JACOBUS LAURENS DE JAGER

SISTEMA EMBARCADO DE AQUISIÇÃO DE DADOS E DE TELEMETRIA PARA VEÍCULOS AÉREOS NÃO TRIPULADOS DE ASA FIXA

Manaus - Amazonas

2018

JACOBUS LAURENS DE JAGER

SISTEMA EMBARCADO DE AQUISIÇÃO DE DADOS E DE TELEMETRIA PARA VEÍCULOS AÉREOS NÃO TRIPULADOS DE ASA FIXA

Pesquisa desenvolvida durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Elétrica da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para a obtenção do título de Engenheiro Eletricista.

Orientador Prof. Dr. Walter Prado de Souza Guimarães

Manaus - Amazonas

2018

Universidade do Estado do Amazonas – UEA
Escola Superior de Tecnologia - EST

Reitora:
Cleinaldo de Almeida Costa
Vice-Reitor:
Cleto Cavalcante de Souza Leal
Diretor da Escola Superior de Tecnologia:
Roberto Higino Pereira da Silva
Coordenador do Curso de Engenharia Elétrica:
Ingrid Sammyne Gadelha Figueiredo

Banca Avaliadora composta por: *Data da defesa: 04/12/2018.*
Prof. Walter Prado de Souza Guimarães *(Orientador)*
Prof. Jozias Parente de Oliveira
Prof. Walfredo da Costa Lucena Filho

CIP – Catalogação na Publicação

Jager, Jacobus Laurens de

SISTEMA EMBARCADO DE AQUISIÇÃO DE DADOS E DE TELEMETRIA PARAVEÍCULOS AÉREOS NÃO TRIPULADOS DE ASA FIXA / Jacobus Laurens; [orientado por] Walter Prado de Souza Guimarães. – Manaus: 2018.
95p.: il.

Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica). Universidade do Estado do Amazonas, 2018.

1. Sistema Embarcado. 2. Telemetria. 3. VANT .
I. Guimarães, Walter Prado de Souza.

JACOBUS LAURENS DE JAGER

SISTEMA EMBARCADO DE AQUISIÇÃO DE DADOS E DE TELEMETRIA PARA VEÍCULOS AÉREOS NÃO TRIPULADOS DE ASA FIXA

Pesquisa desenvolvida durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Elétrica da Escola Superior de Tecnologia da Universidade Estadual do Amazonas, como pré-requisito para a obtenção do título de Engenheiro Eletricista.

Nota obtida: 9,9

Aprovada em 4/12/2018.

Área de concentração: Sistemas Embarcados

BANCA EXAMINADORA

Orientador: Walter Prado de Souza Guimarães, Dr.

Avaliador: Jozias Parente de Oliveira, Dr.

Avaliador: Walfredo da Costa Lucena Filho, Me.

Manaus 2018

RESUMO

O presente trabalho possui como objetivo o desenvolvimento de um sistema embarcado de aquisição de dados e de telemetria para VANTs – veículos aéreos não tripulados. Foi desenvolvido um sistema de aquisição de dados para coleta dos parâmetros: RPM de motor, indicador *Wheight on Wheels*, altitude, velocidade aerodinâmica calibrada, coordenadas de GPS, deflexão de superfícies de comando, ângulo de arfagem e rolamento, fator de carga e rotação de arfagem e rolamento. Foram realizados voos experimentais com um VANT na qual o sistema foi integrado, obtendo-se o registro dos dados mencionados, que são demonstrados graficamente. Por fim, verifica-se que o sistema foi capaz de auxiliar no projeto aeronáutico de um VANT.

Palavras-chave: Sistema Embarcado. Telemetria. VANT.

ABSTRACT

This paper presents the development of an embedded data acquisition and telemetry system for unmanned aerial vehicles. A data acquisition system was developed to collect the following parameters: engine RPM, Wheight on Wheels indicatir, altitude, calibrated airspeed, GPS coordinates, deflection of control surfaces, pitch and yaw angle, load factor and rotation in the axes. Experimental flights were performed by a UAV in which the system was integrated, during which the mentioned data was recorded, which are presented as graphs. Finally, it is verified that the system was able to assist the aeronautical design of a UAV.

Keywords: Embedded System. Telemetry. UAV.

LISTA DE ILUSTRAÇÕES

1 VANT militar Scout 12
2 VANT voltado para fiscalização de linhas de transmissão 13
3 VANT voltado para agricultura de precisão 14
4 Diagrama de blocos de componentes de um microcontrolador 15
5 Esquemático exemplo de protoclo I2C 16
6 Esquemático exemplo de protoclo SPI 17
7 Esquemático exemplo de UART 17
8 Princípio de funcionamento de sensor ultrassônico 19
9 Diafragma para conversão de pressão em deflexão linear 19
10 Princípio de funcionamento de um acelerômetro capacitivo 21
11 Detalhe de construção de um acelerômetro capacitivo MEMS 21
12 Giroscópio mecânico clássico 22
13 Detalhe de construção de um giroscópio MEMS 22
14 Princípio de funcionamento do Efeito Hall 24
15 Função de transferência de histerese 24
16 Unidade de medição inercial 25
17 Constelação de satélites GPS 26
18 Savox SA1230SG - um servomotor típico 27
19 Diagrama de tempo do sinal de controle do servomotor, para 3 posições . . 28
20 Pilha de camadas do padrão Zigbee 29
21 Características de radiofrequência do padrão ZigBee 30
22 Topologias do padrão Zigbee 32
23 Eixos de uma aeronave e superfícies de comando 33
24 Tubo de Pitot e tomada estática 34
25 Tubo de Pitot e tomada estática 35
26 Diagrama de blocos para etapas do projeto 36
27 Relação de dados adquiridos pelo sistema embarcado 37
28 Arquitetura do sistema de aquisição de dados 38
29 Arquitetura do sistema de telemetria 38
30 VANT "Caboquinha"do projeto de extensão Urutau Aerodesign 39
31 Pista utilizada para decolagem e pouso de voo em Manaus 40
32 Pista utilizada para decolagem e pouso de voo em São José dos Campos . . 40
33 Especificações técnicas do VANT "Caboquinha"do Urutau Aerodesign . . . 41
34 Relógio de tempo real - RTC DS3231 42
35 Esquemático de RTC DS3231 42
36 Sensor Hall DRV5013AD e Imã de Neodímio 43

37 Resposta a campo magnética de DRV5013AD . . . 43
38 Detalhe de instalação de sensor RPM . . . 44
39 Trecho de firmware contendo rotina de interrupção . . . 45
40 Sensor ultrassônico HC-SR04 . . . 45
41 Alcance por distância e ângulo . . . 46
42 Detalhe de instalação de sensor WOW . . . 46
43 Sensor pressão barométrica BME680 . . . 47
44 Sensor pressão diferencial SDP600 e tubo de pitot . . . 48
45 Posicionamento tubo de Pitot . . . 48
46 Detalhe de instalação de sensor VCAS . . . 49
47 GPS Ublox Neo M8 . . . 49
48 Software U-center . . . 50
49 Diagrama de tempo do sinal de controle do servomotor, para 3 posições . . . 50
50 Esquemático de circuito para leitura de PWM . . . 51
51 Sinais de entrada e de saída do circuito de leitura de PWM . . . 51
52 Trecho de firmware contendo leitura do sinal PWM . . . 52
53 IMU Invensense MPU9250 . . . 52
54 Esquemático do módulo IMU . . . 53
55 GPIO exigidos no sistema de aquisição de dados . . . 54
56 IDE do Arduino . . . 54
57 Compilador online do MBED . . . 55
58 Especificações técnicas dos microcontroladores . . . 55
59 Leitor/Gravador de cartão microSD . . . 56
60 Formatação de arquivo texto gerado pelo módulo SD . . . 56
61 Xbee S3B Pro . . . 57
62 Estudo de protocolo de compactação de dados . . . 57
63 Esquemático do módulo de recepção . . . 58
64 Compartimento do sistema de aquisição de dados . . . 59
65 Esquemático do circuito primário do DAS . . . 59
66 Placa do módulo de leitura PWM . . . 60
67 Placa do módulo de leitura PWM . . . 60
68 Placa do módulo de leitura PWM . . . 60
69 Resultados experimentais do voo realizado em Manaus . . . 63
70 Resultados experimentais do voo realizado em São José dos Campos . . . 64
71 Análise de dados obtidos pelo IMU . . . 65
72 Trajetória de voo em Manaus . . . 66
73 Trajetória de voo em São José dos Campos . . . 66
74 Trajetória de voo em Manaus em 3D . . . 67
75 Trajetória de voo em São José dos Campos em 3D . . . 67

SUMÁRIO

INTRODUÇÃO **10**

1 REFERENCIAL TEÓRICO **12**
1.1 VEíCULO AÉREO NÃO TRIPULADO - VANT 12
1.2 MICROCONTROLADOR 14
1.2.1 Protocolo I2C **15**
1.2.2 Protocolo SPI **16**
1.2.3 Comunicação serial UART **17**
1.3 SENSORES 18
1.3.1 Sensor ultrassônico **18**
1.3.2 Transdutor de pressão **19**
1.3.3 Acelerômetro **20**
1.3.4 Giroscópio **22**
1.3.5 Magnetômetro **23**
1.4 UNIDADE DE MEDIÇÃO INERCIAL - IMU 25
1.5 SISTEMA DE POSICIONAMENTO GLOBAL - GPS 26
1.6 SERVOMOTOR 27
1.7 PADRÃO ZIGBEE 28
1.7.1 Camadas **29**
1.7.2 Frequências de Rádio **30**
1.7.3 Tipos de dispositivos **31**
1.7.4 Topologias de rede **31**
1.8 AERONÁUTICA 33
1.8.1 Eixos de um aeronave **33**
1.8.2 Velocidade em relação ao ar **34**
1.8.3 Wheight-On-Wheels **35**

2 METODOLOGIA **36**
2.1 DEFINIÇÃO DE DADOS A SEREM ADQUIRIDOS 37
2.2 ARQUITETURA DO SISTEMA EMBARCADO 38
2.3 VALIDAÇÃO DO SISTEMA 39

3 IMPLEMENTAÇÃO **42**
3.1 MÓDULOS DE AQUISIÇÃO DE DADOS 42
3.1.1 Tempo **42**
3.1.2 RPM **43**
3.1.3 WOW **45**
3.1.4 HP **47**
3.1.5 VCAS **48**

3.1.6 GPS **49**
3.1.7 Deflexão de superfícies de comando **50**
3.1.8 IMU **52**
3.2 MICROCONTROLADORES 54
3.3 MÓDULO DE ARMAZENAMENTO 56
3.4 MÓDULO DE TELEMETRIA 56
3.5 INTEGRAÇÃO DE PROJETO 58

4 ANÁLISE E INTERPRETAÇÃO DE RESULTADOS **61**

CONCLUSÃO **68**

REFERÊNCIAS **68**

ANEXO A – FIRMWARE DE MICROCONTROLADOR PRINCIPAL **72**

ANEXO B – FIRMWARE DE MICROCONTROLADOR AUXILIAR **81**

ANEXO C – FIRMWARE DE MÓDULO TERRESTRE **89**

ANEXO D – FIRMWARE DE IMU **92**

INTRODUÇÃO

O tema deste trabalho é o desenvolvimento de um sistema embarcado de aquisição de dados e de telemetria para veículos aéreos não tripulados de asa fixa. Durante a última década, a tecnologia de VANTs se tornou cada vez mais acessível, abrindo um vasto leque de possibilidades para sua aplicação no setor civil (FLOREANO; WOOD, 2015).

No entanto, o projeto aeronáutico de uma aeronave de pequeno porte apresenta dificuldades devido ao *know-how* da indústria aeronáutica ser concentrado em teoria aplicáveis somente a aeronaves de maior porte, logo a necessidade de validação empírica se torna mais pertinente (LAJúS et al., 2015).

Deste modo um sistema que seja capaz de coletar parâmetros estratégicos de aeronaves de pequeno porte, assim como monitorá-lo em tempo real, agregaria valor ao desenvolvimento de um projeto aeronáutico.

Este trabalho tem por hipótese a ideia de que é possível adquirir parâmetros de uma aeronave de pequeno porte durante voo para fins de validação de projeto aeronáutico. O presente trabalho tem como objetivo o desenvolvimento de um sistema de aquisição de dados, no caso parâmetros relevantes para aeronaves de pequeno porte, constituído de microcontroladores, sensores diversos, módulo para *datalogging* e módulo para telemetria. Depois de projetados, esses serão embarcados a uma aeronave rádio controlada de pequeno porte, com o qual serão realizados voos experimentais para aquisição de dados e análise de parâmetros de seu projeto.

A justificativa é o aprofundamento nas disciplinas do curso de engenharia elétrica. Principalmente: Eletrônica Analógica I, II e III, Eletrônica Digital I e II, Microcontroladores, Linguagem de Programação I e II e Redes de Computadores I. Também, justifica-se este trabalho, pela implementação de uma ferramenta útil que disponibilizaria a aquisição de parâmetros de VANTs, auxiliando, por exemplo, a validar um projeto aeronáutico por meio da coleta de dados empíricos de uma aeronave, permitindo a reavaliação do mesmo.

Este trabalho está divido em quatro seções primárias citadas a seguir: Referencial Teórico; Metodologia; Implementação e, por último, Análise e Interpretação dos Resultados.

O primeiro capítulo se trata do referencial teórico, que consiste na revisão dos assuntos citados no parágrafo anterior. A seção acerca o microcontrolador apresenta-se um breve resumo, citando as formas de comunicação usadas. Os diversos sensores utilizados pelo sistema embarcado, sendo eles: sensor ultrassônico, encoder, transdutor de pressão, acelerômetro, giroscópio magnetômetro. A chamada unidade de medição de inércia, que trata sobre a fusão de dados para obter posição angular. O sistema de posicionamento global, dispositivo de grande praticidade usado no projeto. Servomotores, os quais são dispositivos pertencentes à aeronave para manipular a deflexão de superfícies de comando. O padrão ZigBee, que se trata do protocolo usado para transmissão de dados através radioenlace. Por fim se apresenta os conceitos da área de aeronáutica que são pertinentes para o trabalho.

O segundo capítulo é a metodologia, em que serão descritas as etapas para construção do sistema embarcado; quais parâmetros os sistema embarcado vem a adquirir, a arquitetura de dito sistema, como se realiza a telemetria entre o sistema de aquisição de dados e o módulo de monitoramento terrestre, e como foram realizados os voos experimentais de coleta de dados.

O terceiro capítulo é voltado à implementação, nela descreve-se a execução dos passos da metodologia. Apresenta-se os princípios de funcionamento de cada módulo, entrando em detalhes na solução empregada para cada parâmetro a ser adquirido. Os códigos utilizados na implementação são mostrados em anexos.

O quarto capítulo é voltado para a análise e interpretação dos resultados, onde será descrito o sinal de resposta para módulos de sensores selecionados. Em seguida são apresentados os dados coletados durante voos experimentais, em uma aeronave rádio controlada de pequeno porte. Por fim é realizado uma breve análise do desempenho da aeronave, baseado em cima dos dados coletados, comparando com os dados esperados pelo seu projeto aeronáutico.

Por fim, será apresentado uma conclusão com o intuito de finalizar o trabalho com as considerações finais e sugestões para trabalhos da mesma linha. Em seguida pode-se encontrar as referências bibliográficas citadas ao decorrer deste trabalho.

1 REFERENCIAL TEÓRICO

Neste capítulo, será abordada a teoria necessária para a compreensão do conceito e procedimento do projeto.

1.1 VEÍCULO AÉREO NÃO TRIPULADO - VANT

O termo VANT é a sigla para Veículo Aéreo Não-Tripulado. Essa expressão pode ser associada a quaisquer objetos que voem de forma controlada, desde que evidentemente, não levem tripulação consigo. Assim sendo, tal definição seria apropriada para uma significativa quantidade de objetos, que inclui desde pipas, quadricópteros para fotografia, aeromodelos rádio-controlados, até mísseis balísticos. Todavia no meio aeronáutico atual – em particular no meio militar – a expressão se delimita a aeronaves reutilizáveis mais pesadas que o ar (LOPES; AMARAL, 2004).

As primeiras gerações de VANTs datam do início da década de 1970 , sendo desenvolvidos com a particular intenção de uso em missões de reconhecimento. Países como Estados Unidos e Israel iniciaram a fazer experimentos os VANTs equipados com antenas capazes de transmitir vídeo em tempo real para sua base em solo. Esta funcionalidade demonstrou ser de imenso valor para os comandantes de exércitos, os quais poderiam conseguir informações acerca o campo de batalha ou território inimigo em tempo real, auxiliando em sua capacidade de tomada de decisões. Israel foi o pioneiro no uso de VANTs em combate, quando em 1982, aeronaves do tipo Scout, representado na Figura 1 vasculharam o território do Líbano em busca de defesas antiaéreas sírias, antes das incursões aéreas israelenses (LOPES; AMARAL, 2004).

Figura 1 – VANT militar Scout

Fonte: (ISRAELIWEAPONS, 2001)

Ainda que todo seu desenvolvimento seja baseado em missões militares, a previsão é de que no futuro breve, será cada vez mais intenso o uso de VANTs em aplicações civis, seja para diminuir os custos de determinadas missões desempenhadas por aeronaves tripuladas ou até mesmo para realizar missões em ambientes nocivos ao ser humano (LOPES; AMARAL, 2004).

É possível citar vários exemplos de tais missões, como:

- Fiscalização de linhas de transmissão de energia elétrica – As empresas responsáveis pela manutenção das linhas de transmissão são obrigadas a periodicamente inspecionar a qualidade de suas linhas. No momento em que há alguma falha nessas linhas, deve-se percorrer toda sua extensão até achar o ponto no qual houve a falha. Essa missão, atualmente, é realizada tipicamente por helicópteros ou pequenos aviões tripulados. Designar a missão a um VANT corresponderia a uma significativa diminuição nos custos de operação dessas empresas, uma vez que este consumiria um volume de combustível muito menor que as alternativas tripuladas, exemplificado na Figura 2. Ademais um VANT criado especificamente para essa missão teria autonomia para voar por períodos de tempo maior que os veículos utilizadas hoje em dia, permitindo mais flexibilidade no cumprimento dessa missão (LOPES; AMARAL, 2004).

Figura 2 – VANT voltado para fiscalização de linhas de transmissão

Fonte: (LAN, 2017)

- Transmissão de dados – Um VANT com grande autonomia podeira ser usado como uma estação receptora-retransmissora de sinais. Atualmente há o VANT Hélios, que tem motores elétricos movidos a baterias que são recarregadas por painéis fotovoltaicos encontrados nas asas. Desse modo, o mesmo possui capacidade de permanecer no ar por vários meses, tendo o papel de receptor e retransmissor de sinais de celular para áreas remotas (LOPES; AMARAL, 2004).

- Dispersão de defensivos agrícolas – A aviação agrícola é largamente empregada em fazendas de grande porte. Todavia tal atividade apresenta altíssimo perigo para os pilotos, uma vez que exige que a aeronave voe extremamente próxima ao solo. Assim quaisquer erros ou distrações do piloto, ou falhas na aeronave, nessas condições de voo podem vir a causar graves acidentes. O uso de VANTs para essa aplicação eliminaria o risco exposto à tripulação humana, exemplificado na Figura 3 (LOPES; AMARAL, 2004).

Figura 3 – VANT voltado para agricultura de precisão

Fonte: (NIXON, 2018)

- Pesquisas Científicas – Um VANT pode ser orientado a aplicações de pesquisa. a título de exemplo, há a missão de captar amostras de ar atmosférico sobre grandes cidades, a fim de mensurar a qualidade do mesmo. Um aoutra possível missão consistiria em sobrevoar regiões próximas a vulcões, nos quais a concentração de gases nocivos na atmosfera fazem com que o ambiente exponha risco aos humanos (LOPES; AMARAL, 2004).

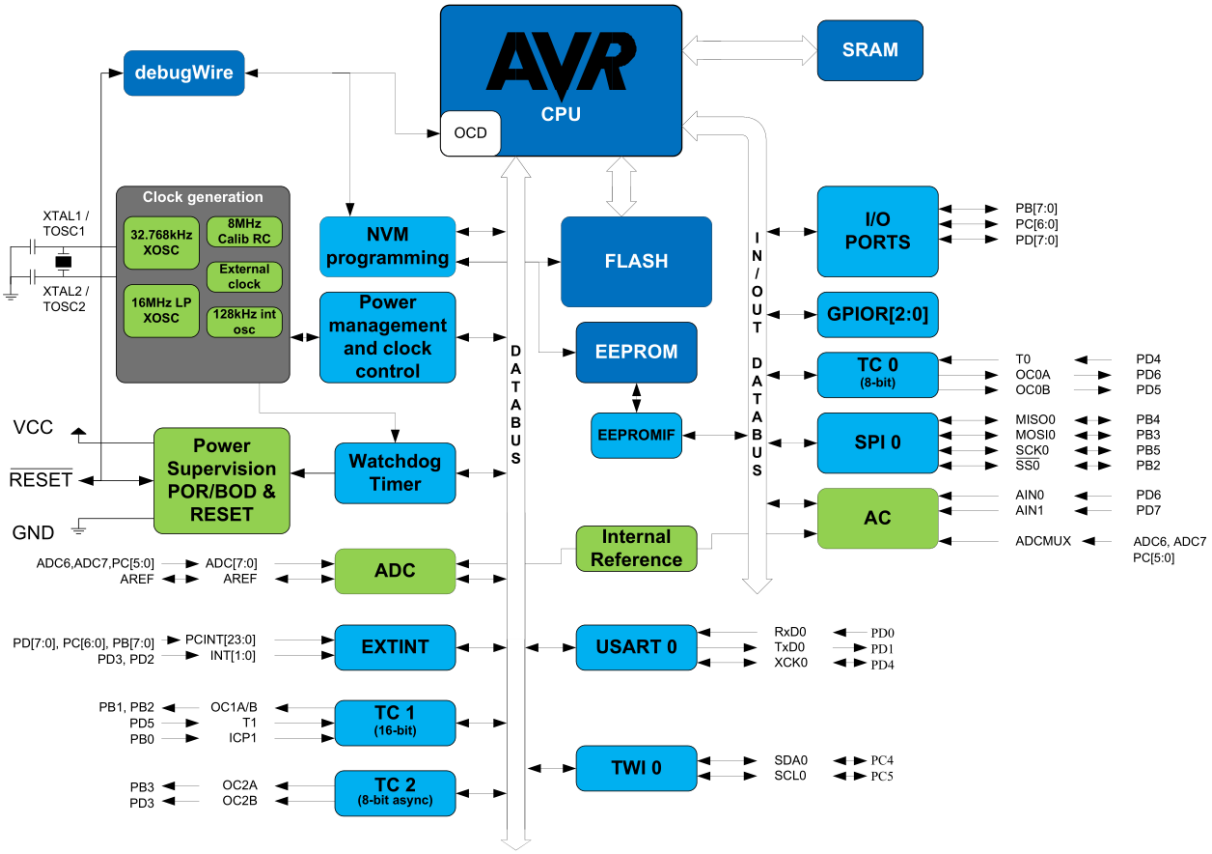
- Geração de imagens de grandes áreas – Um VANT pode ser aplicado para a observação, através de câmera, de grandes porções de terra, como por exemplo para finalidades de fiscalização de uma reserva florestal ou para o monitoramento de plantações numa grande área agrícula. Similarmente ao casos anteriores citados, seria um veículo mais barato e flexível que um avião ou helicóptero tripulado para realizar a mesma missão. Uma outra possibilidade consiste na geração de imagens para redes de televisão, como engarrafamentos de uma metrópole ou um jogo de futebol. Por fim, poderia ajudar a polícia na localização de um criminoso em fuga, atividade que é realizada por helicópteros tripulados atualmente (LOPES; AMARAL, 2004).

1.2 MICROCONTROLADOR

Os microcontroladores são dispositivos que podem ser programados para desempenhar funções específicas, através de linguagem de comando específicas. São utilizados em geral para controlar circuitos, sendo portanto comumente encontrados dentro de outros dispositivos, esses são chamados de "controladores embutidos". A estrutura interna do microcontrolador é constituída de um processador em conjunto de circuitos de memória e de periféricos de entrada e de saída (NISOLOSI, 2009).

A Figura 4 ilustra em forma de diagrama de blocos a estrutura de um microcontrolador, no caso exemplificado por um Atmega 328p.

Figura 4 – Diagrama de blocos de componentes de um microcontrolador



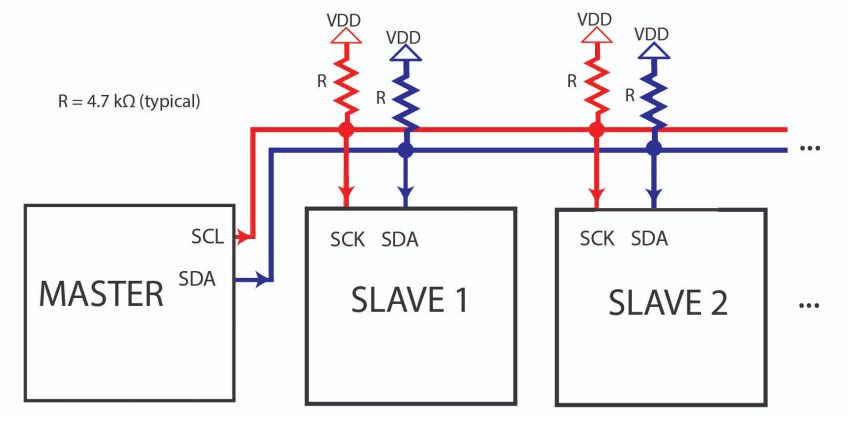
Fonte: (ATMEL, 2016)

São diferenciados dos processadores, uma vez que além dos componentes aritméticos e lógicos típicos de um microprocessador de uso geral, o microcontrolador integra elementos adicionais em sua estrutura interna. Dentre estes elementos adicionais, vale destacar a memória somente de leitura para armazenamento de programas, *EEPROM* memória de leitura e escrita para armazenamento de dados e dispositivos periféricos, como conversores analógico/digitais (ADC), conversores digitais/analógicos (DAC), portas de entrada e saída digitais (I/O). Destaca-se dessa forma a versatilidade e robustez do microcontrolador, pois não necessita de muitos outros componentes para seu funcionamento (NISOLOSI, 2009).

1.2.1 Protocolo I2C

O protocolo I2C, projetada pela NXP Semiconductors (antiga Philips), trata-se de um protocolo de barramento serial com dois fios, cujo barramento é bidirecional, de baixa velocidade e síncrono com um clock em comum, e multimestre, ou seja mais de um dispositivo pode assumir o papel de mestre do barramento ao mesmo tempo. Os fios do protocolo são o SDA (dado serial) e o SCL (clock serial), como apresentadas na Figura 5 (VAHID; GIVARGIS, 2001).

Figura 5 – Esquemático exemplo de protoclo I2C



Fonte: (PELAYO, 2018)

Dispositivos podem ser acrescentados ou removidos do barramento sem afetar os demais, além disso cada dispositivo conectado possui um endereço único e pode operar tanto como um transmissor (mestre) quanto como um receptor (escravo). Cada dispositivo conectado ao barramento possui um endereço único e qualquer transmissão se inicia com o envio dos bits de endereço. O primeiro byte transmitido contém os 7 bits de endereço mais um bit de direção (0 = escrita, 1 = leitura). Dois mestres podem iniciar uma transmissão ao mesmo tempo: como SDA fica em HIGH naturalmente, o mestre que colocar um bit 1 (HIGH), mas perceber que a linha está no nível LOW, entenderá que há outro mestre usando o barramento e interromperá sua ação (VAHID; GIVARGIS, 2001).

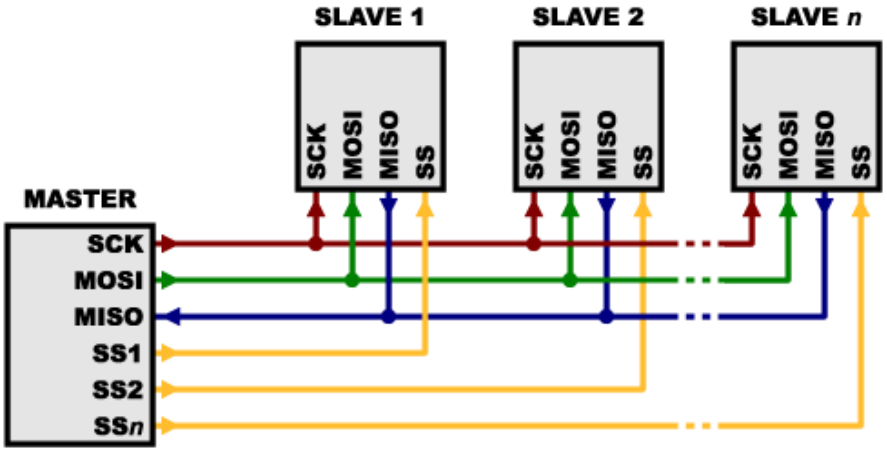
A transferência de dados ocorre segundo uma taxa de 100 kbit/s e usando endereçamento com 7 bits no modo normal e com 3,4 Mbits/s e 10 bits de endereçamento no modo rápido. Entre os dispositivos capazes de interfacear com o protocolo I2C destacam-se: EEPROMS, flash, algumas memórias RAM, relógios de tempo real, temporizadores watchdog, microcontroladores e sensores diversos (VAHID; GIVARGIS, 2001).

1.2.2 Protocolo SPI

O protocol serial SPI é uma interface de comunicação síncrona serial full duplex de quatro fios, originalmente desenvolvido pela Motorola, caracterizado pela sua alta taxa de transferência. O esquema básico de ligação está apresentado na Figura 6. São especificados três sinais lógicos: sinal de SCLK (clock serial), sinal MISO (de escravo para mestre) e sinal MOSI (de mestre para escravo), além de um sinal SS (seleção de escravo) (VAHID; GIVARGIS, 2001).

Apesar de necessitar de fios a mais para comunicação, o SPI tem a vantagem de se comunicar com maior rapidez e possibilita vários dispositivos intercomunicando-se em um barramento compartilhado, além disso o SPI não requer um esquema de endereçamento de dispositivos tão elaborado quanto o I2C por exemplo (VAHID; GIVARGIS, 2001).

Figura 6 – Esquemático exemplo de protoclo SPI



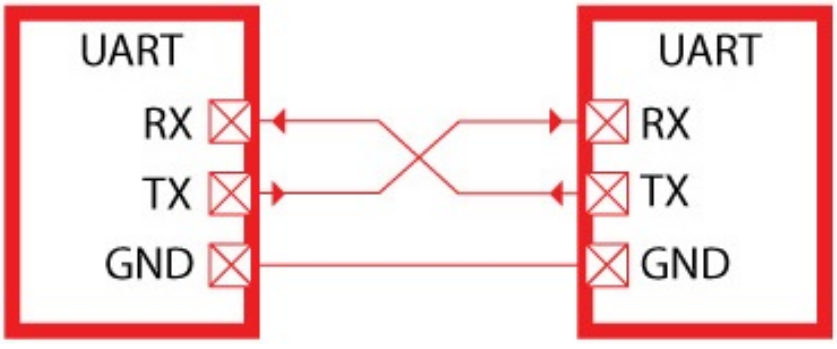
Fonte: (SPARKFUN, 2018)

O mestre, através do controle do clock, decide quando o dado é enviado dele ou é recebido por ele. Dentro de um mesmo ciclo de clock, uma comunicação full duplex pode ser realizada com o mestre que pode enviar dados para um escravo e receber dados dele ou de outro dispositivo simultaneamente. Usando o sinal SS, o mestre pode escolher com qual dos escravos se comunicar (VAHID; GIVARGIS, 2001).

1.2.3 Comunicação serial UART

A sigla UART significa Receptor e Transmsissor Assíncrono Universal (*Universal Assincronous Receiver/Transmitter* em inglês). Não se trata de um protocolo de comunicação, como o SPI ou o I2C mencionados nas seções anteriores, mas de um circuito físico em um microcontrolador ou um circuito integrado autônomo. O objetivo principal da comunicação UART é transmitir e receber dados seriais, entre dois dispositivos que se comunicam diretamente entre si (VAHID; GIVARGIS, 2001).

Figura 7 – Esquemático exemplo de UART



Fonte: (MIKROE, 2016)

O transmissor UART converte dados paralelos de um dispositivo de controle como uma CPU em forma serial, transmite em série para a UART receptora, que então converte os dados seriais de volta em dados paralelos para o dispositivo receptor. Apenas dois fios são necessários para transmitir dados entre dois UARTs. Os dados fluem do pino Tx do transmissor UART para o pino Rx do UART receptor, conforme representado na Figura 7 (VAHID; GIVARGIS, 2001).

Os periféricos UARTs transmitem dados de forma assíncrona, o que significa que não há sinal de clock para sincronizar a saída de bits do transmissor UART para a amostragem de bits pelo UART receptor. Em vez disso, o transmissor adiciona bits de início e fim ao pacote de dados que está sendo transferido. Esses bits definem o início e o fim do pacote de dados, de modo que o receptor sabe quando começar a ler os bits (VAHID; GIVARGIS, 2001).

Quando o receptor detecta um bit inicial, ele começa a ler os bits recebidos em uma frequência específica conhecida como taxa de transmissão. A taxa de transmissão é uma medida da velocidade de transferência de dados, expressa em bits por segundo (bps). Ambos os dispositivos devem operar na mesma taxa de transmissão. Qualquer valor de velocidade para a taxa de transmissão pode ser empregada, mas existem valores convencionais que são usados como padrão. O valor mais comum e padronizado é 9600. Outras taxas de transmissão padrão são: 1200, 2400, 4800, 19200, 38400, 57600 e 115200. Taxas de transmissão superiores a 115200 podem ser usadas, mas geralmente isso causa muitos erros na transmissão (VAHID; GIVARGIS, 2001).

1.3 SENSORES

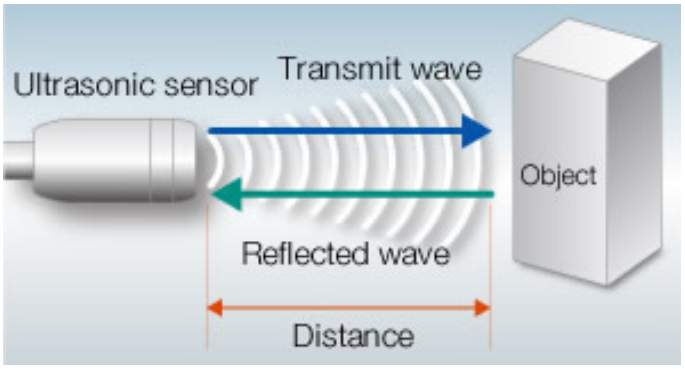
Em esta seção são abordados os sensores que foram utilizados na implementação deste trabalho, descrevendo-os e explanando os seus princípios de funcionamento.

1.3.1 Sensor ultrassônico

O sensor ultrassônico são dispositivos empregados para a medição e detecção de objetos por meio de emissão e recepção de ondas acústicas. O princípio de funcionamento do sensor se dá da seguinte forma: emite-se ondas acústicas ao meio, e devido ao atraso do eco recebido no receptor, é possível identificar objetos na sua faixa de atuação. Através do estímulo elétrico nos transdutores, estes convertem o sinal elétrico recebido em ondas sonoras, que por sua vez quando estimulados pelas ondas sonoras, convertem estas em sinal elétrico. A Figura 8 ilustra o princípio (CARNEIRO; LUGLI, 2014).

Possui diversas áreas de aplicação, como na medicina, em que são usados na geração de imagens de diferentes partes do corpo através de ultrassonografia; na indústria, sendo utilizado para detectar o movimento de objetos em esteiras e para medição de distâncias, como de nível de líquido, detecção de impurezas em rótulos, assim por diante; em aplicações residenciais, em

Figura 8 – Princípio de funcionamento de sensor ultrassônico



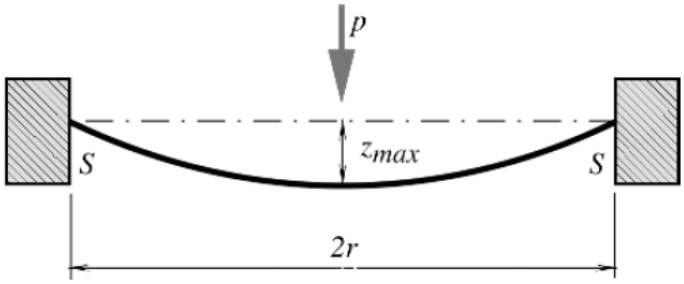
Fonte: (NDK, 2018)

que se destaca os alarmes e detectores de presença; na robótica, em que se usa para determinar a posição do robô em relação ao ambiente em sua volta, em particular aplicado para desviar de obstáculos (CARNEIRO; LUGLI, 2014).

1.3.2 Transdutor de pressão

O transdutor de pressão é um dispositivo eletromecânico que é capaz de sentir pressão e converter para um sinal elétrico. São utilizados para medição em aplicações diversas, a título de exemplo destaca-se: sistemas de refrigeração, sistemas embarcados para equipamentos hospitalares, monitoramento de caixas de vácuo em processos industriais, monitoramento de pressão interna de cabines aeronáuticas, malha de controle em sistemas eletropneumáticos, entre outros. Há vários tipos, cada qual emprega uma solução diferente para conversão de sinal elétrico, no entanto fazem uso do mesmo princípio básico (BECKERATH et al., 2008).

Figura 9 – Diafragma para conversão de pressão em deflexão linear



Fonte: (FRADEN, 2004)

Em sensores de pressão, o elemento sensor é um dispositivo mecânico que sofre alterações estruturais sob carga, de modo que se obtém o valor de pressão pela deflexão sofrido por este dispositivo. Geralmente se usa um diafragma como dispositivo defletor, tipicamente membranas de silicone. Quando pressão é aplicado em um dos lados da membrana, este tende a se deformar esfericamente, como representado na Figura 9. Para pequenos valores de pressão, a

tensão mecânica sobra a área da membrana e o centro de deflexão ocorrem em linearmente em função da pressão (FRADEN, 2004).

Como mencionado anteriormente, existem várias soluções para transdutores de pressão, serão descritos três dos tipos mais facilmente encontrados no mercado nos próximos parágrafos, sendo eles: transdutor extensiométrico, piezorresistivo e capacitivo.

Os transdutores de pressão extensométricos são aqueles que fazem uso de células de carga acopladas a membrana de deflexão. Há duas formas que tal acoplamento é feito: colados diretamente ao diafragma, geralmente fios ou folhas metálicas, sendo o mais comum; e fixado paralelamente a superfície da membrana. O circuito elétrico equivalente da célula de carga é essencialmente uma ponte de Wheatstone. A conversão de sinal se dá pelo valor de força obtido pela célula de carga, que é então relacionado ao deslocamento exercido pelo diafragma. São utilizados tanto para medição de pressão estática quanto pressão diferencial (FRADEN, 2004).

Os transdutores piezorresistivos são baseados no efeito de mesmo nome. O efeito piezorresistivo ocorre em certos materiais semicondutores, tal como o silício e germânio, e consiste na variação da resistência do elemento ao sofrer certa deformação. São fabricados para possuir variações de resistência positivas ou negativas quando deformado. São até 30 vezes mais sensíveis do que os transdutores extensométricos, no entanto são sensíveis a temperatura, difíceis de compensar e possuem resposta não linear. São indicados para aplicações em que se precisa de maior confiabilidade e a temperatura tende a ser estável (FRADEN, 2004).

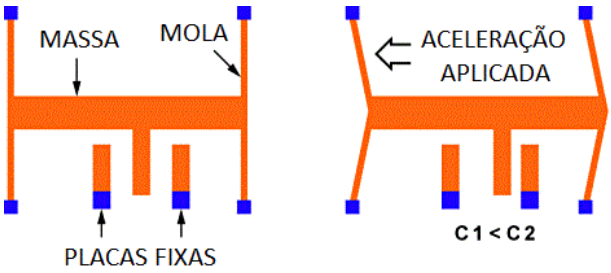
O transdutor capacitivo possui diafragmas fixos e móvel, o espaço entre eles é preenchido com um líquido com propriedades dielétricas. Ao sofrer deflexão devido a presença de pressão, o diafragma móvel se aproxima do diafragma afetado. Tendo em vista que um capacitor é essencialmente duas placas paralelas separadas por meio dielétrico, ocorre que esses diafragmas agem como um capacitor, cuja capacitância varia em função do deslocamento provocado pela pressão. Assim sendo, torna-se possível medir a pressão. Trata-se do tipo de transdutor mais utilizado, fato justificado principalmente por sua extensa faixa de medição. Oferecem respostas lineares e são insensíveis a variação de temperatura. São indicado para instrumentação e sistemas de controle, por possuírem ótimo desempenho (FRADEN, 2004).

1.3.3 Acelerômetro

Um acelerômetro é um sensor cujo sinal de saída indica a aceleração ao qual se encontra submetido, no sentido de apenas uma direção. Existem diversas maneiras de se construir um acelerômetro, sendo a mais típica a que faz uso da variação de capacitância. O princípio de funcionamento deste tipo de sensor se fundamenta na movimentação que uma massa de prova, que é suspensa por molas, realiza entre duas placas fixas ao substrato do circuito integrado, de modo que as capacitâncias entre a massa de prova e cada uma das placas variam proporcionalmente às suas respectivas distâncias. (SICILIANO; KHATIB, 2008).

A Figura 10 ilustra que, no momento em que há aceleração para a esquerda, as placas fixas movimentam-se para a esquerda. Todavia, uma vez que, a massa de prova se encontra suspensa por molas, por causa de sua inércia, ela não move de imediato. Logo, do ponto de vista das placas móveis, o efeito é que a massa é aproximada à placa da direita, aumentando a capacitância entre elas (C2), e reduzindo a capacitância entre a massa e a placa da esquerda (C1). A diferença entre esses dois valores de capacitância ΔC, é portanto proporcional à aceleração aplicada (SICILIANO; KHATIB, 2008).

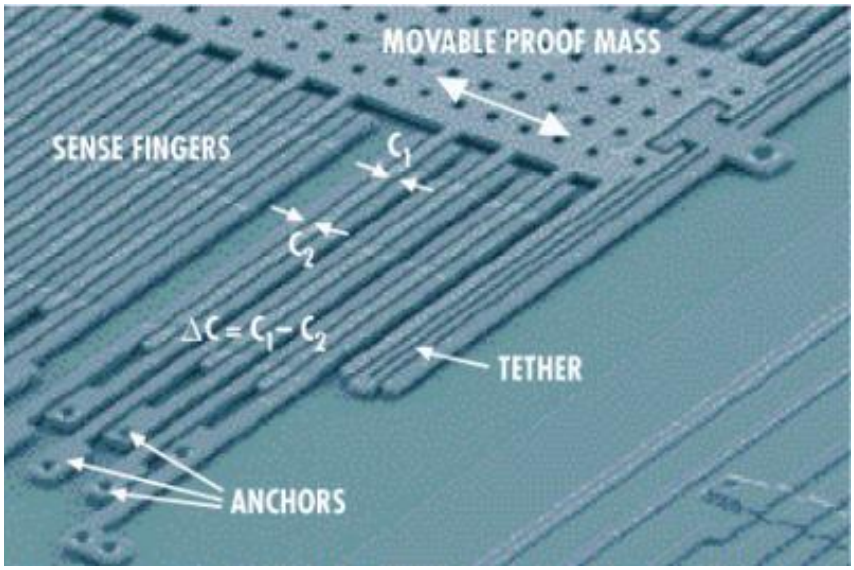
Figura 10 – Princípio de funcionamento de um acelerômetro capacitivo



Fonte: (SICILIANO; KHATIB, 2008)

A Figura 11 apresenta em detalhe as placas fixas e a massa de prova móvel com hastes protuberantes, fixa pela mola. Considerando que a variação de capacitância é diminuta, vários conjuntos de placas fixas e móveis são usados, a fim de se obter um sinal mensurável (MOURA, 2013).

Figura 11 – Detalhe de construção de um acelerômetro capacitivo MEMS

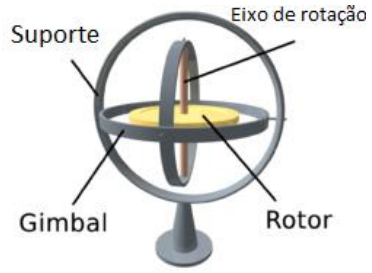


Fonte: (SICILIANO; KHATIB, 2008)

1.3.4 Giroscópio

Giroscópios são sensores capazes de medir a orientação de um objeto, baseados no princípio da conservação de momento angular. A versão clássica de um giroscópio, apresentada na Figura 12, é constituído por um disco (rotor) montado em uma estrutura que oferece 3 graus de liberdade rotacionais, assim o disco tem liberdade para girar em torno de qualquer um dos eixos de rotação (SICILIANO; KHATIB, 2008).

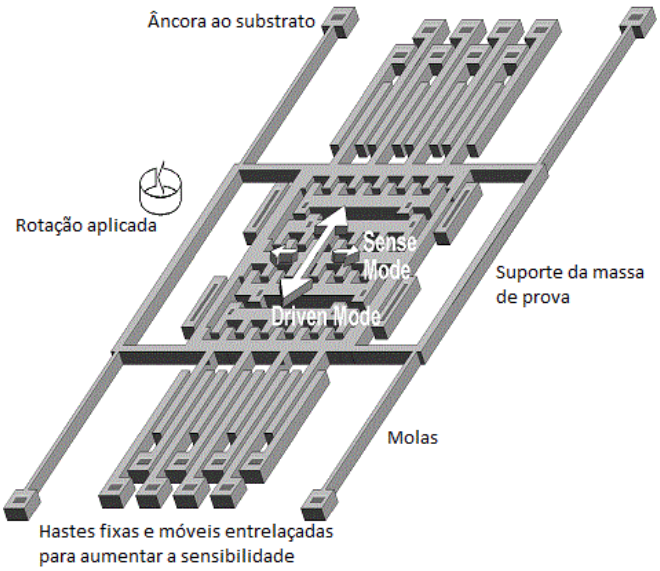
Figura 12 – Giroscópio mecânico clássico



Fonte: (SICILIANO; KHATIB, 2008)

Ao rotacionar, o disco tende a continuar girando no mesmo sentido, por causa de momento angular. Uma vez que o mecanismo é construído de forma a minimizar o atrito nas juntas dos arcos, quando a estrutura é rotacionada, o disco permanece na mesma orientação, rotacionando praticamente de forma independente do restante da estrutura. Dessa forma, é possível mensurar o ângulo entre o eixo de rotação do disco e os arcos mais externos, possibilitando determinar o quanto a estrutura girou em relação à sua orientação original (MOURA, 2013).

Figura 13 – Detalhe de construção de um giroscópio MEMS



Fonte: (SICILIANO; KHATIB, 2008)

Os giroscópios MEMS podem ser construídos de diversas formas, sendo uma das mais simples e baratas aquela que faz uso de microestruturas vibrantes. No lugar de um disco girante, esse tipo de giroscópio utiliza uma massa de prova que oscila em somente uma direção. Similarmente ao acelerômetro, o sinal de saída do sensor é gerada pela aproximação e distanciamento entre as aletas que se projetam do corpo de prova e as aletas fixas ao substrato do sensor (SICILIANO; KHATIB, 2008).

No memento no qual o sensor é rotacionado, a massa de prova tende a permanecer vibrando no mesmo sentido em que se encontrava anteriormente, de forma a induzir uma deflexão entre as hastes proporcional à velocidade angular. A Figura 13 ilustra uma configuração típica de um giroscópio MEMS com estrutura vibrante (SICILIANO; KHATIB, 2008).

1.3.5 Magnetômetro

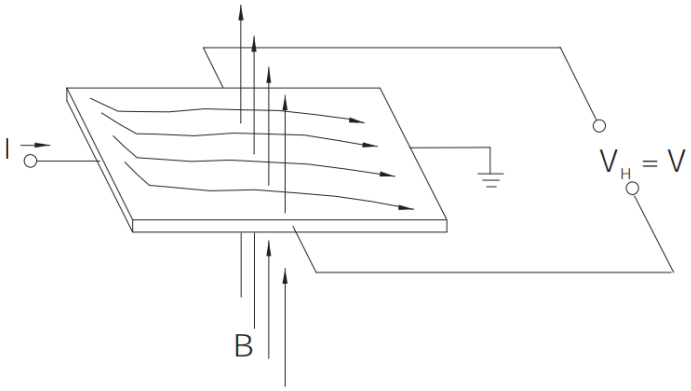
Magnetômetros são dispositivos empregados na medição da intensidade, direção e sentido de campos magnéticos. Magnetômetros são amplamente usados para medição do campo magnético da Terra. Existem dois tipos: escalares e vetoriais; enquanto que os magnetômetros escalares medem somente a intensidade do campo magnético, os vetoriais possuem capacidade de medir a intensidade e sentido do campo magnético na direção em que estão alinhados. Fazendo uso de dois magnetômetros vetoriais dispostos em paralelo à Terra, mas não paralelos entre si, pode-se calcular a direção do norte magnético da Terra, através da soma vetorial das leituras. Para esse sensor há diversos princípios de funcionamento, como, por exemplo, os magnetômetros de Efeito Hall, de bobina rotativa, e de precessão protônica (SICILIANO; KHATIB, 2008).

Quando um condutor de corrente é colocado em um campo magnético, uma tensão será gerada perpendicularmente à corrente e ao campo. Este princípio é conhecido como o efeito Hall. A Figura 14 ilustra o princípio básico do efeito Hall. Ele mostra uma fina folha de material semicondutor (elemento Hall) através da qual uma corrente é passada. As conexões de saída são perpendiculares à direção da corrente. Quando nenhum campo magnético está presente, a distribuição de corrente é uniforme e nenhuma diferença de potencial é vista na saída (HONEYWELL, 2018).

Quando um campo magnético perpendicular está presente, como mostrado na Figura 14, uma força de Lorentz é exercida na corrente. Essa força perturba a distribuição de corrente, resultando em uma diferença de potencial (voltagem) na saída. Essa tensão é a tensão Hall (VH) (HONEYWELL, 2018).

A tensão Hall é proporcional ao produto vetorial da corrente e do campo magnético, na ordem de 7 μv/Vs/Gauss em silício e, logo, requer amplificação para aplicações práticas. O silício tem efeito de piezoresistência, variação na resistência elétrica é proporcional à deformação. É desejável minimizar esse efeito em um sensor Hall. Isso é feito orientando o elemento Hall no CI para minimizar o efeito do estresse e usando vários elementos Hall (HONEYWELL, 2018).

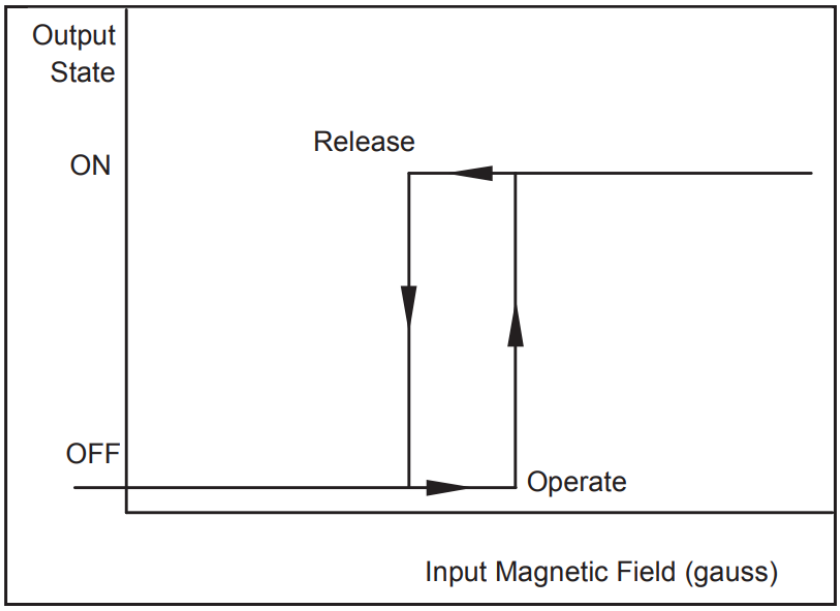
Figura 14 – Princípio de funcionamento do Efeito Hall



Fonte: (HONEYWELL, 2018)

A função de transferência para um sensor de efeito Hall de saída digital incorporando histerese é mostrada na Figura 15. As principais características de entrada/saída são o ponto de operação, o ponto de liberação e a diferença entre os dois ou o diferencial. À medida que o campo magnético é aumentado, nenhuma mudança na saída do sensor ocorrerá até que o ponto de operação seja atingido (HONEYWELL, 2018).

Figura 15 – Função de transferência de histerese



Fonte: (HONEYWELL, 2018)

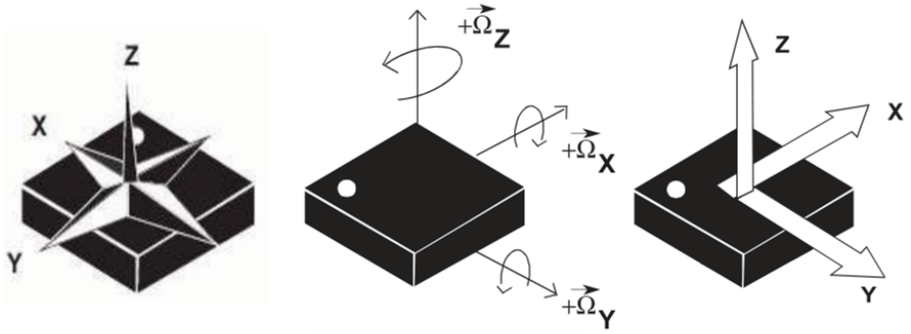
Uma vez atingido o ponto de operação, o sensor mudará de estado. Aumentos adicionais na entrada magnética além do ponto de operação não terão efeito. Se o campo magnético for reduzido para abaixo do ponto de operação, a saída permanecerá a mesma até que o ponto de liberação seja atingido. Nesse ponto, a saída do sensor retornará ao seu estado original (OFF). A finalidade do diferencial entre o ponto de operação e de liberação (histerese) é eliminar o disparo falso que pode ser causado por pequenas variações na entrada (HONEYWELL, 2018).

1.4 UNIDADE DE MEDIÇÃO INERCIAL - IMU

A Unidades de Medição Inercial (do inglês, Inertial Measurement Unit – IMU) é um dispositivo eletrônico que empregam acelerômetros, giroscópios e, eventualmente, magnetômetros para a medição de velocidade, orientação e posição em relação à uma referência. Uma IMU pode ser aplicada em uma ampla gama de aplicações, desde um simples dispositivo para equilíbrio automático, até sistemas complexos de navegação estimada (SICILIANO; KHATIB, 2008).

A Figura 16 ilustra um dispositivo IMU.

Figura 16 – Unidade de medição inercial



Fonte: (SPARKFUN, 2018)

Os valores perdidos entre duas amostras consecutivas de sensores, ou a presença de ruído, são fonte de erros acumulativos em uma IMU. No caso de um sistema ideal com taxa de amostragem infinita (e velocidade de processamento também infinita), IMUs poderiam ser usados sem o auxílio de sensores adicionais, sem sofrerem desvios nos sinais de saída (MOURA, 2013).

A função do acelerômetro é fornecer a característica de movimento a longo prazo. Esse sensores apresentam alta taxa de ruído, não sendo capazes de medir apenas as componentes lineares da aceleração aplicadas ao sistema, medindo ainda as componentes angulares de aceleração. Quando se rotaciona um acelerômetro ao redor de seu centro de massa, por exemplo, ele registrará aceleração, ainda que não esteja sendo submetido a aceleração linear. Todavia os valores de aceleração não sofrem de erros acumulativos (MOURA, 2013).

O papel do giroscópio é de disponibilizar a característica de movimento a curto prazo. Esses sensores tendem a ser muito menos suscetíveis a ruídos que acelerômetros, além de apresentarem resposta mais rápida. Todavia, uma vez que as leituras de um giroscópio fornecem a velocidade angular, há necessidade de se realizar a integração destas leituras a fim de se encontrar o ângulo de rotação do sistema, o qual introduz erros acumulativos de integração. Dessa forma, a longo prazo, a estimativa baseada exclusivamente em giroscópios deriva do valor real (MOURA, 2013).

Ao se juntar o acelerômetro e o giroscópio em uma IMU, os problemas de cada um são compensados pelo outro. O acelerômetro corrige a deriva do giroscópio, através do vetor da força gravitacional a fim de estabelecer a direção vertical correta, reduzindo o erro da posição angular do sistema, enquanto que o giroscópio diminui a sensibilidade do acelerômetro a acelerações não lineares, enquanto que (MOURA, 2013).

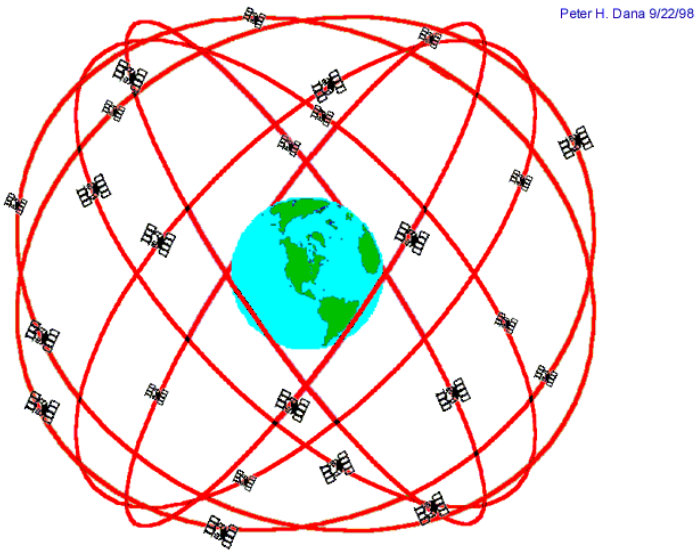
A adição de magnetômetros corrige a deriva da posição angular em torno do eixo da força gravitacional. O acelerômetro não possui capacidade de correção da deriva em rotações em torno deste eixo, devido o vetor aceleração de um objeto ser invariante a rotações sobre o eixo de gravidade. Sendo assim, o magnetômetro inclui a direção do norte magnético da Terra como elemento de correção no sistema (MOURA, 2013).

1.5 SISTEMA DE POSICIONAMENTO GLOBAL - GPS

Fundamentalmente o posicionamento de um objeto nada mais é do que atribuir-lhe coordenadas. Seguindo essa linha de raciocínio, na década de 1970, uma solução para esse tipo de problema foi apresentada com a proposta do GPS (Global Positioning System), desenvolvido pelo Departamento de Defesa dos Estados Unidos da América. Inicialmente denominado como NAVSTAR-GPS (NAVigation Satellite with Time And Ranging), possuía como objetivo servir como auxílio na navegação das forças armadas americanas (MONICO, 2000).

A figura 17 ilustra a constelação de satélites do sistema de radionavegação GPS.

Figura 17 – Constelação de satélites GPS



Fonte: (DANA, 2000)

O principio de funcionamento fundamental de navegação por GPS consiste na medida de distâncias entre o usuário, que assume papel de receptor, e os satélites rastreados. Uma vez conhecidas as coordenadas dos satélites num sistema de referência apropriado, pode-se calcular as coordenadas do receptor no mesmo sistema de referência dos satélites, determinando logo sua posição em tempo real (MONICO, 2000).

O segmento espacial é constituído de 24 satélites distribuídos em seis planos orbitais igualmente espaçados, com quatro satélites por plano, a uma altitude de cerca de 20000 km. O sistema GPS garante que a posição do satélite se repita diariamente, cerca de 4 minutos antes em relação ao dia anterior (DANA, 2000).

O conceito do sistema GPS propicia ao usuário a capacidade de rastrear pelo menos quatro satélites, em qualquer parte da superfície terrestre, a qualquer momento. Tal disponibilidade de satélites possibilita ao usuário obter coordenadas de posicionamento em tempo real (MONICO, 2000).

O funcionamento do sistema GPS é garantido sob maioria das condições climáticas e possui abrangência global, logo passou a ser de uso comum do setor civil. Em particular a partir de maio de 2000, quando houve a eliminação do código SA (Selective Availability), o qual era realizava de forma proposital a degradação do sinal do sistema para fins civis, fato que, por conseguinte, reduzia a precisão fornecida pelo mesmo. Tal medida permitiu aos usuários civis do sistema um posicionamento com precisão até dez vezes maior (MONICO, 2000).

1.6 SERVOMOTOR

Um servomotor se trata de um atuador rotativo que possibilita controle preciso de posição e velocidade angulares. O dispositivo é constituído por um motor acoplado a um sensor que fornece realimentação da posição (EMBEDDED, 2018).

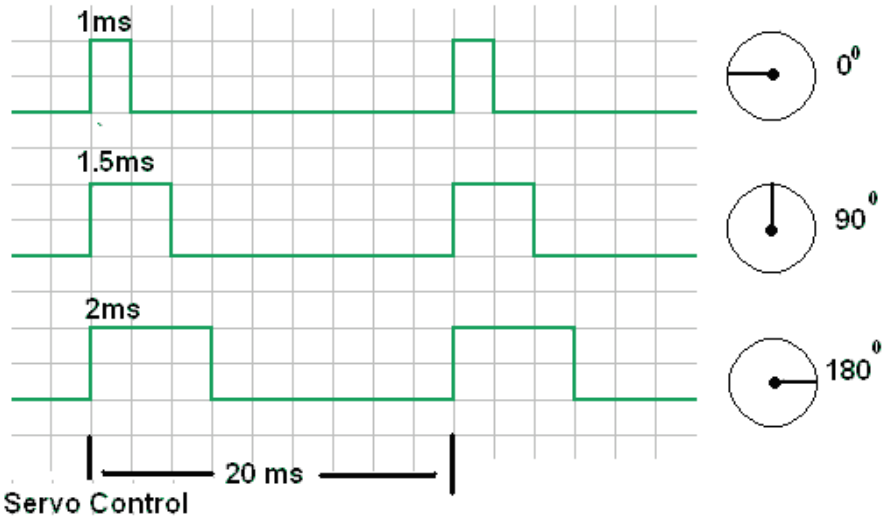
Servomotores como o SA1230SG, da fabricante Savox, apresentado na Figura 18, são tipicamente utilizados em aplicações de aeromodelismo. Essa categoria de servo é controlado através de um sinal de controle do tipo PWM (Pulse Width Modulation) (BAKSHI; BAKSHI, 2009).

Figura 18 – Savox SA1230SG - um servomotor típico

Fonte: (SAVOX, 2018)

O período do sinal de controle possui valor de 20 ms, o qual se encontra divido em duas fases, de período em nível alto e em nível baixo respectivamente. Ao variar o período no qual o sinal se encontra em nível alto entre 1 e 2 milissegundos, o ângulo do eixo do motor rotaciona proporcionalmente entre 0 e 180 graus, tal qual pode ser visto na Figura 19.

Figura 19 – Diagrama de tempo do sinal de controle do servomotor, para 3 posições



Fonte: (FRACAROLLI, 2012)

1.7 PADRÃO ZIGBEE

O ZigBee é um padrão de protocolo baseada na especificação IEEE 802.15.4, voltado para comunicação de alto nível para a criação de redes de área pessoal, isto é, de baixa taxa de dados e proximidade, com frequência de rádio pequena e de baixo consumo energético (BRONZATTI, 2013).

Foi desenvolvido visando ser uma alternativa de comunicação em redes que não necessitem de soluções mais complexas para seu controle, tais como barateando assim os custos com a aquisição, instalação de equipamentos, manutenção e mão-de-obra (BRONZATTI, 2013).

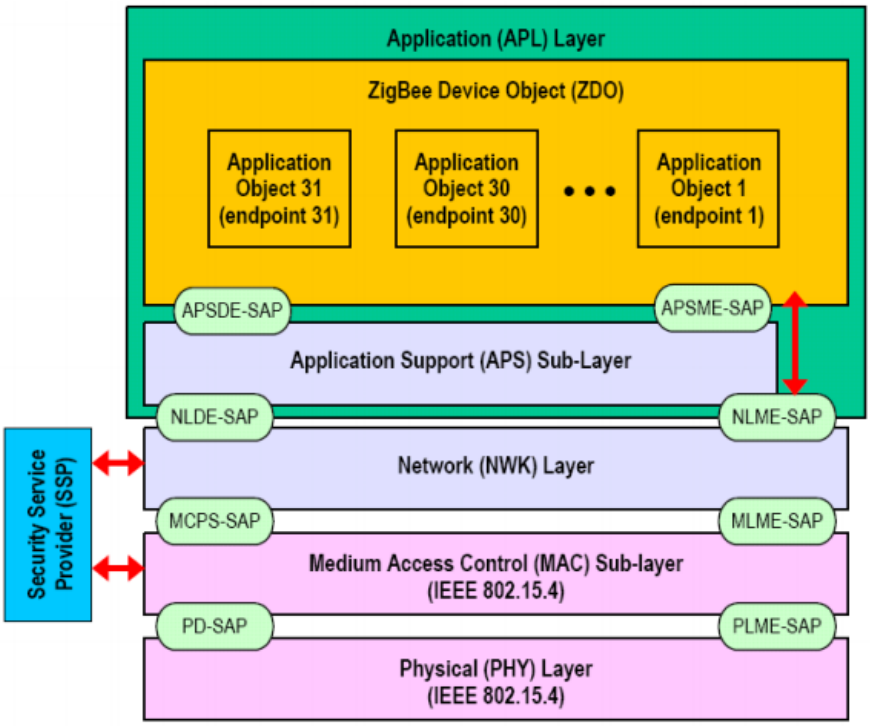
Trata-se de uma tecnologia relativamente simples, que faz uso de pacotes de dados com características específicas, sendo projetado de modo a oferecer flexibilidade em relação aos tipos de dispositivos com o qual se pode comunicar. Dessa forma se torna ideal para aplicações de Internet das Coisas, automação residencial, coleta de dados, monitoramento, dispositivos médicos, entre outros projetos de pequena escala que precisam de conexão sem fio (LIMA; MENDES, 2008).

1.7.1 Camadas

A arquitetura do protocolo Zigbee foi baseada no modelo OSI, sendo formada por um conjunto de blocos denominados de Camadas. É constituída por 4 camadas, das quais as duas primeiras, a Camada Física (PHY) e a Camada de Controle de Acesso ao Meio (MAC) são definidas pela especificação IEEE802.15.4. As duas seguintes, a Camada de Rede (NWK) e a Camada de Aplicação (APL) são definidas pela Zigbee Alliance (BRONZATTI, 2013).

A Figura 20 ilustra a pilha de camadas do padrão Zigbee.

Figura 20 – Pilha de camadas do padrão Zigbee



Fonte: (TOMAR, 2011)

A Camada Física (PHY) foi projetada a fim de acomodar as necessidades de interfaces de baixo custo, possibilitando altos níveis de integração. A DSS (Direct Sequence Spread) é a técnica de transmissão utilizada nesta camada, que visa permitir com que os dispositivos nas redes ZigBee sejam muito simples, possibilitando implementações mais baratas. Definem-se dois tipos de camada PHY, que se diferenciam pelo espectro de frequência de operação: uma em 2,4GHz e outra em 868/915 MHz. (BRONZATTI, 2013).

A Camada de Controle de Acesso ao Meio (MAC) foi projetada com o intuito de permitir topologias múltiplas com baixa complexidade, vantajoso pois a gerenciamento de energia terá modos de operação simples. Ademais a camada permite que um dispositivo com funcionalidade reduzida (RFD) opere na rede usando pouca memória, de modo que estes ainda podem controlar um elevado número de dispositivos sem a necessidade de colocá-los em “modo de espera” (BRONZATTI, 2013).

A Camada de Rede (NWK) é aquela que interliga as camadas de aplicação Zigbee com as camadas IEEE 802.15.4. Fornece as funcionalidades das rede Zigbee, como modos de roteamento, segurança e estrutura da rede e formação de uma nova rede. A camada NWK executa tarefas como formação de uma nova rede, associação e dissociação de dispositivos da rede, atribuição de endereços e descoberta de rotas. (BRONZATTI, 2013).

A Camada de Aplicação (APL) se trata da camada no topo da pilha protocolar. Possui as aplicações que darão funcionalidade para o dispositivo, podendo haver até 240 aplicações em um mesmo dispositivo. A camada APL é dividida em três partes: Subcamada de Suporte a Aplicação (APS), Zigbee Device Object (ZDO) e Framework de Aplicação. Esta última é definida pelo usuário, ou seja, pelo fabricante do dispositivo ZigBee, que tipicamente fornece uma Interface de Programação de Aplicação (API) para implementação de perfis de uso (BRONZATTI, 2013).

1.7.2 Frequências de Rádio

Figura 21 – Características de radiofrequência do padrão ZigBee

Padrão	Frequências	N° de canais	Técnica de Modulação	Taxa de dados
802.15.4	2.4-2.4835 GHz	16	O-QPSK	250 kbit/s
	868-870 MHz	1	BPSK	20 kbit/s
	902-928 MHz	10	BPSK	40 kbit/s

Fonte: Autor

Os dispositivos baseados na tecnologia ZigBee operam na faixa ISM (indústria, cientifica e médica) que não requer licença para funcionamento. Inclui-se as faixas de 2,4 GHz (Global), 915 MHz (América) e 868 MHz (Europa), com taxas de transmissão de dados de 250kbps, 40kbps e 20kbps respectivamente. Resume-se esses dados na Figura 21. (BRONZATTI, 2013).

A faixa de 2,4 GHz utiliza a técnica de modulação O–QPSK (Offset - Quadrature Phase - Shift Keying), enquanto que as faixas de 868 MHz e de 915 MHz fazem uso da a técnica de modulação BPSK (Binary Phase - Shift Keying) (BRONZATTI, 2013).

Os três espectros de frequência são divididos em faixas menores de modo a formar diversos canais de comunicação. No espectro de 868 MHz há apenas 1 canal; em 915 MHz há 10 canais à disposição; e em 2,4 GHz existem 16 canais. A largura dos canais é de 0,6 MHz em 868 MHz; 2 MHz em 915 MHz; e 5 MHz em 2,4 GHz. Totalizando há 27 canais disponíveis pelo padrão Zigbee (BRONZATTI, 2013).

Atualmente os dispositivos Zigbee são capazes de atuar em distância que varia desde 10 m a 9 km, variando de acordo com as características do modelo e da configuração programada (LIMA; MENDES, 2008).

1.7.3 Tipos de dispositivos

Há dois tipos de dispositivos em uma rede ZigBee, quanto a capacidade de processamento, o FFD e o RFD.

Os Dispositivos de Funções Completas (FFD) são dispositivos mais complexos que necessitam de hardware mais potente para a implantação da pilha de protocolos, consumindo por conseguinte mais energia. São capazes de funcionar para qualquer topologia da rede, desempenhando a função de coordenador da rede e por consequência tendo acesso a todos os outros dispositivos da rede (LIMA; MENDES, 2008).

Os Dispositivos de Funções Reduzidas (RFD) são dispositivos mais simples, no qual a pilha de protocolo é implementada usando os mínimos recursos possíveis de hardware. É limitado à topologia em estrela, de modo que pode apenas se comunicar com um coordenador de rede (LIMA; MENDES, 2008).

Em relação ao papel que exercem dentro da rede, os dispositivos Zigbee são classificados em três tipos: Coordenador, Roteador e Dispositivo Final.

O Coordenador tem papel de gerenciar a rede, sendo responsável pela maioria das atividades que nela ocorrem. Seleciona o canal a ser usado, inicia a rede, permite que novos dispositivos se associem a ele e também faz a transferência de mensagens como um roteador. Também estabelece o período permitido para transmissão, dependendo do tipo de rede (BRONZATTI, 2013).

O Roteador possui a capacidade de retransmitir os pacotes de um dispositivo para o outro. Além disso, é capaz de associar outros dispositivos, integrando-os a rede. Todavia não pode iniciar uma rede nova. É útil quando se almeja estender o alcance da rede, criando ramificações. Valem mencionar que pode funcionar como um Dispositivo Final (BRONZATTI, 2013).

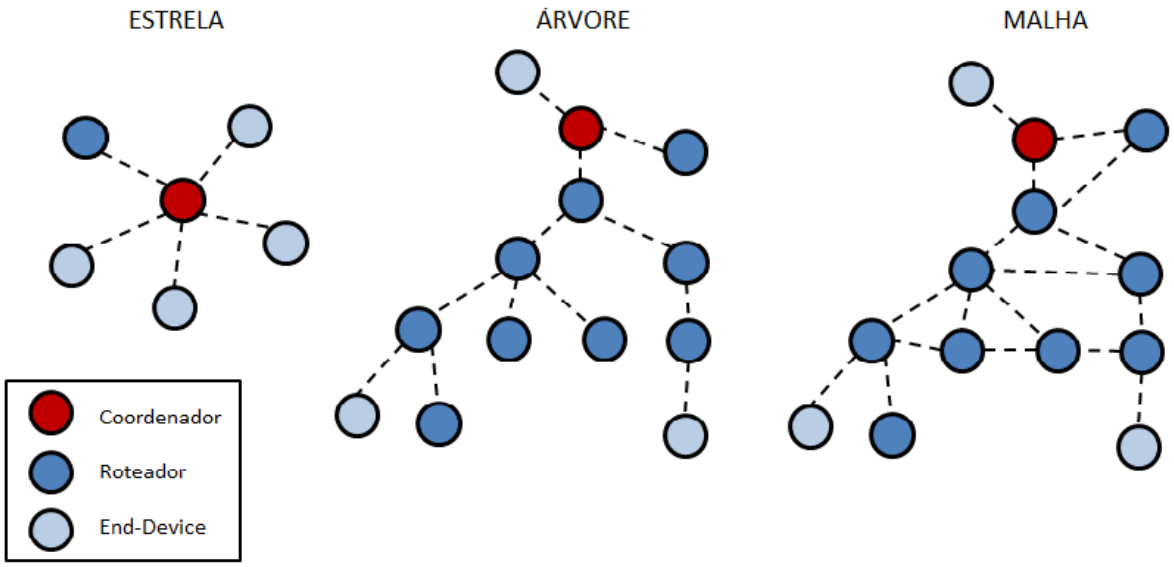
O Dispositivo Final (End-Device) apenas transmite ou recebe mensagens, sem reencaminhar para outros dispositivos, como fazem os roteadores. Além disso não são capazes de conectar dispositivos novos à rede. Trata-se no caso de dispositivos do tipo RFD. São tipicamente alimentados por bateria e costumam entrar em modo de suspensão, para conservação de energia (BRONZATTI, 2013).

1.7.4 Topologias de rede

Quanto à forma na qual os dispositivos se agrupam em uma rede para se comunicarem, há três topologias possíveis no padrão Zigbee, sendo elas: em estrela, em árvore e em malha.

As topologias do padrâo Zigbee são representadas na Figura 22 a seguir.

Figura 22 – Topologias do padrão Zigbee



Fonte: (BRONZATTI, 2013)

A topologia em estrela se trata daquela mais simples, indicada para ambientes que ofereçam poucos obstáculos de transmissão. Nesta topologia um nó central é conectado com todos os outros nós da rede. Essa topologia deixa a cargo do coordenador todo o controle da rede, assumindo este um papel central e de comunicação direta com todos os dispositivos finais. Uma falha no nó central fará todas as comunicações da rede serem cortadas, já que não há de rotas alternativas (BRONZATTI, 2013).

A topologia em malha, é aquela no qual os dispositivos do tipo FFD (Coordenador e Roteador) são livres de comunicar com outro dispositivo FFD. Isso permite, quando necessária, a expansão física da rede possibilitando um maior alcance. Assim, criam-se caminhos redundantes entre dois nós e a estrutura torna-se resiliente com rotas alternativas em caso de falha de algum dispositivo. Um Coordenador registra toda a entrada e saída de dispositivos, mas não assume um papel tão importante em termos de fluxo de informação como na configuração anterior (BRONZATTI, 2013).

A topologia em árvore,assim como a em malha, também faz uso de Roteadores, todavia se efetua a distribuição de dados e mensagens de controle numa estrutura hierárquica. Nesta hierarquia, o coordenador assume o papel de nó central da rede, no qual recebe a transmissão de dados dos dispositivos roteadores da rede. Caso um dispositivo venha a falhar, algumas rotas serão interrompidas, mas nem todas elas, como ocorre na topologia em estrela (BRONZATTI, 2013).

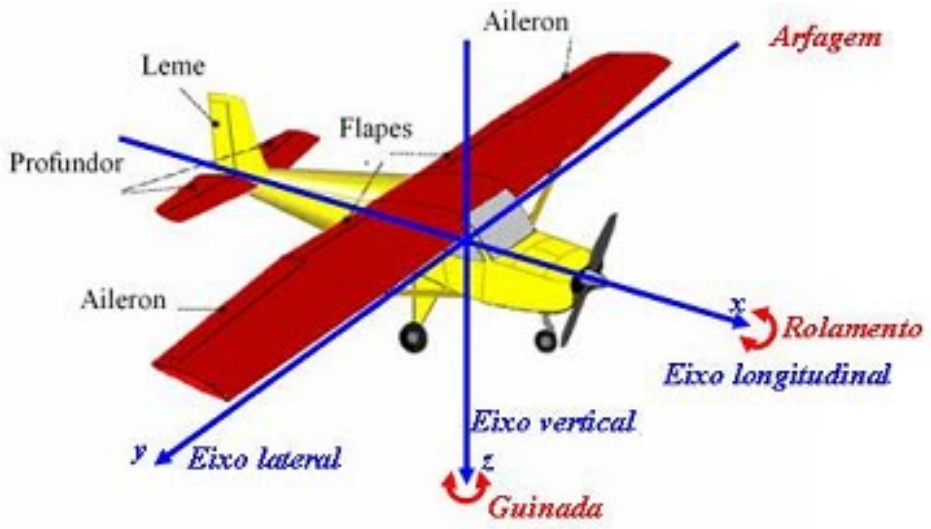
1.8 AERONÁUTICA

Nesta seção são apresentados os conceitos relacionados à área de aeronáutica que foram de relevância para a implementação do presente trabalho.

1.8.1 Eixos de um aeronave

Os eixos de uma aeronave são três linhas imaginárias que passam pelo CG da mesma, que podem ser definidas como eixos em torno dos quais a aeronave gira. Os três eixos passam pelo CG perpendicularmente entre si. O eixo que passa pelo CG e paralelo a uma linha do nariz à cauda é o eixo longitudinal, o eixo que passa paralelamente a uma linha da ponta da asa até a ponta da asa é o eixo lateral, e o eixo que passa pelo CG em ângulo reto com o eixo outros dois eixos é o eixo vertical. Sempre que uma aeronave altera sua atitude ou posição de vôo durante o vôo, ela gira em torno de um ou mais dos três eixos. A Figura 23 apresenta os eixos (FAA, 2016).

Figura 23 – Eixos de uma aeronave e superfícies de comando



Fonte: (RODRIGUES, 2014)

Os nomes usados para descrever o movimento sobre os três eixos de uma aeronave eram originalmente termos náuticos, que foram adaptados para terminologia aeronáutica devido à similaridade de movimento de aeronaves e navios de mar. O movimento em torno do eixo longitudinal da aeronave é denominado rolamento; o movimento em torno do eixo lateral, arfagem; e o movimento em torno de seu eixo vertical, guinada (FAA, 2016).

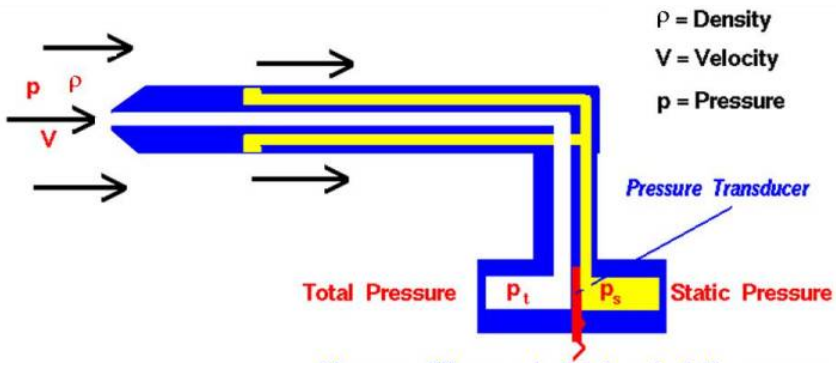
Os três movimentos, rolamento, arfagem e guinada, são controlados por três superfícies de controle. Em uma aeronave de configuração comum, o rolamento é controlado pelos ailerons, o arfagem é controlado pelo profundor, e a guinada é controlada pelo leme. Estas superfícies de comando são ilustrados na Figura 23. Outros tipos de aeronaves podem utilizar diferentes métodos de controle dos movimentos sobre os diversos eixos (FAA, 2016).

1.8.2 Velocidade em relação ao ar

A velocidade em relação ao ar, como o nome sugere, é a velocidade de uma aeronave relativa ao ar. São definidas por convenção classificações distintas de velocidade em relação ao ar, de acordo com o nível de tratamento dado ao dado, sendo elas: a velocidade indicada (IAS), velocidade calibrada (CAS), velocidade equivalente (EAS) e velocidade verdadeira (TAS) (FAA, 2016).

O equipamento pelo qual este dado é obtido é o conjunto de tubo de Pitot e tomada estática, representadas na Figura 24.

Figura 24 – Tubo de Pitot e tomada estática



Fonte: (NASA, 2015)

O tubo de Pitot é utilizado para medir a pressão total que se encontra incidente quando uma aeronave se move pelo ar. A pressão estática, também conhecida como pressão ambiente, se trata da pressão barométrica na área local. A pressão dinâmica está presente apenas quando uma aeronave está em movimento; portanto, pode ser pensado como uma pressão devido ao movimento. O vento também gera pressão dinâmica. Desse modo, conhecida a pressão dinâmica, é possível calcular a velocidade (FAA, 2016).

A velocidade indicada (IAS) se trata da velocidade indicada no velocímetro, isto é, é a velocidade sem a correção para variações na densidade atmosférica, erro de instalação ou erro do instrumento. Usada como referência em procedimentos de decolagem, pouso e cruzeiro (FAA, 2016).

A velocidade calibrada (CAS) é definida como IAS corrigida para erros de instalação e erros de instrumento. Embora os fabricantes tentem minimizar os erros de velocidade no ar, não é possível eliminar todos os erros ao longo da faixa de operação da velocidade aerodinâmica (FAA, 2016).

A velocidade equivalente (EAS) se trata da CAS corrigida para a compressibilidade do ar em um número Mach não trivial. Em voos de baixa velocidade, é a velocidade que seria mostrada por um indicador de velocidade no ar com erro zero. É usada para calcular a estabilidade e controle durante voo, cargas aerodinâmicas etc. (FAA, 2016).

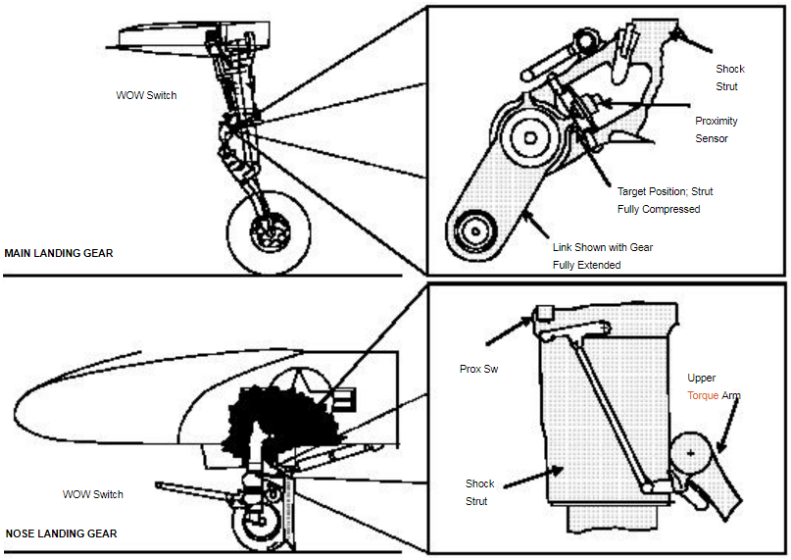
A velocidade verdadeira (TAS) é definida como EAS corrigida para altitude e temperatura. Como a densidade do ar diminui com o aumento da altitude, uma aeronave precisa voar mais rápido em altas altitudes para causar a mesma diferença de pressão entre a pressão de impacto do pitot e a pressão estática. Logo para um determinado EAS, o TAS aumenta conforme a altitude aumenta (FAA, 2016).

1.8.3 Wheight-On-Wheels

O Wheight-On-Wheels é definida como um indicador que diz se a aeronave tem peso sobre suas rodas, isto é, se a aeronave se encontra no ar ou no solo. O peso nas rodas pode ser detectado por um sensor nas rodas, calculado a partir de outros dados do estado ou por uma combinação destes (FAA, 2016).

Exemplos de sensores de WOW são ilustradas na Figura 25, a seguir.

Figura 25 – Tubo de Pitot e tomada estática



Fonte: (NAVYFLIGHTMANUALS, 2018)

A maioria da detecção é realizada por meio de um interruptor ou sensor no trem de pouso. O interruptor abrirá ou fechará sempre que o peso da aeronave for transferido ou retirado do trem de pouso. Por este motivo, estes interruptores são referidos como sensores de agachamento ou peso sobre rodas (*wheight on wheels* em inglês). Enquanto a maioria das aeronaves tem os interruptores de agachamento localizados no trem de pouso principal, alguns usam o trem de pouso do nariz para essa finalidade (FAA, 2016).

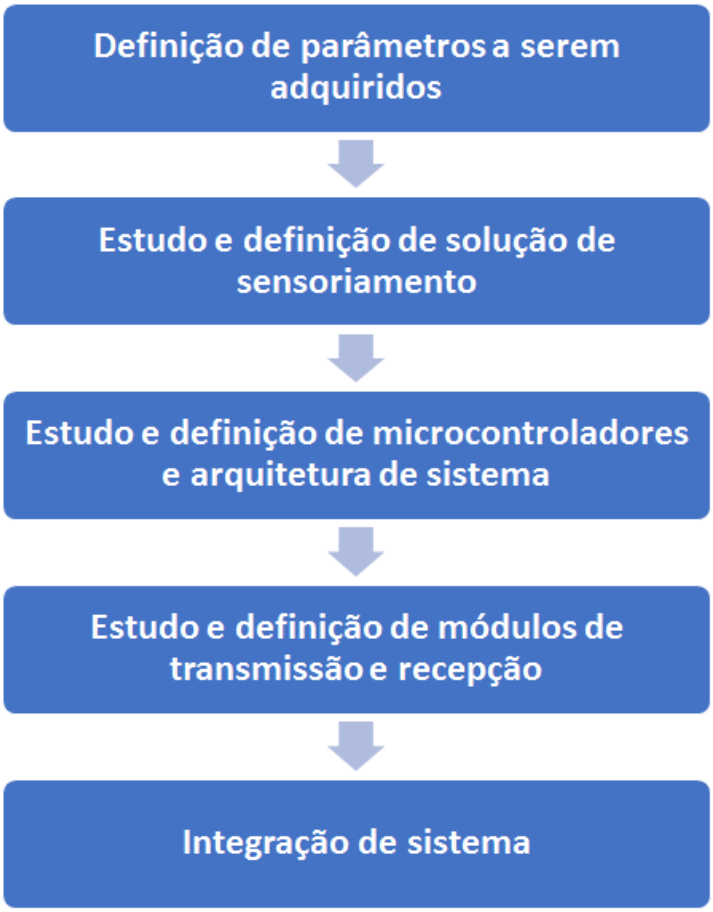
2 METODOLOGIA

O trabalho apresentado se trata de uma pesquisa aplicada, objetivando a realização de pesquisa experimental que é fundamentado em um material bibliográfico, de laboratório e de campo. São aplicados os procedimentos técnicos de pesquisa bibliográfica e experimental. Utiliza-se o método de abordagem hipotético-dedutivo e o método de procedimento monográfico em sua elaboração. Para a coleta de dados é utilizada a observação direta intensiva e documentação indireta, assim como a análise e interpretação de seus dados qualitativos, ocorre globalmente.

O trabalho consiste no desenvolvimento de um sistema embarcado, para veículos aéreos não tripulados, de aquisição de dados, por meio de sensores pertencentes ao sistema, e com transmissão dos parâmetros adquiridos para módulo terrestre.

As etapas das atividades realizadas nesse trabalho está resumido no diagrama de blocos apresentado na Figura 26:

Figura 26 – Diagrama de blocos para etapas do projeto



Fonte: Autor

2.1 DEFINIÇÃO DE DADOS A SEREM ADQUIRIDOS

Inicialmente foram definidos os dados a serem adquiridos pelo sistema embarcado. A Figura 27 apresenta a seleção. A escolha dos dados foi motivada para fins de monitoramento e de validação de projetos aeronáuticos através de parâmetros chave. Cada dado é tratado no capítulo seguinte individualmente, aprofundando o motivo de sua escolha, assim como detalhando a solução implementada.

Figura 27 – Relação de dados adquiridos pelo sistema embarcado

Item	Parâmetro	Unidade	Sensor/Dispositivo
1	Tempo	seg	RTC
2	RPM	rpm	Sensor Hall + Imãs de neodímio
3	WOW	bit	Sonar ultrassônico
4	HP	ft	Sensor de pressão barométrico
5	VCAS	m/s	Sensor de pressão diferencial + Tubo de Pitot
6	XGPS	deg	GPS
7	YGPS	deg	GPS
8	ZGPS	m	GPS
9	MAGHEAD	deg	IMU
10	ELEV	deg	Sinal de PWM de servomotores
11	AIL	deg	Sinal de PWM de servomotores
12	RUD	deg	Sinal de PWM de servomotores
13	P	deg/s	IMU
14	Q	deg/s	IMU
15	NZ	g	IMU
16	THETA	deg	IMU
17	PHI	deg	IMU

Fonte: Autor

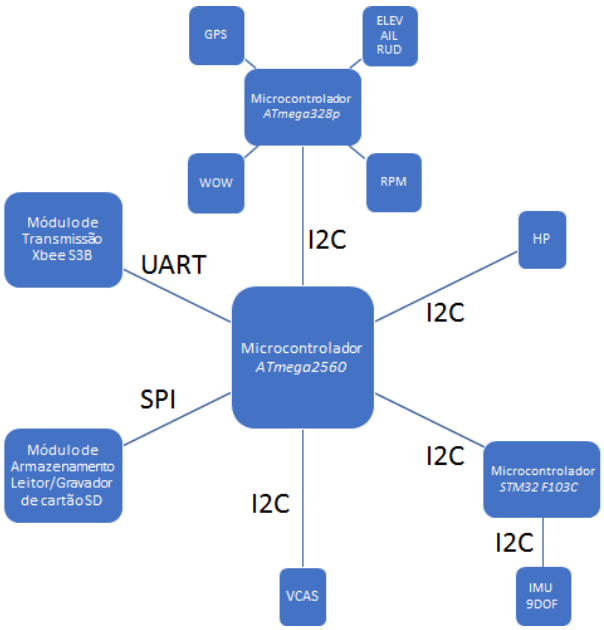
- **Tempo**: Referência de tempo para registro de dados
- **RPM**: Revoluções por minuto do motor
- **WOW**: *Weight on Wheels*, binário que indica se a aeronave se encontra em solo ou em voo
- **HP**: Altitude barométrica
- **VCAS**: Velocidade aerodinâmica calibrada
- **XGPS, YGPS, ZGPS**: Coordenadas de longitude, latitude e altitude, respectivamente
- **MagHead**: Bússola digital
- **ELEV, AIL, RUD**: Deflexão de profundor, aileron direito e leme, respectivamente
- **P, Q**: Taxa de rolamento e de arfagem, respectivamente
- **NZ**: Fator de Carga em eixo Z
- **THETA, PHI**: Inclinação de arfagem e de rolamento, respectivamente

2.2 ARQUITETURA DO SISTEMA EMBARCADO

Uma vez definidos os parâmetros, estudou-se as soluções de sensoriamento possíveis, para escolher o sensor mais viável. Consultou-se os dados de fabricantes e realizou-se testes práticos.

Posteriormente especificou-se um microcontrolador que atenda às necessidades de interface e de processamento para os sensores. Definiu-se então módulos de armazenamento e de telemetria. A Figura 28 apresenta a arquitetura do sistema de aquisição de dados.

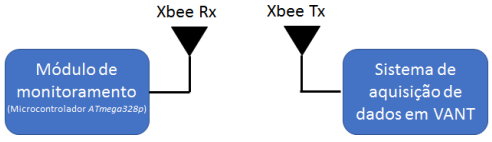
Figura 28 – Arquitetura do sistema de aquisição de dados



Fonte: Autor

Quanto à telemetria, foi escolhido o padrão Zigbee como protocolo de transmissão, de modo a garantir a confiabilidade dos dados sem a necessidade de algoritmos de baixo nível para contornar possíveis problemas de ruído e perda de pacotes. Para o radioenlace, definiu-se distância de 3 km como meta, uma vez que atenderia durante todo o percurso. Pertencente ao sistema há ainda o módulo de monitoramento terrestre, o qual recebe os dados transmitidos em tempo real pelo sistema de aquisição de dados embarcado no VANT. A Figura 29 ilustra a arquitetura de telemetria do sistema.

Figura 29 – Arquitetura do sistema de telemetria



Fonte: Autor

2.3 VALIDAÇÃO DO SISTEMA

Durante a fase de implementação, cada módulo de sensor é tratada individualmente, de forma que foram realizados experimentos para garantir a aquisição fiel de cada dado. Estes testes realizados durante a fase de implementação serão brevemente comentados no Capítulo 3, quando pertinente.

Por fim, o projeto foi integrado a uma aeronave rádio-controlada disponibilizada pelo projeto de extensão Urutau Aerodesign, com a qual foram realizados voos teste com o intuito de validar o sistema. O VANT em questão, denominado "Caboquinha", se encontra apresentado na Figura 30. No Capítulo 4 serão expostos os resultados obtidos durante os voos, assim como serão analisados e discutidos os dados obtidos.

Figura 30 – VANT "Caboquinha"do projeto de extensão Urutau Aerodesign

Fonte: Autor

O VANT se trata de uma aeronave cargueira rádio controlado, projetado para missões em regime de baixo número de Reynolds. Apresenta configuração de asa monoplano, empenagem convencional e motor a combustão.

Os voos experimentais foram realizadas nas pistas disponíveis na cidade universitária da UEA, além disso realizou-se voos com o sistema em São José dos Campos, durante a competição da qual a equipe participa. As Figuras 31 e 32 apresentam as pistas utilizadas.

Figura 31 – Pista utilizada para decolagem e pouso de voo em Manaus

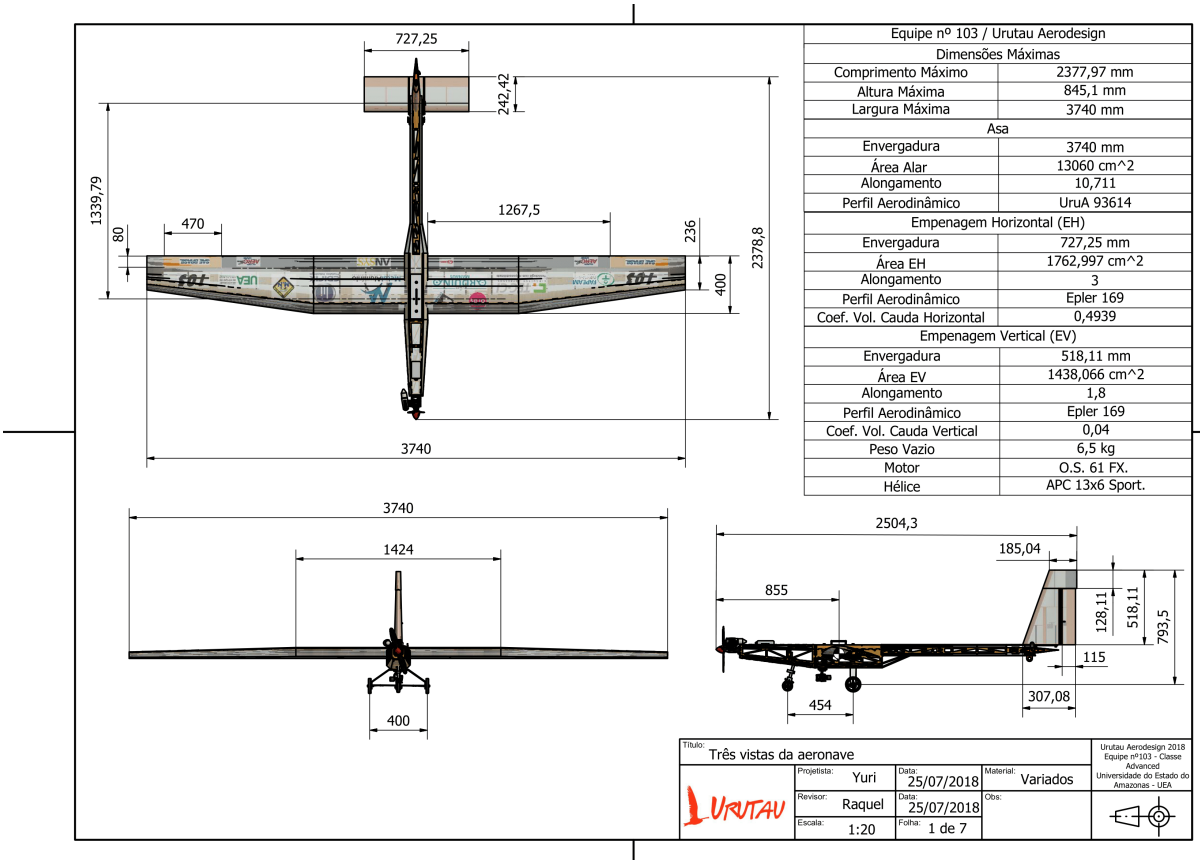
Fonte: (ARCGIS, 2018)

Figura 32 – Pista utilizada para decolagem e pouso de voo em São José dos Campos

Fonte: (ARCGIS, 2018)

As especificações técnicas da aeronave se encontram dispostos na Figura 33 a seguir.

Figura 33 – Especificações técnicas do VANT "Caboquinha"do Urutau Aerodesign



Equipe nº 103 / Urutau Aerodesign	
Dimensões Máximas	
Comprimento Máximo	2377,97 mm
Altura Máxima	845,1 mm
Largura Máxima	3740 mm
Asa	
Envergadura	3740 mm
Área Alar	13060 cm^2
Alongamento	10,711
Perfil Aerodinâmico	UruA 93614
Empenagem Horizontal (EH)	
Envergadura	727,25 mm
Área EH	1762,997 cm^2
Alongamento	3
Perfil Aerodinâmico	Epler 169
Coef. Vol. Cauda Horizontal	0,4939
Empenagem Vertical (EV)	
Envergadura	518,11 mm
Área EV	1438,066 cm^2
Alongamento	1,8
Perfil Aerodinâmico	Epler 169
Coef. Vol. Cauda Vertical	0,04
Peso Vazio	6,5 kg
Motor	O.S. 61 FX.
Hélice	APC 13x6 Sport.

Fonte: (URUTAU, 2018)

3 IMPLEMENTAÇÃO

O presente capítulo vem por descrever os passos seguidos para a implementação do projeto proposto.

3.1 MÓDULOS DE AQUISIÇÃO DE DADOS

Esta seção apresenta os módulos de aquisição de dados, apresentando os componentes utilizados, os princípios de funcionamento, além de explanar as razões acerca às decisões de projeto tomadas.

3.1.1 Tempo

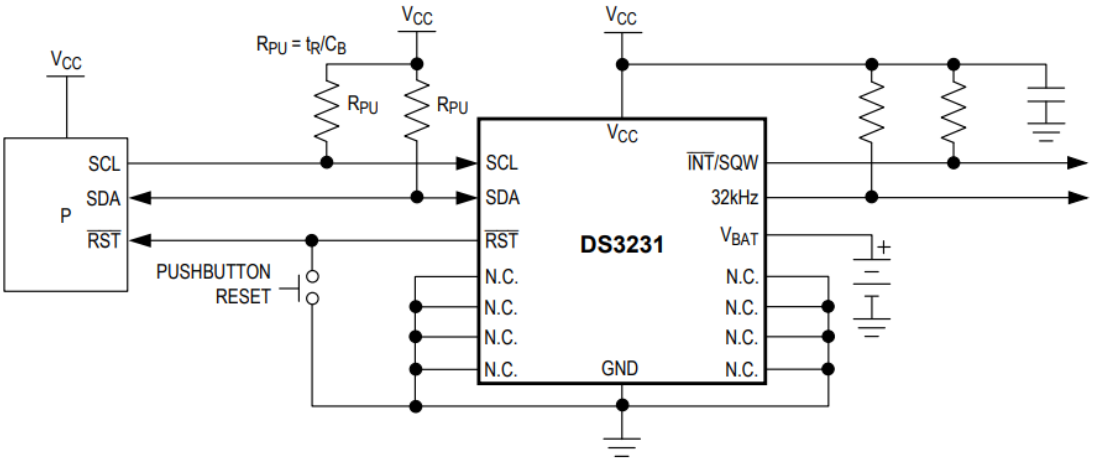
Para fins de datalogging, foi implementado um relógio embarcado ao sistema. O componente utilizado foi o RTC DS3231, caracterizado pelo baixo custo e alta precisão, é capaz de manter informações de segundos, minutos, horas, dia, data, mês e ano. Este módulo se encontra apresentado na Figura 34.

Figura 34 – Relógio de tempo real - RTC DS3231

Fonte: (ALIEXPRESS, 2018)

A Figura 35 apresenta o esquemático proveniente do datasheet.

Figura 35 – Esquemático de RTC DS3231



Fonte: (MAXIM, 2015)

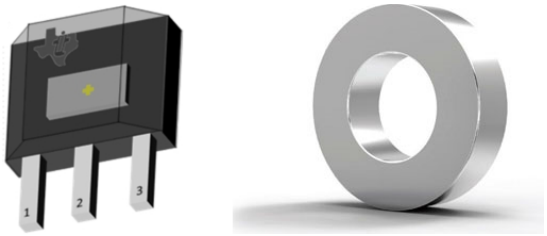
O módulo inclui uma pilha de backup, de modo a manter o tempo mesmo com sistema desenergizado, dessa forma o sistema independe de fatores externos para registrar esse dado, conforme representado no esquemático.

No caso a unidade da gravação é em segundos, iniciando-se às 00:00:00 do dia (horário de Brasília). Uma vez que o sistema grava dados a uma amostragem de 10 Hz, é pertinente o registro do tempo representar também decissegundos. Para isso a unidade de aquisição complementa o registro de tempo, comparando-se com o seu registro de tempo desde o último reset.

3.1.2 RPM

Para obter a rotação do motor por minuto, inicialmente, foram feitos testes com sensores infravermelhos de modelos variados. Porém, foi observado que esses sensores sofrem interferência com a luz solar, não oferecendo resultados coerentes, sendo portanto descartado. Em seguida, deu-se início a testes com sensores hall, com o qual obtiveram-se resultados satisfatórios quando utilizado com ímãs de neodímio. Ambos se encontram representados na Figura 36.

Figura 36 – Sensor Hall DRV5013AD e Imã de Neodímio



Fonte: (TI, 2018) e (ALIEXPRESS, 2018), adaptado por autor

Após análises das folhas de dados e testes práticos com diversos sensores hall, selecionou-se o DRV5013AD, que possui uma porta digital, uma de alimentação e outra de referência. Tem ótimo nível de histerese magnética, como mostrado na Figura 37, o que implica em tempo de resposta rápido, necessário para o nível de RPM alcançados pelo motor.

Figura 37 – Resposta a campo magnética de DRV5013AD

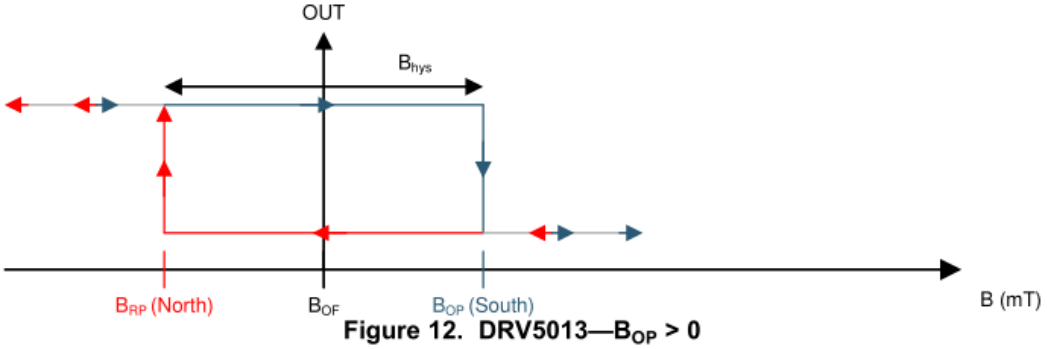


Figure 12. DRV5013—B_{OP} > 0

Fonte: (TI, 2018)

Assim sendo, imãs de neodímio em formato de arruela foram parafusados ao spinner, com cerca de 5 mm de distância do sensor hall, o que proporcionou boa medição de RPM, boa fixação dos imãs e do sensor, sem causar desbalanceamento ou desprendimento.

A Figura 38 apresenta detalhes da instalação.

Figura 38 – Detalhe de instalação de sensor RPM

Fonte: Autor

Quanto ao código implementado no firmware do sistema, foi feito uso de rotinas de interrupção para a leitura do estado do sensor. O código está disposto na Figura 39.

No caso o barramento de sinal do sensor Hall foi ligado a um dos pinos com funcionalidade de interrupção do microcontrolador. De modo que a cada transição de *HIGH* para *LOW*, incrementa-se a variável de contador. Devido à alta frequência de processamento do microcontrolador, as interrupções não afetam a correta execução do firmware, mesmo para rotação máxima do motor.

Figura 39 – Trecho de firmware contendo rotina de interrupção

```
uint16_t counter = 0;
uint32_t last_millis = 0;

#define interrupt_in 1    // Pin used as input for Hall Sensor signal
#define magnets      2    // Quantity of magnets on spinner/interrupts per cycle
#define period       1000 // [ms]

// RPS interruption routine - increments counter
void rps_ISR()
{
  counter++;
}

// RPS loop executed in main loop
void rps_loop()
{
  if (millis() - last_millis >= period)
  {
    detachInterrupt(interrupt_in);                    // Disable interrupt while calculating
    rps = counter / magnets ;                         // Encoder conversion

    rpsArray[0] = rps >> 8;                           // Stores data in buffer for I2C packet
    rpsArray[1] = rps;

    counter = 0;                                      // Restart counter
    last_millis = millis();                           // Update lastmillis
    attachInterrupt(interrupt_in, rps_ISR, FALLING);  // Reenable interrupt
  }
}
```

Fonte: Autor

3.1.3 WOW

O dado *Wheight on Wheels* (WOW) se trata de um binário que representa se a aeronave se encontra em solo ou em voo. De fato em aeronaves de grande porte tipicamente se usa células de carga no trem de pouso, originando o termo.

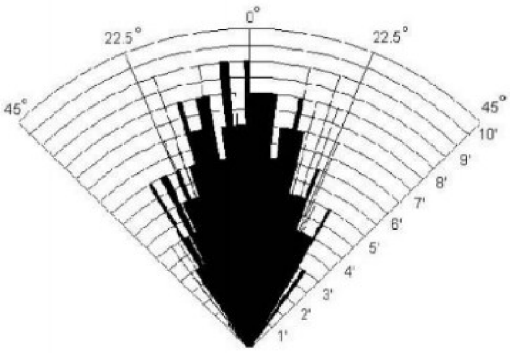
Para este dado, optou-se pelo sensor ultrassônico HC-SR04 localizado na fuselagem, imediatamente atrás do trem de pouso. O sensor está representado na Figura 40. Princípio de funcionamento se dá pela emissão de um sonar, medindo-se o tempo gasto pelo som ir e voltar ao microfone, calcula-se distância, logo determina-se se a aeronave se encontra em solo ou não. Funcionou plenamente durante voos experimentais.

Figura 40 – Sensor ultrassônico HC-SR04

Fonte: (ALIEXPRESS, 2018)

Possui garantia de detecção para um ângulo de $\pm 15°$, resolução de 3mm e margem de operação de 20mm a 500mm, logo adequado para aplicação. Para a distância dessa aplicação em particular, a margem de detecção angular é maior que o nominal, conforme testes experimentais encontrados no datasheet do dispositivo, apresentados na Figura 41.

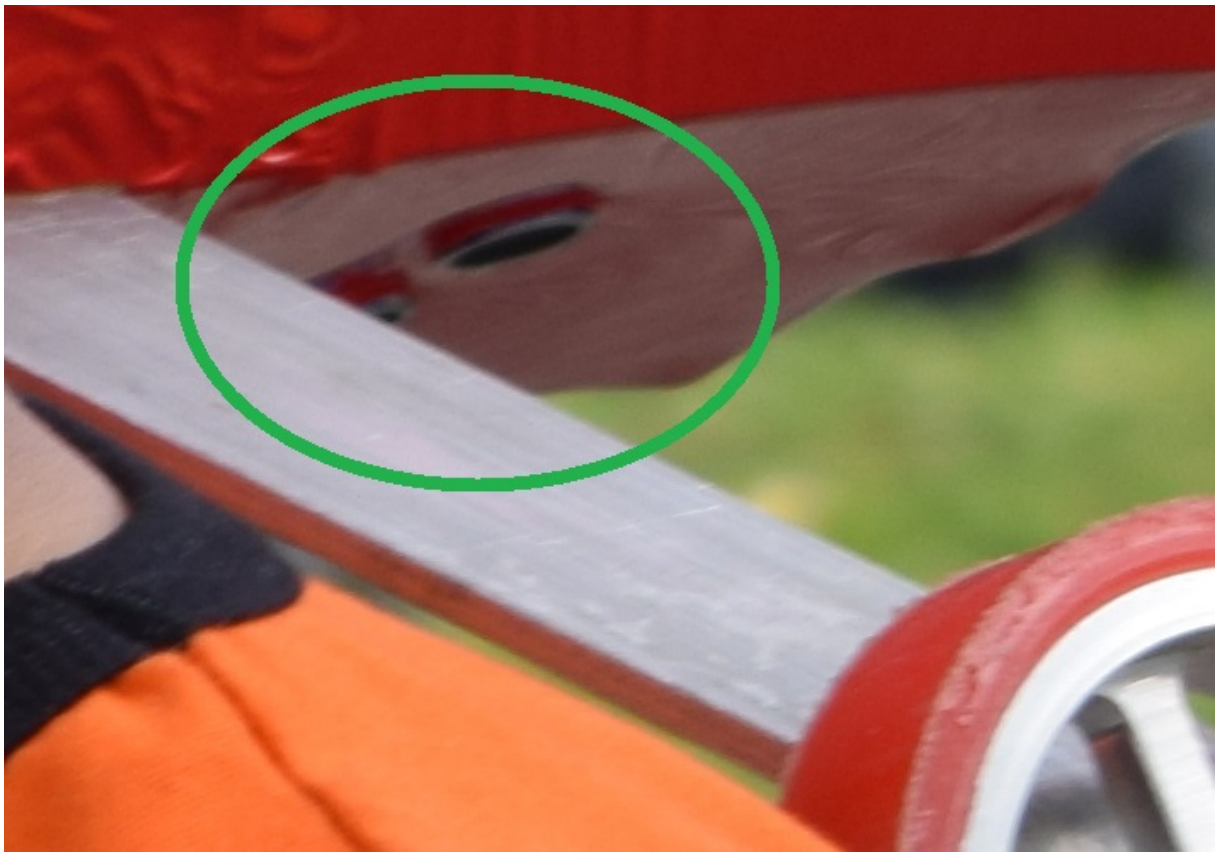
Figura 41 – Alcance por distância e ângulo



Fonte: (MOUSER, 2016)

A Figura 42 apresenta detalhes da instalação do sensor.

Figura 42 – Detalhe de instalação de sensor WOW



Fonte: Autor

Quanto ao posicionamento, buscou-se deixá-lo o mais próximo possível do trem de pouso, pois na decolagem e na aterrissagem, esta estrutura é respectivamente a última e a primeira a ter contato com o solo. Detalhes da instalação podem ser observados na Figura 42.

3.1.4 HP

Em se tratando de sensor de pressão barométrico, o que se teve por mais viável no mercado é o BME680. Este se refere ao sensor da Bosch que apresenta ótimo custo benefício. Possui baixo custo ao mesmo tempo que tem excelente precisão. Trata-se de um sensor ambiental integrado, isto é, apresenta múltiplos sensores em um único dispositivo. O BME680 apresenta além do sensor de pressão, sensores para temperatura, gases e umidade do ar. O sensor está apresentado na Figura 43.

Figura 43 – Sensor pressão barométrica BME680

Fonte: (ALIEXPRESS, 2018)

A plataforma dispõe de duas interfaces de comunicação, uma I2C e outra SPI, para a implementação foi usado o barramento I2C. Pressão de referência usada é de 1013.25 hPa.

Adquire-se o valor de altitude pela Equação 1:

$$h = h_b + \frac{T_b}{L_b}\left[\left(\frac{P}{P_b}\right)^{-\frac{RL_b}{g_0 M}} - 1\right] \tag{1}$$

A pressão ambiental está sujeita a muitas mudanças a curto prazo, causando perturbações externas. Para suprimir distúrbios, como por exemplo a ação do vento, nos dados de saída sem causar carga de trabalho no microcontrolador, o sensor possui um filtro IIR interno. Este filtro reduz a largura de banda dos sinais de saída de temperatura e de pressão, aumentando a resolução dos dados de saída para 20 bits (BOSCH, 2017).

A saída de um próximo passo de medição é filtrada usando a Equação 2, apresentada a seguir.

$$x_{filt}[n] = \frac{x_{filt}[n-1]\cdot(c-1) + x_{ADC}}{c} \tag{2}$$

O filtro IIR pode ser configurado para diferentes coeficientes de filtro, que atrasam a resposta às entradas do sensor. Esses parâmetros, assim como a taxa de amostragem, podem ser configurados escrevendo os valores nos respectivos registradores do dispositivo.

3.1.5 VCAS

Para a velocidade aerodinâmica calibrada, optou-se por tubo de pitot e tomada estática acopladas a um sensor de pressão diferencial. Apresenta-se o conjunto na Figura 44.

Figura 44 – Sensor pressão diferencial SDP600 e tubo de pitot

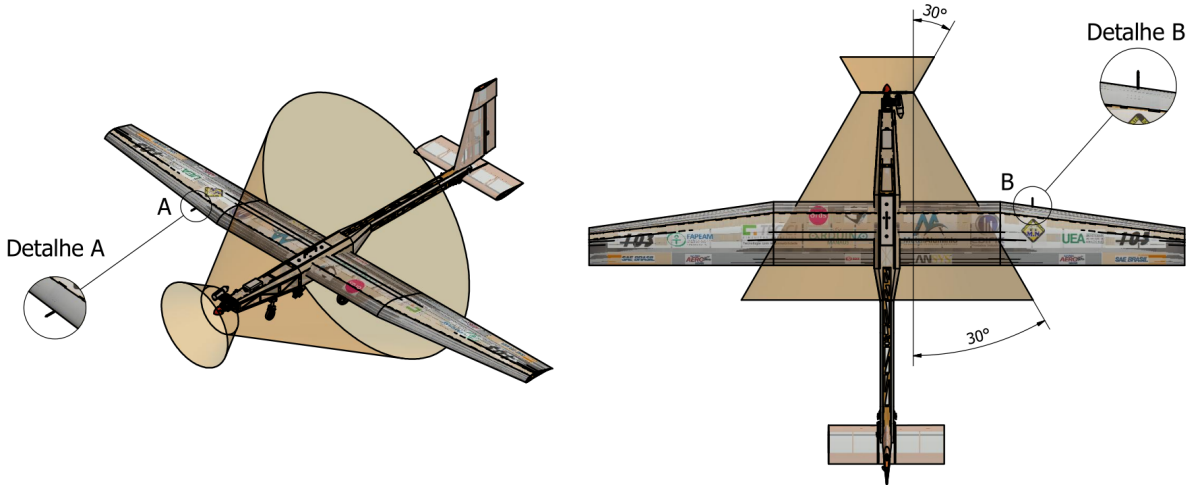
Fonte: (ALIEXPRESS, 2018), adaptado por autor

A partir da Equação de Bernoulli, a diferença entre pressão total e estática, ou seja, a pressão dinâmica, pode-se obter a velocidade através da Equação 3.

$$V_{CAS} = \sqrt{\frac{2\Delta P}{\rho}} \tag{3}$$

O tubo de Pitot foi alocado fora da região de turbulência causada pelo giro da hélice, considerandou-se um cone de 30º, e em uma região aerodinamicamente favorável, em que as partículas de ar não tenham sofrido influência das superfícies, como se pode ver na Figura 45.

Figura 45 – Posicionamento tubo de Pitot



Fonte: Autor

Selecionou-se o Sensiron SDP600-500Pa, caracterizado por apresentar margem de registro $\pm$ 500 Pa, resolução de 16 bits e comunicação em I2C. Portanto verifica-se que a velocidade máxima que consegue registrar, para densidade do ar de 1,052 kg/m^3 é de 30,83 m/s, superior à velocidade máxima projetada do VANT de 23 m/s. Priorizou-se a precisão durante a escolha deste sensor, justificando seu investimento.

Detalhes de instalação podem ser observados na Figura 46.

Figura 46 – Detalhe de instalação de sensor VCAS

Fonte: Autor

3.1.6 GPS

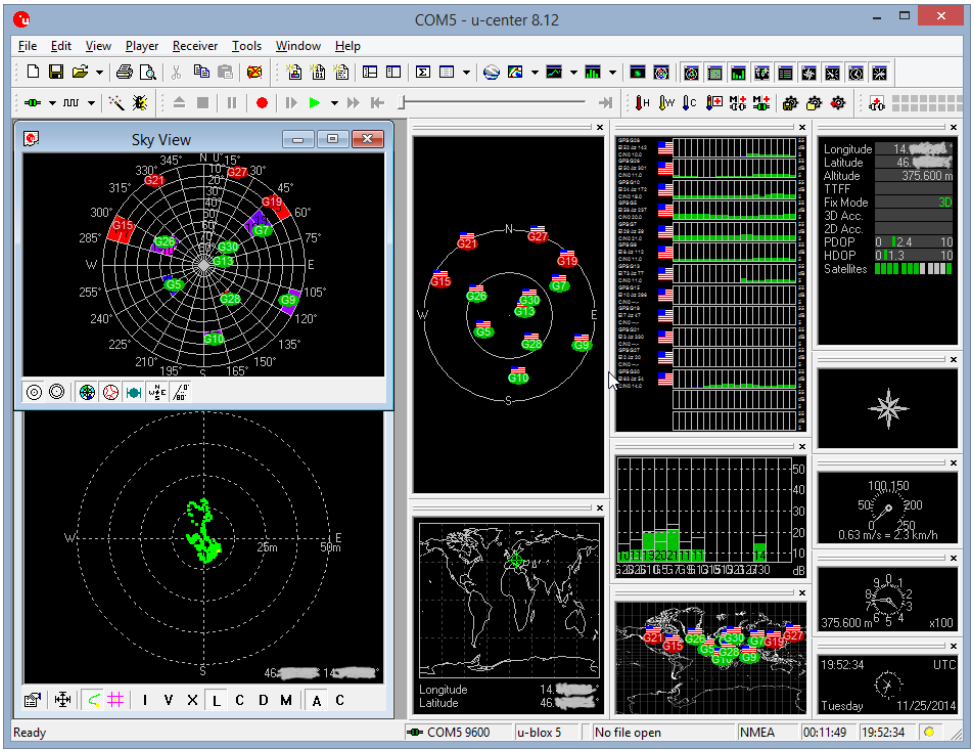
A aquisição de dados de posicionamento por GPS é obtida pelo módulo Ublox Neo M8N, cujo diferencial é o fato de que utiliza recepção simultânea de até três sistemas GNSS (GPS americano e Galileo europeu em conjunto com BeiDou chinês ou GLONASS russo), reconhecendo constelações múltiplas simultaneamente e proporcionando alta precisão. Representa-se o módulo na Figura 47. A comunicação com o microcontrolador se dá por UART.

Figura 47 – GPS Ublox Neo M8

Fonte: (ALIEXPRESS, 2018)

A configuração do módulo se deu pelo software da fabricante, o U-Center, apresentado na Figura 48. Com o uso de um adaptador serial foi possível se comunicar com o dispositivo diretamente por um computador. Inicialmente se configurou o módulo para a taxa de atualização de 10 Hz, no entanto foi constatado que o dispositivo possui um erro relativamente considerável. Assim sendo optou-se pela taxa de 1 Hz que seria o suficiente para a demanda da aplicação.

Figura 48 – Software U-center

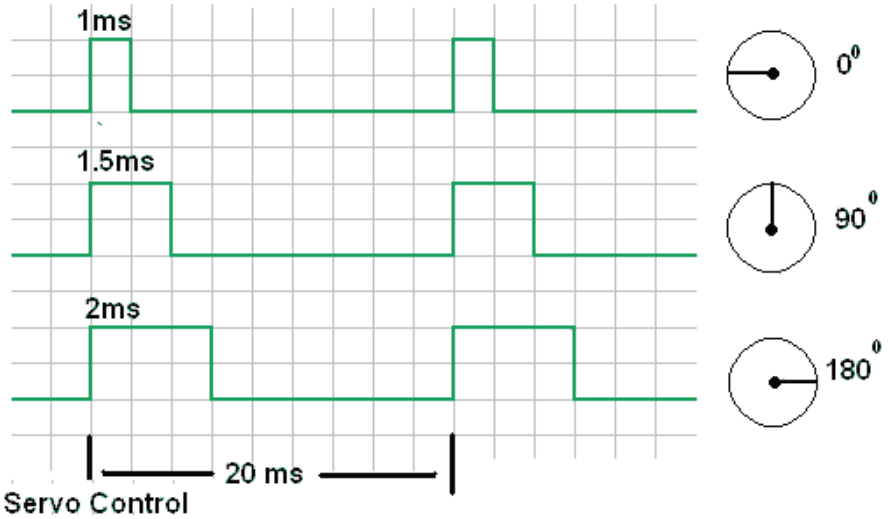


Fonte: (REILABS, 2016)

3.1.7 Deflexão de superfícies de comando

A solução implementada consiste na leitura do sinal de controle emitido aos servos. Trata-se de um sinal PWM cuja largura do período HIGH indica a posição do servo, princípio ilustrado na Figura 49.

Figura 49 – Diagrama de tempo do sinal de controle do servomotor, para 3 posições

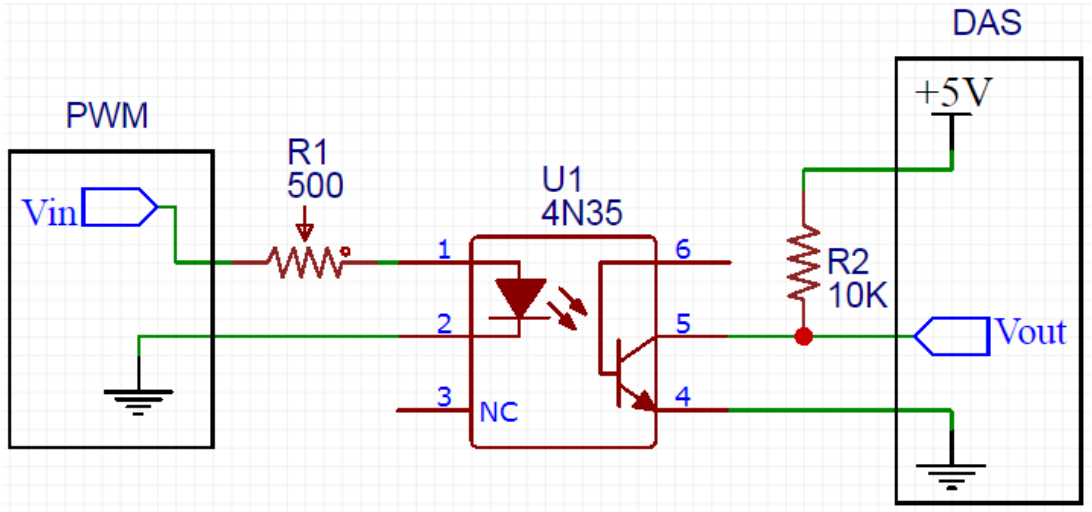


Fonte: (FRACAROLLI, 2012)

A deflexão varia em função da posição do servo, logo conhecida essa relação, é possível encontrar o dado desejado. São três as superfícies de controle selecionadas para leitura: o profundor, o leme e o aileron direito.

A Figura 50 apresenta o esquemático do circuito elaborado para leitura do sinal de PWM.

Figura 50 – Esquemático de circuito para leitura de PWM



Fonte: Autor

O receptor duplica os sinais de controle que vão para os servos em outros canais, estes canais estão conectados ao sistema embarcado. No caso o sinal advindo do receptor passa através de um optoacoplador, a fim de separar eletricamente os dois circuitos, entrando então na GPIO do microcontrolador auxiliar, que calcula o valor de deflexão. A Figura 51 ilustra o comportamento do sinal PWM ao passar pelo circuito de leitura, verifica-se que os estados de HIGH e de LOW ficam invertidos.

Figura 51 – Sinais de entrada e de saída do circuito de leitura de PWM

Fonte: Autor

Vale mencionar que este circuito foi elaborado visando proteger o microcontrolador de sobretensão, visto que o receptor opera com uma tensão de 6.6 V, enquanto que o microcontrolador com 5 V. Além disso o receptor fica protegido de quaisquer disfunções elétricas do circuito externo, que poderiam vir a comprometê-lo durante voo.

Quanto à programação, foi necessário dar um tratamento adicional ao sinal, devido à inconsistência da largura do sinal lido. A Figura 52 apresenta trecho do código que realiza a leitura do sinal PWM.

Figura 52 – Trecho de firmware contendo leitura do sinal PWM

```
pwm1_read = pulseIn(PINO_PWM1, LOW, 30000);
if (pwm1_read > 400)
{
  pwm1_raw = pwm1_read;
  cont1 = 0;
} else if (pwm1_read < 50)
{
  cont1++;
  if (cont1 >= 3)
     pwm1_raw = 0;
}

for (int i = n - 1; i > 0; i--) filter_1[i] = filter_1[i - 1];      //desloca os elementos do vetor de média móvel
  filter_1[0] = pwm1_raw;                                           //posição inicial do vetor recebe a leitura original
for (int i = 0; i < n; i++) acc_1 += filter_1[i];                   //faz a somatória do número de pontos
  pwm1_filtered =  acc_1 / n;                                       //retorna a média móvel
acc_1 = 0;                                                          //reinicia acumulador

servo1 = map (  pwm1_filtered, LimInf_pwm1, LimSup_pwm1, LimInf_servo1, LimSup_servo1);
```

Fonte: Autor

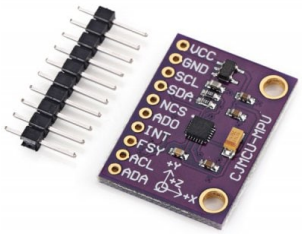
Para uma posição constante de deflexão, isto é, uma largura de PWM constante sendo transmitida pelo receptor, os valores lidos no microcontrolador apresentavam significativa variação, que resultavam em uma margem de erro de ± 4 graus. Assim sendo, o problema foi solucionado por meio do uso de um filtro de média móvel, com o qual a margem de erro foi reduzida para ± 1 grau e com atraso inferior a 1 segundo.

Além disso, esporadicamente havia ruído no sinal, que era interpretado erroneamente como sendo a largura do sinal PWM. A solução implementada para esse empecilho foi o uso de uma lógica condicional para expurgar esses valores falsos.

3.1.8 IMU

Embarcou-se uma unidade de medição de inércia de 9 eixos, o MPU9250 da Invensense, ilustrado na Figura 53, para aquisição de MagHead, P, Q, NZ, THETA e PHI. Trata-se de um CI com dois módulos, o giroscópio/acelerômetro MPU6050 e o magnetômetro AK8963.

Figura 53 – IMU Invensense MPU9250



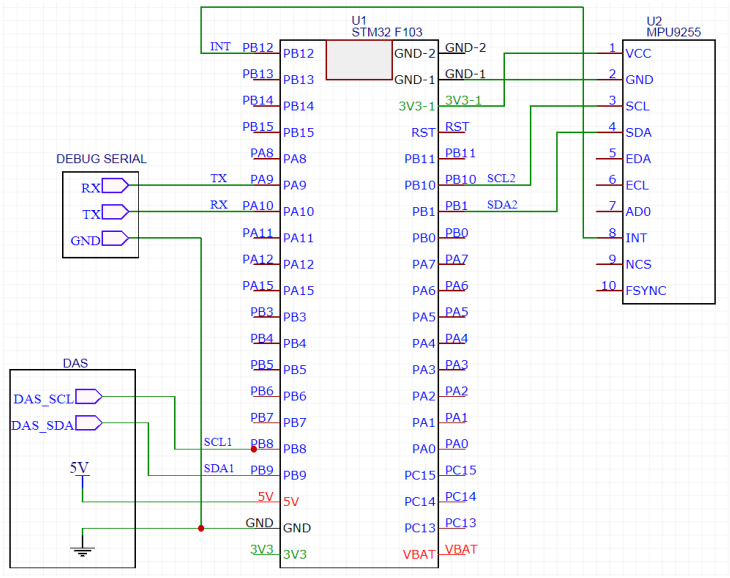
Fonte: (ALIEXPRESS, 2018)

A partir dos valores de rotação, aceleração e campo magnético fornecidos, por meio de filtro de fusão, estima-se a posição da aeronave em relação a seus eixos. A comunicação é por I2C. Posicionou-se na fuselagem o mais próximo possível do centro de gravidade da aeronave.

Durante a fase de implementação foi verificado que a arquitetura disposta não suportaria executar os algoritmos na velocidade esperada, devendo ou sacrificar precisão, rendendo os dados inúteis, ou reduzir a resolução de leitura de dados, também indesejável. Assim sendo, para este dado foi modularizado um microcontrolador dedicado para leitura dos sensores e processamento, aliviando esta última tarefa no microcontrolador principal.

A Figura 54 a seguir apresenta o esquemático do módulo IMU projetado.

Figura 54 – Esquemático do módulo IMU



Fonte: Autor

O microcontrolador para este módulo requer principalmente velocidade de processamento, então para tal foi selecionado o STM32 F103, cuja capacidade de processamento foi apresentada em seção anterior. Uma vez implementada, a aquisição dos dados passou a ser fluída.

A comunicação com o microcontrolador primário se dá através de protocolo I2C, mantendo portanto o hardware idêntico ao definido anteriormente, atuando como escravo. O dispositivo IMU envia os pacotes de dados quando requisitado pelo mestre, que se dá no microcontrolador do IMU através de uma rotina de interrupção.

Quanto ao à fusão de sensores implementado foi utilizado o algoritmo de Madgwick. Sebastian Madgwick desenvolveu um algoritmo de fusão de sensores IMU como parte de sua pesquisa de Ph.D. na Universidade de Bristol. O algoritmo foi posteriormente publicado como *open source* (MADGWICK, 2010).

O filtro emprega uma representação quatérnios de orientação para descrever a natureza acoplada das orientações em três dimensões, sem estar sujeito às problemáticas particulares associadas com uma representação do ângulo de Euler1 (MADGWICK, 2010).

3.2 MICROCONTROLADORES

Para a especificação do microcontrolador priorizou-se a velocidade e facilidade de programação, logo optou-se pelo Atmel Atmega 2560, compatível com o framework Arduino. Este microcontrolador possui portas GPIO suficientes para a aplicação, que está relacionado na Figura 55.

Figura 55 – GPIO exigidos no sistema de aquisição de dados

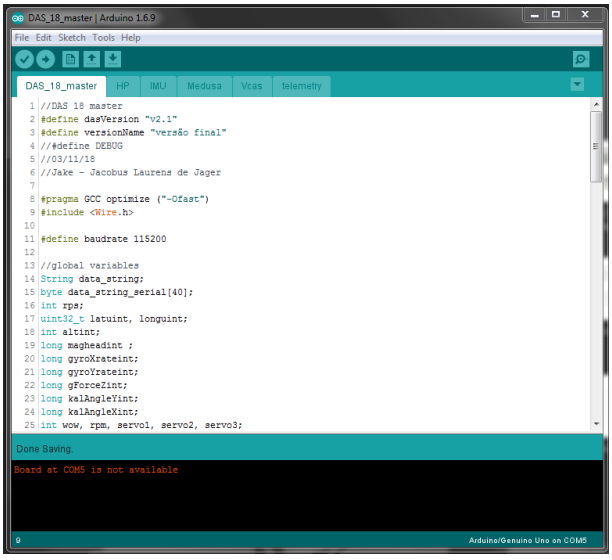
	Total	2560	328p
Pinos Digitais	4	0	4
Inteerrupção	1	0	1
I2C	5	4	1
UART	2	1	1
SPI	1	1	0

Fonte: Autor

Todavia durante prototipagem averiguou-se que não suportava amostragem simultânea de 10 Hz para todos os parâmetros implementados até então. Portanto implementou-se um microcontrolador auxiliar, o Atmega328p, objetivando aliviar a carga de processamento do primeiro. Por conveniência, ligou-se todos os módulos desprovidos de protocolo de comunicação ao microcontrolador auxiliar, assim a interface para o microcontrolador primário com esses módulos se dá por I2C.

A programação do firmware desses microcontroladores centrais foi desenvolvido em framework Arduino, utilizando a IDE disponibilizada pelo fabricantes das placas de prototipagem, que é ilustrada na Figura 56. A escolha dessa framework se deve pela facilidade de uso e disponibilidade imediata de microcontroladores compatíveis.

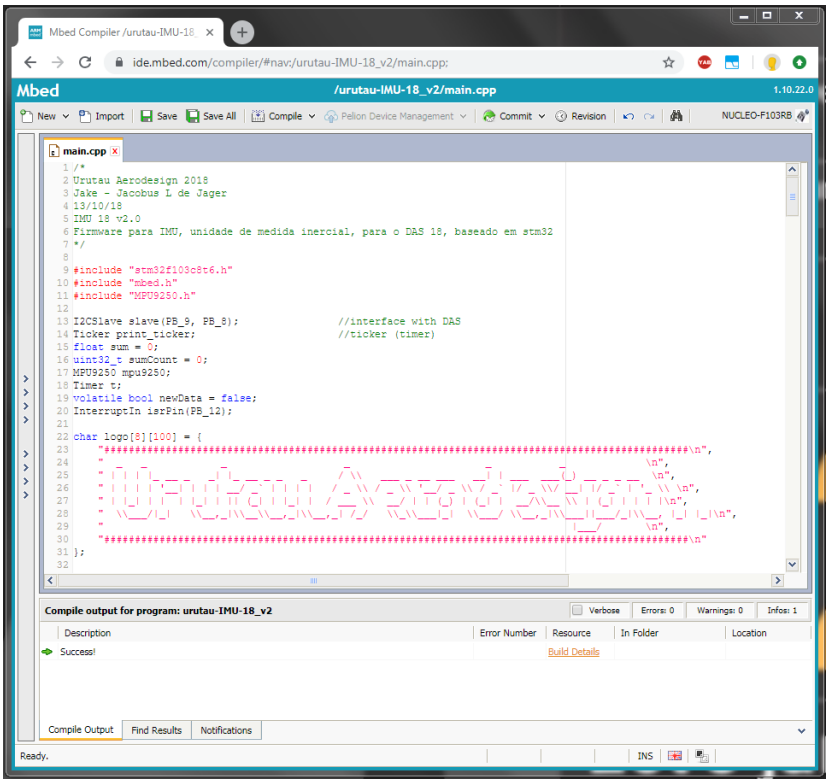
Figura 56 – IDE do Arduino



Fonte: Autor

Conforme descrito na seção anterior referente ao IMU, este módulo passou a contar com um microcontrolador de altíssima frequência dedicado para processamento de dados obtidos pelos sensores. O microcontrolador designado para esta tarefa foi o STM32 F103, cuja programação foi realizada em framework Mbed por meio do compilador online disponibilizada pela ARM, representada na Figura 57.

Figura 57 – Compilador online do MBED



Fonte: Autor

A Figura 58 apresenta as especificações técnicas dos microcontroladores utilizados no sistema embarcado.

Figura 58 – Especificações técnicas dos microcontroladores

	Atmega 2560	Atmega 328p	STM32 F103C
Clock [MHz]	16	16	72
EEPROM [kB]	4	1	-
SRAM [kB]	8	2	20
FLASH [kB]	256	32	64
Digital IO	54	16	48
ADC	16	6	9
UART	4	1	3
SPI	1	1	2
I2C	1	1	2
USB	-	-	1
CAN	-	-	1

Fonte: Autor

3.3 MÓDULO DE ARMAZENAMENTO

O sistema de aquisição de dados possui um módulo de cartão microSD para registrar os parâmetros de voo, tal qual uma "caixa preta". O módulo genérico utilizado possui interface de comunicação SPI. Encontra-se estrategicamente posicionado na fuselagem para rápida retirada após pouso. Representa-se o leitor de cartão na Figura 59.

Figura 59 – Leitor/Gravador de cartão microSD

Fonte: (ALIEXPRESS, 2018)

Gera-se texto com valores separados por tab, com primeiras duas linhas apresentando nomes e unidades dos parâmetros respectivamente e terceira linha em diante apresentando dados registrados, como apresentado na Figura 60. O intuito dessa formatação é tornar o arquivo preparado para manipulação em Excel ou em Matlab.

Figura 60 – Formatação de arquivo texto gerado pelo módulo SD

Tempo	RPM	WOW	HP	VCAS	XGPS	YGPS	ZGPS	MagHead	ELEV	AIL	RUD	P	Q	NZ	THETA	PHI
seg	rpm	bit	m	m/s	deg	deg	m	deg	deg	deg	deg	deg/s	deg/s	g	deg	deg
84725,0	0	0	611,03	0	-23.222.147	-45.866.817	642,23	300	-6	0	1	0,7	-0,07	1	0,31	-0,47
84725,1	0	0	611,03	0,33	-23.222.147	-45.866.817	642,23	295	-6	-1	1	2,37	0,62	0,97	1,31	-0,01
84725,2	0	0	611,03	0	-23.222.147	-45.866.817	642,23	298	-6	0	1	-0,43	0,03	1,05	0,22	-0,54
84725,3	0	0	611,03	0,62	-23.222.147	-45.866.817	642,23	301	-6	0	1	-0,37	-0,31	1,03	0,4	-0,33
84725,4	0	0	611,03	0,4	-23.222.147	-45.866.817	642,23	298	-6	1	1	0,21	-0,51	1	-0,04	-0,51
84725,5	0	0	611,03	0,33	-23.222.147	-45.866.817	642,23	300	-6	0	1	0,08	-0,55	0,98	-0,79	-0,62
84725,6	0	0	611,03	0,33	-23.222.147	-45.866.817	642,34	295	-6	0	1	-0,47	-0,58	0,98	-0,34	-0,08
84725,7	0	0	611,2	0	-23.222.147	-45.866.817	642,34	298	-6	-1	2	0,64	-0,42	0,98	-0,48	-0,08
84725,8	0	0	611,03	0	-23.222.147	-45.866.817	642,34	301	-6	0	2	0,54	-0,32	1	-0,36	-0,26
84725,9	0	0	611,03	0	-23.222.147	-45.866.817	642,34	298	-5	-2	2	-0,14	0,04	1,02	-0,38	-0,16

Fonte: Autor

3.4 MÓDULO DE TELEMETRIA

O sistema de aquisição de dados possui um módulo de transmissão de dados em tempo real sem-fio, para facilitar a verificação de dados durante as inspeções pré decolagem e monitoramento durante voo. Vale salientar que o sistema é constituído também de módulo terrestre, que recebe os dados em tempo real pelo sistema embarcado no VANT.

Quanto à justificativa de escolha do protocolo de transmissão, optou-se pelo padrão Zigbee para garantir a confiabilidade da transmissão de dados sem a necessidade de implementação de algoritmos de baixo nível para contornar possíveis problemas de ruído e perda de pacotes.

Utiliza-se o módulo Xbee S3B Pro, representado na Figura 61, para a transmissão de dados entre o sistema e um computador. A comunicação é realizada através de UART. Alcance do sinal é 6,5km utilizando-se antenas dipolo 2,1 dB, segundo fabricante. Configurou-se os dispositivos para operação ponto-a-ponto e em modo transparente (funcionalidade reduzida).

Figura 61 – Xbee S3B Pro

Fonte: (ALIEXPRESS, 2018)

Durante voos experimentais foi constatado que o alcance era muito menor do que o máximo garantido pela fabricante, na ordem de metros. Verificou-se então que o tamanho de pacote excedia as condições optimais de envio do dispositivo de 115kbps. O tamanho do pacote era cerca de 440 Bytes, com frequência de envio de 10 Hz. Nota-se que a disparidade do valor se deve ao não contar com os pacotes de *handshake*, verificação de erro e redundância do protocolo zigbee. Portanto foi implementado um protocolo de compactação de dados em alto nível, cujo estudo está apresentado na Figura 62.

Figura 62 – Estudo de protocolo de compactação de dados

Estudo de quantidade de bytes necessários para header					
Dado	RTC	RPM	WOW	HP	VCAS
Valor Max.	864.000	20.000	1	30.000	3.000
Byte					

Dado	XGPS	YGPS	ZGPS
Valor Max.	180.000.000	360.000.000	7.000
Byte			

Dado	MAGHEAD	ELEV	AIL	RUD	P
Valor Max.	36.000	1.800	1.800	1.800	40.000
Byte					

Dado	Q	NZ	THETA	PHI	TEMP
Valor Max.	40.000	600	36.000	36.000	100
Byte					

1 byte	256
2 bytes	65.536
3 bytes	16.777.216
4 bytes	4.294.967.296

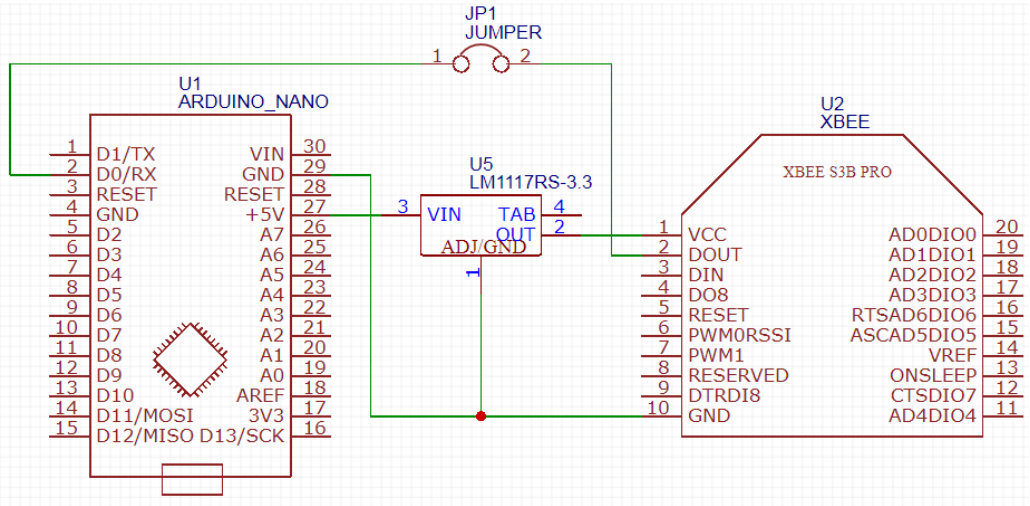
Fonte: Autor

A partir disso se teve uma redução para 10% do tamanho do pacote inicial. Basicamente os dados que antes eram enviados como strings de caracteres passaram a ser enviados como bytes. Inclui-se um cabeçalho para determinar o início de cada pacote, para correta interpretação e logo descompactação no lado de recepção. Uma vez implementado, o sinal passou a ser recebido a até 3km de distância, alcançando a meta estipulada.

O módulo terrestre de recepção consiste em um xbee acoplado a um microcontrolador, que realiza a descompactação dos pacotes recebidos e transmite os valores lidos através de serial. Desse modo, um computador qualquer pode acompanhar a aquisição de dados em tempo real. A representação dos dados no caso é uma réplica daquele registrado no cartão SD, isto é, valores separados por tab, formando colunas, e separados por linha para cada amostra nova.

A Figura 63 apresenta o esquemático do módulo terrestre.

Figura 63 – Esquemático do módulo de recepção



Fonte: Autor

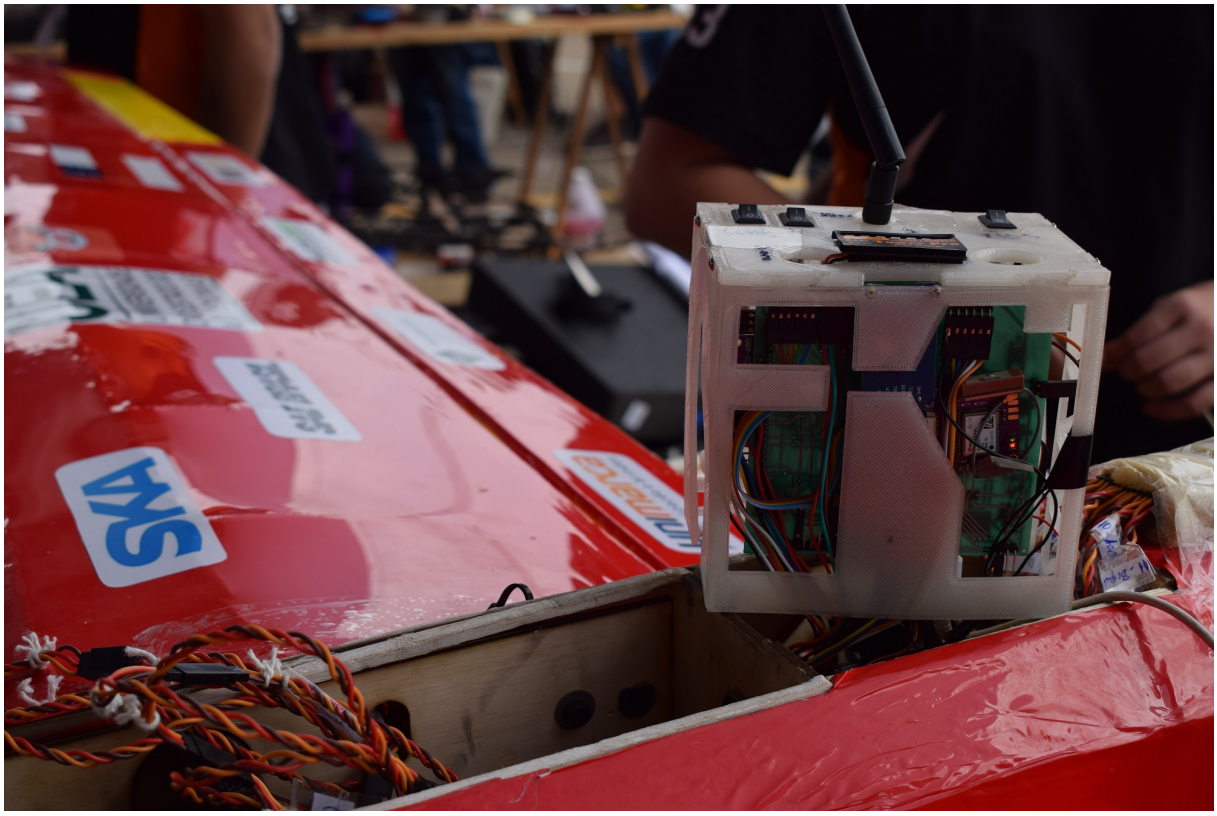
3.5 INTEGRAÇÃO DE PROJETO

Conforme descrito no capítulo anterior, o sistema embarcado foi integrado a um VANT desenvolvido pelo projeto de extensão Urutau Aerodesign.

O hardware desenvolvido foi embarcado na aeronave. O posicionamento de sensores particulares foram descritos nas seções anteriores. O restante dos dispositivos foram concentrados e armazenados dentro de um compartimento apropriado, localizado na seção anterior da fuselagem. A estrutura do compartimento foi feito em termoplástico PLA (impressão 3D), constituindo elemento integral do projeto da aeronave, representado na Figura 64.

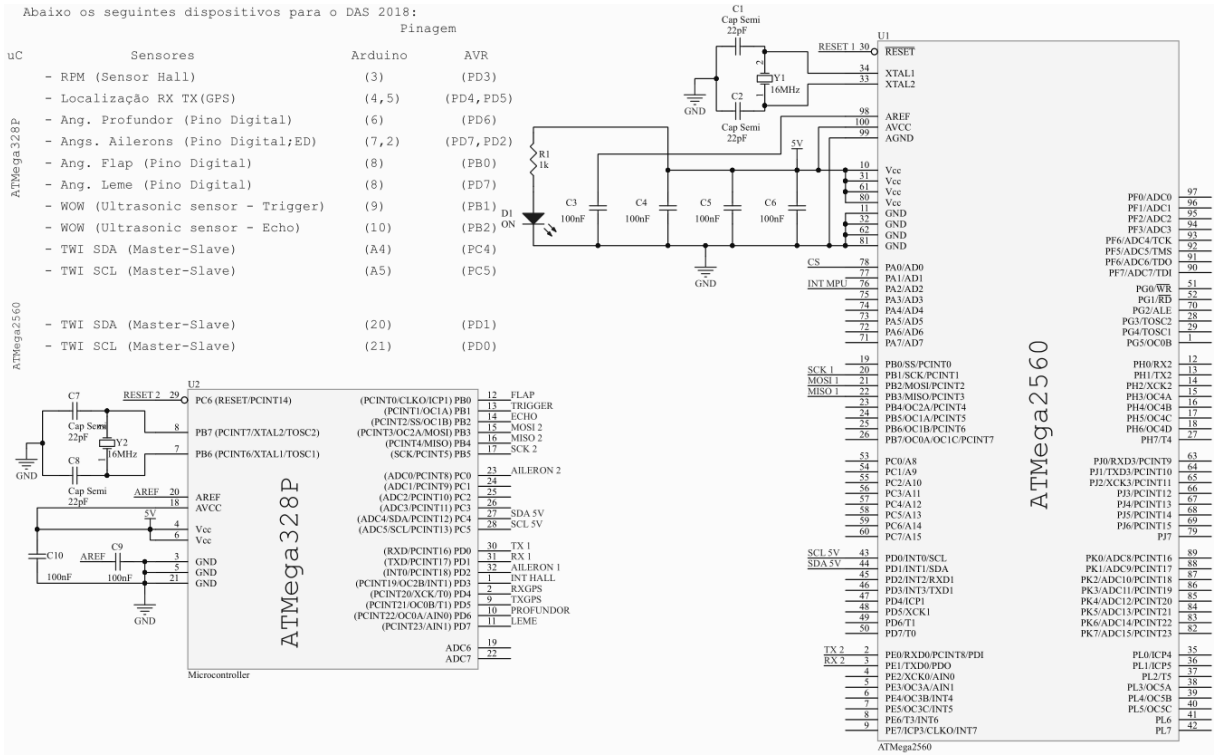
Para circuito primário do DAS foi encomendado uma placa impressa de um patrocinador do projeto de extensão. O restante das placas, no entanto, foi feito em placas perfuradas. A Figura 65 apresenta o esquemático do circuito primário do sistema de aquisição de dados.

Figura 64 – Compartimento do sistema de aquisição de dados



Fonte: Autor

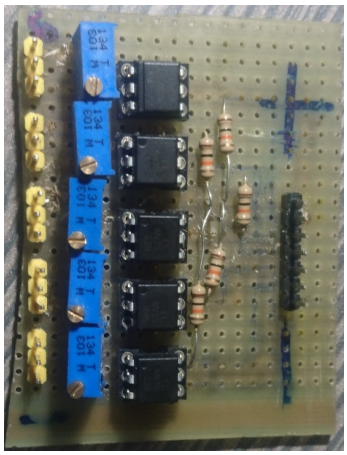
Figura 65 – Esquemático do circuito primário do DAS



Fonte: Autor

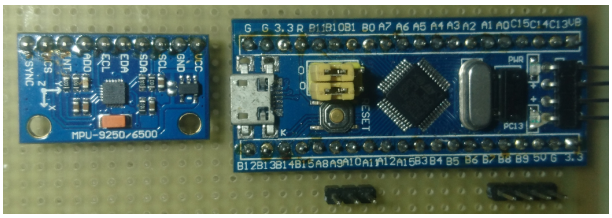
As Figuras 66, 67 e 68 apresentam as placas desenvolvidas para os circuitos auxiliares, respectivamente o módulo de leitura PWM, o módulo IMU e o módulo de recepção.

Figura 66 – Placa do módulo de leitura PWM



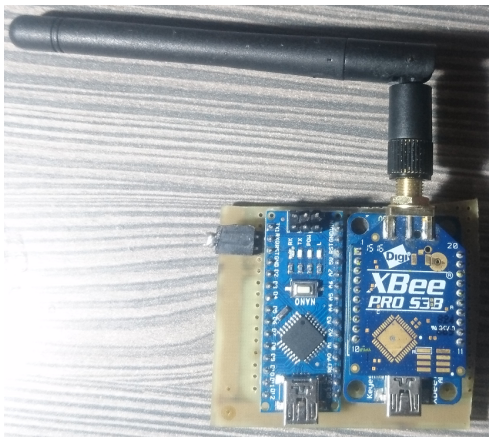
Fonte: Autor

Figura 67 – Placa do módulo de leitura PWM



Fonte: Autor

Figura 68 – Placa do módulo de leitura PWM



Fonte: Autor

4 ANÁLISE E INTERPRETAÇÃO DE RESULTADOS

Conforme descrito nos itens anteriores, foram realizados voos experimentais de validação do sistema. Para efeito de apresentação foram selecionados dois voos: um realizado em Manaus na cidade universitária e outro em São José dos Campos no campus do ITA, durante a Competição SAE Aerodesign, da qual a equipe Urutau Aerodesign participa.

As Figuras 69 e 70, nas páginas a seguir, resumem graficamente os dados gravados durante os voos em Manaus e em São José dos Campos respectivamente. Os dados selecionados para apresentação são RPM, WOW, HP, VCAS e ELEV.

Para conveniência de leitura, as definições são reiteradas a seguir:

- **RPM**: Revoluções por minuto do motor
- **WOW**: *Weight on Wheels*, binário que indica se a aeronave se encontra em solo ou em voo
- **HP**: Altitude barométrica
- **VCAS**: Velocidade aerodinâmica calibrada
- **ELEV**: Ângulo de deflexão do profundor
- **THETA, PHI**: Inclinação de arfagem e de rolamento, respectivamente

Conforme evidenciado pelos gráficos, o sistema embarcado foi capaz de adquirir os dados ao qual o trabalho se dispôs a obter. Além disso, durante a realização dos voos mencionados, o módulo de telemetria foi capaz de transmiti-los em tempo real para monitoramento em solo durante todo o percurso de voo.

Observa-se nas Figuras 69 e 70 que é possível acompanhar as etapas de voo através da interpretação dos gráficos. A consistência dos dados merece destaque, uma vez que o sistema embarcado foi eficaz em reproduzir resultados similares em múltiplos voos.

É importante enfatizar ainda que a telemetria do sistema foi utilizado pela equipe Urutau durante a fase de testes de projeto para avaliação do desempenho da aeronave projetada. Em particular foi usado para a aquisição de velocidade e de RPM, para seleção de hélice para o motor; assim como para obter o comprimento de pista necessário para decolagem. Ressalta-se que estes dois últimos parâmetros são particularmente importantes para a competição SAE Aerodesign, pois são limitados pelo regulamento. Desse modo, cumpriu seu objetivo em auxiliar na otimização de um projeto aeronáutico.

Quanto à calibração dos sensores, foi realizado primariamente por meio de programação, em que se atribui valores a variáveis e se escreve nos registradores dos dispositivos I2C. Infelizmente não havia acesso a ferramentas e equipamentos de laboratório que permitissem a validação empírica da margem de erro dos dados lidos. Desse modo se recorre à credibilidade dos dados fornecidos pelos fabricantes dos dispositivos utilizados.

Vale salientar o sucesso alcançado com o módulo GPS, cujos dados obtidos estão dispostos nas Figuras 72, 73, 74 e 75. Essas duas primeiras apresentam a trajetória de voo desenvolvida em função das coordenadas de latitude e longitude, sobrepostas a fotografias de satélite; enquanto que estas duas últimas apresentam a trajetória levando em conta também a altitude.

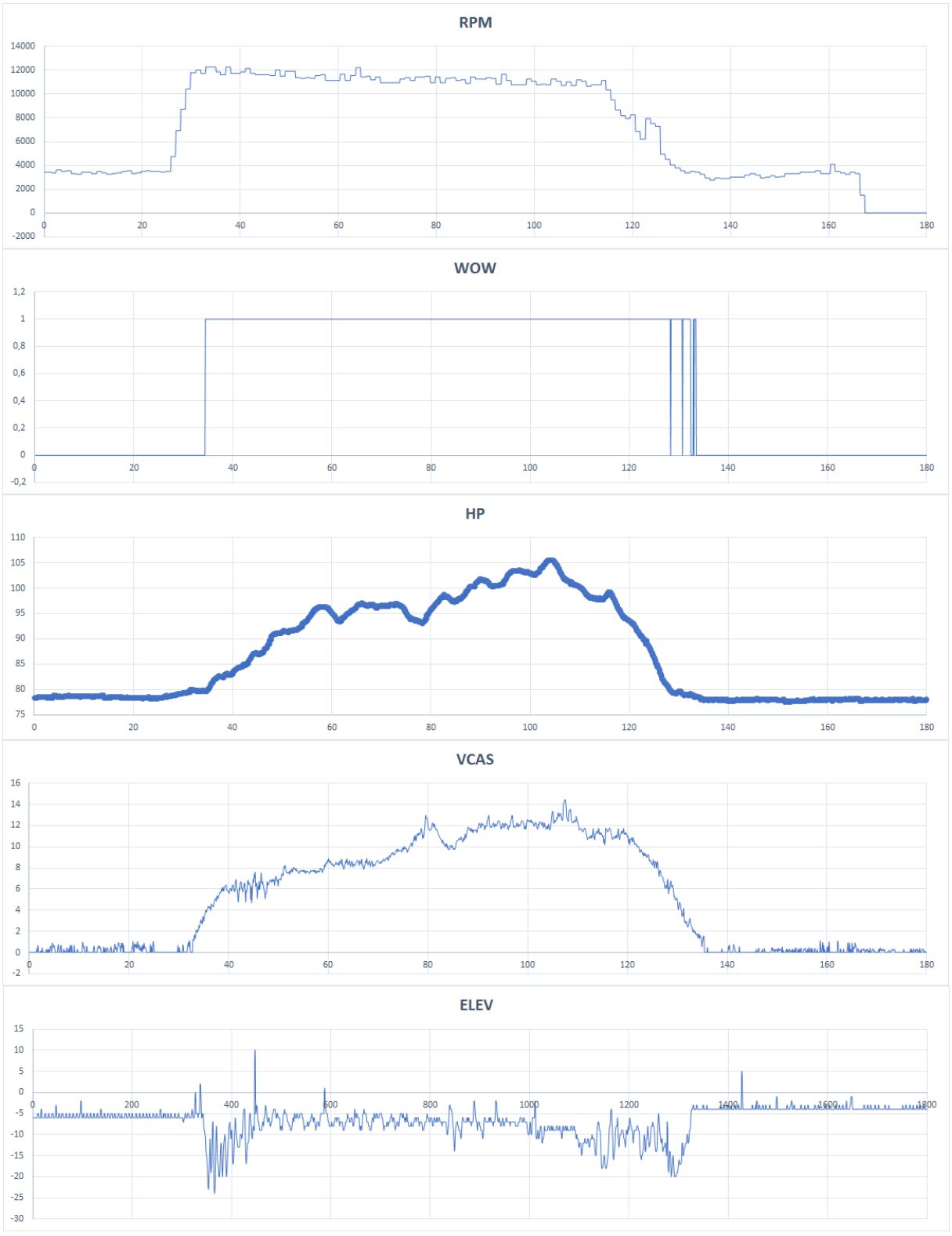
Ressalta-se certas diferenças entre os sistemas utilizados para cada voo, visto que o primeiro se tratava da versão protótipo. Conforme é visível no gráfico de Altitude (HP), apresentou precisão menor que no segundo voo. Isso se deve pelo fato de que o sensor foi posteriormente configurado para uma taxa de amostragem e coeficiente de filtro maiores.

Outra diferença entre os sistemas é a posição da antena de GPS, cujos resultados serão apresentados mais adiante neste capítulo. No primeiro voo verifica-se que em certos trechos se perdeu sinal, de modo a faltar pontos. Mediante essas falhas, foi reposicionado a antena de GPS mais externamente. Como se pode ver no próximo registro, as coordenadas foram integralmente registradas.

Verifica-se no segundo gráfico de WOW um período maior de oscilação entre estado. Isso se deve pelo fato de o asfalto da pista em que percorreu ser muito mais rugoso, visto que é voltado para aeronaves de grande porte.

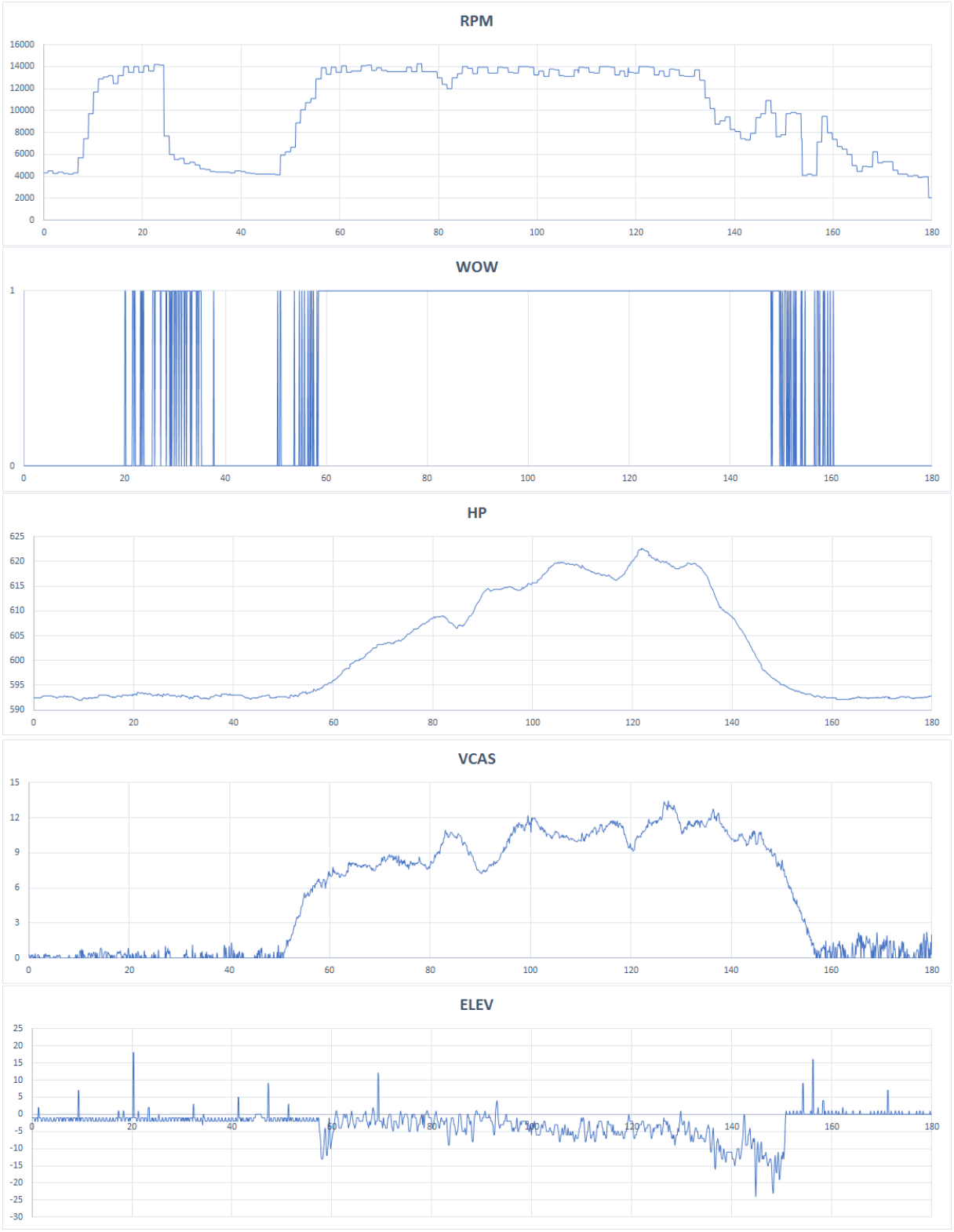
Os dados coletados pelo IMU durante voo experimental em Manaus estão dispostos na Figura 71. Observa-se que apresentou significativo ruído. Este fato se deve devido à influência exacerbada da vibração do motor à combustão. Caso a tecnologia de propulsão fosse diferente, como por exemplo motores a combustão, ou houvesse uma estruturar anti-vibratória na fixação do sensor, os dados possuiriam qualidade superior. Apesar disso, verifica-se que a linha de tendência, em laranja, para Theta e Phi é plausível, sendo condizente com a etapa de voo em que se encontra.

Figura 69 – Resultados experimentais do voo realizado em Manaus



Fonte: Autor

Figura 70 – Resultados experimentais do voo realizado em São José dos Campos

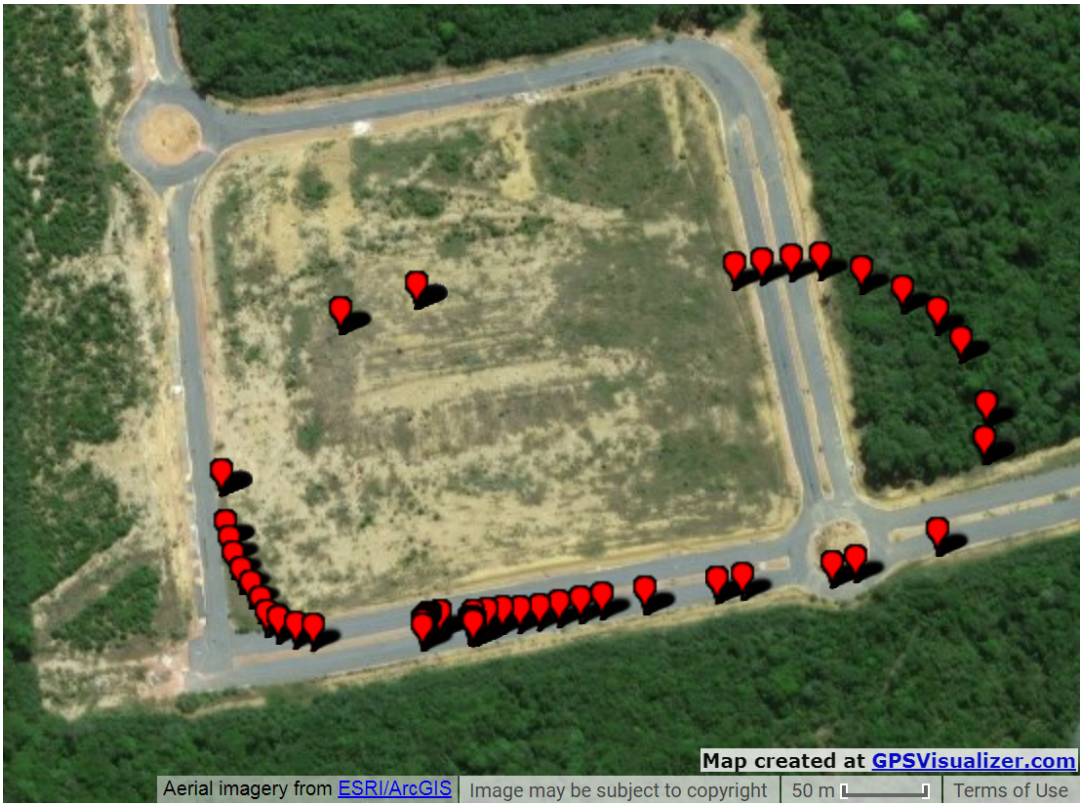


Fonte: Autor

Figura 71 – Análise de dados obtidos pelo IMU

Fonte: Autor

Figura 72 – Trajetória de voo em Manaus

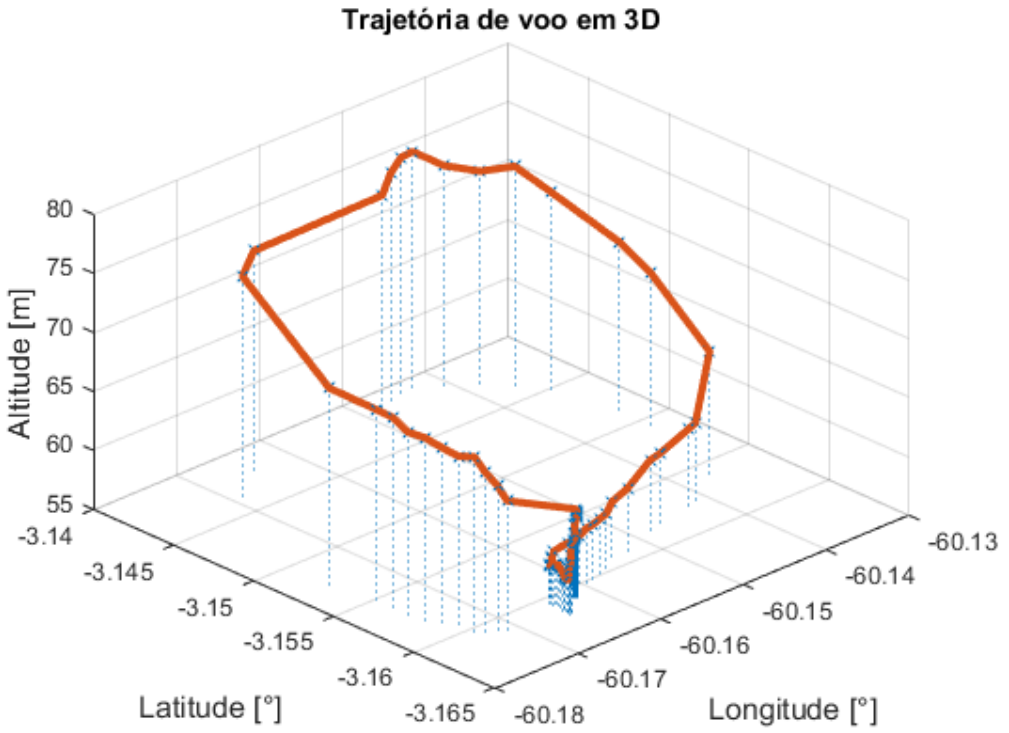


Fonte: Autor

Figura 73 – Trajetória de voo em São José dos Campos

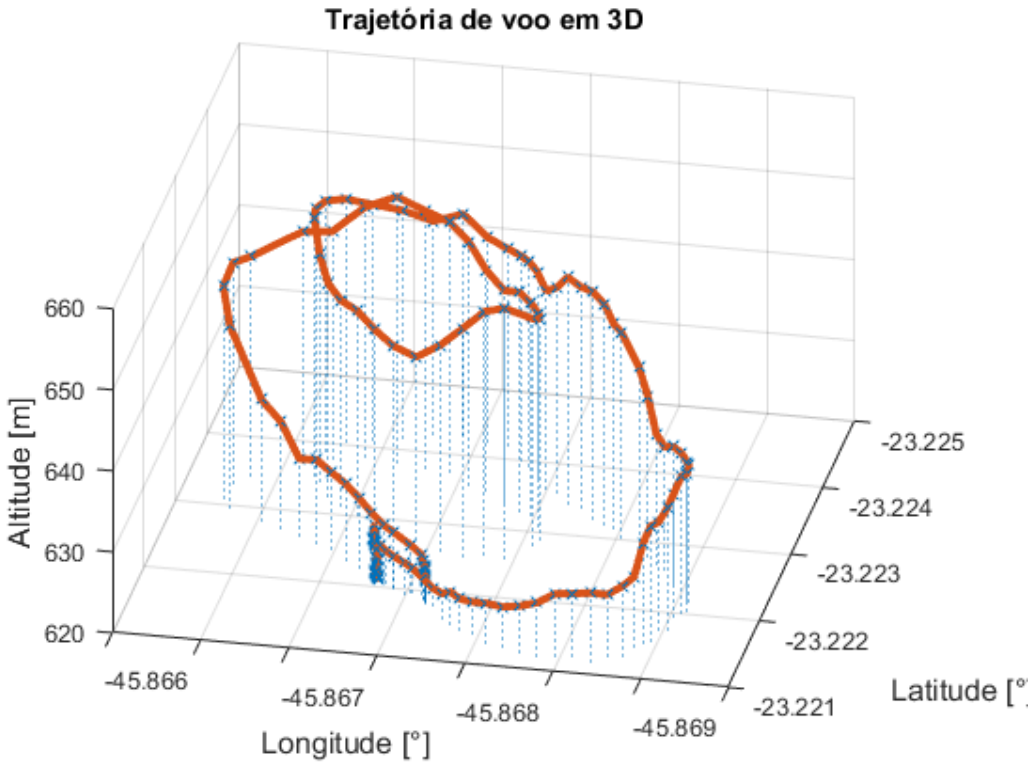
Fonte: Autor

Figura 74 – Trajetória de voo em Manaus em 3D



Fonte: Autor

Figura 75 – Trajetória de voo em São José dos Campos em 3D



Fonte: Autor

CONCLUSÃO

Este trabalho possuía como objetivo o desenvolvimento de um sistema embarcado de aquisição de dados e de telemetria para veículos aéreos não tripulados de asa fixa.

Apresentou-se brevemente uma revisão dos conceitos mais pertinentes para a compreensão do trabalho, em particular acerca todos os princípios de funcionamento dos sensores que foram empregados no sistema, assim como sobre veículos aéreos não-tripulados.

Uma vez que o projeto aeronáutico de uma aeronave de pequeno porte apresenta dificuldades devido ao know-how da indústria aeronáutica ser concentrado em teoria aplicáveis somente a aeronaves de maior porte, existe a necessidade de validação empírica. Assim sendo, o sistema projetado, que através da coleta de parâmetros estratégicos de aeronaves de pequeno porte, assim como pelo monitoramento em tempo real, agrega valor ao desenvolvimento de um projeto aeronáutico.

O sistema embarcado proposto por este trabalho foi integrado a um VANT, apresentando-se eficaz na aquisição de dados durante voos experimentais, assim como na transmissão em tempo real para solo dos dados adquiridos. De fato foi utilizado para avaliação de desempenho do projeto aeronáutico do VANT em questão.

Para futuros trabalhos, sugere-se a implementar tempo de amostragem maior, de modo a melhorar a aquisição de parâmetros derivados de IMU. Para tanto sugere-se embarcar um microcontrolador mais robusto, porém menos trivial de se programar, capaz de processamento maior, permitindo assim também a simplificação da arquitetura do sistema. Além disso, sugere-se o desenvolvimento de software para monitoramento.

REFERÊNCIAS

ALIEXPRESS. *AliExpress: Online shopping for the latest electronics, fashion, phone accessories, computer electronics, toys and more.* 2018. Disponível em: <https://www.aliexpress.com>.

ARCGIS. *Mapping and analysis: location intelligence for everyone*. 2018. Disponível em: <https://www.esri.com/en-us/arcgis/products/arcgis-online/overview>.

ATMEL. *ATmega328/P Datasheet*. [S.l.], 2016.

BAKSHI, U. A.; BAKSHI, V. U. *Electrical Circuits and Machines*. [S.l.]: Technical Publications Pune, 2009.

BECKERATH, A. v. et al. *IKA Handbook: Pressure and Temperature Measurement U.S. Edition*. [S.l.]: WIKA Instrument Corporation, 2008.

BOSCH. *BME680 Low power gas, pressure, temperature humidity sensor*. [S.l.], 2017.

BRONZATTI, L. F. C. *Análise sobre a tecnologia de rede sem fio Zigbee / IEEE 802.15.4*. 2013. Trabalho de Conclusão de Curso, USP (Universidade de São Paulo).

CARNEIRO, J. B. de S.; LUGLI, A. B. Estudo de sensores ultrassônicos e suas aplicações. *I SRST – SEMINÁRIO DE REDES E SISTEMAS DE TELECOMUNICAÇÕES*, 2014.

DANA, P. H. *Global Positioning System Overview*. 2000. Disponível em: <http://foote.geography.uconn.edu/gcraft/notes/gps/gps_f.html>.

EMBEDDED. *IMPLEMENTING your MCU-based system's serial UART in software*. 2018. Disponível em: <http://www.embedded.com/design/other/4025995/Implementing-your-MCU-based-system-s-serial-UART-in-software>.

FAA. *Pilot's Handbook of Aeronautical Knowledge*. [S.l.]: U.S. Department of Transportation, 2016.

FLOREANO, D.; WOOD, R. J. Science, technology and the future of small autonomous drones. *Nature volume 521, pages 460–466 (28 May 2015)*, 2015.

FRACAROLLI, J. P. V. *Implementação do controle remoto de uma miniatura de carro operado via computador utilizando comunicação wireless*. 2012. Trabalho de Conclusão de Curso, USP (Universidade de São Paulo).

FRADEN, J. *Handbook of Modern Sensors: Physics, Designs, and Applications*. [S.l.]: Springer, 2004.

HONEYWELL. *Hall Effect Sensing and Application*. [S.l.]: Honeywell, 2018.

ISRAELIWEAPONS. *IMPLEMENTING your MCU-based system's serial UART in software*. 2001. Disponível em: <http://www.israeli-weapons.com/weapons/aircraft/uav/scout/Scout.html>.

LAJúS, C. R. et al. Prospecção de patentes relacionadas ao uso de vants como inserção tecnológica na agricultura de precisão. *X Congresso Brasileiro de Agroinformática*, 2015.

LAN, Y. *Robot, drone patrol high-voltage substation in Chongqing*. 2017. Disponível em: <http://www.ecns.cn/visual/hd/2017/11-24/147392.shtml>.

LIMA, C. A. de A.; MENDES, L. A. M. Uma análise sistemática da rede sem fio zigbee: proposta de uso na plantação de café. *Faculdade Regional de Ciências Exatas e Sociais de Barbacena*, 2008.

LOPES, R. R.; AMARAL, R. F. *PROJETO CONCEITUAL DE AERONAVE NÃO-TRIPULADA PARA VIGILÂNCIA DE RESERVAS FLORESTAIS*. 2004. Trabalho de Conclusão de Curso, ITA (INSTITUTO TECNOLÓGICO DE AERONÁUTICA).

MADGWICK, S. O. *An efficient orientation filter for inertial and inertial/magnetic sensor arrays*. 2010. Relatório interno de progresso de doutorado, Universidade de Bristol.

MAXIM. *DS3231 Extremely Accurate I2C-Integrated RTC/TCXO/Crystal*. [S.l.], 2015.

MIKROE. *UART - Serial communication*. 2016. Disponível em: <https://www.mikroe.com/blog/uart-serial-communication>.

MONICO, J. F. G. *Posicionamento pelo NAVSTAR-GPS: descrição, fundamentos e aplicações*. [S.l.]: Editora UNESP, 2000.

MOURA, R. S. *Desenvolvimento de um sistema de orientação espacial incercial*. 2013. Trabalho de Conclusão de Curso, USP (Universidade de São Paulo).

MOUSER. *Ultrasonic Ranging Module HC - SR04*. 2016. Disponível em: <https://www.mouser.com/ds/2/813/HCSR04-1022824.pdf>.

NASA. *Pitot Static Tube*. 2015. Disponível em: <https://www.grc.nasa.gov/www/k-12/airplane/pitot.html>.

NAVYFLIGHTMANUALS. *Weight-on-wheels (WOW) and gear-down/up-and-locked contro*. 2018. Disponível em: <http://navyflightmanuals.tpub.com/1278BK1_wch7/Figure-25-Weight-On-Wheels-Wow-321.htm>.

NDK. *What is an ultrasonic sensor?* 2018. Disponível em: <http://www.ndk.com/en/sensor/ultrasonic/index.html>.

NISOLOSI, D. E. C. *Microcontrolador 8051: detalhado*. [S.l.]: Editora Érica, 2009.

NIXON, A. *Best Drones For Agriculture 2018: The Ultimate Buyer's Guide*. 2018. Disponível em: <https://bestdroneforthejob.com/drone-buying-guides/agriculture-drone-buyers-guide/>.

PELAYO, R. *What is I2C? | Protocol Guide*. 2018. Disponível em: <https://www.teachmemicro.com/i2c-primer/>.

REILABS. *How to update NEO M8N firmware*. 2016. Disponível em: <http://www.rei-labs.net/how-to-update-neo-m8n-firmware/>.

RODRIGUES, L. E. M. J. *Fundamentos da Engenharia Aeronáutica com Aplicações ao Projeto SAE*. [S.l.]: Salto/SP, 2014.

SAVOX. *SA1230SG - Coreless Digital Servo 0.16/500 @6V*. 2018. Disponível em: <https://www.savoxusa.com/collections/best-sellers/products/savsa1230sg-coreless-digital-servo>.

SICILIANO, B.; KHATIB, O. *Springer Handbook of Robotics*. [S.l.]: Springer, 2008.

SPARKFUN. *LSM9DS0 Hookup Guide*. 2018. Disponível em: <https://learn.sparkfun.com/tutorials/lsm9ds0-hookup-guide/all/>.

SPARKFUN. *Serial Peripheral Interface (SPI)*. 2018. Disponível em: <https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all/>.

TI. *DRV5013 Digital-Latch Hall Effect Sensor Datasheet*. [S.l.], 2018.

TOMAR, A. *Introduction to Zibgbee Technology*. [S.l.]: element 1, 2011.

URUTAU. *Relatório de Projeto Urutau Aerodesign 2018*. [S.l.], 2018.

VAHID, F.; GIVARGIS, T. *Embedded System Design: A Unified Hardware/Software Introduction*. [S.l.]: John Wiley and Sons, 2001.

ANEXO A: FIRMWARE DE MICROCONTROLADOR PRINCIPAL

MAIN

```
1  //DAS 18 master
2  #define dasVersion "v2.1"
3  #define versionName "versão final"
4  //#define DEBUG
5  //03/11/18
6  //Jake - Jacobus Laurens de Jager
7
8  #pragma GCC optimize ("-Ofast")
9  #include <Wire.h>
10
11 #define baudrate 115200
12
13 //global variables
14 String data_string;
15 byte data_string_serial[40];
16 int rps;
17 uint32_t latuint, longuint;
18 int altint;
19 long magheadint ;
20 long gyroXrateint;
21 long gyroYrateint;
22 long gForceZint;
23 long kalAngleYint;
24 long kalAngleXint;
25 int wow, rpm, servo1, servo2, servo3;
26 float hp, pbmp, vcas, Pvcas;
27 float longfloat, latfloat, altfloat;
28 float maghead;
29 float gForceX, gForceY, gForceZ;
30 float gyroXrate, gyroYrate, gyroZrate;
31 float kalAngleX, kalAngleY;
32 uint16_t servo1uint, servo2uint, servo3uint;
33 double tempo;
34 int teste1, teste2;
35 double temp;
36
37 //rtc
38 #include <DS3231.h>
39 DS3231  rtc(SDA, SCL);
40 Time  t;
41 unsigned long  h, m, s;
42 uint8_t cs;
43 unsigned long  saux, aux, tempaux;
44 bool primeiro_ciclo = true;
45
46 void temporizador_rtc()
47 {
48   //basicamente se aguarda 1 seg, para o contador de centisegundos sincronizar
49   if ( primeiro_ciclo == true )
50   {
51     primeiro_ciclo = false;
52     tempaux = millis() + 1000;
53     while (aux < tempaux)
54     {
55       t = rtc.getTime();
```

```
56        s = t.sec;
57        if (s == saux)
58        {
59          if (millis() - aux >= 100 && cs < 9)
60          {
61            cs = cs + 1;
62            aux = millis();
63          }
64        } else
65        {
66          cs = 0;
67          aux = millis();
68        }
69        saux = t.sec;
70      }
71    }
72  }
73
74  void setup() {
75    Serial.begin(baudrate);
76    Wire.begin();
77
78    rtc.begin();          //relogio
79    vcas_setup();         //velocidade
80    hp_setup();           //altitude barometrica
81    telemetria_setup();
82    temporizador_rtc();  //roda durante primeiro ciclo para sincronizar casa decimal
83  }
84
85  void loop() {
86    t = rtc.getTime();
87    s = t.sec;
88
89    if (s == saux)
90    {
91      if (millis() - aux >= 100 && cs < 9)
92      {
93        cs++;
94        aux = millis();
95        get_data();
96        serial_write();
97        SD_write();
98      }
99    } else
100   {
101     cs = 0;
102     aux = millis();
103     get_data();
104     serial_write();
105     SD_write();
106   }
107
108   saux = t.sec;
109 }
```

HP

```
//HP
#include "Zanshin_BME680.h"

#define SEALEVELPRESSURE_HPA (1013.25)
BME680_Class BME680;

float calcAltitude(float pressureHP, const float seaLevel = 1013.25)
{
  float Altitude = 44330.0 * (1.0 - pow(pressureHP / seaLevel, 0.1903));
  return (Altitude);
}

void hp_setup()
{
  BME680.begin(I2C_STANDARD_MODE);

  BME680.setOversampling(TemperatureSensor, Oversample2);
  BME680.setOversampling(HumiditySensor,   SensorOff);
  BME680.setOversampling(PressureSensor,   Oversample2);
  BME680.setIIRFilter(IIR8);
  BME680.setGas(0, 0);
}

void hp_loop()
{
  static int32_t temperature, humidity, pressure, gas;
  BME680.getSensorData(temperature, humidity, pressure, gas);

  int temp_x100 = temperature;
  temp = temp_x100 / 100.0;
  pbmp = pressure / 100.0;

  hp = calcAltitude(pbmp, SEALEVELPRESSURE_HPA);
}
```

IMU

```
///IMU
#define imu_bytes 12
#define imu_addr 9

byte gyroXrateint0 = 0; byte gyroXrateint1 = 0;
byte gyroYrateint0 = 0; byte gyroYrateint1 = 0;
byte gForceZint0 = 0;   byte gForceZint1 = 0;
byte kalAngleYint0 = 0; byte kalAngleYint1 = 0;
byte kalAngleXint0 = 0; byte kalAngleXint1 = 0;
byte magheaduint0 = 0;  byte magheaduint1 = 0;

void imu_loop()
{
  Wire.requestFrom(imu_addr, imu_bytes);

  while (Wire.available())
  {
```

```
    magheaduint0 = Wire.read();   magheaduint1 = Wire.read();
    gyroXrateint0 = Wire.read();  gyroXrateint1 = Wire.read();
    gyroYrateint0 = Wire.read();  gyroYrateint1 = Wire.read();
    gForceZint0 = Wire.read();    gForceZint1 = Wire.read();
    kalAngleYint0 = Wire.read();  kalAngleYint1 = Wire.read();
    kalAngleXint0 = Wire.read();  kalAngleXint1 = Wire.read();
  }

  magheadint =  (magheaduint0 << 8) | magheaduint1;
  gyroXrateint =  (gyroXrateint0 << 8) | gyroXrateint1;
  gyroYrateint =  (gyroYrateint0 << 8) | gyroYrateint1;
  gForceZint =  (gForceZint0 << 8) | gForceZint1;
  kalAngleYint =  (kalAngleYint0 << 8) | kalAngleYint1;
  kalAngleXint =  (kalAngleXint0 << 8) | kalAngleXint1;

  maghead = magheadint /= 100;
  gyroXrate = gyroXrateint - 25000; gyroXrate /= 100; //(gyroXrateint / 100) - 250;
  gyroYrate = gyroYrateint - 25000; gyroYrate /= 100; //(gyroYrateint / 100) - 250;
  gForceZ = gForceZint - 20000; gForceZ /= 10000; //(gForceZint / 100) - 2;
  kalAngleY = kalAngleYint - 18000; kalAngleY /= 100; //(kalAngleYint / 100) - 180;
  kalAngleX = kalAngleXint - 18000; kalAngleX /= 100; //(kalAngleXint / 100) - 180;
}
```

I2C SLAVES

```
//I2C escravos
//vulgo "Medusa"

#define num_bytes 16
#define end_escravo 8

void medusa_loop()
{
  Wire.requestFrom(end_escravo, num_bytes);

  while (Wire.available())
  {
    byte rpsint0 = Wire.read(); byte rpsint1 = Wire.read();
    rps = rpsint0; rps = (rps << 8) | rpsint1;
    rpm = rps * 60;

    wow = Wire.read();

    servo1uint = Wire.read();    servo2uint = Wire.read();    servo3uint = Wire.read();
    servo1 = servo1uint - 90;    servo2 = servo2uint - 90;    servo3 = servo3uint - 90;

    byte latuint0 = Wire.read(); byte latuint1 = Wire.read(); byte latuint2 = Wire.read(); byte latuint3 = Wire.read();
    latuint = latuint0; latuint = (latuint << 8) | latuint1; latuint = (latuint << 8) | latuint2; latuint = (latuint << 8) | l
    int32_t latint = latuint - 2147483647;
    if (latint == -2147483647)
    {
      latint = 0;
    }
    latfloat = latint; latfloat /= 1000000;

    byte longuint0 = Wire.read(); byte longuint1 = Wire.read(); byte longuint2 = Wire.read(); byte longuint3 = Wire.read();
    longuint = longuint0; longuint = (longuint << 8) | longuint1; longuint = (longuint << 8) | longuint2; longuint = (longuint
```

```
    int32_t longint = longuint - 2147483647;
    if (longint == -2147483647)
    {
      longint = 0;
    }
    longfloat = longint; longfloat /= 1000000;

    byte altint0 = Wire.read(); byte altint1 = Wire.read();
    altint = altint0; altint = (altint << 8) | altint1;
    altfloat = altint; altfloat /= 100;
  }
}
```

VCAS

```
//Vcas
#include "SDP6x.h"

float difPressure;

void vcas_setup()
{
  difPressure = 0.0;
}

void vcas_loop()
{
  Pvcas = SDP6x.GetPressureDiff();
  vcas = 0;
  if (Pvcas > 0)
  {
    vcas = sqrt ((2.0 * Pvcas) / 1.225);
  }
}
```

TELEMETRY

```
///telemetria
// aquisisção, transmissao e gravação de dados

//SD
#include  <SPI.h>
#include "SdFat.h"
SdFat SD;
const int chipSelect = 22;
File cartao_SD;
bool SD_flag = 0;
//SD

void telemetria_setup()
{
  String cabecalho1 = F("Tempo \t RPM \t WOW \t HP \t VCAS \t XGPS \t YGPS \t ZGPS \t MagHead \t ELEV \t AIL  \t RUD  \t
  String cabecalho2 = F("seg \t rpm \t bit \t f \t m/s \t deg \t deg \t m  \t deg \t deg \t deg  \t deg  \t deg/s \t deg/

```

```
18    Serial.println(F("##########################################################################################################################"));
19
20    print_logo();
21
22    Serial.println(F("##########################################################################################################################"));
23    Serial.println(F("DAS 18 Urutau Aerodesign"));
24    Serial.print(dasVersion); Serial.print(" - "); Serial.println(versionName);
25
26    printTime();
27
28    Serial.println(F("##########################################################################################################################"));
29    Serial.println(F("Inicializando sistema..."));
30
31    //SD
32    if (!SD.begin(chipSelect))
33    {
34      Serial.println(F("Status SD:      ERRO, nao encontrado! Verifique porta SD"));
35      //return;
36    } else
37    {
38      SD_flag = 1;
39      if (SD.exists("urutau.txt"))
40      {
41        Serial.println(F("Status SD:      OK! urutau.txt existe em cartao SD"));
42      } else
43      {
44        Serial.println(F("Status SD:      OK! urutau.txt NAO existe em cartao SD, criando arquivo"));
45        cartao_SD = SD.open("urutau.txt", FILE_WRITE);
46        cartao_SD.flush();
47        cartao_SD.println(cabecalho1);
48        cartao_SD.println(cabecalho2);
49        cartao_SD.close();
50      }
51    }
52    //SD
53
54    status_check();
55
56    Serial.println(F("##########################################################################################################################"));
57
58    Serial.println(cabecalho1);
59    Serial.println(cabecalho2);
60  }
61
62  void get_data()
63  {
64    medusa_loop();
65    imu_loop();
66    vcas_loop();
67    hp_loop();
68    h = t.hour, m = t.min, s = t.sec;
69    tempo = h * 3600 + m * 60 + s;
70
71    data_string = String(tempo, 0) + String(F(".")) + String(cs) + String(F(" \t "))
72                  + String(rpm) + String(F(" \t "))
73                  + String(wow) + String(F(" \t "))
74                  + String(hp, 2) + String(F(" \t "))
75                  + String(vcas) + String(F(" \t "))
76                  + String(latfloat, 6) + String(F(" \t "))
77                  + String(longfloat, 6) + String(F(" \t "))
```

```
78                  + String(altfloat, 2) + String(F(" \t "))
79                  + String(maghead, 2) + String(F(" \t "))
80                  + String(servo1) + String(F(" \t "))
81                  + String(servo2) + String(F(" \t "))
82                  + String(servo3) + String(F(" \t "))
83                  + String(gyroXrate,2) + String(F(" \t "))
84                  + String(gyroYrate,2) + String(F(" \t "))
85                  + String(gForceZ,3) + String(F(" \t "))
86                  + String(kalAngleY,2) + String(F(" \t "))
87                  + String(kalAngleX,2) + String(F(" \t "))
88                  + String(temp) + String(F(" \t "))
89                  + String(Pvcas) + String(F(" \t "))
90                  + String(pbmp, 2);
91  }
92
93  void serial_write()
94  {
95
96  #ifdef DEBUG
97    Serial.println(data_string);
98  #else
99
100   unsigned long tempo_serial = tempo * 10 + cs;
101   uint16_t hp_serial = hp * 10;
102   uint16_t vcas_serial = vcas * 100;
103
104   data_string_serial[0] = 255;
105   data_string_serial[1] = 255;
106   data_string_serial[2] = 255;
107   data_string_serial[3] = 255;
108   data_string_serial[4] = tempo_serial >>  16;
109   data_string_serial[5] = tempo_serial >>  8;
110   data_string_serial[6] = tempo_serial;
111   data_string_serial[7] = rps >>  8;
112   data_string_serial[8] = rps;
113   data_string_serial[9] = wow;
114   data_string_serial[10]  = hp_serial >>  8;
115   data_string_serial[11]  = hp_serial;
116   data_string_serial[12]  = vcas_serial >>  8;
117   data_string_serial[13]  = vcas_serial;
118   data_string_serial[14]  = latuint >>  24;
119   data_string_serial[15]  = latuint >>  16;
120   data_string_serial[16]  = latuint >>  8;
121   data_string_serial[17]  = latuint;
122   data_string_serial[18]  = longuint  >>  24;
123   data_string_serial[19]  = longuint  >>  16;
124   data_string_serial[20]  = longuint  >>  8;
125   data_string_serial[21]  = longuint;
126   data_string_serial[22]  = altint  >>  8;
127   data_string_serial[23]  = altint;
128   data_string_serial[24]  = magheadint >>  8;
129   data_string_serial[25]  = magheadint;
130   data_string_serial[26]  = servo1uint  >>  8;
131   data_string_serial[27]  = servo1uint;
132   data_string_serial[28]  = servo2uint  >>  8;
133   data_string_serial[29]  = servo2uint;
134   data_string_serial[30]  = servo3uint  >>  8;
135   data_string_serial[31]  = servo3uint;
136   data_string_serial[32]  = gyroXrateint >>  8;
137   data_string_serial[33]  = gyroXrateint;
```

```
138     data_string_serial[34]  = gyroYrateint >>  8;
139     data_string_serial[35]  = gyroYrateint;
140     data_string_serial[36]  = gForceZint >>  8;
141     data_string_serial[37]  = gForceZint;
142     data_string_serial[38]  = kalAngleYint >>  8;
143     data_string_serial[39]  = kalAngleYint;
144     data_string_serial[40]  = kalAngleXint >>  8;
145     data_string_serial[41]  = kalAngleXint;
146     data_string_serial[42]  = temp;
147
148     for (uint8_t i = 0; i < 42 ; i++)
149     {
150       Serial.write(data_string_serial[i]);
151     }
152
153   #endif
154   }
155
156   void SD_write() {
157     if (SD_flag == 1)
158     {
159       cartao_SD = SD.open("urutau.txt", FILE_WRITE);
160       cartao_SD.print(data_string);
161       cartao_SD.print("\n");
162       cartao_SD.close();
163     }
164   }
165
166   void status_check() {
167     Wire.beginTransmission (0x8);
168     if (Wire.endTransmission () == 0)
169     {
170       Serial.println (F("Status medusa:  OK!"));
171     } else
172     {
173       Serial.println (F("Status medusa:  ERRO, nao encontrado! Placa danificada?!"));
174     }
175     Wire.beginTransmission (0x68);
176     if (Wire.endTransmission () == 0)
177     {
178       Serial.println (F("Status RTC:     OK!"));
179     } else
180     {
181       Serial.println (F("Status RTC:     ERRO, nao encontrado! Placa danificada?! FALHA CRITICA"));
182     }
183     Wire.beginTransmission (0x77);
184     if (Wire.endTransmission () == 0)
185     {
186       Serial.println (F("Status HP:      OK!"));
187     } else
188     {
189       Serial.println (F("Status HP:      ERRO, nao encontrado!"));
190     }
191     Wire.beginTransmission (0x40);
192     if (Wire.endTransmission () == 0)
193     {
194       Serial.println (F("Status Vcas:    OK!"));
195     } else
196     {
197       Serial.println (F("Status Vcas:    ERRO, nao encontrado!"));
```

```
198     }
199     Wire.beginTransmission (0x09);
200     if (Wire.endTransmission () == 0)
201     {
202       Serial.println (F("Status IMU:     OK!"));
203     } else
204     {
205       Serial.println (F("Status IMU:     ERRO, nao encontrado!"));
206     }
207   }
208
209   void printTime()
210   {
211     t = rtc.getTime();
212     uint8_t h = t.hour, m = t.min;
213     Serial.print(t.date, DEC); Serial.print(F("/")); Serial.print(t.mon); Serial.print(F("/")); Serial.print(t.year, DEC);
214     Serial.print("     "); if (h < 9) {
215       Serial.print("0");
216     }  Serial.print(h); Serial.print(F(":")); if (m < 9) {
217       Serial.print("0");
218     } Serial.println(m);
219   }
220
221
222   void print_logo()
223   {
224     String logo1 = F("  _   _            _                    _                      _           _             ");
225     String logo2 = F(" | | | |_ __ _   _| |_ __ _ _   _     / \\   ___ _ __ ___   __| | ___  ___(_) __ _ _ __  ");
226     String logo3 = F(" | | | | '__| | | | __/ _` | | | |   / _ \\ / _ \\ '__/ _ \\ / _` |/ _ \\/ __| |/ _` | '_ \\ ");
227     String logo4 = F(" | |_| | |  | |_| | || (_| | |_| |  / ___ \\  __/ | | (_) | (_| |  __/\\__ \\ | (_| | | | |");
228     String logo5 = F("  \\___/|_|   \\__,_|\\__\\__,_|\\__,_| /_/   \\_\\___|_|  \\___/ \\__,_|\\___||___/_|\\__, |_| |_|");
229     String logo6 = F("                                                                              |___/       ");
230
231     Serial.println(logo1);
232     Serial.println(logo2);
233     Serial.println(logo3);
234     Serial.println(logo4);
235     Serial.println(logo5);
236     Serial.println(logo6);
237   }
```

ANEXO B: FIRMWARE DE MICROCONTROLADOR AUXILIAR

MAIN

```
//DAS 18 slave v2.0
#define dasVersion "v2.0"
//23/9/18
//Jake - Jacobus Laurens de Jager

//global variables
uint8_t wow;
uint16_t rps;
byte rpsArray[2];
int8_t servo1, servo2, servo3;
byte latArray[4], longArray[4], altArray[2];
float latfloat, longfloat, altfloat;
int teste1 = 90, teste2 = 254;
double loop_temp_aux, print_temp_aux;
int pwm1_filtered = 0, pwm2_filtered = 0, pwm3_filtered = 0;

#define refresh_period 50 //milliseconds

void setup() {
  Serial.begin(115200);
  Serial.println("DAS - Medusa");

  i2c_transmission_setup();

  servo_setup();
  gps_setup();
}

void loop() {
  if (millis() - loop_temp_aux >= refresh_period)
  {
    loop_temp_aux = millis();
    rps_loop();
    wow_loop();
    gps_loop();
  }
  servo_loop();
  print_loop(); //for debugging only
}
```

GPS

```
//GPS
int32_t latint = -3.100081 * 1000000;
int32_t longint = -60.02289 * 1000000;
int altint = 26.04 * 100;
uint32_t latuint, longuint;

#include <TinyGPS++.h>
#include <SoftwareSerial.h>

static const int RXPin = 4, TXPin = 5;
```

```
11  static const uint32_t GPSBaud = 9600;
12
13  TinyGPSPlus gps;
14  SoftwareSerial ss(RXPin, TXPin);
15
16  void gps_setup()
17  {
18    ss.begin(GPSBaud);
19  }
20
21  void gps_loop()
22  {
23    smartDelay(0); //necessary for reading GPS values
24
25    latint = gps.location.lat() * 1000000;
26    longint = gps.location.lng() * 1000000;
27    altint = gps.altitude.meters() * 100;
28
29    latfloat = latint; latfloat /= 1000000;
30    longfloat = longint; longfloat /= 1000000;
31    altfloat = altint; altfloat /= 100;
32
33    latuint = latint + 2147483647;
34    latArray[0] = (latuint >> 24) & 0xFF;
35    latArray[1] = (latuint >> 16) & 0xFF;
36    latArray[2] = (latuint >> 8) & 0xFF;
37    latArray[3] = latuint & 0xFF;
38
39    longuint = longint + 2147483647;
40    longArray[0] = (longuint >> 24) & 0xFF;
41    longArray[1] = (longuint >> 16) & 0xFF;
42    longArray[2] = (longuint >> 8) & 0xFF;
43    longArray[3] = longuint & 0xFF;
44
45    altArray[0] = (altint >> 8) & 0xFF;
46    altArray[1] = altint  & 0xFF;
47  }
48
49  static void smartDelay(unsigned long ms)
50  {
51    unsigned long start = millis();
52    do
53    {
54      while (ss.available())
55        gps.encode(ss.read());
56    } while (millis() - start < ms);
57  }
58
59  static void printFloat(float val, bool valid, int len, int prec)
60  {
61    if (!valid)
62    {
63      while (len-- > 1)
64        Serial.print('*');
65      Serial.print(' ');
66    }
67    else
68    {
69      Serial.print(val, prec);
70      int vi = abs((int)val);
```

```
    int flen = prec + (val < 0.0 ? 2 : 1); // . and -
    flen += vi >= 1000 ? 4 : vi >= 100 ? 3 : vi >= 10 ? 2 : 1;
    for (int i = flen; i < len; ++i)
      Serial.print(' ');
  }
  smartDelay(0);
}
```

RPS

```
//rps

uint16_t counter = 0;
uint32_t last_millis = 0;

#define interrupt_in 1    // Pin used as input for Hall Sensor signal
#define magnets      2    // Quantity of magnets on spinner/interrupts per cycle
#define period       1000 // [ms]

// RPS interruption routine - increments counter
void rps_ISR()
{
  counter++;
}

// RPS loop executed in main loop
void rps_loop()
{
  if (millis() - last_millis >= period)
  {
    detachInterrupt(interrupt_in);        // Disable interrupt while calculating
    rps = counter / magnets ;            // Encoder conversion

    rpsArray[0] = rps >> 8;             // Stores data in buffer for I2C packet
    rpsArray[1] = rps;

    counter = 0;                    // Restart counter
    last_millis = millis();              // Update lastmillis
    attachInterrupt(interrupt_in, rps_ISR, FALLING);        // Reenable interrupt
  }
}
```

SERVOS

```
//servos

//#define SERVOS_DEBUG

//  1 ELEV 2 AIL  3 RUD
//  em ordem de placa megadrive 1.0
#define PINO_PWM3 7  //L
#define PINO_PWM1 6  //P
#define PINO_PWM2 A0 //A2

```

```
11  int LimInf_pwm1 = 2070;
12  int LimSup_pwm1 = 1569;
13  int LimInf_servo1 = -25;
14  int LimSup_servo1 = 15;
15
16  int LimInf_pwm2 = 1622;
17  int LimSup_pwm2 = 1249;
18  int LimInf_servo2 = -15;
19  int LimSup_servo2 = 15;
20
21  int LimInf_pwm3 = 1700;
22  int LimSup_pwm3 = 1243;
23  int LimInf_servo3 = -30;
24  int LimSup_servo3 = 30;
25
26  int lastRead = 1;
27  int pwm1_read = 0, cont1 = 0;
28  int pwm2_read = 0, cont2 = 0;
29  int pwm3_read = 0, cont3 = 0;
30  int pwm1_raw = 0, pwm2_raw = 0, pwm3_raw = 0;
31
32  #define n 15                                        //número de pontos da média móvel
33  int     filter_1[n], filter_2[n], filter_3[n];    //vetor com os valores para média móvel
34  long    acc_1 = 0, acc_2 = 0, acc_3 = 0;          //acumulador para somar os pontos da média móvel
35
36  void print_servos()
37  {
38    Serial.print("pwm1 raw: "); Serial.print(pwm1_raw);
39    Serial.print(" filt: ");    Serial.print(pwm1_filtered);
40    Serial.print(" servo: ");   Serial.print(servo1);
41
42    Serial.print("  pwm2 raw: "); Serial.print(pwm2_raw);
43    Serial.print(" filt: ");    Serial.print(pwm2_filtered);
44    Serial.print(" servo: ");   Serial.print(servo2);
45
46    Serial.print("  pwm3 raw: "); Serial.print(pwm3_raw);
47    Serial.print(" filt: ");    Serial.print(pwm3_filtered);
48    Serial.print(" servo: ");   Serial.println(servo3);
49  }
50
51  void servo_setup()
52  {
53    pinMode(PINO_PWM1, INPUT_PULLUP);
54    pinMode(PINO_PWM2, INPUT_PULLUP);
55    pinMode(PINO_PWM3, INPUT_PULLUP);
56  }
57
58  void servo_loop()
59  {
60    //read one channel at time (max read cycle period = 1 timeout)
61    //conditionals for filtering noise (500ms = -90 deg on servo)
62
63    switch ( lastRead ) {
64      case 1:
65        pwm1_read = pulseIn(PINO_PWM1, LOW, 30000);
66        if (pwm1_read > 400)
67        {
68          pwm1_raw = pwm1_read;
69          cont1 = 0;
70        } else if (pwm1_read < 50)
```

```
71        {
72          cont1++;
73          if (cont1 >= 3)
74          {
75            pwm1_raw = 0;
76          }
77        }
78
79        for (int i = n - 1; i > 0; i--) filter_1[i] = filter_1[i - 1];    //desloca os elementos do vetor de média móve
80        filter_1[0] = pwm1_raw;                                          //posição inicial do vetor recebe a leitura o
81        for (int i = 0; i < n; i++) acc_1 += filter_1[i];                 //faz a somatória do número de pontos
82        pwm1_filtered =  acc_1 / n;                                      //retorna a média móvel
83
84        servo1 = map (  pwm1_filtered, LimInf_pwm1, LimSup_pwm1, LimInf_servo1, LimSup_servo1);
85
86        acc_1 = 0;
87
88        lastRead = 2;
89        break;
90
91      case 2:
92        pwm2_read = pulseIn(PINO_PWM2, LOW, 30000);
93        if (pwm2_read > 400)
94        {
95          pwm2_raw = pwm2_read;
96          cont2 = 0;
97        } else if (pwm2_read < 50)
98        {
99          cont2++;
100         if (cont2 >= 3)
101         {
102           pwm2_raw = 0;
103         }
104       }
105
106       for (int i = n - 1; i > 0; i--) filter_2[i] = filter_2[i - 1];
107       filter_2[0] = pwm2_raw;
108       for (int i = 0; i < n; i++) acc_2 += filter_2[i];
109       pwm2_filtered =  acc_2 / n;
110
111       servo2 = map (pwm2_filtered, LimInf_pwm2, LimSup_pwm2, LimInf_servo2, LimSup_servo2);
112
113       servo2 = 0;
114
115       acc_2 = 0;
116
117       lastRead = 3;
118
119       break;
120
121     case 3:
122       pwm3_read = pulseIn(PINO_PWM3, LOW, 30000);
123       if (pwm3_read > 400)
124       {
125         pwm3_raw = pwm3_read;
126         cont3 = 0;
127       } else if (pwm3_read < 50)
128       {
129         cont3++;
130         if (cont3 >= 3)
```

```
131           {
132             pwm3_raw = 0;
133           }
134         }
135
136         for (int i = n - 1; i > 0; i--) filter_3[i] = filter_3[i - 1];
137         filter_3[0] = pwm3_raw;
138         for (int i = 0; i < n; i++) acc_3 += filter_3[i];
139         pwm3_filtered =  acc_3 / n;
140
141         servo3 = map (pwm3_filtered, LimInf_pwm3, LimSup_pwm3, LimInf_servo3, LimSup_servo3);
142
143         servo3 = 0;
144
145         acc_3 = 0;
146
147         lastRead = 1;
148         break;
149     }
150
151 #ifdef SERVOS_DEBUG
152   print_servos();
153 #endif
154
155 }
```

WOW

```
1  //wow
2
3  #include <NewPing.h>
4
5  #define TRIGGER_PIN   9   // Arduino pin tied to trigger pin on ping sensor.
6  #define ECHO_PIN      10  // Arduino pin tied to echo pin on ping sensor.
7  #define MAX_DISTANCE  40  // Maximum distance we want to ping for (in centimeters). Maximum sensor distance
8
9  #define WOW_HEIGHT   19  //altura entre sensor e solo, unidade em cm
10
11 NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing setup of pins and maximum distance.
12
13 unsigned int pingSpeed = 100; // How frequently are we going to send out a ping (in milliseconds). 50ms would
14 unsigned int pingTimer = 0;     // Holds the next ping time.
15
16 void echoCheck()
17 {
18   // Timer2 interrupt calls this function every 24uS where you can check the ping status.
19   if (sonar.check_timer())
20   {
21     if ( (sonar.ping_result / US_ROUNDTRIP_CM) > WOW_HEIGHT)
22     {
23       wow = 1; //em voo
24     } else
25     {
26       wow = 0; //em solo
27     }
28     // Serial.print("WOW: "); Serial.print(wow);
29   }
```

```
30
31  }
32
33  void wow_loop()
34  {
35    if (millis() >= pingTimer)     // pingSpeed milliseconds since last ping, do another ping.
36    {
37      pingTimer += pingSpeed;      // Set the next ping time.
38      sonar.ping_timer(echoCheck); // Send out the ping, calls "echoCheck" function every 24uS where you can check
39    }
40  }
```

DEBUG

```
1   //print
2
3   #define print_period 100 //milisegundos
4
5   void print_loop()
6   {
7     if (millis() - print_temp_aux >= print_period)
8     {
9       print_temp_aux = millis();
10
11     // servo1 = 10; servo2 = 20; servo3 = 30; //teste
12     // rps = 60; wow = 1; //teste
13
14      //rps - sensor hall
15      Serial.print(F("rps:")); Serial.print(rps, 1);    Serial.print(F(",\t"));
16
17      //wow - ultrassonico
18      Serial.print(F("wow:")); Serial.print(wow, 1);    Serial.print(F(",\t"));
19
20      //servo - pwm
21      Serial.print(F("servo1:")); Serial.print(servo1); Serial.print(F("\t"));
22      Serial.print(F("servo2:")); Serial.print(servo2); Serial.print(F("\t"));
23      Serial.print(F("servo3:")); Serial.print(servo3); Serial.print(F("\t"));
24
25      Serial.print(F("pwm1:")); Serial.print(pwm1_filtered); Serial.print(F("\t"));
26      Serial.print(F("pwm2:")); Serial.print(pwm2_filtered); Serial.print(F("\t"));
27      Serial.print(F("pwm3:")); Serial.print(pwm3_filtered); Serial.print(F("\t"));
28
29      //latitude - gps
30      Serial.print(F("lat:"));
31      //Serial.print(latint);     Serial.print(F(",\t")); //usar para debug
32      //Serial.print(latuint);     Serial.print(F(",\t")); //usar para debug
33      Serial.print(latfloat, 6); Serial.print(F("\t"));
34
35      //longitude - gps
36      Serial.print(F("long:"));
37      //Serial.print(longint);     Serial.print(F(",\t"));  //usar para debug
38      //Serial.print(longuint);     Serial.print(F(",\t"));  //usar para debug
39      Serial.print(longfloat, 6); Serial.print(F("\t"));
40
41      //atitude - gps
42      Serial.print(F("alt:"));
43      //Serial.print(altint);     Serial.print(F(",\t"));
```

```
    Serial.print(altfloat, 2); Serial.print(F("\t"));

    Serial.println();
  }
}
```

I2C

```
//I2C
#include <Wire.h>

#define addr_i2c_slave 8

void i2c_transmission_setup()
{
  Wire.begin(addr_i2c_slave);  // join i2c bus with address #8
  Wire.onRequest(I2C_packet);   // register event
}

void I2C_packet()
{
  //servo1 = 10+90; servo2 = 20+90; servo3 = 30+90; //for debugging
  //rps = 60; wow = 1; //para debug
  //rps
  Wire.write(rpsArray, 2);
  //wow
  Wire.write(wow);
  //servos
  uint8_t servo1_i2c = 90; servo1_i2c = servo1_i2c + servo1;
  uint8_t servo2_i2c = 90; servo2_i2c = servo2_i2c + servo2;
  uint8_t servo3_i2c = 90; servo3_i2c = servo3_i2c + servo3;
  Wire.write(servo1_i2c); Wire.write(servo2_i2c); Wire.write(servo3_i2c);
  //gps
  Wire.write(latArray, 4); Wire.write(longArray, 4); Wire.write(altArray, 2);
}
```

ANEXO C: FIRMWARE DE MÓDULO TERRESTRE

MAIN

```
byte data_byte[45];
uint32_t last_millis = 0;
byte incoming_byte;
uint8_t i_header = 0;
uint8_t i_data = 4;
bool sync_flag = 0;
bool print_flag = 0;

#pragma GCC optimize ("-Ofast")

void setup()
{
  Serial.begin(115200);
  Serial.println("DAS - Reciever Unit");
}

void print_DAS()
{
  uint32_t rtc = (data_byte[4]);
  rtc = (rtc << 8) | (data_byte[5]);
  rtc = (rtc << 8) | (data_byte[6]);
  uint32_t rtc_sec = rtc / 10;
  uint8_t  cs =  rtc - (rtc_sec * 10);

  uint32_t rpm = (data_byte[7] << 8) | data_byte[8];
  rpm = rpm * 60;

  uint8_t wow = data_byte[9];

  float hp = (data_byte[10] << 8) | data_byte[11];
  hp = hp / 10;

  float vcas = (data_byte[12] << 8) | data_byte[13];
  vcas = vcas / 100;

  uint32_t latuint = (data_byte[14]);
  latuint = (latuint << 8) | (data_byte[15]);
  latuint = (latuint << 8) | (data_byte[16]);
  latuint = (latuint << 8) | (data_byte[17]);
  int32_t latint = latuint - 2147483647;
  if ( latint == -2147483647)
  {
    latint = 0;
  }
  float latfloat = latint;
  latfloat /= 1000000;

  uint32_t longuint = (data_byte[18]);
  longuint = (longuint << 8) | (data_byte[19]);
  longuint = (longuint << 8) | (data_byte[20]);
  longuint = (longuint << 8) | (data_byte[21]);
  float longfloat = (longuint);
  longfloat = (longfloat  - 2147483647);
  if ( longfloat == -2147483647)
  {
```

```
56        longfloat = 0;
57      }
58      longfloat = longfloat / 1000000;
59
60      uint16_t altint = (data_byte[22] << 8) | data_byte[23];
61      float altfloat = altint / 100;
62
63      float maghead = (data_byte[24] << 8) | data_byte[25];
64      maghead = maghead / 100;
65
66      float servo_1 = (data_byte[26] << 8) | data_byte[27];
67      servo_1 = servo_1 - 90;
68
69      float servo_2 = (data_byte[28] << 8) | data_byte[29];
70      servo_2 = servo_2 - 90;
71
72      float servo_3 = (data_byte[30] << 8) | data_byte[31];
73      servo_3 = servo_3 - 90;
74
75      float gyroXrate = (data_byte[32] << 8) | data_byte[33];
76      gyroXrate -= 25000;
77      gyroXrate /= 100;
78
79      float gyroYrate = (data_byte[34] << 8) | data_byte[35];
80      gyroYrate -= 25000;
81      gyroYrate /= 100;
82
83      float gforceZ = (data_byte[36] << 8) | data_byte[37];
84      gforceZ -= 20000;
85      gforceZ /= 10000;
86
87      float kalAngleY = (data_byte[38] << 8) | data_byte[39];
88      kalAngleY -= 18000; kalAngleY /= 100;
89
90      float kalAngleX = (data_byte[40] << 8) | data_byte[41];
91      kalAngleX -= 18000; kalAngleX /= 100;
92
93      float temp =  data_byte[42];
94
95      Serial.print(rtc_sec);   Serial.print(F("."));   Serial.print(cs);   Serial.print(F(" \t "));
96      Serial.print(rpm);   Serial.print(F(" \t "));
97      Serial.print(wow);   Serial.print(F(" \t "));
98      Serial.print(hp, 2);   Serial.print(F(" \t "));
99      Serial.print(vcas);   Serial.print(F(" \t "));
100     Serial.print(latfloat, 6);   Serial.print(F(" \t "));
101     Serial.print(longfloat, 6);   Serial.print(F(" \t "));
102     Serial.print(altfloat, 2);   Serial.print(F(" \t "));
103     Serial.print(maghead, 2);   Serial.print(F(" \t "));
104     Serial.print(servo_1);   Serial.print(F(" \t "));
105     Serial.print(servo_2);   Serial.print(F(" \t "));
106     Serial.print(servo_3);   Serial.print(F(" \t "));
107     Serial.print(gyroXrate);   Serial.print(F(" \t "));
108     Serial.print(gyroYrate);   Serial.print(F(" \t "));
109     Serial.print(gforceZ);   Serial.print(F(" \t "));
110     Serial.print(kalAngleY);   Serial.print(F(" \t "));
111     Serial.print(kalAngleX);   Serial.print(F(" \t "));
112     Serial.print(temp);
113     Serial.print(F("\n"));
114   }
115
```

```
116 void loop()
117 {
118   if (Serial.available() > 0)
119   {
120     incoming_byte = Serial.read();
121     if (incoming_byte == 255 && sync_flag == 0)
122     {
123       data_byte[i_header] = incoming_byte;
124       i_header++;
125       if (i_header > 3)
126       {
127         i_header = 0;
128         sync_flag = 1;
129       }
130     } else if (sync_flag == 1)
131     {
132       data_byte[i_data] = incoming_byte;
133       i_data = i_data + 1;
134       if (i_data > 43)
135       {
136         i_data = 4;
137         sync_flag = 0;
138         print_flag = 1;
139       }
140     } else
141     {
142       i_header = 0;
143     }
144
145     if (print_flag == 1)
146     {
147       print_DAS();
148       print_flag = 0;
149     }
150   }
151 }
```

ANEXO D: FIRMWARE DE IMU

MAIN

```
/*
Urutau Aerodesign 2018
Jake - Jacobus L de Jager
13/10/18
IMU 18 v2.0
Firmware para IMU, unidade de medida inercial, para o DAS 18, baseado em stm32
*/

#include "stm32f103c8t6.h"
#include "mbed.h"
#include "MPU9250.h"

I2CSlave slave(PB_9, PB_8);               //interface with DAS
Ticker print_ticker;                      //ticker (timer)
float sum = 0;
uint32_t sumCount = 0;
MPU9250 mpu9250;
Timer t;
volatile bool newData = false;

InterruptIn isrPin(PB_12);

void mpuisr()
{
    newData=true;
}

bool print_flag = 1;
#define data_qt 12
uint8_t i2c_data[data_qt];
float maghead;

char logo[8][100] = {
    "##########################################################################################\n",
    "  _   _           _                 _                 _          _            \n",
    " | | | |_ __ _   _| |_ __ _ _   _     / \\   ___ _ __ ___   __| | ___  ___(_) __ _ _ __  \n",
    " | | | | '__| | | | __/ _` | | | |   / _ \\ / _ \\ '__/ _ \\ / _` |/ _ \\/ __| |/ _` | '_ \\ \n",
    " | |_| | |  | |_| | || (_| | |_| |  / ___ \\  __/ | | (_) | (_| |  __/\\__ \\ | (_| | | | |\n",
    "  \\___/|_|   \\__,_|\\__\\__,_|\\__,_| /_/   \\_\\___|_|  \\___/ \\__,_|\\___||___/_|\\__, |_| |_|\n",
    "                                                                          |___/       \n",
    "##########################################################################################\n"
};

void set_print_flag()
{
    print_flag = 1;
}


int main()
{
    confSysClock();     //Configure system clock (72MHz HSE clock, 48MHz USB clock)

    Serial      pc(PA_2, PA_3);
    pc.baud(115200);
```

```
56      i2c.frequency(400000);
57
58      slave.address((0x09)<< 1);
59      //print_ticker.attach(&set_print_flag, 0.1); // ticker for print function
60
61      for(uint8_t i = 0; i < 8; i++) {
62          pc.printf(logo[i]);
63      }
64      pc.printf("IMU - 18 Urutau Aerodesign\n");
65      //serial.printf("accX\taccY\taccZ\trotX\trotY\trotZ\tmagX\tmagY\tmagZ\n");
66
67      pc.printf("CPU SystemCoreClock is %d Hz\r\n", SystemCoreClock);
68
69      t.start();
70      isrPin.rise(&mpuisr);
71
72      // Read the WHO_AM_I register, this is a good test of communication
73      uint8_t whoami = mpu9250.readByte(MPU9250_ADDRESS, WHO_AM_I_MPU9250);  // Read WHO_AM_I register for MPU-9250
74      pc.printf("I AM 0x%x\n\r", whoami);
75      pc.printf("I SHOULD BE 0x73\n\r");
76
77      if (whoami == 0x73) { // WHO_AM_I should always be 0x68
78          pc.printf("MPU9250 is online...\n\r");
79          wait(1);
80
81
82          mpu9250.resetMPU9250(); // Reset registers to default in preparation for device calibration
83          mpu9250.calibrateMPU9250(gyroBias, accelBias); // Calibrate gyro and accelerometers, load biases in bias re
84          pc.printf("x gyro bias = %f\n\r", gyroBias[0]);
85          pc.printf("y gyro bias = %f\n\r", gyroBias[1]);
86          pc.printf("z gyro bias = %f\n\r", gyroBias[2]);
87          pc.printf("x accel bias = %f\n\r", accelBias[0]);
88          pc.printf("y accel bias = %f\n\r", accelBias[1]);
89          pc.printf("z accel bias = %f\n\r", accelBias[2]);
90          wait(2);
91          mpu9250.initMPU9250();
92          pc.printf("MPU9250 initialized for active data mode....\n\r"); // Initialize device for active mode read of a
93          mpu9250.initAK8963(magCalibration);
94          pc.printf("AK8963 initialized for active data mode....\n\r"); // Initialize device for active mode read of ma
95          pc.printf("Accelerometer full-scale range = %f  g\n\r", 2.0f*(float)(1<<Ascale));
96          pc.printf("Gyroscope full-scale range = %f  deg/s\n\r", 250.0f*(float)(1<<Gscale));
97          if(Mscale == 0) pc.printf("Magnetometer resolution = 14  bits\n\r");
98          if(Mscale == 1) pc.printf("Magnetometer resolution = 16  bits\n\r");
99          if(Mmode == 2) pc.printf("Magnetometer ODR = 8 Hz\n\r");
100         if(Mmode == 6) pc.printf("Magnetometer ODR = 100 Hz\n\r");
101         wait(2);
102     } else {
103         pc.printf("Could not connect to MPU9250: \n\r");
104         pc.printf("%#x \n",  whoami);
105
106
107         while(1) ; // Loop forever if communication doesn't happen
108     }
109
110     mpu9250.getAres(); // Get accelerometer sensitivity
111     mpu9250.getGres(); // Get gyro sensitivity
112     mpu9250.getMres(); // Get magnetometer sensitivity
113     pc.printf("Accelerometer sensitivity is %f LSB/g \n\r", 1.0f/aRes);
114     pc.printf("Gyroscope sensitivity is %f LSB/deg/s \n\r", 1.0f/gRes);
115     pc.printf("Magnetometer sensitivity is %f LSB/G \n\r", 1.0f/mRes);
```

```
116     magbias[0] = +470.;
117     magbias[1] = +120.;
118     magbias[2] = +125.;
119 
120     while(1) {
121 
122         uint8_t i2c_status = slave.receive();
123         switch (i2c_status) {
124             case I2CSlave::ReadAddressed:
125                 for (uint8_t i = 0; i < data_qt; i++) {
126                     slave.write(i2c_data[i]);
127                 }
128                 myled= !myled;
129                 break;
130             case I2CSlave::WriteGeneral:
131                 break;
132             case I2CSlave::WriteAddressed:
133                 break;
134         }
135 
136 
137         static int readycnt=0;
138         // If intPin goes high, all data registers have new data
139 
140 #if USE_ISR
141         if(newData) {
142             newData=false;
143             mpu9250.readByte(MPU9250_ADDRESS, INT_STATUS);  //? need this with ISR
144 #else
145         if(mpu9250.readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01) {  // On interrupt, check if data ready interrupt
146 #endif
147             readycnt++;
148             mpu9250.readAccelData(accelCount);  // Read the x/y/z adc values
149             // Calculate the accleration value into actual g's
150             ax = (float)accelCount[0]*aRes - accelBias[0];  // get actual g value, this depends on scale being set
151             ay = (float)accelCount[1]*aRes - accelBias[1];
152             az = (float)accelCount[2]*aRes - accelBias[2];
153 
154             mpu9250.readGyroData(gyroCount);  // Read the x/y/z adc values
155             // Calculate the gyro value into actual degrees per second
156             gx = (float)gyroCount[0]*gRes - gyroBias[0];  // get actual gyro value, this depends on scale being set
157             gy = (float)gyroCount[1]*gRes - gyroBias[1];
158             gz = (float)gyroCount[2]*gRes - gyroBias[2];
159 
160             mpu9250.readMagData(magCount);  // Read the x/y/z adc values
161             // Calculate the magnetometer values in milliGauss
162             // Include factory calibration per data sheet and user environmental corrections
163             mx = (float)magCount[0]*mRes*magCalibration[0] - magbias[0];  // get actual magnetometer value, this depe
164             my = (float)magCount[1]*mRes*magCalibration[1] - magbias[1];
165             mz = (float)magCount[2]*mRes*magCalibration[2] - magbias[2];
166         }
167 
168         Now = t.read_us();
169         deltat = (float)((Now - lastUpdate)/1000000.0f) ; // set integration time by time elapsed since last filter
170         lastUpdate = Now;
171 
172         sum += deltat;
173         sumCount++;
174 
175         // Pass gyro rate as rad/s
```

```
176         uint32_t us = t.read_us();
177         mpu9250.MadgwickQuaternionUpdate(ax, ay, az, gx*PI/180.0f, gy*PI/180.0f, gz*PI/180.0f,  my,  mx, mz);
178         us = t.read_us()-us;
179 
180         // Serial print and/or display at 0.5 s rate independent of data rates
181         delt_t = t.read_ms() - count;
182         if (delt_t > 50) { // update LCD once per half-second independent of read rate
183             pc.printf("readycnt %d us %d\n",readycnt,us);
184             readycnt=0;
185             pc.printf("ax = %f", 1000*ax);
186             pc.printf(" ay = %f", 1000*ay);
187             pc.printf(" az = %f  mg\n\r", 1000*az);
188 
189             pc.printf("gx = %f", gx);
190             pc.printf(" gy = %f", gy);
191             pc.printf(" gz = %f  deg/s\n\r", gz);
192 
193             pc.printf("gx = %f", mx);
194             pc.printf(" gy = %f", my);
195             pc.printf(" gz = %f  mG\n\r", mz);
196 
197             tempCount = mpu9250.readTempData();  // Read the adc values
198             temperature = ((float) tempCount) / 333.87f + 21.0f; // Temperature in degrees Centigrade
199             pc.printf("temperature = %f  C\n\r", temperature);
200 
201             pc.printf("q0 = %f\n\r", q[0]);
202             pc.printf("q1 = %f\n\r", q[1]);
203             pc.printf("q2 = %f\n\r", q[2]);
204             pc.printf("q3 = %f\n\r", q[3]);
205 
206             yaw   = atan2(2.0f * (q[1] * q[2] + q[0] * q[3]), q[0] * q[0] + q[1] * q[1] - q[2] * q[2] - q[3] * q[3]);
207             pitch = -asin(2.0f * (q[1] * q[3] - q[0] * q[2]));
208             roll  = atan2(2.0f * (q[0] * q[1] + q[2] * q[3]), q[0] * q[0] - q[1] * q[1] - q[2] * q[2] + q[3] * q[3]);
209             pitch *= 180.0f / PI;
210             yaw   *= 180.0f / PI;
211             yaw   -= 13.8f; // Declination at Danville, California is 13 degrees 48 minutes and 47 seconds on 2
212             roll  *= 180.0f / PI;
213 
214             pc.printf("Yaw, Pitch, Roll: %f %f %f\n\r", yaw, pitch, roll);
215             pc.printf("average rate = %f\n\r", (float) sumCount/sum);
216 
217             float heading = atan2(my, mx);
218             if (heading < 0)
219             {
220                 heading += 2 * PI;
221             }
222             maghead = heading * 180 / PI;
223 
224             pc.printf("MagHead: %f\n\r", maghead);
225 
226             count = t.read_ms();
227             sum = 0;
228             sumCount = 0;
229 
230             uint16_t uint_maghead = (uint16_t)(maghead*100);
231             uint16_t uint_gx = (uint16_t)((gx+250)*100);
232             uint16_t uint_gy = (uint16_t)((gy+250)*100);
233             uint16_t uint_az = (uint16_t)((az+2)*10000);
234             uint16_t uint_roll = (uint16_t)((roll+180)*100);
235             uint16_t uint_pitch = (uint16_t)((pitch+180)*100);
```

```
236
237             i2c_data[0] = uint_maghead >> 8;
238             i2c_data[1] = uint_maghead;
239             i2c_data[2] = uint_gx >> 8;
240             i2c_data[3] = uint_gx;
241             i2c_data[4] = uint_gy >> 8;
242             i2c_data[5] = uint_gy;
243             i2c_data[6] = uint_az >> 8;
244             i2c_data[7] = uint_az;
245             i2c_data[8] = uint_roll >> 8;
246             i2c_data[9] = uint_roll;
247             i2c_data[10] = uint_pitch >> 8;
248             i2c_data[11] = uint_pitch;
249
250             pc.printf("uint MagHead: %d  ", uint_maghead);
251             pc.printf("uint Gx: %d   ", uint_gx);
252             pc.printf("uint Gy: %d   ", uint_gy);
253             pc.printf("uint Az: %d   ", uint_az);
254             pc.printf("uint roll: %d   ", uint_roll);
255             pc.printf("uint pitch: %d   ", uint_pitch);
256             pc.printf("\n");
257
258             for (uint8_t i = 0; i < data_qt; i++)
259             {
260                 pc.printf("%d\t", i2c_data[i]);
261             }
262             pc.printf("\n");
263         }
264     }
265 }
```