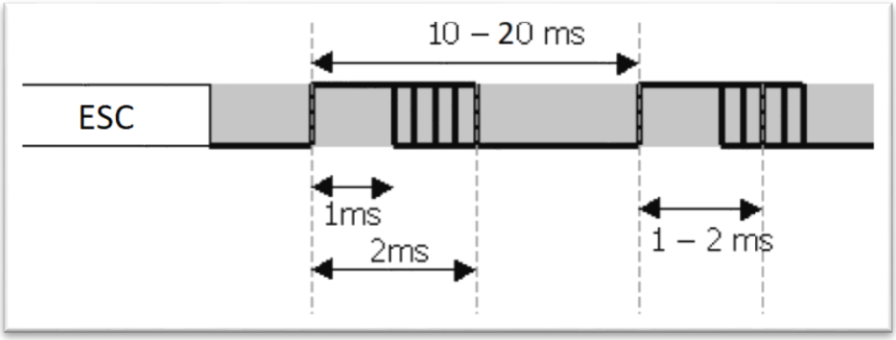
O sistema de rádio controle é composto por dois dispositivos principais, o transmissor (TX) e o receptor (RX). A técnica utilizada na transmissão de dados entre estes dois dispositivos pode variar de acordo com os vários modelos de transmissores presentes no mercado. Podendo utilizar modulação AM (modulação em amplitude), FM (modulação em frequência) ou PCM (modulação por código de pulso) em transmissores mais antigos que trabalham na faixa de (27~72MHz) e modulação DSSS (sequência direta de espalhamento do espectro) ou FHSS (espectro de difusão em frequência variável) em transmissores mais modernos de 2.4GHz. Os transmissores MHz são mais sensíveis a interferências, podendo dois transmissores no mesmo canal de frequência interferir um ao outro, característica essa, inibida nos transmissores 2.4GHz. (MOECKE, 2004; SCOFANO, 2003)

Um sistema de rádio controle pode conter vários canais, sendo 4 deles essências para a movimentação de um VANT, onde serão controlados os movimentos de arfagem, guinada, rolagem e a aceleração. Outros canais podem ser utilizados para acessar funções disponíveis no sistema, como a ativação ou não de um piloto automático.

Para a aplicação deste trabalho basta apenas entender qual o sinal gerado pelo receptor ao sistema microcontrolador presente nas placas de controle. A taxa de

atualização da maioria dos receptores é de 50Hz, e este envia pulsos de duração entre 1ms até 2ms. A Figura 16 demonstra a logística de um sinal provindo de um receptor com destino à um ESC. (SCOFANO, 2003).

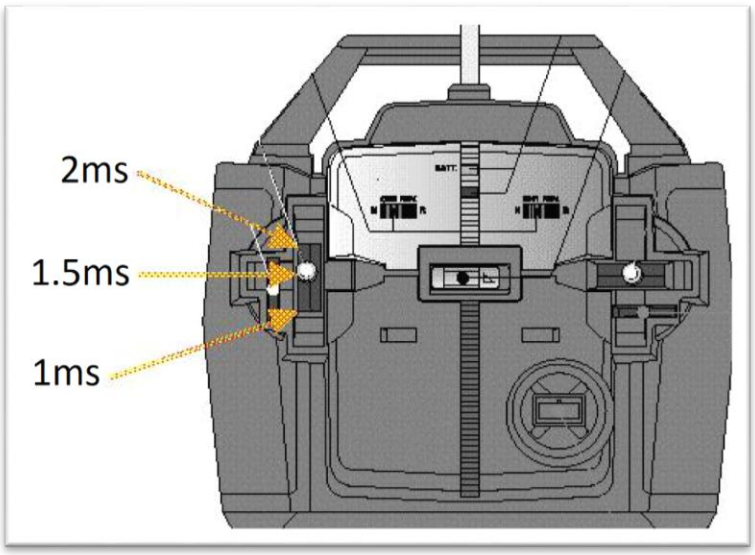
Figura 16 – Logística do sinal de um receptor.



Fonte: Adaptado de SCOFANO (2003).

Isolando um canal do sistema rádio controle, e analisando o pulso entre 1ms e 2ms com progressão linear, tem-se a relação destes valores com a posição dos sticks do transmissor através da Figura 17.

Figura 17 – Posições dos sticks do transmissor.



Fonte: Adaptado de SCOFANO (2003).

É bom ainda ressaltar, que em um sistema quadrirrotor, os canais de saída do receptor não são ligados diretamente aos ESCs, mas sim na placa controladora,

que irá, a partir de algoritmos com malhas de controle, gerenciar o sinal para cada ESC.

4.5 Bateria

Baterias são dispositivos que armazenam energia e facilitam a conversão de energia química para energia elétrica. No mercado, já há diversos tipos de baterias recarregáveis, porém as baterias de Li-ion (íon-lítio) sofrem destaque em diversas aplicações tecnológicas. (CUONG,2011). As características responsáveis por este destaque são:

- Os eletrodos de uma bateria de Li-ion são feitos de lítio leve (ânodo) e carbono (cátodo);
- Densidade de energia maior em comparação com outras químicas, como chumbo-ácido, NiCd (níquel-cádmio) ou NiMH (níquel-hidreto metálico);
- Taxa de auto descarga baixa, ou seja, as cargas permanecem por um longo período de tempo;
- Podem realizar centenas de cargas e descargas sem degradação significativa de sua capacidade.

Baterias de Li-Po (íon-polímero) possuem o mesmo processo químico que as células de Li-ion, mudando apenas o fato de que o eletrólito é em gel e o cátodo é a base de polímeros. (CUONG, 2011).

As Baterias de Li-Po geralmente possuem suas células conectadas em paralelo, sendo que cada célula possui uma tensão nominal de 3.7V. (NIKITA, 2017).

A corrente de saída, além de desempenhar papel importante na determinação de perdas internas de uma bateria, também é um parâmetro importante para analisar o desempenho desta. O termo utilizado para indicar a taxa na qual a bateria é descarregada é o C-Rate. (CUONG, 2011).

Para exemplificar, uma bateria de 5000mAh possui uma taxa de descarga de 10C, indicando a quantidade de corrente máxima que pode ser drenada continuamente pelos motores por:

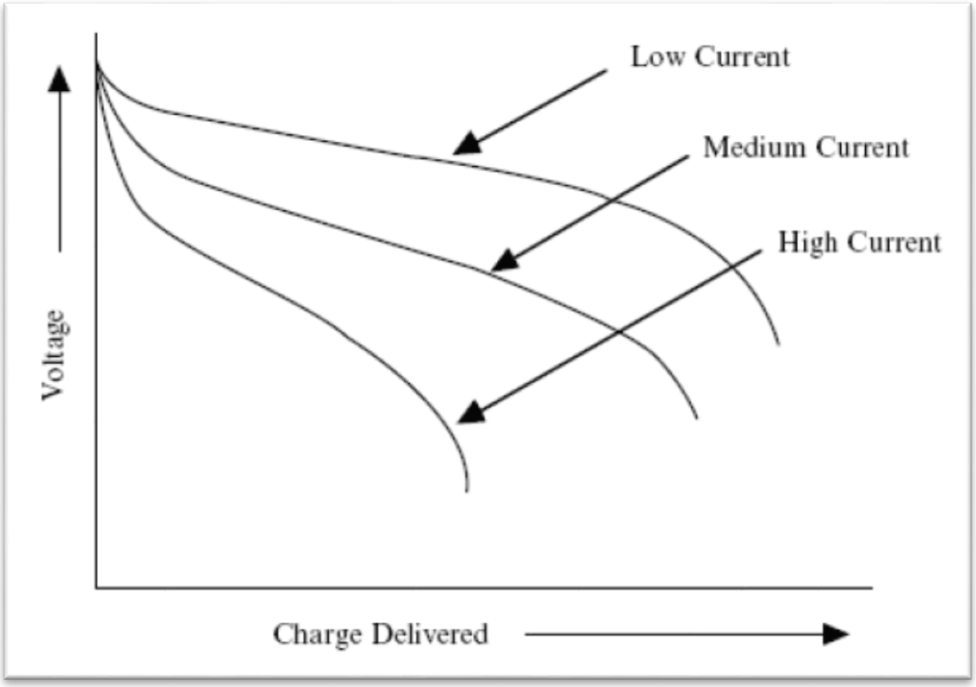
$$5000mAh \times 10C = 50A \quad [2]$$

A essa taxa de descarga máxima constante, a bateria teria uma autonomia de:

$$\frac{5000mAh}{50A} = 0.1h = 6\ min \quad [3]$$

A Figura 18 demonstra ainda, diferentes tipos de curva de descarga de uma bateria para diferentes tipos de taxa de descarga. Quanto maior a taxa, maior a corrente de saída.

Figura 18 – Influência da taxa de descarga sobre a densidade de corrente de uma bateria qualquer.



Fonte: CUONG (2011).

4.6 Controladoras

Quando se fala de controladoras, pode-se encontrar tanto OSPs, quanto projetos totalmente fechados, como a controladora NAZA desenvolvida pela DJI. Porém OSPs são totalmente adequados em um ambiente acadêmico, pois permitem

que os pesquisadores repliquem e ampliem os resultados de outros e providenciem uma linha de base para comparação entre várias abordagens, e tudo isso, utilizando sites de hospedagem comunitária totalmente acessíveis, garantindo um selo de licença pública geral (GPL). (KIM et al., 2012).

A placa controladora é um sistema embarcado, que além do microcontrolador, protagonista no processamento, possui inúmeros periféricos, dependendo do modelo em questão. A infinidade de abordagens entre um projeto e outro acarreta em inúmeras configurações diferentes, podendo existir:

- Sensores inerciais embutidos, como giroscópio, acelerômetro e magnetômetro ou um barramento de comunicação já pronto para receber uma IMU externa, geralmente I2C/TWI;
- Filtro Complementar e suas variações na fusão de sensores inerciais quando se utiliza um processador com menor capacidade ou Filtro de Kalman quando se utiliza um processador com maior capacidade;
- Conversor USB/Serial embutido para comunicação com softwares, GCS (do inglês Ground-Control Software) ou programas para desenvolvimento acadêmico, como o SIMULINK;
- Comunicação com GPS (geralmente USART), permitindo monitoramento de posição global ou, em modelos mais elaborados, a configuração de uma navegação pré-programada.

Além dos itens supracitados, o firmware das placas controladoras, em geral, realizam o tratamento dos dados do receptor, criando destes, referências para malhas de controle. Geralmente utilizam-se controles PID, havendo uma malha para cada movimento do quadrirrotor, ou ainda mais se houver mais de um modo de voo oferecido, como o modo acrobático e o modo estabilizado. O sinal de controle é passado para os atuadores, no caso os motores, por meio de uma função, que contêm a dinâmica de movimentos do quadrirrotor. Por fim os novos ângulos gerados na correção são realimentados para uma nova iteração do controle

5. CONSTRUÇÃO

É substancial o estudo e entendimento de todos os componentes do sistema quadrirrotor, tanto para suas instalações corretas, quanto para modificações no firmware, sendo necessária a compreensão da interação entre componente e microcontrolador.

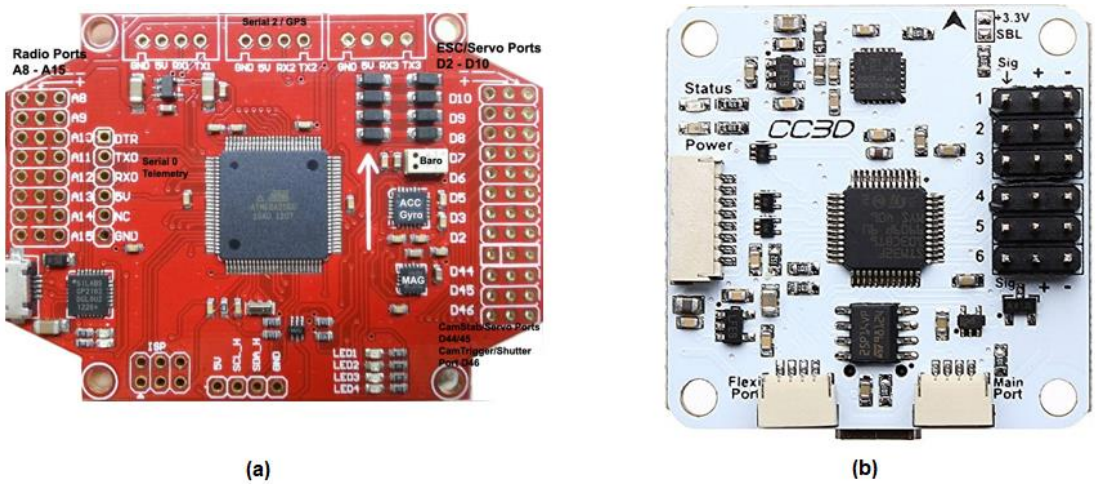
A fim de ambientar-se entre os sistemas quadrirrotores e, principalmente, entre as controladoras OSPs, foram construídos dois sistemas diferentes, um com orientação em cruz (+) e outro com orientação em (X), ambos simétricos.

Foram utilizados frames já disponíveis no mercado, e toda adaptação necessária foi realizada com a produção em medida de peças através do sistema de controle numérico computadorizado (do inglês Computer Numeric Control, CNC).

Tendo a estrutura pronta, foi realizada a instalação dos componentes eletrônicos, acomodando todo o sistema embarcado. Foram utilizadas duas controladoras diferentes em cada frame, a *MultiWii Pro* e a *CC3D OpenPilot*, ambas em detalhes na Figura 19. Observa-se que a controladora *MultiWii Pro* é integrada com um *ATmega2560* e a *CC3D OpenPilot* com um *STM32*.

Ao final da construção dos dois sistemas, foi analisado qual o mais propício a ser aprofundado no desenvolver deste trabalho. Considerando a menor dependência de software, maior facilidade de leitura do firmware, menor complexidade no filtro de fusão dos sensores (IMU) e maior facilidade na programação do microcontrolador, foi então escolhido o sistema com a controladora *MultiWii Pro*.

Figura 19 - Controladoras. (a) *MultiWii Pro*. (b) *CC3D OpenPilot*



(a) (b)

Fonte: Autoria Própria.

Em sequência, segue-se a apresentação de cada um dos quadrirrotores desenvolvidos com os dispositivos utilizados em cada.

5.1 Quadrirrotor (X)

Os materiais utilizados para o quadrirrotor com orientação X são:

- 1x MultiWii Pro – ATMega2560;
- 4x RC Timer - BC-2208/12 – 1800KV;
- 1x EMAX Multirotor 4IN1 ESC 4x 25A;
- 1x SPEKTRUM AR6110e 6-Channel Receiver;
- 1x LP5DSM 2.4GHz 5-Channel Transmitter;
- 2x Transmetic Li-Po 2200mAh 11.1V 35c;
- 1x Alware Quad Flyer Spider 500X;
- 4x Hélices 10x4.5”.

As duas baterias foram conectadas em paralelo, visando aumentar a corrente de descarga máxima, e assim o tempo de voo máximo.

As conexões em paralelo entre as baterias e ESC foram feitas através de uma placa de distribuição. A Figura 20 demonstra o quadrirrotor já finalizado.

A adaptação necessária do frame para o encaixe das baterias e placa distribuidora foi realizada com a produção em medida de peças através da máquina CNC. A peça confeccionada pode ser encontrada no APÊNDICE A.

O módulo GPS foi instalado e testado, porém este não terá influência neste trabalho.

Figura 20 – Quadrirrotor (X) com controladora *MultiWii Pro.*

Fonte: Autoria Própria.

5.2 Quadrirrotor (+)

Os materiais utilizados para o quadrirrotor com orientação + são:

- 1x CC3D – STM32f103c8t6
- 4x EMAX - 2213 – 935KV
- 4x EMAX Simon Series 25A
- 1x SPEKTRUM AR6100e 6-Channel Receiver
- 1x HK-T6A V2 2.4GHz 6-Channel Transmitter
- 1x Lion Power Li-Po 5200mAh 11.1V 30c
- 1x CSL-x450
- 4x Hélices 10x4.5”

As conexões em paralelo entre a bateria e ESC foram feitas através de uma placa de distribuição. A Figura 21 demonstra o quadrirrotor já finalizado.

Figura 21 – Quadrirrotor (+) com com controladora *CC3D.*

Fonte: Autoria Própria.

Foi necessário fazer uma peça para o encaixe dos motores e outra para o encaixe da bateria e placa controladora. Foi utilizada a máquina CNC, e as medidas das peças podem ser encontradas no APÊNDICE B.

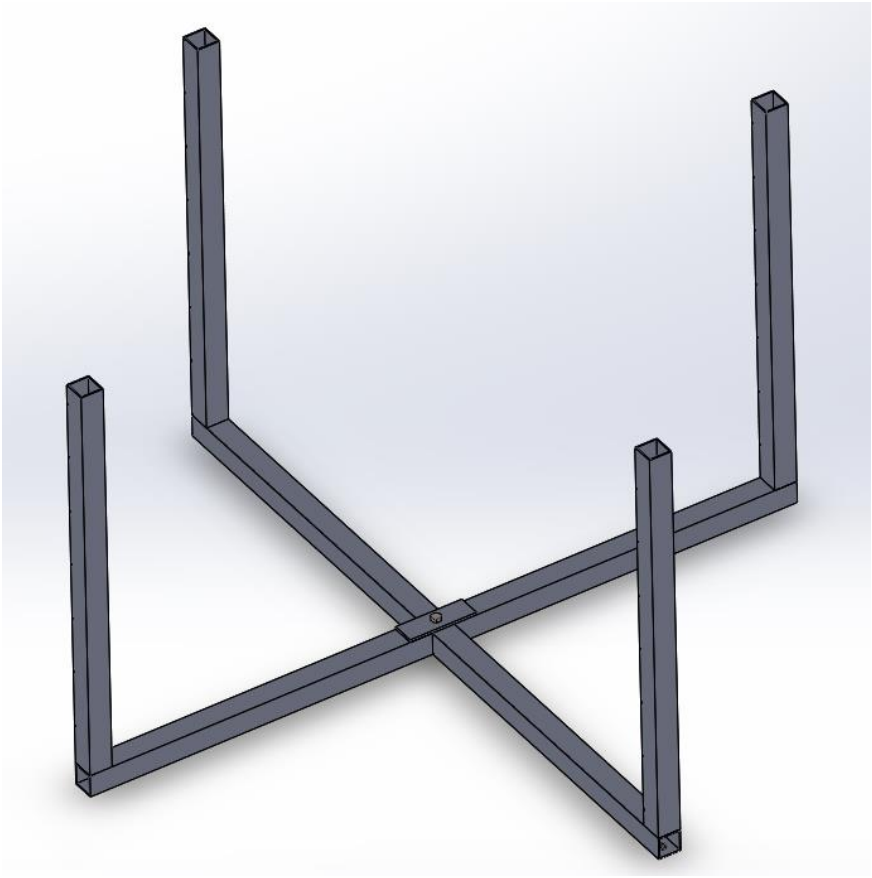
6 ESTRUTURA PARA SINTONIA

O controle clássico, seja ele Proporcional (P), Proporcional Derivativo (PD) ou Proporcional Integral Derivativo (PID), necessita de ajustes em seus ganhos para atingir a estabilidade requerida. Neste processo de sintonização, pode-se ter um sistema totalmente instável, e num quadrirrotor, se este processo é feito em voos livres, há o risco de ocasionar danos irreparáveis na estrutura provenientes de quedas ou aterrissagens bruscas. Deve-se então, construir uma estrutura de testes, onde o quadrirrotor fique fixo, sem riscos de colisões físicas.

Com o quadrirrotor na estrutura, pode-se observar a reposta do mesmo de acordo com as oscilações apresentadas. Variando os ganhos, e apresentando várias configurações diferentes, busca-se a estabilidade utilizando algum método de sintonização.

A Figura 22 demonstra a estrutura para sintonia projetada exclusivamente para este trabalho.

Figura 22 – Estrutura para sintonia projetada.



Fonte: Autoria Própria.

O quadrirrotor é preso na estrutura por elásticos em uma adaptação feita em seu frame. A posição e o número de elásticos usados podem variar de acordo com o movimento a ser calibrado, rolagem ou arfagem.

A estrutura foi dimensionada para quadrirrotores maiores ao usado deste trabalho, garantindo certa margem para oscilações e uma instabilidade do sistema. A Figura 23 demonstra a estrutura já confeccionada com o quadrirrotor preso nesta.

Figura 23 – Estrutura para sintonia confeccionada.

Fonte: Autoria Própria.

As dimensões da estrutura estão apresentadas no APÊNCIDE C.

7 O MULTIWII PRO

A controladora de voo *MultiWIii Pro* é o OSP escolhido a se detalhar neste trabalho. Apesar das vantagens que levaram a escolha desta controladora, há um grande contra. Como é um projeto em que o desenvolvimento foi cancelado, o mesmo não possui documentação detalhada, que é crucial na análise de abordagens e aplicações de controle. Portanto, foi necessário um estudo extremamente detalhado de todo firmware, a fim de entender todas as abordagens passo a passo, identificar as dependências e então adaptar o firmware com o objetivo do trabalho.

As modificações do firmware foram feitas a fim de minimizar o extenso código a apenas o necessário, retirando vários modos de voos, funções extras, algoritmos para componentes de telemetria desnecessários em uma análise laboratorial, todo o sistema de GPS (desde a comunicação até o sistema de navegação programada) e programações para configurações diferentes a utilizada.

A simplificação do firmware, com apenas os elementos necessários para análise de controle, juntamente com toda sua documentação detalhada e didática feita e apresentada a partir do capítulo 9, garantem a continuidade da pesquisa e de uma plataforma de desenvolvimento e testes de controles para quadrirrotores.

7.1 Detalhes da *MultiWii Pro*

A *MultiWii Pro* apresenta os seguintes sensores:

- Acelerômetro - BMA180
- Giroscópio - ITG32-00
- Magnetômetro - HMC5883L
- Barômetro - BMP085

O filtro de fusão utilizado para os sensores inerciais é o filtro complementar linear, sendo aplicado aos pares acelerômetro+giroscópio, para estimar os ângulos de rolagem e arfagem, e magnetômetro+giroscópio, para estimar o ângulo de guinada.

A versão do firmware escolhida para aplicar os estudos e modificações é a MultiWii 2.1. Esta apresenta os seguintes blocos de código e bibliotecas:

- BUZZER
- EEPROM
- GPS
- IMU
- LCD
- LED
- MAIN
- OUTPUT
- RX
- SENSORS
- SERIAL
- config.h
- def.h
- tinygps.h

Os modos primários de voo presentes nessa versão são:

- **Acro Mode -** utiliza apenas os dados do giroscópio na realimentação do controle, ou seja, o sinal de referência do controle atua na velocidade em que o quadrirrotor rotaciona em determinado movimento. Os movimentos de rolagem, arfagem e guinada sofrem interferência neste modo.
- **Level Mode -** utiliza os dados do acelerômetro e giroscópio na realimentação do controle, sendo assim, o sinal de referência do controle é o ângulo desejado para determinado movimento. Apenas os movimentos de rolagem e arfagem são interferidos por esse modo.
- **Mag Mode -** quando ativado, gera parte do sinal de referência para a malha de controle do movimento de guinada no modo Acro Mode. Este sinal é um proporcional ao erro do ângulo de guinada desejado (determinado quando ativa o modo) com o ângulo de guinada real.
- **Altitude Hold -** este modo, no instante que é ativado, guarda o valor atual de aceleração dos motores utilizando-o como referência, e

realimenta-se com o valor de altitude estimado advindo do barômetro. O sinal de controle desta malha de controle atua simultaneamente nos quatro atuadores. Este valor de referência é atualizado quando a aceleração advinda do stick, ou seja, quando o usuário acelerar, passar de uma determinada margem de diferença.

Dentre esses modos de voo, é importante dizer que o Acro Mode e Level Mode nunca podem atuar juntos, um possui uma malha de controle independente da do outro. Já Mag Mode e Altitude Hold podem atuar com qualquer outro modo de voo primário.

Os modos secundários de voo são:

- **Return to Home –** utiliza o GPS e magnetômetro para retornar ao ponto de inicialização. A estabilização de voo é feita a partir de um modo de voo primário (Acro Mode ou Level Mode).
- **Hold Position –** Mantém a posição atual utilizando o GPS e barômetro.
- **Passthru Mode –** Os sinais recebidos pelos receptores, são diretamente passados aos ESCs, não havendo qualquer influência do controle.
- **HeadFree Mode –** Não interfere no Acro Mode ou Level Mode, apenas mantém uma orientação de voo em relação ao ângulo de guinada, mantendo uma perspectiva 2D para os movimentos de rolagem e arfagem. É utilizado para pilotos novatos, que perdem a perspectiva da “frente” do quadrirrotor.
- **HEADADJ –** Seleciona uma nova orientação para o HeadFree Mode.

As funções extras são:

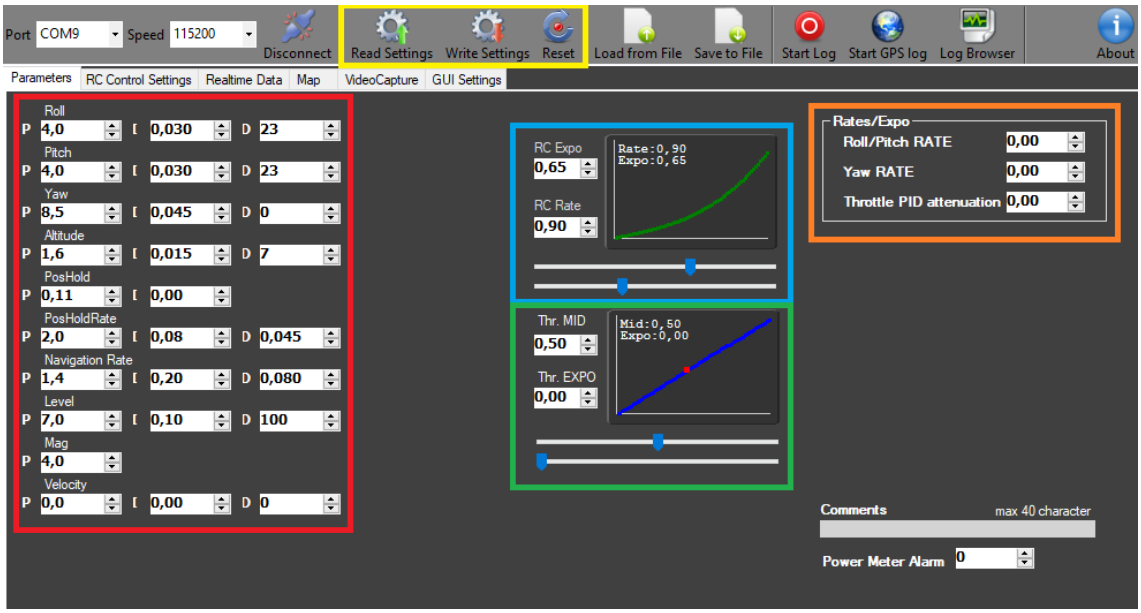
- **ARM** – Arma/Desarma o quadrirrotor. O quadrirrotor consegue aumentar sua aceleração ou potência apenas quando estiver armado.
- **LEDMAX** – Altera iluminação do LED do quadrirrotor.

- **LLIGHTS** – Ativa/Desativa as luzes de aterrisagem.
- **CAMSTAB** – Atua como um gimbal para a câmera do quadrirrotor.
- **CAMTRIG** – Gatilho para ativar ou desativar a câmera.
- **BEEPER –** Ativa/Desativa buzzer de indicação de nível da bateria.

7.2 Ground Control Software (GCS)

A maioria das controladoras OSPs possuem uma interface gráfica do usuário (do inglês Graphical-User-Interface) baseada no software GCS. A GUI do MultiWii Pro é dividida em 6 telas, porém o foco deste trabalho tem funcionalidade para apenas 3 delas.

Figura 24 – Ajuste de parâmetros pela GUI.



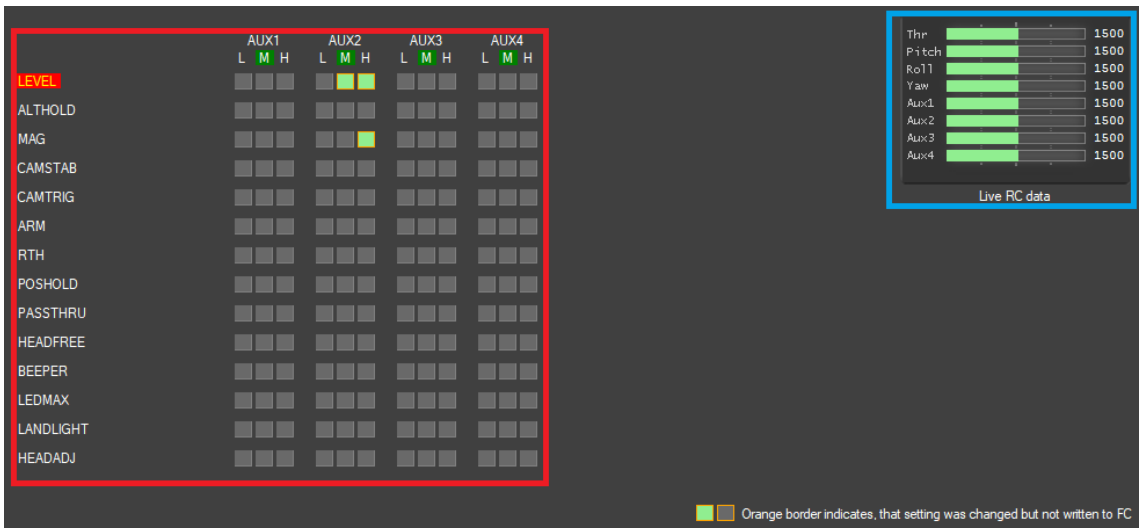
Fonte: Autoria Própria.

A Figura 24 demonstra a primeira tela. Esta é responsável por:

- **1 (vermelho) –** Ajuste dos ganhos das malhas de controle;
- **2 (azul) -** Configuração da curva de comando para os movimentos de rolagem e arfagem;
- **3 (verde) –** Configuração da curva de comando de potência dos motores.

- **4 (alaranjado) –** Configuração de atenuadores dos ganhos do controle de acordo com a aceleração e angulação. Quanto maior Roll/Pitch RATE e Yaw RATE, maior a atenuação dos ganhos para angulações maiores. Quanto maior Throttle PID, maior a atenuação dos ganhos de acordo com a aceleração do sistema.
- **5 (amarelo) –** Pode-se ler/escrever ou reiniciar as configurações gravadas na EEPROM.

Figura 25 – Configurações do rádio controle pela GUI para ativação de modos de voo primário, secundário e funções extras.

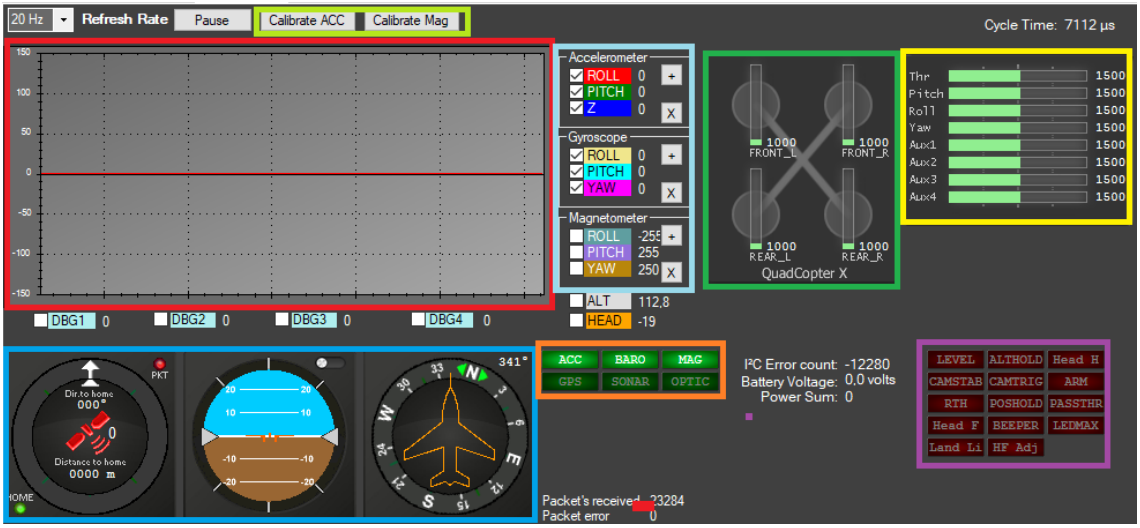


Fonte: Autoria Própria.

A Figura 25 demonstra a segunda tela. Esta é responsável por:

- **1 (vermelho)** – Configura qual o nível necessário dos canais auxiliares do sistema rádio controle para ativar os modos voo primários, secundários ou funções extras.
- **2 (azul) -** Demonstra qual o nível de cada canal do sistema rádio controle.

Figura 26 – Leitura de dados em tempo real pela GUI



Fonte: Autoria Própria.

A Figura 26 demonstra a terceira tela. Esta é responsável por:

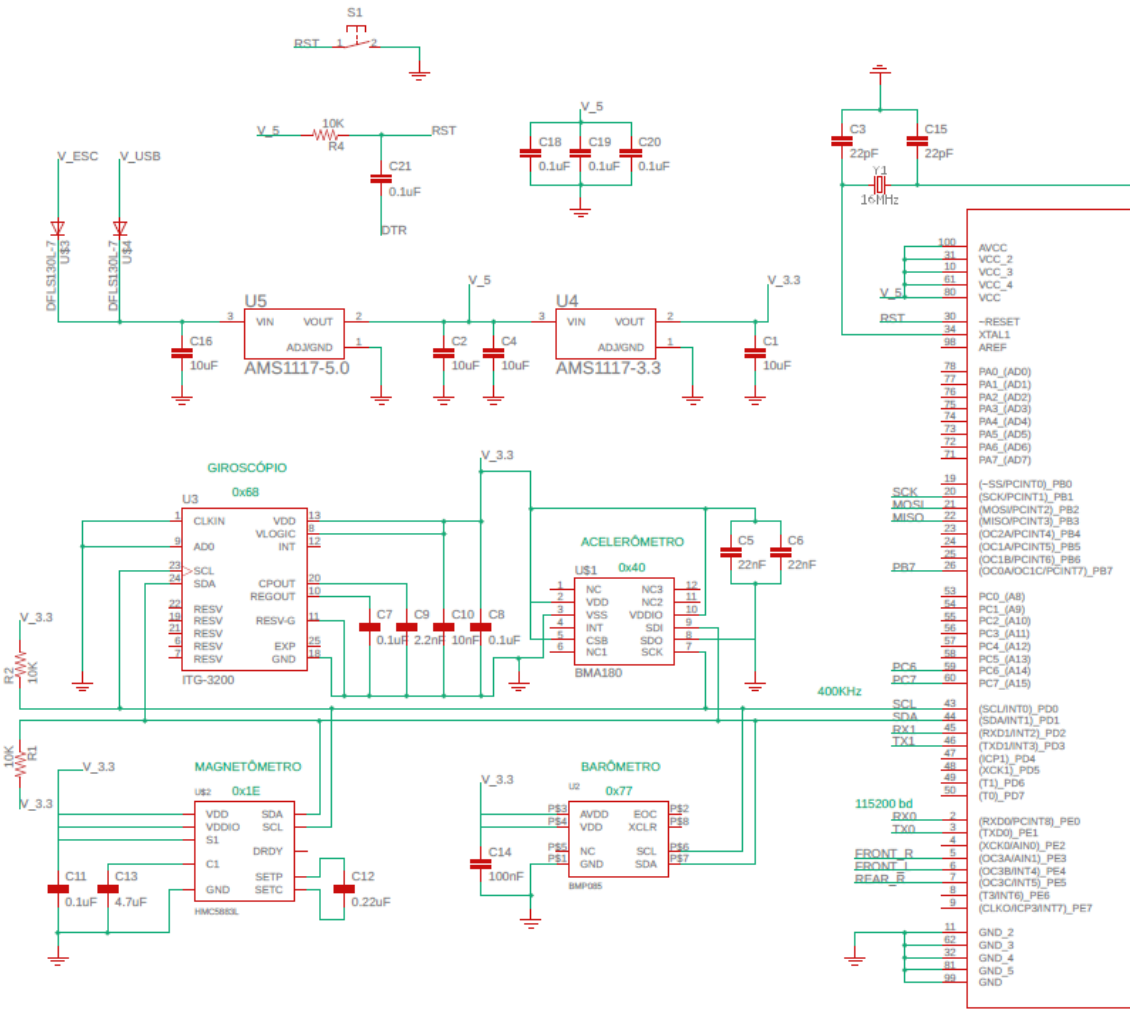
- **1 (vermelho) –** Gráfico mostrando os valores da saída do conversor A/D dos sensores inerciais.
- **2 (azul) –** Três visores, da direita para esquerda: o primeiro visor demonstra quantos satélites o GPS tem conexão, e qual a distância e orientação do local de inicialização. O segundo visor demonstra as angulações de rolagem e arfagem, usando um horizonte aeronáutico. O terceiro visor demonstra a angulação de guinada, utilizando uma bússola.
- **3 (verde) –** Demonstra a duração de pulso enviado para cada ESC dos motores.
- **4 (alaranjado) –** Demonstra quais os periféricos ligados na comunicação I2C ou USART.
- **5 (amarelo) –** Demonstra qual o nível de cada canal do sistema rádio controle.
- **6 (roxo) –** Indica em tempo real qual o modo de voo primário, secundário e funções extras ativados.
- **7 (azul claro) –** Indica os valores da saída do conversor A/D dos sensores inerciais.

- **8 (verde claro) –** Botões para ativar a calibração do acelerômetro ou giroscópio.

8 ESQUEMÁTICO

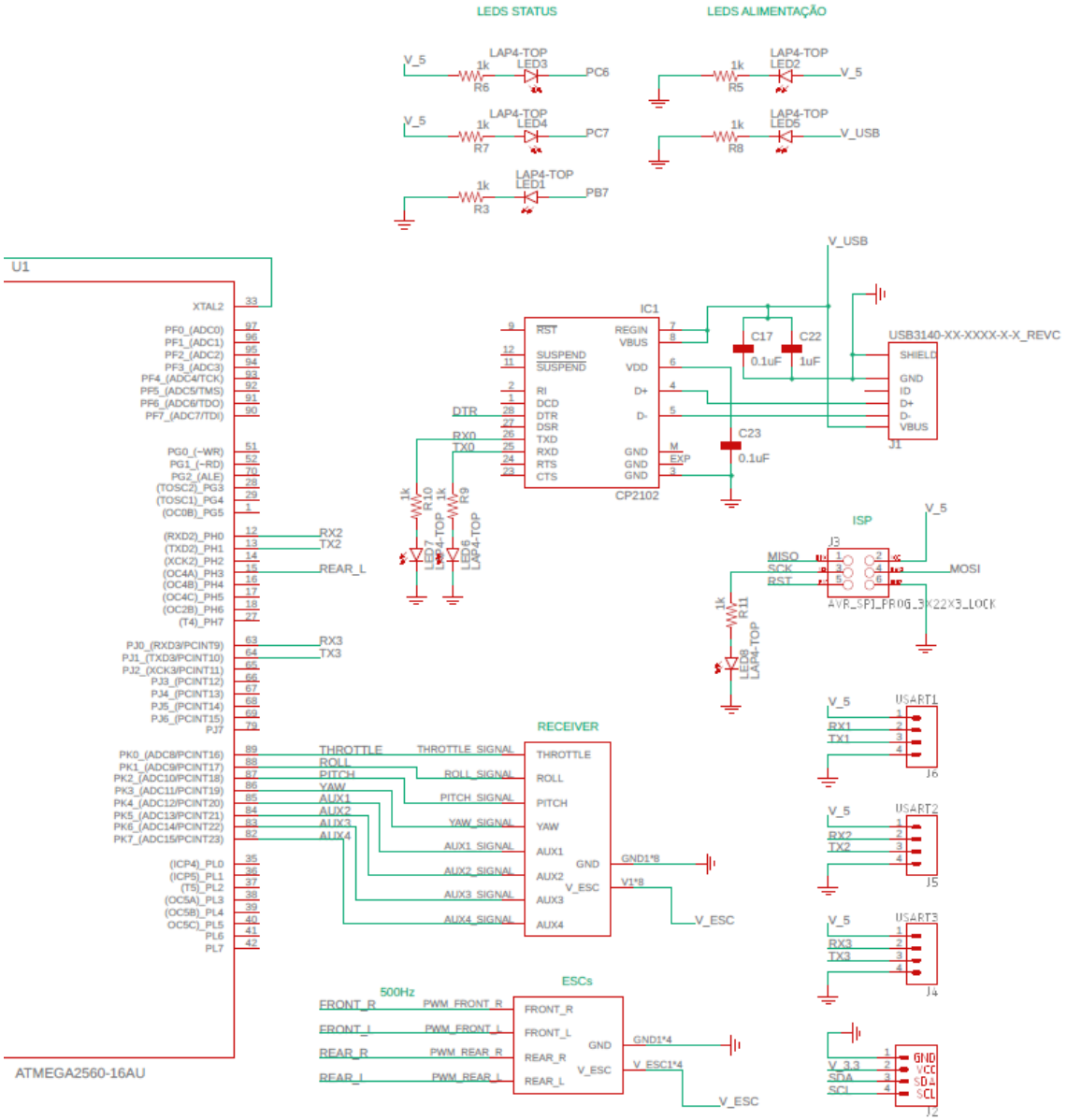
Como já enfatizado, o MultiWii Pro possui uma documentação muito limitada, nem havendo sequer o esquemático de sua placa eletrônica. Para contornar isso, e manter a continuidade de pesquisa em cima deste trabalho, foi desenvolvido um esquemático, tendo como base os circuitos integrados presentes na placa controladora e o estudo do firmware na versão 2.1. Os circuitos integrados e toda ligação do esquemático podem ser encontrados nas Figuras 27 e 28.

Figura 27 – Esquemático A.



Fonte: Autoria Própria.

Figura 28 – Esquemático B.



Fonte: Autoria Própria.

9 ESTUDO DO FIRMWARE

Para a leitura do firmware presente no microcontrolador, se faz necessário um ambiente de desenvolvimento integrado (do inglês Integrated Development Environment, IDE). No caso do MutiWii, foram utilizadas duas IDEs, a Arduíno e *a Atmel Studio*. Os firmwares são fornecidos em sites de hospedagem comunitária, como *Google Code* e *Github*, e são disponibilizadas como licença pública geral.

Faz-se o estudo detalhado do firmware MultiWii 2.1 visando entender as abordagens listadas abaixo:

- Recepção de dados fornecidos pelo receptor;
- Sinal PWM gerado para os ESCs;
- Comunicação serial entre componentes e entre o software de controle em solo (do inglês Ground Control Software, GCS);
- Aquisição de dados fornecidos pelos sensores inerciais;
- Aplicação da unidade de medida inercial (IMU);
- Leitura e escrita na memória EEPROM;
- Aplicação do controle embarcado.

Como a controladora é para uso generalizado, foram feitas modificações autorais no código fonte, visando a simplificação para o uso específico. Foi retirada a biblioteca “tinygps.h” e os blocos de código “BUZZER”, “GPS”, “LCD” e “LED”. Toda as dependências que esses blocos de código geravam em outros blocos foram removidas ao máximo. Todos os modos de voos secundários foram retirados, e apenas a função “ARM” das funções extras foi mantida.

Os capítulos em sequência fazem a apresentação do estudo feito nas bibliotecas e blocos de código remanescentes. Este é o estudo do firmware já modificado.

10 CONFIG.H

A biblioteca “config.h” seria, pressupostamente, o único bloco de código, na qual o usuário teria acesso. E neste, realizar as modificações para configurar de acordo com o sistema desejado.

Esta biblioteca foi modificada e reduzida, a fim de focar apenas no sistema, modos de voo e funções utilizadas neste trabalho. A Tabela 2 demonstra os macros definidos, realizando as configurações necessárias, que serão utilizados ao decorrer de outros blocos de códigos apresentados aqui neste capítulo.

Tabela 2 – Macros da biblioteca config.h

QUADX	-	Definição da estrutura utilizada para controle, no caso um quadrirrotor orientado em X
MINTHROTTLE	1050	Valor mínimo permitido de aceleração enviado para o ESC (quando armado).
MAXTHROTTLE	1850	Valor máximo permitido de aceleração enviado para o ESC (quando armado).
MINCOMMAND	1000	Valor enviado para o ESC quando o quadrirrotor não está armado.
I2C_SPEED	100000	Velocidade geral da comunicação I2C. (este valor pode ser alterado de acordo com o sensor).
FFIMUv2	-	Indica quais os sensores utilizados na IMU.
YAW_DIRECTION	1	Indica qual o sentido de rotação para correção do ângulo de guinada.

ALLOW_ARM_DISARM_ VIA_TX_YAW	-	Habilita a armar ou desarmar por meio do canal de guinada do rádio controle.
SERIA_COM_SPEED	115200	Velocidade da comunicação USART (baud rate).
MAG_DECLINATION	-19.48	Declinação magnética calculada.
MIDRC	1500	Ponto médio dos canais do receptor.
ESC_CALIB_LOW	MINCOMMAND	Valor mínimo para calibração dos ESCs.
ESC_CALIB_HIGH	2000	Valor máximo para calibração dos ESCs.
ESC_CALIB_ CANNOT_FLY	-	Habilita calibração dos ESCs.

Fonte: Autoria Própria.

11 DEF.H

A biblioteca “def.h”, de acordo com os macros definidos na biblioteca “config.h”, realiza as definições necessárias para o tipo de placa em uso.

As definições, também realizadas com macros, são feitas para o uso do microcontrolador ATMega2560, - uma simplificação realizada, já que o firmware original apresenta definições para uma infinidade de microcontroladores e placas controladoras - e estas fazem referência para as operações e informações apresentadas na Tabela 3.

As definições, ou macros, não estão explicados com profundidade nesta secção, porém conforme serão apresentados em outros blocos de código, nos próximos capítulo, terão sua forma bem exemplificada.

Tabela 3 – Macros da biblioteca def.h

MEGA	-
LEDPIN_PINMODE	pinMode (13, OUTPUT); pinMode (30, OUTPUT);
LEDPIN_TOGGLE	PINB |= (1<<7); PINC |= (1<<7);
LEDPIN_ON	PORTB |= (1<<7); PORTC |= (1<<7);
LEDPIN_OFF	PORTB &= ~(1<<7); PORTC &= ~(1<<7);
I2C_PULLUPS_ENABLE	PORTD |= 1<<0; PORTD |= 1<<1;
I2C_PULLUPS_DISABLE	PORTD &= ~(1<<0); PORTD &= ~(1<<1);
STABLEPIN_PINMODE	pinMode (31, OUTPUT);
STABLEPIN_ON	PORTC |= 1<<6;
STABLEPIN_OFF	PORTC &= ~(1<<6);
THROTTLEPIN	0
ROLLPIN	1

PITCHPIN	2
YAWPIN	3
AUX1PIN	4
AUX2PIN	5
AUX3PIN	6
AUX4PIN	7
PCINT_PIN_COUNT	8
PCINT_RX_BITS	(1<<2),(1<<4),(1<<5),(1<<6), (1<<7),(1<<0),(1<<1),(1<<3)
PCINT_RX_PORT	PORTK
PCINT_RX_MASK	PCMSK2
PCIR_PORT_BIT	(1<<2)
RX_PC_INTERRUPT	PCINT2_vect
RX_PCINT_PIN_PORT	PINK
ISAR_UART	ISR(USART0_UDRE_vect)
ITG3200	-
BMA180	-
BMP085	-
HMC5883	-
ACC_ORIENTATION	accADC[ROLL] = -X accADC[PITCH] = -Y accADC[YAW] = Z
GYRO_ORIENTATION	gyroADC[ROLL] = Y gyroADC[PITCH] = -X gyroADC[YAW] = -Z
MAG_ORIENTATATION	magADC[ROLL] = Y magADC[PITCH] = -X magADC[YAW] = -Z
ACC	1
MAG	1
GYRO	1
BARO	1
GPS	0

SONAR	0
MULTITYPE	3
STANDARD_RX	-
NUMER_MOTOR	4
ALTITUDE_HOLD_THROTTLE_NEUTRAL_ ZONE	20

Fonte: Autoria Própria.

12 RX.INO

12.1 Interrupções

A interrupção, no conceito relacionado aos microcontroladores, é um processo onde um dispositivo externo ou interno pode interromper a execução de uma determinada tarefa do microcontrolador e solicitar a execução de outra. É um conceito indispensável para manter-se o alto desempenho de processamento e a eficácia de tempo. (LIMA e VILLAÇA, 2012).

As interrupções nos microcontroladores AVR são vetorizadas, ou seja, cada interrupção possui um endereço correspondente fixo na memória de programa. (LIMA e VILLAÇA, 2012). A Tabela 4 apresenta o endereço de cada interrupção na memória de programa do ATMega2560. Há uma ordem de prioridade das interrupções, quanto menor o endereço, maior sua prioridade.

Tabela 4 – Vetor, endereço e fonte das interrupções no ATMEGA2560

Vetor	Endereço	Fonte	Definição da Interrupção
1	$0000	RESET	Pino Externo, Powe-on Reset, Brown-out Reset e Watchdog Reset.
2	$0002	INT0	Interrupção externa 0
3	$0004	INT1	Interrupção externa 1
4	$0006	INT2	Interrupção externa 2
5	$0008	INT3	Interrupção externa 3
6	$000A	INT4	Interrupção externa 4
7	$000C	INT5	Interrupção externa 5
8	$000E	INT6	Interrupção externa 6
9	$0010	INT7	Interrupção externa 7
10	$0012	PCINT0	Interrupção 0 por mudança de pino
11	$0014	PCINT1	Interrupção 1 por mudança de pino
12	$0016	PCINT2	Interrupção 2 por mudança de pino
13	$0018	WDT	Estouro de temporizador Watchdog
14	$001A	TIMER2 COMPA	Igualdade de comparação A do TC2

15	$001C	TIMER2 COMPB	Igualdade de comparação B do TC2
16	$001E	TIMER2 OVF	Estouro do TC2
17	$0020	TIMER1 CAPT	Evento de captura do TC1
18	$0022	TIMER1 COMPA	Igualdade de comparação A do TC1
19	$0024	TIMER1 COMPB	Igualdade de comparação B do TC1
20	$0026	TIMER1 COMPC	Igualdade de comparação C do TC1
21	$0028	TIMER1 OVF	Estouro do TC1
22	$002A	TIMER0 COMPA	Igualdade de comparação A do TC0
23	$002C	TIMER0 COMPB	Igualdade de comparação B do TC0
24	$002E	TIMER0 OVF	Estouro do TC0
25	$0030	SPI, STC	Transferência serial completa - SPI
26	$0032	USART0 RX	USART0, recepção completa
27	$0034	USART0 UDRE	USART0, limpeza do registrador de dados
28	$0036	USART0 TX	USART0, transmissão completa
29	$0038	ANALOG COMP	Comparador analógico
30	$003A	ADC	Conversão ADC completa
31	$003C	EE READY	EEPROM pronta
32	$003E	TIMER3 CAPT	Evento de captura do TC3
33	$0040	TIMER3 COMPA	Igualdade de comparação A do TC3
34	$0042	TIMER3 COMPB	Igualdade de comparação B do TC3
35	$0044	TIMER3 COMPC	Igualdade de comparação C do TC3
36	$0046	TIMER3 OVF	Estouro do TC3
37	$0048	USART1 RX	USART1, recepção completa
38	$004A	USART1 UDRE	USART1, limpeza do registrador de dados
39	$004C	USART1 TX	USART1, transmissão completa
40	$004E	TWI	Interface serial TWI-I2C
41	$0050	SPM READY	Armazenagem na memória de programa pronta
42	$0052	TIMER4 CAPT	Evento de captura do TC4
43	$0054	TIMER4 COMPA	Igualdade de comparação A do TC4
44	$0056	TIMER4 COMPB	Igualdade de comparação B do TC4

45	$0058	TIMER4 COMPC	Igualdade de comparação C do TC4
46	$005A	TIMER4 OVF	Estouro do TC4
47	$005C	TIMER5 CAPT	Evento de captura do TC5
48	$005E	TIMER5 COMPA	Igualdade de comparação A do TC5
49	$0060	TIMER5 COMPB	Igualdade de comparação B do TC5
50	$0062	TIMER5 COMPC	Igualdade de comparação C do TC5
51	$0064	TIMER5 OVF	Estouro do TC5
52	$0066	USART2 RX	USART2, recepção completa
53	$0068	USART2 UDRE	USART2, limpeza do registrador de dados
54	$006A	USART2 TX	USART2, transmissão completa
55	$006C	USART3 RX	USART3, recepção completa
56	$006E	USART3 UDRE	USART3, limpeza do registrador de dados
57	$0070	USART3 TX	USART3, transmissão completa

Fonte: ATMEGA2560 Datasheet.

Na aplicação em linguagem C, se faz o uso da biblioteca avr/interrupt[6], que emprega nomes para os tratadores de interrupção. As interrupções são escritas como funções, que recebem em seu parâmetro o nome do tratador específico, relacionando diretamente a algum endereço da Tabela 4.

Todas as interrupções do AVR podem ser habilitadas ou desabilitadas individualmente, porém há também a possibilidade de habilitar ou desabilitar todas as interrupções de forma geral através do bit 7 do registrador de status (SREG). (LIMA e VILLAÇA, 2012). A Figura 29 ilustra as chaves de habilitação das interrupções de um ATMega 328, que possui menor número de portas que um ATMega 2560, porém o mesmo conceito de aplicação.

Tem-se a execução de um código em uma rotina, ao ocorrer uma interrupção, a CPU completa a instrução em andamento, carrega o endereço da próxima instrução que seria executada em uma pilha, e direciona-se para a posição de memória que corresponde à interrupção. O código da interrupção é executado até

[6] avr-libc, “Standard C library for AVR-GCC,” [Online]. Disponível: http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html. [Acesso em Dezembro 2018].

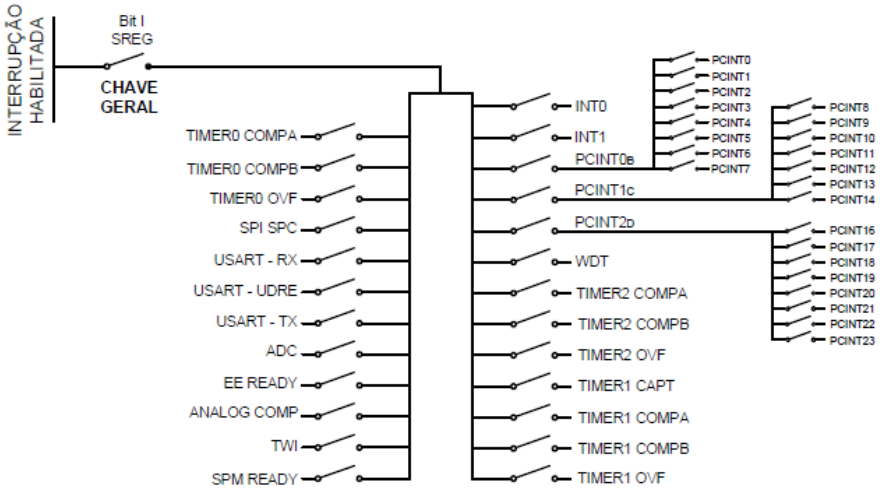
a identificação de um retorno de interrupção, onde então o endereço armazenado na pilha é executado, voltando para a rotina inicial. (LIMA e VILLAÇA, 2012).

É válido ainda ressaltar, que flags são utilizadas para a sinalização de uma ou mais interrupções durante uma execução de código de interrupção já vigente. O microcontrolador executa tais interrupções de acordo com sua ordem de prioridade (Tabela 4), porém, por regra, o AVR sempre executará uma instrução de programa principal antes de acionar qualquer uma dessas interrupções em espera. Essas sinalizações são essenciais já que o microcontrolador, ao atender uma interrupção, desabilita as interrupções através do registrador geral (SREG) e só as habilitam novamente após seu término. (LIMA e VILLAÇA, 2012).

Há também interrupções ausentes de sinalizações, onde basta existir a condição de interrupção quando a chave geral das interrupções estiver fechada. (LIMA e VILLAÇA, 2012). A Figura 29 demonstra diferentes tipos de interrupções, que são habilitadas por um bit específico no seu registrador de controle, e a interrupção é efetivamente ativada quando o bit 7 do registrador de estado (SREG) estiver em nível lógico alto. Compiladores mais modernos utilizam a função “sei()” para ligar a chave geral das interrupções, e a função “cli()” para desligá-la.

Quando se entra em uma interrupção, o registrador de estado (SREG) deve ter seu conteúdo salvo, pois os valores dos registradores podem ter sido alterados fora da sequência normal do programa. (LIMA e VILLAÇA, 2012).

Figura 29 - Chaves de habilitação das interrupções



Fonte: LIMA e VILLAÇA (2012).

12.2 Interrupções Externas

Caso haja a necessidade de monitorar algum evento externo, desde o acionamento de um botão, até uma variação de um sinal digital de um sistema externo, configura-se os pinos do microcontrolador para gerar interrupções externas.

No ATMEGA2560 as interrupções externas são desencadeadas através dos pinos INT7:0 e PCINT23:0, tendo diferenças na configuração entre estes dois tipos. Os INT7:0 podem gerar interrupções na borda de descida, subida, em nível lógico baixo ou na alteração do nível lógico, através da configuração dos registradores EICRA e EICRB. E os pinos PCINT23:0 podem apenas gerar interrupções através da variação de estado de nível lógico no pino. A interrupção, quando habilitada, é desencadeada mesmo com a configuração de saída no registrador de direção dos pinos, garantido a aplicação de geração de interrupção por software. (ATMEGA2560 Datasheet).

No firmware da controladora *MultiWii Pro* é utilizada a interrupção externa nos pinos PCINT23:16 para a recepção dos sinais de rádio frequência do receptor, havendo a necessidade de configurar os registradores PCICR, PCIFR, PCMSK2.

A explicação detalhada da configuração dos registradores será dada junto com a explicação do firmware relacionado à recepção dos sinais de rádio frequência.

12.3 Estudo do Firmware (rx.ino)

Através do rádio transmissor é feito uma transmissão em rádio frequência para o quadrirrotor, contendo dados referentes a posição dos sticks. No quadrirrotor, o receptor faz a recepção dos dados e transmite para a controladora um sinal de cada canal do rádio transmissor. Este sinal é um pulso PWM com duração de 1000µs para um stick em seu extremo menor e 2000µs para um sitck em seu extremo maior. A controladora faz a aquisição e processa o controle remoto de atitude, altitude e funções de voo.

Dando início a análise do firmware, tem-se a declaração de variáveis que serão utilizadas ao longo do bloco de código.

Declaração de variáveis

```
volatile uint16_t rcValue[8] = {1502, 1502, 1502, 1502, 1502, 1502,
 1502, 1502};

static uint8_t rcChannel[8]  = {ROLLPIN, PITCHPIN, YAWPIN,
THROTTLEPIN, AUX1PIN, AUX2PIN, AUX3PIN, AUX4PIN};

static uint8_t PCInt_RX_Pins[PCINT_PIN_COUNT] = {PCINT_RX_BITS};

//{1, 2, 3, 0, 4, 5, 6, 7}      -> macros definidos em def.h
```

Uma variável volátil[7] deve ser declarada quando seu valor pode ser mudado por algo além do controle da seção de código na qual ela aparece, como em seções associadas com interrupções. Uma variável estática[8] é usada para variáveis que são visíveis para apenas uma função, porém persistem entre chamadas de função, preservando seu valor. Estas variáveis estáticas são criadas e inicializadas apenas a primeira vez que sua função é chamada.

O vetor “rcValue[8]” armazena valores entre [1000µs, 2000µs] dos 8 canais do rádio controle. Todos os canais são inicializados com o valor de 1502, garantindo segurança ao inicializar o sistema. A posição 0 do vetor faz referência ao canal 0, posição 1 ao canal 1 e segue a lógica para os 8 canais.

A posição do vetor “rcChannel[8]” faz referência ao número do canal para determinada ação do rádio controle e o valor que cada posição carrega faz referência à porta física que este canal está conectado. Devido ao macro definido em “def.h” tem-se “rcChannel[8] = {1, 2, 3, 0, 4, 5, 6, 7} ”, sendo assim, o valor 1 faz referência à porta PK1, e assim por diante.

O vetor “PCInt_RX_Pins[8]”, recebe os bits que fazem referência as portas do PORTK, sendo carregado com a seguinte ordem {00000100, 00010000, 00100000, 01000000, 10000000, 00000001, 00000010, 00001000}, ou seja {PK2, PK4, PK5, PK6, PK7, PK0, PK1, PK3}.

A Tabela 5 relaciona os pinos físicos do ATMEGA2560 com os canais do receptor, de acordo com observações físicas.

---

[7] Arduino, “volatile” [Online]. Disponível: https://www.arduino.cc/reference/pt/language/variables/variable-scope--qualifiers/volatile/ [Acesso em Dezembro 2018].

[8] Arduino, “static” [Online]. Disponível: https://www.arduino.cc/reference/pt/language/variables/variable-scope--qualifiers/static/ [Acesso em Dezembro 2018].

Tabela 5 – Relação dos pinos do ATMEGA2560 com os canais do receptor e código.

Pino Físico	Comando	Pino de Interrupção	Canal do Código
PK0	THROTTLE	PCINT16	chan3
PK1	ROLL	PCINT17	chan0
PK2	PITCH	PCINT18	chan1
PK3	YAW	PCINT19	chan2
PK4	AUX1	PCINT20	chan4
PK5	AUX2	PCINT21	chan5
PK6	AUX3	PCINT22	chan6
PK7	AUX4	PCINT23	chan7

Fonte: Autoria Própria

configureReceiver()

```
void configureReceiver() {
  DDRK = 0;
  for(uint8_t i = 0; i < PCINT_PIN_COUNT; i++){
    PCINT_RX_PORT |= PCInt_RX_Pins[i];
    PCINT_RX_MASK |= PCInt_RX_Pins[i];
  }
  PCICR = PCIR_PORT_BIT;
}
// PCINT_RX_PORT = PORTK
// PCINT_RX_MASK = PCMSK2
// PCIR_PORT_BIT = 00000100  -> PCIE2
```

A função “configureReceiver()” é responsável por fazer a configuração dos registradores de interrupção externa. Em sequência, essa função declara as portas K como entradas, ativa os resistores de pull-up nos pinos PCINT23:16 (PK7:0), ativa todos os bits do registrador PCMSK2 e ativa o bit PCIE2 do registrador PCICR.

O registrador PCICR (Pin change Interrupt Control Register), possui o bit PCIE2 (Pin Change Interrupt Enable 2). Quando este bit está em valor lógico alto e o bit 7 do registrador de status (SREG) está ativado (Interrupção global habilitada), qualquer mudança em qualquer pino PCINT23:16 habilitado causará uma interrupção. Esta interrupção é referenciada pelo vetor de interrupção PCINT2, conforme identificado na Tabela 4.

O registrador PCMSK2 (Pin Change Mask Register 2) possui bits que habilitam individualmente a interrupção por mudança de pino nos pinos PCINT23:16.

Através da biblioteca avr/interrupt tem-se a função de interrupção com o tratamento PCINT2_vect como parâmetro.

ISR(PCINT2_vect)

```
ISR(PCINT2_vect) {
  uint8_t mask;
  uint8_t pin;
  uint16_t cTime,dTime;
  static uint16_t edgeTime[8];
  static uint8_t PCintLast;

  pin = PINK;
  mask = pin ^ PCintLast;
  sei();
  PCintLast = pin;
  cTime = micros();

  RX_PIN_CHECK(0,2);      //(PK2)    -> PITCH
  RX_PIN_CHECK(1,4);      //(PK4)    -> AUX1
  RX_PIN_CHECK(2,5);      //(PK5)    -> AUX2
  RX_PIN_CHECK(3,6);      //(PK6)    -> AUX3
  RX_PIN_CHECK(4,7);      //(PK7)    -> AUX4
  RX_PIN_CHECK(5,0);      //(PK0)    -> THROTTLE
  RX_PIN_CHECK(6,1);      //(PK1)    -> ROLL
  RX_PIN_CHECK(7,3);      //(PK3)    -> YAW
}
```

Toda vez que ocorrer uma mudança de estado em qualquer pino PCINT23:16, esta ISR (Interrupt Service Routine) será chamada, onde acontecerá a declaração de 6 variáveis (2 estáticas). A Tabela 6 descreve a função de cada variável dentro da interrupção.

Tabela 6 – Variáveis para a rotina de interrupção (PCINT2_vect)

Variável	Descrição
mask	Variável para verificar se houve mudança entre um valor atual e um valor antigo vindo do receptor.
pin	Variável para guardar a leitura PINK.
ctime	Armazena o tempo decorrido em microssegundos.
dtime	Tempo de Duty Cycle em nível alto.
edgeTime[8]	Tempo de Duty Cycle em nível baixo.
PCintLast	Variável de memorização do valor PINK para comparação em interrupção futura.

Fonte: Autoria Própria

Seguindo o código da interrupção, logo após a declaração das variáveis, tem-se a variável “pin” fazendo a leitura do nível lógico de cada porta K. A variável “mask” armazena o valor de uma operação XOR entre a variável “pin” e a variável “PCintLast”, que armazenou o valor “pin” de uma interrupção passada, ou seja, contém o nível lógico de cada porta numa iteração passada. A variável “mask” tem, então, a função de identificar futuramente em qual porta houve uma mudança de estado lógico.

Segue-se então para a habilitação das interrupções pela função “sei( )”, podendo haver uma interrupção dentro de outra (aninhamento), e a atualização da variável “PCintLast” com o valor atual de “pin”.

A variável “Ctime” recebe o tempo retornado pela função “micros( )[9]”. A função “micros( )” retorna o tempo de contagem em microssegundos desde que a placa começou a rodar o programa. Esta contagem terá um overflow depois de apenas 70 minutos, ou seja, tempo totalmente seguro para qualquer voo de quadrirrotor.

Tem-se então a chamada do macro do tipo função “RX_PIN_CHECK(pin_pos, rc_value_pos)”.

[9] Arduino, “micros()” [Online]. Disponível: https://www.arduino.cc/reference/en/language/functions/time/micros/ [Acesso em Dezembro 2018].

RX_PIN_CHECK

```
#define RX_PIN_CHECK(pin_pos, rc_value_pos)

  if (mask & PCInt_RX_Pins[pin_pos]) {
    if (!(pin & PCInt_RX_Pins[pin_pos])) {
      dTime = cTime-edgeTime[pin_pos];
      if (900<dTime && dTime<2200) rcValue[rc_value_pos] = dTime;
    } else edgeTime[pin_pos] = cTime;
  }
```

Para a explicação do macro do tipo função, será considerada a condição “pin_pos = 0” e “rc_value_pos = 2”, que é justamente a primeira condição de chamada na ISR.

A variável “pin_pos” faz referência à posição do vetor “PCInt_RX_Pins[8]”, ou seja, se 0, retorna-se o byte presente na posição 0 (00000100), fazendo ligação com o pino PK2.

A variável “rc_value_pos” faz referência à posição do vetor “rcValue”. Quando o vetor “rcValue” faz a atualização de seus valores, que ainda será apresentada, recebe como parâmetro de posição o vetor “rcChannel”, tendo a seguinte estruturação:

$$rcValue[rcChannel[chan]] \quad [4]$$

Sendo “chan” os canais apresentados na Tabela 5.

Portanto, para “rcChannel” retonar um valor 2, este deve fazer referência ao pino PK2. A segunda posição do vetor “rcValue” é o canal 1 do sistema, sendo referente a ação de PITCH.

As equações [5] até [17] descritas abaixo, fazem a demonstração da lógica desenvolvida, caso aconteça uma interrupção pelo pino PK2.

Na condição:

$$if\ (mask\ \&\ PCInt\_RX\_Pins[0]) \quad [5]$$

É verificado se houve mudança de nível lógico no pino PK2 através da variável “mask” e a posição do pino no vetor “PCInt_RX_Pins”, sendo:

$$PCInt\_RX\_Pins = \{00000100,\ 00010000,\ 00100000,\ 01000000,\ 10000000,\ 00000001,\ 00000010,\ 00001000\} \quad [6]$$

$$PCInt\_Rx\_Pins[0] = 00000100 \quad [7]$$

Se houve mudança em PK2:

$$mask = 00000100 \quad [8]$$

$$mask\ \&\ PCInt\_RX\_Pins[0] = 1 \quad [9]$$

Retorno será diferente que 0, entrando dentro da condição, que logo após verifica novamente outra condição:

$$if\ (!\ (pin\ \&\ PCInt\_RX\_Pins[0])) \quad [10]$$

Essa condição é satisfeita apenas na segunda variação de nível lógico no pino PK2, ou seja, quando:

$$pin = 00000000 \quad [11]$$

$$PCInt\_RX\_Pins[0] = 00000100 \quad [12]$$

$$!\ (pin\ \&\ PCInt\_RX\_Pins[0] = 0\ ) = 1 \quad [13]$$

Ao entrar nessa condição, é feio o cálculo de “dtime” através da diferença de tempo decorrido na função “micros( )” e o tempo de “edgeTime” do pino referente.

$$dTime = cTime - edgeTime[0] \quad [14]$$

A variável “dTime” calculada passa por uma condição:

$$if\ (900 < dTime\ \&\&\ dTime < 2200) \quad [15]$$

Se satisfeito o intervalo (900 < “dTime” > 2200), encara-se, por fim, a atualização do valor do canal 1 (“rcValue[2]”) enviada pelo rádio controle. É observável certo intervalo de segurança, desde que a operação do receptor é de [1000µs, 2000µs].

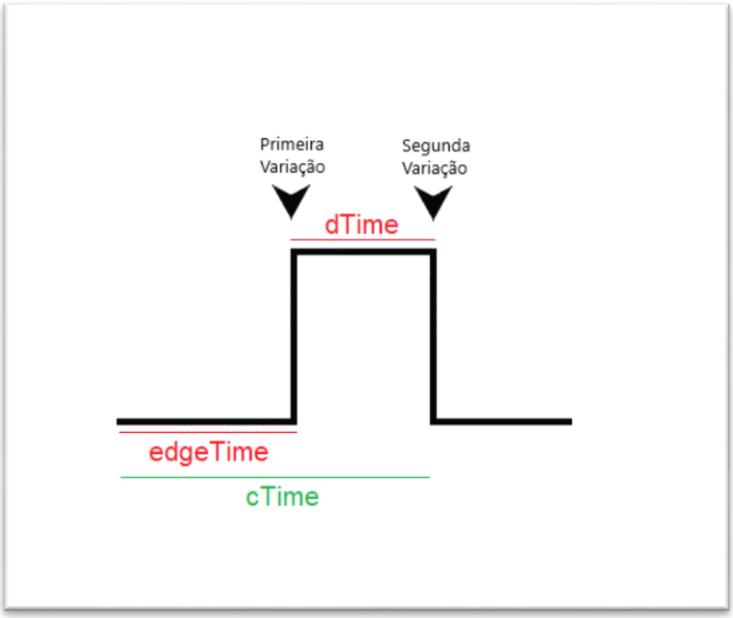
$$rcValue[2] = dTime \quad [16]$$

Porém, ainda é necessário, analisar a condição, caso ainda esteja na primeira variação de nível lógico no pino PK2, atualizando o valor de “edgeTime”.

$$edgeTime[0] = cTime; \quad [17]$$

A Figura 30 demonstra a lógica das variáveis de tempo para a leitura do valor de PWM enviado pelo receptor.

Figura 30 - Captura do sinal do receptor para cálculo de Duty Cycle.



Fonte: Autoria Própria

Fechando-se a abordagem de capturar o sinal PWM enviado do receptor à controladora, introduz-se ao tratamento desses dados pela função “computeRC( )”. Onde os valores armazenados no vetor “rcValue” serão tratados e repassados ao

vetor “rcData”, que também possui uma posição para cada canal, e é declarado e utilizado no bloco de código principal “MultiWii_2_1” na função “annexCode”.

computeRC()

```
void computeRC() {

  static int16_t rcData4Values[8][4], rcDataMean[8];
  static uint8_t rc4ValuesIndex = 0;
  uint8_t chan,a;

  rc4ValuesIndex++;

  for (chan = 0; chan < 8; chan++) {
    rcData4Values[chan][rc4ValuesIndex%4] = readRawRC(chan);

    rcDataMean[chan] = 0;

    for (a=0;a<4;a++) rcDataMean[chan] += rcData4Values[chan][a];

    rcDataMean[chan]= (rcDataMean[chan]+2)/4;

    if ( rcDataMean[chan] < rcData[chan] -3)  rcData[chan] =
rcDataMean[chan]+2;
    if ( rcDataMean[chan] > rcData[chan] +3)  rcData[chan] =
rcDataMean[chan]-2;      }

}
```

De costume, tem-se inicialmente, a declaração das variáveis utilizadas na função. Estas se encontram comentadas na Tabela 7.

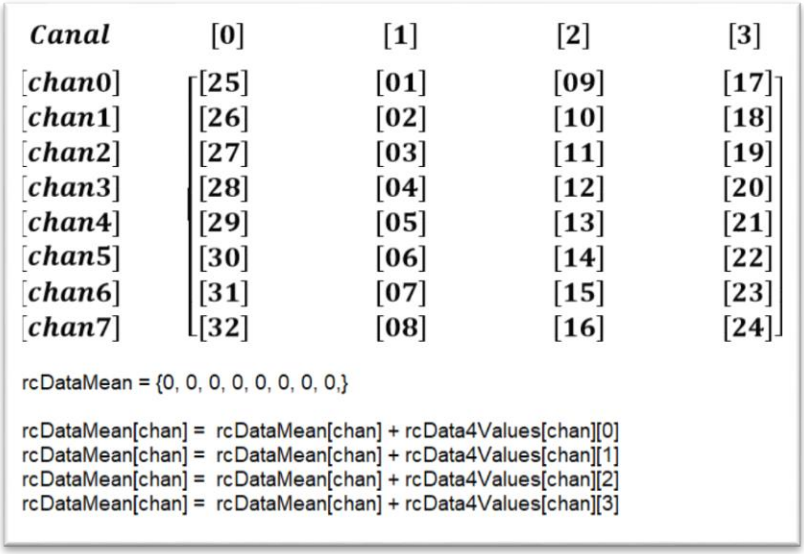
Tabela 7 – Variáveis para a função computeRC.

Variável	Descrição
rcData4Values[8][4]	Matriz para armazenar 4 valores de cada um dos 8 canais.
rcDataMean[8]	Vetor para armazenar a média de cada canal.
rc4ValuesIndex	Indexador para coluna da matriz rcData4Values[8][4].
chan	Variável de referência aos canais.
a	Indexador para cálculo da média.

Fonte: Autoria Própria

A função “readRawRC(chan)”, que será explicada adiante, faz o retorno de uma posição do vetor “rcValue” para o preenchimento da matriz “rcData4Values”. Quatro valores para cada canal são armazenados na matriz e utilizados para o cálculo de uma média. Essa média é armazenada no vetor “rcDataMean”, que possui uma posição para cada canal. O preenchimento da matriz “rcData4Values” e do vetor “rcDataMean” é explicado na numeração presente na Figura 31.

Figura 31 - Ordem de preenchimento da matriz e do vetor.



Fonte: Autoria Própria.

A média é calculada conforma equação abaixo. Soma-se os 4 valores acumulados com 2 unidades, e divide-se por 4:

$$rcDataMean[chan] = (rcDataMean[chan] + 2)/4 \quad [18]$$

Se faz a seguinte lógica para a atualização do vetor “rcData”:

$$\boldsymbol{if}\ (\boldsymbol{rcDataMean}[\boldsymbol{chan}] < \boldsymbol{rcData}[\boldsymbol{chan}] - \mathbf{3})$$
$$rcData[chan] = rcDataMean[chan] + 2 \quad [19]$$

$$\boldsymbol{if}\ (\boldsymbol{rcDataMean}[\boldsymbol{chan}] > \boldsymbol{rcData}[\boldsymbol{chan}] + \mathbf{3})$$
$$rcData[chan] = rcDataMean[chan] - 2 \quad [20]$$

Basta agora entender a lógica da função “readRawRC(chan)”, apresentada no bloco de código abaixo.

readRawRC

```
uint16_t readRawRC(uint8_t chan) {

  uint16_t data;
  uint8_t oldSREG;

  oldSREG = SREG; cli();

  data = rcValue[rcChannel[chan]];
  SREG = oldSREG; sei();

  return data;
}
```

Esta função recebe o canal, referente a linha da matriz “rcData4Values”, retornando o valor armazenado no vetor “rcValue[8]” para o canal correspondente.

Duas variáveis são criadas, “data”, para armazenar o valor de retorno, e “oldSREG”, para guardar o valor do registrador geral ao desabilitar as interrupções.

De forma sucinta, ao entrar na função, as interrupções são desabilitadas, guarda-se o valor armazenado em “rcValue” na variável “data”, habilita-se novamente as interrupções e retorna-se “data”.

Lembrando que:

$$rcChannel[8] = \begin{Bmatrix} ROLLPIN, PITCHPIN, \\ YAWPIN, THROTTLEPIN, \\ AUX1PIN, AUX2PIN, \\ AUX3PIN, AUX4PIN \end{Bmatrix} \quad [21]$$

Onde devido ao macro:

$$rcChannel[8] = \{1, 2, 3, 0, 4, 5, 6, 7\} \quad [22]$$

Portando, se for recebido o canal 3 como parâmetro, será retornado a posição 0 do vetor “rcValue”, sendo referente ao THROTTLE, conforme pode ser analisado na Tabela 5.

Ainda pode ser observado que as interrupções são desabilitadas na atualização da variável “data” e habilitadas em sequência. Tanto interrupções

externas, quanto interrupções referentes à comunicação serial são habilitadas neste trecho. A interrupção externa “PCINT2_vect” permite o aninhamento de outras interrupções.

13 OUTPUT.INO

13.1 Temporizadores/Contadores

Contadores e Temporizadores (TCs) são periféricos dos microcontroladores que, principalmente, contam pulsos de clock, os quais podem ser externos ou internos, sendo assim utilizados para gerar sinais periódicos e eventos, algo essencial em muitos projetos. (LIMA e VILLAÇA, 2012).

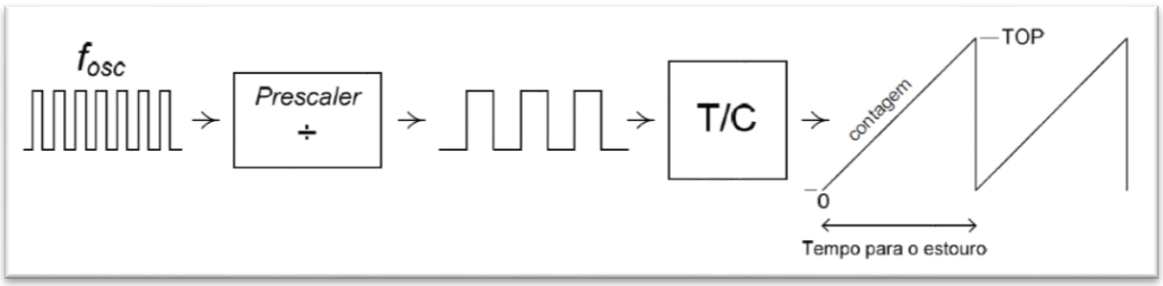
O ATmega2560 possui dois temporizadores/contadores de 8 bits (TC0 e TC2), contando de 0 até 255, e quatro de 16 bits (TC1, TC3, TC4 e TC5), contando de 0 até 65535. O tempo que um TC leva para estourar é dado pela equação [23].

$$t_{estouro} = \frac{(TOP + 1) \times prescaler}{f_{osc}} \quad [23]$$

Sendo TOP o valor máximo de contagem, f_{osc} a frequência de clock e $prescaler$ o divisor de frequência configurável pelos registradores.

A Figura 32 ilustra o funcionamento de um TC no ATmega2560.

Figura 32 - Funcionamento de um temporizador/contador no ATmega2560.



Fonte: LIMA e VILLAÇA (2012).

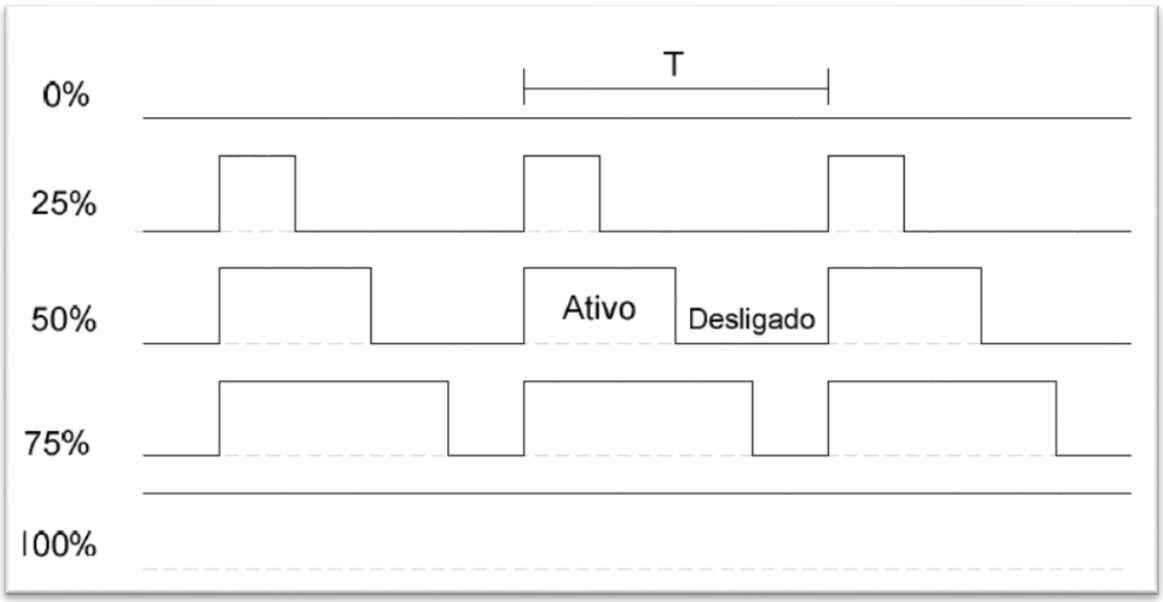
13.1.1 Modulação por largura de pulso

Outra função dos TCs é a geração de sinais PWM. Um sinal PWM é utilizado baseando-se no conceito de valor médio de uma forma de onda periódica. (LIMA e VILLAÇA, 2012).

A partir deste valor médio, muitos componentes podem ser controlados, no caso de um quadrirrotor, este valor é passado para os ESCs para o controle de velocidade dos motores.

Em um sinal digital, o valor médio de uma forma de onda é dado pelo tempo em que o sinal fica em nível lógico alto dentro de um período de oscilação (T). Este período em nível alto é chamado de ciclo ativo (Duty Cycle). (LIMA e VILLAÇA, 2012). A Figura 33 ilustra incrementos de Duty Cycle, de 0% até 100%.

Figura 33 - Incrementos de Duty Cycle.



Fonte: LIMA e VILLAÇA (2012).

Portanto, o valor médio da tensão de saída de um sinal digital ($V_{médio}$) é dado pela equação [24].

$$V_{médio} = V_{max} \times DC \quad [24]$$

Onde V_{max} é a tensão do nível lógico alto e DC o Duty Cycle, que é dado por:

$$DC = \frac{T_a}{T} \quad [25]$$

Onde T_a é o período em nível lógico alto e T o período de oscilação do sinal PWM.

Conclui-se que ao alterar o Duty Cycle do sinal PWM, altera-se o valor médio de saída. Porém, nesta variação o período do sinal PWM deve se manter o mesmo. (LIMA e VILLAÇA, 2012).

13.1.1.1 Modo com fase corrigida

Os contadores/temporizadores utilizados no firmware do *MultiWii Pro* são o TC3 e TC4, cuja as características são idênticas.

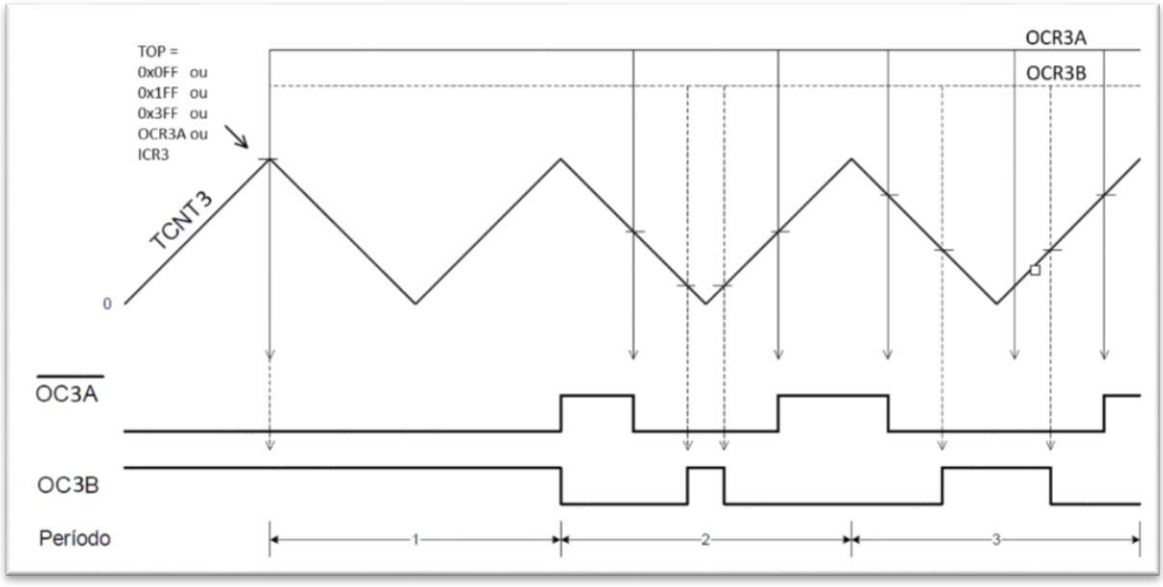
Para estes contadores/temporizadores a contagem é feita no par de registradores TCNT3H e TCNT3L para o TC3 e no par TCNT4H e TCNT4L para o TC4. Ambos possuem três registradores de controle cada, sendo TCCR3A, TCCR3B e TCCR3C para TC3 e TCCR4A, TCCR4B e TCCR4C para TC4. Ambos também possuem três registradores de comparação de saída, OCR3A, OCR3B e OCR3C para TC3 e OCR4A, OCR4B e OCR4C para TC4, sendo estes constantemente comparados com o valor de contagem (TCNT3/TCNT4). O resultado de comparação pode ser usado para gerar sinais PWMs ou com frequência variável nos pinos de saída de comparação (OC3A, OC3B e OC3C / OC4A, OC4B e OC4C). O valor máximo de contagem (TOP) para TC3 e TC4 pode ser definido por OCR3A/OCR4A, ICR3/ICR4 ou por um conjunto fixo de valores.

No firmware do *MultiWii Pro* é utilizada a geração de pulso PWM no modo de fase corrigida para envio de sinais aos ESCs dos motores. Este modo permite ajustar a fase do PWM, ou seja, o início e o fim do Duty Cycle, baseando-se na contagem crescente e decrescente de TCNT3/TCNT4, tornando a saída PWM simétrica dentro de um período do PWM. O início e fim de um ciclo PWM é dado quando o contador atinge o valor TOP, podendo atualizar neste instante o valor de comparação que determina a razão cíclica. Assim, se não houver alteração no valor TOP, o pulso gerado será sempre simétrico em relação ao ponto médio do período, mesmo após a atualização da razão cíclica. (LIMA e VILLAÇA, 2012).

A Figura 34 ilustra um diagrama de tempo para o PWM com fase corrigida, onde a saída OC3A é invertida e OC3B é não invertida. É possível notar que o valor máximo de contagem (TOP) pode ser definido como 0x0FF, 0x1FF, 0x3FF, OCR3A

ou ICR3. Atentando que, se OCR3A for utilizado para determinar o valor TOP, a saída gerará apenas uma onda quadrada.

Figura 34 - Diagrama de tempo para o PWM com fase corrigida



Fonte: Adaptado de LIMA e VILLAÇA (2012).

A resolução do PWM é dada por:

$$R = \frac{\log(TOP + 1)}{\log(2)} \quad [26]$$

A frequência de saída do PWM com fase corrigida é dada por:

$$f_{OCnx\_PWM} = \frac{f_{osc}}{2\mathrm{N}.\mathrm{TOP}} \quad [27]$$

Onde N é o fator prescaler (1, 8, 64, 256 ou 1024).

13.2 Estudo do Firmware (output.ino)

O bloco de “output” contém os algoritmos responsáveis pela inicialização e configuração do PWM, calibração dos ESCS e a atualização dos valores de PWM de acordo com a ação de controle necessária.

Os contadores/temporizadores utilizados no firmware do *MultiWii Pro* para a geração de PWM no modo com fase corrigida são o TC3 e TC4. A inicialização e configuração de ambos se encontram na função “initOutput”.

initOutput

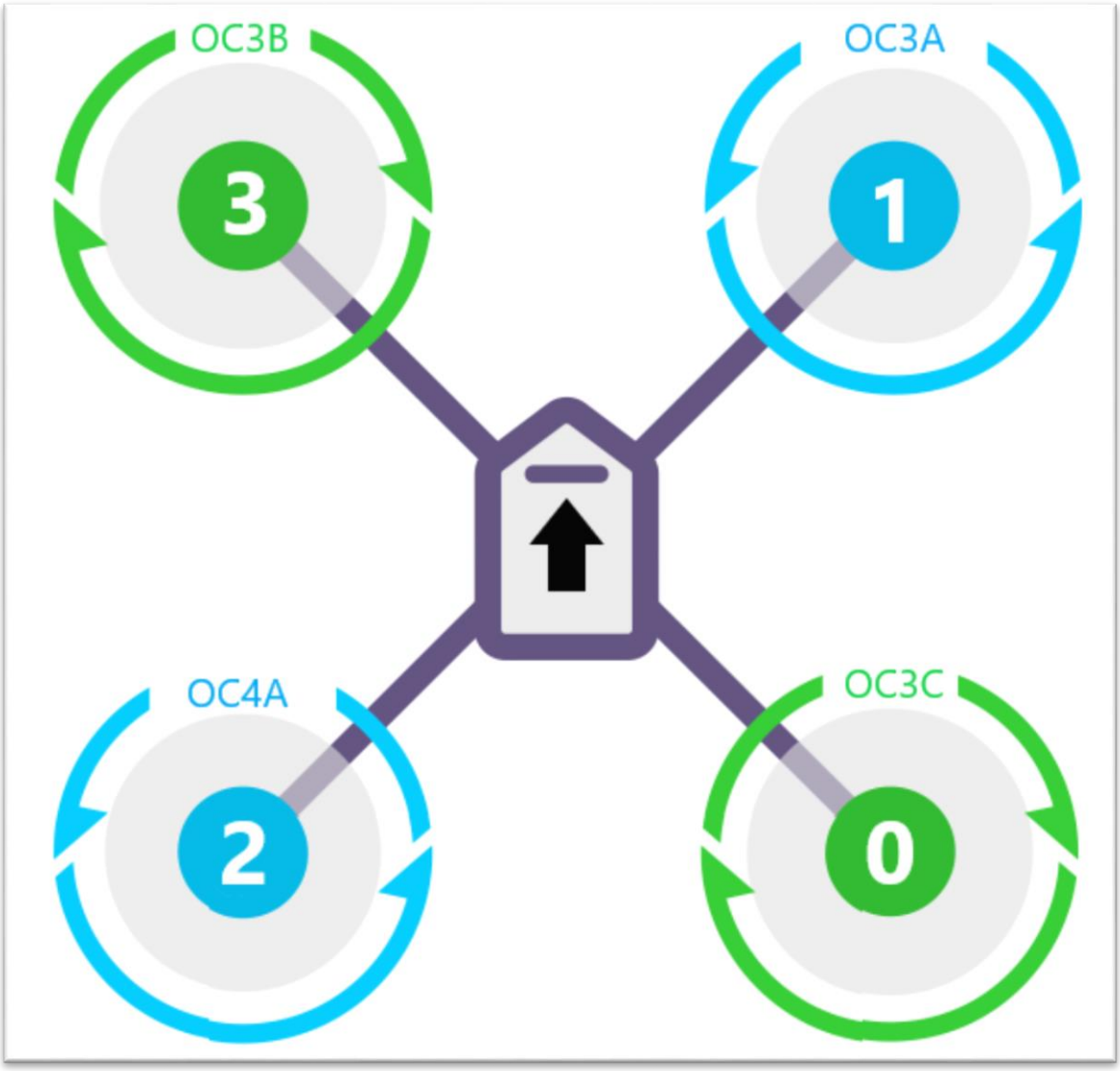
```
//PWM_PIN[8] = {3,5,6,2,7,8,9,10};

Void initOutput() {
  for(uint8_t i=0;i<NUMBER_MOTOR;i++) {
    pinMode(PWM_PIN[i],OUTPUT);
  }
  //  Inicialização do TIMER 3
  TCCR3A |= (1<<WGM31);
  TCCR3A &= ~(1<<WGM30);
  TCCR3B |= (1<<WGM33);
  TCCR3B &= ~(1<<CS31);
  ICR3   |= 0x3FFF; // TOP to 16383;
  TCCR3A |= _BV(COM3C1);
  TCCR3A |= _BV(COM3A1);
  //  Inicialização do TIMER 4
  TCCR4A |= (1<<WGM41);
  TCCR4A &= ~(1<<WGM40);
  TCCR4B |= (1<<WGM43);
  TCCR4B &= ~(1<<CS41);
  ICR4   |= 0x3FFF; // TOP to 16383;
  TCCR4A |= _BV(COM4A1);
  TCCR3A |= _BV(COM3B1);

  writeAllMotors(MINCOMMAND);
  delay(300);
}
```

A função “initOutput” inicia declarando quais são os pinos de saída para a geração do sinal PWM. Sendo o número de motores limitado em 4 (quadrirrotor), tem-se a seguinte declaração dos pinos digitais PWM:

- **motor[0]** = PWM 3 = PE5 (0C3C)
- **motor[1]** = PWM 5 = PE3 (0C3A)
- **motor[2]** = PWM 6 = PH3 (0C4A)
- **motor[3]** = PWM 2 = PE4 (0C3B)

A Figura 35 demonstra a posição de cada um dos pinos PWM em um quadrirrotor em orientação X.

Figura 35 - Orientação das saídas de comparação para atuador.



Fonte: Autoria Própria.

Logo após a declaração das portas de saída para sinal PWM, há a configuração dos registradores do TC3 e TC4. A Tabela 8 demonstra quais foram os registradores alterados e quais os objetivos de cada alteração na inicialização e configuração do sinal PWM.

Tabela 8 – Registradores para configuração de PWM.

TC3		
TCCR3A	WGM30 = 0 WGM31 = 1	Modo PWM com fase corrigida; TOP = ICR3; Atualiza OCR3A/OCR3B/OCR3C no TOP.
TCCR3B	WGM32 = 0 WGM33 = 1	
TCCR3A	COM3C1 = 1 COM3A1 = 1	OC3A/OC3B/OC3C = 0 (quando comparado na contagem crescente); OC3A/OC3B/OC3C = 1 (quando comparado na contagem decrescente).
TCCR3B	CS30 = 1 CS31 = 0 CS32 = 0	Prescaler = 1
ICR3	0x3FFF	TOP = 16383
TC4		
TCCR4A	WGM40 = 0 WGM41 = 1	Modo PWM com fase corrigida; TOP = ICR4; Atualiza OCR4A/OCR4B/OCR4C no TOP.
TCCR4B	WGM43 = 1 WGM42 = 0	
TCCR4A	COM4C1 = 1 COM4A1 = 1	OC4A/OC4B/OC4C = 0 (quando comparado na contagem crescente); OC4A/OC4B/OC4C = 1 (quando comparado na contagem decrescente).
TCCR4B	CS40 = 1 CS41 = 0 CS42 = 0	Prescaler = 1
ICR4	0x3FFF	TOP = 16383

Fonte: Autoria Própria.

No final da função há a escrita do comando mínimo em cada uma das portas PWM através da função “writeAllMotors”, a qual será apresentada em breve.

Sabe-se que os ESCs trabalham com um pulso entre [1000µs, 2000µs], sendo 1000µs para rotação mínima dos motores e 2000µs para rotação máxima.

Sabe-se que a partir da equação [28] se tem:

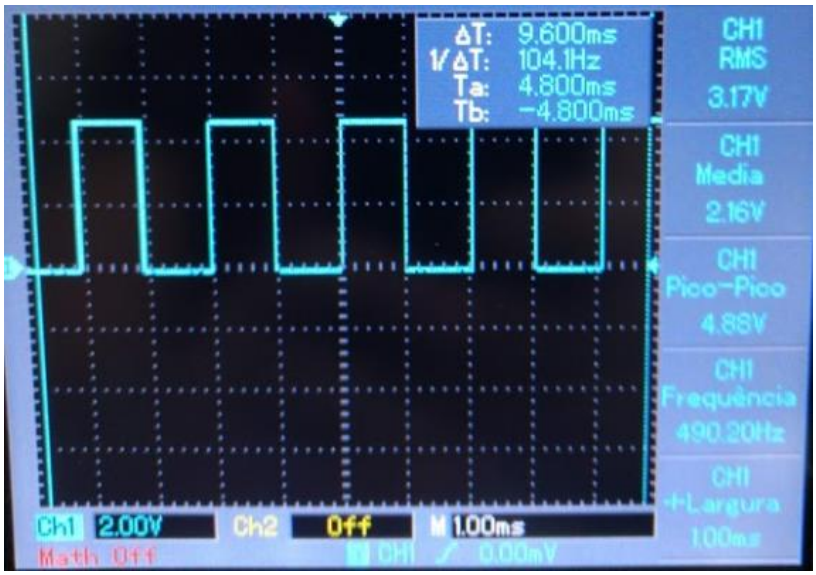
$$f_{OCnx\_PWM} = \frac{f_{osc}}{2\mathrm{N}.\,\mathrm{TOP}} = \frac{16M}{2 \times 16383} \approx 488.31Hz \qquad [28]$$

Gerando um período máximo do sinal PWM de:

$$T_{OCnx\_PWM} = \frac{1}{488.31Hz} \approx 2047\mu s \qquad [29]$$

Para gerar um pulso de nível alto de aproximadamente 1000µs de duração (valor mínimo de trabalho dos ESCs), é necessário, portanto, um ciclo ativo de 50% do sinal de PWM. A Figura 36 ilustra uma medida real do sinal PWM de uma das saídas do *MultiWii Pro* para um pulso de aproximadamente 1000µs.

Figura 36 – Saída PWM real do MultiWii Pro (1000µs).



Fonte: Autoria Própria.

Sendo assim, aproximadamente, 1000µs deve equivaler a 50% de DC e 2000µs a 100% de DC. Os registradores de comparação de saída, devem então,

atualizar os seus valores de acordo com essa regra. A função “writeMotors” demonstra como isso é feito.

writeMotors

```
void writeMotors() {
 OCR3C = motor[0]<<3; //  pin 3
 OCR3A = motor[1]<<3; //  pin 5
 OCR4A = motor[2]<<3; //  pin 6
 OCR3B = motor[3]<<3; //  pin 2
}
```

Os valores de “motor[i]” são atualizados logo após a aplicação do sinal de controle, lógica a qual será explicada no capítulo de identificação dos controles. Sabe-se que estes valores variam entre [1000, 2000]. Sendo o valor TOP dos contadores igual a 16383, os valores de “motor[i]” passam pela seguinte modificação para a atualização dos registradores de comparação de saída:

$$motor[i] = [1000, 2000] \rightarrow [8000, 16000] \quad [30]$$

Esta modificação linear, multiplicando-se os valores em 8x, gera-se a seguinte relação entre o valor dos registradores de comparação e o ciclo ativo gerado:

$$OCRnx = 8000 \rightarrow 50\%\text{DC} \rightarrow 1000\mu s \quad [31]$$

$$OCRnx = 16000 \rightarrow 100\%\text{DC} \rightarrow 2000\mu s \quad [32]$$

Qualquer valor entre os limites de operação do ESC é incrementado de maneira linear.

writeAllMotors

```
void writeAllMotors(int16_t mc) {
  for (uint8_t i =0;i<NUMBER_MOTOR;i++) {
    motor[i]=mc;
  }
  writeMotors();
}
```

Quando se deseja que todos os 4 motores fiquem na mesma rotação, utiliza-se a função “writeAllMotors”. Esta função, basicamente atualiza os 4 valores do

vetor “motor[i]” de acordo com o parâmetro repassado, podendo este variar entre [1000, 2000].

ESC_CALIB

```
#if defined(ESC_CALIB_CANNOT_FLY)
  writeAllMotors(ESC_CALIB_HIGH);
  delay(3000);

  writeAllMotors(ESC_CALIB_LOW);
  delay(500);

  while (1) {
    delay(5000);
    blinkLED(2,20, 2);
  }
  exit;
#endif
```

Adentrando a função de inicialização e configuração, “initOutput”, encontra-se a opção de calibrar os ESCs. Isto é feito ativando o macro “ESC_CALIB_CANNOT_FLY” na biblioteca de configuração (“config.h”).

O modo de calibração vai levar aproximadamente 9 segundos para ser concluído, devendo, após isso, atualizar o firmware com o macro desativado para acessar o protocolo normal de voo.

A calibração ocorre simultaneamente nos 4 ESCs logo após a energização do sistema. Os passos para calibração são padronizados entre a maioria dos ESCs no mercado, e seguem abaixo:

- 3s com 100%DC (2000µs);
- 0.5s com 50%DC (1000µs).

Depois de calibrar, um LED fica acendendo e apagando a cada 5 segundos, indicando que o firmware está preso em um loop. Deve-se, então, atualizar o firmware, conforme foi dito.

14 SERIAL.INO

O bloco de código “serial.ino” é responsável pela comunicação USART (Universal Synchronous and Asynchronous serial Receiver and Transmitter) do *ATmega2560*, contida na placa de controle *MultiWiii Pro*. Aqui são inicializados os registradores de configuração da comunicação e também realizada toda a dinâmica de transmissão e recepção com buffers.

Para melhor compreensão das técnicas utilizadas nesse bloco de código foram realizadas mudanças no código original, deixando-o mais enxuto a partir de simplificações e com a retirada da comunicação serial com o módulo GPS. Devido tais modificações, este bloco tratará principalmente de toda comunicação entre o software GUI e a placa de controle.

Visando uma expressão didática, a ordem de apresentação do código feita aqui neste trabalho é diferente do código fonte final. Aqui a pauta se inicia com os registradores de configuração, passando para as técnicas de transmissão de dados, técnicas de recepção de dados e finalizando com o protocolo de comunicação entre o software GUI e a placa.

14.1 Configuração da Comunicação

Antes de analisar as linhas de códigos, é válido apresentar as características do módulo de comunicação USART do *ATmega2560*, segundo o manual do fabricante.

- Operação Full Duplex (registradores independentes de recepção e transmissão);
- Operação síncrona ou assíncrona;
- Operação síncrona com clock mestre ou escravo;
- Gerador de taxa de comunicação de alta resolução (Baud Rate Generator);
- Suporta frames seriais com 5, 6, 7, 8 ou 9 bits de dados e 1 ou 2 bits de parada;

- Gerador de paridade par ou ímpar e conferência de paridade por hardware;
- Detecção de colisão de dados e erros de frames;
- Filtro para ruído, incluindo bit de início falso e digital passa-baixa;
- Três fontes separadas de interrupção (transmissão completa, recepção completa e esvaziamento do registrador de dados);
- Modo de comunicação assíncrono com velocidade duplicável;
- Pode ser utilizada como interface SPI mestre.

Tendo conhecimento da amplitude e das opções desse microcontrolador, fica mais compreensível as decisões e técnicas que serão apresentadas pelas funções a seguir.

SerialOpen()

```
static void inline SerialOpen(uint8_t port, uint32_t baud) {

  uint8_t h = ((F_CPU  / 4 / baud -1) / 2) >> 8;

  uint8_t l = ((F_CPU  / 4 / baud -1) / 2);

  switch (port) {
    case 0: UCSR0A  = (1<<U2X0); UBRR0H = h; UBRR0L = l; UCSR0B |=
(1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0); break;
    case 1: UCSR1A  = (1<<U2X1); UBRR1H = h; UBRR1L = l; UCSR1B |=
(1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1); break;
    case 2: UCSR2A  = (1<<U2X2); UBRR2H = h; UBRR2L = l; UCSR2B |=
(1<<RXEN2)|(1<<TXEN2)|(1<<RXCIE2); break;
    case 3: UCSR3A  = (1<<U2X3); UBRR3H = h; UBRR3L = l; UCSR3B |=
(1<<RXEN3)|(1<<TXEN3)|(1<<RXCIE3); break;
  }
}
```

A função “SerialOpen” é responsável por inicializar e configurar a comunicação USART.

O parâmetro “baud”, um dos dois parâmetros que a função recebe, é a taxa de comunicação (baud rate), no caso 115200, e com esta pode-se definir um valor no UBRR0 (USART baud Rate Registrer) de acordo com o modo de operação desejado. Os modos de operação possíveis no ATmega2560 estão apresentados na Tabela 9.

Um contador recebe o valor definido em UBBR0 e faz uma contagem decrescente na velocidade de clock da CPU, toda vez que o contador é zerado, um pulso de clock é gerado, resultando na taxa de comunicação desejada. O contador

também, quando zera, recebe novamente o valor de UBBR0 para manter a geração periódica da taxa de comunicação. (LIMA e VILLAÇA, 2012).

Tabela 9 - Modos de Operação da USART.

Modo de Operação	Taxa de transmissão	Valor de UBRR0
Modo Normal Assíncrono (U2X0 = 0)	$TAXA = \frac{fosc}{16(UBRR0 + 1)}$	$UBBR0 = \frac{fosc}{16TAXA} - 1$
Modo de Velocidade Dupla Assíncrono (U2X0 = 1)	$TAXA = \frac{fosc}{8(UBRR0 + 1)}$	$UBBR0 = \frac{fosc}{8TAXA} - 1$
Modo Mestre Síncrono	$TAXA = \frac{fosc}{2(UBRR0 + 1)}$	$UBBR0 = \frac{fosc}{2TAXA} - 1$

Fonte: Autoria Própria.

Analisando as primeiras linhas de código da função “SerialOpen”, mais necessariamente as variáveis “h” e “l”, é evidente a escolha do modo de velocidade dupla assíncrono, justamente pela opção da equação.

Como UBRR0 tem em seu valor máximo 12 bits, este é composto por dois registradores de 8 bits, UBRR0H e UBRR0L, a variável de 8 bits “h” faz o deslocamento de 8 bits à direita, justamente para armazenar os 4 bits mais significativos de UBRR0.

O parâmetro “port” apenas seleciona quais portas de comunicação serão ativadas e configuradas, podendo ser selecionados os pares RX0 e TX0 (pinos digitais 0 e 1), RX1 e TX1 (pinos digitais 19 e 18), RX2 e TX2 (pinos digitais 17 e 16) e RX3 E TX3 (pinos digitais 15 e 14), todos com a mesma configuração.

Os pinos RX0 e TX0 são os responsáveis pela a comunicação entre software GUI e placa controladora, já os outros pinos estão pré-configurados aqui para futuras aplicações de telemetria por bluetooth, análise de altura por sensor sonar e navegação por módulo GPS.

A configuração da comunicação é constituída pela ativação do bit U2Xn no registrador UCSRnA, confirmando o modo de velocidade dupla assíncrono; pela atribuição dos registradores UBRRnH e UBRRnL; pela ativação dos bits RXENn e

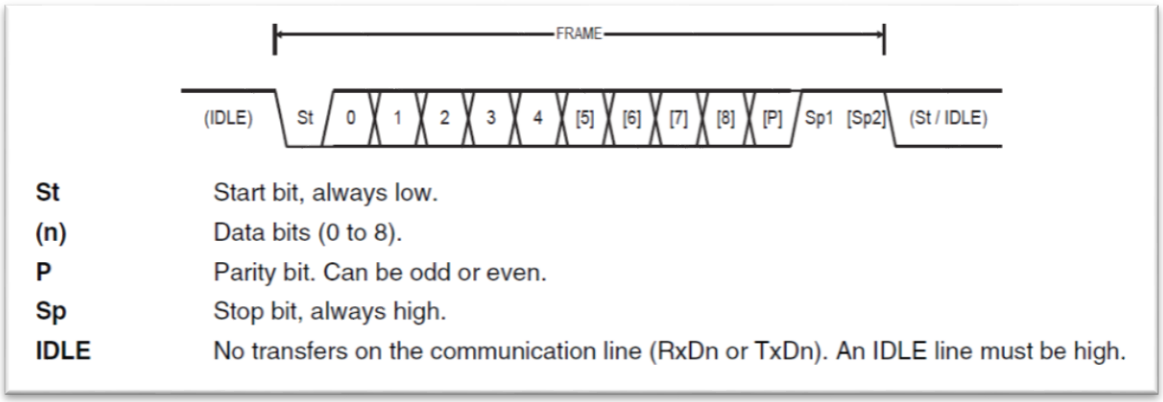
TXEN0n no registrador UCSRnB, habilitando a recepção e transmissão de dados seriais; pela ativação do bit RXCIEn no registrador UCSRnB, que habilita a interrupção de recepção completa.

O registrador UCSRnC é mantido em seu valor inicial, resultando nas seguintes configurações:

- USART assíncrono;
- Modo de paridade desativado;
- 1-bit de stop bit;
- 8-bit de tamanho de caractere.

Para entender melhor essas configurações, A Figura 37 ilustra o formato do frame na comunicação USART.

Figura 37 – Formato do frame na USART.



Fonte: ATMEGA2560: Manual do fabricante.

Uma função para encerrar a comunicação USART também é disponibilizada, “SerialEnd”. Esta apenas coloca em nível lógico baixo nos bits RXENn, TXENn, RXCIEn e UDRIE0 do registrador UCSRnB. Ou seja, desabilita a recepção, transmissão e suas interrupções.

14.2 Técnica de Transmissão

A transmissão de dados na porta TX0, ou seja, dados de origem na placa controladora com destino a um hospedeiro rodando o software GUI, é feita utilizando um buffer, em termos de precisão na linguagem C, um vetor.

Este vetor, denominado “bufTX”, é declarado com 128 posições, sendo cada uma ocupada por uma variável inteira sem sinal de 8 bits, podendo dizer então, que este vetor tem poder de armazenagem de 128 bytes.

Duas variáveis para indicação de posição do vetor “bufTX” foram criadas, “headTX” e “tailTX”. Estas variáveis são essenciais na técnica de transmissão utilizada aqui, onde uma variável, “headTX”, é responsável por indicar a posição do vetor em que o dado será primeiramente escrito, e a outra variável, “tailTX”, indica a posição do dado, já escrito anteriormente, que será transmitido. Isto permite dinâmica, segurança e confiabilidade no envio de dados.

As funções que estão demonstradas abaixo são responsáveis pela escrita dos dados no vetor “bufTX”. É intuitivo analisar o código na ordem “serialize8”, “serialize16” e “serialize32”.

Serialize()

```
void serialize32(uint32_t a) {
  serialize8((a    ) & 0xFF);
  serialize8((a>> 8) & 0xFF);
  serialize8((a>>16) & 0xFF);
  serialize8((a>>24) & 0xFF);
 }

void serialize16(int16_t a) {
  serialize8((a   ) & 0xFF);
  serialize8((a>>8) & 0xFF);
 }

void serialize8(uint8_t a) {
  uint8_t t = headTX;
  if (++t >= TX_BUFFER_SIZE) t = 0;
  bufTX[t] = a;
  checksum ^= a;
  headTX = t;
                          }
```

A função “serialize8” é responsável por escrever um dado de 8 bits, seu parâmetro, em uma posição no vetor “bufTX”. Como é o processo de escrita, a variável indicativa de posição do vetor “bufTX”, “headTX”, logo aparece sendo armazenada em uma variável “t” criada dentro da função. Esta variável é incrementada e verificado se

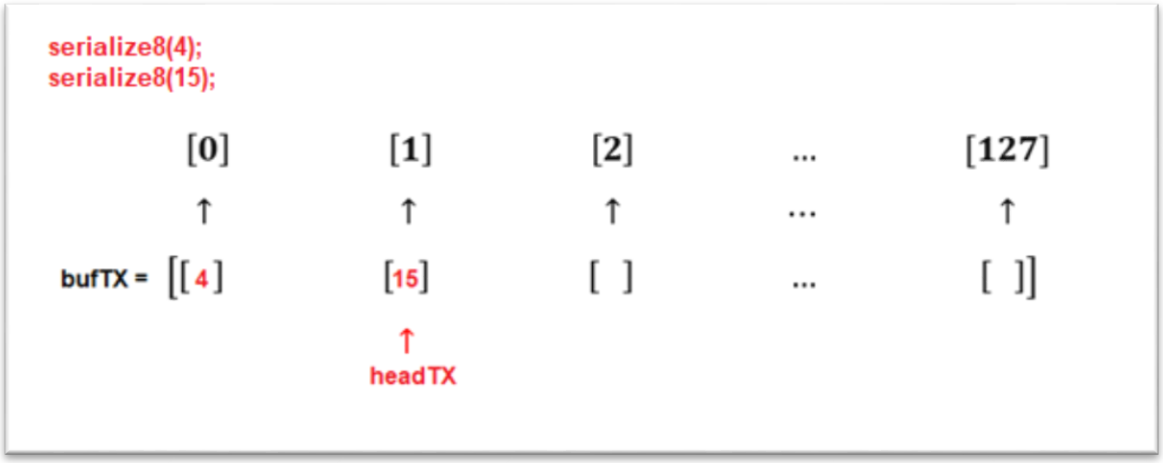
este incremento ultrapassa o tamanho limite do buffer, caso ultrapasse a variável recebe valor 0, indicando voltar para a primeira posição do vetor. Após a verificação é feita a aquisição do dado na posição “t” incrementada do vetor.

É realizada uma operação OU EXCLUSIVO vista na variável “checksum”. Este valor é utilizado pelo software no hospedeiro destinatário para verificar erros na comunicação.

O último feito da função “serialize8” é atualizar o valor de “headTX” com a última posição escrita do vetor “bufTX”. Isso é necessário para uma próxima chamada dessa função.

A Figura 38 ilustra duas iterações da função “serialize8”.

Figura 38 - Posição de "headTX" pós duas iterações da função "serialize8"



Fonte: Autoria própria.

As funções “serialize16” e “serialize32” são responsáveis por escrever no vetor dados de 16 bits e 32 bits respectivamente. Para tal feito, é necessário utilizar o deslocamento dos bits à direita e a função “serialize8” citada anteriormente. Por exemplo, um dado com 32 bits, sendo aleatoriamente (1000 0111 0110 0101 0100 0011 0010 0001), de acordo com a função “serialize32”, descrita no bloco de códigos acima, serão feitas as chamadas de funções na ordem “serialize8(0010 0001) ”, “serialize8(0100 0011) ”, “serialize8(0110 0101) ” e “serialize8(1000 0111) ”.

Tendo os valores para transmissão já escritos no buffer, agora é hora de transmiti-los de fato. Para isso será usada uma interrupção, conforme em exposição no código a seguir.

ISR_UART

```
ISR_UART {

  uint8_t t = tailTX;
  if (headTX != t) {
    if (++t >= TX_BUFFER_SIZE) t = 0;
    UDR0 = bufTX[t];
    tailTX = t;
  }
  if (t == headTX) UCSR0B &= ~(1<<UDRIE0);

}
```

A interrupção aqui mencionada é referente à biblioteca “<avr/interrupt.h>”, onde encontra-se o vetor de interrupção “USART0_UDRE_vect” referente à flag “Data Register Empty 0”.

Uma transmissão tem início quando um dado a ser transmitido é escrito em UDR0, denominado buffer de transmissão. Quando algo é escrito no buffer de transmissão e o transmissor está habilitado, os dados serão carregados parar o registrador de deslocamento de transmissão quando este estiver vazio, ou seja, quando ele estiver ocioso ou logo após o último stop-bit do frame anterior ter sido deslocado para saída. (LIMA e VILLAÇA, 2012, p. 351).

Portanto, se o registrador de deslocamento de transmissão está vazio, os dados podem ser passados do registrador UDR0 para ele, e neste momento a flag UDRE0, disponível no registrador UCSR0A, é posta em nível lógico alto, indicando que o registrador UDR0 pode receber o próximo frame. (LIMA e VILLAÇA, 2012, p. 351).

A flag UDRE0 pode gerar uma interrupção de esvaziamento de registrador de dados (Data Register Empty), interrupção usada no caso em discussão. Para essa interrupção ser ativada o bit UDRIE0 do registrador UCSR0B deve estar em nível lógico alto, algo que acontece na função “UartSendData”. Toda vez que necessitar realizar uma transmissão, essa função deve ser chamada.

UartSendData

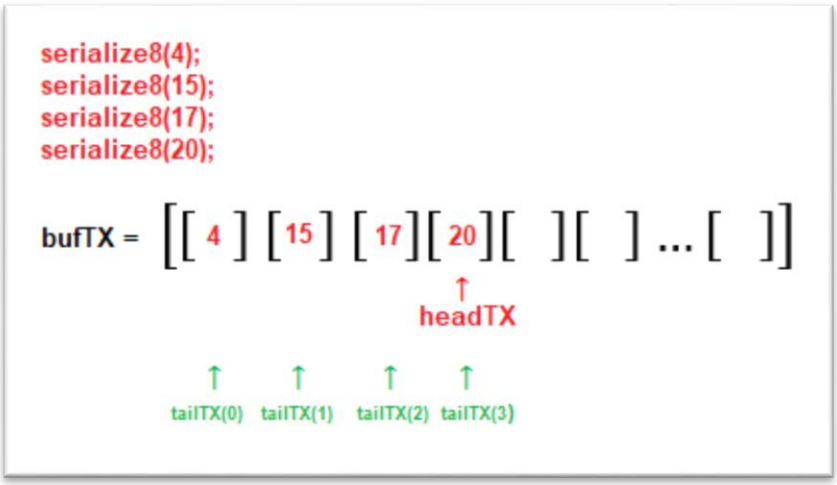
```
void UartSendData() {
    UCSR0B |= (1<<UDRIE0);
```

Voltando ao bloco de interrupção, pode-se dizer que apenas entrará neste quando a flag UDRE0 estiver em nível lógico alto, ou seja, quando o registrador UDRE0 estiver vazio e pronto para receber um novo caractere.

Logo após a interrupção iniciar, o valor da variável indicativa de posição do vetor “bufTX”, “tailTX”, é armazenada numa variável criada na interrupção, “t”. Esta variável passa por uma verificação, onde a posição de “tailTX” deve ser diferente da posição de “headTX” para ocorrer a transmissão, se esta condição não se cumprir a interrupção de transmissão é desabilitada, pois todos os dados escritos no vetor “bufTX” já teriam sido transmitidos – caso necessite realizar uma nova transmissão, a interrupção deve ser habilitada novamente por “UartSendData”-. Na condição em que se continua a transmissão, a variável “t” é incrementada e verificado se este incremento ultrapassa o tamanho limite do buffer. Caso ultrapasse, a variável recebe valor 0, indicando voltar para a primeira posição do vetor. Após esta segunda verificação, o registrador UDR0 recebe o dado armazenada na posição “t” do vetor “bufTX”, seguindo este dado posteriormente para o registrador de deslocamento de transmissão. Por fim, a interrupção atualiza o valor de “tailTX” para iterações futuras.

A Figura 39 exemplifica iterações tanto de escrita, quanto de envio no vetor “bufTX”. Nela pode-se observar a escrita no vetor sendo administrada pela variável “headTX”, a transmissão dos dados sendo administrada pela variável “tailTX” e o caso em que “tailTX” é igual “headTX”, indicando que todos os dados já foram transmitidos.

Figura 39 - Iteração de escrita e envio no buffer "bufTX".



Fonte: Autoria própria.

As portas de transmissão são selecionadas e os dados a ser transmitidos são repassados com a função “SerialWrite”. Essa função deixa evidente a diferença de método entre a porta TX0 e as portas TX1, TX2 e TX3. Onde TX0 exerce a técnica de buffer com uma interrupção de fundo, explicado até agora, já as outras portas apenas enviam o dado recebido diretamente ao buffer de transmissão correspondente, se e somente, este estiver vazio e pronto para receber um novo caractere, podendo gerar um grande atraso.

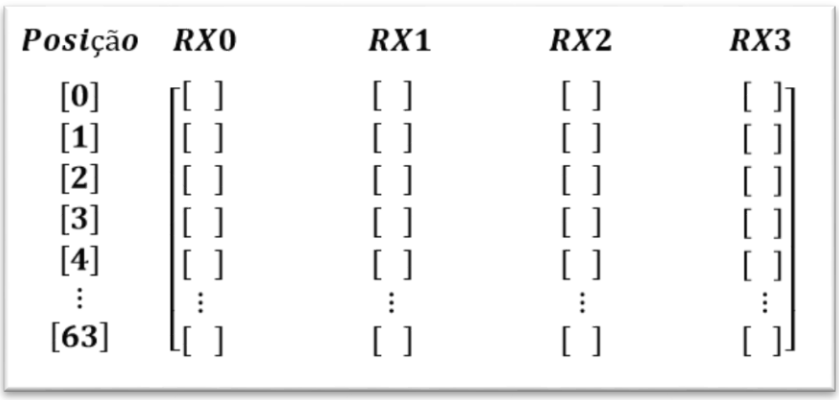
SerialWrite

```
void SerialWrite(uint8_t port,uint8_t c){

 switch (port) {

    case 0: serialize8(c);UartSendData(); break;
    case 1: while (!(UCSR1A & (1 << UDRE1))) ; UDR1 = c; break;
    case 2: while (!(UCSR2A & (1 << UDRE2))) ; UDR2 = c; break;
    case 3: while (!(UCSR3A & (1 << UDRE3))) ; UDR3 = c; break;
  }
}
```

14.3 Técnica de Recepção

A recepção de dados é feita utilizando buffers para as portas RX0, RX1, RX2 e RX3, resultando numa matriz com 4 colunas, “serialBufferRX”. Cada coluna possui 64 linhas que podem armazenar variáveis de 8 bits sem sinal, totalizando em 64 bytes para cada porta de recepção. A Figura 40 demonstra com mais clareza a disposição da matriz.

Figura 40 - Matriz "seiralBufferRX".



Fonte: Autoria própria.

Dois vetores, “serialHeadRX” e “serialTailRX”, são declarados com 4 posições cada para auxiliar na armazenagem e leitura de dados, respectivamente, em cada coluna.

A função “store_uart_in_buf” é responsável por armazenar os dados receptados pelas portas RX0, RX1, RX2 e RX3, através de interrupções de recepção completa.

As interrupções aqui mencionadas são referentes à biblioteca “<avr/interrupt.h>”, onde encontram-se os vetores de interrupção “USART0_RX_vect”, “USART1_RX_vect”, “USART2_RX_vect”, “USART3_RX_vect” referentes às flags “Receive Complete 0”, “Receive Complete 1”, “Receive Complete 2”, “Receive Complete 3”, respectivamente.

Uma recepção tem início quando um start bit é detectado. Os bits recebidos em sequência, conforme a configuração do frame na Figura 37, são amostrados e transferidos para o registrador de deslocamento de recepção até o primeiro stop bit ser detectado. Neste instante, o conteúdo transferido para o registrador de deslocamento de recepção é passado para o buffer de recepção (UDRn), e a flag RXCn, referente à interrupção em discussão, alterna-se para nível lógico alto. (LIMA e VILLAÇA, 2012, p. 352).

store_uart_in_buf

```
static void inline store_uart_in_buf(uint8_t data, uint8_t portnum)
{
  uint8_t h = serialHeadRX[portnum];

  if (++h >= RX_BUFFER_SIZE) h = 0;
  if (h == serialTailRX[portnum]) return;
  serialBufferRX[serialHeadRX[portnum]][portnum] = data;

  serialHeadRX[portnum] = h;

}

ISR(USART1_RX_vect) { store_uart_in_buf(UDR1, 1); }
ISR(USART0_RX_vect) { store_uart_in_buf(UDR0, 0); }
ISR(USART2_RX_vect) { store_uart_in_buf(UDR2, 2); }
ISR(USART3_RX_vect) { store_uart_in_buf(UDR3, 3); }
```

A função “store_uart_in_buf” tem como parâmetro o dado recebido através do buffer de recepção e a porta referente a esse dado (coluna da matriz). O mesmo método utilizado para escrever no vetor “bufTX”, explicada na técnica de transmissão,

é utilizado aqui para armazenar os valores recebidos numa coluna específica da matriz “serialBufferRX”, porém com um adendo.

Pode-se observar que há a verificação de extrapolação do tamanho limite do buffer, assim como na técnica de transmissão, a diferença é que ocorre uma outra verificação (serialHeadRX = serialTailRX) para garantir que um dado recebido não vá subscrever um outro dado antes de sua leitura.

Uma outra função, “SerialRead”, é utilizada para fazer a leitura dos valores armazenados na matriz “serialBufferRX”.

SerialRead

```
uint8_t SerialRead(uint8_t port) {
  uint8_t t = serialTailRX[port];
  uint8_t c = serialBufferRX[t][port];
  if (serialHeadRX[port] != t) {
    if (++t >= RX_BUFFER_SIZE) t = 0;
    serialTailRX[port] = t;
  }
  return c;
}
```

A função recebe como parâmetro a porta (coluna da matriz) referente à recepção. Após feito isso, é identificado e guardado o valor da posição, referente à porta de recepção, do vetor “serialTailRX” numa nova variável, “t”. Outra variável, “c”, também criada dentro da função, recepciona o valor já armazenado na matriz e o retorna para a leitura.

Há uma verificação se as posições “serialHeadRX” e “serialTailRX” da porta de recepção específica são diferentes. Se sim, ocorre o incremento da posição de “serialTailRX” junto com a verificação de extrapolação de tamanho do buffer. Isso apenas verifica se há ou não novos dados para serem lidos no buffer.

14.4 Protocolo de Comunicação

O protocolo de comunicação *MultiWii* consiste em uma sequência de operações indispensáveis para o envio e recepção de dados pelas portas TX0 e RX0, ou seja, a comunicação entre placa controladora e o software GUI.

O protocolo de comunicação se inicia sempre com a recepção de uma sequência correta de dados pela placa, estes podem ser enviados pela GUI, ou por

qualquer hospedeiro que cumpra essa sequência. A Tabela 10 demonstra essa sequência obrigatória de dados na recepção.

Tabela 10 – Sequência de dados a serem recepcionados pela placa controladora. (caracteres em ASCii)

Caracter 1	Caracter 2	Caracter 3	Caracter 4	Caracter 5
$	M	$<$	$datasize$	$cmdMSP$

Fonte: Autoria Própria

Sendo $ e M apenas caracteres de segurança, $<$ um caractere para indicar a direção de comunicação, $datasize$ o tamanho dos dados requisitados, e $cmdMSP$ o comando de requisição.

A Tabela 11 ilustra os dados que são administrados por esse protocolo junto com seu valor de identificação relacionado com a variável “cmdMSP”. Indica-se também a direção de envio de dados, se é da controladora para o software (C => S) ou do software para a controladora (S => C).

Tabela 11 - Protocolo Multiwii.

Tipo de dados	Direção	cmdMSP	Descrição
MSP_IDENT	C => S	100	Tipo do quadrirrotor, versão do *MultiWii* e versão do protocolo.
MSP_STATUS	C => S	101	Tempo do ciclo, erros de comunicação, presença de sensores e modos de voo.
MSP_RAW_IMU	C => S	102	Os 9 graus de liberdade da IMU. Valores crus do acelerômetro,

			giroscópio e magnetômetro.
MSP_SERVO	C => S	103	Fora de uso.
MSP_MOTOR	C => S	104	Rotação dos motores.
MSP_RC	C => S	105	Valores do Receptor.
MSP_RAW_GPS	C => S	106	Fora de uso.
MSP_COMP_GPS	C => S	107	Fora de uso.
MSP_ATTITUDE	C => S	108	Ângulos de atitude.
MSP_ALTITUDE	C => S	109	Altura Estimada.
MSP_BAT	C => S	110	Fora de uso.
MSP_RC_TUNING	C => S	111	Configuração de parâmetros do rádio controle.
MSP_PID	C => S	112	Ganhos do controle.
MSP_BOX	C => S	113	Caixas do software.
MSP_MISC	C => S	114	Fora de uso.
MSP_MOTOR_PINS	C => S	115	Pinos de saída PWM para os motores.
MSP_BOXNAMES	C => S	116	Nomes das caixas do software.
MSP_PIDNAMES	C => S	117	Nomes dos ganhos.
MSP_WP	C => S	118	Fora de uso.
MSP_SET_RAW_RC	S => C	200	Valores do Receptor.
MSP_SET_RAW_GPS	S => C	201	Fora de uso.
MSP_SET_PID	S => C	202	Ganhos do controle.
MSP_SET_BOX	S => C	203	Nomes das caixas de software.
MSP_SET_RC_TUNING	S => C	204	Configuração de parâmetros do rádio controle.
MSP_ACC_CALIBRATION	S => C	205	Requisição de calibração do acelerômetro.

MSP_MAG_CALIBRATION	S => C	206	Requisição de calibração do magnetômetro.
MSP_SET_MISC	S => C	207	Fora de uso.
MSP_RESET_CONF	S => C	208	Reseta configurações da EEPROM.
MSP_WP_SET	S => C	209	Fora de uso.
MSP_EEPROM_WRITE	S => C	250	Escreve na memória EEPROM.
MSP_DEBUG	S => C	254	Fora de uso.

Fonte: Autoria própria.

Na Tabela 11, ainda se encontra a identificação de dados que foram retirados na simplificação do programa. Estes estão destacados em vermelho e são os dados responsáveis pela comunicação dos servos motores, do GPS e da telemetria.

Dependendo da direção de envio de dados, leitura (C => S) ou envio (S => C), tem-se uma abordagem diferente no firmware, sendo isso administrado pela função “evaluateCommand” – função esta acessada apenas após uma requisição lida pela controladora -, que será apresentada no final desde capítulo. A Tabela 12 demonstra as diferenças aplicadas ao protocolo para uma requisição de leitura, requisição de envio e uma requisição que não cumpra a sequência de caracteres apresentada na Tabela 10.

Os dados recebidos ou enviados podem conter vários bytes, saber o seu tamanho e a sequência do protocolo para determinada requisição é fundamental para uma operação correta.

Tabela 12 – Protocolo de comunicação.

REQUISIÇÃO DE LEITURA (software lendo $DADOS$)							
REQUISIÇÃO	$	M	$<$	$datasize$	$cmdMSP$		
RETORNO	$	M	$>$	$datasize$	$cmdMSP$	$DADOS$	$checksum$

REQUISIÇÃO DE ENVIO (software enviando $DADOS$)							
REQUISIÇÃO	$	M	$<$	$datasize$	$cmdMSP$	$DADOS$	$checksum$
RETORNO	$	M	$>$	0	$cmdMSP$	$checksum$	

ERRO DE REQUISIÇÃO							
REQUISIÇÃO	$	M	$<$	$datasize$	$erro$		
RETORNO	$	M	!	0	$cmdMSP$	$checksum$	

Fonte: Autoria Própria.

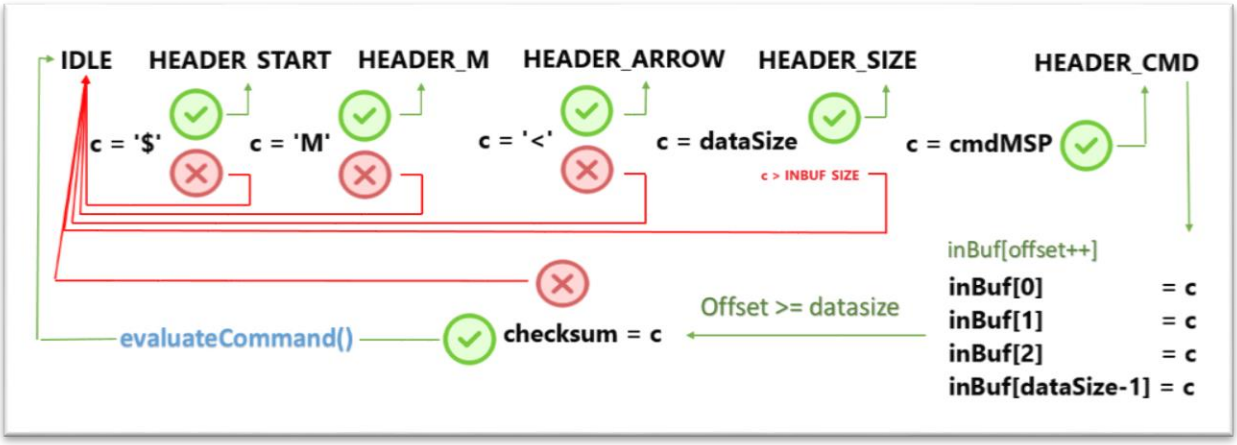
A leitura da requisição pela controladora pode ser entendida analisando a função “serialCom”. Esta inicia-se com a declaração de variáveis fundamentais para a sua dinâmica, tendo destaque a enum criada, que possui uma variável, “c_state” para evoluir de estado de acordo com os valores recepcionados pela controladora apresentados na Tabela 10. A variável “c_state” assume de início o valor IDLE.

Mantendo a sequência, têm-se a verificação de retorno da função “SerialAvailable” com o parâmetro “0” para a estrutura de repetição “while”. Este retorno apenas será verdadeiro se o valor do vetor “serialHeadRX[0]” for diferente do valor do vetor “serialTailRX[0]”, ou seja, quando há dados a serem recepcionados por essa porta pela placa. Tendo esse retorno verdadeiro, se faz a verificação em relação à transmissão, onde verifica-se se o vetor “bufTX” há no mínimo 40 bytes livres para transmissão – essa verificação é feita devido ao retorno transmitido pela placa, apresentado na Tabela 12 -, caso negado, a função é interrompida, caso confirmado, têm-se continuidade ao protocolo, que tem suas linhas de código explicadas na Figura 41.

serialCom

```
void serialCom() {
  uint8_t c;
  static uint8_t offset;
  static uint8_t dataSize;
  static enum _serial_state {
    IDLE,
    HEADER_START,
    HEADER_M,
    HEADER_ARROW,
    HEADER_SIZE,
    HEADER_CMD,
  } c_state = IDLE;
  while (SerialAvailable(0)) {
    uint8_t bytesTXBuff = ((uint8_t)(headTX-
tailTX))%TX_BUFFER_SIZE;
    if (bytesTXBuff > TX_BUFFER_SIZE - 40 ) return;
    c = SerialRead(0);
    if (c_state == IDLE) {
      c_state = (c=='$') ? HEADER_START : IDLE;
      if (c_state == IDLE) evaluateOtherData(c);
    } else if (c_state == HEADER_START) {
      c_state = (c=='M') ? HEADER_M : IDLE;
    } else if (c_state == HEADER_M) {
      c_state = (c=='<') ? HEADER_ARROW : IDLE;
    } else if (c_state == HEADER_ARROW) {
      if (c > INBUF_SIZE) {
        c_state = IDLE;
        continue;
      }
      dataSize = c;
      offset = 0;
      checksum = 0;
      indRX = 0;
      checksum ^= c;
      c_state = HEADER_SIZE;
    } else if (c_state == HEADER_SIZE) {
      cmdMSP = c;
      checksum ^= c;
      c_state = HEADER_CMD;
    } else if (c_state == HEADER_CMD && offset < dataSize) {
      checksum ^= c;
      inBuf[offset++] = c;
    } else if (c_state == HEADER_CMD && offset >= dataSize) {
      if (checksum == c) {
        evaluateCommand();
      }
      c_state = IDLE;
    }}}
```

Figura 41 - Protocolo de comunicação (Recepção).



Fonte: Autoria Própria

A Figura 41 ilustra os passos das linhas de códigos remanescentes da função “serialCom”. Apresentando a verificação dos dados recebidos através de ternários, o preenchimento do vetor “inBuf” e a verificação de erros de recepção através da variável “cheksum” para a chamada da função “evaluateCommand”.

O vetor inBuf possui 64 posições de 8 bits, portanto é um buffer que armazena os dados recebidos pela porta RX0, deixando-os prontos para futura leitura através da chamada das funções “read8”, “read16” ou “read32”, todas apresentadas a seguir.

read

```
static uint8_t indRX;

uint32_t read32() {
  uint32_t t = read16();
  t+= (uint32_t)read16()<<16;
  return t;
}
uint16_t read16() {
  uint16_t t = read8();
  t+= (uint16_t)read8()<<8;
  return t;
}
uint8_t read8()  {
  return inBuf[indRX++]&0xff;
}
```

Observa-se que a função “read8” faz o retorno de um valor de 8 bits armazenado no vetor “inBuf”, onde a posição de leitura é incrementada através da

variável “indRX”. A função “read16” é a função “read8” chamada duas vezes com um deslocamento para esquerda de 8 bits entre as duas chamadas. A função “read32” é a função “read16” chamada duas vezes com um deslocamento para esquerda de 16 bits entre as duas chamadas.

Tendo em mente como é feita a leitura da requisição do protocolo pela controladora, é apresentado agora as funções responsáveis pela transmissão do retorno (Tabela 12) pela placa controladora. Estas funções serão chamadas dentro da função “evaluateCommand”.

headSerialResponse / headSerialReply / headSerialError

```
static uint8_t cmdMSP;
static uint8_t checksum;

void headSerialResponse(uint8_t err, uint8_t s) {
  serialize8('$');
  serialize8('M');
  serialize8(err ? '!' : '>');
  checksum = 0;
  serialize8(s);
  serialize8(cmdMSP);
}
void headSerialReply(uint8_t s) {
  headSerialResponse(0, s);
}
void inline headSerialError(uint8_t s) {
  headSerialResponse(1, s);
}
```

As funções “headSerialReply” e “headSerialError” apenas passam parâmetros para a função “headSerialResponse”, parâmetros esses que correspondem à indicativa de erro no requerimento e o tamanho do dado a ser transmitido.

A “headSerialResponse” recebendo esses parâmetros, envia uma sequência de dados ao software através da função “serialize8”. A sequência ('$' 'M' '>' ‘s’ ‘cmdMSP’) para a função “headSerialReply” e a sequência ('$' 'M' '!' s cmdMSP) para a função “headSerialError”. Observa-se que essa sequência é similar à sequência do protocolo de requisição, alterando apenas o caractere “<” pelo “>”, indicando a direção dos dados.

Por fim, a função “evaluateCommand” faz a gerência de comunicação entre todos os dados a partir do “cmdMSP” enviado pelo software. Um exemplo de recepção e transmissão é dado abaixo.

evaluateCommand

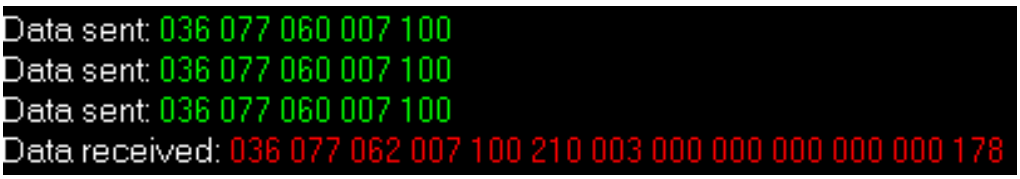
```
void evaluateCommand() {
  switch(cmdMSP) {
    case MSP_SET_RAW_RC:  //recepção
     for(uint8_t i=0;i<8;i++) {
       rcData[i] = read16();
     }
     headSerialReply(0);
     break;
    case MSP_IDENT:       //transmissão
     headSerialReply(7);
     serialize8(VERSION);
     serialize8(MULTITYPE);
     serialize8(MSP_VERSION);
     serialize32(0);
     break;
  }
  tailSerialReply();
}
```

Para a apresentação completa da função “evaluateCommand” deve-se consultar o código completo no GITHUB[10].

A função “tailSerialReply” envia o valor de “checksum” ao software para verificação de erros na transmissão e reabilita a interrupção de transmissão, pois a mesma estaria desabilitada, caso houvesse uma transmissão completa anteriormente.

A Figura 42 exemplifica o protocolo de comunicação, tendo uma requisição através do RS232 Analyser e um retorno vindo da controladora.

Figura 42 – Analise do protocolo pelo RS232 Analyser.



Fonte: Autoria Própria

Na Figura 42, o requerimento enviado pelo RS232 Analyser é:

$$036,077,060,007,100 = \$,M,<,7,100$$

[10] GITHUB CODE [Online]. Disponível: https://github.com/umbertoxavier/MultiWiiLasiscVersion. [Acesso em Julho de 2019].

Ou seja, se faz um requerimento de leitura (C=>S) do “cmdMSP” = 100 (MSP_IDENT), que possui 7 bytes de dados. Por fim, os dados retornados estão listados em sequência na Tabela 13.

Tabela 13 – Dados retornados pela controladora (cmdMSP = 100).

Valor em decimal	Correspondente
$036, 077, 062, 007, 100$	$\$, M, >, 7, 100$
210	VERSION
003	MULTITYPE
000	MSP_VERSION
000	Serialize8(0)
000	Serialize8(0)
000	Serialize8(0)
000	Serialize8(0)
178	Checksum.

Fonte: Autoria Própria.

É importante frisar que a variável $ckecksum$ é o cálculo de “ou exclusivos” (XOR) dos caracteres e dados da requisição do protocolo, a partir do $datasize$. Portando de acordo com a função “serialCom”, mais precisamente no preenchimento do vetor “inBuf”, e de acordo com a Tabela 12, para existir a chamada da função “evaluateCommand” é necessário que:

- Requisição de Leitura: Seja enviada 3 requisições seguidas sem o $ckecksum$, assim o $ckecksum$ calculado é comparado com os dados das duas últimas requisições armazenados em “inBuf”;
- Requisição de Escrita: Seja enviada apenas 1 requisição junto com o $ckecksum$ calculado.

15 EEPROM.INO

Um sistema microprocessado contém, basicamente, duas memórias, uma memória de programa, que armazena o código a ser executado, e a memória de dados, onde os dados de trabalho da CPU podem ser escritos e lidos. Nos microcontroladores mais atuais, são empregados dois tipos de memórias regraváveis eletricamente, a memória flash EEPROM para o programa, e a standard EEPROM para armazenamento de dados que não devem ser perdidos. Para a armazenagem de dados que não precisam ficar retidos após a desenergização do circuito, se emprega uma memória volátil, chamada de memória SRAM, sendo essa uma memória para armazenagem temporária de dados do programa usuário. (LIMA e VILLAÇA, 2012).

É interessante a ideia de manter valores de configuração mesmo após a desenergização da controladora de um quadrirrotor, e estes estarem disponíveis na reinicialização. Com o objetivo dessa aplicação, utiliza-se a gravação na memória standard EEPROM do ATMega2560, com o auxílio da biblioteca <avr/eeprom.h>.

15.1 Estudo do Firmware (EEPROM.ino)

O bloco de código abaixo ilustra a leitura e escrita, respectivamente, das configurações presentes em uma estrutura (“conf”).

eeprom_read_block/eeprom_write_block

```
void readEEPROM() {
  eeprom_read_block((void*)&conf, (void*)0, sizeof(conf));
}

void writeParams(uint8_t b) {
  conf.checkNewConf = EEPROM_CONF_VERSION;
  eeprom_write_block((const void*)&conf, (void*)0, sizeof(conf));
  readEEPROM();
  if (b == 1) blinkLED(15,20,1);
}
```

A função “eeprom_read_block” da biblioteca <avr/eeprom.h> faz a leitura da EEPROM para a SRAM. Ou seja, faz a leitura de um bloco do tamanho da estrutura “conf” a partir do endereço 0x00 da EEPROM, e repassa os valores para a memória SRAM das variáveis da estrutura “conf”.

A função “eeprom_write_block” da biblioteca <avr/eeprom.h> faz a escrita na EEPROM a partir de valores da SRAM. Ou seja, faz a leitura dos valores da estrutura “conf” na memória SRAM, e escreve estes valores no endereço 0x00 da EEPROM, considerando o tamanho da estrutura.

A reescrita de dados pode ocorrer por mais de um motivo, podendo ser a alteração de ganhos e configurações pela GUI, ou a calibração do acelerômetro e magnetômetro, por exemplo.

A função “writeParams” é a responsável por reescrever dados na EEPROM, e desde a primeira vez que esta reescreve um dado, a variável “conf.checkNewConf” recebe um valor de referência para saber se esta configuração se difere do padrão ou não.

Quando se deseja resetar os dados de configuração para o padrão, a variável “conf.checkNewConf” é incrementada por uma unidade a partir do protocolo de comunicação USART. Esse incremento faz a diferenciação na condição feita na função “checkfirstTime”, apresentada no código abaixo.

checkfirstTime

```
void checkFirstTime() {
  if (EEPROM_CONF_VERSION == conf.checkNewConf) return;
  conf.P8[ROLL]  = 40;  conf.I8[ROLL] = 30; conf.D8[ROLL]  = 23;
  conf.P8[PITCH] = 40; conf.I8[PITCH] = 30; conf.D8[PITCH] = 23;
  conf.P8[YAW]   = 85;  conf.I8[YAW]  = 45;  conf.D8[YAW]  = 0;
  conf.P8[PIDALT]   = 16; conf.I8[PIDALT]   = 15; conf.D8[PIDALT]
= 7;

  conf.P8[PIDLEVEL] = 70; conf.I8[PIDLEVEL] = 10; conf.D8[PIDLEVEL]
 = 100;

  conf.P8[PIDMAG] = 40;

  conf.rcRate8 = 90; conf.rcExpo8 = 60;
  conf.rollPitchRate = 0;
  conf.yawRate = 0;
  conf.dynThrPID = 0;
  conf.thrMid8 = 50; conf.thrExpo8 = 0;

for(uint8_t i=0;i<CHECKBOXITEMS;i++) {conf.activate[i] = 0;}
  conf.angleTrim[0] = 0; conf.angleTrim[1] = 0;
  conf.powerTrigger1 = 0;
  writeParams(0); // this will also (p)reset checkNewConf with the
current version number again.
}
```

A Tabela 14 apresenta os dados da estrutura “conf” junto com seu valor padrão na EEPROM. Para poupar informações, os dados de modos de voo secundários e funções extras não estão disponível na tabela.

Tabela 14 - Estrutura de configuração e seus valores padrão.

Variável	V.P	Descrição
checkNewConf	161	Checa se houve configuração diferente ou não da inicial na EEPROM
P8[ROLL]	40	Ganho proporcional (Acro Mode)
I8[ROLL]	30	Ganho integrativo (Acro Mode)
D8[ROLL]	23	Ganho derivativo (Acro Mode)
P8[PITCH]	40	Ganho proporcional (Acro Mode)
I8[PITCH]	30	Ganho integrativo (Acro Mode)
D8[PITCH]	23	Ganho derivativo (Acro Mode)
P8[YAW]	16	Ganho proporcional (Acro Mode)
I8[YAW]	15	Ganho integrativo (Acro Mode)
D8[YAW]	7	Ganho derivativo (Acro Mode)
P8[PIDALT]	16	Ganho proporcional (Altitude Hold)
I8[PIDALT]	15	Ganho integrativo (Altitude Hold)
D8[PIDALT]	7	Ganho derivativo (Altitude Hold)
P8[PIDLEVEL]	70	Ganho proporcional (Level Mode)
I8[PIDLEVEL]	10	Ganho integrativo (Level Mode)
D8[PIDLEVEL]	100	Ganho derivativo (Level Mode)
P8[PIDMAG]	40	Ganho proporcional (Mag Mode)
rcRate8	90	Variável para configuração das curvas de comando dos canais de rolagem, arfagem e guinada
rcExpo8	60	Variável para configuração das curvas de comando dos canais de rolagem, arfagem e guinada

rollPitchRate	0	Taxa para atenuar ação de controle nos movimentos de rolagem e arfagem
yawRate	0	Taxa para atenuar ação de controle no movimento de guinada
dynThrPID	0	Taxa para atenuar o ganho proporcional e derivativo dos movimentos de rolagem e arfagem de acordo com a potência.
thrMid8	50	Variável para configuração das curvas de comando do canal de potência
thrExpo8	0	Variável para configuração das curvas de comando do canal de potência
accZero[3]	A calcular	Valores de calibração do acelerômetro
magZero[3]	A calcular	Valores de calibração do magnetômetro
angleTrim[2]	0	Ajuste fino á angulação do quadrirrotor {rolagem, arfagem}
activate[CHECKBOXITEMS]	0	Configuração das opções selecionáveis pelo rádio controle (software envia essa variável, e seu valor muda de acordo com a configuração)

Fonte: Autoria Própria.

16 SENSORS.INO

16.1 TWI (Two Wire Serial Interface) – I2C

Antes de apresentar os sensores e suas configurações para comunicação, é necessário falar sobre o protocolo Inter-Integrated Circuit (I2C), que nos microcontroladores AVR é conhecido como Two Wire Serial Interface (TWI) por questão de royalties. (LIMA e VILLAÇA, 2012; ATMEGA 2560 Datasheet). Neste trabalho quando for citado I2C ou TWI é compreendida a singularidade entre os dois termos.

A interface serial de duas vias (do inglês, Two Wire Serial Interface – TWI) é ideal para inúmeras aplicações típicas de comunicação dos microcontroladores, sendo utilizada em um grande número de circuitos integrados, sendo estes empregados como circuitos periféricos, como memórias EEPROM, sensores de temperatura, barômetros, acelerômetros, giroscópios e magnetômetros. (LIMA e VILLAÇA, 2012; ATMEGA 2560 Datasheet).

O protocolo TWI utiliza apenas duas vias bidirecionais, uma para o clock serial (SCL), responsável por sincronizar o transmissor e o receptor, e outra para dados seriais (SDA), responsável por transportar endereços, controles e dados. É permitido vários mestres no mesmo barramento, porém é mais comum encontrar um único mestre e diversos escravos. O endereçamento no ATMEGA é de 7bits, permitindo a conexão de até 128 dispositivos diferentes. Como pode-se observar na Figura 43, o único hardware externo necessário para implementar o barramento é um resistor de pull-up para cada via. (LIMA e VILLAÇA, 2012; ATMEGA 2560 Datasheet).

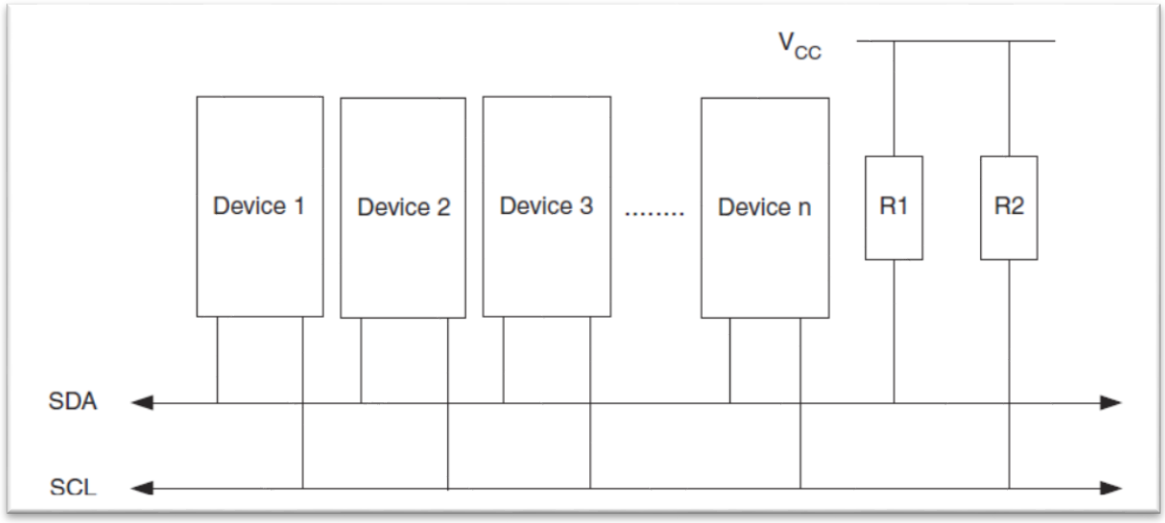
Todos os dispositivos conectados no barramento possuem endereços individuais, sendo que os fabricantes dos dispositivos pagam à Philips, desenvolvedora do protocolo I2C, para obter um endereço para seu circuito integrado. Isso significa que cada dispositivo possui um endereço único que o referencia. Quando deseja-se conectar dispositivos iguais ao barramento, são fornecidas soluções alternativas por cada dispositivo. (LIMA e VILLAÇA, 2012; ATMEGA 2560 Datasheet).

Os dispositivos conectados ao barramento TWI podem ser classificados como mestre ou escravo. O mestre é o responsável por iniciar a transmissão, terminar a transmissão e gerar o sinal de clock (SCL). O escravo é o dispositivo que possui o

endereço único e responde aos comandos do mestre. Um dispositivo pode ser apenas mestre, apenas escravo ou comutar entre esses dois. (LIMA e VILLAÇA, 2012; ATMEGA 2560 Datasheet).

A principal ideia da comunicação I2C, é que, quando um mestre inicia uma mensagem com o endereço do escravo no início desta, todos os dispositivos no barramento recebem a mensagem, porém só o escravo com o endereço reconhecido responde. (LIMA e VILLAÇA, 2012; ATMEGA 2560 Datasheet).

Figura 43 - Barramento I2C com N dispositivos.



Fonte: (ATMEGA 2560 Datasheet).

As especificações do protocolo I2C para o ATMEGA 2560 são:

- Operação mestre e escravo suportadas;
- Endereçamento de 7 bits;
- Velocidade de até 400 KHz na transferência de dados;
- Drives de saída com limitação da taxa de subida (Slew Rate);
- Circuito atenuador de ruídos no barramento;
- O reconhecimento do endereço pode causar o “Wake-up” do AVR quando em modo Sleep.

A limitação de 128 dispositivos, dado o endereçamento de 7 bits, só é contrariada pela limitação da máxima capacitância ao barramento (400pF). Essa

capacitância e a velocidade de transferência dos dados têm influência direta no cálculo dos resistores de pull-up. (LIMA e VILLAÇA, 2012; ATMEGA 2560 Datasheet).

Sendo C_b a capacitância do barramento, tem-se os valores máximos e mínimos das resistências de pull-up pelas equações [33], [34], [35] e [36].

$$R_{\min(f_{SCL=100KhZ})} = \frac{V_{cc} - 0.4V}{3mA} \quad [33]$$

$$R_{\max(f_{SCL=100KhZ})} = \frac{1000ns}{C_b} \quad [34]$$

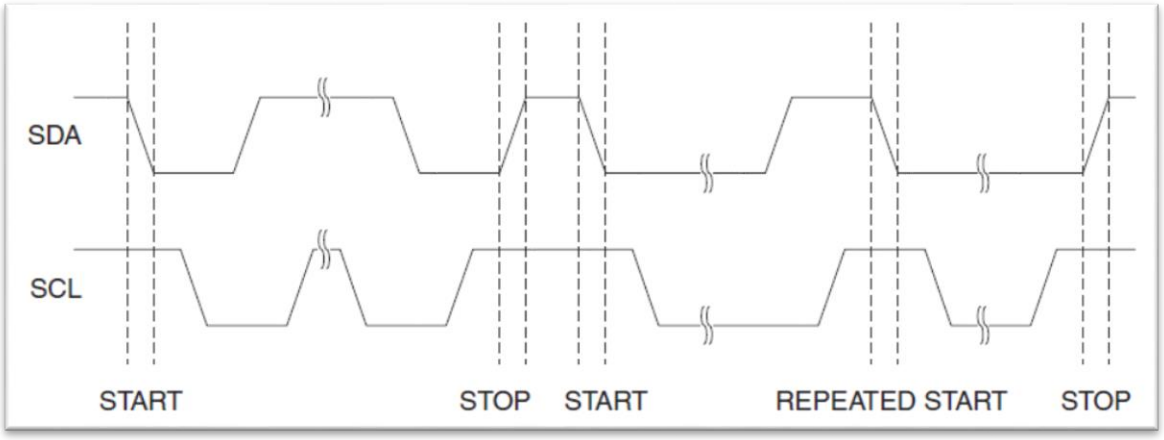
$$R_{\min(f_{SCL=400KhZ})} = \frac{V_{cc} - 0.4V}{3mA} \quad [35]$$

$$R_{\max(f_{SCL=400KhZ})} = \frac{300ns}{C_b} \quad [36]$$

Cada bit transferido para o barramento (SDA) é acompanhado por um pulso na linha de clock (SCL). O nível lógico da via de dados deve se manter estável enquanto a via do clock estiver alta. (ATMEGA 2560 Datasheet).

O mestre inicia e termina a transmissão de dados. A transmissão é iniciada quando o mestre executa uma condição de início (START) no barramento, e é terminada quando o mestre executa uma condição de parada (STOP). Entre a condição de START e STOP, o barramento é considerado ocupado, e nenhum outro mestre deve tentar tomar o controle deste. Um caso especial ocorre quando uma nova condição de START é executada entre as condições de START e STOP, chamando-a de condição de início repetido (REPEATED START), necessitando de uma próxima condição de STOP para terminar a transmissão. Essas três condições podem ser analisadas na Figura 44. É visto que as condições de START e REPEATED START ocorrem sob a mesma operação (SDA muda de alto para baixo quando SCL está alto), sendo esta o contrário da condição de STOP (SDA muda de baixo para alto quando SCL está alto). (ATMEGA 2560 Datasheet).

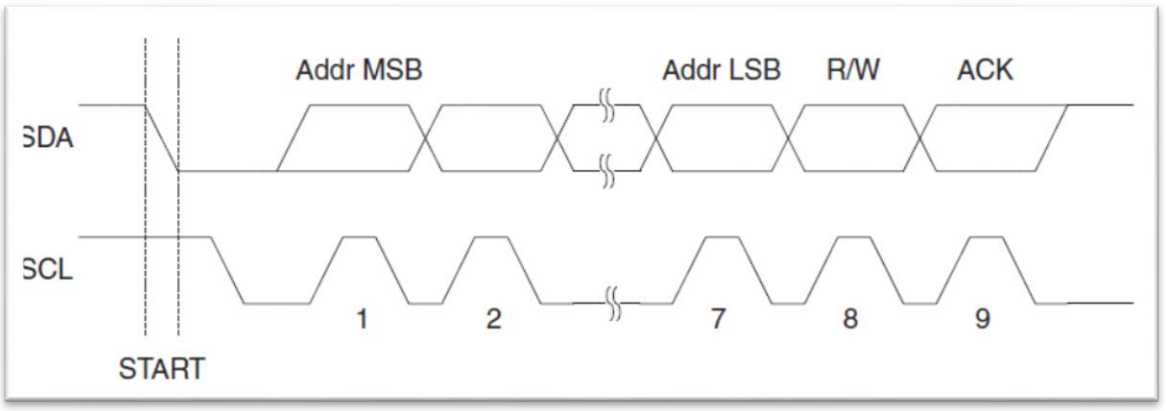
Figura 44 – Condições START, STOP e REPEATED START.



Fonte: (ATMEGA 2560 Datasheet).

Todo pacote de endereço (Figura 45) transmitido pelo barramento TWI contem 9 bits, sendo 7 bits referente ao endereço de um dispositivo, um bit de controle de leitura/escrita (R/W) e um bit de entendimento (ACK). Se o bit R/W está setado (1), uma operação de leitura é requisitada, caso contrário é requisitada uma operação de escrita. Quando um escravo reconhece seu endereço, este deve enviar um ACK (SDA baixo no nono ciclo SCL). Se o escravo endereçado está ocupado, ou por algum outro motivo não pode servir a requisição do mestre, o nono ciclo é mantido alto (NACK), podendo o mestre então, transmitir uma condição STOP ou uma condição REPEATED START para iniciar uma nova transmissão. (ATMEGA 2560 Datasheet).

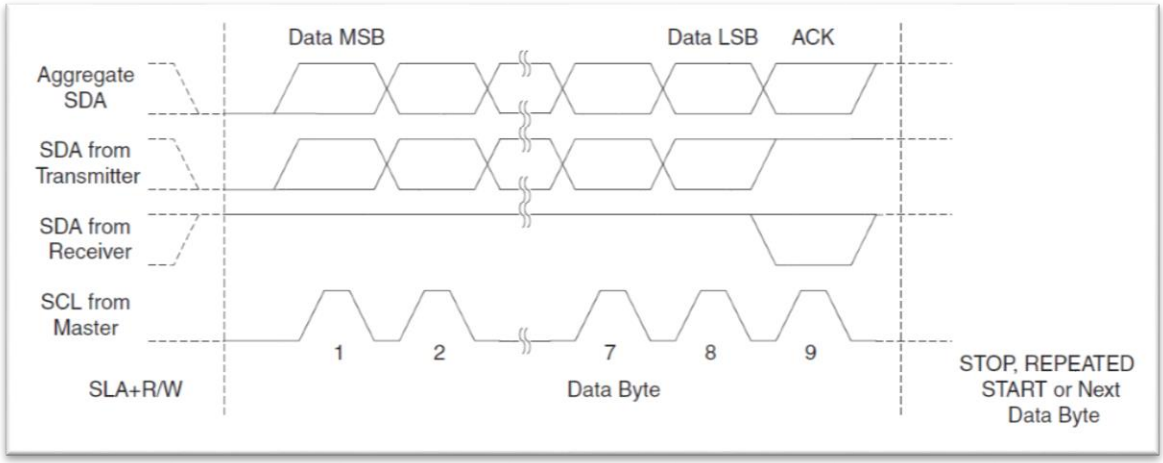
Figura 45 – Pacote de endereço.



Fonte: (ATMEGA 2560 Datasheet).

Todo pacote de dados transmitidos pelo barramento TWI contem 9 bits, sendo 1 byte de dados e um bit de entendimento (ACK). Durante uma transferência de dados, o mestre gera o clock e as condições de START e STOP, enquanto o receptor é responsável por reconhecer a recepção (ACK). Caso o receptor receber o último byte, ou por alguma razão não puder receber mais bytes, este deve informar o transmissor, enviando um NACK depois do último byte. A Figura 46 ilustra esse processo. (ATMEGA 2560 Datasheet).

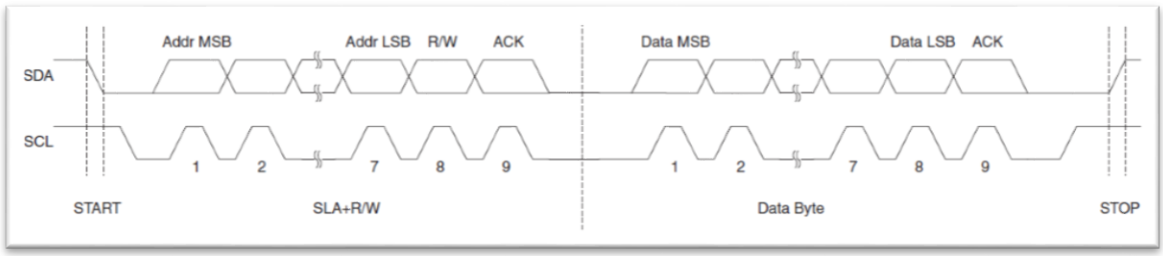
Figura 46 – Pacote de dados.



Fonte: (ATMEGA 2560 Datasheet).

Juntando o pacote de endereço com o pacote de dados, tem-se o processo de como se segue uma comunicação TWI. A comunicação basicamente consiste de uma condição START, os bits Addr+R/W, um ou mais pacotes de dados e uma condição STOP. A Figura 47 demonstra essa junção de pacotes. (ATMEGA 2560 Datasheet).

Figura 47 – Pacote de endereço e dados.



Fonte: (ATMEGA 2560 Datashee

16.2 Funções Gerais do Protocolo TWI

Tendo o conceito da comunicação TWI, nesta secção serão apresentadas as funções gerais do protocolo TWI aplicadas no firmware do *MultiWii Pro.* Mais adiante serão apresentadas a aplicação de cada uma dessas funções na configuração e leitura de dados dos sensores inerciais.

I2c_init

```
void i2c_init(void) {
  I2C_PULLUPS_DISABLE
  TWSR = 0;
  TWBR = ((F_CPU / I2C_SPEED) - 16) / 2;
  TWCR = 1<<TWEN;
}
```

A função “I2c_init” é responsável pela inicialização da comunicação TWI, nela encontra-se a configuração da frequência do sinal SCL, a partir do registrador da taxa de bit (TWBR) e dos bits de prescaler (TWPS) do registrador de status do TWI (TWSR). Mais precisamente, a frequência do sinal SCL é dada por:

$$f_{SCL} = \frac{f_{osc}}{16 + (2 \times TWBR \times TWPS)} \quad [37]$$

$$TWBR = \frac{\frac{f_{osc}}{f_{SCL}} - 16}{2 \times TWPS} \quad [38]$$

Considerando um prescaler de 1 (TWPS1 = 0; TWPS0 = 0). Sendo a frequência de trabalho da CPU $f_{osc} = 16MHz$, para um sinal SCL de frequência $f_{SCL} = 100KHz$, tem-se:

$$TWBR = \frac{\frac{16MHz}{100KHz} - 16}{2 \times 1} = 72 \quad [39]$$

Lembrando que esse valor pode ser alterado de acordo com a limitação do dispositivo escravo, podendo chegar em até 400KHz.

Por fim, a função “I2c_init”, através do registrador de controle do TWI (TWCR) e seu bit TWEN, habilita a operação TWI e ativa a interface TWI.

O macro “I2C_PULLUPS_DISABLE” apenas garante que os registradores de pull-up nos pinos PD0 (SCL) e PD1 (SDA) do ATMEGA2560 estejam desabilitados, mantendo assim a característica da comunicação.

I2c_rep_start

```
void i2c_rep_start(uint8_t address) {
  TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);
  waitTransmissionI2C();
  TWDR = address;
  TWCR = (1<<TWINT) | (1<<TWEN);
  waitTransmissionI2C();
}
```

A função “I2c_rep_start” é responsável por enviar uma condição START ou REPEATED START e em seguida o Addr+R/W do protocolo de endereço.

Uma condição START é enviada escrevendo os seguintes valores no registrador de controle (TWCR): TWEN = 1; TWINT = 1; TWSTA = 1. Onde TWEN deve ser setado para habilitar a interface TWI, TWSTA deve ser 1 para transmitir uma condição START e TWINT deve ser 1 para limpar a flag TWINT. Assim, será gerada uma condição START o mais breve a partir do momento que o barramento estiver livre.

A flag TWINT é ativa (TWINT = 0) por hardware quando o TWI conclui sua atividade corrente e espera uma resposta do software de aplicação. A função “waitTransmissionI2C” espera um determinado tempo para que a flag TWINT seja ativa, caso esse tempo seja excedido, incrementa-se um contador de erro e a comunicação TWI é desabilitada.

Após a condição de START ser transmitida, e por isso a flag TWINT ser setada, deve ser transmitido o protocolo de endereço Addr+R/W. Isto é feito escrevendo Addr+R/W no registrador de dados TWI (TWDR). Depois disso TWINT deve ser 1 para limpar a flag e continuar a transferência. Chama-se então a função “waitTransmissionI2C” para esperar Addr+R/W ser transmitido (portanto, a flag TWINT ser ativa) e o bit de ACK ser recebido.

De acordo com o valor de R/W, é feita uma requisição de leitura ou escrita, tendo em sequência a chamada de funções específicas para cada requisição.

I2c_write

```
void i2c_write(uint8_t data ) {
  TWDR = data;
  TWCR = (1<<TWINT) | (1<<TWEN);
  waitTransmissionI2C();
}
```

Quando SLA+W (**requisição de escrita**) for transmitido com sucesso, um pacote de dados deve ser transmitido em sequência. Isto é feito a partir da função “i2c_write”, escrevendo o byte de dados no registrador de dados TWI (TWDR). Após atualizar TWDR, a flag TWINT deve ser abaixada para continuar a transferência. Chama-se então a função “waitTransmissionI2C” para esperar o byte ser transmitido.

A função “i2c_write” pode ser chamada em sequência até o último byte ser enviado e então a transferência pode ser terminada na geração da condição STOP ou REPEATED START.

I2c_read

```
uint8_t i2c_read(uint8_t ack) {
  TWCR = (1<<TWINT) | (1<<TWEN) | (ack? (1<<TWEA) : 0);
  waitTransmissionI2C();
  uint8_t r = TWDR;
  if (!ack) i2c_stop();
  return r;
}
```

Quando SLA+R (**requisição de leitura**) for transmitido com sucesso, um pacote de dados deve ser lido em sequência. Isto é feito a partir da função “i2c_read”, armazenando o byte lido através do registrador de dados TWI (TWDR) em uma variável, “r” no caso.

A função “i2c_read” recebe como parâmetro os valores 1 ou 0, realizando as seguintes operações:

- **1 –** Sabe-se que não é o último byte a ser lido, portanto escreve-se os seguintes valores no registrador de controle (TWCR): TWEN = 1; TWINT = 1; TWEA = 1. Onde TWEN deve ser setado para habilitar a interface TWI, TWINT deve ser 1 para limpar a flag TWINT e TWEA deve ser 1 para habilitar a geração de pulso ACK, assim o escravo que está enviando as informações sabe se deve continuar ou não.

Os dados recebidos podem ser lidos através do registrador TWDR apenas quando a flag TWINT estiver setada (TWINT = 0), o que ocorre após a função “waitTransmissionI2C”. É então retornada a variável “r” com o byte lido.

- **0 -** Sabe-se que é o último byte a ser lido, portanto escreve-se os seguintes valores no registrador de controle (TWCR): TWEN = 1; TWINT = 1. Onde TWEN deve ser setado para habilitar a interface TWI e TWINT deve ser 1 para limpar a flag TWINT.
  Após a flag TWINT estiver setada, o que ocorre após a função “waitTransmissionI2C”, inicia-se uma condição de STOP e retorna-se o último byte lido, “r”.

I2c_readAck

```
uint8_t i2c_readAck() {
  return i2c_read(1);
}
```

“I2c_readAck” envia o parâmetro “1” para a função “i2c_read”

I2c_readNak

```
uint8_t i2c_readNak(void) {
  return i2c_read(0);
}
```

“I2c_readNak” envia o parâmetro “0” para a função “i2c_read”

I2c_stop

```
void i2c_stop(void) {
  TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
}
```

A função “I2c_stop” é responsável por gerar uma condição STOP na transmissão TWI. Uma condição STOP é gerada escrevendo-se os seguintes valores no registrador de controle (TWCR): TWEN = 1; TWINT = 1; TWST0 = 1. Onde TWEN deve ser setado para habilitar a interface TWI, TWINT deve ser 1 para limpar a flag TWINT, TWST0 deve ser 1 para transmitir uma condição STOP.

I2c_read_reg_to_buf

```
size_t i2c_read_reg_to_buf(uint8_t add, uint8_t reg, void *buf,
size_t size) {
  i2c_rep_start(add<<1);
  i2c_write(reg);
  return i2c_read_to_buf(add, buf, size);
}
```

I2c_read_to_buf

```
size_t i2c_read_to_buf(uint8_t add, void *buf, size_t size) {
  i2c_rep_start((add<<1) | 1);
  size_t bytes_read = 0;

  uint8_t *b = (uint8_t*)buf;

  while (size--) {
    *b++ = i2c_read(size > 0);
    bytes_read++;

  }
  return bytes_read;
}
```

As funções “I2c_read_reg_to_buf” e “I2c_read_to_buf” são funções sequenciais, respectivamente. Estas são utilizadas para ler os registradores de um escravo, podendo ser registradores de comando ou registradores de dados.

A função “I2c_read_reg_to_buf” recebe como parâmetro as seguintes informações:

- **add –** Endereço do dispositivo escravo sem o bit R/W;
- **rg –** Endereço do registrador do escravo;
- ***buf –** Endereço da variável que irá armazenar os dados lidos;
- **size –** Tamanho dos dados que serão lidos (bytes).

Tendo os parâmetros, é iniciada uma condição START seguida da transmissão de Addr+W, ou seja, requisição de escrita. Escreve-se então o registrador de controle do escravo, qual queira ter acesso.

A função “I2c_read_to_buf” inicia uma REPEATED START seguida da transmissão de Addr+R, ou seja, requisição de leitura. Como exemplo, considerando uma leitura de 4 bytes, tem-se a seguinte sequência de chamadas da função “i2c_read”: (i2c_read(1) -> i2c_read(1) -> i2c_read(1)-> i2c_read(0)). Onde cada

chamada retorna o byte lido para a posição referente do vetor de 4 posições recebido por “*buf”.

I2c_getSixRawADC

```
void i2c_getSixRawADC(uint8_t add, uint8_t reg) {
  i2c_read_reg_to_buf(add, reg, &rawADC, 6);
}
```

“I2c_getSixRawADC” faz a leitura do registrador de dados do escravo através das funções “I2c_read_reg_to_buf” e “I2c_read_to_buf” para o vetor de 6 posições “rawADC”.

I2c_readReg

```
uint8_t i2c_readReg(uint8_t add, uint8_t reg) {
  uint8_t val;
  i2c_read_reg_to_buf(add, reg, &val, 1);
  return val;
}
```

“I2c_readReg” faz a leitura de apenas um byte do registrador de comando do escravo através das funções “I2c_read_reg_to_buf” e “I2c_read_to_buf” para a variável “val”, qual é retornada.

I2c_writeReg

```
void i2c_writeReg(uint8_t add, uint8_t reg, uint8_t val) {
  i2c_rep_start(add<<1);
  i2c_write(reg);
  i2c_write(val);
  i2c_stop();
}
```

“I2c_writeReg” faz a escrita no registrador de comando do escravo, podendo mudar valores de configuração deste registrador. Recebe como parâmetro o endereço do dispositivo escravo (“add”), o endereço do registrador de comando (“reg”) e o valor a ser escrito neste registrador de comando (“val”).

Tendo os parâmetros, é iniciada uma condição START seguida da transmissão de Addr+W (requisição de escrita). Escreve-se em sequência então, o valor do registrador de comando e o valor a ser escrito neste. A transmissão é encerrada com a geração de uma condição STOP.

waitTransmissionI2C

```
void waitTransmissionI2C() {
  uint16_t count = 255;
  while (!(TWCR & (1<<TWINT))) {
    count--;
    if (count==0) {
      TWCR = 0;
      neutralizeTime = micros();
      i2c_errors_count++;
      break;
    }
  }
}
```

A função “waitTransmissionI2C” espera um determinado tempo para que a flag TWINT seja ativa. Tempo este de 255 ciclos de máquina. Se a flag TWINT não for ativa neste período, ou seja, caso haja algum atraso na comunicação, ignora-se a operação atual, incrementa-se um contador de erro e desabilita-se a comunicação TWI.

swap_endianness

```
void swap_endianness(void *buf, size_t size) {
  uint8_t tray;
  uint8_t *from;
  uint8_t *to;
  for (from = (uint8_t*)buf, to = &from[size-1]; from < to; from++,
to--) {
    tray = *from;
    *from = *to;
    *to = tray;
  }
}
```

A função “swap_endianness” troca a terminação de bits de um ou mais bytes de little-endian para big-endian ou vice-versa. Todos os parâmetros são passados por ponteiros, alterando a sequência de armazenagem dos endereços.

Os microcontroladores AVR utilizam a terminação little-endian, porém outros dispositivos podem utilizar a terminação “big-endian”, podendo causar um erro na sequência de bits em operações de comunicação.

A equação [40] demonstra o funcionamento da função “swap_endianness”.

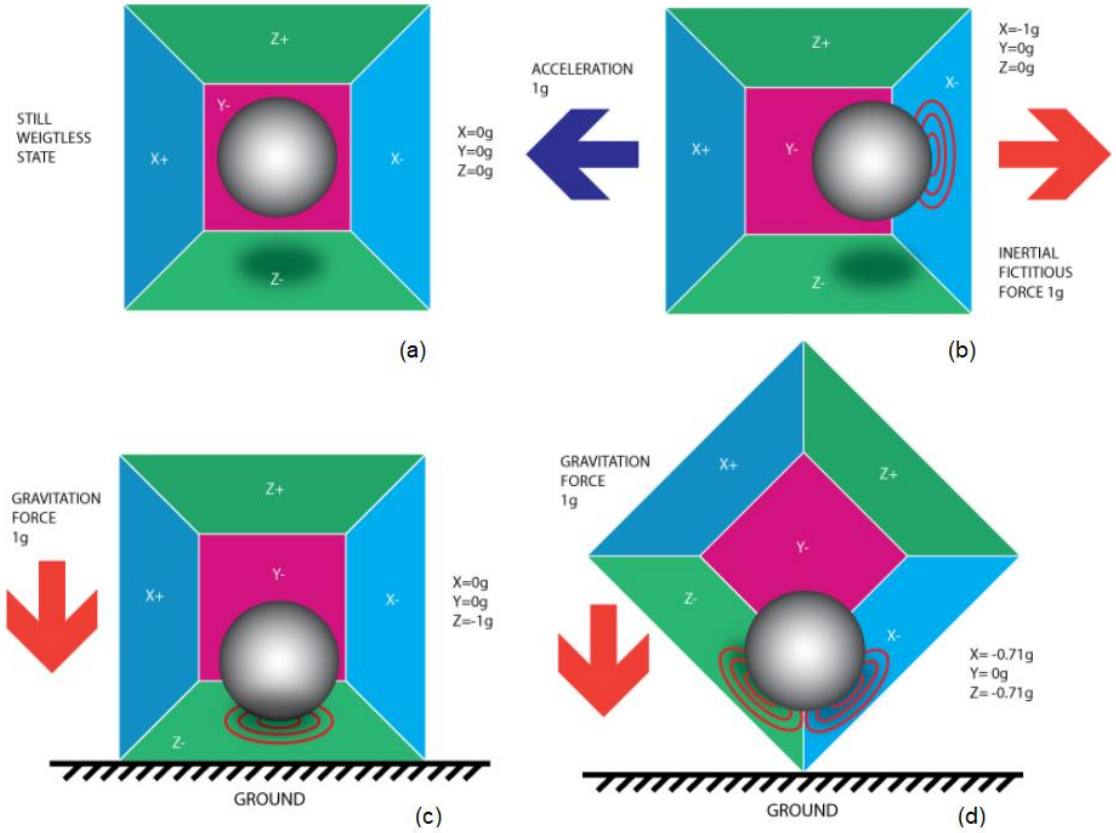
$$0001\ 0010\ 0011\ 0100 \Rightarrow \text{swap}_{\text{endianness}} \Rightarrow\ 0011\ 0100\ 0001\ 0010 \qquad [40]$$

16.3 Acelerômetro

A aceleração é uma grandeza física que mede a variação de velocidade de um corpo em relação ao tempo. Sensores de aceleração eletrônicos são indispensáveis para aplicações práticas em tempo real. (Rocha e Marranghello, 2013). Uma analogia simples para entender a essência do acelerômetro, é imaginar uma esfera de metal no meio de uma caixa, conforme a Figura 48, que apresenta 4 situações de forças atuantes sobre esse sistema.

Através da obtenção das forças em cada eixo, consegue-se obter a angulação dos 3 eixos em relação à uma posição inicial.

Figura 48. Acelerômetro. (a) Condição em ausência de campo gravitacional sobre a caixa. (b) Caixa se movendo para esquerda com aceleração de 1g. (c) Caixa em repouso na Terra. (d) Exemplo de captura em mais de um eixo.

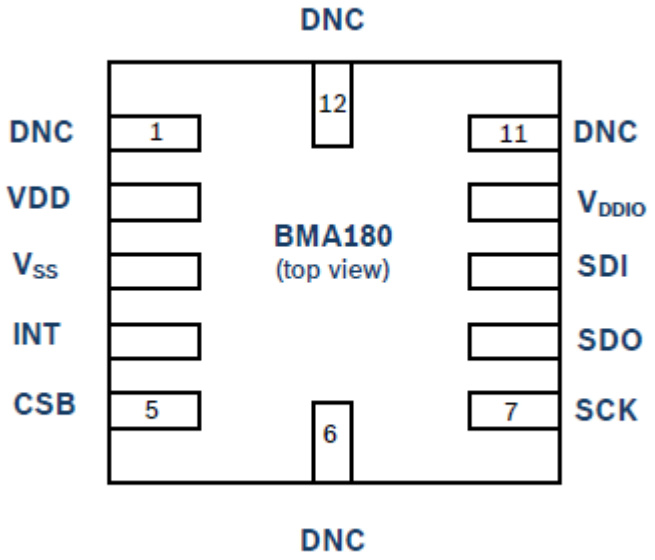


Fonte: Starlino eletronics (2009).

16.3.1 BMA180

O acelerômetro de três eixos utilizado na placa *MultiWii Pro* é o BMA180, que possui alcance de medição de até 16g. A comunicação TWI contida no firmware será discutida nesta seção.

Figura 49 - Esquemático do BMA180. VDD = Supply Voltage; VSS = Ground Connection; INT = Interrupt PIN; CSB = Chip Select; SCK = Serial Clock; SDO = SPI output / Set-up of I2C address; SDI = SPI input / I2C serial data; VDDIO = Supply voltage Connection Digital. DNC = Do not Connect.



Fonte: BMA180 BOSCH Datasheet (2010).

Para habilitar a funcionalidade TWI do BMA180, seu pino CSB, mostrado na Figura 49, deve estar em nível lógico alto. Seu endereço de escravo é então codificado por 7 bits, sendo os 6 primeiros bits fixos e definidos pelo próprio sensor, e o último bit, o menos significativo, determinado pelo valor usado no pino SDO (Set-up of I2C address) como entrada digital. Portando, por padrão, o endereço TWI é 0x40 para SDO conectado no VSS, ou 0x41 para o SDO conectado no VDDIO. (BMA 180 BOSCH Datasheet, 2010). VDDIO é a tensão de referência da comunicação, sendo 3.3V, algo que pode ser visto no esquemático do capítulo 8.

Lembrando que a comunicação TWI usa duas vias, na nomenclatura do BMA: SCK (Serial clock) e SDI (Serial Data Input). O SDI é bidirecional com um dreno

de pull-down aberto, este deve ser externamente conectado ao VDDIO com resistores de pull up. (BMA 180 BOSCH Datasheet, 2010).

Através do protocolo TWI, há possibilidade de transações com o BMA180 de Single byte read, Single byte write, Multiple byte read e Multiple byte write. (BMA 180 BOSCH Datasheet, 2010). As etapas para as transações de bytes múltiplos estão ilustradas nas Figura 50 e Figura 51.

Figura 50 - Muliple write.

Fonte: BMA180 BOSCH Datasheet (2010).

Figura 51 - Muliple read.

Fonte: BMA180 BOSCH Datasheet (2010).

16.3.2 Firmware (BMA180)

Há três funções reservadas para o acelerômetro BMA180, uma para inicialização, outra para aquisição de dados, e a última para calibração e atualização dos dados do sensor. Todas essas funções fazem o uso das funções gerais do protocolo TWI apresentadas na seção precedente.

A função “ACC_init” inicializa e configura o sensor BMA180. As modificações em cada registrador do sensor estão apresentadas na Tabela 15.

Tabela 15 – Modificações da configuração do BMA180.

Nome do Registrador	Endereço	Modificação	Descrição
ctrl_reg0	0x0D	ee_w = 1	Habilita escrita nos registradores de imagem.
bw_tcs	0x020	bw = 0b0001	Filtro Butterworth de segunda ordem com frequência de polo de 20Hz.
tco_z	0x030	mode_config = 0b00	Low noise mode.
offset_lsb1	0x035	range = 0b101	Configura alcance de medida para 8G.

Fonte: Autoria própria.

ACC_init

```
void ACC_init () {
  delay(10);
  i2c_writeReg(BMA180_ADDRESS,0x0D,1<<4);
  delay(5);
  uint8_t control = i2c_readReg(BMA180_ADDRESS, 0x20);
  control = control & 0x0F;
  control = control | (0x01 << 4);
  i2c_writeReg(BMA180_ADDRESS, 0x20, control
  delay(5);
  control = i2c_readReg(BMA180_ADDRESS, 0x30);
  control = control & 0xFC;
  control = control | 0x00;
  i2c_writeReg(BMA180_ADDRESS, 0x30, control);
  delay(5);
  control = i2c_readReg(BMA180_ADDRESS, 0x35);
  control = control & 0xF1;
  control = control | (0x05 << 1);
  i2c_writeReg(BMA180_ADDRESS, 0x35, control);
  delay(5);
  acc_1G = 255;
}
```

A função “ACC_init” faz a leitura dos registradores de controle do sensor através da função “i2c_readReg”. E utilizada de variáveis de controle para modificar apenas os bits necessários, mantendo as configurações imprescindíveis. A escrita é feita através da função “i2c_writeReg” repassando a variável de controle como o novo valor do registador.

Por último, verifica-se a linha com “acc_1G = 255”, que é nada mais o valor para a aceleração da gravidade. Esta afirmação pode ser entendida com mais clareza através da explicação da aquisição de dados crus pela função “ACC_getADC” e da Tabela 17.

ACC_getADC

```
void ACC_getADC () {
  TWBR = ((F_CPU / 400000L) - 16) / 2;
  i2c_getSixRawADC(BMA180_ADDRESS,0x02);

  ACC_ORIENTATION( ((rawADC[1]<<8) | rawADC[0])/16 ,
                   ((rawADC[3]<<8) | rawADC[2])/16 ,
                   ((rawADC[5]<<8) | rawADC[4])/16 );

  ACC_Common(); // Aqui se faz a calibração.
}
```

Inicialmente na função “ACC_getADC”, pode-se ver o aumento da frequência da comunicação TWI (400 KHz). Isso só é possível, pois nas especificações do sensor, há descrita tal limitação.

Os dados fornecidos pelo sensor são lidos através da chamada da função geral “i2c_getSixRawADC” e armazenados no vetor de 6 posições “rawADC”. Os dados recebidos podem ser vistos no mapa de memória global do BMA180, ilustrado na Figura 52.

Figura 52 - Mapa de memória global BMA180 (apenas dados da aceleração dos 3 eixos).

acc_z_msb	07h	acc_z<13:6> (msb)		
acc_z_lsb	06h	acc_z<5:0> (lsb)	0	new_data_z
acc_y_msb	05h	acc_y<13:6> (msb)		
acc_y_lsb	04h	acc_y<5:0> (lsb)	0	new_data_y
acc_x_msb	03h	acc_x<13:6> (msb)		
acc_x_lsb	02h	acc_x<5:0> (lsb)	0	new_data_x

Fonte: BMA180 BOSCH Datasheet (2010).

A Figura 52 ilustra como os dados do acelerômetro estão armazenados entre os registradores de endereço 02h à 07h do BMA180. Cada eixo contém dois

registradores, considerando apenas os dados de aceleração, um contem 6 bits menos significativos e outro contem 8 bits mais significativos, somando-se uma resolução de 14 bits de interesse. Cada posição do vetor “rawADC” recebe os dados de cada registrador, a Tabela 16 ilustra os registradores em cada posição do vetor.

Tabela 16 – Dados do vetor rawADC (acelerômetro)

Posição	Registrador
rawADC[0]	acc_x_lsb
rawADC[1]	acc_x_msb
rawADC[2]	acc_y_lsb
rawADC[3]	acc_y_msb
rawADC[4]	acc_z_lsb
rawADC[5]	acc_z_msb

Fonte: Autoria própria.

Após armazenar os dados do acelerômetro para o vetor “rawADC”, estes são passados de seu estado cru para seu estado tratado. O macro “ACC_ORIENTATION (X,Y,Z)” declarado na biblioteca “def.h” recebe os valores tratados do acelerômetro. O tratamento consiste em retirar os bits que não participam dos dados de aceleração dos eixos (2 bits menos significativos de cada eixo), passando de 16 bits para 14 bits, e ainda, diminuir a resolução dos dados, passando para 12 bits. A Tabela 17 demonstra a resolução e o alcance dos dados tratados.

O macro “ACC_ORIENTATION” contém as seguintes orientações para: “accADC[ROLL] = -X, accADC[PITCH] = -Y, accADC[YAW] = Z”. Portanto, o vetor “accADC” de três posições de 16 bits armazena as leituras tratadas do acelerômetro, ainda em escala da saída do ADC do sensor.

Tabela 17 – Alcance e resolução das medidas (acelerômetro).

Aceleração	Valor em binário	Valor em Decimal
-8g	1000 0000 0000	-2048
-7.99609	1000 0000 0001	-2047
⋮	⋮	⋮
-0.00390g	1111 1111 1111	-1
0.0g	0000 0000 0000	0
+0.00390g	0000 0000 0001	1
⋮	⋮	⋮
+7.99218g	0111 1111 1110	2046
+7.99609g	0111 1111 1111	2047

Fonte: Autoria própria.

A função termina com a chamada da função “ACC_Common”, que possui as etapas de calibração e atualização dos dados do acelerômetro.

```
void ACC_Common() {
  static int32_t a[3];

  if (calibratingA>0) {
    for (uint8_t axis = 0; axis < 3; axis++) {
      if (calibratingA == 400) a[axis]=0;
      a[axis] +=accADC[axis];
      accADC[axis]=0;
      conf.accZero[axis]=0;
    }
    if (calibratingA == 1) {
      conf.accZero[ROLL]  = a[ROLL]/400;
      conf.accZero[PITCH] = a[PITCH]/400;
      conf.accZero[YAW]   = a[YAW]/400-acc_1G;
      conf.angleTrim[ROLL]   = 0;
      conf.angleTrim[PITCH]  = 0;
      writeParams(1);
    }
    calibratingA--;
  }
  accADC[ROLL]  -=  conf.accZero[ROLL] ;
  accADC[PITCH] -=  conf.accZero[PITCH];
  accADC[YAW]   -=  conf.accZero[YAW] ;
```

A calibração tem início em resposta a posição dos sticks, algo a ser discutido no estudo do código principal, sendo breve, quando entra-se no processo de

calibração, a variável “calibratingA” recebe o valor decimal de 400, que é o número de medidas de aceleração de cada eixo necessárias para calcular a média de calibração. Volta-se ao estado normal de operação quando esta variável assumir valor 0.

A função “ACC_Common” inicia-se declarando um vetor de 3 posições “a” que armazenará a soma das 400 medidas de aceleração, já tratadas, em cada eixo.

Se “calibratingA” possui valor maior que 0, ou seja, um processo de calibração entra-se num loop de “for” para referenciar os 3 eixos. Quando “calibratingA” tem seu valor inicial de 400, dentro desse loop a variável “a” referente a um dos 3 eixos, é reiniciada para o início das 400 somas. O processo de soma é repetido 399 vezes, sempre reiniciando as variáveis “accADC” e “conf.accZero” no final do loop. A cada soma, diminui-se uma unidade de “calibratingA” até esta valer 1.

Quando a variável “calibratingA” possui valor 1. Entra-se na condição onde acontece o cálculo das médias, dividindo os valores acumulados em “a” por 400 para os eixos x e y, e dividindo por 400 e subtraindo pela aceleração gravitacional para o eixo z (não perdendo este referencial). O vetor “conf.accZero” armazena a média calculada desses três eixos.

O vetor “conf.accZero”, portanto, armazenará a posição de calibração do acelerômetro ao final desse processo.

Os valores de “conf.angleTrim” para os eixos de pitch e roll também são reiniciados ao final da calibração, e são gravados na EEPROM junto com “conf.accZero”.

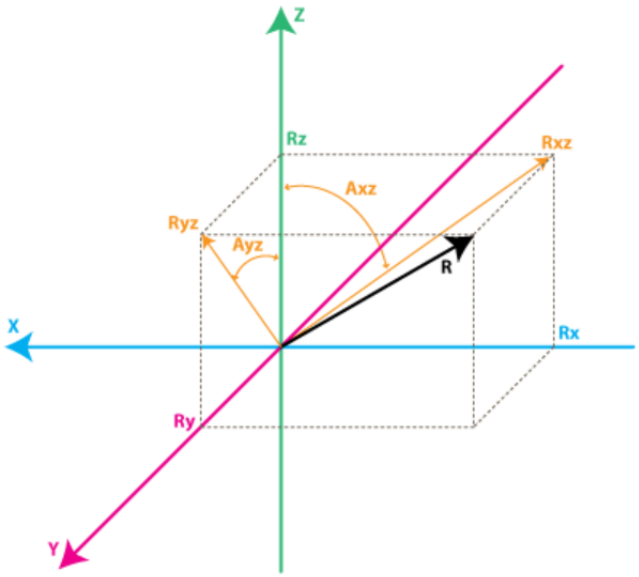
A função “ACC_Common” é chamada em todo loop da programação, e além de realizar a calibração descrita acima, também faz a atualização dos valores de aceleração de acordo com a média de calibração já calculada. A diferença do valor de aceleração atual em relação ao valor de calibração é o valor final que será repassado ás operações do sistema, como pode ser visto nas últimas 3 linhas da função “ACC_Common”.

A variável “conf.angleTrim” tem sua declaração no código principal, onde será feita uma discussão detalhada sobre esta. Porém, resumidamente, esta é um ajuste fino em cima da configuração de calibração do acelerômetro.

16.4 Giroscópio

O giroscópio nos fornece a taxa de variação do ângulo em torno de um eixo, ou seja, a medida de uma rotação.

Figura 53 – Sistema para exemplificação de rotação em torno do eixo X e Y.



Fonte: Starlino eletronics (2009).

A Figura 53 fornece ferramentas para exemplificar a rotação em torno do eixo X e Y em números:

- $\vec{R} = (-\vec{x} + \vec{y} + \vec{z})$;
- R_{xz} = Projeção de R no plano XZ;
- R_{yz} = Projeção de R no plano YZ;
- A_{xz} = Ângulo entre R_{xz} e o eixo Z;
- A_{yz} = Ângulo entre R_{yz} e o eixo Z.

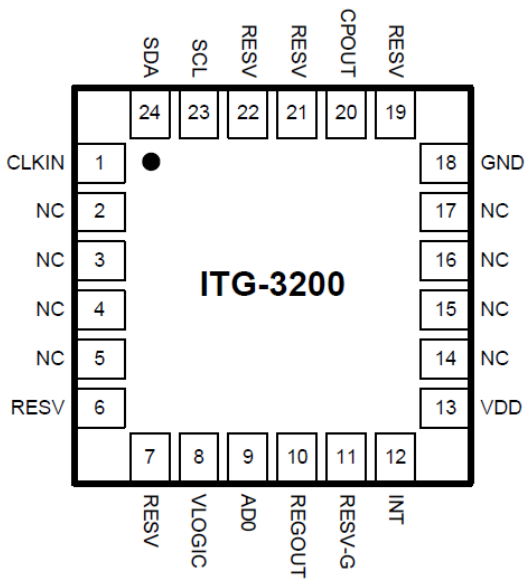
Assume-se que a medida do ângulo A_{xz} no instante $t0$ seja A_{xz0}, e no instante t1 seja A_{xz1}. A taxa de variação desse ângulo será:

$$TaxaA_{xz} = \frac{A_{xz1} - A_{xz0}}{t1 - t0} \quad [41]$$

A $TaxaA_{xz}$ fornece, então, a rotação em torno do eixo Y em $°/seg$, valor este de saída do giroscópio. A obtenção da rotação em torno do eixo X ($TaxaA_{yz}$) é feita de maneira análoga.

16.4.1 ITG32-00

Figura 54 – Esquemático do ITG-3200. CLKIN = Clock external; VLOGIC = Digital IO supply voltage; AD0 = 12C Slave Address LSB; REGOUT = Regulator filter capacitor connection; INT = Interrupt pin; VDD = Power Supply voltage; GND = Power supply ground; RESV-G = Reserved; RESV = Reserved; CPOUT = Charge pump capacitor connection; SCL = I2C serial clock; SDA = I2C serial data; NC = Not connected.



Fonte: ITG-3200 InvenSense Datasheet (2010)

O giroscópio de três eixos utilizado na placa *MultiWii Pro* é o ITG32-00, Figura 54. Este sensor possui alcance de escala de até $2000°/seg$. Esta seção focará na comunicação TWI desde sensor contida no firmware em estudo.

O ITG-3200 sempre opera como um escravo quando em comunicação com um sistema operacional. As linhas SDA (I2C Serial Data) e SCL (I2C Serial Clock), conforme a nomenclatura do dispositivo, precisam de resistores de pull-up conectados à alimentação VDD. (ITG-3200 InvenSense Datasheet, 2010).

O endereçamento do ITG-3200 funciona de forma similar ao BMA180. Seu endereço é codificado por 7bits, 0b110100x, sendo o bit menos significativo

selecionado pelo pino 9, AD0 (I2C Slave Address LSB), portanto, o dispositivo pode assumir o endereço 0x68 (AD0 em nível lógico baixo) ou 0x69 (AD0 em nível lógico alto). (ITG-3200 InvenSense Datasheet, 2010).

O pino VLOGIC do ITG32-00 assume a mesma função do pino VDDIO do acelerômetro, fazendo um referencial de 3.3V para a comunicação.

Através do protocolo TWI, há possibilidade de transações com o ITG-3200 de Single byte read, Single byte write, Multiple byte read e Multiple byte write. (ITG-3200 InvenSense Datasheet, 2010). As etapas para as transações de bytes múltiplos estão ilustradas nas Figura 55 e Figura 56.

Figura 55 - Muliple write.

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

Fonte: ITG-3200 InvenSense Datasheet (2010).

Figura 56- Muliple read.

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

Fonte: ITG-3200 InvenSense Datasheet (2010).

16.4.2 Firmware (ITG32-00)

Há três funções reservadas para o giroscópio ITG32-00, uma para inicialização, outra para aquisição de dados, e a última para calibração e atualização dos dados do sensor. Assim como no acelerômetro, todas essas funções fazem o uso das funções gerais do protocolo TWI.

A função “Gyro_init” inicializa e configura o sensor ITG32-00. As modificações em cada registrador do sensor estão apresentadas na Tabela 18.

Tabela 18 – Modificações da configuração do ITG32-00.

Nome do Registrador	Endereço	Modificação	Descrição
PWR_MGM	0x3E	H_RESET = 1 CLK_SEL = 0b11 H_RESET = 0	Reinicializa o dispositivo e registradores internos para as configurações padrão. Fonte de clock do dispositivo = PLL com referência no giroscópio Z.
SMPLRT_DIV	0x15	SMPLRT_DIV = 0	Divisor da taxa de amostragem = 1.
DLPF_FS	0x16	FS_SEL = 0b11 DLPF_CFG = 0b00	Máximo alcance de escala do sensor de gyro = $\pm 2000°/seg$. Configuração da largura de banda do filtro passa baixa digital = 256Hz. Taxa de amostragem interna = 8KHz

Fonte: Autoria própria.

Gyro_init

```
void Gyro_init() {
  delay(100);
  i2c_writeReg(ITG3200_ADDRESS, 0x3E, 0x80);
//  delay(5);
//  i2c_writeReg(ITG3200_ADDRESS, 0x15, 0);
  delay(5);
  i2c_writeReg(ITG3200_ADDRESS, 0x16, 0x18);
  delay(5);
  i2c_writeReg(ITG3200_ADDRESS, 0x3E, 0x03);
  delay(100);
}
```

A função “Gyro_init” faz a escrita nos registradores de controle do sensor através da função “i2c_writeReg” com a passagem do endereço de escravo do

giroscópio, endereço do registrador de controle e o valor a ser escrito no registrador de controle.

Pode-se ver o aumento da frequência da comunicação TWI (400 KHz) no início da função “Gyro_getADC”. Esta configuração é possível por atender as especificações do dispositivo giroscópio. A função “Gyro_getADC” é a responsável pela aquisição dos dados crus fornecidos pelo sensor giroscópio através da chamada da função geral “i2c_getSixRawADC”, que dentro desta, armazena estes num vetor de 6 posições “rawADC”. Processo totalmente semelhante ao que ocorre no BMA180.

A disposição dos dados recebidos pode ser vista no mapa de memória global do ITG32-00, ilustrado na Figura 57.

Gyro_getADC

```
void Gyro_getADC () {
  TWBR = ((F_CPU / 400000L) - 16) / 2;
  i2c_getSixRawADC(ITG3200_ADDRESS,0X1D);
  GYRO_ORIENTATION( ((rawADC[0]<<8) | rawADC[1])/4 ,
                    ((rawADC[2]<<8) | rawADC[3])/4 ,
                    ((rawADC[4]<<8) | rawADC[5])/4 );
  GYRO_Common();
}
```

Figura 57 - Mapa de memória global ITG32-00 (apenas dados da taxa de variação angular em torno dos 3 eixos).

1D	29	GYRO_XOUT_H	R	GYRO_XOUT_H
1E	30	GYRO_XOUT_L	R	GYRO_XOUT_L
1F	31	GYRO_YOUT_H	R	GYRO_YOUT_H
20	32	GYRO_YOUT_L	R	GYRO_YOUT_L
21	33	GYRO_ZOUT_H	R	GYRO_ZOUT_H
22	34	GYRO_ZOUT_L	R	GYRO_ZOUT_L

Fonte: ITG-3200 InvenSense Datasheet (2010).

Da Figura 57, observa-se que cada eixo contém dois registradores contendo seus dados, um com os 8 bits MSB e outro com 8 bits LSB, sendo 2 destes bits complementares. Cada posição do vetor “rawADC” recebe um dado de cada registrador, conforme visto na Tabela 19.

Tabela 19 – Dados do vetor rawADC (giroscópio).

Posição	Registrador
rawADC[0]	GYRO_XOUT_H
rawADC[1]	GYRO_XOUT_L
rawADC[2]	GYRO_YOUT_H
rawADC[3]	GYRO_YOUT_L
rawADC[4]	GYRO_ZOUT_H
rawADC[5]	GYRO_ZOUT_L

Fonte: Autoria própria.

Após armazenar os dados do giroscópio para o vetor “rawADC”, estes são passados de seu estado cru para seu estado tratado. O macro “GYRO_ORIENTATION (X,Y,Z)” declarado na biblioteca “def.h” recebe os valores tratados do giroscópio. O tratamento consiste em retirar os bits complementares dos dados da taxa de variação angular, passando de 16 bits para 14 bits. Essa modificação gera uma alteração na sensibilidade do sensor, como demonstrado a seguir:

$$Sensitivity_{16\ bitis} = 14.375\,\frac{LSBs}{°/seg} \quad [42]$$

$$Sensitivity_{14\ bitis} = \frac{Sensitivity_{16\ bitis}}{4} = 3.59\,\frac{LSBs}{°/seg} \quad [43]$$

Portando a conversão do valor da saída do conversor A/D do IT32-00 para $°/seg$ é:

$$ADC_{value} \times \frac{1}{Sensitivity_{14\ bitis}} \quad [44]$$

O macro “GYRO _ORIENTATION” contém as seguintes orientações: “gyroADC[ROLL] = Y, gyroADC[PITCH] = -X, gyroADC[YAW] = -Z”. Portanto, o vetor “gyroADC” de três posições de 16 bits armazena as leituras tratadas do giroscópio, ainda em escala da saída do ADC do sensor.

Ao finalizar a função “Gyro_getADC” há a chamada da função “GYRO_Common”, responsável pela parte de calibração do giroscópio.

GYRO_Common

```
void GYRO_Common() {
  static int16_t previousGyroADC[3] = {0,0,0};
  static int32_t g[3];
  uint8_t axis;
  if (calibratingG>0) {
    for (axis = 0; axis < 3; axis++) {
      if (calibratingG == 400) g[axis]=0;
      g[axis] +=gyroADC[axis];
      gyroADC[axis]=0;
      gyroZero[axis]=0;
      if (calibratingG == 1) {
        gyroZero[axis]=g[axis]/400;
        blinkLED(10,15,1);
      }
    }
    calibratingG--;
  }
  for (axis = 0; axis < 3; axis++) {
    gyroADC[axis]  -= gyroZero[axis];
    gyroADC[axis] = constrain(gyroADC[axis],previousGyroADC[axis]-
    800,previousGyroADC[axis]+800);
    previousGyroADC[axis] = gyroADC[axis];
  }
}
```

A calibração do giroscópio, além de ter início em resposta a posição dos sticks, como no acelerômetro, também se inicia ao energizar o quadrirrotor. O algoritmo das condições para que a calibração ocorra é algo a ser estudado no código principal, porém, assim como na explicação do código de calibração, em síntese, quando inicia-se o processo de calibração, a variável “calibratingG” recebe o valor decimal de 400, que é o número de medidas de taxa de variação angular em relação a cada eixo necessárias para calcular a média de calibração. A cada ciclo diminui-se uma unidade deste valor, voltando a operação normal quando esta assumir valor 0.

A função “GYRO_Common” inicia-se declarando: um vetor de 3 posições “g” que armazenará a soma das 400 medidas da taxa de variação angular já tratadas em relação a cada eixo; um vetor de 3 posições “previousGyroADC” referenciando a iteração anterior com o valor de “gyroADC”; variável “axis” para referenciar os 3 eixos.

Tendo um processo bem semelhante ao acelerômetro, se “calibratingG” possui valor maior que 0, ou seja, estado de calibração, entra-se num loop de for para referenciar os 3 eixos. Quando “calibratingG” tem seu valor inicial de 400, dentro desse

loop a variável “g” referente a um dos 3 eixos, é reiniciada para o início das 400 somas. O processo de soma é repetido 399 vezes, sempre reiniciando as variáveis “gyroADC” e “gyroZero” no final do loop. A cada soma, diminui-se uma unidade de “calibratingG” até esta valer 1.

Quando a variável “calibratingG” possui valor 1, calcula-se a média para cada eixo do vetor “g” fazendo a divisão por 400. A variável que recebe a média dos três eixos é o vetor de 3 posições “gyroZero” Ao final do processo de calibração há a indicação pelos leds.

Por fim, calcula-se a diferença do valor da taxa de variação angular atual em relação ao valor desta calibrada, repassando-a para futuros tratamentos no sistema.

Ainda, com o objetivo de amenizar erros na medida do giroscópio, limita-se a variação da taxa de variação angular entre duas leituras consecutivas. Isso é feito nas duas últimas linhas da função “GYRO_Common”, onde através de uma função “constrain” limita-se o valor lido e tratado de “gyroADC” em $\pm$800 unidades da saída do converSor A/D. Assim, de acordo com a equação [44].

$$800 \times \frac{1}{Sensitivity_{14\ bitis}} = 222.60\ °/seg. \qquad [45]$$

Portanto, entre duas medidas consecutivas do giroscópio, há a limitação de variação de $\pm\ 222.60\ °/seg$.

16.5 Barômetro

O uso do barômetro é feito com o intuito de estimar a altitude através da mensura da pressão atmosférica. Apesar de ser um dispositivo eletrônico, seu conceito de funcionamento é compatível com os barômetros mais simples (SILVA, 2013).

O barômetro eletrônico possui duas câmaras de pressão, sendo uma selada com a pressão de referência, e a outra aberta para a pressão do ambiente externo. Entre essas duas câmaras, fazendo a separação destas, há um conjunto de “strain gauges” resistivos dispostos numa membrana, o piezo resistivo. (SILVA, 2013).

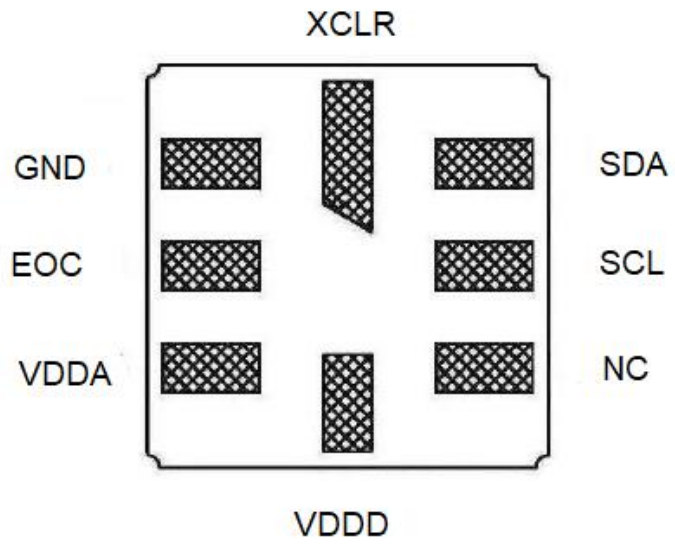
A diferenciação entre a pressão de referência e a pressão externa, gera uma deformidade na membrana, variando a resistência do piezo resistivo, que quantifica a deformidade. (SILVA, 2013).

16.5.1 BMP085

O BMP085, Figura 58, é um sensor de pressão digital de alta precisão para aplicações comerciais, tendo alcance de 9000m acima do nível do mar e -500m abaixo. (BMP085 BOSCH Datasheet, 2009). E este é o barômetro presente na placa controladora MultiWii Pro. Este sensor foi desenvolvido para ser conectado diretamente ao microcontrolador pela linha de comunicação I2C. (BMP085 BOSCH Datasheet, 2009).

Além do cálculo da pressão, o BMP085 também fornece a informação sobre a temperatura do ambiente, tendo alcance de -40˚C até 85˚C. (BMP085 BOSCH Datasheet, 2009).

Figura 58 – Esquemático do BMP085. GND = Ground; EOC = End of conversion; VDDA = Power supply; VDDD = Digital power supply; NC = no internal connection; SCL = 12C serial bus clock input; SDA = 12C serial bus data; XCLR = master clear (low active) input.



Fonte: Adaptado de BMP085 BOSCH Datasheet (2009).

O BMP085 entrega valores descompensados de temperatura e pressão. Estes dados devem ser compensados pelos dados de calibração presentes na memória EEPROM do BMP085. (BMP085 BOSCH Datasheet, 2009). Os 176 bits da EEPROM são particionados em 11 palavras de 16 bits cada, conforme visto na Tabela 20.

Tabela 20 – Dados de calibração do BMP085.

	Endereço do registrador do BMP085	
Parâmetro	**MSB**	**LSB**
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xBO	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

Fonte: BMP085 BOSCH Datasheet (2009).

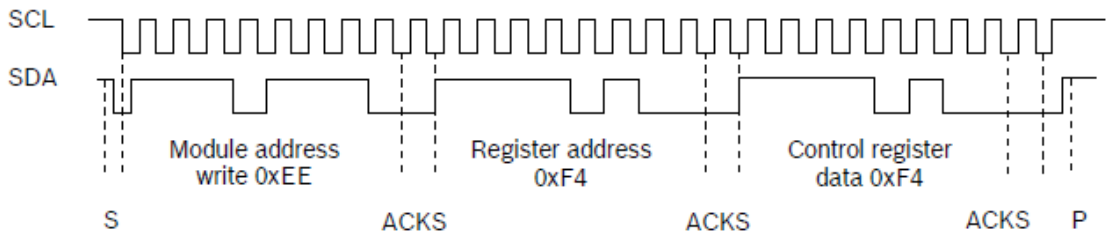
Sendo operado apenas por comunicação TWI, o BMP085 não necessita de nenhuma configuração especial. Apenas, como padrão, o SCL (serial clock) e o SDA (serial data) precisam de resistores de pull-up conectados ao VDD. (BMP085 BOSCH Datasheet, 2009).

A comunicação TWI é usada para controlar o sensor, ler os dados de calibração da EEPROM e ler os dados de medidas quando a conversão A/D estiver finalizada. (BMP085 BOSCH Datasheet, 2009).

O endereçamento do dispositivo para a comunicação TWI é composta por 7 bits fixos (0x77). Portanto, para conectar mais de um dispositivo é necessário o uso do pino XCLR. (BMP085 BOSCH Datasheet, 2009).

A transação do BMP085 é, de certa forma, mais complexa ao comparar com a do acelerômetro e giroscópio. Isso porque é necessário enviar um comando de início para começar as medições. O processo para enviar esse comando está ilustrado na Figura 59. Os tipos de comandos que podem ser requisitados para o registrador de comando (0xF4) do BMP085 estão dispostos na Tabela 21. O parâmetro “OSRS” faz referência ao modo de operação selecionado, que ao variar este, muda-se o consumo de potência, velocidade e resolução. Os modos existentes são: Ultra low power; Standard; High resolution; Ultra high resolution. (BMP085 BOSCH Datasheet, 2009).

Figura 59 – Comando de início de medição do BMP085.



Fonte: BMP085 BOSCH Datasheet (2009).

Tabela 21 – Valores para o registrador de controle (0xF4) do BMP085.

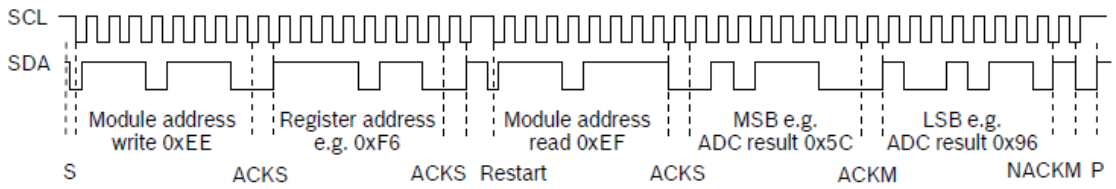
Medida	Valor de comando para o Registrador	Tempo máximo de conversão [ms]
Temperatura	0x2E	4.5
Pressão (OSRS = 0)	0x34	4.5
Pressão (OSRS = 1)	0x74	7.5
Pressão (OSRS = 2)	0xB4	13.5
Pressão (OSRS = 3)	0xF4	22.5

Fonte: BMP085 BOSCH Datasheet (2009).

Após o processo de comando, espera-se o tempo de conversão, e se faz a leitura dos dados da conversão A/D. Os dados descompensados de temperatura possuem 16 bits, e os dados descompensados de pressão podem variar de 16 até 19 bits, estes dados são lidos ao acessar os registradores de endereços 0xF6 (MSB),

0xF7 (LSB), e opcionalmente, e exclusivo para os dados de pressão, 0xF8 (XLSB). (BMP085 BOSCH Datasheet, 2009). O processo a ser seguido para a leitura de 16 bits está ilustrado na Figura 60.

Figura 60 – Leitura dos dados descompensados do BMP085.



Fonte: BMP085 BOSCH Datasheet (2009).

16.5.2 Firmware (BMP085)

Antes de estudar cada função exclusiva para a comunicação TWI do BMP085 existente no firmware, é essencial discutir sobre a criação da estrutura “bmp085_ctx”.

bmp085_ctx

```
static struct {
  int16_t  ac1, ac2, ac3;
  uint16_t ac4, ac5, ac6;
  int16_t  b1, b2, mb, mc, md;
  union {uint16_t val; uint8_t raw[2]; } ut;
  union {uint32_t val; uint8_t raw[4]; } up;
  uint8_t  state;
  uint32_t deadline;
} bmp085_ctx;
```

A estrutura “bmp085_ctx” tem a missão de armazenar: os dados de calibração armazenados na memória EEPROM do BMP085, Tabela 20; os dados descompensados de temperatura e pressão; o estado do processo dentro da função de atualização dos dados lidos; tempo decorrido na função de atualização dos dados lidos.

A principal vantagem, e objetivo, da criação dessa estrutura é manter os endereços de memória de cada variável pertencente a esta em sequência, de acordo com a ordem de declaração. Isso facilita o uso das funções gerais do protocolo TWI,

tanto para a escrita em um registrador, quanto na leitura e armazenamento dos valores deste.

Vale a pena destacar as variáveis “ut” e “up”, responsáveis por armazenar os dados descompensados de temperatura e pressão, respectivamente. Estas estão declaradas por uma “union”, que gerencia o reaproveitamento do mesmo endereço de memória entre uma variável inteira “val” e um vetor “raw”. O vetor “raw” é utilizado no processo de leitura dos dados descompensados no registrador do BMP085, armazenando 8bits em cada posição, já a variável “val” é a junção destes bits armazenados no vetor. Pode-se observar a necessidade de declarações compatíveis ao número de bits de cada dado descompensado: 16 bits para a temperatura e 19 bits para a pressão.

Há 8 funções reservadas para o barômetro BMP085, algumas destas, assim como no acelerômetro e giroscópio, fazem o uso das funções gerais do protocolo TWI. A Tabela 22 demonstra o nome das funções exclusivas do barômetro junto com suas descrições.

Tabela 22 – Funções exclusivas do BMP085.

Nome da Função	Descrição
BMP085_readCalibration	Faz a leitura dos dados de calibração presente na memória EEPROM do BMP085.
I2c_BMP085_UT_Start	Comando de início de medição da temperatura.
BMP085_UP_Start	Comando de início de medição da pressão.
I2c_BMP085_UT_Read	Leitura dos dados descompensados da temperatura.
I2c_BMP085_UP_Read	Leitura dos dados descompensados da pressão.
Baro_init	Função de inicialização do sensor, fazendo um comando de início e a leitura dos dados descompensados referentes à temperatura.
I2c_BMP085_Calculate	Cálculo para temperatura em ˚C e altitude em m.
Baro_update	Atualização dos dados.

Fonte : Autoria própria.

BMP085_readCalibration

```
void i2c_BMP085_readCalibration(){
  delay(10);
  size_t s_bytes = (uint8_t*)&bmp085_ctx.md -
(uint8_t*)&bmp085_ctx.ac1 + sizeof(bmp085_ctx.ac1);
  i2c_read_reg_to_buf(BMP085_ADDRESS, 0xAA, &bmp085_ctx.ac1,
s_bytes);
  int16_t *p;
  for (p = &bmp085_ctx.ac1; p <= &bmp085_ctx.md; p++) {
    swap_endianness(p, sizeof(*p));
  }
}
```

"i2c_BMP085_readCalibration" faz a leitura dos dados de calibração presente na memória EEPROM do BMP085.

A variável "s_bytes" recebe o cálculo do número de bytes necessários para a leitura dos coeficientes de calibração, sendo 22, conforme verificado também na Tabela 20.

Através da função geral da comunicação TWI "i2c_read_reg_to_buf" armazena-se nas variáveis (ac1, ac2, ac3, ac4, ac5, ac6, b1, b2, mb, mc e md) da estrutura "bmp085_ctx" cada coeficiente de calibração, sendo 16 bits para cada um, com a transmissão do mais significativo primeiro. Portando, como o BMP085 utiliza-se de uma organização "big endianness" e o microcontrolador AVR ATMEGA2560 "little endianness" é necessário inverter os bytes armazenados com a chamada da função geral "swap_endianess".

I2c_BMP085_UT_Start

```
void i2c_BMP085_UT_Start() {
  i2c_writeReg(BMP085_ADDRESS,0xf4,0x2e);
  i2c_rep_start(BMP085_ADDRESS<<1);
  i2c_write(0xF6);
  i2c_stop();
}
```

"i2c_BMP085_UT_Start" gera a condição de início da medição da temperatura.

Através da função geral "i2c_writeReg" gera-se uma condição de início do protocolo TWI, enviando em seguida o endereço do barômetro com o oitavo bit em modo de escrita. Em sequência, escreve-se o endereço do registrador de comando (0xF4) para ter acesso à escrita deste. O registador (0xF4) é responsável pela requisição do início de medição, os valores que este pode receber estão dispostos na

Tabela 21. Escrevendo (0x2E) neste registrador, inicia-se a medição de temperatura. Todo o processo descrito nesse parágrafo pode ser acompanhado na Figura 59.

A função geral “i2c_rep_start” dá início ao processo de leitura, iniciando o processo da Figura 60. Esta gera a condição de início, seguida do endereço do barômetro em modo de escrita. Essa função, “I2c_BMP085_UT_Start”, termina com a geração de uma condição de parada com a função geral “i2c_stop” logo após a escrita do endereço do registrador de controle (0xF6) que dará acesso aos dados descompensados.

Observe, que, assim como toda a explicação do texto até aqui, fica-se implícito a transmissão e recepção de sinais de ACK e NACK.

BMP085_UP_Start

```
void i2c_BMP085_UP_Start () {
  i2c_writeReg(BMP085_ADDRESS,0xf4,0x34+(OSS<<6));
  i2c_rep_start(BMP085_ADDRESS<<1);
  i2c_write(0xF6);
  i2c_stop();
}
```

“i2c_BMP085_UT_Start” gera a condição de início da medição da pressão.

Ocorre o mesmo processo da função “I2c_BMP085_UT_Start”, com a única diferença de que o registrador de controle (0xF4) recebe (0xB4), dando início a medição de temperatura no modo High resolution (OSS = 2).

I2c_BMP085_UT_Read

```
void i2c_BMP085_UT_Read() {
  i2c_rep_start((BMP085_ADDRESS<<1) | 1);
  bmp085_ctx.ut.raw[1] = i2c_readAck();
  bmp085_ctx.ut.raw[0] = i2c_readNak();
}
```

Continua-se o processo da Figura 60, através de “i2c_BMP085_UT_Read”, fazendo a leitura dos dados descompensados de temperatura.

Através da função geral “i2c_rep_start” é gerada uma condição de início seguida do endereço do barômetro em modo de leitura. Leia-se então os dados em sequência do registrador (0xF6) (MSB) e (0xF7) (LSB), totalizando 16 bits, armazenados nas duas posições do vetor “ut.raw” da estrutura “bmp085_ctx”

I2c_BMP085_UP_Read

```
void i2c_BMP085_UP_Read () {
  i2c_rep_start((BMP085_ADDRESS<<1) | 1);
  bmp085_ctx.up.raw[2] = i2c_readAck();
  bmp085_ctx.up.raw[1] = i2c_readAck();
  bmp085_ctx.up.raw[0] = i2c_readNak();
}
```

Continua-se o processo da Figura 60, através de “i2c_BMP085_UP_Read”, fazendo a leitura dos dados descompensados de pressão.

Através da função geral “i2c_rep_start” é gerada uma condição de início seguida do endereço do barômetro em modo de leitura. Leia-se então os dados em sequência do registrador (0xF6) (MSB), (0xF7) (LSB) e (0xF8) (XLSB), totalizando 24 bits, sendo 19 usuais, armazenados nas três posições do vetor “up.raw” da estrutura “bmp085_ctx”

Baro_init

```
void  Baro_init() {
  delay(10);
  i2c_BMP085_readCalibration();
  i2c_BMP085_UT_Start();
  delay(5);
  i2c_BMP085_UT_Read();
}
```

“Baro_init” é a função de inicialização do sensor, fazendo um comando de início e a leitura dos dados descompensados referentes à temperatura.

I2c_BMP085_Calculate

```
void i2c_BMP085_Calculate() {
  int32_t  x1, x2, x3, b3, b5, b6, p, tmp;
  uint32_t b4, b7;

  // Cálculo de temperatura
  x1 = ((int32_t)bmp085_ctx.ut.val - bmp085_ctx.ac6) * bmp085_ctx.a
c5 >> 15;
  x2 = ((int32_t)bmp085_ctx.mc << 11) / (x1 + bmp085_ctx.md);
  b5 = x1 + x2;
  // temperatura = (b5 + 8) >> 4

  // Cálculo de pressão
  b6 = b5 - 4000;
  x1 = (bmp085_ctx.b2 * (b6 * b6 >> 12)) >> 11;
  x2 = bmp085_ctx.ac2 * b6 >> 11;
  x3 = x1 + x2;
  tmp = bmp085_ctx.ac1;
  tmp = (tmp*4 + x3) << OSS;
  b3 = (tmp+2)/4;
  x1 = bmp085_ctx.ac3 * b6 >> 13;
  x2 = (bmp085_ctx.b1 * (b6 * b6 >> 12)) >> 16;
  x3 = ((x1 + x2) + 2) >> 2;
  b4 = (bmp085_ctx.ac4 * (uint32_t)(x3 + 32768)) >> 15;
  b7 = ((uint32_t) (bmp085_ctx.up.val >> (8-
OSS)) - b3) * (50000 >> OSS);
  p = b7 < 0x80000000 ? (b7 * 2) / b4 : (b7 / b4) * 2;
  x1 = (p >> 8) * (p >> 8);
  x1 = (x1 * 3038) >> 16;
  x2 = (-7357 * p) >> 16;
  pressure = p + ((x1 + x2 + 3791) >> 4);
}
```

“i2c_BMP085_Calculate” é a função responsável por retornar os valores reais de temperatura (˚C) e pressão (Pa). O algoritmo é indicação do Datasheet do BMP085.

Pode-se ver que a temperatura é calculada, porém seu valor não possui utilização no firmware, deixando-a apenas engatilhada para uma aplicação futura. Já a pressão é calculada e armazenada na variável global de 32 bits “pressure”.

Baro_update

```
void Baro_update() {
  if (currentTime < bmp085_ctx.deadline) return;
  bmp085_ctx.deadline = currentTime;
  TWBR = ((F_CPU / 400000L) - 16) / 2;
  switch (bmp085_ctx.state) {
    case 0:
      i2c_BMP085_UT_Start();
      bmp085_ctx.state++; bmp085_ctx.deadline += 4600;
      break;
    case 1:
      i2c_BMP085_UT_Read();
      bmp085_ctx.state++;
      break;
    case 2:
      i2c_BMP085_UP_Start();
      bmp085_ctx.state++; bmp085_ctx.deadline += 14000;
      break;
    case 3:
      i2c_BMP085_UP_Read();
      i2c_BMP085_Calculate();
      BaroAlt = (1.0f - pow(pressure/101325.0f, 0.190295f)) * 44330
00.0f; //centimeter
      bmp085_ctx.state = 0; bmp085_ctx.deadline += 5000;
      break;
  }
```

“baro_update” faz toda a logística das funções exclusivas do barômetro vistas até aqui.

A primor, limita-se a frequência de operação da função para 42.37Hz. A variável responsável por armazenar o tempo de operação é a “deadline”, que se encontra também na estrutura “bmp085_ctx”

Em sequência há atualização da frequência da comunicação TWI, assim como houve no acelerômetro e giroscópio, essa é atualizada para 400KHz, respeitando a limitação do dispositivo.

Há então o início da operação de logística, que é atualizada pela variável “state” da estrutura “bmp085_ctx” como parâmetro de um “switch”. A Figura 61 faz a explicação da sequência.

Figura 61 – Logística das funções exclusivas do barômetro

Fonte: Autoria própria.

Onde para o cálculo da altitude em metros, utiliza-se:

$$altitude = 44330 \times \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5{,}255}}\right) \quad [46]$$

Sendo p_0 = 1013.25hPa, referente à pressão ao nível do mar.

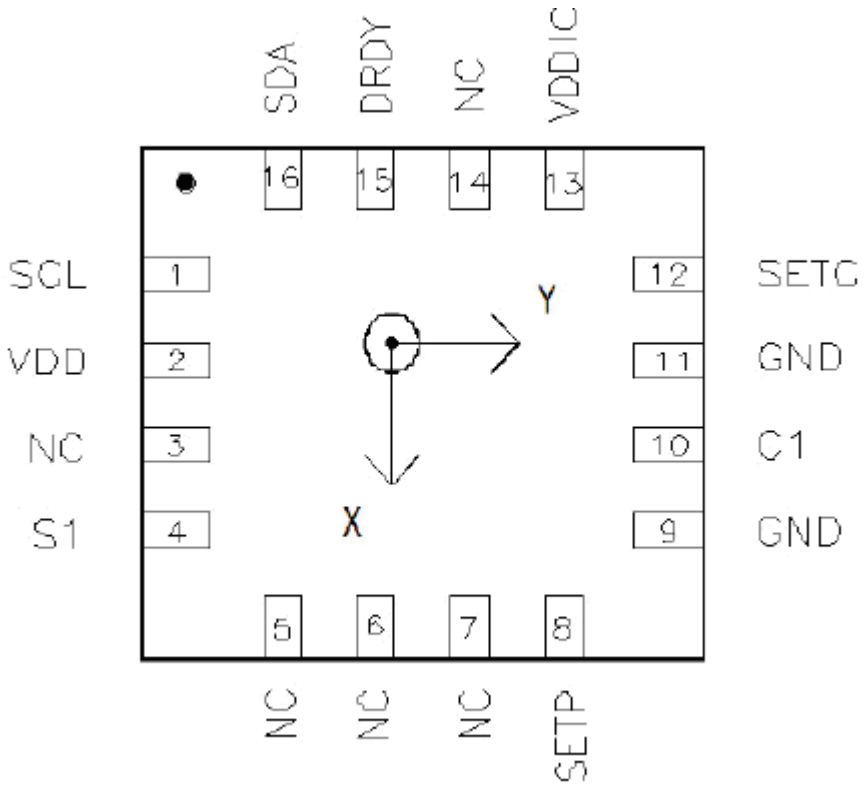
A altitude em centímetros é armazenada na variável global “BaroAlt”.

16.6 Magnetômetro

O magnetômetro, a partir de transdutores de efeito Hall, é capaz de mensurar o campo magnético em sua volta, ou seja, quando ausente de influenciadores de alta intensidade de campo magnético, como imãs, fornece como retorno, o campo magnético gerado pelos polos do Planeta Terra. (SILVA, 2013).

16.6.1 HMC5883L

Figura 62 - Esquemático básico do HMC5883L



Fonte: HMC5883L Honeywell Datasheet (2010).

O magnetômetro utilizado na placa *MultiWii Pro* é o HMC5883L, Figura 62. O HMC5883L, possui 3 sensores magnético-resistivos (magnetoresistive) para cada eixo. Sua funcionalidade é a medição de campos magnéticos, com a possibilidade de alcance da medição de densidade de fluxo magnético de até ± 8 Gauss com resolução de 5mGa em um conversor analógico/digital de 12bits. (HMC5883L Honeywell Datasheet, 2010).

Sobre seus sensores magnéticos-resistivos, mais precisamente, segundo o datasheet do dispositivo e com tradução de autoria própria, informa-se que:

> Os sensores magnético-resistivos são feitos de um filme fino de níquel-ferro e modelados como um elemento de tira resistiva. Na presença de um campo magnético, uma mudança nos elementos da ponte resistiva causa uma mudança correspondente na tensão entre as saídas da ponte.
>
> Estes elementos resistivos são alinhados juntos para ter um eixo sensível comum, que irá prover mudança positiva de tensão no incremento de campo magnético na direção sensível. Devido a saída ser apenas proporcional para a componente de campo magnético ao longo de seu eixo, foram colocados sensores de ponte adicionais em direções ortogonais para permitir medidas precisas de campo magnético em qualquer orientação.

A diferença mais destacável, em relação à configuração do HMC5883L, é a função de auto teste, que checa o dispositivo para a operação apropriada. Basicamente, nesta função, o dispositivo é excitado internamente, com um campo magnético nominal, podendo escolher entre uma polarização positiva ou negativa através da configuração de seus registradores. Este campo gerado é, portanto, mensurado e reportado para o dispositivo mestre. (HMC5883L Honeywell Datasheet, 2010).

Destacando a citação de um mestre no parágrafo passado, é importante de imediato dizer que a configuração e a troca de dados do HMC5883L são exclusivamente feitas pelo protocolo 12C, onde conecta-se como um dispositivo escravo. O endereço de identificação de 7 bits do dispositivo é 0x1E (portando, sendo 0x3C para operações de escrita e 0x3D para operações de leitura), suportando a velocidade de até 400KHz na comunicação. É interessante ainda destacar, que as atividades requeridas pelo mestre têm prioridade sobre as atividades internas, como as medidas. Fazendo da comunicação algo mais eficaz. (HMC5883L Honeywell Datasheet, 2010).

Através do protocolo I2C, há possibilidade de transações Single byte read, Single byte write, Multiple byte read e Multiple byte write idênticas ao BMA180 e, portanto, também idênticas ao ITG32-00. As Figuras 50, 51, 55 e 56 ilustram tais transações.

O HMC5883L possui modos selecionáveis, permitindo gestão de potência, tais como:

- Modo de medida contínua:
  Faz medições contínuas na taxa selecionada pelo usuário, colocando os dados obtidos nos registradores de saída de dados. Estes registradores são logo atualizados logo após a medição seguinte terminar. (HMC5883L Honeywell Datasheet, 2010).
- Modo de medida única:
  Modo automático, logo após energizar o dispositivo. Durante este modo, o dispositivo faz apenas uma única medida e coloca em seus registradores de saída. Logo após os registradores receberem a medida, o dispositivo entre em modo de espera.
  Este é o modo selecionado para a realização do auto teste. (HMC5883L Honeywell Datasheet, 2010).
- Modo de espera:
  Durante este modo, o dispositivo ainda é acessado pela comunicação I2C, porém fontes principais de consumo são desabilitadas, como os conversores analógico/digital, amplificadores e a corrente de polarização interna. Os registradores de saída de dados são inalterados quando este modo é ativado. (HMC5883L Honeywell Datasheet, 2010).

Os registradores para configuração e acesso de dados estão destacados na Tabela 23, junto com seus endereços correspondentes.

Tabela 23 - Registradores do HMC5883L

$	Nome	Acesso
00	Registrador de Configuração A	Leitura/Escrita
01	Registrador de Configuração B	Leitura/Escrita
02	Registrador de Modo	Leitura/Escrita
03	Registrador de Saída de dados (X MSB)	Leitura
04	Registrador de Saída de dados (X LSB)	Leitura
05	Registrador de Saída de dados (Z MSB)	Leitura
06	Registrador de Saída de dados (Z LSB)	Leitura
07	Registrador de Saída de dados (Y MSB)	Leitura
08	Registrador de Saída de dados (X LSB)	Leitura
09	Registrador de Status	Leitura
10	Registrador de Identificação A	Leitura
11	Registrador de Identificação B	Leitura
12	Registrador de Identificação C	Leitura

Fonte: HMC5883L Honeywell Datasheet (2010)

16.6.2 Distorção Hard Iron

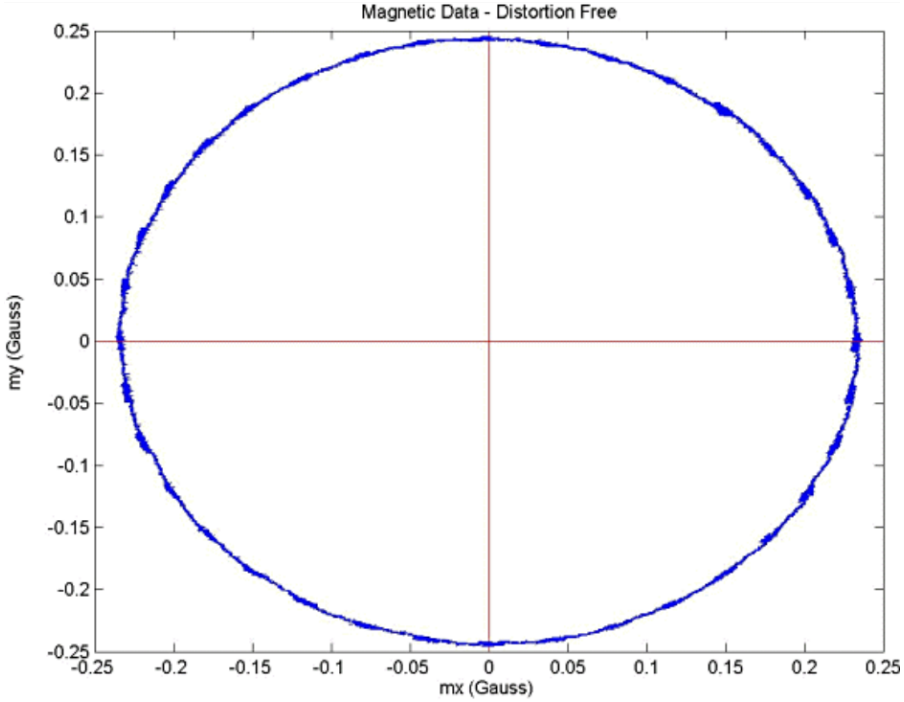
Distorções no campo magnético da Terra são resultados de influências magnéticas externas. Uma dessas influências é classificada como efeito Hard Iron.

O efeito Hard Iron é causado por materiais que apresentam um campo constante e aditivo ao campo magnético da Terra, gerando assim, um valor aditivo constante nas leituras de saída de cada eixo do magnetômetro. É ainda importante dizer, que é possível compensar apenas distorções provenientes daquilo que se movimenta junto com o sensor magnetômetro, no caso do quadrirrotor seria a estrutura e outros componentes eletrônicos, pelo fato da adição ser constante na rotação. Distorções originadas fora do sistema em que o sensor está preso são praticamente impossíveis de se compensar.

Quando ausente de influenciadores, ao rotacionar o magnetômetro em 360˚ e plotar o gráfico de suas leituras nos eixos x e y, obtém-se um círculo centralizado em (0,0), conforme a Figura 63 demonstra.

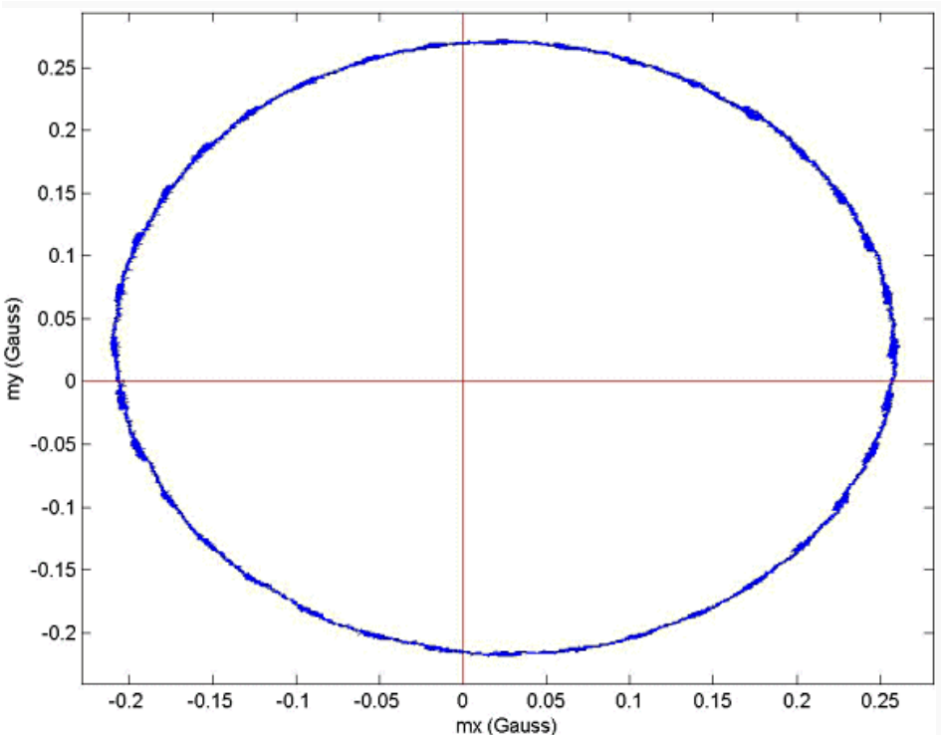
A presença do efeito Hard Iron produz um deslocamento (offset) no centro do círculo, conforme a Figura 64 demonstra.

Figura 63 – Gráfico das leituras do magnetômetro sem influenciadores externos. É possível notar uma centralização em na coordenada (0,0).



Fonte: Sensors Online (2009).

Figura 64 – Gráfico das leituras do magnetômetro com influenciadores externos. É possível notar um deslocamento na centralização.



Fonte: Sensors Online (2009).

Para compensar o efeito Hard Iron é necessário rotacionar o sensor por 360˚, e assim armazenar o valor máximo e mínimo lidos de cada eixo. O valor de offset ϵ para o eixo x é dado por:

$$\epsilon_x = \frac{(x_{max} + x_{min})}{2} \quad [47]$$

Sendo este offset aditivo e linear por todo o eixo x, basta então subtraí-lo dos dados lidos neste eixo. A mesma lógica é aplicada para os outros eixos.

Com estas informações repassadas, fica acessível o estudo do firmware relacionado ao HMC5883L no MultiWii Pro.

16.6.3 Firmware (HMC5883L)

Para o HMC5883L, assim como no giroscópio e acelerômetro, há três funções reservadas. Estas funções possuem os mesmos objetivos que as dos outros sensores, porém suas execuções possuem particularidades necessárias para a configuração, calibração e remoção de efeitos externos presentes na leitura deste magnetômetro. Das funções tem-se uma para inicialização, outra para aquisição de dados, e a última para calibração e remoção do efeito Hard Iron. Todas essas funções fazem o uso das funções gerais do protocolo TWI apresentadas na seção da comunicação TWI.

A função “Mag_init” inicializa e configura o sensor HMC5883L, porém esta, diferente dos outros dois sensores, acelerômetro e giroscópio, possui duas etapas. A primeira etapa realiza a operação de auto teste e a segunda coloca o sensor para medição contínua.

A operação de auto teste serve para checar o HMC5883L para operação apropriada. As modificações em cada registrador do sensor para este modo estão apresentadas na Tabela 24.

Tabela 24 – Modificações dos registradores do HMC5883L para o auto teste

Nome do Registrador	Endereço	Modificação	Descrição
Registrador de Configuração A	0x00	MS0 = 1 MS1 = 0	Excitação interna, com um campo magnético nominal de polarização positiva para os 3 eixos.
Registrador de Configuração A	0x00	DO0 = 0 DO1 = 0 DO2 = 1	Taxa de medições para modo contínuo = 15Hz. Porém não será usado este modo.
Registrador de Configuração A	0x00	MA0 = 1 MA1 = 1	Número de amostras para o cálculo da média de saída = 8.
Registrador de Configuração B	0x01	GN0 = 1 GN1 = 1 GN2 = 0	Ganho do dispositivo = ±2.5Ga.
Registrador de Modo	0x02	MD0 = 1 MD1 = 0	Modo de media única.

Fonte: Autoria própria.

A função “Mag_init” faz a escrita nos registradores de controle do sensor através da função “i2c_writeReg”

Logo após escrever nos registradores é feito a chamada da função “getADC”,

Mag_init

```
void Mag_init() {
    //test mode
    delay(100);
    i2c_writeReg(MAG_ADDRESS ,0x00 ,0x71 );
    delay(50);
    i2c_writeReg(MAG_ADDRESS ,0x01 ,0x60 );
    i2c_writeReg(MAG_ADDRESS ,0x02 ,0x01 );
    delay(100);
      getADC();
    delay(10);
    magCal[ROLL]  =  766.0 / abs(magADC[ROLL]);
    magCal[PITCH] =  766.0 / abs(magADC[PITCH]);
    magCal[YAW]   =  750.0 / abs(magADC[YAW]);
    // leave test mode
    i2c_writeReg(MAG_ADDRESS ,0x00 ,0x70 );
    i2c_writeReg(MAG_ADDRESS ,0x01 ,0x20 );
    i2c_writeReg(MAG_ADDRESS ,0x02 ,0x00 );
    magInit = 1;
  }
```

getADC

```
void getADC() {
  i2c_getSixRawADC(MAG_ADDRESS,MAG_DATA_REGISTER);
  MAG_ORIENTATION( ((rawADC[0]<<8) | rawADC[1]) ,
                   ((rawADC[4]<<8) | rawADC[5]) ,
                   ((rawADC[2]<<8) | rawADC[3]) );

}
```

A função “getADC” é a responsável pela aquisição dos dados crus fornecidos pelo sensor giroscópio através da chamada da função geral “i2c_getSixRawADC”, que dentro desta, armazena estes num vetor de 6 posições “rawADC”. Processo totalmente semelhante ao que ocorre no BMA180 e ITG32-00.

Cada posição do vetor “rawADC” recebe um dado de cada registrador, a Tabela 25 ilustra os dados em cada posição do vetor.

Tabela 25 – Dados do vetor rawADC (magnetômetro).

Posição	Registrador
rawADC[0]	X MSB
rawADC[1]	X LSB
rawADC[2]	Z MSB
rawADC[3]	Z LSB
rawADC[4]	Y MSB
rawADC[5]	Y LSB

Fonte: Autoria própria.

Após armazenar os dados do magnetômetro para o vetor “rawADC”, estes são passados de seu estado cru para seu estado tratado. O macro “MAG_ORIENTATION (X,Y,Z)” declarado na biblioteca “def.h” recebe os valores tratados do giroscópio. O tratamento é apenas um deslocamento de bits para compor uma variável de 16 bits, e, portanto, ter a sequência MSB-LSB unida para cada eixo. Este deslocamento também ocorre para o acelerômetro e giroscópio, porém estes também executam outras tarefas, como a retirada de bits não usuais e a diminuição da resolução. Para este sensor a resolução se mantém a mesma da saída do conversor A/D, 12 bits.

Voltando à função “Mag_init”, há o cálculo do fator de escala de calibração, razão da aplicação do auto teste. Segundo o datasheet do componente e com tradução autoral:

> Quando selecionado o modo de medida única, serão feitos dois ciclos de aquisição de dados em cada vetor magnético (cada eixo). A primeira aquisição é dada por um pulso seguido da medição dos dados do campo magnético externo. A segunda aquisição terá uma excitação de 10mA em polarização positiva para os três eixos, criando um campo magnético de ~1.1 Ga, fazendo então, offset com o campo magnético externo. O valor da primeira aquisição é subtraído com o da segunda. Este valor líquido é colocado nos registradores de saída de dados.

No auto teste, como mencionado, é adicionado ~1.1 Gauss ao campo já existente, e isto é o motivo principal de ter sido escolhido um ganho de $\pm$2.5Ga e não

maior, evitando saturar os registradores de dados. Tendo a Tabela 26 como referência, é possível determinar qual o valor da saída do ADC equivalente à ~1.1Ga.

Tabela 26 – Alcance e resolução das medidas (magenetômetro).

Alcance	LSB/Gauss	Output
±0.88Ga	1370	-2048-0-2047
±1.3Ga	1090	-2048-0-2047
±1.9Ga	820	-2048-0-2047
±2.5Ga	660	-2048-0-2047
±4.0Ga	440	-2048-0-2047
±4.7Ga	390	-2048-0-2047
±5.6Ga	330	-2048-0-2047
±8.1Ga	230	-2048-0-2047

Fonte: HMC5883L Honeywell Datasheet (2010)

O Datasheet do dispositivo recomenda 1.16Ga para os eixos x e y, e 1.08Ga para o eixo z, obtendo:

$$1.16G \times 660\frac{LSB}{Ga} = 766LSB \quad [48]$$

$$1.08G \times 660\frac{LSB}{Ga} = 713LSB \quad [49]$$

Tendo o valor calculado no auto teste, e o valor esperado de medida, para ~1.1Ga, é possível calcular um fator de escala de calibração, com:

$$magCal = \frac{Valor_{experado}}{Valor_{medido}} = \frac{766}{magADC} \quad [50]$$

Sendo que para o eixo z, o valor esperado é 713.

O fator de escala de calibração deve ser multiplicado com todas as futuras leituras de cada eixo.

Essa operação pode ser vista na função “Mag_init”, dando fim ao auto teste, e iniciando o modo de medida continua. Isso é feito refazendo a escrita nos registradores de controle do sensor através da função “i2c_writeReg”. A Tabela 27 apresenta as modificações.

Tabela 27 – Modificações dos registradores do HMC5883L para o modo de medida contínua.

Nome do Registrador	Endereço	Modificação	Descrição
Registrador de Configuração A	0x00	MS0 = 0 MS1 = 0	Configuração de medição normal.
Registrador de Configuração A	0x00	DO0 = 0 DO1 = 0 DO2 = 1	Taxa de medições para modo contínuo = 15Hz.
Registrador de Configuração A	0x00	MA0 = 1 MA1 = 1	Número de amostras para o cálculo da média de saída = 8.
Registrador de Configuração B	0x01	GN0 = 1 GN1 = 0 GN2 = 0	Ganho do dispositivo = ±1.3Ga.
Registrador de Modo	0x02	MD0 = 0 MD1 = 0	Modo de médida contínua.

Fonte: Autoria própria.

Após as configurações é feita a atribuição “magInit = 1”, indicando que a inicialização do sensor foi realizada.

Falta-se por fim explicar a função “Mag_getADC”, função responsável pela calibração para a remoção do efeito Hard Iron, além de atualizar os valores dos dados provenientes da saída dos registradores de dados do magnetômetro.

Mag_getADC

```
#if MAG
static float   magCal[3] = {1.0,1.0,1.0};
static uint8_t magInit = 0;

void Mag_getADC() {
  static uint32_t t,tCal = 0;
  static int16_t magZeroTempMin[3];
  static int16_t magZeroTempMax[3];
  uint8_t axis;
  if ( currentTime < t ) return;
  t = currentTime + 100000;
  TWBR = ((F_CPU / 400000L) - 16) / 2;
  getADC();
  magADC[ROLL]  = magADC[ROLL]  * magCal[ROLL];
  magADC[PITCH] = magADC[PITCH] * magCal[PITCH];
  magADC[YAW]   = magADC[YAW]   * magCal[YAW];
  if (f.CALIBRATE_MAG) {
    tCal = t;
    for(axis=0;axis<3;axis++) {
      conf.magZero[axis] = 0;
      magZeroTempMin[axis] = magADC[axis];
      magZeroTempMax[axis] = magADC[axis];
    }
    f.CALIBRATE_MAG = 0;
  }
  if (magInit) {
    magADC[ROLL]  -= conf.magZero[ROLL];
    magADC[PITCH] -= conf.magZero[PITCH];
    magADC[YAW]   -= conf.magZero[YAW];
  }

  if (tCal != 0) {
    if ((t - tCal) < 30000000) {
      LEDPIN_TOGGLE;
      for(axis=0;axis<3;axis++) {
        if (magADC[axis] < magZeroTempMin[axis])
magZeroTempMin[axis] = magADC[axis];
        if (magADC[axis] > magZeroTempMax[axis])
magZeroTempMax[axis] = magADC[axis];
      }
    } else {
      tCal = 0;
      for(axis=0;axis<3;axis++)
        conf.magZero[axis] = (magZeroTempMin[axis] + magZeroTempMax
[axis])/2;
      writeParams(1);
    }
  }
}
#endif
```

O trecho de código que contém a função “Mag_getADC” se inicia com a declaração das variáveis que estão apresentadas e explicadas na Tabela 28.

Tabela 28 – Variáveis locais da função

Variável	V.I	Descrição
magCal[3]	{1.0; 1.0; 1.0}	Variável para armazenar o fator de escala de calibração de cada eixo.
magInit	0	Indica se inicialização já foi feita ou não.
t	0	Variável para garantir a leitura a cada 100ms
tCAl	0	Tempo decorrente da calibração.
magZeroTempMIn[3]	-	Armazena o valor mínimo lido em cada eixo após rotacioná-los 360˚.
magZeroTempMax[3]	-	Armazena o valor máximo lido em cada eixo após rotacioná-los 360˚.
axis	-	Variável para referenciar os eixos.

Fonte: Autoria própria.

Após entrar na função e declarar as variáveis locais, é feita uma condição em relação ao tempo de operação. Essa condição garante uma leitura no mínimo a cada 100ms. Isso é feito para não atrasar operações mais importantes no loop principal.

Pode-se ver em sequência, o aumento da frequência da comunicação TWI (400 KHz). Isso só é possível, pois nas especificações do sensor, há descrita tal limitação. Essa operação é um resquício de um firmware originalmente genérico, que atende várias placas para vários sistemas.

É feito então a leitura das medições presentes nos registradores de saída de dados do dispositivo a partir da função “getADC”, já mencionada e explicada aqui. É aplicada nessas leituras a multiplicação do fator de escala de calibração, conforme a recomendação da documentação do dispositivo.

Em sequência é analisada a condição de requisição de calibração do dispositivo através da flag “f.CALIBRATE_MAG”. Se esta flag está levantada, entra-se na condição de calibração. Com a entrada na condição, há a atualização do valor de “tCal”, armazenando o valor de tempo de operação no exato momento em que se inicia a calibração. Há a atualização das variáveis que realizam a operação de calibração, com “conf.magZero” atribuindo o valor de 0 para cada um dos eixos do magnetômetro, “magZeroTempMin” e “magZeroTempMax” recebem o valor atual de leitura de cada eixo.

Há então a operação de calibração. Esta operação é realizada enquanto a condição “(tCal !=0)” for verdadeira. Nesta condição há o tratamento de tempo de calibração, fazendo a lógica necessária para que durante um intervalo de 30s ocorra a atualização dos valores de “magZeroTempMin” com o valor mínimo lido em cada eixo durante toda a rotação de 360˚, e “magZeroTempMin” com o valor máximo lido.

Após os 30s, “(tCal = 0)” é gravado na EEPROM a configuração “cong.magZero”, que possui o offset causado pelo efeito Hard Iron, dado pela equação [47].

Tendo realizada a inicialização e calibração, basta atribuir para cada leitura do registrador de saída de dados, com a aplicação do fator de escala de calibração, a subtração do offset. Tendo assim o valor final que será utilizado para a estimativa de ângulo de guinada pós aplicação da fusão dos sensores.

16.7 Inicialização dos sensores

initSensors()

```
void initSensors() {
  delay(200);
  delay(100);
  i2c_init();
  delay(100);
  if (GYRO) Gyro_init();
  if (BARO) Baro_init();
  if (MAG) Mag_init();
  if (ACC) {ACC_init();acc_25deg = acc_1G * 0.423;}
  f.I2C_INIT_DONE = 1;
}
```

Para inicializar a comunicação TWI e os sensores, é utilizada a função “initSensors”.

Pode-se verificar ainda, que há a atribuição da variável “acc_25deg”. Essa variável será utilizada na aplicação do filtro complementar para identificar se o sistema está com inclinação acima de 25˚ ou não, e tem seu cálculo utilizando apenas dados do acelerômetro, como pode ser visto.

17. IMU.INO

17.1 IMU

Estimação de atitude é um tópico crucial em qualquer sistema aéreo autônomo, e é a etapa que antecede a aplicação do sistema de controle. Sensores baseados em sistemas micro-eletromecânicos (do inglês Microelectromechanical Systems, MEMS) de três eixos, sendo eles o acelerômetro, giroscópio e magnetômetro, fazem parte da combinação mais utilizada em uma unidade de medida inercial (do inglês Inertial Measurement Unit, IMU). Devido a popularização dos quadrirrotores, e consequentemente destes sensores, gerou-se a necessidade da produção em larga escala visando um produto de baixo custo. Portanto, são dispositivos sujeitos a ruídos e altamente tortuosos. (RAMOS et al., 2017; BARANEK e SOLE, 2013).

Cada um dos 3 sensores que compõem o sistema de medida inercial fornece alguma informação em relação à atitude. Entretanto, para uma estimativa com erro limitado seria inviável utilizar apenas essa informação, necessitando de um algoritmo sofisticado. (BARANEK e SOLE, 2013)

Das técnicas utilizadas para estimar a atitude de um quadrirrotor, destacam-se: Filtro Complementar (BARANEK e SOLE, 2013), Filtro Complementar Não Linear (MAHONY et al., 2008), Filtro Complementar Variante no Tempo (PASCOAL et al., 2000), Filtro de Kalman Convencional (LEFFERTS et al., 1983), Filtro de Kalman Estendido (Gao et al., 2006).

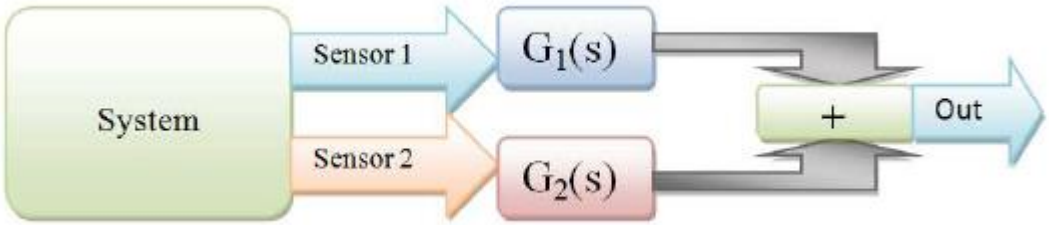
Todas essas técnicas citadas acima possuem sua eficiência para determinado objetivo. A técnica inspirada no *MultiWii Pro, e* que, portanto, será enunciada, é do Filtro Complementar. Seu conceito será passado aqui neste tópico e a adaptação para sua implementação pode ser observada nos mínimos detalhes no estudo do firmware.

17.1.1 Filtro Complementar

O Filtro Complementar é uma técnica em função do domínio da frequência, ou seja, dois ou mais sensores, considerados entradas do sistema, cobrem todo o espectro de frequência, assim, portanto, cada sensor é responsável por apenas parte

do espectro. Pode-se dizer, que um sensor complementa o outro no domínio da frequência. (BARANEK e SOLE, 2013). Um diagrama de blocos, exemplificando o princípio do filtro é mostrado na Figura 65.

Figura 65 - Diagrama de blocos de um Filtro Complementar Linear para dois sensores.



Fonte: BARANEK e SOLE. (2013).

O termo “Filtro Complementar” é usado quando se refere a um algoritmo digital que mistura dados redundantes de diferentes sensores, na qual as frequências de seus ruídos são diferentes. O filtro se refere ao uso de duas ou mais funções de transferências, na qual o somatório delas gera um valor unitário. Geralmente com um sistema de duas entradas, uma das entradas fornece dados com ruídos em baixa frequência, onde é filtrada por um filtro passa-alta, enquanto a outra entrada fornece dados com ruídos em alta frequência, sendo filtrada por um filtro passa-baixa. Matematicamente, se o filtro passa-baixa e o filtro passa-alta são complementares, a saída do filtro é uma reconstrução completa do sinal. (HYSTAD, 2015).

Um filtro complementar aplicado a um sistema de duas entradas com sinais $y1$ e $y2$, onde $y1$ contém ruído em alta frequência e $y2$ ruído em baixa frequência, é dado por:

$$Y_{cf} = G_{lp}(s)Y_1 + G_{hp}(s)Y_2 \quad [51]$$

Sendo:

$$G_{lp}(s) = \frac{1}{\tau s + 1} \quad [52]$$

$$G_{hp}(s) = \frac{\tau s}{\tau s + 1} \quad [53]$$

Com a frequência de corte sendo dada por:

$$fc = \frac{1}{\tau\pi} \quad [54]$$

17.1.2 Discretização

Pensando em uma aplicação para sistemas microcontrolados, deve-se discretizar as equações [51], [52] e [53].

17.1.2.1 Filtro Passa-Baixa

Sendo um sinal, aplicado a equação da função de transferência do filtro passa-baixa (equação [52]) obtem-se:

$$Y_{lp} = \frac{1}{\tau s + 1} Y \quad [55]$$

$$Y_{lp}(\tau s + 1) = Y \overset{\mathcal{L}^{-1}}{\Longrightarrow} \dot{y_{lp}}\tau + y_{lp} = y \quad [56]$$

A aproximação por diferenças (backward) é dada pela equação [57]:

$$\dot{x} \approx \frac{x[n] - x[n-1]}{h} \quad [57]$$

Onde $x[n]$ é a enésima amostra de x, e h é o tempo entre amostras. Aplicando a equação [57] na equação [56], obtém-se:

$$\frac{y_{lp}[n] - y_{lp}[n-1]}{h}\tau + y_{lp}[n] = y[n] \quad [58]$$

$$y_{lp}[n] = \frac{\tau}{\tau + h} y_{lp}[n-1] + \frac{h}{\tau + h} y[n-1] \quad [59]$$

Definindo que:

$$a = \frac{\tau}{\tau + h} \quad [60]$$

$$\frac{h}{\tau + h} = 1 - a \quad [61]$$

Onde a pode ser configurado como um valor constante, e é considerado um parâmetro de sintonização. Tem-se então como equação para aplicação do filtro passa-baixa em um algoritmo discreto:

$$\boldsymbol{y_{lp}[n] = a y_{lp}[n-1] + (1-a) y[n]} \quad [62]$$

17.1.2.2 Filtro Passa-Alta

Sendo um sinal, aplicado a equação da função de transferência do filtro passa-alta (equação [53]) obtem-se:

$$Y_{hp} = \frac{\tau s}{\tau s + 1} Y \quad [63]$$

$$Y_{hp}(\tau s + 1) = \tau s Y \overset{\mathcal{L}^{-1}}{\Longrightarrow} \dot{y_{hp}}\tau + y_{hp} = \tau \dot{y} \quad [64]$$

Aplicando a equação [57] na equação [64], obtém-se:

$$\frac{y_{hp}[n] - y_{hp}[n-1]}{h}\tau + y_{hp}[n] = \frac{y[n] - y[n-1]}{h}\tau \quad [65]$$

Já tendo a definição de a, tem-se então como equação para aplicação do filtro passa-alta em um algoritmo discreto:

$$\boldsymbol{y_{hp}[n] = a y_{hp}[n-1] + a(y[n] - y[n-1])} \quad [66]$$

17.1.2.3 Filtro Complementar

Tendo as equações [62] e [66], correspondentes a aplicação do filtro passa-baixa e passa-alta, respectivamente. Em uma aplicação para sistemas discretos, pode-se equacionar o Filtro Complementar:

$$Y_{cf} = \frac{1}{\tau s + 1} Y_1 + \frac{\tau s}{\tau s + 1} Y_2 \quad [67]$$

$$Y_{cf}(\tau s + 1) = Y_1 + \tau s Y_2 \overset{\mathcal{L}^{-1}}{\Longrightarrow} \dot{y_{cf}}\tau + y_{cf} = y_1 + \tau \dot{y_2} \quad [68]$$

Aplicando a equação [57] na equação [68], obtém-se:

$$\frac{y_{cf}[n] - y_{cf}[n-1]}{h}\tau + y_{cf}[n] = y_1[n] + \frac{y_2[n] - y_2[n-1]}{h}\tau \quad [69]$$

$$\boldsymbol{y_{cf}[n] = a y_{cf}[n-1] + (1-a) y_1[n] + a(y_2[n] - y_2[n-1])} \quad [70]$$

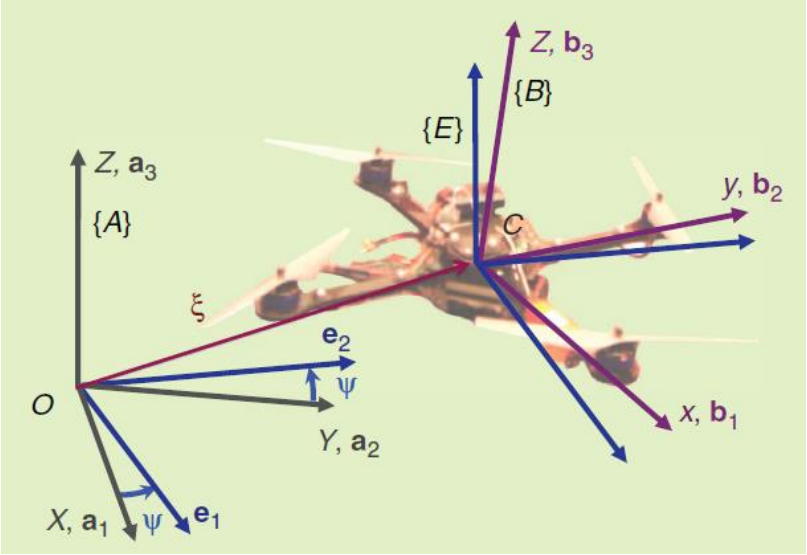
17.2 Rotação

Em relação aos sistemas de coordenadas, MAHONY et al., (2012) denomina de acordo com a Figura 66:

- {A} o sistema de referência inercial, com vetores unitários dos eixos $\{\overrightarrow{a_1}, \overrightarrow{a_2}, \overrightarrow{a_3}\} = \{\vec{x}, \vec{y}, \vec{z}\}$.
- {B} o sistema de referência do corpo, com vetores unitários $\{\overrightarrow{b_1}, \overrightarrow{b_2}, \overrightarrow{b_3}\}$, onde estes são os eixos do sistema {B} em relação ao sistema {A}.
- {E} o sistema de referência estimado, com vetores unitários $\{\overrightarrow{e_1}, \overrightarrow{e_2}, \overrightarrow{e_3}\}$, onde estes são os eixos do sistema {E} em relação ao sistema {A}.

O sistema de referência estimado, pode ser considerado como a posição intermediária na transição por um modelo de rotação de {A} para {B}. A Figura 66 ilustra isso com a primeira rotação referente ao eixo a_3, ou seja, angulação de guinada (ψ).

Figura 66 - Sistemas de Coordenadas.



Fonte: MAHONY et al. (2012).

Para transformar vetores entre diferentes sistemas de coordenadas, utilizam-se as matrizes de rotação, conforme foi introduzido pela Figura 66. Uma matriz de rotação que transforma um vetor com representação em A para B é denominada R_B^A. Geralmente uma matriz de rotação é qualquer matriz satisfazendo:

$$RR^T = R^T R = I, \qquad det(R) = 1 \tag{71}$$

Isso implica que R é ortogonal, e como consequência, a inversa da matriz de rotação é dada por: $R^{-1} = R^T$. Assim:

$$R_B^A = (R_A^B)^{-1} = (R_A^B)^T \tag{72}$$

Ângulos de Euler não são comutativos, e uma sequência de rotação deve ser imposta. A convenção mais utilizada na engenharia aeroespacial é: guinada,

arfagem, rolagem (Z-Y-X). Isto implica que a matriz de rotação R_B^A, pode ser definida por:

$$R_B^A = R_{(z,\psi)} R_{(y,\theta)} R_{(x,\varphi)} \quad [73]$$

$$R_B^A = \begin{bmatrix} c\psi & -s\psi & 0 \\ s\psi & c\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\theta & 0 & s\theta \\ 0 & 1 & 0 \\ -s\theta & 0 & c\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\varphi & -s\varphi \\ 0 & s\varphi & c\varphi \end{bmatrix} \quad [74]$$

$$R_B^A = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\varphi - s\psi c\varphi & c\psi s\theta c\varphi + s\psi s\varphi \\ s\psi c\theta & s\psi s\theta s\varphi + c\psi c\varphi & s\psi s\theta c\varphi - c\psi s\varphi \\ -s\theta & c\theta s\varphi & c\theta c\varphi \end{bmatrix} \quad [75]$$

17.3 Firmware (imu.ino)

Toda a programação apresentada nesta seção possui referência[11] didática do Filtro Complementar aplicado em técnicas[12] apropriadas e adaptadas.

O bloco de programação IMU.ino se resume na chamada da função "ComputeIMU" no looping principal do programa geral.

[11] HYSTAD, A. V. **Model, Design and Controle of a Quadcopter**. Master of Science in Cybernetics and Robotics (Engineering Cybernetics) Norwegian University of Science and Technology –NTNU. 181p, 2015.

[12] STARLINO ELETRONICS. **A guide to using IMU (Accelerometer and Gyroscope devices) in embedded applications**. Disponível em: http://www.starlino.com/imu_guide.html. Acesso em: Maio de 2019.

ComputeIMU

```
void computeIMU () {
  uint8_t axis;
  static int16_t gyroADCprevious[3] = {0,0,0};
  int16_t gyroADCp[3];
  int16_t gyroADCinter[3];
  static uint32_t timeInterleave = 0;

  ACC_getADC();
  getEstimatedAttitude();
  Gyro_getADC();

  for (axis = 0; axis < 3; axis++)
    gyroADCp[axis] =  gyroADC[axis];

  timeInterleave=micros();
  annexCode();

  if ((micros()-timeInterleave)>650) {
     annex650_overrun_count++;
  } else {
     while((micros()-timeInterleave)<650) ;
  }

  Gyro_getADC();

  for (axis = 0; axis < 3; axis++) {
    gyroADCinter[axis] =  gyroADC[axis]+gyroADCp[axis];
     gyroData[axis] = (gyroADCinter[axis]+gyroADCprevious[axis])/3;
      gyroADCprevious[axis] = gyroADCinter[axis]/2;
  }
}
```

Em “computeIMU”, resumidamente, ocorrem duas leituras consecutivas da saída do conversor analógico/digital do giroscópio. Essas leituras devem ser separadas por um intervalo de 650ms para que não ocorra interferência no controle do quadrirrotor.

Em ordem sequencial, a função “computeIMU” se inicia com a declaração de 5 variáveis. Estas variáveis estão descritas na Tabela 29.

Tabela 29 – Variáveis da função “computeIMU”

Variável	Descrição
axis	Variável que referencia os eixos (roll, pitch e yaw).
gyroADCp[3]	Vetor que armazena a primeira leitura do conversor A/D do giroscópio.
gyroADCinter[3]	Vetor que armazena a soma da primeira e segunda leitura do conversor A/D do giroscópio.
gyroADCprevious[3]	Vetor que armazena a média da primeira e segunda leitura da iteração anterior do conversor A/D do giroscópio.
timeInterleave	Variável para armazenar o tempo de duração da função “annexCode”.

Fonte: Autoria Própria.

Em seguida tem-se a geração de novos dados do conversor A/D do acelerômetro a partir de “ACC_getADC” para que então seja chamada a função “getEstimatedAttitude”, que será ainda explicada neste capítulo, mas que basicamente aplica a fusão dos sensores para ter os valores dos ângulos de rolagem, arfagem e guinada. A função “Gyro_getADC” realiza a geração de novos dados do conversor A/D do giroscópio e os armazenam no vetor “gyroADCp”. É importante ter em mente, que na primeira iteração do código, a função “getEstimatedAttitude” terá apenas os valores do acelerômetro, apenas aplicando os valores do giroscópio a partir da segunda iteração.

Ainda na função “computeIMU”, há a chamada da função “annexCode”, que resumidamente, trata os valores recebidos pelo receptor do rádio controle, para que assim, aplique estes dados tratados para o controle de atitude. Essa função é comentada em detalhes na seção do código principal. Ainda, um adendo importante sobre esta função, é que ela deve durar sempre 650ms para que não afete o loop do controle, caso não ocorra o mesmo, o contador “annex650_overrun_count” é incrementado para análise via software de controle de solo.

Em sequência, atualizam-se os dados do conversor A/D do giroscópio e, trabalhando as duas leituras consecutivas do giroscópio, tem-se a aquisição de valor

do vetor “gyroData”, valor este que será aplicado diretamente no controle de atitude do quadrirrotor.

$$\text{gyroData} = \frac{(\text{gyroADCinter} + \text{gyroADCprevious})}{3} \quad [76]$$

De maneira mais compreensiva:

$$= \frac{(\text{LEITURA1}[n] + \text{LEITURA2}[n]) + \frac{(\text{LEITURA1}[n-1] + \text{LEITURA2}[n-1])}{2}}{3} \quad [77]$$

Sendo LEITURA1[n] a enésima primeira leitura do giroscópio, e LEITURA2[N] a enésima segunda leitura do giroscópio.

17.3.1 Macros Definidos

Nas funções que serão explicadas subsequentemente, têm-se a necessidade de apresentar os macros definidos utilizados nestas. Estes estão descritos na Tabela 30.

Tabela 30 - Macros definidos em IMU.H

Macro	Valor
ACC_LPF_FACTOR	100
MG_LPF_FACTOR	4
GYR_CMPF_FACTOR	400.0f
GYR_CMPFM_FACTOR	200.0f
INV_GYR_CMPF_FACTOR	$\frac{1{,}0f}{\text{GYR}_{\text{CMPF}_{\text{FACTOR}}} + 1.0f}$
INV_ GYR_CMPFM_FACTOR	$\frac{1{,}0f}{\text{GYR}_{\text{CMPFM}_{\text{FACTOR}}} + 1.0f}$
GYRO_SCALE	$\frac{2279{,}44\pi}{2^{13} \times 180 \times 10^6}$

Fonte: Autoria Própria.

Onde, “ACC_LPF_FACTOR” é fator do filtro passa-baixa para suavizar a aquisição dos dados do acelerômetro;

“MG_LPF_FACTOR” é fator do filtro passa-baixa para suavizar a aquisição dos dados do magnetômetro;

“GYR_CMPF_FACTOR” é o peso de relevância dos sensores do filtro complementar (acelerômetro + giroscópio);

“GYR_CMPFM_FACTOR” é o peso de relevância dos sensores do filtro complementar (magnetômetro + giroscópio);

“INV_GYR_CMPF_FACTOR” e “INV_ GYR_CMPFM_FACTOR” são os divisores da aplicação dos pesos do filtro complementar.

O termo "$\frac{2279,44}{2^{13}}$" de “GYRO_SCALE”, é o escalar responsável por transformar os valores de saída do conversor analógico/digital do giroscópio em $^{\circ}/s$, este valor é equivalente ao inverso da sensibilidade do sensor para 14 bits, "$\frac{1}{\frac{14,375}{4}}$". O termo "$\frac{\pi}{180}$" apenas faz a conversão para rad/s. O termo "$\frac{1}{10^6}$" é proveniente de uma futura operação de integração simples para a aquisição do ângulo em radianos. Esta integração é demonstrada na função “getEstimatedAttitude”

17.3.2 Orientações e Matriz de Rotação

Para a aplicação do filtro complementar, é necessário padronizar os valores entregues pelos sensores. Em alguns casos é necessário inverter os valores devido a posição física do giroscópio em relação ao acelerômetro. Deve-se também considerar as grandezas mensuradas por cada sensor, onde por exemplo, quando há uma rotação em um eixo, e, portanto, um valor entregue pelo giroscópio, o acelerômetro tende a entregar um valor em um sentido contraposto naquele mesmo eixo, sendo devido a este mensurar as componentes do vetor de gravitação. Neste último caso, os valores dos dois sensores devem ser ajustados para serem incrementados e decrementados juntos.

O magnetômetro fornece apenas saídas positivas em seus três eixos. Quando usado para estimativa do ângulo de guinada, estes valores devem ser ajustados de acordo com o sentido de progressão das saídas do giroscópio.

As mudanças nas saídas dos sensores foram realizadas por um macro na biblioteca de definição, e podem ser vistas no bloco de código abaixo. Para entender a fundo as adaptações feitas, o documento de cada sensor deve ser consultado.

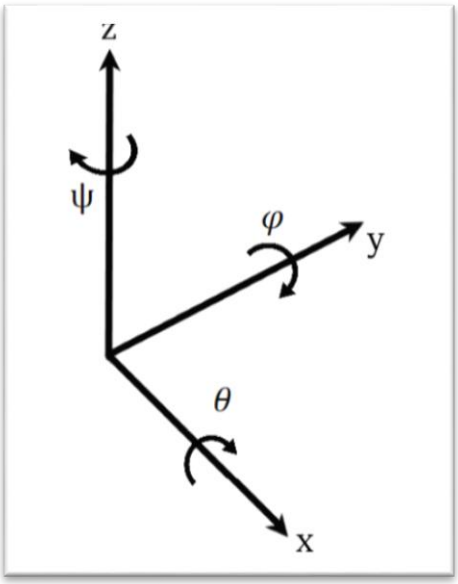
Macros(def.h)

```
ACC_ORIENTATION(X, Y, Z)  {accADC[ROLL]  = -X; accADC[PITCH]  = -Y;
accADC[YAW]  =  Z;}
GYRO_ORIENTATION(X, Y, Z) {gyroADC[ROLL] =  Y; gyroADC[PITCH]= -X;
gyroADC[YAW] = -Z;}
MAG_ORIENTATION(X, Y, Z)  {magADC[ROLL]  = -Y;  magADC[PITCH]  = X;
magADC[YAW]  = -Z;}
```

O MultiWii utiliza a aplicação de dois filtros complementares, um contendo o giroscópio e acelerômetro para a estimativa dos ângulos de rolagem e arfagem e outro contendo o giroscópio e magnetômetro para a estimativa do ângulo de guinada. Como pode-se ver, o giroscópio é o sensor comum entre os dois sistemas, e é ele que decide quais os sentidos de rotação, e consequentemente, a matriz de rotação aplicada no firmware.

Segundo o documento do ITG32-00 e as mudanças nas saídas apresentadas no bloco de código acima, tem-se as orientações dos ângulos de Euler apresentadas na Figura 67, onde θ = arfagem, φ = rolagem e ψ = guinada.

Figura 67 – Orientação de referência para a IMU

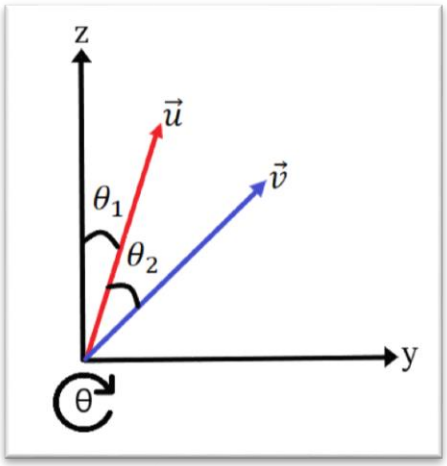


Fonte: Autoria Própria.

Com essas orientações, pode-se equacionar a matriz de rotação para cada um dos planos.

A Figura 68 demonstra um sistema, considerando o plano Y-Z da Figura 67, com um vetor $\vec{u}$ indicando a posição inicial de um sistema com angulação θ_1, e o vetor $\vec{v}$ indicando uma variação dessa posição por θ_2.

Figura 68 – Rotação no plano Y-Z.



Fonte: Autoria Própria.

Sendo e o módulo dos vetores, equacionando as componentes dos vetores, tem-se:

$$\vec{u}_z = e\cos(\theta_1) \quad [78]$$

$$\vec{u}_y = e\,\mathrm{sen}(\theta_1) \quad [79]$$

$$\vec{v}_z = e\cos(\theta_1 + \theta_2) \quad [80]$$

$$\vec{v}_y = e\,\mathrm{sen}(\theta_1 + \theta_2) \quad [81]$$

Aplicando identidades trigonométricas para [80] e [81], obtém-se:

$$\vec{v}_z = e\,[\cos(\theta_1)\cos(\theta_2) - sen(\theta_1)\,sen(\theta_2)] \quad [82]$$

$$\vec{v}_z = e\cos(\theta_1)\cos(\theta_2) - e\,sen(\theta_1)\,sen(\theta_2) \quad [83]$$

$$\boldsymbol{\vec{v}_z = \vec{u}_z cos(\theta_2) - \vec{u}_y sen(\theta_2)} \quad [84]$$

$$\vec{v}_y = e\,[\mathrm{sen}(\theta_1)\cos(\theta_2) + \ sen(\theta_2)\ cos(\theta_1)] \quad [85]$$

$$\vec{v}_y = e\,\mathrm{sen}(\theta_1)\cos(\theta_2) + \ e\ sen(\theta_2)\ cos(\theta_1) \quad [86]$$

$$\boldsymbol{\vec{v}_y = \vec{u}_y\, cos(\theta_2) + \ \vec{u}_z sen(\theta_2)} \quad [87]$$

Portanto:

$$\begin{bmatrix}\vec{v}_x\\ \vec{v}_y\\ \vec{v}_z\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & \boldsymbol{cos(\theta_2)} & \boldsymbol{sen(\theta_2)}\\ 0 & \boldsymbol{-sen(\theta_2)} & \boldsymbol{cos(\theta_2)}\end{bmatrix} \times \begin{bmatrix}\vec{u}_x\\ \vec{u}_y\\ \vec{u}_z\end{bmatrix} \quad [88]$$

Assim, a matriz de rotação generalizada para qualquer variação de ângulo θ, é dada por:

$$R_{(x,\theta)} = \begin{bmatrix}1 & 0 & 0\\ 0 & c\theta & s\theta\\ 0 & -s\theta & c\theta\end{bmatrix} \quad [89]$$

Fazendo os mesmos passos para os planos X-Z e X-Y, se obtém as seguintes matrizes de rotação:

$$R_{(y,\varphi)} = \begin{bmatrix}c\varphi & 0 & s\varphi\\ 0 & 1 & 0\\ -s\varphi & 0 & c\varphi\end{bmatrix} \quad [90]$$

$$R_{(x,\theta)} = \begin{bmatrix}c\psi & s\psi & 0\\ -s\psi c\psi & 0 & \\ 0 & 0 & 1\end{bmatrix} \quad [91]$$

Na convenção: guinada⟶ arfagem⟶ rolagem (Z-X-Y), tem-se:

$$R_B^A = \begin{bmatrix}c\psi & s\psi & 0\\ -s\psi c\psi & 0 & \\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & c\theta & s\theta\\ 0 & -s\theta & c\theta\end{bmatrix}\begin{bmatrix}c\varphi & 0 & s\varphi\\ 0 & 1 & 0\\ -s\varphi & 0 & c\varphi\end{bmatrix} \quad [92]$$

$$R_B^A = \begin{bmatrix}c\psi c\varphi - s\varphi s\psi s\theta & s\psi c\theta & c\psi s\varphi + c\varphi s\psi s\theta\\ -s\psi c\varphi - s\varphi c\psi s\theta & c\psi c\theta & -s\psi s\varphi + c\psi s\theta\\ -s\varphi c\theta & -s\theta & c\theta\end{bmatrix} \quad [93]$$

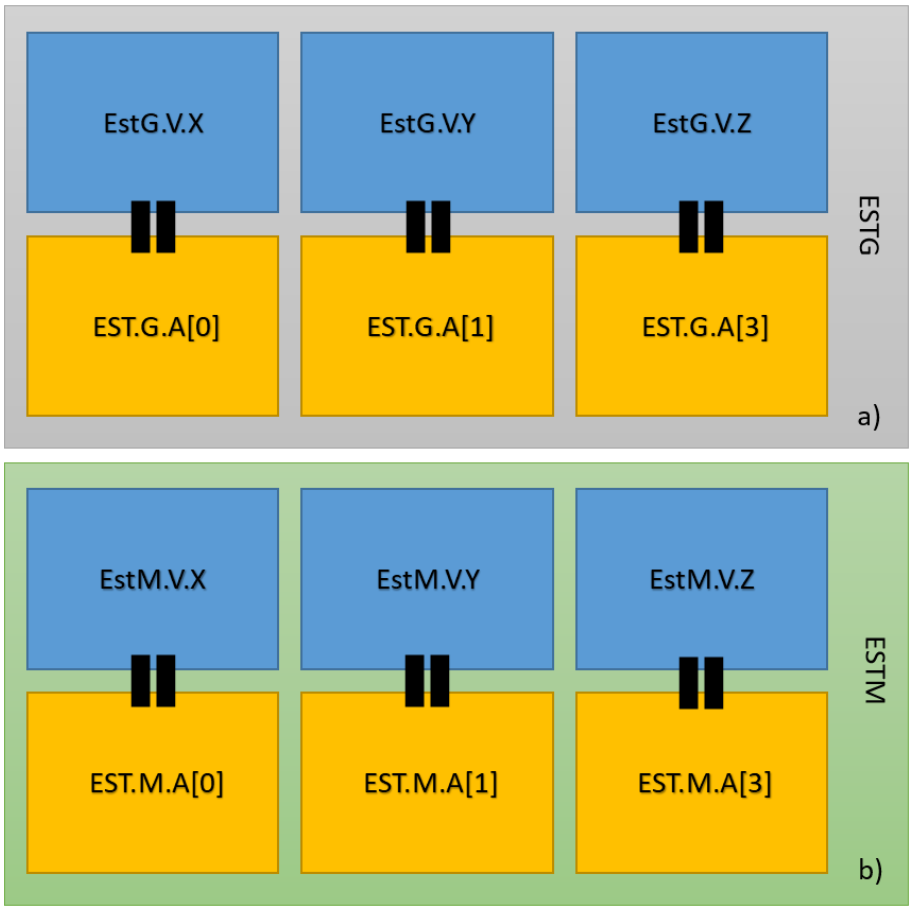
17.3.3 Estrutura para vetores estimados.

Para facilitar as abordagens de programação, se estabelece criações de estruturas e uniões de endereços para referenciar as componentes dos vetores estimados. Tem-se uma estrutura para a rotação estimada para o sistema giroscópio + acelerômetro e outra para sistema giroscópio + magnetômetro. A Figura 69 faz a ilustração do código abaixo com as estruturas já declaradas.

Estruturas das componentes dos vetores estimados

```
typedef struct fp_vector {
  float X;
  float Y;
  float Z;
} t_fp_vector_def;

typedef union {
  float   A[3];
  t_fp_vector_def V;
} t_fp_vector;

//static t_fp_vector EstM;
//static t_fp_vector EstG;
```

Figura 69 – Estruturas para as componentes dos vetores estimados; O símbolo de igualdade simboliza o compartilhamento de endereço; a) Estrutura para o sistema giroscópio + acelerômetro; b) Estrutura para o sistema giroscópio + magnetômetro.



Fonte: Autoria Própria.

Aplicando as notações do firmware para a equação [93], tem-se:

$$\begin{bmatrix} EstGVx[n] \\ EstGVy[n] \\ EstGVz[n] \end{bmatrix} = R_B^A \times \begin{bmatrix} EstGVx[n-1] \\ EstGVy[n-1] \\ EstGVz[n-1] \end{bmatrix} \quad [94]$$

$$\begin{bmatrix} EstMVx[n] \\ EstMVy[n] \\ EstMVz[n] \end{bmatrix} = R_B^A \times \begin{bmatrix} EstMVx[n-1] \\ EstMVy[n-1] \\ EstMVz[n-1] \end{bmatrix} \quad [95]$$

17.3.4 ATAN_2

Para o cálculo do arco tangente, com o objetivo de calcular os ângulos de rolagem, arfagem e guinada pós aplicação dos pesos de relevância dos sensores para os dois filtros presentes no firmware, utiliza-se a função “_atan2”.

_atan2

```
int16_t _atan2(float y, float x){
  #define fp_is_neg(val) ((((uint8_t*)&val)[3] & 0x80) != 0)

  float z = y / x;
  int16_t zi = abs(int16_t(z * 100));
  int8_t y_neg = fp_is_neg(y);

  if ( zi < 100 ){
    if (zi > 10)
     z = z / (1.0f + 0.28f * z * z);
   if (fp_is_neg(x)) {
     if (y_neg) z -= PI;
     else z += PI;
   }
  } else {
   z = (PI / 2.0f) - z / (z * z + 0.28f);
   if (y_neg) z -= PI;
  }
  z *= (180.0f / PI * 10);   //retorna em graus*10
  return z;
}
```

A função “_atan2” recebe como parâmetro duas coordenadas ortogonais dos vetores estimados, e calcula, então, o ângulo da inclinação deste vetor.

Baseando-se nas aproximações de HASTINGS. (1955) e fazendo os ajustes dos valores de acordo com o quadrante, tem-se:

$$\_atan2(z), \qquad z = \frac{y}{x} \tag{95}$$

Se 1>|z|>0.1 ;x>0 =

$$z = \frac{z}{1 + 0{,}28 \times z^2} \tag{96}$$

Se 1>|z| ;x<0 ;y<0 =

$$z = \frac{z}{1 + 0{,}28 \times z^2} - \pi \quad [96]$$

Se 1>|z| ; x<0 ;y>0 =

$$z = \frac{z}{1 + 0{,}28 \times z^2} + \pi \quad [97]$$

Se |z|>1 ;y>0=

$$z = \frac{\pi}{2} - \frac{z}{z^2 + 0{,}28} \quad [98]$$

Se |z|>1 ;y<0 =

$$z = \frac{\pi}{2} - \frac{z}{z^2 + 0{,}28} - \pi \quad [99]$$

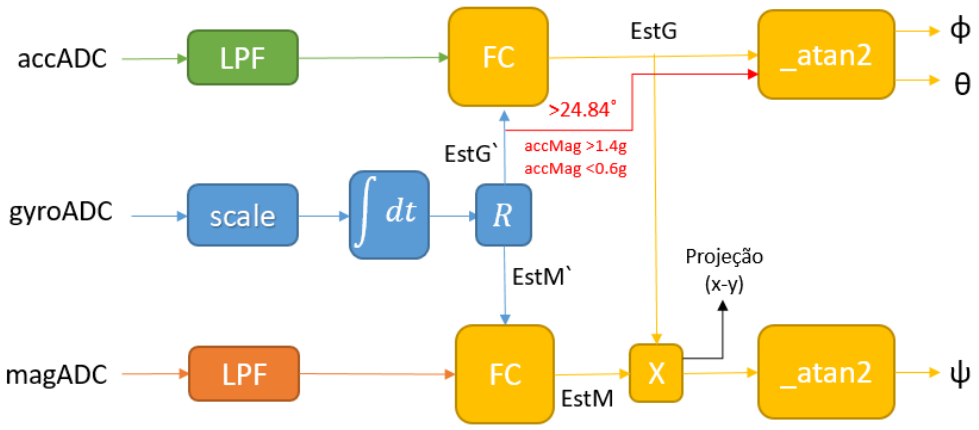
Por fim, retorna-se z convertido em graus e com um multiplicador de 10x. Portando, se o ângulo calculado for de 299˚, "_atan2" retornará 2999˚

É importante destacar a técnica de identificação de valores negativos a partir da função "fp_is_neg". Onde tendo uma orientação "little endianess" e complemento para dois, o bit mais significativo de uma variável sempre terá nível alto, se esta carregar um valor negativo. Como esta análise é feita com uma variável float (32 bits), divide-se esta em 4 posições de um vetor, e se faz a análise do bit significativo apenas na última posição deste.

17.3.5 Aplicação do Filtro Complementar

A função "getEstimatedAttitude", responsável por aplicar o Filtro Complementar e fazer a geração dos ângulos de rolagem, arfagem e guinada, é totalmente resumível a partir da Figura 70. As variáveis declaradas dentro desta função estão dispostas na Tabela 31.

Figura 70 – Fases para obtenção dos ângulos de atitude.



Fonte: Autoria própria.

Tabela 31 – Variáveis da função "getEstimatedAttitude"

Variável	Descrição
axis	Variável que referencia os eixos (roll, pitch e yaw).
accMag	Armazena a magnitude do vetor de aceleração.
EstG	Vetores Estimados para o Filtro Complementar (Giroscópio + Acelerômetro). Ver Figura 70.
EstM	Vetores Estimados para o Filtro Complementar (Giroscópio + Magnetômetro). Ver Figura 70.
mgSmooth[3]	Vetor para armazenar valores suavizados do magnetômetro, pós passa-baixa.
accSmooth[3]	Vetor para armazenar valores suavizados do acelerômetro, pós passa-baixa.
previousT	Termo anterior em microssegundos. Utilizado para realizar integração simples.
currentT	Termo atual em microssegundos. Utilizado para realizar integração simples.
scale	GYRO_SCALE multiplicado pelo tempo de integração.
deltaGyroAngle[3]	Vetor para armazenar a variação dos ângulos de atitude, pós integração.

Fonte: Autoria Própria.

getEstimatedAttitude - (variáveis declaradas)

```
void getEstimatedAttitude(){
  uint8_t axis;
  int32_t accMag = 0;
  static t_fp_vector EstG;
  static t_fp_vector EstM;
  static int16_t mgSmooth[3];
  static float accLPF[3];
  static uint16_t previousT;
  uint16_t currentT = micros();
  float scale, deltaGyroAngle[3];


  scale = (currentT - previousT) * GYRO_SCALE;

  previousT = currentT;
//...
```

getEstimatedAttitude - (integração e filtros)

```
//...
  for (axis = 0; axis < 3; axis++) {
    deltaGyroAngle[axis] = gyroADC[axis]  * scale;

    accLPF[axis] = accLPF[axis] * (1.0f - (1.0f/ACC_LPF_FACTOR)) +
accADC[axis] * (1.0f/ACC_LPF_FACTOR);
    accSmooth[axis] = accLPF[axis];

    #define ACC_VALUE accSmooth[axis]

    mgSmooth[axis] = (mgSmooth[axis] * (MG_LPF_FACTOR - 1) + magAD
C[axis]) / MG_LPF_FACTOR;

    #define MAG_VALUE mgSmooth[axis]

    accMag += (int32_t)ACC_VALUE*ACC_VALUE ;

  }

  accMag = accMag*100/((int32_t)acc_1G*acc_1G);
//...
```

17.3.5.1 Aplicação da escala e Integração

O vetor de três posições “deltaGyroAngle” armazena a variação dos ângulos de atitude em radianos, ou seja, é equivalente dizer que armazena o valor logo após a integração da Figura 70, pois o valor da saída do conversor A/D do giroscópio faz a multiplicação por sua escala de conversão e posteriormente pelo tempo de integração. Algebricamente:

$$\text{deltaGyroAngle} = gyroADC \times \frac{2380\pi}{2^{13}.180.} \times \frac{t}{10^6} \quad [100]$$

O tempo de integração é calculado em segundos, justificando a constante "$\frac{1}{10^6}$" (micros() retorna em microssegundos), e é o tempo decorrido entre duas aquisições de “deltaGyroAngle”.

17.3.5.2 Filtros passa-baixas

Além do filtro passa-baixa implícito em cada filtro complementar, há a aplicação de mais dois filtros passa-baixas, um para o acelerômetro e outro para o magnetômetro. O filtro apenas suaviza as leituras da saída do conversor A/D, havendo a necessidade de mais iterações para atingir o valor instantâneo.

A partir da equação [62], e aplicando o fator “ACC_LPF_FACTOR”, tem-se para cada eixo do acelerômetro:

$$\text{accSmooth}[n] = \text{accSmooth}[\text{n} - 1] \times 0.99 + \text{accADC} \times 0.01 \quad [101]$$

A partir da equação [62], e aplicando o fator “MG_LPF_FACTOR”, tem-se para cada eixo do acelerômetro:

$$\text{mgSmooth}[n] = \text{mgSmooth}[n - 1] \times 0.75 + \text{magADC} \times 0.25 \quad [102]$$

Os macros “ACC_VALUE” e “MAG_VALUE” são criados, por fim, para fazer referência aos vetores com os valores filtrados “accSmooth” e “mgSmooth”, respectivamente.

Com a referência “ACC_VALUE”, se faz o cálculo da magnitude do vetor de aceleração:

$$accMag = 100 \times \frac{accSmooth_x{}^2 + accSmooth_y{}^2 + accSmooth_z{}^2}{acc\_1G^2} \quad [103]$$

No cálculo da magnitude do vetor de aceleração, a constante 100 apenas facilita a notação para futuras aplicações, a constante “acc_1G” equivale à leitura da

saída do conversor A/D do acelerômetro correspondente ao valor da aceleração gravitacional, que seria 255 no caso.

17.3.5.3 Matriz de Rotação

Com os valores em radianos dos ângulos de atitude no vetor “deltaGyroAngle”, calculados exclusivamente pelo integração dos valores do giroscópio, chama-se a função da matriz de rotação, “rotateV”, para cada uma das estruturas de vetores estimados.

getEstimatedAttitude - (chamada da função da matriz rotação)

```
//...

  rotateV(&EstG.V,deltaGyroAngle);
  rotateV(&EstM.V,deltaGyroAngle);

//...
```

rotateV – (função da matriz rotação)

```
void rotateV(struct fp_vector *v,float* delta) {
  fp_vector v_tmp = *v;

  v->Z -= delta[ROLL]  * v_tmp.X + delta[PITCH] * v_tmp.Y;
  v->X += delta[ROLL]  * v_tmp.Z + delta[YAW]   * v_tmp.Y;
  v->Y += delta[PITCH] * v_tmp.Z - delta[YAW]   * v_tmp.X;
}
```

Saindo do escopo da função “getEstimatedAttitude” e entrando no escopo da função “rotateV”, é necessário analisar primeiramente os parâmetros recebidos por essa função, onde “*v” aponta para a estrutura de vetores estimados “EstG.V” ou “EstM.V” e “*delta” recebe o vetor “deltaGyroAngle” (lembrando da característica de que vetores são ponteiros).

Fazendo uma simplificação por pequenos ângulos na equação [93], obtém-se:

$$R_B^A = \begin{bmatrix} 1-\varphi\psi\theta & \psi & \varphi+\psi\theta \\ -\psi-\varphi\theta & 1 & -\psi\varphi+\theta \\ -\varphi & -\theta & 1 \end{bmatrix} \quad [104]$$

A partir de [104] e analisando [94] e [95], chega-se a:

$$EstGVx[n] = EstGVx[n-1] + \Delta\varphi EstGVz[n-1] + \Delta\psi EstGVy[n-1] \quad [105]$$

$$EstGVy[n] = EstGVy[n-1] + \Delta\theta EstGVz[n-1] - \Delta\psi EstGVx[n-1] \quad [106]$$

$$EstGVz[n] = EstGVz[n-1] - \Delta\varphi EstGVx[n-1] - \Delta\theta EstGVy[n-1] \quad [107]$$

$$EstMVx[n] = EstMVx[n-1] + \Delta\varphi EstMVz[n-1] + \Delta\psi EstMVy[n-1] \quad [108]$$

$$EstMVy[n] = EstMVy[n-1] + \Delta\theta EstMVz[n-1] - \Delta\psi EstMVx[n-1] \quad [109]$$

$$EstMVz[n] = EstMVz[n-1] - \Delta\varphi EstMVx[n-1] - \Delta\theta EstMVy[n-1] \quad [110]$$

As equações [105] até [110] batem completamente com a lógica colocada no firmware, vista no último bloco de código acima.

17.3.5.4 Filtro Complementar

Para que o vetor que realiza a rotação, e, portanto, é estimado, obtenha as características de reação de outro sensor, junto ao giroscópio, é necessário aplicar a fusão destes pelo Filtro Complementar. A Figura 70 demonstra a aplicação de dois Filtros, um para o giroscópio e acelerômetro e outro para o giroscópio e magnetômetro. Para analisar a aplicação deste filtro, deve-se voltar ao corpo da função “getEstimatedAttitude”.

getEstimatedAttitude - (Filtro Complementar)

```
if ( ( 36 < accMag && accMag < 196 ) || f.SMALL_ANGLES_25 )
    for (axis = 0; axis < 3; axis++) {
      int16_t acc = ACC_VALUE;

      EstG.A[axis] = (EstG.A[axis] * GYR_CMPF_FACTOR + acc) * INV_G
YR_CMPF_FACTOR;

    }

  for (axis = 0; axis < 3; axis++)
    EstM.A[axis] = (EstM.A[axis] * GYR_CMPFM_FACTOR  + MAG_VALUE) *
 INV_GYR_CMPFM_FACTOR;
```

As estruturas “EstG” e “EstM” atualizam os valores das componentes do vetor representativo a partir de pesos definidos pelo programador. Estes são pesos

de relevância para cada sensor participante do filtro. Tendo a equação [70] para análise:

$$y_{cf}[n] = \ ay_{cf}[n-1] + (1-a)y_1[n] + a(y_2[n] - y_2[n-1])$$

$$y_{cf}[n] = \ a(y_{cf}[n-1] + y_2[n] - y_2[n-1]) + (1-a)y_1[n] \quad [111]$$

Para o Filtro Complementar (Giroscópio + Acelerômetro), tem-se a atualização das componentes de cada eixo do vetor por:

$$EstG.A = \left(EstG.A \times \mathrm{GYR}_{\mathrm{CMPF}_{\mathrm{FACTOR}}} + \mathrm{ACC}_{\mathrm{VALUE}}\right) \times \mathrm{INV}_{\mathrm{GYR}_{\mathrm{CMPF}_{\mathrm{FACTOR}}}} \quad [112]$$

$$EstG.A = \frac{(EstG.A \times 400 + \ \mathrm{ACC}_{\mathrm{VALUE}})}{401} \quad [113]$$

$$EstG.A = \ EstG.A \times 0.9975 + \ \mathrm{ACC}_{\mathrm{VALUE}} \times 0.0025 \quad [114]$$

Comparando a equação [114] com a [111] encontram-se as seguintes relações:

$$(1-a)y_1[n] = \ \mathrm{ACC}_{\mathrm{VALUE}} \times 0.0025 \quad [115]$$

$$a\left(y_{cf}[n-1] + y_2[n] - y_2[n-1]\right) = \ EstG.A \times 0.9975 \quad [116]$$

Sendo $EstG.A$ derivado diretamente das equações [105] a [107], que contém implícito as duas medidas de y_2 a partir da variação dos ângulos calculada pelo giroscópio.

Para o Filtro Complementar (Giroscópio + Magnetômetro), tem-se a atualização das componentes de cada eixo do vetor por:

$$EstM.A = (EstM.A \times \mathrm{GYR}_{\mathrm{CMPFM}_{\mathrm{FACTOR}}} + \mathrm{MAG}_{\mathrm{VALUE}}) \times \mathrm{INV}_{\mathrm{GYR}_{\mathrm{CMPFM}_{\mathrm{FACTOR}}}} \quad [117]$$

$$EstM.A = \frac{(EstM.A \times 200 + \ \mathrm{MAG}_{\mathrm{VALUE}})}{201} \quad [118]$$

$$EstM.A = EstM.A \times 0.995 + \mathrm{MAG_{VALUE}} \times 0.005 \quad [119]$$

Comparando com a equação [119] com a [111] encontram-se as seguintes relações:

$$(1-a)y_1[n] = \mathrm{MAG_{VALUE}} \times 0.005 \quad [120]$$

$$a\big(y_{cf}[n-1] + y_2[n] - y_2[n-1]\big) = EstM.A \times 0.995 \quad [121]$$

Sendo $EstM.A$ derivado diretamente das equações [108] a [110], que contém implícito as duas medidas de y_2 a partir da variação dos ângulos calculada pelo giroscópio.

Quanto maior "$\mathrm{GYR_{CMPF_{FACTOR}}}$" e "$\mathrm{GYR_{CMPFM_{FACTOR}}}$" menor a influência do acelerômetro e magnetômetro, respectivamente, em relação ao giroscópio, ou seja, o giroscópio tem maior influência na estimação de atitude.

Para o filtro (Giroscópio + Acelerômetro) há um adendo importante. Há apenas a implementação do filtro, e consequentemente a influência do acelerômetro na estimativa de atitude, se o quadrirrotor se encontrar com inclinação menor de 24.84˚ (flag SMALL_ANGLES_25 ativa) ou se a amplitude do vetor resultante do acelerômetro, “accMag”, for maior que 36 (0.6g) e menor que 196 (1.4g), caso contrário, toda a estimativa é por conta do giroscópio. Isso é feito para retirar o efeito do acelerômetro da estimação quando o quadrirrotor estiver em uma velocidade relativamente alta.

17.3.5.5 Cálculo de atitude.

Com o Filtro Complementar realizado, o sistema está apto a calcular os ângulos de rolagem, arfagem e guinada através da chamada da função “_atan2”

getEstimatedAttitude - (obtenção dos ängulos de atitude)

```
//...
  angle[ROLL]  =  _atan2(EstG.V.X , EstG.V.Z) ;
  angle[PITCH] =  _atan2(EstG.V.Y , EstG.V.Z) ;

  heading = _atan2( EstG.V.X * EstM.V.Z - EstG.V.Z * EstM.V.X , Est
G.V.Z * EstM.V.Y - EstG.V.Y * EstM.V.Z  );
  heading += MAG_DECLINIATION * 10;

  heading = heading /10;
  if ( heading > 180)      heading = heading - 360;
  else if (heading < -180) heading = heading + 360;
    }
```

Para o ângulo de rolagem e arfagem, respectivamente, se faz:

$$\varphi = \_atan2\left(\frac{EstG.V.X}{EstG.V.Z}\right) \quad [122]$$

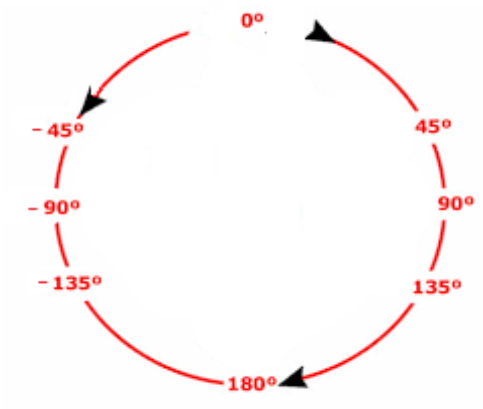
$$\theta = \_atan2\left(\frac{EstG.V.Y}{EstG.V.Z}\right) \quad [123]$$

Para o ângulo de guinada, deve-se projetar as componentes do vetor estimado de sentido magnético para o plano horizontal (X-Y), e então aplicar o resultado na função "_atan2":

$$\psi = \_atan2\left(\frac{(EstG.V.X + EstG.V.Z) \times (EstM.V.X + EstM.V.Z)}{(EstG.V.Y + EstG.V.Z) \times (EstM.V.Y + EstM.V.Z)}\right) \quad [124]$$

Soma-se ao ângulo de guinada á declinação magnética da região de operação para ter orientação do norte verdadeiro. Por fim, basta tratar os valores dentro de um arco de 360˚, conforme a Figura 71.

Figura 71 - Valor para cada orientação de guinada.



Fonte: Autoria Própria.

18.MAIN.INO

O código principal é onde acontece a chamada de todas outras funções que já foram apresentadas até aqui. A apresentação deste bloco será dividia em 4 partes. A primeira consiste na apresentação de todas as declarações de variáveis globais e macros, indicando a função de cada. A segunda terá a apresentação da dinâmica do firmware “MultiWii” através de um diagrama. A terceira parte terá a explicação da função “annexCode”, uma função que é executada no loop principal através da chamada de outra função, “computeIMU”. A última parte consiste na explicação da lógica criada para a interação do usuário, através do rádio transmissor, com o objetivo de acessar as funções presentes no quadrirrotor, tais como calibração e modos de voo, levando em conta a configuração prévia feita por ele mesmo no software de controle em solo.

18.1 Declarações

A Tabela 32 apresenta macros e variáveis globais, usados ao decorrer de todo o firmware, sendo que muitos já foram introduzidos. A Tabela divide as variáveis e macros por setores de operação. Há variáveis e macros destacados em vermelho, estes são pertencentes à funções e modos retirados na modificação do firmware, porém para um entendimento panorâmico, é importante citá-los.

Tabela 32 - Macros e Variáveis

MACROS CANAIS DO RÁDIO CONTROLE		
ROLL	0	Referência à rolagem
PITCH	1	Referência à arfagem
YAW	2	Referência à guinada
THROTTLE	3	Referência à potência
AUX1	4	Canal auxiliar 1
AUX2	5	Canal auxiliar 2
AUX3	6	Canal auxiliar 3
AUX4	7	Canal auxiliar 4

MACROS DO PID [PIDITEMS]		
PIDALT	3	Referência aos ganhos do controle de ALTITUTE
PIDPOS	4	Referência para os ganhos do controle de GPS
PIDPOSR	5	Referência para os ganhos do controle de POSITION HOLD
PIDNAVR	6	Referência para ganhos do controle de NAVEGAÇÃO
PIDLEVEL	7	Referência para os ganhos do controle de LEVEL MODE
PIDMAG	8	Referência para os ganhos do controle de MAG MODE
PIDVEL	9	Referência aos ganhos do controle de VELOCIDADE
CAIXAS DA INTERFACE GRÁFICA DO USUÁRIIO (GUI)		
BOXACC	0	Referência de configuração para ativar/desativar LEVEL MODE
BOXBARO	1	Referência de configuração para ativar/desativar ALTITUTE HOLD MODE
BOXMAG	2	Referência de configuração para ativar/desativar MAG MODE
BOXCAMSTAB	3	Referência de configuração para ativar/desativar estabilização da câmera por servo.
BOXCAMTRIG	4	Referência de configuração para ativar/desativar gatilho da câmera
BOXARM	5	Referência de configuração para armar e desarmar quadrirrotor
BOXGPSHOME	6	Referência de configuração para ativar/desativar RETURN TO HOME

BOXGPSHOLD	7	Referência de configuração para ativar/desativar HOLD POSITION
BOXPASSTHRU	8	Referência de configuração para ativar/desativar PASSTHRU MODE
BOXHEADFREE	9	Referência de configuração para ativar/desativar HEAD FREE MODE
BOXBEEPERON	10	Referência de configuração para ativar/desativar buzzer de acordo com a bateria
BOXLEDMAX	11	Referência de configuração para ativar luzes no máximo
BOXLIGHTS	12	Referência de configuração para ativar/desativar luzes de pouso
BOXHEADADJ	13	Referência de configuração para selecionar nova perspectiva para HEAD FREE MODE
PIDITEMS	10	Número de modos de controles diferentes
CHECKBOXITEMS	14	Número de configurações selecionáveis pela GUI possíveis
DECLARAÇÃO DE VARIÁVEIS GLOBAIS GERAIS		
currentTime	0	Tempo decorrido no loop principal
previousTime	0	Tempo do loop principal anterior
cycletime	0	Tempo de um loop principal completo
calibratingA	0	Variável de calibração do acelerômetro. Se seu valor é diferente de 0, se encontra em estado de calibração.
calibratingG	-	Variável de calibração do giroscópio. Se seu valor é

		diferente de 0, se encontra em estado de calibração.
acc_1G	255	Valor da saída do conversor A/D do acelerômetro equivalente à 1g
acc_25deg	-	Valor da saída do conversor A/D do acelerômetro equivalente à 0.42g, ou 25˚ em relação ao eixo z, se aplicado nos eixos x ou y
headFreeModeHold	-	Ângulo de guinada fixo para orientação de voo fixa
gyroADC[3]	-	Valores de saída do conversor A/D do giroscópio para os 3 eixos
accADC[3]	-	Valores de saída do conversor A/D do acelerômetro para os 3 eixos
accSmooth[3]	-	Valores dos 3 eixos do acelerômetro pós filtro passa-baixa
magADC[3]	-	Valores dos 3 eixos do magnetômetro pós filtro passa-baixa
heading	-	Ângulo de guinada
magHold	-	Referência de ângulo de guinada para o controle no MAG MODE
rcOptions[CHECKBOXITEMS]	-	Indicador se certa opção, seletiva pelo rádio controle, está ativa ou não
BaroAlt	-	Altitude dada pelo barômetro (cm).
EstAlt	-	Altitude estimada
BaroPID	0	Sinal de controle para controle de altitude
AltHold	-	Altitude selecionada como referência do controle de altitude
errorAltitudeI	0	Integrador da malha de controle de altitude

Debug[4]	-	Varíavel para protocolo de debug.
FLAGS DE STATUS (ESTRUTURA f)		
OK_TO_ARM	1	1: pronto para armar 0: não é possível armar
ARMED	1	1: armado 0: desarmado
I2C_INIT_DONE	1	1: inicialização da i2c completa
ACC_CALIBRATED	1	0: inicialização da i2c incompleta
ACC_MODE	1	1: LEVEL MODE ativado 0: LEVEL MODE desativado
MAG_MODE	1	1: MAG MODE ativado 0: MAG MODE desativado
BARO_MODE	1	1: BARO MODE ativado 0: BARO MODE desativado
GPS_HOME_MODE	1	1: RETURN TO HOME ativado 0: RETURN TO HOME desativado
GPS_HOLD_MODE	1	1: HOLD POSITION ativado 0: HOLD POSITION desativado
HEADFREE_MODE	1	1: HEADFREE MODE ativado 0: HEADFREE MODE desativado
PASSTHRU_MODE	1	1: PASSTHRU MODE ativado 0: PASSTHRU MODE desativado
SMALL_ANGLES_25	1	1: angulação menor que 25˚ 0: angulação maior que 25˚
CALIBRATE_MAG	1	1: calibração do magnetômetro necessária 0: calibração do magnetômetro realizada
CONTADORES GLOBAIS		
I2c_errors_count	0	Número de erros na comunicação I2C
Annex650_overrun_count	0	Vezes em que a função “annexCode” superou 650ms

MACROS FUNÇÕES DO RECEPTOR		
MINCHECK	1100	Valo máximo em que o canal está em estado de MINCHECK
MAXCHECK	1900	Valor mínimo em que o canal está em estado de MAXCHECK
VARIÁVEIS FUNÇÕES DO RECEPTOR		
rcData[8]	-	Média dos valores de cada canal do receptor
rcCommand[4]	-	Curvas de comando dos canais de rolagem, arfagem, guinada e potência após função “annexCode”
lookupPitchRollRC[6]	-	6 pontos para criação das curvas de comando de atitude
lookupThrottleRC[11]	-	11 pontos para criação das curvas de comando de potência
VARIÁVEIS IMU GYRO+ACC		
gyroData[3]	{0,0,0}	Média entre leituras consecutivas do giroscópio
gyroZero[3]	{0,0,0}	Valores de calibração do giroscópio
angle[2]	{0,0}	Ângulos {rolagem, arfagem}
VARIÁVEIS FUNÇÕES DOS MOTORES		
axisPID[3]	-	Sinal de controle {rolagem, arfagem, guinada}
motor[NUMBER_MOTOR]	-	Motores do quadrirrotor [4]
VARIÁVEIS DA EEPROM		
dynP8[3]	-	Ganho dinâmico P
dynD8[3]	-	Ganho dinâmico D
ESTRUTURA DE CONFIGURAÇÃO DA EEPORM (conf)		
checkNewConf	-	Checa se houve configuração diferente ou não da inicial na EEPROM
P8[PIDITEMS]	-	Ganho proporcional

I8[PIDITEMS]	-	Ganho integrativo
D8[PIDITEMS]	-	Ganho derivativo
rcRate8	-	Variável para configuração das curvas de comando dos canais de rolagem, arfagem e guinada
rcExpo8	-	Variável para configuração das curvas de comando dos canais de rolagem, arfagem e guinada
rollPitchRate	-	Taxa para atenuar ação de controle nos movimentos de rolagem e arfagem
yawRate	-	Taxa para atenuar ação de controle no movimento de guinada
dynThrPID	-	Taxa para atenuar o ganho proporcional e derivativo dos movimentos de rolagem e arfagem de acordo com a potência.
thrMid8	-	Variável para configuração das curvas de comando do canal de potência
thrExpo8	-	Variável para configuração das curvas de comando do canal de potência
accZero[3]	-	Valores de calibração do acelerômetro
magZero[3]	-	Valores de calibração do magnetômetro
angleTrim[2]	-	Ajuste fino á angulação do quadrirrotor {rolagem, arfagem}
activate[CHECKBOXITEMS]	-	Configuração das opções selecionáveis pelo rádio controle (software envia essa variável, e

		seu valor muda de acordo com a configuração)

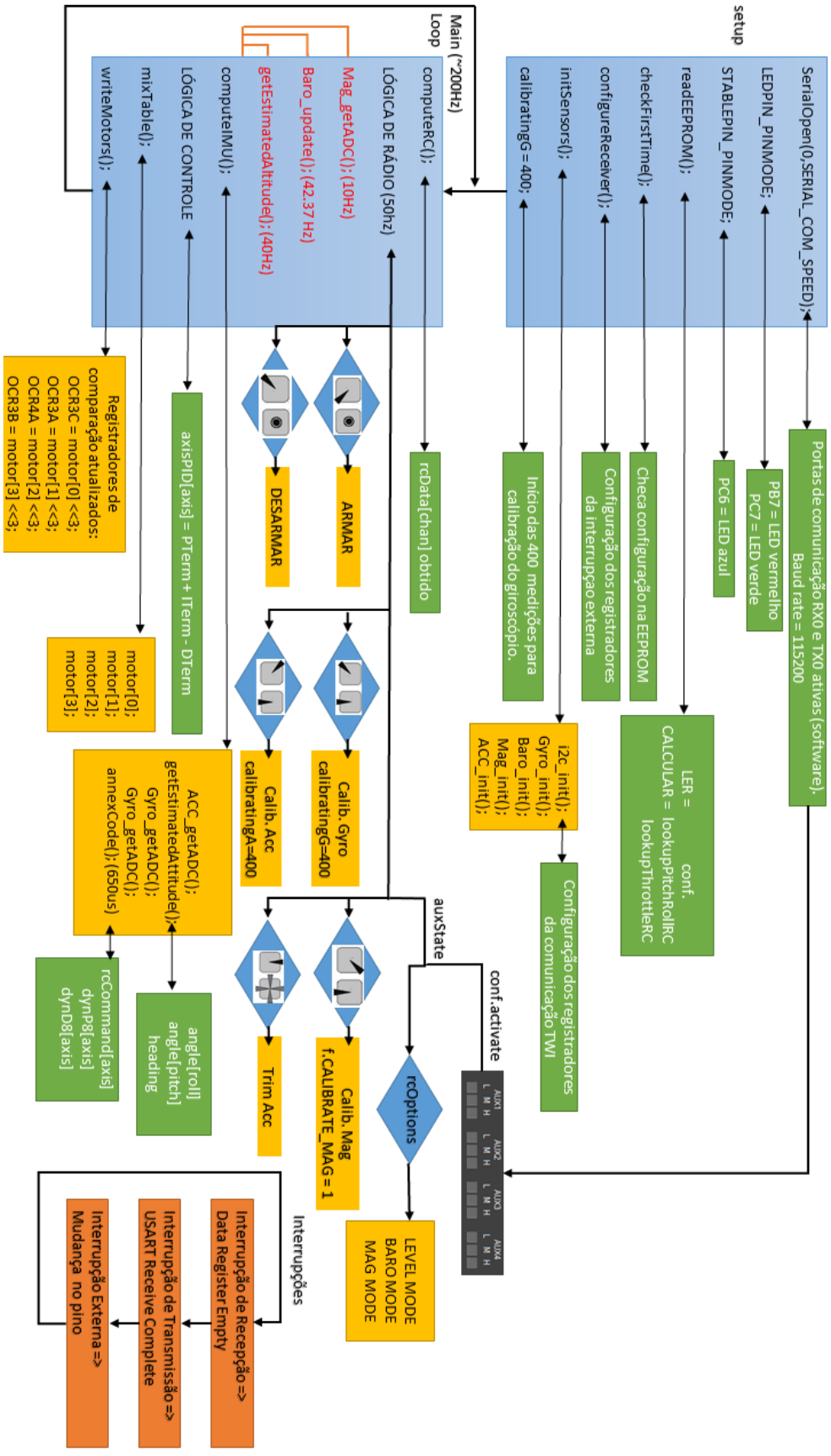
Fonte: Autoria Própria

18.2 Diagrama

Como pode ser visto no desenvolvimento do estudo do firmware, tem-se diversas funções com particularidades próprias. A integração destas no código principal pode ser algo complicado de se apresentar. Pensando nisso, foi desenvolvido o diagrama da Figura 72.

Este diagrama contém a fase de configuração e inicialização (setup), seguida do loop principal. São destacáveis várias funções estudadas nas seções anteriores, como a inicialização, calibração e captura dos valores dos sensores, configuração da EEPROM e configuração dos registradores, tanto os de interrupção, quanto os responsáveis pela geração do pulso PWM. A função “annexCode” e a lógica de rádio serão apresentadas na sequência desse diagrama. A identificação e lógica dos controles presentes no sistema serão apresentadas no capítulo posterior a este, onde será introduzida toda a teoria de controle, junto com abordagens auxiliares para estabilização.

Figura 72 - Dinâmica do Firmware



Fonte: Autoria Própria

18.3 Annex Code

A função “annexCode” é responsável por criar as funções de entrada para a malha de controle, levando em consideração os movimentos de rolagem, arfagem e guinada acionados por um canal específico do sistema rádio transmissor/receptor. Outra função desta, é utilizar os coeficientes de atenuação dos parâmetros de controle.

Visando melhor estrutura e didática de cada funcionalidade da função, esta será dividida em partes, e sua ordem será reestruturada em relação ao código em uso.

18.3.1 Geração dos valores de entrada para a malha de controle

Através do rádio transmissor é feito uma transmissão em rádio frequência para o quadrirrotor, contendo dados referentes a posição dos sticks. No quadrirrotor, o receptor faz a recepção dos dados e transmite para a controladora um sinal de cada canal do rádio transmissor. Este sinal é um pulso PWM com duração de 1000µs para um stick em seu extremo menor e 2000µs para um sitck em seu extremo maior. A controladora faz a aquisição e processa o controle remoto de atitude, altitude e funções de voo.

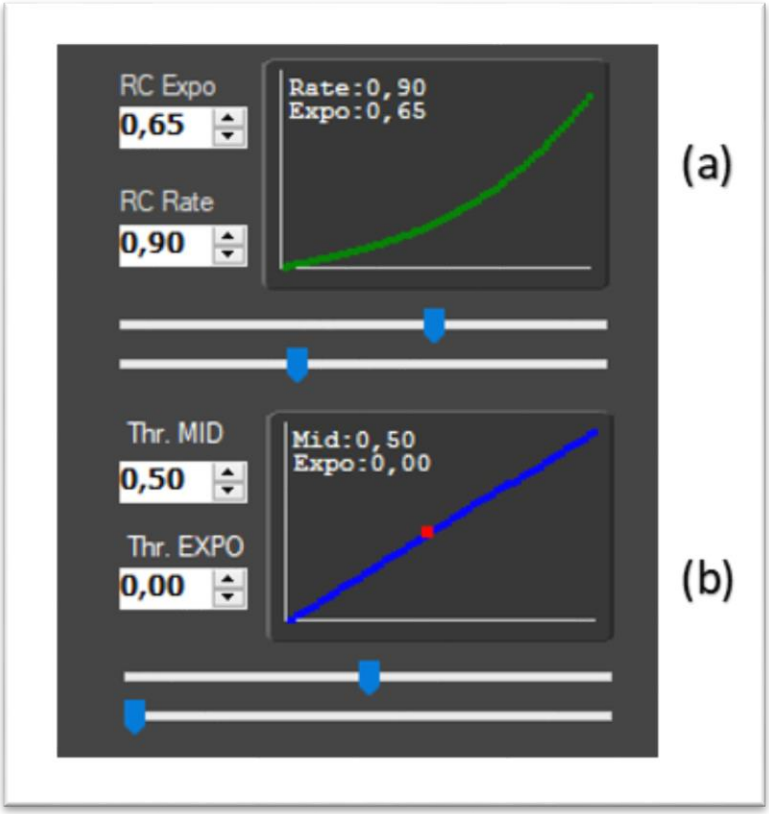
A Figura 73 demonstra o software de controle em solo fazendo a leitura dos valores de pulso de cada canal do receptor.

Ainda no GCS, é possível configurar a curva de evolução dos valores de cada canal do receptor. A Figura 74 (a) apresenta duas variáveis responsáveis por modificar a curva das ações de rolagem, arfagem e guinada, “RC Expo” e “RC Rate”. A Figura 74 (b), com as variáveis “Thr.MID” e “Thr.EXPO” modificando a curva de potência dos motores.

Figura 73 - Leitura dos canais do receptor através do software de controle em solo.

Fonte: Autoria própria.

Figura 74 – Modificação nas curvas de evolução dos canais do receptor. (a) ROLL, PITCH, YAW. (b) THROTTLE.



Fonte: Autoria própria.

A justificativa para a alteração das curvas advindas de cada canal do receptor é o tratamento de dados para a aplicação na malha de controle. Como será visto na identificação dos controles, a entrada dos valores dos canais de rolagem, arfagem e guinada na malha de controle do sistema exerce a função de referência (setpoint). Com a possibilidade de alterar a evolução dessas curvas, o usuário pode configurar uma dinâmica de voo do seu interesse, por exemplo, sendo mais suave em certo intervalo do stick e mais agressivo quando enviar sinais extremos de respectivo canal.

Para analisar mais a fundo como esse tratamento de dados é feito, é valido analisar a função “annexCode” linha a linha, e apresentá-la de forma interativa. É dado início, analisando a geração da curva modificável apresentada na Figura 74 (a).

annexCode (ROLL, PITCH, YAW)

```
void annexCode() {
  static uint32_t calibratedAccTime;
  uint16_t tmp,tmp2;
  uint8_t axis,prop1,prop2;

   for(axis=0;axis<3;axis++) {
    tmp = min(abs(rcData[axis]-MIDRC),500);
    #if defined(DEADBAND)
      if (tmp>DEADBAND) { tmp -= DEADBAND; }
      else { tmp=0; }
    #endif

    if(axis!=2) { //ROLL & PITCH
      tmp2 = tmp/100;
      rcCommand[axis] = lookupPitchRollRC[tmp2] + (tmp-
tmp2*100) * (lookupPitchRollRC[tmp2+1]-
lookupPitchRollRC[tmp2]) / 100;
    } else {      // YAW
      rcCommand[axis] = tmp;
    }
    if (rcData[axis]<MIDRC) rcCommand[axis] = -rcCommand[axis];
  }
  //...
```

Depois da declaração das variáveis que serão utilizadas na função, é chamado um loop “for” a fim de referenciar os três eixos de movimento, roll, pitch e yaw, que terão suas curvas dos canais respectivos alteradas.

Primeiro é mapeado a intensidade da posição do stick do rádio controle, isso é feito a partir do valor do respectivo canal por meio da variável “rcData”, já estudada na seção do receptor. Essa variável é subtraída pelo valor que indica a posição central do stick ($1500\mu s$) e passada por um módulo, limitando então em até

500μs do pulso recebido, ou seja, se o valor do canal lido é 1700 μs, a intensidade armazenada em “tmp” será 200μs, se é 1200μs, a intensidade armazenada é de 300μs. Tem-se ainda a atualização da variável “tmp2”, que é a variável “tmp” na escala de um centésimo.

No bloco de código de configuração, é possível adicionar uma zona morta (DEADBAND) na posição central do stick, se feito isso, a variável “tmp”, terá seu valor alterado, apenas se sua intensidade for maior que a da zona morta.

Seguindo o código, para os canais de rolagem e arfagem, tem-se a chamada do vetor “lookupPitchRollRC”, declarado no bloco de código da EEPROM dentro da função “readEEPROM”. O vetor recebe como posição o valor da variável “tmp2”.

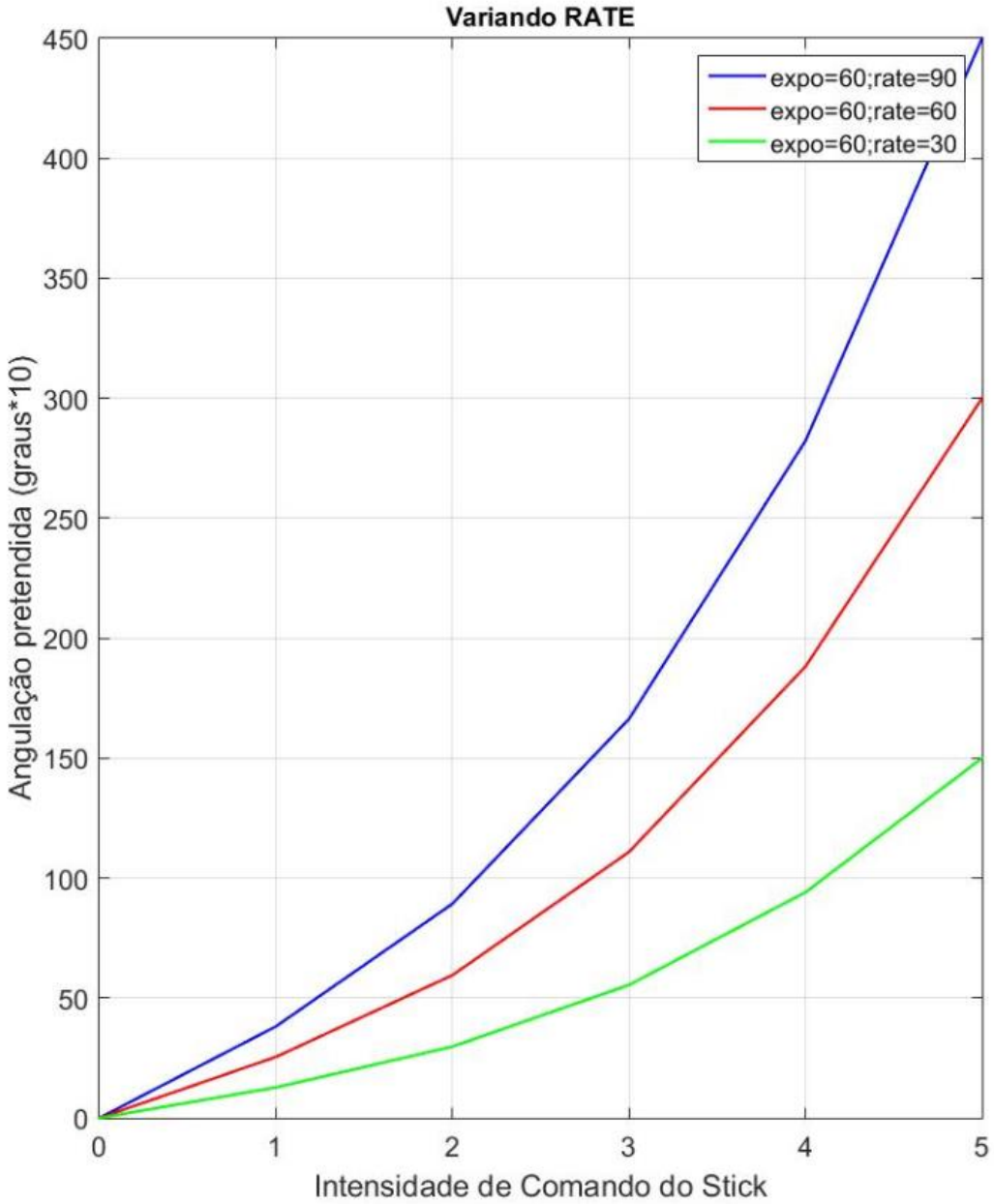
readEEPROM

```
void readEEPROM() {
  uint8_t i;

  eeprom_read_block((void*)&conf, (void*)0, sizeof(conf));

  for(i=0;i<6;i++) {
    lookupPitchRollRC[i] = (2500+conf.rcExpo8*(i*i-
25))*i*(int32_t)conf.rcRate8/2500;
  }

  for(i=0;i<11;i++) {
    int16_t tmp = 10*i-conf.thrMid8;
    uint8_t y = 1;
    if (tmp>0) y = 100-conf.thrMid8;
    if (tmp<0) y = conf.thrMid8;
    lookupThrottleRC[i] = 10*conf.thrMid8 + tmp*( 100-
conf.thrExpo8+(int32_t)conf.thrExpo8*(tmp*tmp)/(y*y) )/10;

    lookupThrottleRC[i] = MINTHROTTLE + (int32_t)(MAXTHROTTLE-
MINTHROTTLE)* lookupThrottleRC[i]/1000;
  }
```

Analisando o vetor “lookupPitchRollRC” e a aquisição dos dados pela variável “rcCommand” na função “annexCode”, com o auxílio do *MatLab* plota-se os gráficos referente ás curvas modificadas (“rcCommand”) de roll e pitch, variando “RC Expo” e “RC Rate”, a fim de verificar os impactos de cada variável na curva. A Figura 75 demonstra a curva para as ações de rolagem e arfagem, variando-se o “RC Rate”.A

Figura 76[13] demonstra a curva para as ações de rolagem e arfagem, variando-se o “RC Expo”.

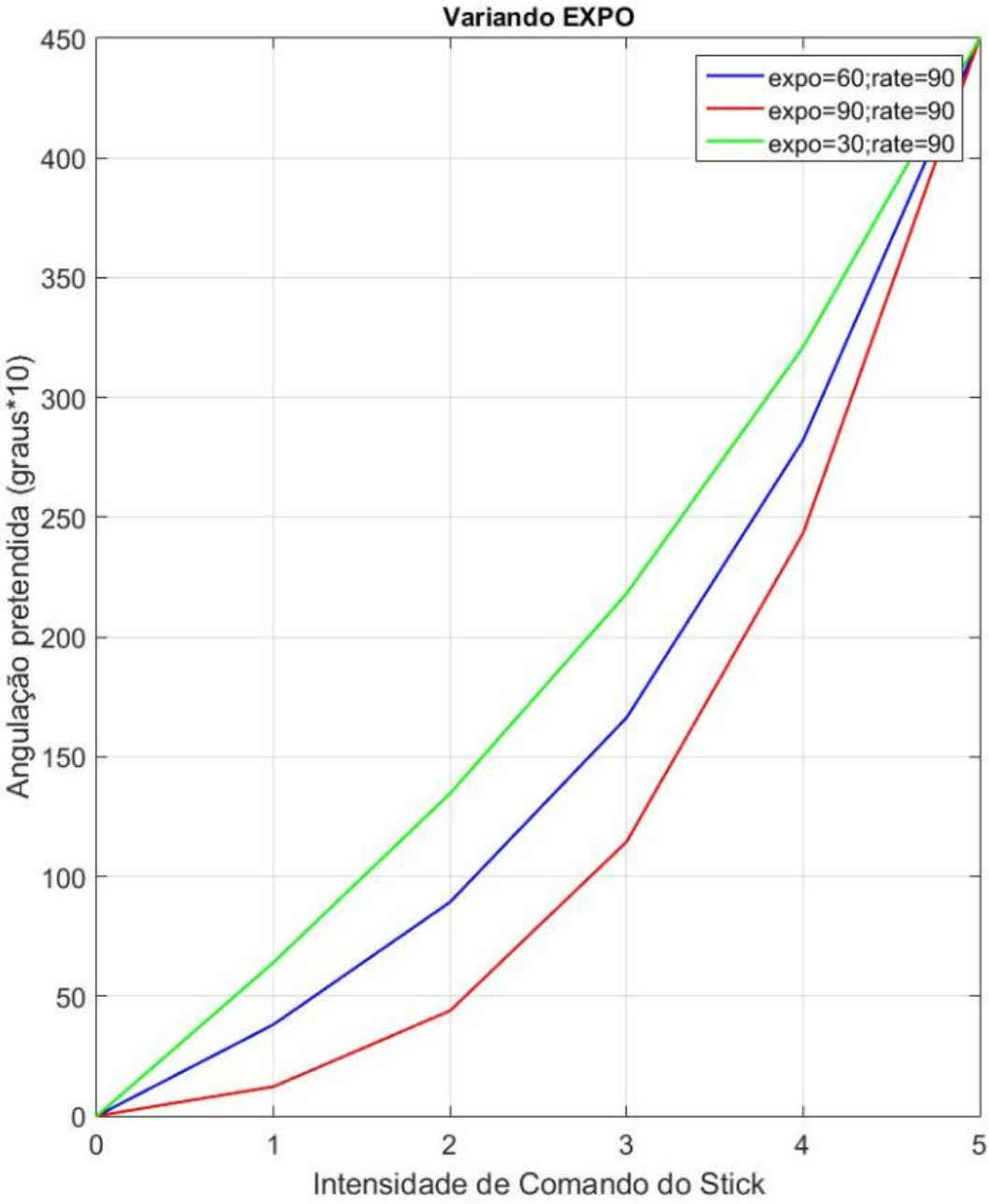
Figura 75 – Curva para as ações de rolagem e arfagem, variando-se o “RC Rate”



Fonte: Autoria própria.

[13] É importante destacar que as varáveis “RC RATE” e “RX EXPO” possuem valores diferentes de exibição na GUI e na EEPROM. Se o software apresenta um valor de 0,50, este tem equivalência de 50 na EEPROM. O limite máximo de cada variável seria 1,00 na apresentação no software.

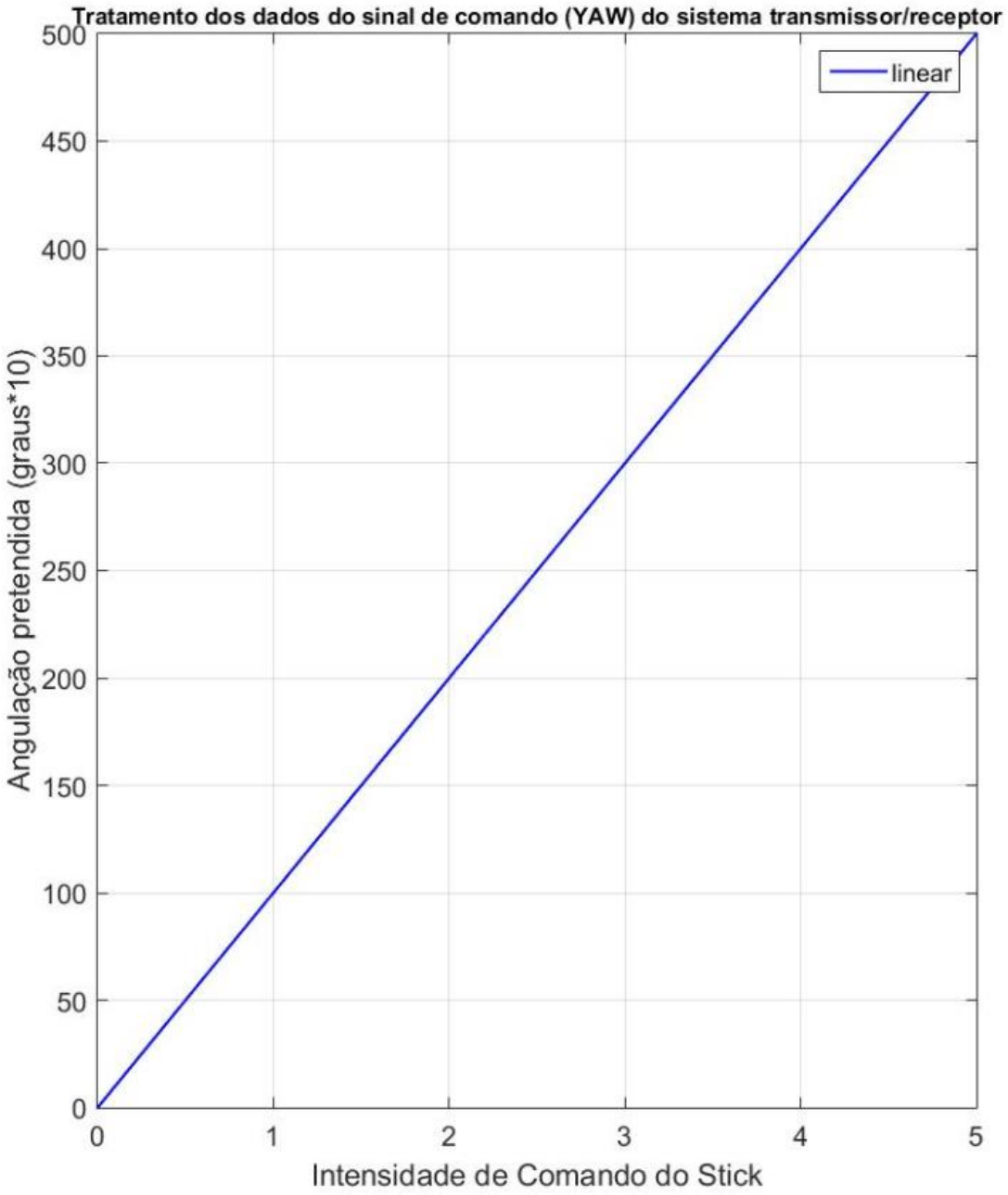
Figura 76 – Curva para as ações de rolagem e arfagem, variando-se o “RC Expo”.



Fonte: Autoria própria.

A Figura 77 demonstra a curva para a ação de guinada. Como pode-se ver, esta não sofre alteração pelo vetor “lookupPitchRollRC”, e a variável “rcCommand” recebe o valor direto da intensidade da posição do stick do rádio controle (“tmp”)

Figura 77 – Curva para a ação de guinada.



Fonte: Autoria própria.

Como pode-se observar, a curva de setpoint criada (“rcCommand”) possui um retorno com amplitude 10x maior que o ângulo real. Isso se deve porque a função “atan2”, vista no estudo da IMU, retorna os valores de ângulo com a mesma amplitude (10x). Essa escolha é dada para se evitar variáveis flutuantes e não comprometer a eficiência da malha de controle de acordo com o microprocessador presente.

As curvas de atitude, dadas pela variável “rcCommand” possuem 6 posições calculáveis pelo vetor “lookupPitchRollRC”. A ligação entre essas posições é de maneira linear.

Ainda é valido mencionar, que para valores negativos de intensidade (<1500μs), têm-se respostas de funções ímpares em relação ás Figuras 75, 76 e 77.

Passando a análise para a geração da curva modificável apresentada na Figura 74 (b), tem-se a configuração da curva do canal de potência dos motores.

annexCode (THROTTLE)

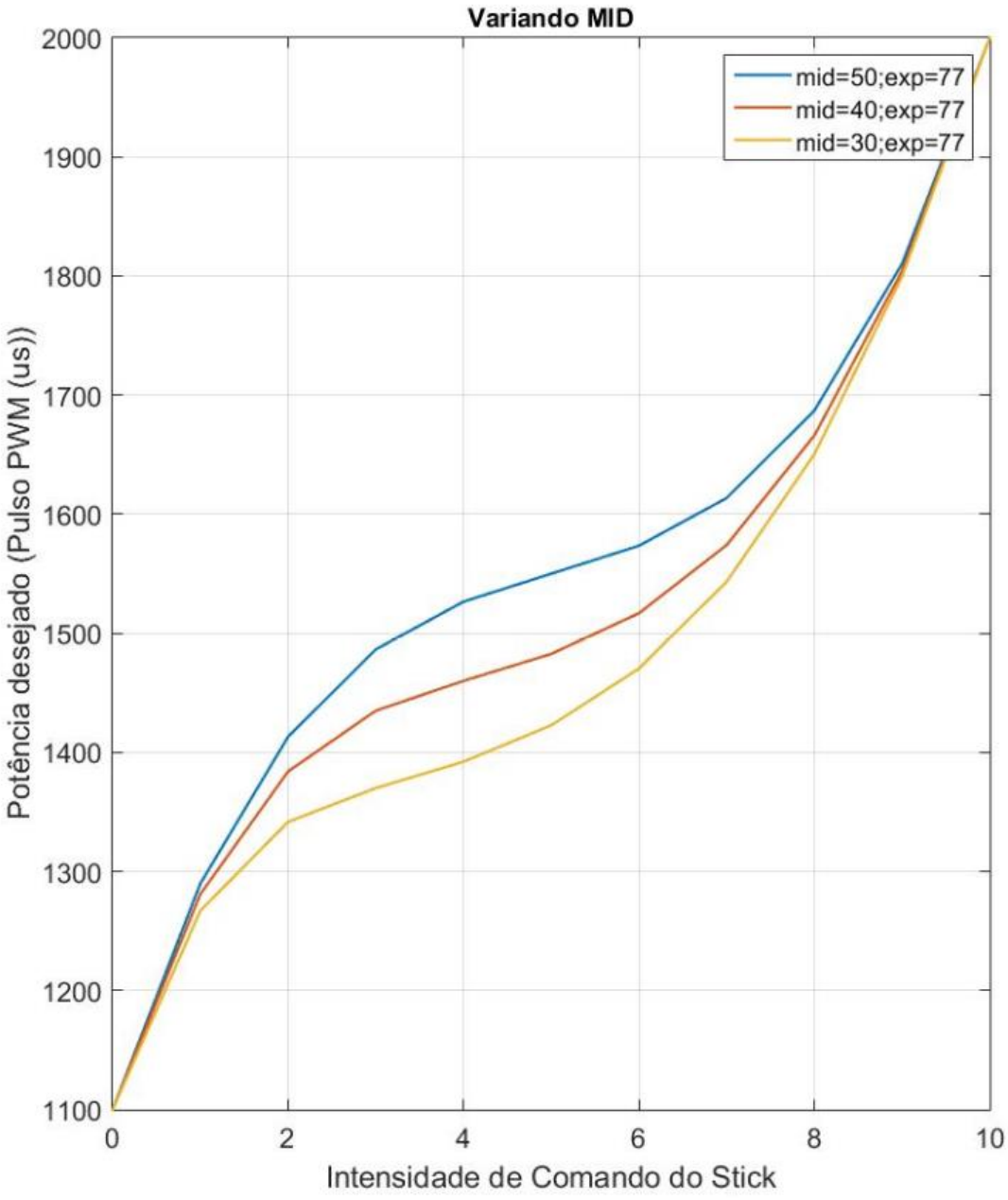
```
tmp = constrain(rcData[THROTTLE],MINCHECK,2000);
tmp = (uint32_t)(tmp-MINCHECK)*1000/(2000-MINCHECK);
tmp2 = tmp/100;

rcCommand[THROTTLE] = lookupThrottleRC[tmp2] + (tmp-
tmp2*100) * (lookupThrottleRC[tmp2+1]-
lookupThrottleRC[tmp2]) / 100;
```

Analisando o código, é possível verificar que a variável “tmp” ainda exerce a função de capturar a intensidade do stick do rádio controle, porém aqui ocorre de maneira diferente. Primeiro, a partir da variável “rcData” tem-se a limitação entre [MINCHECK, 2000]. “MINCKECK” é a posição do stick, que considera o canal em ponto mínimo, podendo realizar inúmeras outras operações. Depois dessa limitação, a variável “tmp” muda seu intervalo de análise linearmente de [MINCHECK, 2000] para [0, 1000]. A variável “tmp2” recebe o intervalo de “tmp” na escala de um centésimo, tendo intervalo entre [0, 10]. A variável “tmp2” é a posição que será passada para o vetor “lookupThrottleRC”, que também se encontra dentro da função “readEEPROM”.

Analisando o vetor “lookupThrottleRC” e a aquisição dos dados pela variável “rcCommand” na função “annexCode”, com o auxílio do *MatLab* plota-se os gráficos referente ás curvas modificadas (“rcCommand”) de potência dos motores, variando “Thr.MID” e “Thr.EXPO”, a fim de verificar os impactos de cada variável na curva. A Figura 78[14] demonstra a curva, variando-se o “Thr.MID”. A Figura 79 demonstra a curva, variando-se o “Thr.EXPO”.

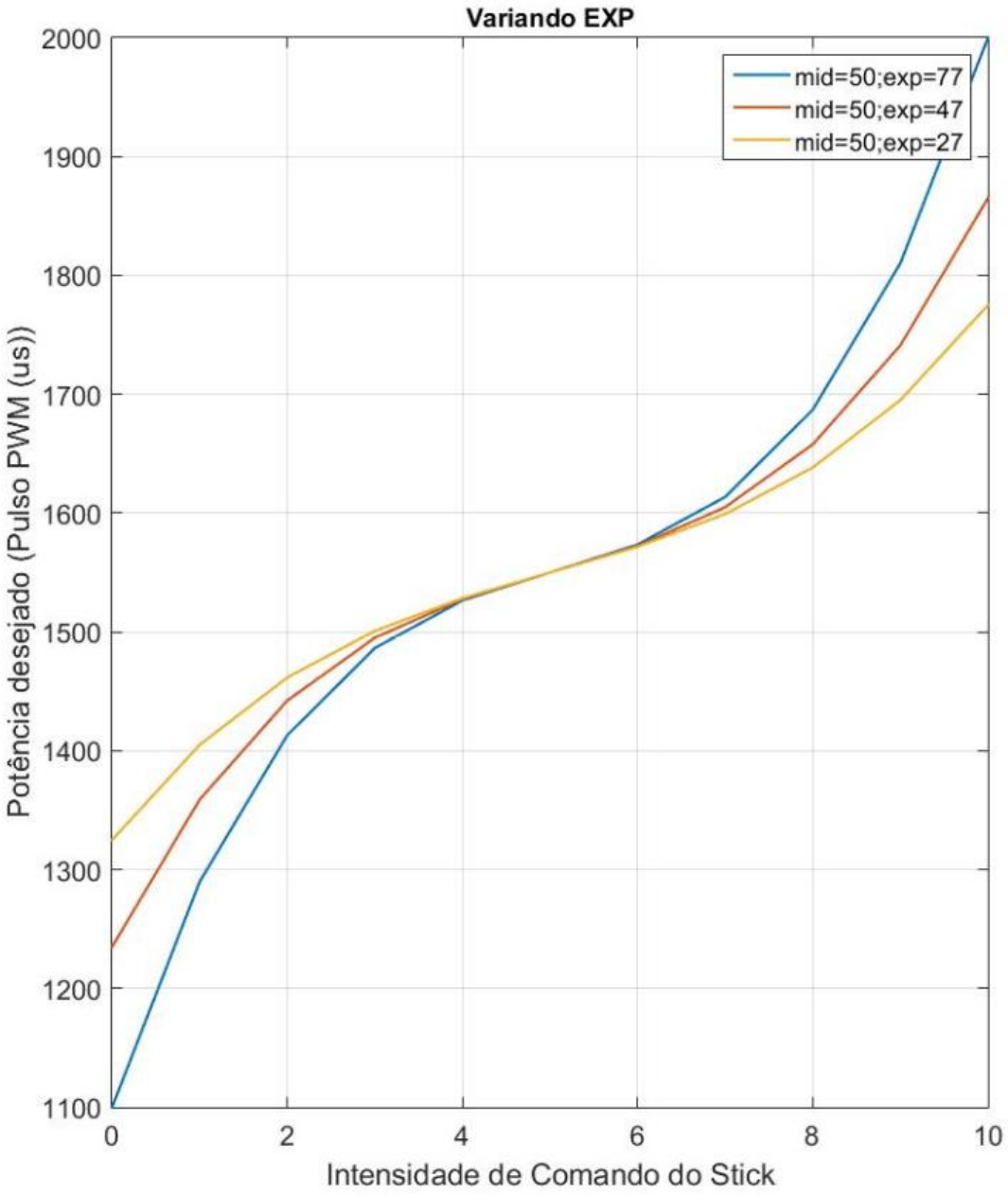
[14] É importante destacar que as varáveis “Thr.MID” e “Thr.EXPO” possuem valores diferentes de exibição na GUI e na EEPROM. Se o software apresenta um valor de 0,50, este tem equivalência de 50 na EEPROM. O limite máximo de cada variável seria 1,00 na apresentação no software.

Figura 78 – Curva para a potência do motor.



Fonte: Autoria própria.

Figura 79– Curva para a potência do motor.



Fonte: Autoria própria.

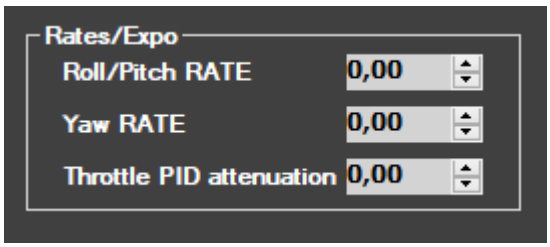
A curva de potência do motor, dada pela variável “rcCommand” possuem 11 posições calculáveis pelo vetor “lookupThrottleRC”. A ligação entre essas posições é de maneira linear.

Todos os algoritmos gerados no Matlab estão disponíveis no APÊNDICE D

18.3.2 Ajuste dinâmico dos ganhos do controle PID.

É possível ajustar dois ganhos da malha de controle do sistema, de acordo com os valores do stick. Sendo ganhos dinâmicos ao decorrer do funcionamento do quadrirrotor. A Figura 80 demonstra o local onde este ajuste de ganho dinâmico pode ser configurado no software de controle em solo.

Figura 80 – Valores para ajuste de ganho dinâmico



Fonte: Autoria própria.

Como foi dito, para melhor estrutura e didática de cada funcionalidade da função, esta foi dividida em partes. O trecho de código abaixo, pode ser muito bem encaixado em outros trechos de código da função “annexCode” já apresentados.

annexCode (Ajuste Dinâmico)

```
//BREAKPOINT = 1500;
if   (rcData[THROTTLE]<BREAKPOINT) {
    prop2 = 100;
  } else {
    if (rcData[THROTTLE]<2000) {
      prop2 = 100 - (uint16_t)conf.dynThrPID*(rcData[THROTTLE]-
BREAKPOINT)/(2000-BREAKPOINT);
    } else {
      prop2 = 100 - conf.dynThrPID;
    }
  }
    for(axis=0;axis<3;axis++) {
    tmp = min(abs(rcData[axis]-MIDRC),500);

    if(axis!=2) { //ROLL & PITCH
      prop1 = 100-(uint16_t)conf.rollPitchRate*tmp/500;
      prop1 = (uint16_t)prop1*prop2/100;
    } else {      // YAW
      prop1 = 100-(uint16_t)conf.yawRate*tmp/500;
    }
    dynP8[axis] = (uint16_t)conf.P8[axis]*prop1/100;
    dynD8[axis] = (uint16_t)conf.D8[axis]*prop1/100;
  }
```

Analisando a Figura 80 e o trecho de código, verifica-se que Roll/Pitch RATE tem como referência a variável “conf.rollPitchRate”, Yaw RATE a variável “conf.yawRate” e Throttle PID attenuation a variável “conf.dynThrPID”. A atenuação dos ganhos é feita de maneira diferente entre os eixos. É importante mencionar a discrepância entre os valores apresentados no software (Figura 80) e os valore gravados na EEPROM. Se é apresentado no software 0,10, este possui valor 10 na EEPROM, esta relação vale para as três variáveis. O limite de cada variável seria 1,00 na apresentação software.

18.3.2.1 Ajuste dinâmico para Roll e Pitch

O ajuste do ganho dinâmico para malha de controle referente aos movimentos de rolagem e arfagem depende das variáveis “conf.rollPitchRate” e “conf.dynThrPID”. A “conf.dynThrPID” é referente e exclusiva ao canal de potência dos motores, onde para a aquisição da variável “prop2”, variável de proporção para o cálculo do dinamismo do ganho, se considera três condições:

- **rcData[THROTTLE] < 1500.** Se a potência é menor ao valor referente á 1500μs do stick do rádio transmissor, “prop2” recebe o seguinte valor:

$$prop2 = 100 \quad [125]$$

- **1500 >rcData[THROTTLE] < 2000.** Se a potência é maior ao valor referente a 1500μs do rádio transmissor e menor que 2000μs, “prop2” recebe o seguinte valor:

$$prop2 = 100 - conf.dynThrPID \times \frac{rcData[THROTTLE] - 1500}{2000 - 1500} \quad [126]$$

- **rcData[THROTTLE] > 2000.** Por redundância considera-se o caso em que a potência é maior ao valor referente a 2000μs do rádio transmissor, “prop2” recebe o seguinte valor:

$$prop2 = 100 - conf.dynThrPID \quad [127]$$

A “conf.rollPitchRate” está relacionada com a variável “prop1”, onde se tem a seguinte aquisição:

$$prop1 = \left(100 - conf.rollPitchRate \times \frac{tmp}{500}\right) \times \frac{prop2}{100} \quad [128]$$

Onde “tmp” exerce a mesma função de capturar a intensidade da posição do stick no intervalo de [0, 500].

Tendo a variável “prop1”, dependente da variável “prop2”, atualizam-se os ganhos “P8” e “D8” nas malhas de rolagem e arfagem:

$$dynP8 = conf.P8 \times \frac{prop1}{100} \quad [129]$$

$$dynD8 = conf.D8 \times \frac{prop1}{100} \quad [130]$$

18.3.2.2 Ajuste de ganho para Yaw

Depende de apenas “conf.yawRate”, ou seja, não há interferência da potência dos motores em cima da atualização do ganho na malha de controle de guinada. A variável “prop1” é atualizada da seguinte forma:

$$prop1 = 100 - conf.yawRate \times \frac{tmp}{500} \quad [131]$$

Tendo a variável “prop1”, atualizam-se os ganhos “P8” e “D8” na malha de guinada de maneira semelhante ás malhas de rolagem e arfagem, ou seja, Equações [129] e [130]

Pode ser observado que as varáveis “conf.rollPitchRate”, “conf.dynThrPID” e “conf.yawRate”, conforme incrementadas, atenuam os ganhos “P8” e “D8” da malha de controle, deixando o sistema mais livre da ação de controle, e esses ganhos diminuem na medida que o comando cresce. É importante reforçar, que

“conf.dynThrPID” faz isso de maneira dinâmica, atualizando os ganhos de acordo com a intensidade de potência dos motores.

18.4 Lógica de Rádio

A lógica criada para a interação do usuário, através do rádio transmissor, com o objetivo de acessar as funções presentes no quadrirrotor, ocorre no loop principal do firmware, portanto, para uma explicação detalhada é necessário apresentar as variáveis declaradas dentro do loop principal. Estas encontram-se na Tabela 33. É possível notar, que a maioria das variáveis declaradas são para a utilização na malha de controle. A identificação e explicação das malhas de controle encontra-se no capítulo posterior a esse.

Tabela 33 – Variáveis declaradas no loop principal.

Variável	V.I	Descrição
rcDelayCommand	-	Contador de 20ms (50Hz) para cada ciclo do rádio controle.
axis	-	Variável para referenciar os eixos de rolagem, arfagem e guinada.
error	-	Armazena o erro entre setpoint (rcCommand) e a média de leitura do giroscópio para malha de controle do ACRO MODE, ou do LEVEL MODE para o eixo de guinada.
errorAngle	-	Armazena o erro entre setpoint (2*rcCommand) e o ângulo de rolagem ou arfagem para malha de controle do LEVEL MODE.
delta	-	Variação entre duas médias de leituras do giroscópio. (ADC)
detalsum	-	Soma de 3 variações das médias de leituras do giróspio.

Pterm	-	Termo proporcional.
Iterm	-	Termo integrativo.
Dterm	-	Termo derivativo.
lastGyro[3]	{0,0,0}	Variação entre duas médias de leituras do giroscópio da iteração anterior.
delta1[3]	-	Variação entre duas médias de leituras do giroscópio de uma iteração anterior.
detal2[3]	-	Variação entre duas médias de leituras do giroscópio de duas iterações anteriores.
errorGyroI[3]	{0,0,0}	Variável para armazenar a integração dos erros na malha de controle do ACRO MODE.
errorAngleI[2]	{0,0}	Variável para armazenar a integração dos erros na malha de controle do LEVEL MODE.
rcTime	0	Período de atualização dos valores do rádio controle.
initialThrottleHold	-	Valor de rcCommand[THROTTLE] no momento em que se seleciona o ALTITUDE HOLD
i	-	Variável para iteração das leituras dos canais auxiliares do rádio controle.

Fonte: Autoria Própria.

Para se ter noção dos tipos das variáveis declaradas, segue o bloco de código referente.

void loop() (declaração das variáveis)

```
void loop () {
  static uint8_t rcDelayCommand;
  uint8_t axis,i;
  int16_t error,errorAngle;
  int16_t delta,deltaSum;
  int16_t PTerm,ITerm,DTerm;
  static int16_t lastGyro[3] = {0,0,0};
  static int16_t delta1[3],delta2[3];
  static int16_t errorGyroI[3] = {0,0,0};
  static int16_t errorAngleI[2] = {0,0};
  static uint32_t rcTime  = 0;
  static int16_t initialThrottleHold;

  //...
```

Seguindo o código, logo encontra-se a lógica do rádio controle, onde tem-se o algoritmo desenvolvido para interpretar a intensidade dos canais, e com isso realizar certas funções.

void loop() (lógica de rádio)

```
 //...
 #define RC_FREQ 50

  if (currentTime > rcTime ) {
    rcTime = currentTime + 20000;
    computeRC();

    if (rcData[THROTTLE] < MINCHECK) {
      errorGyroI[ROLL]=0; errorGyroI[PITCH]=0; errorGyroI[YAW]=0;
      errorAngleI[ROLL]=0; errorAngleI[PITCH]=0;
      rcDelayCommand++;

      if (rcData[YAW] < MINCHECK && rcData[PITCH] < MINCHECK &&
!f.ARMED) {
        if (rcDelayCommand == 20) {
          calibratingG=400;}
      }
      else if (conf.activate[BOXARM] > 0) {
        if ( rcOptions[BOXARM] && f.OK_TO_ARM) {
          f.ARMED = 1;
        } else if (f.ARMED) f.ARMED = 0;
        rcDelayCommand = 0;
      #ifdef ALLOW_ARM_DISARM_VIA_TX_YAW
      } else if ( (rcData[YAW] < MINCHECK )  && f.ARMED) {
        if (rcDelayCommand == 20) f.ARMED = 0;
      } else if ( (rcData[YAW] > MAXCHECK ) && rcData[PITCH] <
MAXCHECK && !f.ARMED && calibratingG == 0 && f.ACC_CALIBRATED) {
        if (rcDelayCommand == 20) {
          f.ARMED = 1;
        }
      #endif
      } else
        rcDelayCommand = 0;
    } else if (rcData[THROTTLE] > MAXCHECK && !f.ARMED) {
      if (rcData[YAW] < MINCHECK && rcData[PITCH] < MINCHECK) {
        if (rcDelayCommand == 20) calibratingA=400;
        rcDelayCommand++;
      } else if (rcData[YAW] > MAXCHECK && rcData[PITCH] <
MINCHECK) {
        if (rcDelayCommand == 20) f.CALIBRATE_MAG = 1;
        rcDelayCommand++;
      } else if (rcData[PITCH] > MAXCHECK) {
         conf.angleTrim[PITCH]+=2;writeParams(1);
      } else if (rcData[PITCH] < MINCHECK) {
         conf.angleTrim[PITCH]-=2;writeParams(1);
      } else if (rcData[ROLL] > MAXCHECK) {
         conf.angleTrim[ROLL]+=2;writeParams(1);
      } else if (rcData[ROLL] < MINCHECK) {
         conf.angleTrim[ROLL]-=2;writeParams(1);
      } else {
        rcDelayCommand = 0;
      }
    }
 //...
```

Para este trecho de código é interessante levantar algumas características antes de demonstras as funções que este realiza. Primeiro, é notável que este trecho só vai ser realizado a cada 20ms, ou seja, numa frequência de 50Hz, isto se deve pela primeira condição, onde o tempo decorrente do loop (“currentTime”) deve ser maior que “rcTime”. Uma vez dentro da condição, a variável “rcTime” incrementa 20000 (20ms) em cima da variável de tempo decorrente, garantindo a condição para a próxima iteração.

Outro aspecto importante é a variável “rcDelayCommand”, responsável por determinar quanto tempo é necessário permanecer os sticks em determinada posição para realizar uma função desejada. Cada incremento de “rcDelayCommand” equivale a 20ms, portanto, a maioria das funções são ativadas após os sticks permanecerem por 0.4 segundos na posição referente à esta.

Pode ser notado que se o stick referente ao canal de potência enviar um sinal menor que “MINCHECK” (<1100μS), as variáveis que armazenam a integração do erro do controle, tanto em LEVEL MODE, quanto em ACRO MODE, são zeradas.

A Figura 81 demonstra as posições dos sticks e as funções referentes a cada uma delas.

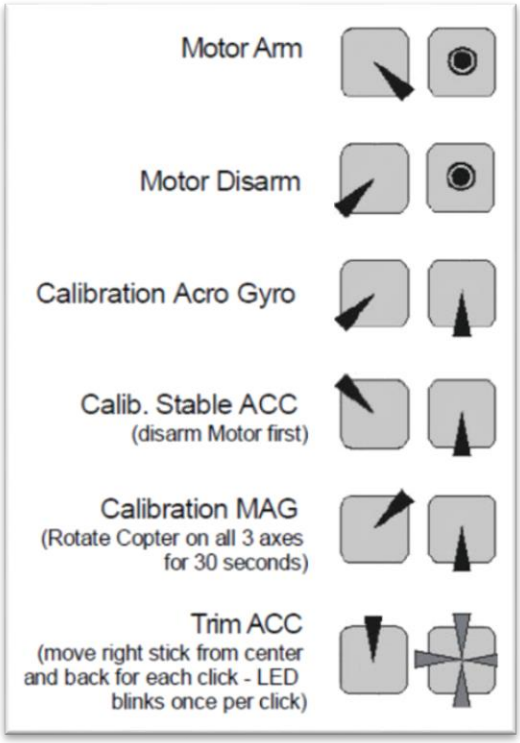
Da Figura 81, trecho de código e analisando o sistema, pode ser levantado:

- A indicação de que os motores estão armados é dada pela flag “f. ARMED” levantada. Caso esta esteja abaixada, indica-se que os motores não estão armados. A função “mixTable”, responsável pela ação de controle, na qual será vista no próximo capítulo, faz a gestão dos motores de acordo com a informação da flag.
- A calibração do giroscópio é iniciada pelo atributo de 400 pela variável “calibratingG”, ocorrendo assim, em cada eixo, 400 leituras para o cálculo da referência de calibração. OBS: Toda vez que o quadrirrotor é energizado, ocorre a calibração automática do giroscópio.
- A calibração do acelerômetro é iniciada pelo atributo de 400 pela variável “calibratingA”, ocorrendo assim, em cada eixo, 400 leituras para o cálculo da referência de calibração. É bom lembrar que as variáveis “conf.angleTrim” dos eixos de rolagem e arfagem tem seus valores nulos ao fim da calibração. A calibração do acelerômetro não

ocorre na inicialização do sistema, como no giroscópio, porém, feita sua calibração, seus valores de referência e ajuste são armazenados na EEPROM.

- A calibração do magnetômetro é iniciada quando a flag “f.CALIBRATE_MAG” é setada, necessitando assim rotacionar o quadrirrotor em 360˚ nos 3 eixos dentro de um intervalo de 30s. Assim é armazenado o maior valor e menor valor lidos para o cálculo da referência de calibração (retirada do efeito Hard Iron). Após o término da calibração a flag é abaixada. Assim como no acelerômetro, a calibração do magnetômetro não ocorre ao energizar o quadrirrotor, porém feita sua calibração, seus valores de referência são armazenados na EEPROM.
- O trim digital do acelerômetro para os eixos de rolagem e arfagem, ajustável pela variável “conf.angleTrim”, conforme a variação do stick da direita, tem seu valor incrementado ou decrementado em módulo de 2. O mais interessante, é que essa regulagem não altera em nada a referência de calibração do acelerômetro. Esta tem influência direta na malha de controle do LEVEL MODE, onde ocorre o aumento ou diminuição do setpoint (2*rcCommand). Uma regulagem de +2 em “conf.angleTrim” equivale à um offset de ($\pm 0.2^{\circ}$) no setpoint. É válido ainda mencionar, que a configuração feita é gravada na EEPROM.

Figura 81 – Sticks e suas funções.

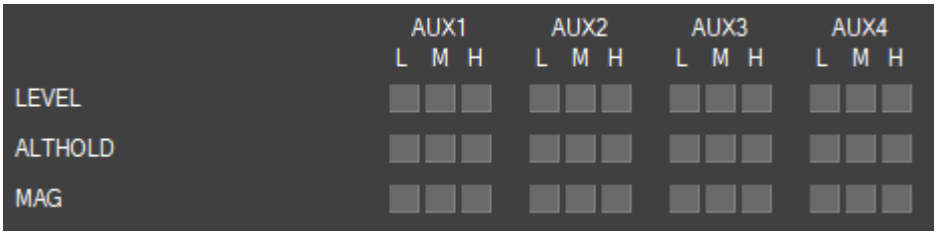


Fonte: Autoria própria.

Ainda pelo código é possível verificar uma outra opção de armar e desarmar os motores do quadrirrotor. O vetor “conf.activate” é recebido pelo quadrirrotor através da comunicação serial com o software de controle em solo. Este contém as configurações relacionadas aos canais auxiliares, portanto uma delas é armar o quadrirrotor, conforme o macro “BOXARM”, como parâmetro do vetor, indica.

Há um macro para cada função passível de se acionar através dos canais auxiliares do rádio controle. A Figura 82 demonstra a interface de configuração de ativação dos modos pelos canais auxiliares. O macro referente ao LEVEL MODE é “BOXACC”, ao ALTITUDE HOD é “BOXBARO” e ao MAG MODE é “BOXMAG”. Há diversos outros macros, todos apresentados na Tabela 32.

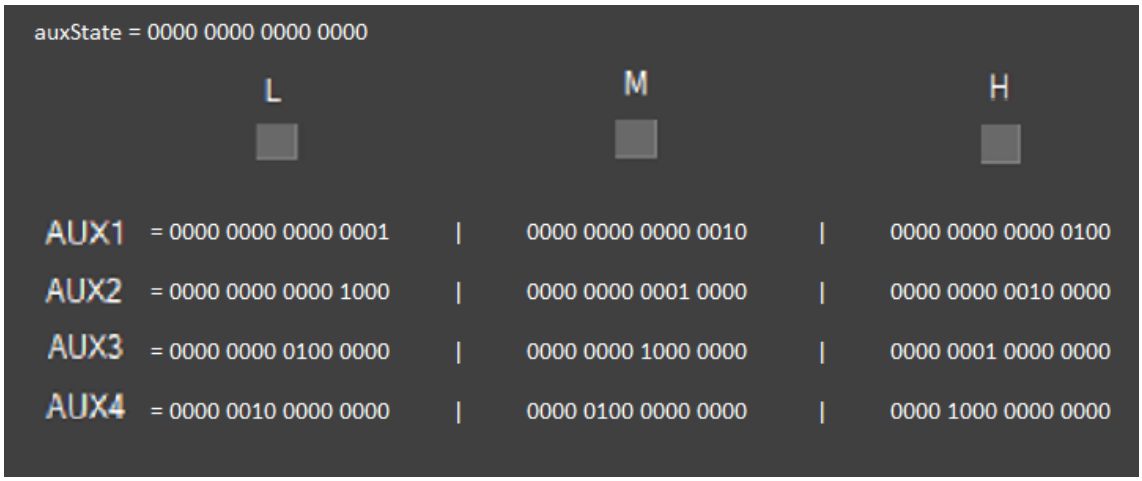
Figura 82 – Configuração dos modos de voo através dos canais auxiliares.



Fonte: Autoria própria.

O bloco de código de programação na sequência (sequência do código anterior no loop principal) demonstra o algoritmo para a leitura da configuração. Nele, em seu início, é possível verificar a declaração de uma variável de 16 bits sem sinal, “auxState”, que tem a explicação da aquisição de seus valores na Figura 83.

Figura 83 – Variação da variável “auxState” de acordo com o valor do canal auxiliar.



Fonte: Autoria própria.

Ainda da Figura 83, tem-se:

- L (baixa intensidade) = rcData[auxn]<1300μs;
- M (média intensidade) = 1300μs<rcData[auxn] <1700μs;
- H (alta intensidade) = 1700μs<rcData[auxn].

Se o canal 2 estiver em baixa intensidade, a variável “auxState” terá valor: 0000 0010 0100 1001; em média intensidade: 0000 0010 0101 1001; em alta intensidade: 0000 0010 0111 1001. Ou seja, há um operador “OU” entre as

intensidades de todos os canais auxiliares (Figura 83), portanto, se o canal 2 estiver em alta intensidade e o canal 4 em média, tem-se: 0000 0110 0111 1001.

Para explicação, considere a ativação do LEVEL MODE, que foi supostamente configurada para ativar com o canal auxiliar 3 em média e alta intensidade. A posição do vetor “rcOpitions[BOXACC]” será diferente de 0, se a intensidade do canal auxiliar 3 for pelo menos média. A variável “conf.activate[BOXACC]” é proveniente da configuração feita no software de controle em solo, portanto, esta é recebida por comunicação serial e gravada na EEPROM logo em seguida, seu valor, para a configuração dita, é: 0000 0001 1000 0000. Assim:

$$\mathrm{rcOpitions[BOXACC]} = \mathrm{auxState} \times \mathrm{conf.activate[BOXACC]} \quad [132]$$

Se o canal auxiliar 3 encontra-se em intensidade média, e outros em baixa:

$$\mathrm{rcOpitions[BOXACC]} = 0000\ 0010\ 1100\ 1001 \times 0000\ 0001\ 1000\ 0000 \quad [133]$$

$$\mathrm{rcOpitions[BOXACC]} = 0000\ 0000\ 1000\ 0000 \quad [134]$$

void loop() (leitura de configuração dos canais auxiliares)

```
  //...
 uint16_t auxState = 0;
    for(i=0;i<4;i++)
      auxState |= (rcData[AUX1+i]<1300)<<(3*i) | (1300<rcData[AUX1+
i] && rcData[AUX1+i]<1700)<<(3*i+1) | (rcData[AUX1+i]>1700)<<(3*i+2
);
    for(i=0;i<CHECKBOXITEMS;i++)
      rcOptions[i] = (auxState & conf.activate[i])>0;

    if ( rcOptions[BOXACC] && ACC ) {
      if (!f.ACC_MODE) {
        errorAngleI[ROLL] = 0; errorAngleI[PITCH] = 0;
        f.ACC_MODE = 1;
      }
    } else {
      f.ACC_MODE = 0;
    }
    if (f.ACC_MODE) {STABLEPIN_ON;} else {STABLEPIN_OFF;}
    #if BARO
      if (rcOptions[BOXBARO]) {
        if (!f.BARO_MODE) {
          f.BARO_MODE = 1;
          AltHold = EstAlt;
          initialThrottleHold = rcCommand[THROTTLE];
          errorAltitudeI = 0;
          BaroPID=0;
        }
      } else {
        f.BARO_MODE = 0;
      }
    #endif
    #if MAG
      if (rcOptions[BOXMAG]) {
        if (!f.MAG_MODE) {
          f.MAG_MODE = 1;
          magHold = heading;
        }
      } else {
        f.MAG_MODE = 0;
      }
    #endif
}
 //...
```

Os modos de voo são ativados por um dos quatro canais auxiliares, dependendo da preferência do usuário. Quando selecionados ocorrem as seguintes atualizações:

- **LEVEL MODE –** Quando o modo é ativado, este zera as variáveis que armazenam a integração do erro do controle dos eixos de rolagem e arfagem no LEVEL MODE, “errorAngleI”, e seta a flag “f.ACC_MODE”. Quando não ativo, mantem a flag abaixada. O led

azul do sistema fica ligado quando o modo está inativo e desligado quando ativo.

- **ALTITUDE HOLD –** Toda vez que o modo é ativado, a flag “f.BARO_MODE” é ativa, a variável de referência do controle de altitude, “AltHold” é atualizada com o valor estimado de altitude atual, o valor atual referente ao canal de aceleração dos motores é armazenado na variável “initialThrottleHold” e há a reinicialização do valor da variável que armazena a integração do erro do controle do ALTITUDE HOLD e de seu valor de ação de controle, “BaroPID”.
- **MAG MODE –** Quando ativado, seta a flag “f.MAG_MODE” e guarda o ângulo de orientação de guinada atual na variável “magHold”. Quando não ativo, mantem a flag abaixada.

Por fim termina o trecho que é realizado a cada 20ms. Antes de entrar na lógica das malhas de controle, ainda é necessário realizar mais algumas aquisições de dados para o loop, estas que estão apresentadas no bloco de código a seguir.

void loop()

```
//...
else { // not in rc loop
    static uint8_t taskOrder=0;
    switch (taskOrder++ % 5) {
      case 0:
        #if MAG
          Mag_getADC();
        #endif
        break;
      case 1:
        #if BARO
          Baro_update();
        #endif
        break;
      case 2:
        #if BARO
          getEstimatedAltitude();
        #endif
        break;
      case 3:
        break;
      case 4:
        break;
    }
  }
  computeIMU();
  currentTime = micros();
  cycleTime = currentTime - previousTime;
  previousTime = currentTime;

 //... CONTROLE PID
```

O trecho de código apresenta um “switch case” com 5 tipos de casos selecionáveis. A função deste switch case é não chamar todas as funções presentes nele em apenas um loop, evitando picos de tempo. Portando é realizada a sequência (0,1,2,3,4,0,1,2,3,4...) ao decorrer das iterações. O caso 0 é responsável por atualizar os valores dos três eixos do magnetômetro; O caso 1 atualiza o valor da altura lida pelo barômetro; O caso 2 realiza a malha de controle do modo ATITUDE HOLD; Os casos 3 e 4 estão reservados para qualquer implementação futura.

Após o switch case, é atualizado o tempo de “cycletime”, que indica o tempo de duração do loop principal. Este valor varia próximo de 5000μs, portanto, a uma taxa de 200Hz.

19 IDENTIFICAÇÃO DOS CONTROLES

19.1 Sistemas de Controle

A engenharia de controle parte do princípio da realimentação com o objetivo de controle de determinadas variáveis de um sistema. Sua aplicação é essência em diversos outros campos da engenharia e ciência. Exemplos de sua implementação são encontrados de maneira intrínseca em sistemas de veículos espaciais, sistemas robóticos, sistemas de manufaturas, etc. (BAYER; ARAÚJO, 2012; OGATA, 2010). Dada a introdução ao tema, de acordo com Ogata (2010), apresenta-se a termologia básica sobre os sistemas de controle:

- **Planta:** parte de um equipamento ou um conjunto de componentes presentes em um equipamento, com funcionamento integrado, tendo como objetivo a realização de uma determinada operação. (OGATA, 2010; ARAÚJO, 2007). Este termo, neste trabalho, será utilizado para representar o objetivo alvo de controle, o quadrirrotor.
- **Processo:** conjunto de atividades ou passos que tendem a um objetivo. (BAYER; ARAÚJO, 2012). Também pode ser definido como uma operação ou desenvolvimento, progredindo em etapas, com uma série de mudanças graduais sequenciais, levando a um resultado. (ARAÚJO, 2007). De maneira mais sucinta, Ogata (2010) define que processo é toda operação a ser controlada.
- **Sistemas:** combinação de componentes agindo em conjunto para atingir determinado objetivo. Não se restringindo apenas a algo físico, mas também a sistemas biológicos, econômicos e outros. (OGATA, 2010).
- **Distúrbios:** mais comumente chamando de ruído, é um sinal que pode afetar a variável de saída do sistema. (OGATA, 2010).
- **Controle com realimentação:** forma de controle que busca reduzir a diferença entre a variável controlada (variável de saída) e o valor desejado (variável de entrada), atuando com base no erro entre essas duas em cima de uma variável manipulada. (BAYER; ARAÚJO, 2012; OGATA, 2010).

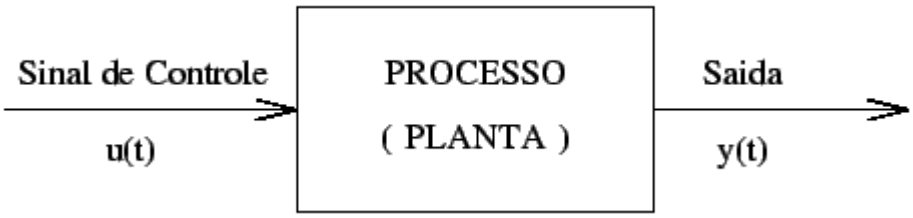
Sistemas de controle são ainda classificados entre controle em malha aberta e controle em malha fechada. A diferença está diretamente relacionada à atuação do controle na produção da saída desejada. (BAYER; ARAÚJO, 2012).

19.1.1 Controle em Malha Aberta

O sinal de controle ($u(t)$) não é calculado a partir de uma medição no sinal de saída ($y(t)$), ou seja, não há a medição do sinal de saída nem a comparação deste com o sinal de entrada. (OGATA, 2010; SILVA, 2000).

O sinal de entrada de referência deste sistema é um sinal de controle predeterminado com uma condição fixa de operação, condições estas baseadas em experiências passadas, portanto, passíveis de calibração. (BAYER; ARAÚJO, 2012; OGATA, 2010; SILVA, 2000).

Figura 84 - Controle em malha aberta.

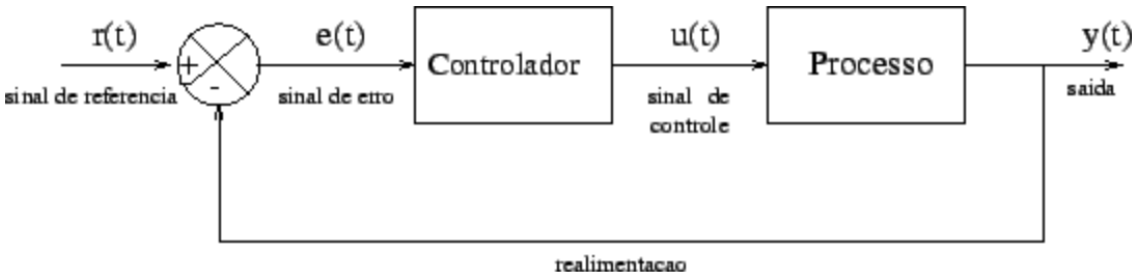


Fonte: (SILVA, 2000).

19.1.2 Controle em Malha Fechada

Num sistema em malha fechada, tem-se a realimentação do sinal de saída ($y(t)$) para comparação com o sinal de entrada ($r(t)$), gerando um sinal de erro ($e(t)$), que é enviado para o controlador determinar a ação a ser tomada por meio de um sinal de controle ($u(t)$) com a finalidade de ajustar a saída do sistema a um valor predefinido. (BAYER; ARAÚJO, 2012).

Figura 85 - Controle em malha fechada.



Fonte: (SILVA, 2000).

19.2 Estabilidade

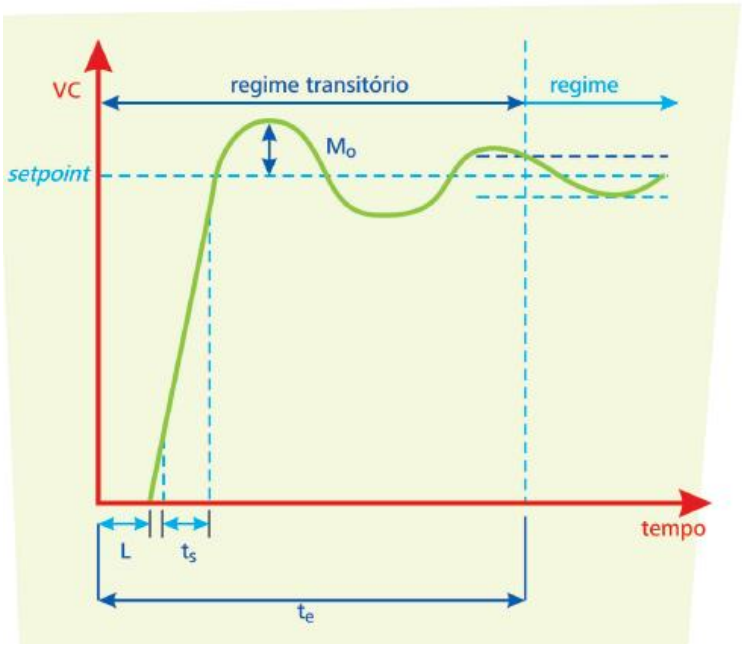
A estabilidade de um sistema pelo conceito de BIBO-estabilidade (bounded input – bounded output). Determina que um sistema é estável se, para todo sinal de entrada com amplitude limitada, o sinal de saída é limitado. Assim sendo instável, se um sinal de entrada com amplitude limitada gera um sinal de saída divergente. (BAYER; ARAÚJO, 2012).

Dada uma alteração de carga no sistema, a resposta típica pode ser decomposta em duas: regime transitório e regime permanente. (BAYER; ARAÚJO, 2012). A Figura 86 ilustra os dois regimes de resposta e ainda indica as seguintes características:

- $\boldsymbol{M_O}$ **– pico de resposta (overshoot):** é o máximo valor que a variável controlada atinge em relação ao setpoint devido à primeira oscilação do sistema. O overshoot pode ser utilizado como um indicativo de estabilidade relativa do sistema, onde quanto maior o seu valor, mais próximo o sistema estará de um comportamento instável. (BAYER; ARAÚJO, 2012).
- $\boldsymbol{t_e}$ **– tempo de estabilização (acomodação):** tempo para que a variável controlada atinja uma variação de $\pm$5% em relação ao setpoint. (BAYER; ARAÚJO, 2012).
- $\boldsymbol{t_s}$ **– tempo de subida:** tempo entre a variação de 10% á 90% da variável controlada em relação ao setpoint. (BAYER; ARAÚJO, 2012).

- **L – atraso (tempo morto):** tempo para que a variável controlada esboce qualquer reação inicial. É importante citar, que em um sistema em malha fechada, quanto maior o tempo morto, maior serão as oscilações deste e maior serão as chances de instabilidade do sistema. (BAYER; ARAÚJO, 2012).

Figura 86 – Resposta a uma alteração de carga: regime transitório e regime permanente.



Fonte: (BAYER; ARAÚJO, 2012).

19.3 Controladores

Serão considerados aqui apenas sistemas em malha fechada, onde o erro gerado entre o setpoint e a variável manipulada é enviado para o controlador aplicar algoritmos que determinam a ação de controle necessária.

As ações de controle podem ser utilizadas separadamente ou em associação para a aplicação de um controle automático. (BAYER; ARAÚJO, 2012).

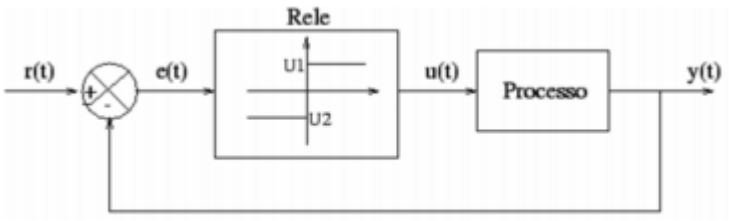
19.3.1 Ação Liga-Desliga (on-off)

Os controladores liga-desliga, ou controladores de duas posições, conforme o nome já diz, fornecem apenas dois valores de saída, sendo ligado ou desligado. Seu funcionamento é composto por uma simples chave, onde o controlador

compara o sinal de entrada com o de saída (realimentado), se o sinal de saída exceder o sinal de entrada, desliga-se a chave, se o sinal de saída for menor, liga-se a chave. (BAYER; ARAÚJO, 2012).

Segundo Lotufo (2015), tem-se o diagrama de blocos ilustrado na Figura 87, levando à equação [135].

Figura 87 – Controlador liga-desliga



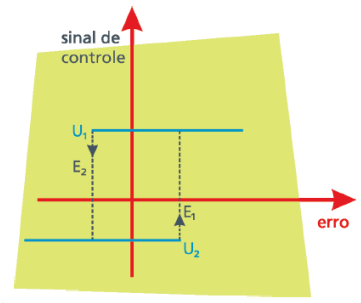
Fonte: (LOTUFO, 2015).

$$u(t) = \begin{cases} U_1, & para\ e(t) > 0 \\ U_2, & para\ e(t) < 0 \end{cases} \qquad [135]$$

Com U_1 e U_2 representando os dois valores de ações possíveis do controlador.

Um controlador liga-desliga com ruídos em sua entrada resultaria em oscilações que abriria e fecharia o contato do controlador constantemente, podendo ocasionar algum desgaste físico. Pensando nisso, aplica-se um controlador liga-desliga com histerese, onde é dado certa margem de erro para que haja algum chaveamento (mudanças entre U_1 e U_2). (BAYER; ARAÚJO, 2012). O funcionamento deste controlador pode ser resumido na Figura 88.

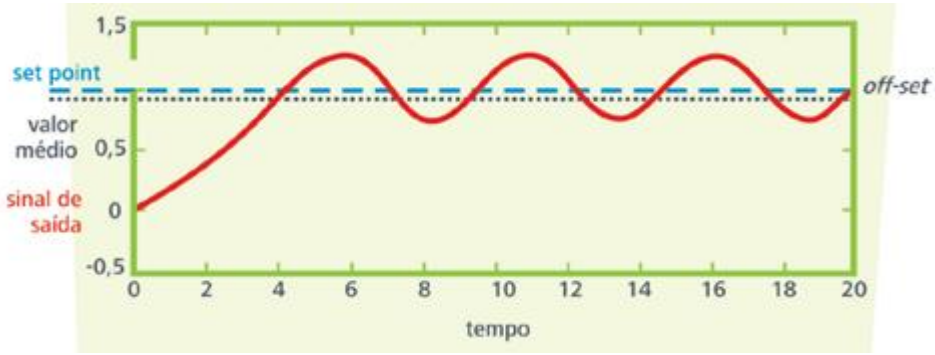
Figura 88 – Controlador liga-desliga com histerese.



Fonte: (BAYER; ARAÚJO, 2012).

Devido a própria aplicação da histerese no controlador e pela variação da carga, tem-se certa oscilação na resposta em malha fechada em regime permanente, conforme pode ser analisado na Figura 89. (BAYER; ARAÚJO, 2012).

Figura 89 – Resposta característica de um controlador liga-desliga com histerese.



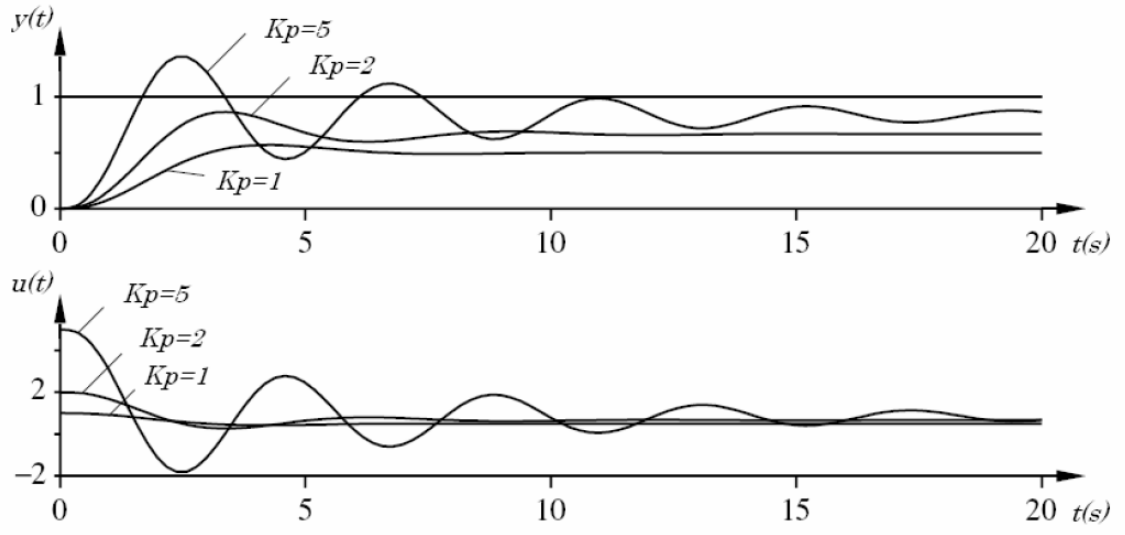
Fonte: (BAYER; ARAÚJO, 2012).

19.3.2 Ação Proporcional (P)

Na ação proporcional, o sinal de saída do controlador ($u(t)$)) é proporcional à magnitude do sinal de erro ($e(t)$). (SUNADA, 2007). Sendo k_p a constante de ganho proporcional, equaciona-se:

$$u(t) = k_p \times e(t) \qquad [136]$$

Figura 90 – Controle proporcional com variação em seu ganho. $y(t)$ Denota a saída do sistema e $u(t)$ a ação de controle.



Fonte: (ASTROM, 1995).

A Figura 90 apresenta a resposta de um sistema com controle proporcional, onde é possível verificar, com a variação da constante de ganho proporcional (k_p), as características transitórias e de regime permanente da saída do sistema ($y(t)$) junto à ação de controle ($u(t)$) tomada. Fazendo a análise, é possível verificar, que quanto maior k_p, menor será o erro em regime permanente, porém maior será a oscilação em regime transitório, podendo, em muitos casos, levar o sistema à instabilidade caso o ganho seja muito alto. (ASTROM, 1995; SUNADA, 2007).

É possível notar que a ação proporcional elimina as oscilações presentes no controle liga-desliga (Figura 89), porém qualquer variação na carga do sistema irá persistir uma diferença (off-set) entre o setpoint e a saída do sistema em regime permanente. O off-set será menor se k_p aumentar, porém, como já dito, aumenta-se a oscilação da resposta com isso. (ASTROM, 1995; BAYER; ARAÚJO, 2012; SUNADA, 2007).

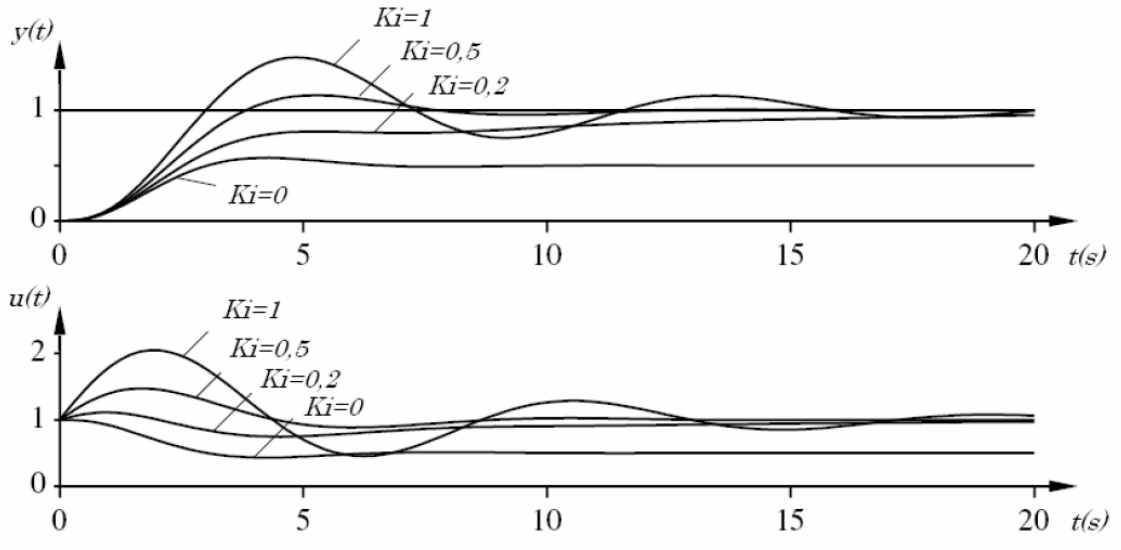
19.3.3 Ação Integral (I)

Na ação integral, o sinal de controle ($u(t)$) é originado a partir do erro acumulado no tempo. (SUNADA, 2007). Sendo k_i a constante de ganho integral, equaciona-se:

$$u(t) = k_i \int_0^t e(t)dt \qquad [137]$$

A principal razão de aplicação dessa ação é a melhoria da precisão do sistema, sendo assim não aplicada de maneira isolada. O erro acumulado até um instante de tempo, garante o seguimento de uma referência com erro nulo em regime permanente, podendo assim, retirar o erro de off-set apresentado no controle proporcional. De maneira mais sucinta, a ação integral atua no processo ao longo do tempo enquanto existir erro entre setpoint e sinal de saída. (BAYER; ARAÚJO, 2012; SUNADA, 2007).

Figura 91 – Controle proporcional e integral com variação no ganho integral. $\mathbf{y(t)}$ Denota a saída do sistema e $\mathbf{u(t)}$ a ação de controle.



Fonte: (ASTROM, 1995).

A Figura 91 apresenta a resposta do sistema de um controlador proporcional e integral. O ganho proporcional é mantido constante com k_p = 1, e o ganho integral (k_i) é variado a fim de analisar seu efeito na resposta. Pode-se observar que o erro de off-set em regime permanente é removido com a aplicação do ganho integral. Para valores menores, a resposta de saída caminha lentamente para o setpoint. Aumentando-se este valor, têm-se uma aproximação mais rápida, porém com um aumento de oscilação no regime transitório. (ASTROM, 1995; SUNADA, 2007).

19.3.4 Ação Derivativa (D)

Na ação derivativa, o sinal de controle ($u(t)$) é proporcional à derivada do sinal de erro ($e(t)$) originado do sistema. (SUNADA, 2007). Sendo k_d a constante de ganho integral, equaciona-se:

$$u(t) = k_d \frac{de(t)}{dt} \quad [138]$$

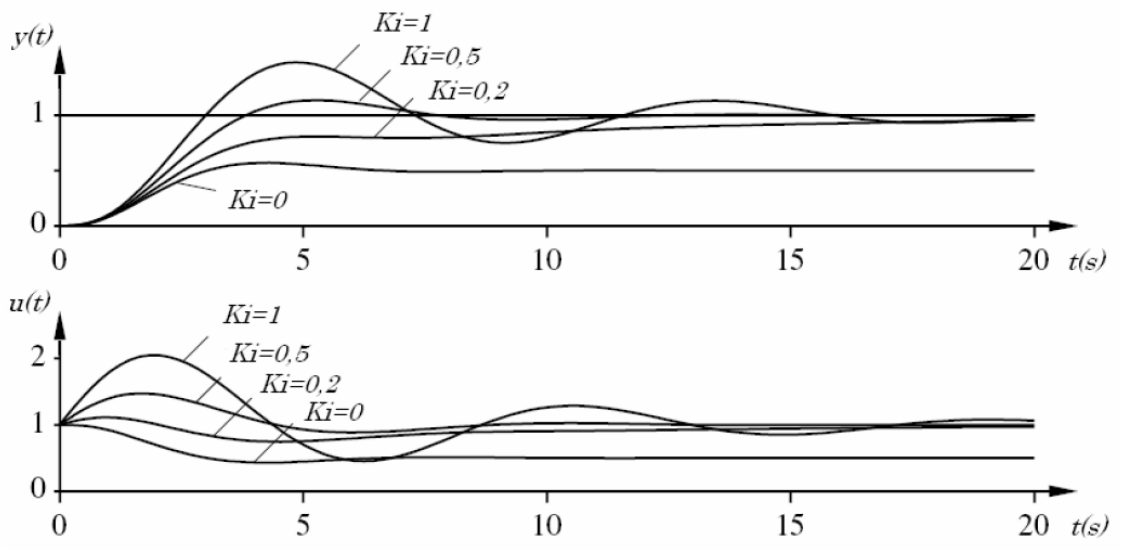
A definição de derivada para qualquer função é, basicamente, a variação desta em um instante de tempo infinitesimal. Portanto, ao usar-se disto em uma ação de controle, adiciona-se uma característica preditiva, obtendo repostas transitórias

mais rápidas. (SUNADA, 2007). É possível dizer que este controle antecipa as variações excessivas, diminuindo consideravelmente um overshoot na resposta. (BAYER; ARAÚJO, 2012).

É destacável, que a ação derivativa não tem efeito no regime permanente (onde o erro é constante), portanto, como a ação integral, não é aplicada de maneira isolada. (SUNADA, 2007).

A Figura 92 apresenta a resposta do sistema de um controlador proporcional, integral e derivativo. Os ganhos proporcional e integral são mantidos constantes em k_p = 3 e k_i = 1,5, e o ganho derivativo (k_d) é variado a fim de analisar seu efeito na resposta. Com o aumento do ganho derivativo, tem-se amortecimento nas oscilações no regime transitório, diminuindo a frequência oscilatória, porém, com o aumento demasiado do ganho, volta-se a diminuir o amortecimento e aumentar a frequência de oscilação. (ASTROM, 1995; SUNADA, 2007).

Figura 9*2* – Controle proporcional, integral e derivativo com variação no ganho derivativo. $y(t)$ Denota a saída do sistema e $u(t)$ a ação de controle.



Fonte: (ASTROM, 1995).

É importante dizer que a ação derivativa não deve ser utilizada em sistemas com reposta rápida ou com sinais ruidosos, o que pode provocar mudanças súbitas na ação de controle e, consequentemente, à instabilidade. (BAYER; ARAÚJO, 2012; SUNADA, 2007).

19.3.5 Associação de ações de controle

As ações integral e derivativa não devem ser aplicadas separadamente a um sistema, estas devem sempre estar associadas a uma ação proporcional. As possíveis combinações de ações de controle são: PI, PD e PID. A escolha de qual combinação utilizar vai depender da vantagem que essa traz ao processo. (BAYER; ARAÚJO, 2012).

O controlador proporcional-integral-derivativo (PID) aproveita das características de cada uma das ações, assim, além de atuar conforme a amplitude do erro, têm-se um controle da velocidade de atuação e a exatidão da saída no regime permanente. (BAYER; ARAÚJO, 2012). A equação [139] expressa a saída do controlador PID:

$$u(t) = k_p \times e(t) + k_i \int_0^t e(t)dt + k_d \frac{de(t)}{dt} \qquad [139]$$

19.4 Discretização

Para implementar uma regra de controle do tempo contínuo, como um controlador PID em um computador digital, é necessário aproximar a derivada e integral que aparecem na regra de controle em [139]

19.4.1 Ação Proporcional

É simplesmente implementada, apenas trocando as variáveis de tempo contínuo em [136], por amostradas.

$$u(k) = k_p \times e(k) \qquad [140]$$

Onde:

$$e(k) = r(k) - y(k) \qquad [141]$$

Sendo k os instantes de amostragem.

19.4.2 Ação Integral

Aplicando a proximação por diferenças (Backward) em [137], conduz a:

$$u(t) = k_i \int_0^t e(t)dt$$

$$\frac{du}{dt} = k_i\, e \qquad [142]$$

$$\frac{u(k) - u(k-1)}{h} = k_i\, e(k) \qquad [143]$$

$$u(k+1) = u(k) + k_i\, h\, e(k+1) \qquad [144]$$

Sendo h o tempo entre amostras.

O termo h contido no denominador é suprimido, pois além das amostras já estarem devidamente espaçadas em relação ao tempo, esse valor é computado no momento em que é ajustado o parâmetro k_i do controlador na etapa de sintonia.

19.4.3 Ação Derivativa

Aplicando a proximação por diferenças (Backward) em [138], conduz a:

$$u(t) = k_d \frac{de(t)}{dt}$$

$$\frac{e(k) - e(k-1)}{h} = \frac{u(k)}{k_d} \qquad [145]$$

$$u(k) = k_d\, \frac{e(k) - e(k-1)}{h} \qquad [146]$$

Sendo h o tempo entre amostras, e também é suprimido em aplicações digitais.

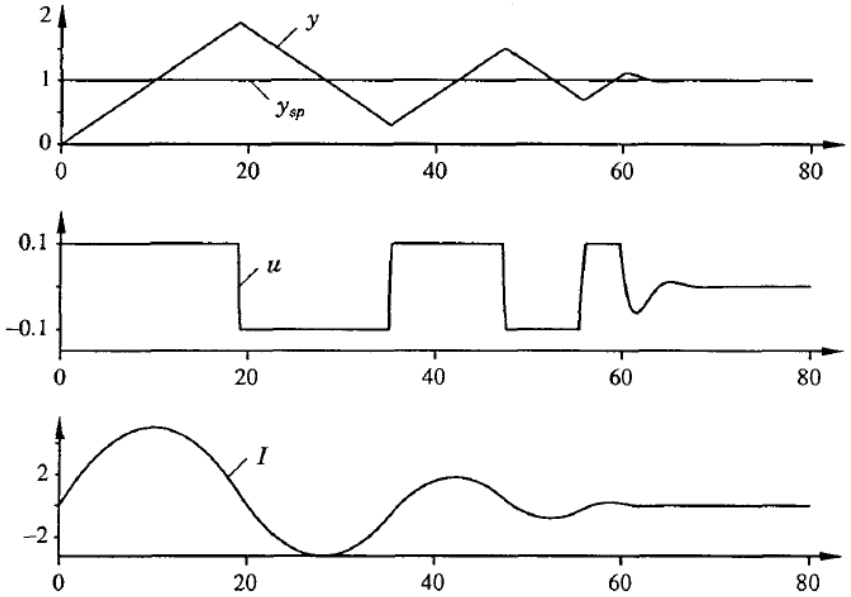
19.5 Wind-up do integrador

Devem ser considerados inúmeros aspectos na aplicação de um controlador. Deve-se saber que todo atuador tem limitações: o motor tem um limite de velocidade, uma válvula não pode abrir ou fechar mais que seu limite, etc. Para um sistema de controle com um grande alcance das condições de funcionamento, pode acontecer com que a variável de controle alcance os limites do atuador. Quando isso acontece, a realimentação é quebrada e o sistema se direciona para malha aberta, pois o atuador permanecerá em seu limite independente do processo de saída. Se um controlador com ação integral é usado, o erro continuará sendo integrado. Isso significa que o termo integral pode ficar muito grande, na linguagem de controle, isso é chamado de “Wind-up”. É então requerido que o erro tenha sinal de oposição por um longo período para que assim o atuador saia da saturação. É valido citar então, que qualquer controlador com ação integral, pode resultar em grandes transientes quando o atuador é saturado. (ASTROM, 1995).

O fenômeno Wind-up é ilustrado na Figura 93, na qual é mostrado o controle de um processo de integração com um controlador PI. O setpoint inicial muda tão bruscamente que o atuador é saturado em seu limite mais alto. O termo integrativo inicialmente é incrementado porque o erro é positivo, e este alcança seu valor mais alto no tempo t=10, quando o erro passa por seu valor nulo. A saída permanece saturada neste ponto por causa do alto valor acumulado do termo integrativo. A saída apenas sai do limite de saturação até que o erro seja negativo por um longo período de tempo, diminuindo o termo integrativo. Note que o sinal de controle oscila entre seus limites por várias vezes antes de se comportar de maneira linear. (ASTROM, 1995).

Existem diversas maneiras de contornar o Wind-up apresentadas na literatura, porém, nas controladoras open-source, isso é comumente implementado com apenas um saturador no termo integrativo.

Figura 93 – Efeito de Wind-up. y = saída; y_{sp}= setpoint; u = ação de controle; I = ação integral.



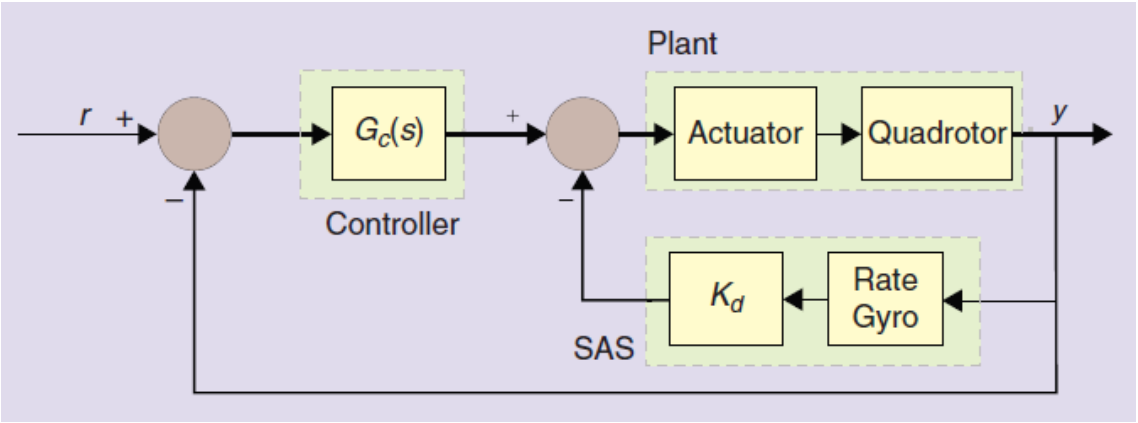
Fonte: (ASTROM, 1995)

19.6 SAS

Maioria das aeronaves militares e comerciais de alta performance apenas alcançam os requerimentos de qualidade de voo com o uso de um Sistema de Aumento de Estabilidade (do inglês Stability Augmentation System, SAS). Muitas dessas aeronaves são instáveis e seria impossível voar sem um sistema de controle automático. O SAS basicamente utiliza sensores para medir a taxa de variação angular da aeronave e realimenta dados processados desses sinais para os servomecanismos que comandam as superfícies aerodinâmicas de controle. Deste modo, um momento aerodinâmico proporcional à velocidade angular e sua derivada pode ser gerado e usado para produzir um efeito de amortecimento no movimento. (STEVENS; LEWIS, 2003).

Colocando o foco nos quadrirrotores, modelos genéricos revelam que os polos do sistema estão localizados no plano direito do eixo imaginário, tendo como consequência uma taxa de amortecimento negativa, portanto é necessário, para sua estabilização, a implementação de um algoritmo de controle com realimentação, na qual seria o SAS. (KIM et al., 2012). O SAS pode ser visto na Figura 94.

Figura 94 – Configuração de um controle de atitude genérico. A taxa de amortecimento do sistema é regulada através da realimentação do SAS.

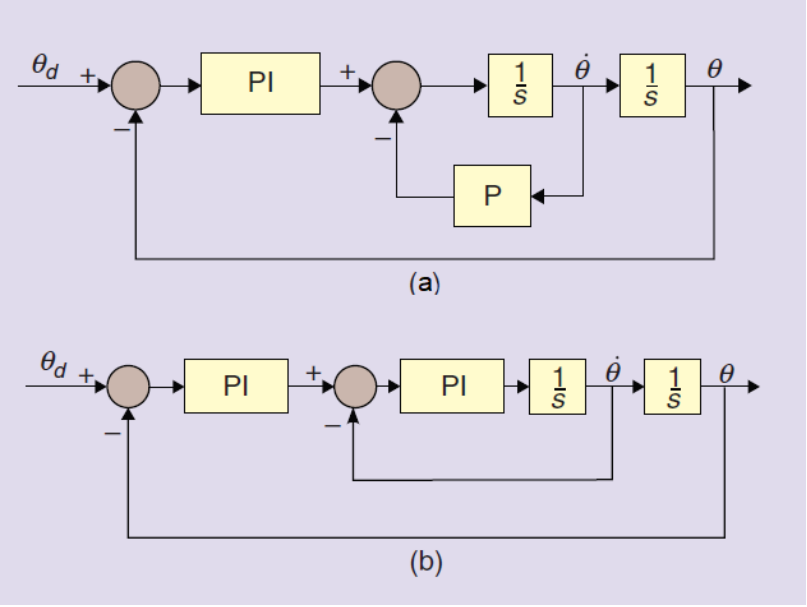


Fonte: KIM et al. (2012).

É visto que o SAS atinge a estabilidade do sistema por meio da taxa de medição do giroscópio em realimentação com um ganho.

Para aplicação do SAS, controladores PID ($G_c(s)$) são utilizados nas configurações multi-malha e cascata. (THUMS et al., 2012). Exemplo desses controladores nas placas OSPs MultiWii e Openpilot podem ser vistos na Figura 95.

Figura 95 – Estrutura de Controle. (a) MultiWii. (b) OpenPilot.



Fonte: KIM et al. (2012).

Para investigação em análise matemática, Nelson (1998) demonstra o aumento da taxa de amortecimento através de uma realimentação SAS, utilizando a equação diferencial do movimento de arfagem de uma aeronave comercial.

19.7 Controles do MultiWii Pro

Nesta seção serão apresentadas as configurações das malhas de controle presentes no MultiWii Pro. Serão apresentadas as malhas dos modos de voo primário: Level Mode, Acro Mode, Mag Mode e Altitude Hold Mode.

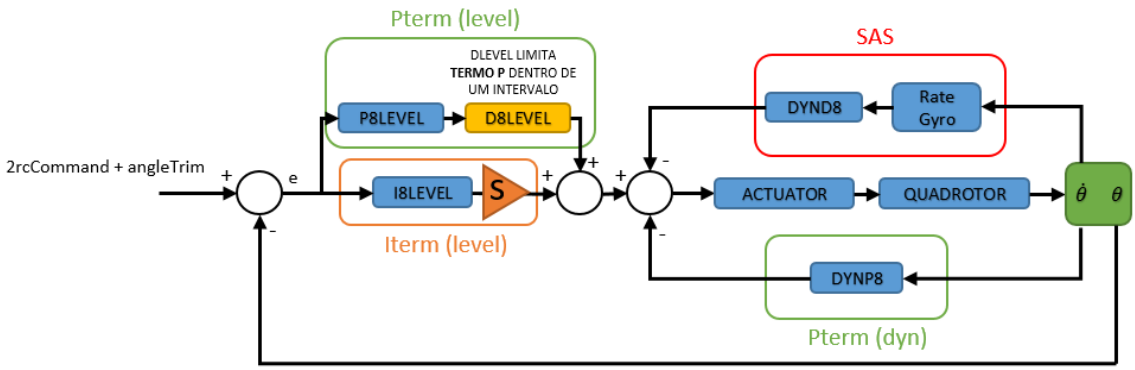
A estrutura de código abaixo contém os modos Level Mode e Acro Mode.

Level Mode / Acro Mode

```
 for(axis=0;axis<3;axis++) {
  if (f.ACC_MODE && axis<2 ) { //LEVEL MODE
    errorAngle = constrain(2*rcCommand[axis],-300,+300) -
angle[axis] + conf.angleTrim[axis];
    PTerm = (int32_t)errorAngle*conf.P8[PIDLEVEL]/100 ;
    PTerm = constrain(PTerm,-
conf.D8[PIDLEVEL]*5,+conf.D8[PIDLEVEL]*5);
    errorAngleI[axis]  = constrain(errorAngleI[axis]+errorAngle,-
10000,+10000);
    ITerm = ((int32_t)errorAngleI[axis]*conf.I8[PIDLEVEL])>>12;
  } else { //ACRO MODE or YAW axis
      if (abs(rcCommand[axis])<350) error =
rcCommand[axis]*10*8/conf.P8[axis] ;
      else error = (int32_t)rcCommand[axis]*10*8/conf.P8[axis] ;
      error -= gyroData[axis];
      PTerm = rcCommand[axis];
      errorGyroI[axis]  = constrain(errorGyroI[axis]+error,-
16000,+16000);
      if (abs(gyroData[axis])>640) errorGyroI[axis] = 0;
      ITerm = (errorGyroI[axis]/125*conf.I8[axis])>>6;
    }
  if (abs(gyroData[axis])<160) PTerm -=
gyroData[axis]*dynP8[axis]/10/8;
  else PTerm -= (int32_t)gyroData[axis]*dynP8[axis]/10/8;

  delta          = gyroData[axis] - lastGyro[axis];
  lastGyro[axis] = gyroData[axis];
  deltaSum       = delta1[axis]+delta2[axis]+delta;
  delta2[axis]   = delta1[axis];
  delta1[axis]   = delta;

  if (abs(deltaSum)<640) DTerm = (deltaSum*dynD8[axis])>>5;
  else DTerm = ((int32_t)deltaSum*dynD8[axis])>>5;

    axisPID[axis] =  PTerm + ITerm - DTerm;
  }
  mixTable();
  writeMotors();
}
```

19.7.1 Level Mode.

Figura 96 – Malha de Controle do Level Mode.



Fonte: Autoria Própria.

Pela análise das linhas do último código apresentado, chega-se a malha de controle da Figura 96. Nesta é possível observar a utilização de uma configuração multi-malha com um controlador PI-P e uma realimentação para a aplicação do SAS.

A referência aplicada ao sistema de controle é a angulação desejada. Este valor vem diretamente dos valores tratados do receptor, apresentados nas Figuras 75 e 76. Portanto, cada posição do stick referente ao canal de movimento, determina a angulação que o sistema ficará.

A realimentação para a malha de controle externa é feita através dos valores calculados e retornados pela IMU. Já a realimentação para as malhas internas é puramente dados do giroscópio gerados na equação [77]. Relembrando que tanto a realimentação externa, quanto a referência, estão na escala de 10x a angulação real.

O termo proporcional aplicado na malha interna, adicionando a ação de controle geral uma característica proporcional à derivada da saída, permite o controle mais eficiente da velocidade de resposta no regime transiente. Esse controle de velocidade geralmente é limitado apenas com o controlador PI da malha externa, pois aumenta-se a oscilação no regime transiente quando o ganho integral é muito alto, algo que acontece de maneira bem mais atenuada com o termo proporcional na malha interna.

O sistema de aumento de estabilidade, o SAS, por meio da taxa de medição do giroscópio em realimentação com um ganho, aumenta a taxa de amortecimento do sistema. Este funcionaria de maneira similar a um termo derivativo em uma malha

externa, porém, como trabalha com um valor da segunda derivada da saída, gera-se uma predição mais efetiva.

Os termos Proporcional e Integral do controlador da malha externa (PI) são saturados em determinado valor. Isto é feito para que não ocorra a saturação do atuador do sistema. Portanto funciona como um sistema ante Wind-up.

Os atuadores, os quatros motores, tem seu funcionamento administrado de acordo com o valor da ação de controle. Esta dinâmica de funcionamento dos quatros motores, será explicada ao final deste capítulo através da função “mixTable”.

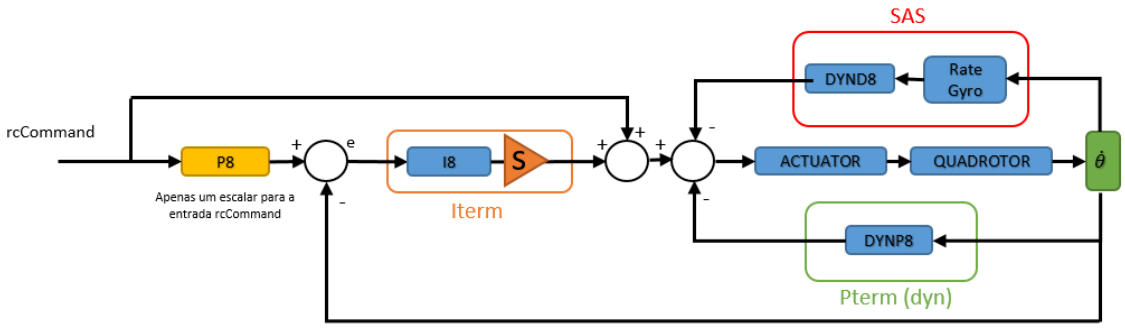
O funcionamento do Level Mode é restrito aos ângulos de arfagem e rolagem. Se este ativo, o ângulo de guinada é controlado pela malha de controle do modo Acro Mode.

De acordo com a Figura 96, tem-se:

- **P8LEVEL = P8[PIDLEVEL] =** Ganho proporcional do termo proporcional aplicado na malha externa;
- **D8LEVEL = D8[PIDLEVEL] =** Saturador do termo proporcional aplicado na malha externa;
- **I8LEVEL = I8[PIDLEVEL] =** Ganho integral do termo integrativo aplicado na malha externa;
- **S =** Saturador do termo integrativo aplicado na malha externa;
- **DYNP8 =** Ganho proporcional do termo proporcional aplicado na malha interna. Este ganho é calculado através da equação [129];
- **DYND8 =** Ganho do SAS. Este ganho é calculado através da equação [130];
- **Rate Gyro =** Cálculo da taxa de medição do giroscópio;
- **2rcCommand =** Referência do controlador;
- **angleTrim =** Ajuste fino da referência do controlador;
- $\boldsymbol{\theta}$ **=** Saída do sistema e realimentação da malha externa;
- $\dot{\boldsymbol{\theta}}$ **=** Saída do sistema e realimentação da malha interna.

19.7.2 Acro Mode

Figura 97 – Malha de Controle do Acro Mode.



Fonte: Autoria Própria.

A malha de controle do modo Acro Mode pode ser acessada de dois modos. Uma é quando o Level Mode está ativado, sendo necessário fazer o controle da angulação de guinada. Outra é quando o Level Mode está desativado, onde o controle dos três movimentos, arfagem, rolagem e guinada, são realizados na malha de controle do Acro Mode.

Através do último código apresentado, chega-se na elaboração da estrutura da malha de controle do modo Acro Mode, apresentada na Figura 97.

As malhas internas de controle funcionam de maneira totalmente similar ao Level Mode. A mudança está no controle da malha externa, e isto é devido ao dinamismo do movimento desejado neste modo.

Quando em Acro Mode, o stick do transmissor do rádio controle indica a intensidade de rotação desejada ao movimento do canal respectivo. Isso pode ser observado na malha externa de controle. Onde o erro é gerado por meio de uma referência (Figuras 75, 76 e 77) com um ganho de ajuste e a realimentação do giroscópio (equação [77]). Este erro é atuado apenas sobre um termo integrativo, que gera uma rotação contínua no eixo em controle de acordo com o erro acumulado e o incremento da ação de controle pela própria referência.

A IMU não é utilizada nesse modo de voo, há apenas a presença do giroscópio na realimentação da malha externa e interna.

De acordo com a Figura 97, tem-se:

- **P8 = P8[ROLL/PITCH/YAW] =** Ganho de ajuste da referência;
- **I8 = I8[ROLL/PITCH/YAW] =** Ganho integral do termo integrativo aplicado na malha externa;
- **S =** Saturador do termo integrativo aplicado na malha externa;
- **DYNP8 =** Ganho proporcional do termo proporcional aplicado na malha interna. Este ganho é calculado através da equação [129];
- **DYND8 =** Ganho do SAS. Este ganho é calculado através da equação [130];
- **Rate Gyro =** Cálculo da taxa de medição do giroscópio;
- **rcCommand =** Referência do controlador;
- $\dot{\boldsymbol{\theta}}$ **=** Saída do sistema e realimentação da malha interna e externa.

19.7.3 Mag Mode

Este modo acrescenta um controle proporcional na malha de controle do modo Acro mode no movimento de guinada. Portanto, este modo pode ser ativado em conjunto com o Level Mode ou Acro Mode.

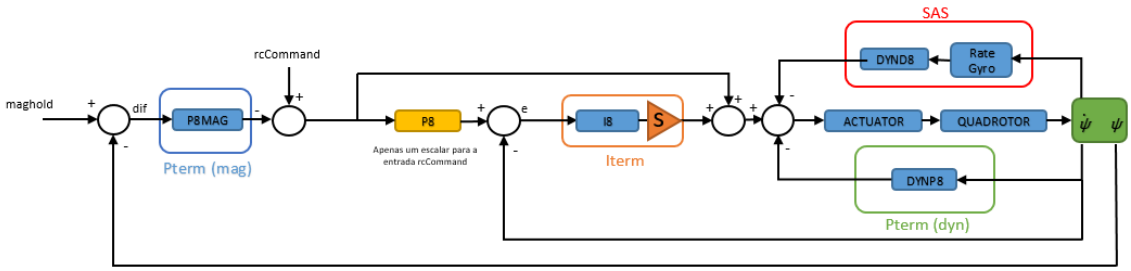
Quando o modo Mag Mode é ativado, este guarda o valor atual do ângulo de guinada na variável “magHold” como uma referência. De acordo com o bloco de código abaixo, quando o comando do movimento de guinada (Figura 77) for menor que 70, este valor de referência continua inalterado, porém, quando maior, esta referência é alterada até que o comando seja menor que 70 novamente.

Analisando o controle aplicado no código abaixo, chega-se na elaboração da estrutura de controle apresentada na Figura 98.

Level Mode / Acro Mode

```
#if MAG
    if (abs(rcCommand[YAW]) <70 && f.MAG_MODE) {
      int16_t dif = heading - magHold;
      if (dif <= - 180) dif += 360;
      if (dif >= + 180) dif -= 360;
      if ( f.SMALL_ANGLES_25 ) rcCommand[YAW] -=
dif*conf.P8[PIDMAG]/30;  // 18 deg
    } else magHold = heading;
  #endif
```

Figura 98 – Malha de Controle do Mag Mode.



Fonte: Autoria Própria.

É observável através dessa estrutura de controle, a geração de uma nova referência para estrutura de controle do Acro Mode (Figura 97), sendo parcela desta proporcional ao erro do ângulo de guinada de acordo com a referência (“magHold”) determinada.

Aqui é utilizada a angulação de guinada proveniente da IMU na realimentação da malha de controle mais externa.

Analisando a Figura 98, incrementa-se da Figura 97:

- **P8MAG = P8[MAG] =** Ganho proporcional do termo proporcional da malha mais externa;
- **Maghold =** Referência de ângulo de guinada;
- **Ψ =** Saída do sistema e realimentação da malha mais externa.

19.7.4 Altitude Hold Mode

Quando o modo Altitude Hold Mode é ativado, este guarda o valor da aceleração atual na variável “initialThrottleHold” e a estimativa de altura atual como referência na variável “AltHold”. O valor de aceleração guardado apenas será atualizado, se um novo comando de aceleração (Figuras 78 e 79) variar mais que uma margem pré-determinada. Este valor de aceleração somado à ação de controle, proveniente de uma malha de controle que tenha a altitude estimada (“AltHold”) como referência, e aplicado a aceleração dos quatros motores, é o algoritmo necessário para manter a altitude de um sistema quadrirrotor, conforme o trecho de código abaixo.

Baro Mode

```
  #if BARO
    if (f.BARO_MODE) {
      if (abs(rcCommand[THROTTLE]-
initialThrottleHold)>ALT_HOLD_THROTTLE_NEUTRAL_ZONE) {
        f.BARO_MODE = 0;       }
      rcCommand[THROTTLE] = initialThrottleHold + BaroPID;
    }
  #endif
```

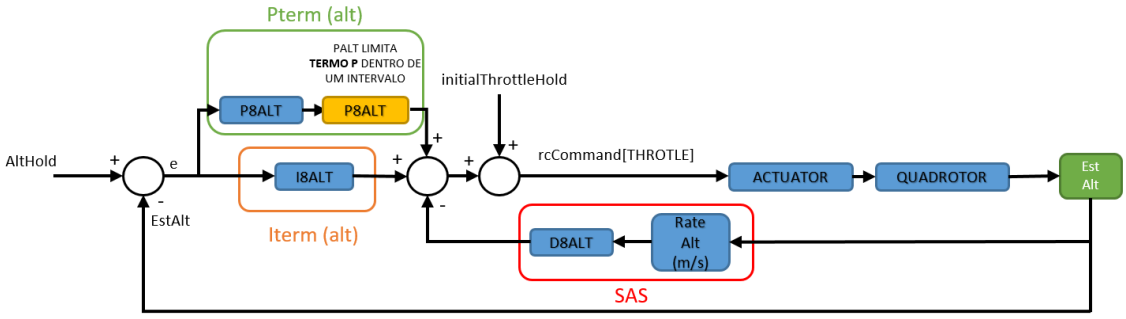
A variável “BaroPID” contém a ação de controle para a altitude de referência. Esta ação de controle é calculada de acordo com o trecho de código abaixo.

BaroPID

```
#define UPDATE_INTERVAL 25000
#define INIT_DELAY      4000000
#define BARO_TAB_SIZE   40
void getEstimatedAltitude(){
uint8_t index;
static uint32_t deadLine = INIT_DELAY;
static int16_t BaroHistTab[BARO_TAB_SIZE];
static int8_t BaroHistIdx;
static int32_t BaroHigh,BaroLow;
int32_t temp32;
int16_t last;
if (abs(currentTime - deadLine) < UPDATE_INTERVAL) return;
deadLine = currentTime;
last = BaroHistTab[BaroHistIdx];
BaroHistTab[BaroHistIdx] = BaroAlt/10;
BaroHigh += BaroHistTab[BaroHistIdx];
index = (BaroHistIdx + (BARO_TAB_SIZE/2))%BARO_TAB_SIZE;
BaroHigh -= BaroHistTab[index];
BaroLow  += BaroHistTab[index];
BaroLow  -= last;
BaroHistIdx++;
if (BaroHistIdx == BARO_TAB_SIZE) BaroHistIdx = 0;

BaroPID = 0;
temp32 = conf.D8[PIDALT]*(BaroHigh - BaroLow) / 40;
BaroPID-=temp32;
EstAlt = BaroHigh*10/(BARO_TAB_SIZE/2);
temp32 = AltHold - EstAlt;
if (abs(temp32) < 10 && abs(BaroPID) < 10) BaroPID = 0;

BaroPID += conf.P8[PIDALT]*constrain(temp32,(-
2)*conf.P8[PIDALT],2*conf.P8[PIDALT])/100;
BaroPID = constrain(BaroPID,-150,+150);

errorAltitudeI += temp32*conf.I8[PIDALT]/50;
errorAltitudeI = constrain(errorAltitudeI,-30000,30000);
temp32 = errorAltitudeI / 500;
BaroPID+=temp32;
}
```

Analisando o controle aplicado nos dois blocos de códigos acima, chega-se na elaboração da estrutura de controle apresentada na Figura 99.

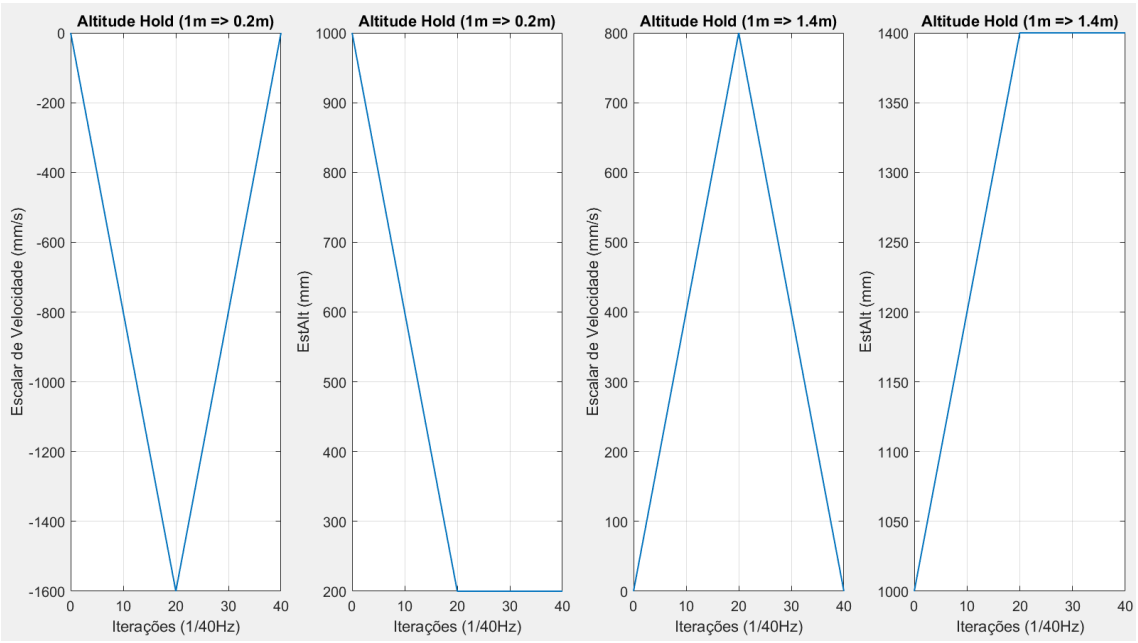
Figura 99 – Malha de Controle do Altitude Hold Mode.



Fonte: Autoria Própria.

Na Figura 99 é observável um controlador PI com uma malha interna contendo um SAS. O mais interessante no desenvolvimento dessa estrutura de controle é o algoritmo de cálculo da taxa de altura (Rate Alt), também responsável por calcular a altura estimada. A figura 100 demonstra a dinâmica deste cálculo para duas variações de altitude, uma de 1m para 0.2m, e outra de 1m para 1.4m.

Figura 100 – Cálculo de velocidade e altura estimada.



Fonte: Autoria Própria.

De acordo com a Figura 99, tem-se:

- **P8ALT = P8[PIDALT] =** Ganho proporcional do termo proporcional da malha externa;
- **I8ALT = I8[PIDALT] =** Ganho integral do termo integrativo da malha externa;
- **D8ALT = D8[PIDALT] =** Ganho do SAS;
- **AltHold =** Referência do controlador;
- **EstAlt =** Saída do sistema e realimentação da malha externa e interna.

19.7.5 mixTabble

Uma vez calculada a ação de controle para cada um dos movimentos (arfagem, rolagem e guinada), independentemente do modo de voo aplicado, deve-se administrar os 4 atuadores de acordo com a orientação do quadrirrotor, que neste caso é uma estrutura em X (Figura 35).

A função mixTabble descreve a dinâmica dos atuadores para cada movimento do quadrirrotor. Nela pode-se observar que:

$$rcCommand[Throttle] + axisPID[ROLL] \times X + axisPID[PITCH] \times Y + axisPID[YAW] \times Z \quad [147]$$

Para o motor traseiro da direita, aplica-se em [147]:

$$rcCommand[Throttle] - axisPID[ROLL] + axisPID[PITCH] - axisPID[YAW] \quad [148]$$

Para o motor frontal da direita, aplica-se em [147]:

$$rcCommand[Throttle] - axisPID[ROLL] - axisPID[PITCH] + axisPID[YAW] \quad [149]$$

Para o motor traseiro da esquerda, aplica-se em [147]:

$$rcCommand[Throttle] + axisPID[ROLL] + axisPID[PITCH] + axisPID[YAW] \quad [150]$$

Para o motor frontal da esquerda, aplica-se em [147]:

$$rcCommand[Throttle] + axisPID[ROLL] - axisPID[PITCH] - axisPID[YAW] \quad [151]$$

Onde, $rcCommand[Throttle]$ é o comando de aceleração (Figuras 78 e 79) e $axisPID$ é a ação de controle, tendo uma para cada movimento.

De acordo com o valor aplicado para cada motor nas equações [148], [149], [150] e [151] atualizam-se os contadores que administram a largura do pulso PWM enviado aos ESCs dos motores.

mixTabble

```
void mixTable() {
  int16_t maxMotor;
  uint8_t i;

  #define PIDMIX(X,Y,Z) rcCommand[THROTTLE] + axisPID[ROLL]*X + axi
sPID[PITCH]*Y + YAW_DIRECTION * axisPID[YAW]*Z

  axisPID[YAW] = constrain(axisPID[YAW],-100-
abs(rcCommand[YAW]),+100+abs(rcCommand[YAW]));

  motor[0] = PIDMIX(-1,+1,-1);   //REAR_R
  motor[1] = PIDMIX(-1,-1,+1);   //FRONT_R
  motor[2] = PIDMIX(+1,+1,+1);   //REAR_L
  motor[3] = PIDMIX(+1,-1,-1);   //FRONT_L

  maxMotor=motor[0];

  for(i=1;i< NUMBER_MOTOR;i++){
    if (motor[i]>maxMotor) maxMotor=motor[i];}

  for (i = 0; i < NUMBER_MOTOR; i++) {
    if (maxMotor > MAXTHROTTLE)
      motor[i] -= maxMotor - MAXTHROTTLE;
    motor[i] = constrain(motor[i], MINTHROTTLE, MAXTHROTTLE);
    if ((rcData[THROTTLE]) < MINCHECK)
      motor[i] = MINTHROTTLE;
    if (!f.ARMED)
      motor[i] = MINCOMMAND;
  }
}
```

Ainda na função “mixTabble” ocorre a limitação do pulso PWM de cada motor. A saturação ocorre no intervalor [MINTHROTTLE, MAXTRHOTTLE].

Também é configurada a rotação dos motores para quando o quadrirrotor não está armado, e para quando o quadrirrotor está armado, porém com comando de aceleração mínimo.

20 RESULTADOS

Utilizando a estrutura para sintonizar o quadrirrotor (Figura 23), através de tentativas de diferentes valores de ganhos com auxílio gráfico para monitorar a estabilidade, através da troca de dados pela comunicação serial, chegou-se nas configurações apresentadas na Tabela 34. Nesta estão dispostos os valores apresentados na GUI, o valor que é realmente lido na EEPROM, o escalar de modificação entre GUI-EEPROM, o valor dentro da malha de controle e o escalar para modificação entre EEPROM-controle.

Tabela 34 – Configurações sintonizadas.

	GUI	EEPROM(a)	scale	controle	scale
P8[ROLL]	4	40	10	2	80/a
PDYN[ROLL]	4	40	10	0.5	1/80
I8[ROLL]	0.030	30	1000	0.00375	1/8000
DDYN[ROLL]	23	23	1	0.71875	1/32
P8[PITCH]	4	40	10	2	80/a
PDYN[PITCH]	4	40	10	0.5	1/80
I8[PITCH]	0.030	30	1000	0.00375	1/8000
DDYN[PITCH]	23	23	1	0.71875	1/32
P8[YAW]	8.5	85	10	0.9411	80/a
PDYN[YAW]	8.5	85	10	1.0625	1/80
I8[YAW]	0.045	45	1000	0.005625	1/8000
DDYN[YAW]	0	0	1	0	1/32
P8[PIDLEVEL]	7	70	10	0.7	1/100
I8[PIDLEVEL]	0.01	10	1000	0.00244	1/2^12
D8[PIDLEVEL]	100	100	1	100	1
P8[MAG]	4	40	10	1.3333	1/30
thrMid8	0.50	50	100	-	-
thrExpo8	0.00	0	100	-	-
rcRate	0.90	90	100	-	-

rcExpo	0.65	65	100	-	-

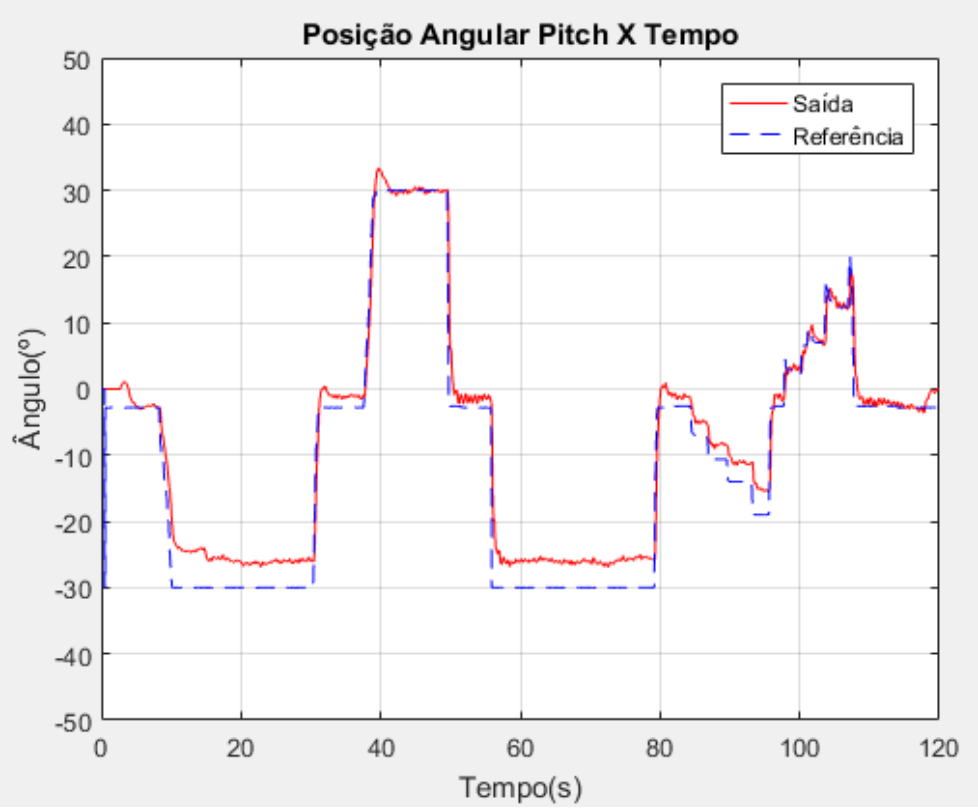
Fonte: Autoria Própria.

Foi desenvolvido, a partir do protocolo de comunicação serial (Tabela 12), uma aquisição de dados para monitorar a resposta do controle para determinadas configurações (análise da estabilidade). Foi utilizada a plataforma do Simulink para receber os dados, trata-los e apresenta-los. O procedimento utilizado está presente no APÊNDICE F.

A estrutura de calibração limita a aquisição de dados apenas para os ângulos de arfagem e rolagem, e são esses, de acordo com a configurações da Tabela 34, que serão apresentados nos modos Level Mode e Acro Mode.

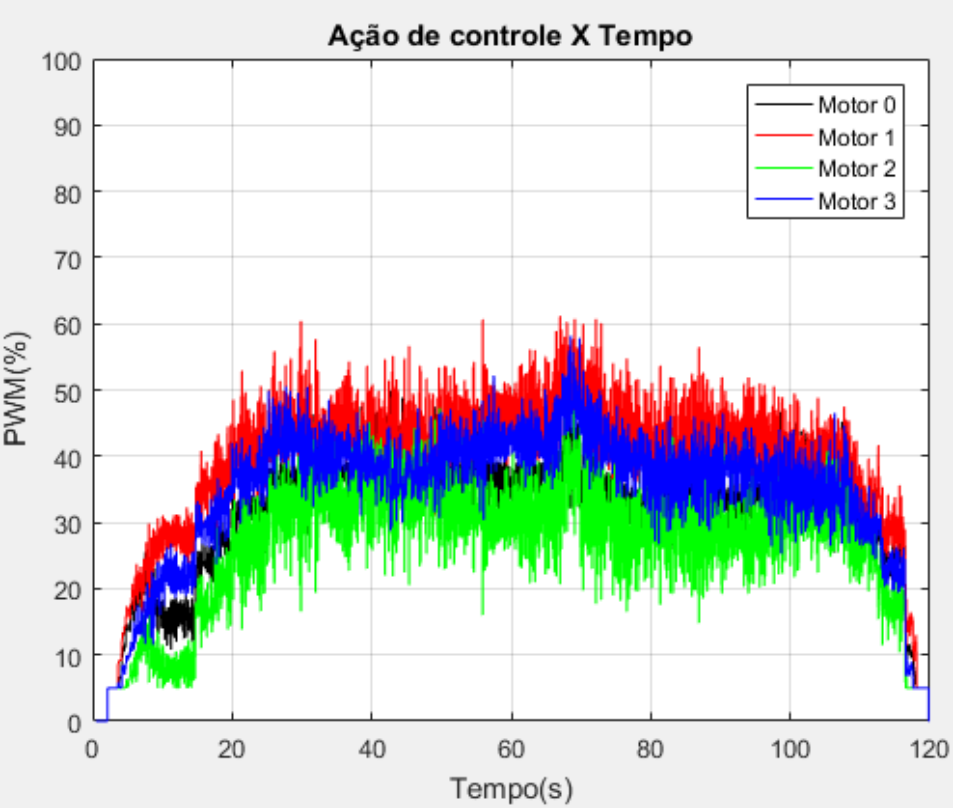
20.1 Level Mode

Figura 101 – Resposta no Level Mode para o ângulo de arfagem.



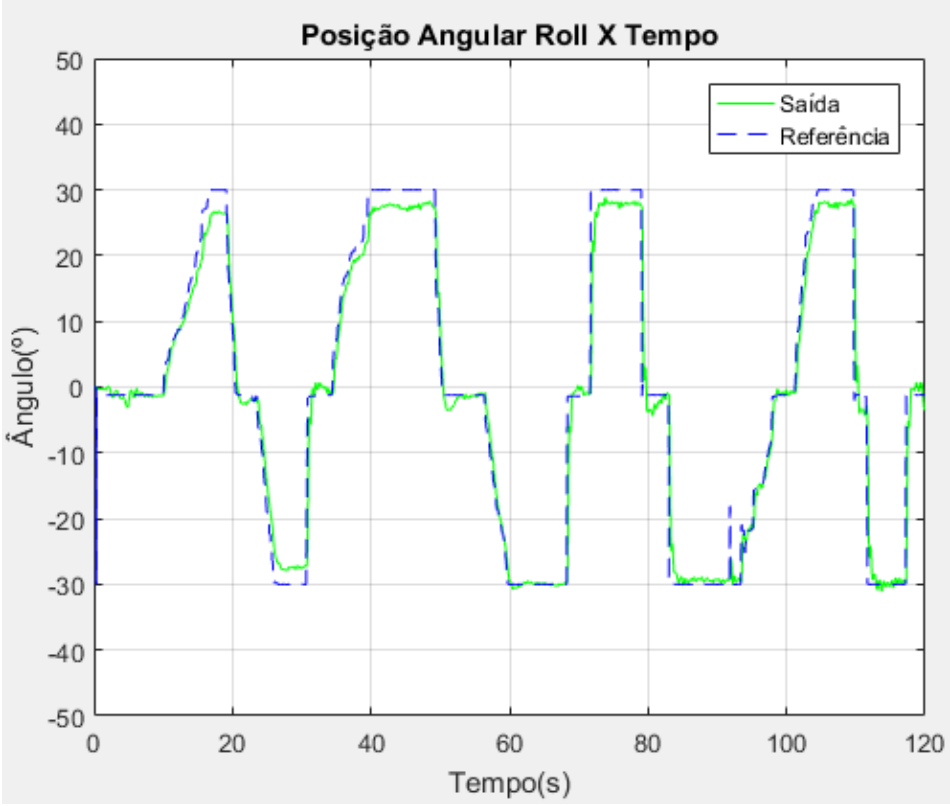
Fonte: Autoria Própria.

Figura 102 – Ação de controle no Level Mode para o ângulo de arfagem.



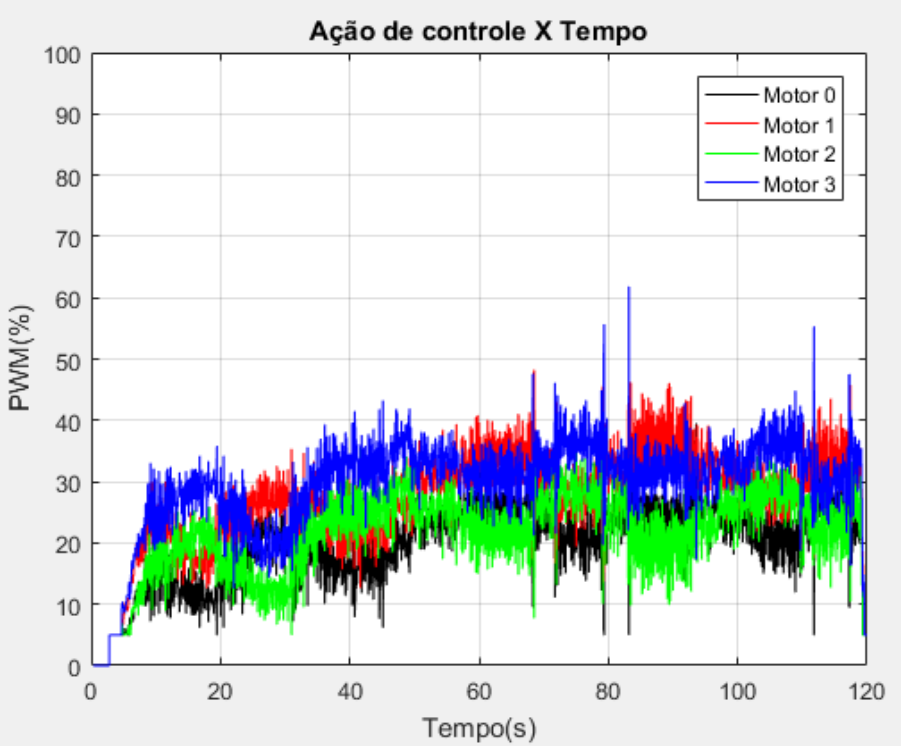
Fonte: Autoria Própria.

Figura 103 – Resposta no Level Mode para o ângulo de rolagem.



Fonte: Autoria Própria.

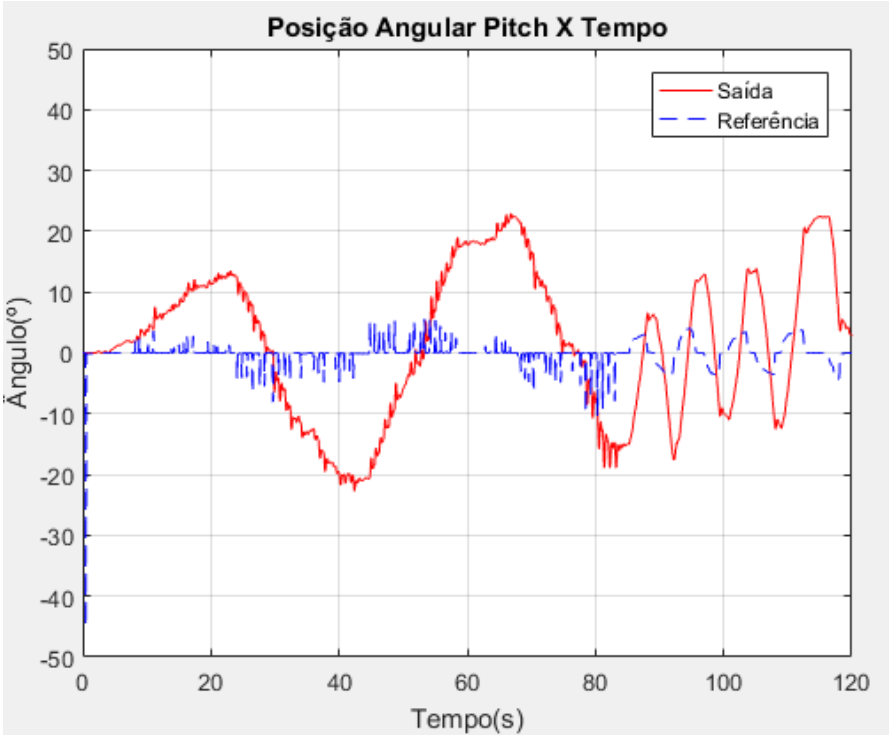
Figura 104 – Ação de controle no Level Mode para o ângulo de arfagem.



Fonte: Autoria Própria.

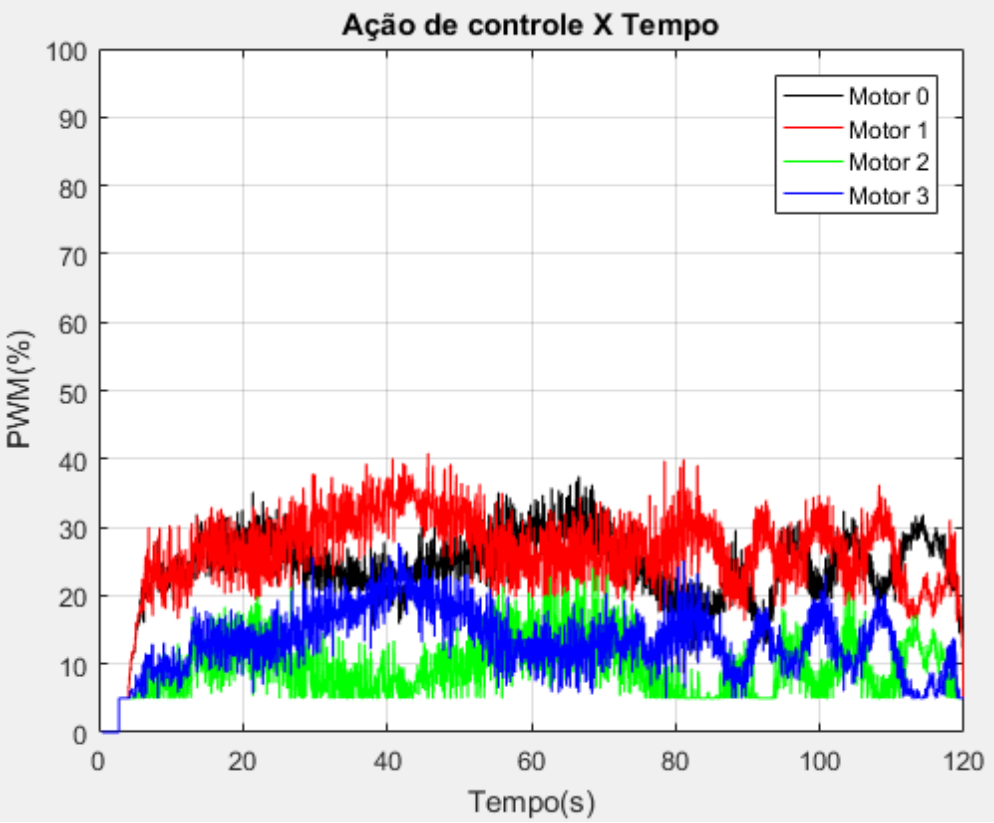
2.1 Acro Mode

Figura 105 – Resposta no Acrol Mode para o ângulo de arfagem



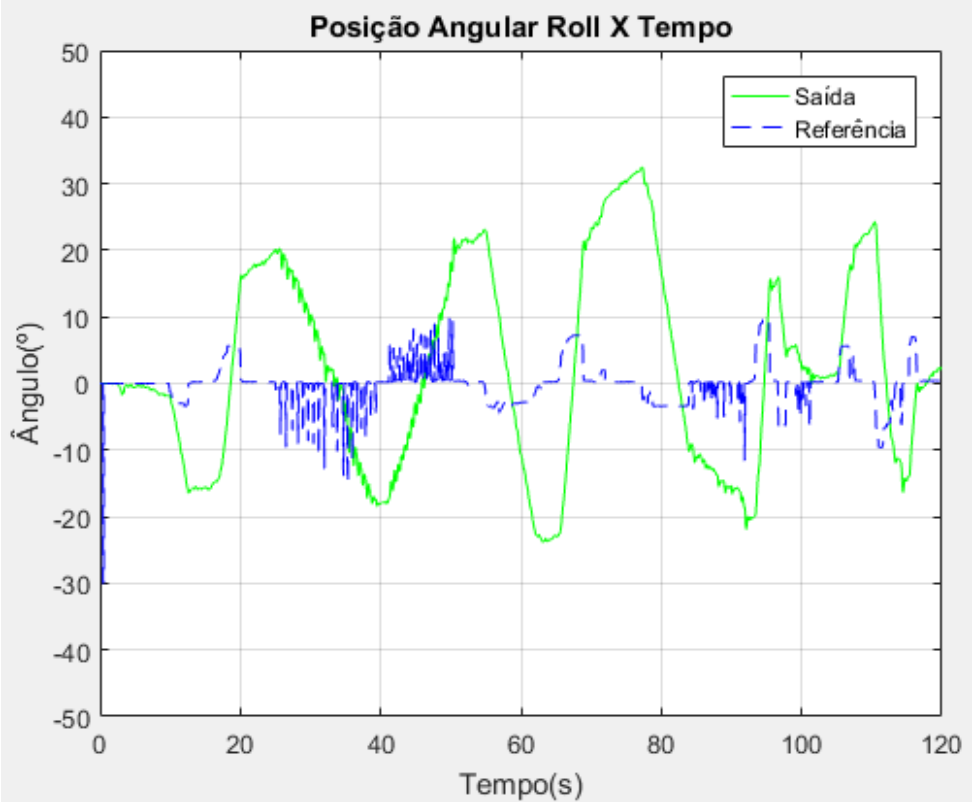
Fonte: Autoria Própria.

Figura 106 – Ação de controle no Acro Mode para o ângulo de arfagem.



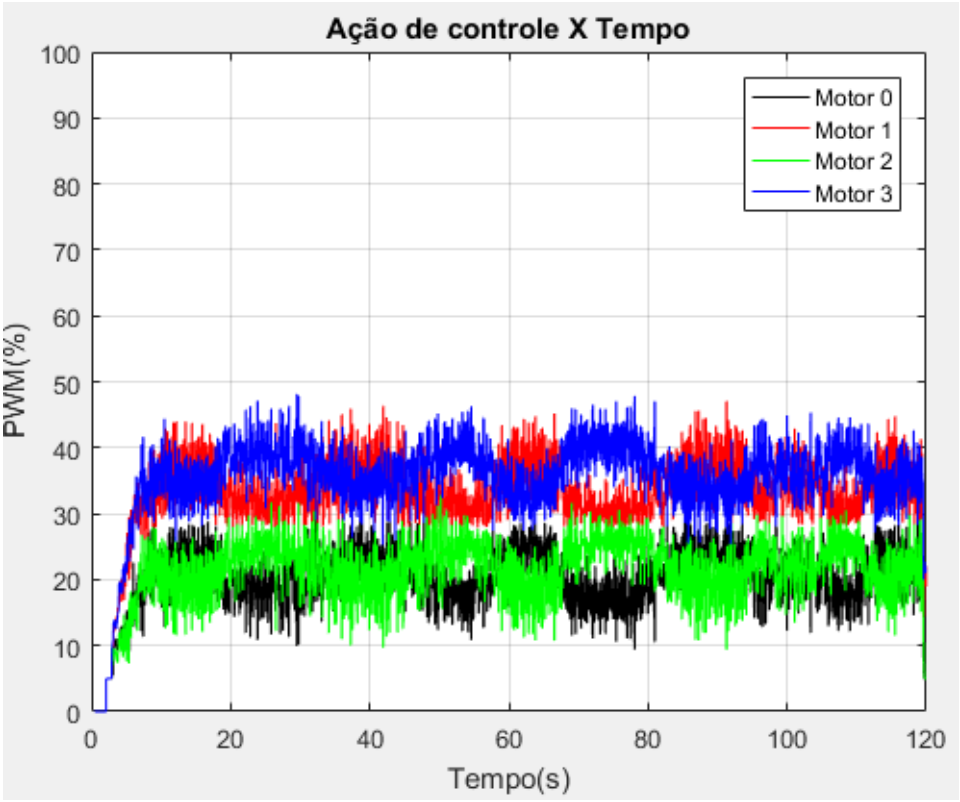
Fonte: Autoria Própria.

Figura 107 – Resposta no Acrol Mode para o ângulo de rolagem.



Fonte: Autoria Própria.

Figura 108 – Ação de controle no Acro Mode para o ângulo de rolagem.



Fonte: Autoria Própria.

21 CONCLUSÃO

Tendo apresentado todas as abordagens tomadas, de forma detalhada na placa embarcada, torna-se possível a continuidade desta pesquisa, buscando melhores resultados, podendo analisar cada setor de desenvolvimento apresentado e tomar decisões para uma análise, buscando diferente performances.

Fica pronta, também, uma estrutura para testes de diferentes configurações de malhas de controle, desde a estrutura física para sintonia, até o firmware detalhado, facilitando a modificação e aplicação de uma nova configuração de controle.

Dos resultados apresentados, fica visível uma excelente resposta nas transições de referências, porém em regime permanente é apresentado um offset em apenas um dos extremos de referência (Figuras 101 e 103). Teoricamente isso poderia ser atenuado aumentando o ganho integrativo da malha externa do controlador, porém, na prática, para evitar maiores oscilações, é necessário incrementar um termo integrativo na malha interna do controlador, trabalhando então com a derivada da saída. Por questão de processamento e simplicidade na sintonia, já que as transições se comportaram com excelência, o MultiWii não apresenta esse termo para calibração.

REFERÊNCIAS

AIRFORCE-TECHNOLOGY. **Global Hawk High-Altitude, Long Endurance.** Disponível em: http://www.airforce-technology.com/projects/rq4-global-hawk-uav/. Acesso em: outubro de 2018

AIRFORCE-TECHNOLOGY. **Predator RQ-1 / MQ-1 / MQ-9**. Disponível em: http://www.airforce-technology.com/projects/predator/. Acesso em: outubro de 2018.

ARAÚJO, F. M. U. **Sistemas de Controle**. Universidade Federal do Rio Grande do Norte – UFRN. 101p Natal – RN 2007.

ASTROM, K.; HÃGGLUND, T. **PID Controllers: Theory, Design, and Tuning**. 2 Ed. Research Triangle Park: Instrument Society of America, 1995.

BARANEK, R.; SOLC, F. **Tuning of complementary filter attitude estimator using precise model of multicopter.** Electroscope, 6p, 2013.

BAYER, F. M.; ARAÚJO, O. C. B**. Controle Automático de Processos**. Escola Técnica Aberta do Brasil – e-Tec Brasil. 92p Santa Maria – RS 2011.

BORSOI, B. T.; FAVARIM, F; LINARES, K. C.; MATTIELLO, C. D. **Controle de atitude para veículos aéreos não tripulados do tipo quadricóptero: PID vs Lógica Fuzzy.** Computer on the Beach, p 111-120, 2015.

BOSCH, Datasheet: **BMP085 Digital pressure sensor**. Eletronic Publication, 2009.

BOSCH, Datasheet: BMA180 Digital, triaxial acceleration sensor. Eletronic Publication, 2010.

BOUABDALLAH, S. **Design and control of quadrotors with application to autonomous flying.** PhD Thesis. École Polytechnique Fédérale de Lausanne. Lausanne 115p. 2007.

BRAGA, M. A; SANTANA, P. H. de R. Q. e A. **Concepção de um veículo aéreo não-tripulado do tipo quadrirrotor.** Trabalho de Graduação (Engenharia Mecatrônica) Universidade de Brasília. 164p – DF 2008.

COSTA, E. B. **Algoritmos de controle aplicados a estabilização de voo de um quadrotor.** Dissertação (Engenharia Elétrica – Sistemas de Energia). Universidade Federal de Juiz de Fora. 133p. Juiz de Fora – MG 2012.

CRAIG, J. J. **Robótica**. 3. Ed. São Paulo: Pearson, 2013.

CUONG, C. Q.; SAHA, B.; KOSHIMOTO, E.; HOGGE, E. F.; STROM, T. H.; HILL, B. L.; VASQUEZ, S. L.; GOEBEL, K. **Battery Health Management System for Electric UAVs**. IEEE Aerospace Conference, p 1–9, 2011.

DA SILVA, L. P. **Projeto e construção de um quadricóptero remotamente pilotado e do seu sistema de determinação e controle de atitude**. Monografia (Engenharia de Controle e Automação) Instituto Federal de Educação – IFE. 109p Campos Goytacazes 2011.

D'AVILA, C. E. P.; GERTZ, L. C.; SILVEIRA, M. A. da; CERVIEIRI, A. **Estudo de um motor CC brushless aplicado no acionamento de um carro elétrico de pequeno porte**. Liberato, p 107-206, 2011.

GENERAL ATOMICS. **MQ-1 Predator B.** Disponível em: http://www.ga-asi.com/aircraft-platforms-library. Acesso em: outubro de 2018.

GULIAEV, N. **The design, construction and implementation of an autonomous outdoor quadcopter using an RPI microcomputer and a MultiWii flight controller.** PhD Thesis (Mechanical Engineering) Saimaa University of Applied Sciences. 50p. 2017.

HASTINGS JR, C. **Approximations for Digital Computers.** New Jersey: Princeton University Press, 1955.

HENDERSHOT JUNIOR, J. R.; MILLHER, T. J. T. **Design of Brushless Permanent-Magnet Motors**. 1. Ed. New York: Oxford University Press, 1994.

HOBBYKING: **How Brushed and Brushless Eletronic Speed Controllers Work.** Disponível em: https://hobbyking.com/en_us/news/brushed-brushless-electronic-speed-controllers-work?___store=en_us. Acesso em : Maio de 2019.

HOMA, J. **Aerodinâmica e teoria de voo**. 28. Ed. São Paulo: ASA, 2003

Honeywell, Datasheet: **3-Axis Digital Compass IC HMC5883L**. Electronic Publication, 2010

HYSTAD, A. V. **Model, Design and Controle of a Quadcopter**. Master of Science in Cybernetics and Robotics (Engineering Cybernetics) Norwegian University of Science and Technology –NTNU. 181p, 2015.

InvenSens, Datasheet: **ITG-3200 Product Speification**. Eletronic Publication, 2010.

ISLAM, T.; ISLAM, Md. S.; SHAJID-UI-MAHMUD, Md.; HOSSAM-E-HAIDER, Md. **Comparison of complementary and Kalman filter based data fusion for attitude heading reference system.** AIP Conference Proceedings 1919, 020002, 10p, 2017.

KIM, H. J.; LEE, D.; LIM, B. H.; PARK, J. **Build your own quadrotor.** IEE Robotics & Automation Magazine, p 33-45, 2012.

LIMA, C. B. de; VILLAÇA, M. V. **AVR e arduino: técnicas de projeto.** 2. Ed. Florianópolis: Ed. Dos autores, 2012.

LISBOA, L. P. **Controlador não-linear para veículo aéreo não tripulado.** Dissertação de Mestrado (Engenharia Elétrica) Pontifícia Universidade do Rio Grande do Sul – PUCRS. 84p Porto Alegre – RS 2014.

LOTUFO, F. A. Material didático – **Análise e Controle de Sistemas Dinâmicos.** UNESP/Guaratinguetá (Departamento de Engenharia Elétrica), 2013.

MAHONY, R.; HAMEL, T.; PFLIMLIN, JM. **Nonlinear Complementary Filters on the Special Orthogonal Group.** IEEE Transactions on Automatic Control, hal-00488379, 17p, 2010.

MOECKE, M. Telegonia Digital: Modulação por código de Pulso. Curso Técnico em Telecomunicações (CEFET-SC). 45p São José – SC 2004.

MOURA, R. L. **O uso de microcontroladores no acionamento e controle de motores Brushless DC**. Trabalho de Graduação (Engenharia Elétrica – Sistemas de Energia e Automação) Escola de Engenharia de São Carlos - USP. 40p São Carlos - SP 2010.

NELSON, R. C. **Flight Stability and Automatic Control.** 2 Ed.B Singapore: McGraw-Hill Book Co, 1998.

OGATA, K. Engenharia de controle moderno. 5 Ed. São Paulo: Pearson Education do Brasil, 2011

PAULA, J. C. de. **Desenvolvimento de um VANT do tipo quadrirrotor para obtenção de imagens aéreas em alta definição.** Dissertação de Mestrado (Engenharia Elétrica) Universidade Federal do Paraná – UFPR. 110p Curitiba – PR 2012.

PEDLEY, M. **Tilt sensing using a three-axis accelerometer.** Freescale semiconductor, AN3461, 22p, 2013.

RAMOS, J. V. S.; PUGLIESE, L. F.; GUARACY, F. H. D. **Estimação de atitude pelo filtro complementar passivo não linear com adaptação de ganho via lógica Fuzzy.** XIII Simpósio Brasileiro de Automação Inteligente, p418-423, 2017.

SCOFANO, F. S. **Desenvolvimento de um sistema de controle remoto para um robô de combate**. Trabalho de Graduação (Engenharia de Controle e Automação) Pontifícia Universidade do Rio de Janeiro – PUCRIO. 117p Rio de Janeiro – RJ 2003.

SENSOR OLINE. **Compensating for Tilt, Hard-Iron, and Soft-Iron Effects**. Disponível em: https://www.sensorsmag.com/components/compensating-for-tilt-hard-iron-and-soft-iron-effects. Acesso em: Maio de 2019.

STARLINO ELETRONICS. **A guide to using IMU (Accelerometer and Gyroscope devices) in embedded applications**. Disponível em: http://www.starlino.com/imu_guide.html. Acesso em: Maio de 2019.

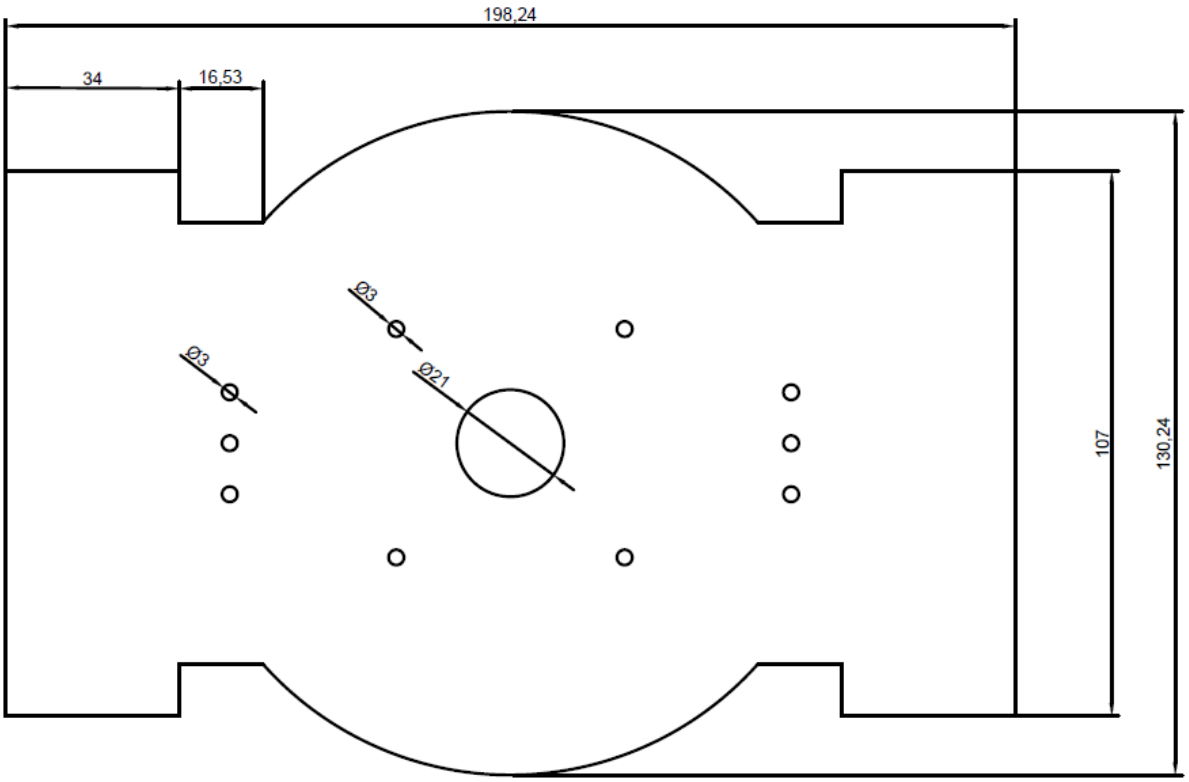
TRINDADE, R. H. **Estudo de máquinas elétricas não-convencionais: Motor Brushless DC**. Trabalho de Graduação (Engenharia Elétrica – Sistemas de Energia e Automação) Escola de Engenharia de São Carlos – USP. 39p São Carlos – SP 2009.

STEVENS, B. L.; LEWIS, F. L. **Aircraft Control and Simulation**. 1Ed. New York: John Wiley & Sons, 1992

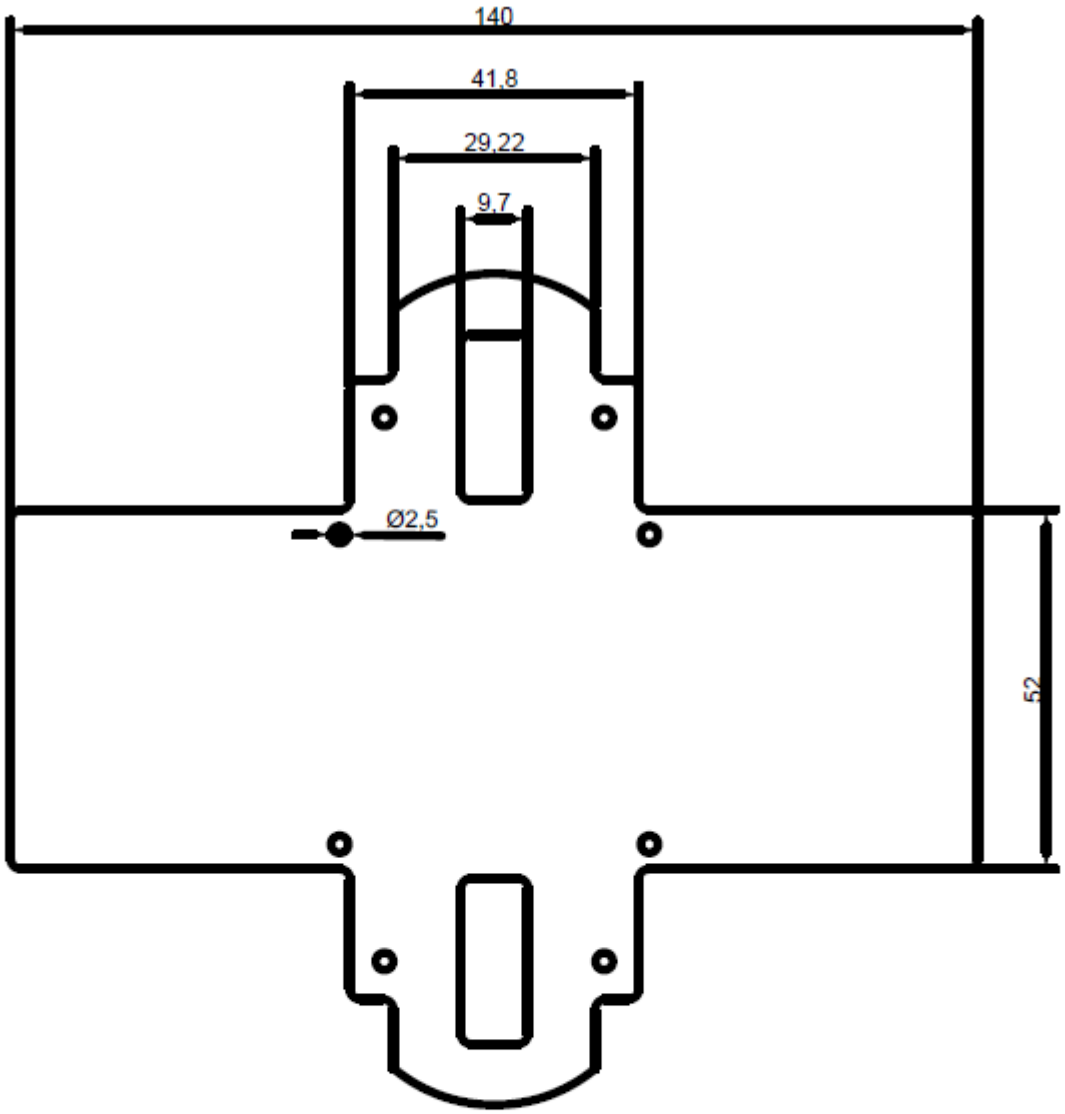
SUNADA, M. M. **Acionamento de um posicionador linear de ultraprecisão empregando uma redução harmonic drive com controle de velocidade**. Dissertação de mestrado (Engenharia Mecânica) Universidade Federal de Santa Catarita – UFSC. 162p Florianópolis – SC 2007.

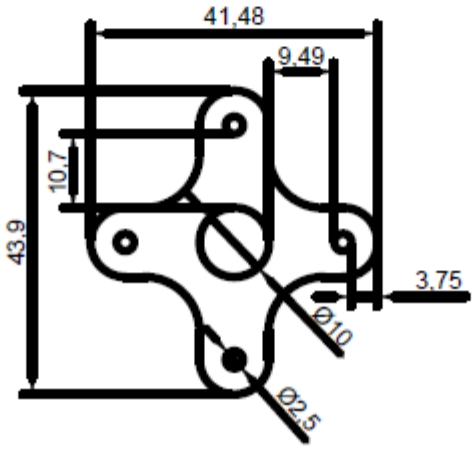
THUMS, G. D.; TORRES, L. A. B.; PALHARES, R. M. **Metodologia de sintonia PID Multi-Malha para veículos aéreos não tripulados: Dinâmica longitudinal.** Anais do XIX Congresso Brasileiro de Automática – CBA. p358-365, 2012.

APÊNDICE A – Quadrirrotor X

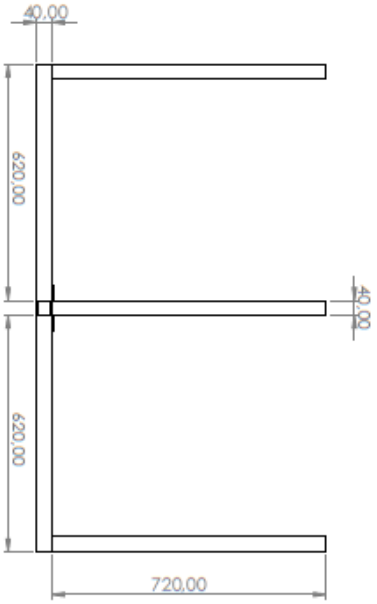

198,24
34
16,53
Ø3
Ø3
Ø21
107
130,24

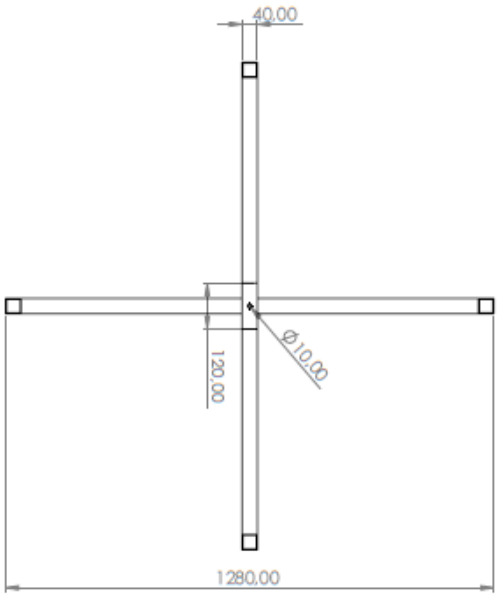
APÊNDICE B – Quadrirrotor +

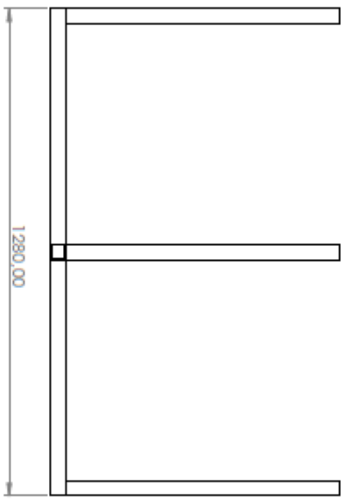

140
41,8
29,22
9,7
Ø2,5
52

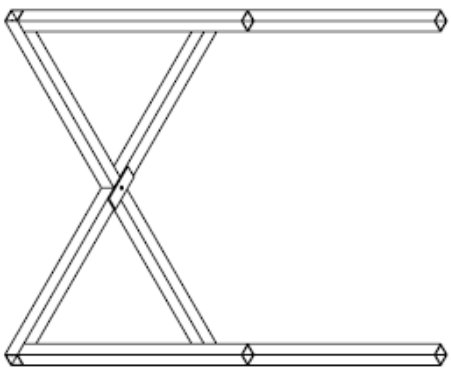

41,48
9,49
10,7
43,9
3,75
Ø10
Ø2,5

APÊNDICE C – Estrutura


40.00
620.00
40.00
620.00
720.00


40.00
120.00
Ø10.00
1280.00


1280.00



APÊNDICE D – Algoritmos Matlab (rcCommand)

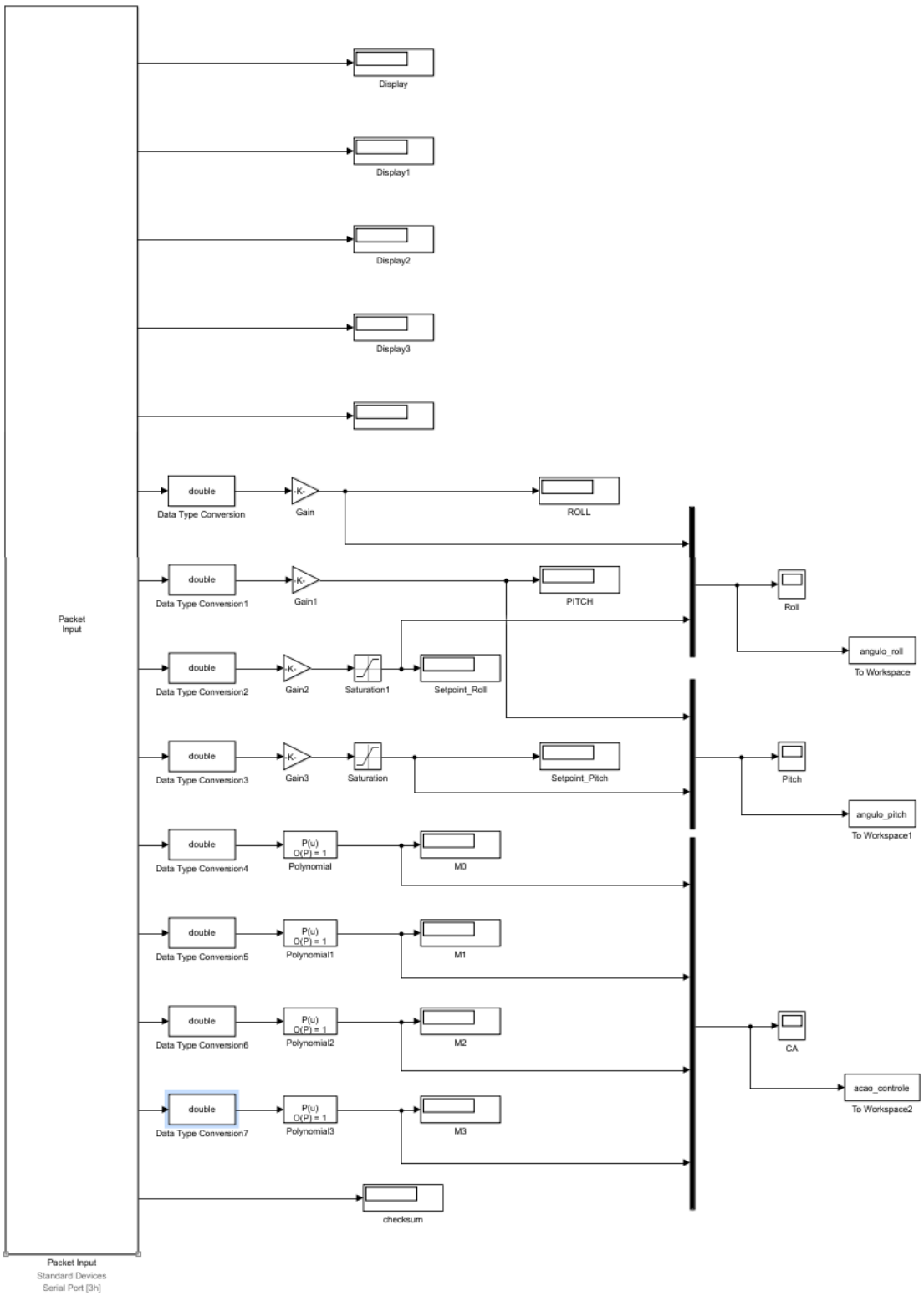
```
%PITCH AND ROLL

x = 0:1:5;

%expo=60;rate=90
k =((2500 + 60*(x.*x-25)).*(x*90/2500));
%expo=60;rate=60
Z =((2500 + 60*(x.*x-25)).*(x*60/2500));
%expo=60;rate=30
J =((2500 + 60*(x.*x-25)).*(x*30/2500));

%expo=60;rate=90
A =((2500 + 60*(x.*x-25)).*(x*90/2500));
%expo=90;rate=90
B =((2500 + 90*(x.*x-25)).*(x*90/2500));
%expo=30;rate=90
C =((2500 + 30*(x.*x-25)).*(x*90/2500));

subplot(1,2,1)
plot(x,k,'b',x,Z,'r',x,J,'g','linewidth',1.5)
title('Variando RATE','FontSize',14);
xlabel('Intensidade de Comando do Stick');
ylabel('Entrada da malha (graus*10)')
legend('expo=60;rate=90','expo=60;rate=60','expo=60;rate=30');
grid on
set(gca,'FontSize',15)

subplot(1,2,2)
plot(x,A,'b',x,B,'r',x,C,'g','linewidth',1.5)
title('Variando EXPO','FontSize',14);
xlabel('Intensidade de Comando do Stick');
ylabel('Entrada da malha (graus*10)')
legend('expo=60;rate=90','expo=90;rate=90','expo=30;rate=90');
grid on
set(gca,'FontSize',15)


%YAW

x = 0:1:5;

%expo=0;rate=100
k =((2500 + 0*(x.*x-25)).*(x*100/2500));

subplot(1,2,1)
plot(x,k,'b','linewidth',1.5)
title('Tratamento dos dados do sinal de comando (YAW) do sistema
transmissor/receptor','FontSize',13);
xlabel('Intensidade de Comando do Stick');
ylabel('Entrada da malha (graus*10)')
legend('linear');
grid on
set(gca,'FontSize',15)


%THROTTLE
x = 0:1:10;

%mid=50;exp=77
```

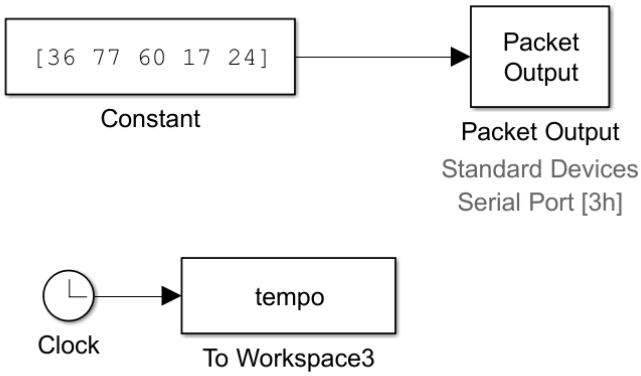
```
tmp = 10.*x - 50;
y = [50 50 50 50 50 1 50 50 50 50 50]
z = y.*y
n = tmp.*tmp
k = 10*50 + (tmp/10).*( 100-77 + 77*(n./z));
j = 1100 + 900*k/1000;

%mid=40;exp=77
tmp2 = 10.*x - 40;
y2 = [40 40 40 40 1 60 60 60 60 60 60]
z2 = y2.*y2
n2 = tmp2.*tmp2
k2 = 10*40 + (tmp2/10).*( 100-77 + 77*(n2./z2));
j2 = 1100 + 900*k2/1000;

%mid=30;exp=77
tmp3 = 10.*x - 30;
y3 = [30 30 30 1 70 70 70 70 70 70 70]
z3 = y3.*y3
n3 = tmp3.*tmp3
k3 = 10*30 + (tmp3/10).*( 100-77 + 77*(n3./z3));
j3 = 1100 + 900*k3/1000;

%mid=50;exp=77
k4 = 10*50 + (tmp/10).*( 100-77 + 77*(n./z));
j4 = 1100 + 900*k4/1000;

%mid=50;exp=47
k5 = 10*50 + (tmp/10).*( 100-77 + 47*(n./z));
j5 = 1100 + 900*k5/1000;

%mid=50;exp=27
k6 = 10*50 + (tmp/10).*( 100-77 + 27*(n./z));
j6 = 1100 + 900*k6/1000;


subplot(1,2,1)
plot(x,j,x,j2,x,j3,'linewidth',1.5)
title('Variando MID','FontSize',14);
xlabel('Intensidade de Comando do Stick');
ylabel('Potência desejada (Pulso PWM (us))')
legend('mid=50;exp=77','mid=40;exp=77','mid=30;exp=77');
grid on
set(gca,'FontSize',15)

subplot(1,2,2)
plot(x,j4,x,j5,x,j6,'linewidth',1.5)
title('Variando EXP','FontSize',14);
xlabel('Intensidade de Comando do Stick');
ylabel('Potência desejada (Pulso PWM (us))')
legend('mid=50;exp=77','mid=50;exp=47','mid=50;exp=27');
grid on
set(gca,'FontSize',15)
```

APÊNDICE E – Altitude Hold Mode

```
x = 0:1:40;
y = [0 -80 -160 -240 -320 -400 -480 -560 -640 -720 -800 -880 -960 -1040 -
1120 -1200 -1280 -1360 -1440 -1520 -1600 -1520 -1440 -1360 -1280 -1200 -
1120 -1040 -960 -880 -800 -720 -640 -560 -480 -400 -320 -240 -160 -80 0]

subplot(1,4,1)
plot(x,y,'linewidth',1.5)
title('Altitude Hold (1m => 0.2m)','FontSize',14);
xlabel('Iterações (1/40Hz)');
ylabel('Escalar de Velocidade (mm/s)')
grid on
set(gca,'FontSize',15)


x3 = 0:1:40;
y3 = [1000 960 920 880 840 800 760 720 680 640 600 560 520 480 440 400 360
320 280 240 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200 200
200 200 200 200 200 ];
subplot(1,4,2)
plot(x3,y3,'linewidth',1.5)
title('Altitude Hold (1m => 0.2m)','FontSize',14);
xlabel('Iterações (1/40Hz)');
ylabel('EstAlt (mm)')
grid on
set(gca,'FontSize',15)


x1 = 0:1:40;
y1 = [0 40 80 120 160 200 240 280 320 360 400 440 480 520 560 600 640 680
720 760 800 760 720 680 640 600 560 520 480 440 400 360 320 280 240 200 160
120 80 40 0]

subplot(1,4,3)
plot(x1,y1,'linewidth',1.5)
title('Altitude Hold (1m => 1.4m)','FontSize',14);
xlabel('Iterações (1/40Hz)');
ylabel('Escalar de Velocidade (mm/s)')
grid on
set(gca,'FontSize',15)


x2 = 0:1:40;
y2 = [1000 1020 1040 1060 1080 1100 1120 1140 1160 1180 1200 1220 1240 1260
1280 1300 1320 1340 1360 1380 1400 1400 1400 1400 1400 1400 1400 1400 1400
1400 1400 1400 1400 1400 1400 1400 1400 1400 1400 1400 1400];
subplot(1,4,4)
plot(x2,y2,'linewidth',1.5)
title('Altitude Hold (1m => 1.4m)','FontSize',14);
xlabel('Iterações (1/40Hz)');
ylabel('EstAlt (mm)')
grid on
set(gca,'FontSize',15)
```

APÊNDICE F – Simulink (Resultados)


Packet Input
Display
Display1
Display2
Display3
double
Data Type Conversion
-K-
Gain
ROLL
Data Type Conversion1
Gain1
PITCH
Roll
angulo_roll
To Workspace
Data Type Conversion2
Gain2
Saturation1
Setpoint_Roll
Data Type Conversion3
Gain3
Saturation
Setpoint_Pitch
Pitch
angulo_pitch
To Workspace1
Data Type Conversion4
P(u)
O(P) = 1
Polynomial
M0
Data Type Conversion5
Polynomial1
M1
Data Type Conversion6
Polynomial2
M2
CA
acao_controle
To Workspace2
Data Type Conversion7
Polynomial3
M3
checksum
Packet Input
Standard Devices
Serial Port [3h]




```
%% parâmetros para salvar
save_str='teste_trottle_1';
%%
figure(1)
plot(tempo,angulo_pitch(:,1),'-r',tempo,angulo_pitch(:,2),'--b');
legend('Saída','Referência');
axis([0 120 -50 50]);
grid on
xlabel('Tempo(s)');
ylabel('Ângulo(°)');
title('Posição Angular Pitch X Tempo');


figure(2)
plot(tempo,angulo_roll(:,1),'-g',tempo,angulo_roll(:,2),'--b');
legend('Saída','Referência');
axis([0 120 -50 50]);
grid on
xlabel('Tempo(s)');
ylabel('Ângulo(°)');
title('Posição Angular Roll X Tempo');


figure(3)
plot(tempo,acao_controle(:,1),'-k');
hold on
plot(tempo,acao_controle(:,2),'-r');
hold on
plot(tempo,acao_controle(:,3),'-g');
hold on
plot(tempo,acao_controle(:,4),'-b');

legend('Motor 0','Motor 1','Motor 2','Motor 3');
axis([0 120 0 100]);
grid on
xlabel('Tempo(s)');
ylabel('PWM(%)');
title('Ação de controle X Tempo');

save(save_str);
```