UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Bruno Lotto Bagarini

Primeiros passos no desenvolvimento de um sistema embarcado para controle e instrumentação de um veículo aéreo não tripulado

São Carlos
2018

Bruno Lotto Bagarini

Primeiros passos no desenvolvimento de um sistema embarcado para controle e instrumentação de um veículo aéreo não tripulado

Monografia apresentada ao Curso de Engenharia Elétrica com Ênfase em Eletrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Edson Gesualdo

VERSÃO CORRIGIDA

São Carlos

2018

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Bagarini, Bruno Lotto
B144p Primeiros passos no desenvolvimento de um sistema embarcado para controle e instrumentação de um veículo aéreo não tripulado / Bruno Lotto Bagarini; orientador Edson Gesualdo. São Carlos, 2018.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2018.

1. Veículo Aéreo Não Tripulado. 2. Sistema Embarcado. 3. Controladora de voo. 4. Hardware. 5. Software. 6. Transceptor RF. 7. Receptor GPS. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Bruno Lotto Bagarini

Título: "Primeiros passos no desenvolvimento de um sistema embarcado para controle e instrumentação de um veículo aéreo não tripulado"

Trabalho de Conclusão de Curso defendido e aprovado em 26/06/2018,

com NOTA 9,5 (nove, cinco), **pela Comissão Julgadora:**

Prof. Assistente Edson Gesualdo - Orientador - SEL/EESC/USP

Prof. Associado Evandro Luis Linhari Rodrigues - SEL/EESC/USP

Dra. Tania Regina Tronco - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

Ao livre mercado, à democracia e à liberdade de expressão, pilares fundamentais da civilização.

AGRADECIMENTOS

A toda minha família, meus pais, minha avó e minha irmã, por todo o carinho e compreensão durante a graduação.

Ao meu pai, que sempre me ensinou o valor do trabalho duro.

Ao meu avô, que sempre olhou por mim onde quer que eu esteja.

A minha mãe, por sempre querer o meu melhor.

A Deus, por me dar força e saúde para todos os dias acordar cedo e trabalhar.

A todos os docentes e funcionários que se dedicam todos os dias a fazer da USP a melhor universidade latino-americana e que contribuíram com a elaboração deste trabalho.

Aos amigos da turma Eletrônica 014 por todo o companheirismo e momentos marcantes vividos juntos.

Aos amigos da EESC USP Formula SAE, por todas as noites em claro, churrascos na oficina e conquistas ao longo destes cinco anos.

“For what is a man, what has he got?
If not himself, then he has naught
To say the things he truly feels
And not the words of one who kneels
The record shows: I took the blows
And did it my way!”
Frank Sinatra

RESUMO

BAGARINI, B. L. **Primeiros passos no desenvolvimento de um sistema embarcado para controle e instrumentação de um veículo aéreo não tripulado.** 2018. B144 p. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

Veículos aéreos não tripulados (VANTs) estão se tornando cada vez mais comuns na sociedade e estão sendo utilizados nas mais diversas aplicações. Nessa monografia foi descrito o início do desenvolvimento de um sistema embarcado para controle e instrumentação de um VANT para pulverização agrícola. Este VANT tem por finalidade resolver uma série de problemas dos meios tradicionais de pulverização, como desperdício de defensivos agrícolas e contaminação ambiental e pessoal. Esse documento contempla o projeto de *hardware*, desde a coleta de requisitos, escolha de componentes até o projeto e manufatura de placa de circuito impresso. Apresenta também a modelagem de *software* orientado a objetos e a implementação de dispositivos para comunicação com a estação de controle de solo via RF e recepção de sinal GPS. Os testes preliminares de alcance do transceptor RF e precisão do receptor GPS indicaram que o sistema ainda precisa ser melhorado em alguns aspectos. Testes adicionais serão necessários quando o VANT estiver finalizado para mais detalhes a respeito da performance do sistema embarcado. Os trabalhos futuros compreendem o desenvolvimento de leitura e calibração do IMU, implementação de um filtro estimador, desenvolvimento de malhas de controle para estabilização e navegação da aeronave, interpretação dos sinais do rádio controlador e modelagem do sistema de pulverização.

Palavras-chave: Veículo Aéreo Não Tripulado. Sistema Embarcado. Controladora de voo. *Hardware*. *Software*. Transceptor RF. Receptor GPS.

ABSTRACT

BAGARINI, B. L. **First steps in the development of an embedded system for control and instrumentation of an unmanned aerial vehicle.** 2018. B144 p. Monografia (Trabalho de Conclusão de Curso) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2018.

Unmanned aerial vehicles (UAV) are becoming more and more common in society and they are being used in various applications. In this thesis was described the beginning of the development of an embedded system for control and instrumentation of an UAV for crop dusting. This UAV aims to solve a number of problems in spraying traditional means, such as wasting of agricultural pesticides, environmental and personal contamination. This document includes hardware design, from the collection of requirements, components choice to design and manufacture of a printed circuit board. It also presents object-oriented software modeling and implementation of devices for communication with a ground control station via RF and for GPS signal reception. Preliminary tests of RF transceiver range and GPS receiver accuracy have indicated that the system still needs to be improved in some ways. Additional tests will be necessary when the UAV is finalized for more details about the embedded system performance. Future work includes the development of reading and calibration of the IMU, implementation of an estimator filter, development of control meshes for aircraft stabilization and navigation, interpretation of signals from radio controller and spraying system modeling.

Keywords: Unmanned Aerial Vehicle. Embedded System. Flight Controller. Hardware. Software. RF Transceiver. GPS Receiver.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AHRS	*Attitude and Heading Reference System*
ANAC	Agência Nacional de Aviação Civil
ANATEL	Agência Nacional de Telecomunicações
ASCII	*American Standard Code for Information Interchange*
Bps	*Bytes per second*
bps	*bits per second*
CAD	*Computer Aided Design*
CAN	*Controller Area Network*
DSP	*Digital Signal Processor*
FMU	*Flight Management Unit*
FPGA	*Field Programmable Gate Array*
GCS	*Ground Control Station*
GNSS	*Global Navigation Satellite System*
GPS	*Global Position System*
I2C	*Inter-Integrated Circuit*
IDE	*Integrated Development Environment*
IMU	*Inertial Measurement Unit*
LED	*Light Emitting Diode*
LNA	*Low Noise Amplifier*
MAVLink	*Micro Air Vehicle Link*
MEMS	*MicroElectroMechanical Systems*
NMEA	*National Marine Electronics Association*
OOAD	*Object-Oriented Analysis and Design*
PA	*Power Amplifier*
PCB	*Printed Circuit Board*
PCM	*Pulse Code Modulation*
PLL	*Phase-Locked Loop*
PPM	*Pulse Position Modulation*
PWM	*Pulse Width Modulation*
RF	Radiofrequência
SD	*Secure Digital*

SMD	*Surface Mounted Device*
SPI	*Serial Peripheral Interface*
UART	*Universal Asynchronous Receiver/Transmitter*
UAV	*Unmanned Aerial Vehicle*
UML	*Unified Modeling Language*
USB	*Universal Serial Bus*
USP	Universidade de São Paulo
VANT	Veículo Aéreo Não Tripulado

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo é exposta uma breve apresentação do trabalho, bem como a motivação e os objetivos para a realização do mesmo. Por fim, é apresentada a estrutura em que o documento foi organizado.

1.1 Apresentação

Este documento tem por objetivo apresentar os primeiros passos do desenvolvimento de um sistema embarcado para controle e instrumentação de um Veículo Aéreo Não Tripulado. Tal veículo está sendo projetado e fabricado pela HeliDrop, *startup* nascida no campus da USP em São Carlos voltada ao desenvolvimento de soluções otimizadas para pulverização agrícola.

O desenvolvimento de um VANT para pulverização agrícola é bastante relevante uma vez que o agronegócio constantemente sofre pressões da sociedade e do mercado para tornar-se mais social e economicamente sustentável e os métodos atuais de controle de pragas ainda possuem grandes margens para melhorias.

Os métodos tradicionais de pulverização podem ser classificados em via aérea, com aviões agrícolas e, recentemente, helicópteros e via terrestre, com tratores e pulverizadores costais. Todos eles apresentam restrições quanto ao uso em relevos mais acidentados (SECCO, ROSA, *et al.*, 2013), o que não é um problema quando utiliza-se um VANT. Embora o avião seja o mais rápido, sua velocidade prejudica a qualidade das aplicações e agrava o efeito de deriva, que é o carregamento das partículas de defensivos pelo vento. A deriva está associada à morte de abelhas (PACÍFICO-DA-SILVA, MELO e SOTO-BLANCO, 2016) e à contaminação do meio ambiente, visto que os agrotóxicos podem ser levados a rios, zonas protegidas e áreas residenciais (FILIZOLA, FERRACINI, *et al.*, 2002). Tais métodos de pulverização estão defasados porque ainda não permitem um controle preciso da quantidade de defensivos aplicada em cada parte do terreno (REIS, QUEIROZ, *et al.*, 2010) e sua forma de comercialização também restringe a utilização das tecnologias de ponta aos grandes grupos agrícolas.

O projeto que está sendo desenvolvido atualmente pela HeliDrop é um VANT Classe 3 da ANAC (Drones Classe 3 (RPA com peso máximo de decolagem maior que 250g e até 25 kg), 2017), com peso máximo de decolagem de 25 kg e carga útil de até 15 kg, com rotinas de voo e pulverização totalmente automatizadas para possibilitar uma melhor uniformidade na pulverização e evitar o desperdício e sobredosagem de defensivos químicos. A opção por

desenvolver um VANT Classe 3 foi motivada principalmente pela facilidade no registro e regulamentação da aeronave junto à ANAC. Este modelo será capaz de pulverizar uma área de até 500 hectares por semana e possuirá autonomia de duas horas de voo. A capacidade de carga e autonomia de voo foram determinadas por meio de entrevistas com produtores e agrônomos que possuem a necessidade de um equipamento que apresente bom desempenho para realizar pulverização pontual e dispersão de agentes biológicos para controle de pragas.

Conforme pode-se notar pela Figura 1, a configuração da aeronave será um helicóptero de dois rotores entrelaçados, que possui algumas vantagens em relação aos modelos tradicionais, como maior estabilidade, maior capacidade de carga que helicópteros de dimensões próximas e o fato de não ser necessário um rotor de cauda.

Figura 1: Aeronave em CAD a ser equipada com a placa controladora.

Fonte: Autor.

O projeto está sendo realizado de maneira modular para permitir a evolução dos componentes sem perda de compatibilidade. Assim, à medida que o projeto atingir maior maturidade, será possível aumentar a capacidade de carga e autonomia de voo a partir de modificações mecânicas e eletrônicas no projeto inicial.

1.2 Motivação e objetivos

O sucesso de qualquer *drone* depende enormemente da performance e confiabilidade de sua placa controladora. Era necessária a utilização de uma placa controladora de qualidade a um baixo custo no protótipo apresentado anteriormente, porém ainda não há no Brasil um produto que possua tais características.

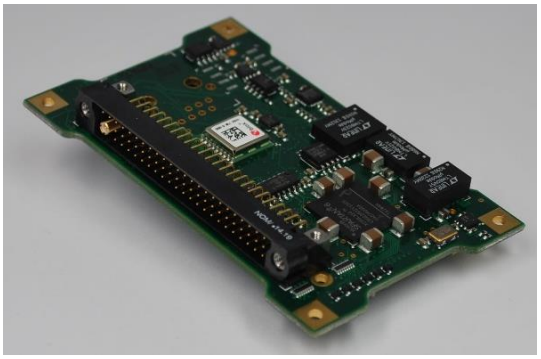
Entre as opções mais baratas e com desempenho razoável, há a controladora Pixhawk 2.1, ilustrada na Figura 2, a um preço de pouco mais de 250 dólares. Esta é uma controladora mantida pela comunidade de desenvolvedores Ardupilot, que se dedica a projetar placas controladoras para veículos autônomos (Ardupilot Home, 2018). Seu projeto de *hardware* e *software* são *open-source* e qualquer pessoa pode ter acesso ao seu conteúdo. A principal desvantagem desta placa é que, por ser desenvolvida para *hobby*, pode apresentar pequenos defeitos de *software* que são detectados apenas durante o voo. Além disso, por não se tratar de um projeto comercial, ela não possui um setor estruturado de suporte técnico, consistindo apenas de grupos de redes sociais em que os usuários trocam experiências e dúvidas.

Figura 2: Placa controladora de voo Pixhawk 2.1.

Fonte: (Pixhawk 2.1, The Cube).

Como opção que melhor privilegia performance e confiabilidade, existem os sistemas embarcados fornecidos pela empresa suíça weControl (weControl Home, 2018). A empresa é bastante sólida e opera desde 2000. Em 2003 a empresa participou de um consórcio que desenvolveu aeronaves não tripuladas para forças militares francesas (weControl History, 2018). Atualmente, possui como produto mais acessível a wePilot4000, exposta na Figura 3, cujo preço supera facilmente os 4 mil dólares, sendo, portanto, inacessível à empresa em que o trabalho foi desenvolvido.

Figura 3: Placa controladora de voo wePilot4000.



Fonte: (weControl Products , 2018).

Logo, optou-se pelo desenvolvimento de uma prova de conceito de uma placa controladora de voo dentro da própria *startup*, de forma a se obter um produto que pudesse ser customizável de acordo com o desenvolvimento do VANT, com boa performance e a um baixo custo. Além de fornecer uma maior liberdade para a escolha de periféricos e sensores, conta como ponto positivo a independência em relação a variações no câmbio e procedimentos de importação, que poderiam inviabilizar as opções importadas. A grande desvantagem em relação às opções importadas é a questão do tempo de desenvolvimento. Por se tratar de uma equipe ainda pequena, levará um tempo para se atingir a maturidade do projeto.

O sistema a ser desenvolvido deve possuir como requisitos:

- periféricos e sensores de qualidade e precisão,
- robustez em relação a falhas de seus componentes e
- poder de processamento para gerenciar todas as malhas de controle do veículo.

1.3 Estrutura deste documento

Este documento está dividido em capítulos. O Capítulo 1, Introdução, visa familiarizar o leitor com as motivações que levaram à elaboração deste trabalho. Em seguida temos o Capítulo 2, Embasamento teórico, que procura fornecer o conhecimento necessário ao leitor para acompanhar o capítulo seguinte, Desenvolvimento, no qual é detalhado o trabalho em seus pormenores. Já o Capítulo 4, Testes, resultados e discussões detalha alguns testes realizados no sistema e discute os resultados obtidos. Por fim, o Capítulo 5, Conclusão, recapitula brevemente o desenvolvimento do projeto e aponta para os trabalhos futuros a serem realizados com base na plataforma aqui descrita.

2 EMBASAMENTO TEÓRICO

Neste capítulo são expostas explicações a respeito dos quatro assuntos principais do trabalho: controladoras de voo, modelagem de sistemas orientados a objetos, transceptores wireless RF e sistema de navegação global por satélite.

2.1 Controladoras de voo

Uma placa controladora de voo nada mais é que um sistema que integra dispositivos de comunicação, sensores e atuadores a fim de controlar o voo de um VANT.

As subseções a seguir irão apresentar cada um dos componentes e periféricos mais comuns que integram uma placa controladora de voo.

2.1.1 *Flight Management Unit*

A FMU, ou *Flight Management Unit*, é o núcleo de uma placa controladora de voo. Por se tratar de uma unidade de gerenciamento, ela é responsável pela coleta e fusão de dados de sensores, comunicação com a estação de solo, salvamento de informações, gerenciamento de malhas de controle e acionamento de atuadores.

Para que a FMU apresente desempenho satisfatório, é desejável que possua muitas entradas e saídas digitais para futura expansão do sistema, e que possua alguns dos protocolos de comunicação mais utilizados, como UART, I2C, SPI, USB e CAN, além de possuir saídas PWM para os atuadores.

A FMU pode ser realizada em diversas tecnologias de *hardware*, como microcontroladores, DSPs e FPGAs. Pode ser também composta por apenas uma destas tecnologias ou por várias, formando um sistema híbrido. Além disso, pode ser sintetizada com diversos núcleos de processamento, sendo um deles o principal e os restantes coprocessadores para algumas tarefas específicas do sistema ou para obter redundância em caso de falhas.

2.1.2 Leitor de cartão SD

As controladoras de voo possuem um leitor de cartão SD, que possui vital importância uma vez que registra os dados de voo *on-board*. Estes dados são utilizados em caso de quedas, para se descobrir qual componente apresentou falha, ou em caso de mau funcionamento de

algum dispositivo, para se tentar entender o que exatamente está errado com o mesmo. Para que possa desempenhar esta função de depuração de falhas corretamente, o arquivo de dados a ser gravado no cartão SD geralmente é customizável. Desta forma, o usuário pode selecionar quais informações considera mais relevante registrar para posterior análise em *softwares* adequados. O cartão SD utiliza o protocolo de comunicação digital SPI, que garante altas taxas de transferência de dados, da ordem de alguns megabits por segundo (Mbps).

2.1.3 *Ground Control Station*

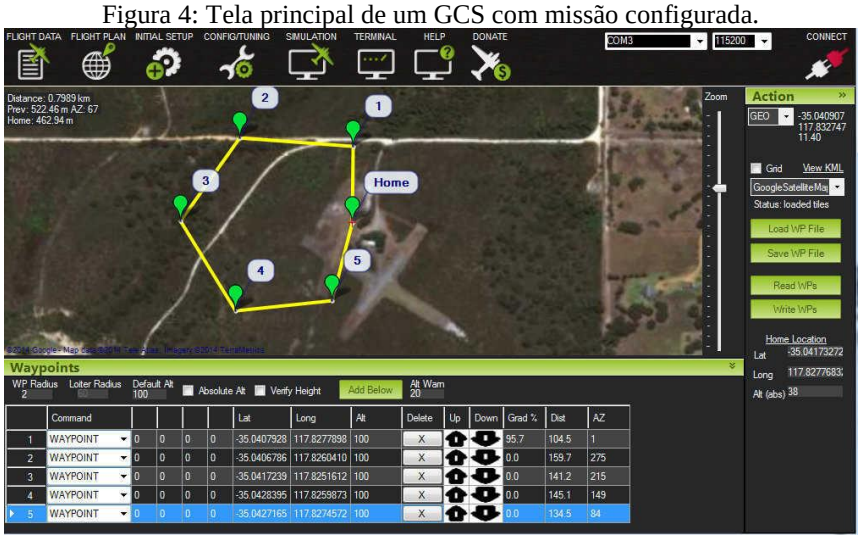
O *Ground Control Station*, ou GCS, é um *software* para computadores pessoais que possui duas funcionalidades principais: configuração dos diversos parâmetros da controladora de voo e monitoramento da aeronave durante o voo.

Entre os parâmetros principais a serem configurados via GCS, destacam-se: velocidade de transmissão dos dados, valores máximos e mínimos dos servos motores, taxa de aquisição de sensores, valores dos parâmetros das malhas de controle e as coordenadas a serem percorridas em voo autônomo.

Já os principais dados monitorados pelo usuário através do GCS são: localização, altura, atitude, velocidade e alertas de erros.

Na Figura 4 abaixo é apresentada a tela principal de um GCS *open-source*, o *Mission Planner*.

Figura 4: Tela principal de um GCS com missão configurada.



Fonte: (Mission Planner Home, 2016).

Na figura anterior, nota-se em seu centro o mapa da localidade onde a missão autônoma será operada. Os marcadores em verde indicam cada um dos *waypoints*, que são os pontos a serem percorridos durante a missão, e as linhas amarelas indicam o caminho a ser percorrido pelo VANT. Na aba superior é possível acessar as demais telas e estabelecer comunicação com a aeronave. No canto direito da tela é possível adicionar, remover e configurar *waypoints* e por fim, na parte de baixo da tela são apresentadas as informações de todos os *waypoints* já configurados em sequência.

2.1.4 Rádio Controlador

Os rádios controladores são dispositivos que operam em radiofrequência e que são responsáveis por enviar comandos do usuário para a aeronave. Sua função é realizar um *by-pass* da controladora de voo, ou seja, é possível a intervenção do usuário a qualquer momento do voo, permitindo o controle direto dos atuadores do *drone*.

A maioria destes dispositivos operam em frequência modulada e nas faixas de 900 MHz e 2,4 GHz, possuindo pelo menos 8 canais de saída. O protocolo de saída do receptor na grande maioria dos casos é o PWM, uma vez que estes controladores são herdados do aeromodelismo, no qual não há controladora de voo e o piloto controla diretamente cada atuador. Há também dispositivos com outros protocolos, como o PPM e o PCM, que são universais e possuem como vantagem a utilização de um único sinal de controle, bem como protocolos proprietários, como o SBUS da Futaba e o FPort da Frsky (LIANG, 2018).

Estes dispositivos também podem ser configurados para melhor atender ao usuário. Pode-se por exemplo inverter o sentido dos controles, trocar referências e regular o ponto médio e os extremos da saída de cada atuador conforme a configuração da aeronave.

A figura abaixo ilustra um rádio controlador comercial.

Figura 5: Rádio controlador Futaba T12K.

Fonte: (Futaba T12K 12 Channel 2.4GHz Combo with R3008SB (Mode 2)).

2.1.5 Receptor GPS

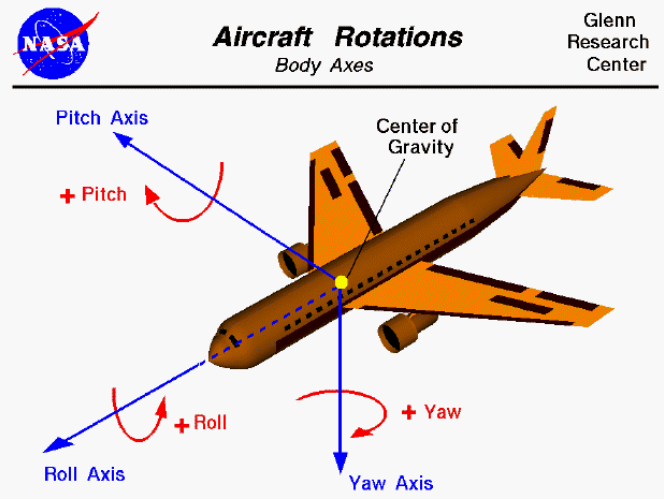
Este componente é muito importante para qualquer veículo autônomo, pois é ele que dá referência ao sistema de navegação. Em linhas gerais, o receptor GPS recebe o sinal de localização via satélite, com as coordenadas de latitude e longitude bem como a altitude da aeronave e repassa à FMU para que seja realizado o controle de rota do veículo. Este componente será explicado em maiores detalhes em uma seção posterior dentro deste mesmo capítulo.

2.1.6 *Inertial Measurement Unit*

A IMU, ou *Inertial Measurement Unit*, é um conjunto de sensores que tem por objetivo estimar a orientação de um corpo rígido, servindo desta maneira para indicar a atitude da aeronave (CAVALLO, CIRILLO, *et al.*, 2014).

Por atitude da aeronave entende-se como a orientação da mesma no espaço em relação a um referencial dado, como por exemplo, a base de decolagem. Conforme a figura abaixo, as aeronaves possuem três graus de liberdade para movimento angular em cada um de seus eixos, a saber: *roll*, *pitch* e *yaw*. O objetivo da IMU é obter cada um destes valores através de medições para que a controladora de voo possa estimar a atitude através do sistema de referência de atitude e direção (em inglês, AHRS).

Figura 6: Eixos de rotação em aeronaves.



Fonte: (BENSON).

Em VANTs é comum a utilização de sensores do tipo MEMS para realizar a medição da atitude da aeronave, devido a seu tamanho reduzido e baixo custo. Porém estes sensores

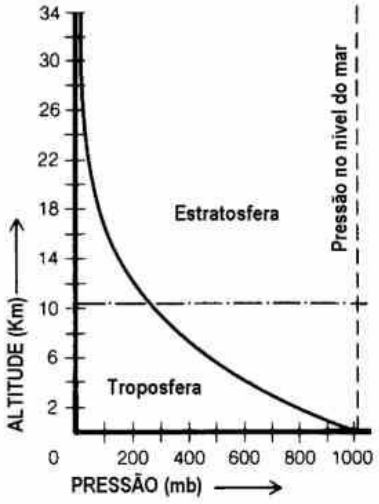
possuem baixa resolução, alto nível de ruído e erros de leitura dependentes do tempo, o que torna necessária a utilização de diversos sensores em conjunto, seguidos por algoritmos de filtragem e fusão de dados, como o Filtro Extendido de Kalman (CAVALLO, CIRILLO, *et al.*, 2014).

Via de regra são utilizados sistemas de medição com nove graus de liberdade. A integração numérica de um giroscópio triaxial acarreta em um erro de leitura que vai aumentando com o tempo. A implementação de um acelerômetro triaxial em conjunto com o giroscópio ajuda a reduzir tal erro, mas não é capaz de reduzi-lo a níveis toleráveis. Para isso, é utilizado ainda um magnetômetro triaxial (CAVALLO, CIRILLO, *et al.*, 2014), que por sua vez também apresenta pequenos erros devido a campos magnéticos próximos e à declinação magnética em diferentes pontos do planeta. Entretanto, com a informação dos três sensores nos três eixos a controladora de voo é capaz de estimar com boa tolerância a atitude do UAV através do AHRS.

2.1.7 Barômetro

O barômetro é um sensor que mede a pressão atmosférica. Há uma correlação bastante conhecida entre pressão atmosférica e altitude, que está ilustrada na figura abaixo.

Figura 7: Variação conjunta da pressão atmosférica com a altitude.



Fonte: (Variação com a altitude).

Desta forma, é possível inferir a respeito da altitude do veículo com a leitura da pressão atmosférica do barômetro. Porém, deve-se observar que a temperatura ambiente gera pequenas flutuações na pressão atmosférica e quando se deseja obter a altitude dentro de uma faixa de

algumas dezenas de centímetros, tais flutuações devem ser tratadas em *software* para que não haja um erro de leitura de altitude tão grande.

2.1.8 Saídas para servos motores

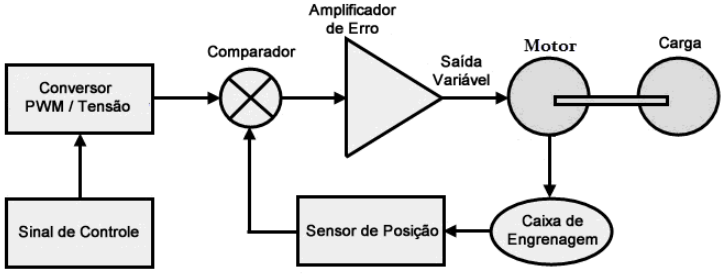
Servos motores se diferenciam dos demais através do acoplamento de um controlador e de um *encoder* em um motor comum. Garante-se assim controle, precisão e velocidade para aplicações em malha fechada (SILVEIRA). A figura a seguir mostra um servo motor bastante utilizado em aplicações de aeromodelismo.

Figura 8: Servomotor Futaba S3003.

Fonte: (Servo Motor - 5Kg Torque, 2018).

A estrutura básica de controle de um servo motor está exposta a seguir. Basicamente, é recebido um sinal de controle digital vindo da controladora de voo, através da modulação PWM. Um conversor interno transforma este sinal em níveis analógicos de tensão que são comparados com um sinal de erro vindo de um potenciômetro acoplado ao eixo do motor. A saída do comparador é então amplificada e aplicada ao motor (SILVEIRA).

Figura 9: Diagrama de blocos do servo motor.



Fonte: (SILVEIRA).

Devido às suas características de grande precisão e boa velocidade de reversão, os servos motores são amplamente utilizados em *drones*. As controladoras de voo possuem várias saídas PWM para o controle destes dispositivos, e estas saídas podem ser ajustadas para que os valores máximos e mínimos de posição do atuador não excedam os limites mecânicos da aeronave.

2.2 Modelagem de sistemas orientados a objetos

Antes de começar a escrever o código-fonte propriamente dito de um sistema complexo orientado a objeto, é interessante iniciar um processo de análise e projeto orientado a objeto (em inglês, OOAD) e para isso, pode ser utilizada a linguagem gráfica UML, que surgiu em meados da década de 90 com o intuito de padronizar os esquemas de modelagem de projetos orientados a objetos (DEITEL e DEITEL, 2001).

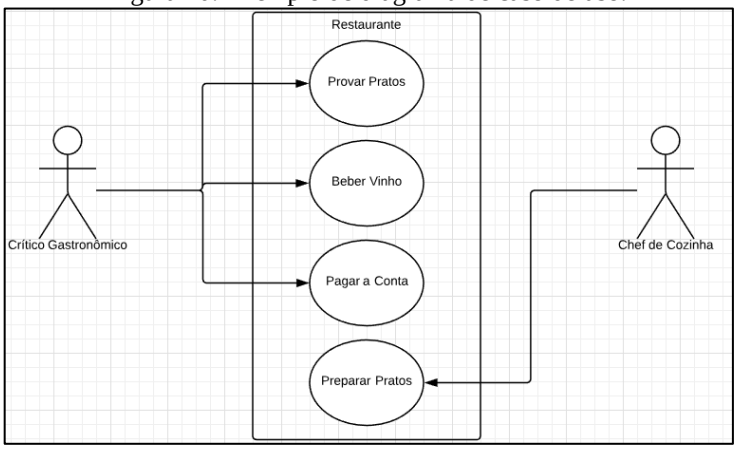
Esta seção irá apresentar as etapas mais importantes da modelagem de um sistema orientado a objeto através da utilização da UML e irá apresentar um exemplo simples de cada diagrama.

2.2.1 Diagrama de caso de uso

A primeira etapa da modelagem de um sistema consiste em especificar os seus clientes e cada um dos casos de uso do sistema pelos mesmos. Assim, este diagrama especifica qual ou quais tarefas cada cliente do sistema espera executar através de sua utilização. Os desenvolvedores devem ficar atentos a este diagrama, uma vez que, ao fim do desenvolvimento, o sistema deve, necessariamente, ser capaz de executar cada uma das tarefas especificadas (DEITEL e DEITEL, 2001).

A figura abaixo ilustra de maneira simples um diagrama de caso de uso. O retângulo da figura indica o sistema, no caso, um restaurante. Este mesmo restaurante terá duas pessoas usufruindo de seus serviços: um crítico gastronômico e um chef de cozinha. O chef de cozinha (cliente) utilizará o restaurante (sistema) para preparar pratos (caso de uso), enquanto o crítico gastronômico irá ao restaurante provar pratos, beber vinho e pagar a conta.

Figura 10: Exemplo de diagrama de caso de uso.

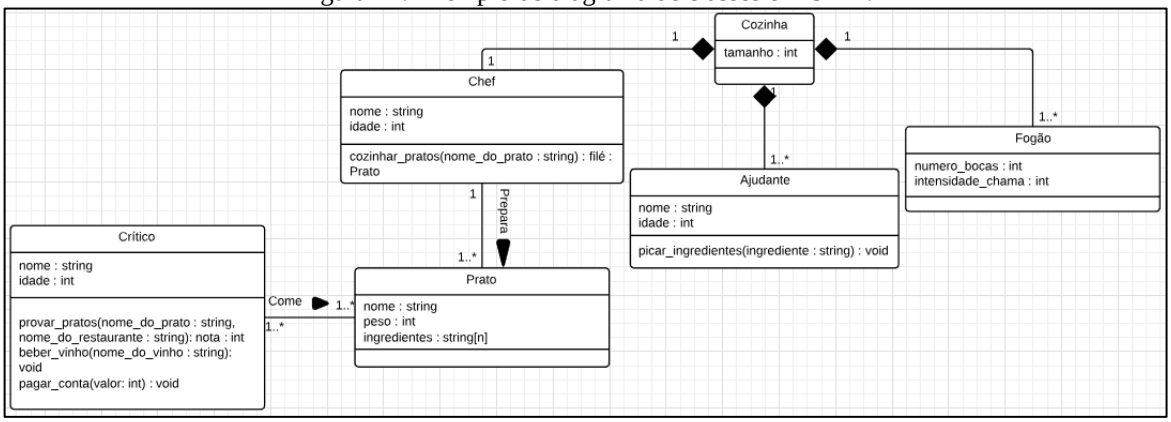


Fonte: Adaptado de (Diagrama de caso de uso).

2.2.2 Diagrama de classes

O diagrama de classes da UML tem por objetivo especificar cada uma das classes do sistema, seus atributos e operações. Cada classe é representada por um retângulo com três seções horizontais, que indicam de baixo pra cima: o nome da classe, os atributos e operações. As setas que interligam as classes indicam os relacionamentos entre elas. O número em cada extremidade das setas indica a multiplicidade da relação, ou seja, quantos objetos de uma classe se relacionam com tantos objetos da outra classe. Um losango na extremidade indica uma relação de composição. O nome em cima das setas acompanhado de um triângulo indica que tipo de serviço uma classe presta a outra (DEITEL e DEITEL, 2001). Pensando no exemplo anterior do restaurante, pode-se definir como classes: o crítico, a cozinha, o chef, ajudantes de cozinha, fogões e pratos preparados. Cada um deles possui características (o crítico, por exemplo, tem um nome), e pode ou não realizar operações (o chef, por exemplo, prepara pratos, mas o prato em si não realiza nenhuma operação). Note agora os índices junto às extremidades de cada seta: uma cozinha é composta por um chef, um ou mais fogões e um ou mais ajudantes. O chef prepara um ou mais pratos e o crítico come um ou mais destes mesmos pratos. Por simplicidade de visualização, os dois retângulos de atributos e operações podem ser omitidos. O diagrama de classes para este problema é exposto a seguir.

Figura 11: Exemplo de diagrama de classes em UML.

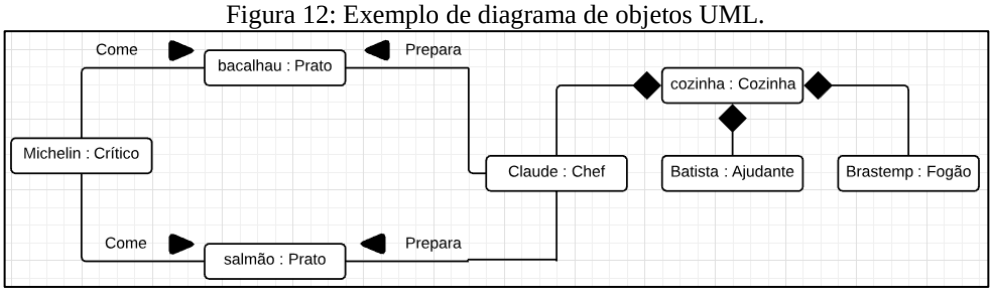


Fonte: Autor.

2.2.3 Diagrama de objetos

Os diagramas de objetos são similares aos diagramas de classes, mas são mais específicos uma vez que mostram a fotografia de um sistema em execução, modelando seus objetos e *links*, que são os relacionamentos entre os objetos (DEITEL e DEITEL, 2001). Cada retângulo indica um objeto de uma classe específica, e as linhas indicam composição ou relacionamentos entre os objetos, da mesma forma como é feito em um diagrama de classes. Partindo do diagrama exposto na figura anterior, é possível modelar este sistema mais especificamente como uma cozinha composta pelo chef Claude, seu ajudante Batista e um fogão Brastemp. O chef prepara dois pratos para o crítico Michelin: bacalhau e salmão. O diagrama que ilustra este exemplo é mostrado abaixo.

Figura 12: Exemplo de diagrama de objetos UML.



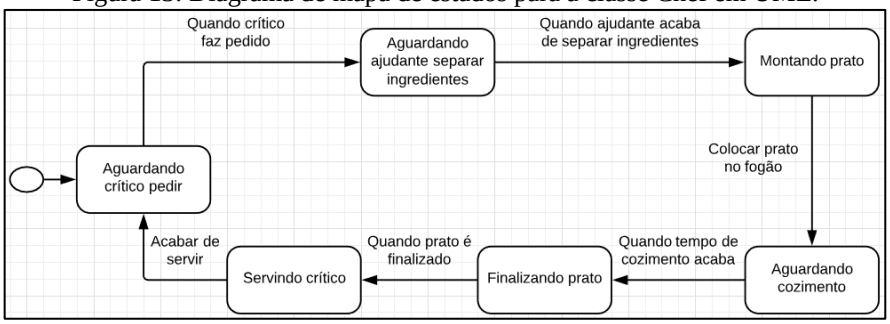
Fonte: Autor.

2.2.4 Diagrama de mapa de estados

Estados são descrições da condição de um objeto em um determinado instante de tempo. Um mesmo objeto pode possuir diversos estados, e diagramas de mapas de estado indicam os possíveis estados para um determinado objeto e mapeiam as ações que levam um objeto a mudar

seu estado. Tais diagramas são formados por retângulos, dentro dos quais estão indicados os estados e setas, que indicam transição de estados. Junto dessas setas há um texto que indica a ação necessária para mudança de estado. Por fim, um círculo indica o estado inicial do objeto (DEITEL e DEITEL, 2001). Para exemplificação, abaixo está exposto o mapa de estados para a classe Chef discutida anteriormente. O chef começa aguardando o crítico fazer um pedido. Quando o pedido é feito, o chef passa a aguardar o ajudante acabar de separar ingredientes. Quando a separação acaba, o chef se concentra em montar o prato. Quando ele coloca o prato no fogão, fica aguardando o tempo de cozimento acabar. Assim que acaba o cozimento o chef finaliza o prato. Com o prato finalizado o chef o serve ao crítico e volta a aguardar um novo pedido.

Figura 13: Diagrama de mapa de estados para a classe Chef em UML.

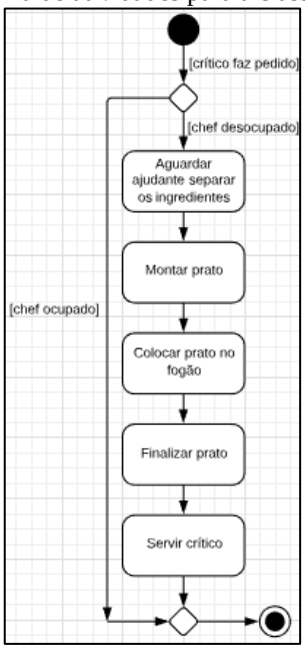


Fonte: Autor.

2.2.5 Diagrama de atividades

O diagrama de atividades nada mais é que uma variação do diagrama de mapa de estados. Enquanto o anterior possui seu foco nos estados da classe, o diagrama de atividades foca em suas ações. As atividades são representadas dentro de retângulos interligados por setas que indicam a sequência das ações. Um texto em colchetes junto a algumas setas indica qual condição deve ser verdadeira para que tal caminho seja percorrido. Um pequeno losango indica quando há coincidência de caminhos. Um círculo indica o início das atividades e um círculo envolvido por outro círculo, mais conhecido como “olho de boi”, indica o término do conjunto de atividades (DEITEL e DEITEL, 2001). A figura abaixo ilustra um diagrama de atividades desenvolvido a partir do diagrama de mapa de estados da seção anterior.

Figura 14: Diagrama de atividades para a classe Chef em UML.

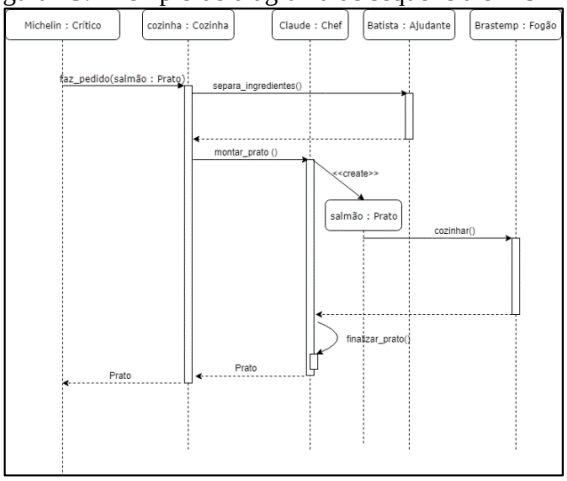


Fonte: Autor.

2.2.6 Diagrama de sequência

Este diagrama tem por objetivo organizar as operações de cada objeto. Por operação entende-se como um serviço que uma classe presta às suas classes clientes. Assim, uma classe cliente invoca uma operação de classe, essa operação é executada pela outra classe e os resultados desta operação são retornados de volta à classe cliente. O diagrama de sequência organiza estas operações em ordem cronológica dentro de um determinado processo. Cada objeto é representado por um retângulo no topo do diagrama. Abaixo de cada objeto há uma linha tracejada, que é a linha de vida do objeto. Esta linha indica a existência ou não do objeto durante a execução. Cada operação de classe é ilustrada por retângulos em cima da linha de vida, o seu comprimento indica a duração temporal da operação. Uma chamada de operação é indicada por uma seta com o nome da função a ser chamada e seu retorno é indicado por uma linha tracejada. Casos condicionais de fluxo de programa aparecem entre colchetes e quando um novo objeto é criado dinamicamente utiliza-se a notação <<*create*>>. Um X no fim da linha de vida indica a destruição de um objeto (DEITEL e DEITEL, 2001). A figura a seguir exemplifica o processo de pedido de prato do crítico.

Figura 15: Exemplo de diagrama de sequência em UML.

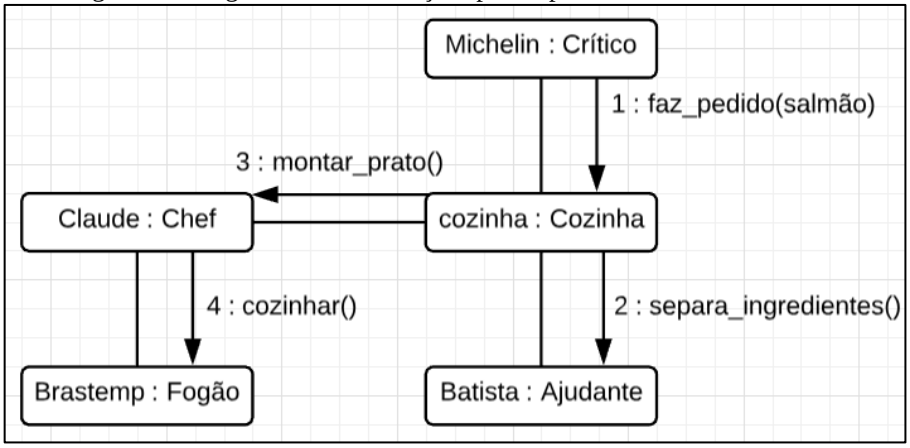


Fonte: Autor.

2.2.7 Diagrama de colaborações

Os diagramas de colaborações indicam as interações entre os objetos. Por interações entende-se como o envio de uma mensagem particular de um objeto de uma classe para um objeto de outra classe. Desta forma, enquanto o diagrama de sequências dá ênfase em quando as interações ocorrem, os diagramas de colaborações indicam quais objetos participam da interação. Neste diagrama, os objetos são representados mais uma vez em retângulos, e os objetos que interagem entre si são conectados com uma linha. Uma seta indica o sentido da interação, e um texto junto à seta indica o nome da função responsável pela interação. Um numeral junto ao nome da função indica a sequência em que as interações ocorrem (DEITEL e DEITEL, 2001). O diagrama de colaborações a seguir é derivado do diagrama de sequências apresentado anteriormente.

Figura 16: Diagrama de colaborações para o pedido do crítico em UML.



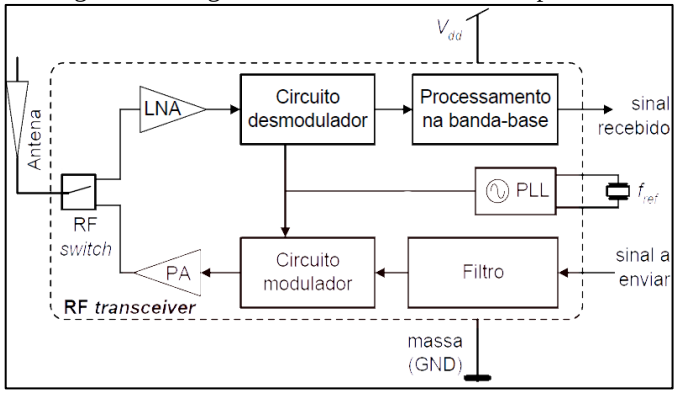
Fonte: Autor.

2.3 Transceptores *wireless* RF

Esta seção irá abordar os conceitos e funcionamento de transceptores *wireless* RF. Radiofrequência é definida pela ANATEL como a faixa do espectro eletromagnético que cobre de 8,3 kHz a 3000 GHz, onde é possível a radiocomunicação. Desta forma, o espectro é um recurso limitado e administrado pela ANATEL no Brasil, que divide e detalha o uso de faixas de radiofrequências para os diversos serviços e atividades de telecomunicações (ANATEL, 2015). A distribuição das faixas de radiofrequências para as mais diversas atividades de telecomunicações pode ser encontrada em (ANATEL, 2018). O transceptor *wireless* RF utilizado neste trabalho é definido pela ANATEL como um equipamento de radiocomunicação de radiação restrita, que são dispositivos que utilizam radiofrequência para aplicações diversas e são regulados pela Resolução n°680 de 27 de junho de 2017 (ANATEL, 2018).

A estrutura interna dos transceptores RF encapsulados em *chips* pode ser genericamente esquematizada pela figura abaixo.

Figura 17: Diagrama de blocos de um transceptor RF.



Fonte: Adaptado de (CARMO e CORREIA, 2010).

A parte superior do diagrama anterior contém os blocos do receptor RF e a parte inferior contém os blocos do transmissor RF.

O receptor RF funciona da seguinte maneira (CARMO e CORREIA, 2010):

1. O sinal que chega pela antena passa pelo RF *switch*, que realiza o chaveamento entre emissão e recepção de sinais.
2. O sinal chega ao LNA, ou *Low Noise Amplifier*, que tem por função amplificar o sinal vindo da antena sem degradar a relação sinal-ruído e sem distorcer significativamente o sinal à saída.

3. O sinal vindo do LNA passa pelo circuito demodulador, que remove a frequência da onda portadora do sinal RF, deixando passar à sua saída apenas o sinal de dados recebido em sua banda-base.
4. Por fim, o sinal demodulado passa por um último estágio que realiza processamento na banda-base, além de possuir buffers que isolam eletricamente o *chip* no caso de ausência de sinal a ser recebido e garantem o casamento de impedância com as cargas que irão operar com o sinal recebido.

Já a parte do transmissor de sinais RF opera da seguinte maneira (CARMO e CORREIA, 2010):

1. Primeiramente, o sinal a ser enviado passa por um filtro para condicionamento do sinal.
2. Em seguida, um circuito modulador mistura a onda portadora com os dados a serem transmitidos. A frequência da portadora é gerada através do componente PLL, *Phase-Locked Loop*, que nada mais é que um sistema multiplicador de uma frequência de referência, realimentada negativamente, gerando uma frequência mais alta e estável em sua saída.
3. Finalmente, um PA, ou *Power Amplifier*, amplifica a potência do sinal RF para um nível adequado a ser transmitido pela antena.

Os transceptores *wireless* RF implementados em *chips* tem se tornado cada vez mais populares devido ao seu baixo custo e tamanho reduzido, sendo utilizados para as mais diversas aplicações como sistemas de telemetria (aplicação deste trabalho), sistemas biomédicos e sistemas de identificação via radiofrequência. Cada uma destas aplicações exige diferentes rejeições a ruído, potências e frequências de transmissão.

2.4 Sistema de navegação global por satélite

A tecnologia de sistemas de navegação global por satélites, ou GNSS, permite obter a posição e tempo em qualquer ponto no planeta com precisão no intervalo de 20m a 1mm no caso da posição e de 60ns a 5ns no caso do tempo (U-BLOX AG, 2009).

O GPS foi o primeiro dos sistemas GNSS a serem implementados e foi desenvolvido pelo governo norte-americano para fins civis e militares. O sistema conta com 32 satélites operacionais e cada um deles possui a bordo quatro relógios atômicos. Suas órbitas estão inclinadas 55° em relação à linha do equador, garantindo que em qualquer ponto do planeta haja pelo menos quatro satélites capazes de estabelecer radiocomunicação. Entre outros

sistemas que merecem destaque é possível citar o GALILEO da União Europeia e o GLONASS da Rússia (U-BLOX AG, 2009).

O princípio de funcionamento do sistema GPS se baseia em medir o tempo de propagação da onda eletromagnética do satélite até o ponto dado no planeta através da Equação 1 abaixo.

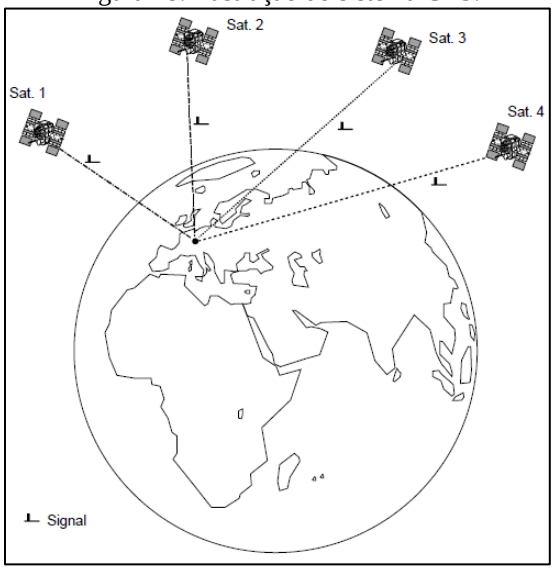
$$D = \Delta t \cdot c \quad (1)$$

Em que D é a distância do satélite até o ponto o qual se deseja saber a posição, Δt é o tempo que leva para a onda alcançar o receptor e c é a velocidade da onda eletromagnética, que vale aproximadamente $3{,}0 \cdot 10^8$m/s. Se a posição do satélite e o tempo de partida da onda forem conhecidos, a equação 1 é capaz de fornecer a posição desconhecida de um ponto em uma dimensão através da aferição do tempo de chegada da onda eletromagnética no receptor. Este sistema apresenta uma falha: um erro de sincronismo de apenas $1\mu s$ entre os relógios do satélite e do receptor resulta em um erro de 300m. Como seria inviável utilizar um relógio atômico no receptor, o erro de sincronismo pode ser minimizado através da adição de um segundo satélite com posição conhecida. Este sistema proposto funciona para um caso de espaço unidimensional, mas é possível extrapolar este raciocínio para o caso tridimensional. Assim, as incógnitas seriam:

- Longitude,
- Latitude,
- Altitude e
- Erro de sincronismo de tempo.

Por causa da existência destas quatro incógnitas é que é necessário que os sistemas GNSS possuam pelo menos quatro satélites visíveis em qualquer posição do globo, pois um sistema com quatro variáveis requer pelo menos quatro equações para serem resolvidos. Este sistema possibilitou que não fossem necessários relógios atômicos em cada receptor, o que ajudou a popularizar e baratear a tecnologia (U-BLOX AG, 2009). Uma ilustração do sistema é exposta a seguir.

Figura 18: Ilustração do sistema GPS.



Fonte: (U-BLOX AG, 2009).

Quanto ao protocolo de saída de dados dos receptores GPS, há o protocolo NMEA 0183, que é bastante utilizado em todo o mundo por diferentes receptores. Os dados são recebidos via caracteres ASCII, o que facilita seu entendimento através de um simples monitor serial na saída, podendo ser diretamente interpretada pelo ser humano. Cada informação transmitida se inicia com um '$' seguido da identificação da mensagem, como 'GPRMC' indicando, por exemplo, uma mensagem GPS com as informações mínimas recomendadas. Um exemplo de saída do protocolo NMEA é dado a seguir.

Figura 19: Saída de um terminal com protocolo NMEA de recepção GPS.

```
$GPGSA,A,3,10,07,05,02,29,04,08,13,,,,,1.72,1.03,1.38*0A
$GPGSV,3,1,11,10,63,137,17,07,61,098,15,05,59,290,20,08,54,157,30*70
$GPGSV,3,2,11,02,39,223,16,13,28,070,17,26,23,252,,04,14,186,15*77
$GPGSV,3,3,11,29,09,301,24,16,09,020,,36,,,*76
$GPRMC,092751.000,A,5321.6802,N,00630.3371,W,0.06,31.66,280511,,,A*45
```

Fonte: (NMEA 0183, 2017).

A utilização do protocolo NMEA em sistemas microprocessados não é tão otimizada, uma vez que é necessária a manipulação de *strings*. Isso fez com que diversos fabricantes de receptores desenvolvessem seus próprios protocolos binários de saída, de forma a passar também alguns dados e métricas que não são disponíveis via protocolo NMEA.

3 DESENVOLVIMENTO

Este capítulo apresenta o desenvolvimento do trabalho, que foi dividido em quatro seções: *Hardware*, Modelagem de *software*, Transceptor RF e Receptor GPS.

O trabalho exposto nesse capítulo contempla o desenvolvimento apenas das rotinas de voo do VANT, pois a parte de controle de pulverização será desenvolvida em um momento posterior quando o voo da aeronave estiver estável.

3.1 *Hardware*

Nesta seção será detalhado o desenvolvimento do *hardware* da placa controladora de voo, cobrindo os seguintes aspectos: coleta de requisitos, escolha da unidade de processamento, escolha dos periféricos e por fim o projeto e manufatura da PCB.

3.1.1 Coleta de requisitos

O projeto do *hardware* iniciou-se com a coleta de requisitos do sistema.

Inicialmente, o requisito mais forte levantado foi o baixo custo dos componentes a serem utilizados, por se tratar de uma prova de conceito. Em segundo lugar, a agilidade no desenvolvimento é bastante relevante, uma vez que há a urgência de lançar o primeiro produto no mercado. Outro ponto que surgiu durante o levantamento de requisitos é que o projeto deveria ser facilmente customizável, possibilitando sua evolução ao longo do tempo. Como última exigência, era necessária uma boa documentação do sistema, facilitando sua manutenção no futuro.

Feito esse levantamento inicial de requisitos, foi realizada a enumeração de todos os componentes de *hardware* que o sistema deveria possuir para cumprir com as especificações do projeto. Foi construída assim a Tabela 1, que traz a relação de cada componente e sua função.

Tabela 1: Relação dos componentes de *hardware* e sua função no sistema.

Componente	Função
Unidade de Processamento	Gerenciar e executar tarefas
Transceptor RF	Manter comunicação entre controladora e GCS
Rádio Controlador	Permitir intervenção do usuário no voo
Leitor de cartão SD	Salvar os dados da aeronave *on-board*
IMU (Acelerômetro, Giroscópio e Magnetômetro)	Mensurar a atitude da aeronave
Receptor GPS	Fornecer as coordenadas da aeronave
Barômetro	Fornecer a altitude do veículo
Sensor de Vazão	Medir a vazão de defensivos a serem aplicados
Sensor de Rotação do Motor	Medir a rotação no eixo do motor a combustão
Sensor de Distância do Solo	Medir a altura da aeronave em relação ao solo
Leitor de Tensão da Bateria	Monitorar a tensão da bateria
Sensor de Nível para Líquidos	Monitorar a quantidade de combustível e defensivos
Potenciômetros	Fornecer a posição angular de sistemas mecânicos
Leitor de Termopares	Medir a temperatura em pontos críticos
Conjunto de Relés	Realizar acionamento de cargas e atuadores
Atuador para embreagem	Permitir acoplamento ou não do motor aos rotores
Servos motores	Utilizado no acelerador e *swashplate*

As subseções a seguir irão detalhar o processo de escolha tanto da unidade de processamento quanto dos demais componentes.

3.1.2 Escolha da unidade de processamento

Uma vez definida a estrutura do sistema, foi possível realizar a escolha da unidade de processamento.

Dentre as prováveis opções de arquiteturas, optou-se por utilizar microcontroladores, devido à sua facilidade de programação e à experiência do autor, que trabalhou com microcontroladores durante toda a sua graduação. Assim, foram descartadas as alternativas de uso de FPGAs e DSPs.

Além da decisão de utilizar um microcontrolador, optou-se por utilizar algum microcontrolador que possuísse placa de desenvolvimento no mercado para facilitar o trabalho de prototipagem de *hardware*. Após uma prospecção inicial, foram eleitas as três opções de placas de desenvolvimento mais populares atualmente: Arduino Mega 2560, Raspberry Pi 3 e Beaglebone Black Rev. C. Para auxiliar na escolha da melhor plataforma, foi elaborada a tabela a seguir que lista os elementos mais relevantes para a elaboração do projeto.

Tabela 2: Quadro comparativo entre placas de desenvolvimento.

Parâmetro	Arduino Mega 2560	Raspberry Pi 3	Beaglebone Black Rev. C
Preço (R$)	50,00	150,00	400,00
Processador	Atmel Atmega2560 8 bits	4x ARM Cortex A-53	AM335x ARM Cortex-A8
Clock (MHz)	16	1200	1000
Memória de programa (Bytes)	256 K	Cartão SD	4 G
Memória de dados (Bytes)	8 K	1 G	512 M
Pinos I/O	54 (5V)	40 (3.3V)	65 (3.3V)
Entradas Analógicas	16(Máx. 5V, 10bits)	0	7 (Máx. 1,8V)
I2C	1	1	1
SPI	1	2	2
CAN	0	0	1
UART	4	2	4
USB	1	2	1
Leitor de SD	0	1	1
Saídas PWM	15	2	8
Timers	2x 8bits, 4x 16bits	4x 32bits, 1x 64bits	4
Interrupção Externa	6	3	2

Fonte: (Arduino MEGA 2560 & Genuino MEGA 2560), (RASPBERRY PI 3 IS OUT NOW! SPECS, BENCHMARKS & MORE, 2016), (Beagleboard:BeagleBoneBlack, 2017).

Dentre as opções analisadas, as que possuem melhor performance computacional são a Raspberry Pi 3 e a Beaglebone Black. Entretanto, tais opções possuem diversos outros recursos, como GPU e Ethernet, que não seriam utilizados em um primeiro momento no projeto, mas que impactam negativamente no preço. Apesar da frequência de *clock* bem menor, contaram como vantagens para o Arduino Mega três fatores: o baixo custo, a experiência anterior do autor com a plataforma e a existência do projeto APM do Ardupilot, que foi um dos primeiros projetos da comunidade e que era baseada no mesmo processador utilizado no Arduino Mega (Archived:APM 2.5 and 2.6 Overview, 2016), o ATmega 2560. Além disso, essa placa de desenvolvimento possui todos os recursos básicos necessários para o projeto, como entradas analógicas, principais barramentos de comunicação, saídas PWM e boa quantidade de pinos digitais de entrada e saída, fazendo com que fosse escolhida como plataforma inicial para o desenvolvimento do projeto.

3.1.3 Escolha dos componentes periféricos

Com a escolha da placa de desenvolvimento definida, foi possível passar a decidir sobre os demais componentes de *hardware*.

O transceptor RF escolhido para o projeto foi o RFD 900+ da fabricante australiana RFDesign. Esta escolha se justificou por esse modelo ser desenvolvido e recomendado pela comunidade Ardupilot, apresentando algumas vantagens extras que serão expostas na seção

mais adequada deste capítulo. O autor já teve experiência com o transceptor XBee 900HP, porém a sua potência de 250mW não seria capaz de fornecer um alcance satisfatório para o *drone*, mesmo em campo aberto.

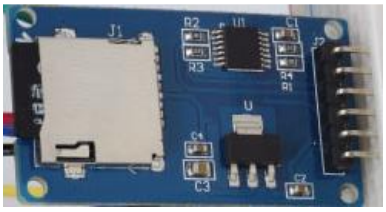
Já o rádio controlador utilizado no projeto foi o Turnigy 9X, que foi doado para a empresa por um dos seus membros. Este é um modelo de entrada de rádios para aeromodelismo com bom alcance e confiabilidade, satisfazendo as necessidades do início do desenvolvimento. O receptor deste rádio possui nove canais em modulação PWM. Para injetar esse sinal de saída na controladora, utilizou-se um Arduino Nano como conversor dos canais PWM para o protocolo UART. A utilização deste coprocessador se justifica pelo fato da leitura de canais PWM ser uma operação muito custosa computacionalmente. Assim, livrou-se a controladora de uma operação crítica que tomaria muito tempo de processamento durante a execução do voo.

Figura 20: Rádio Controlador Turnigy 9X.

Fonte: (TURNIGY 9X REVIEW – 9 CHANNEL RC TRANSMITTER, 2013).

Como leitor de cartão SD, foi utilizada uma *shield* para prototipagem comercial, que possui *slot* para microSD, regulador de tensão de 3,3V para alimentar o cartão e um conversor de níveis lógicos que permite a comunicação com dispositivos de nível lógico 5V, como é o caso do Arduino Mega utilizado. A seguir é apresentada uma fotografia da *shield* utilizada.

Figura 21: *Shield* leitora de cartão microSD.

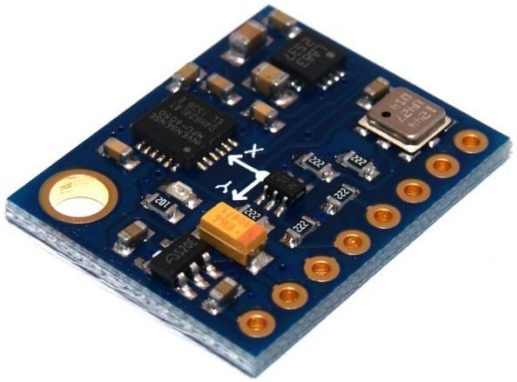


Fonte: Autor.

Optou-se por utilizar o receptor GPS here fabricado pela ProfiCNC. Este receptor utiliza o *chip* Neo-M8 da fabricante u-blox. Esse componente será apresentado com maiores detalhes em uma seção posterior dentro desse mesmo capítulo.

O conjunto formado pelo IMU e barômetro foram adquiridos na forma da *shield* de prototipagem GY-87, que está ilustrada abaixo.

Figura 22: *Shield* GY-87 com IMU e barômetro.



Fonte: (MÓDULO MEDIÇÃO INERCIAL GY-87, 2018).

Essa *shield* contém os *chips*:

- Invensense MPU6050: Acelerômetro e giroscópio nos três eixos,
- Honeywell HMC5883L: Magnetômetro nos três eixos e
- Bosch BMP180: Sensor de pressão digital.

Todos esses componentes possuem sua saída fornecida através de barramento I2C, o que facilita a leitura das variáveis de interesse.

O sensor de vazão utilizado no projeto inicialmente é o YF-S201 da figura abaixo, sensor hall que produz na saída uma onda quadrada com frequência proporcional à vazão. Será utilizada a interrupção externa da placa de desenvolvimento para medir a frequência da onda gerada e obter a vazão.

Figura 23: Sensor de vazão utilizado.

Fonte: Autor.

O sensor de rotação escolhido foi o sensor indutivo Metaltex I12-4-DNC-K12. Ele possui distância sensora de 4mm, o que permite uma boa distância entre ele e a peça que gira junto do eixo. Esse sensor possui saída digital e pode ser lido através da interrupção externa do microprocessador.

Figura 24: Sensor de rotação selecionado para o projeto.

Fonte: Autor.

Para realizar a leitura da tensão da bateria foi utilizado um circuito analógico de condicionamento de sinal, permitindo que o processador com entrada de tensão limitada em 5V faça a leitura de uma bateria com tensão nominal 12V.

Por fim, para realizar a leitura de temperatura via termopares, foi empregada uma *shield* que possui o *chip* MAX6675, que faz todo o condicionamento de sinal do termopar e fornece a leitura via protocolo SPI.

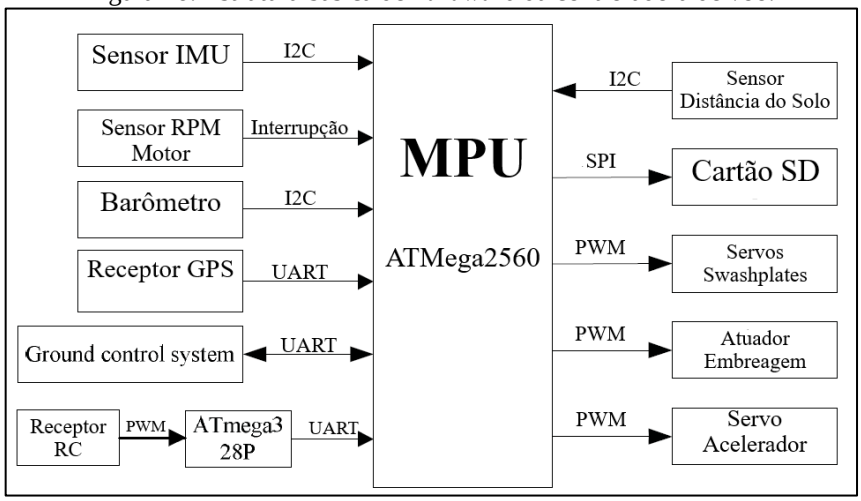
Figura 25: *Shield* com o leitor de termopares MAX6675.

Fonte: Autor.

Por motivos diversos, os componentes a seguir ainda não haviam sido definidos quando este trabalho foi entregue: sensor de distância do solo, sensor de nível para líquidos, potenciômetros, conjunto de relés, atuador para embreagem e servos motores.

A estrutura básica de *hardware* que permite os primeiros testes de voo com a controladora é apresentada em seguida.

Figura 26: Estrutura básica de *hardware* da controladora de voo.



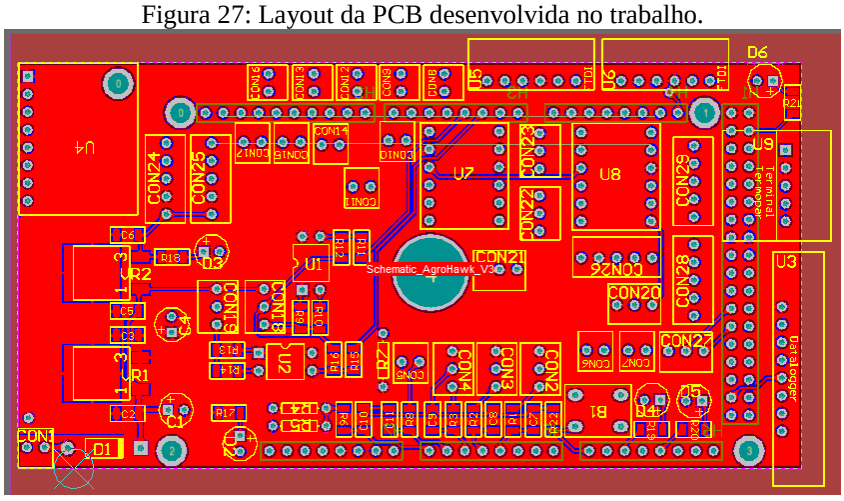
Fonte: Autor.

3.1.4 Projeto e manufatura da PCB

Definidos todos os componentes de *hardware*, foi realizado o projeto e manufatura de uma *shield* a ser encaixada no Arduino Mega para embarcar todos os demais componentes do sistema.

O esquemático e o *layout* da PCB foram projetados em software CAD adequado. Foram utilizados um plano com alimentação 5V e um plano aterrado em cada face da placa, facilitando o roteamento dos componentes. Empregou-se a utilização de reguladores de tensão 5V e 3,3V de forma a não impedir a utilização de dispositivos de baixa potência no projeto. Foram utilizados também alguns componentes SMD de forma a otimizar a utilização do espaço na placa. O *layout* desenvolvido está exposto na figura a seguir.

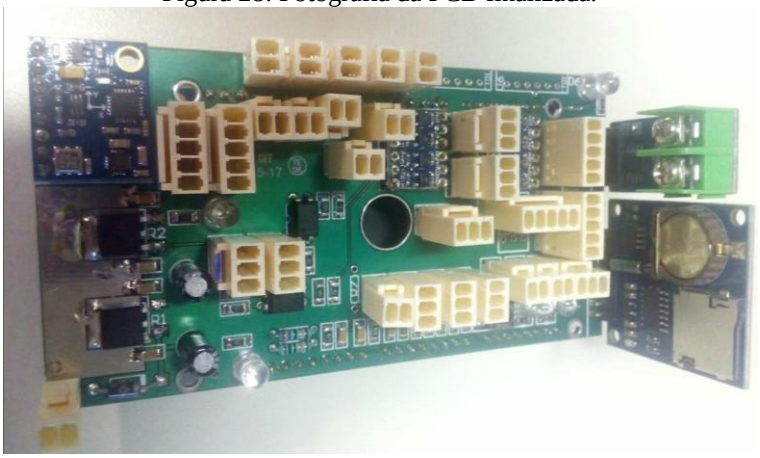
Figura 27: Layout da PCB desenvolvida no trabalho.



Fonte: Autor.

A *shield* foi manufaturada em empresa profissional de fabricação de placas de circuito impresso e soldada pelo autor. Abaixo é apresentada uma fotografia da PCB finalizada.

Figura 28: Fotografia da PCB finalizada.



Fonte: Autor.

3.2 Modelagem de *software*

Estando finalizada a fabricação do *hardware* da controladora, deu-se início ao processo de modelagem de *software*, que englobou a escolha da linguagem de programação e ferramentas de trabalho, implementação da arquitetura de *software*, modelagem da camada de aplicação e elaboração de diagramas em UML.

3.2.1 Escolha da linguagem de programação e ferramentas de trabalho

A linguagem de programação empregada no projeto foi C++. A utilização de uma linguagem orientada a objetos traz algumas vantagens interessantes como maior nível de abstração, melhor encapsulamento de dados e maior facilidade no reuso de código-fonte. Além disso, o autor já havia trabalhado com a linguagem e se sentia bastante confortável em relação a ela. Pesou também para a sua utilização o fato de que quase todo o material *open-source* disponível para a plataforma utilizada é escrito nessa linguagem, facilitando o desenvolvimento do *software* uma vez que há uma boa base de informações disponíveis. Os projetos da comunidade Ardupilot também foram desenvolvidos nessa linguagem e serviram diversas vezes como parâmetro e suporte para o desenvolvimento do código-fonte do trabalho.

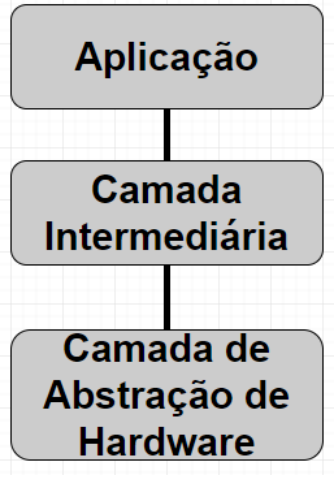
Uma vez definida a linguagem de programação, concentrou-se em escolher ferramentas de software adequadas para o desenvolvimento do trabalho. Como compilador, utilizou-se o

compilador nativo de C++ para plataformas AVR instalado junto da IDE disponibilizada livremente para projetos na plataforma Arduino, possibilitando a utilização da camada de abstração de *hardware* já toda implementada. Entretanto optou-se pela não utilização dessa IDE do Arduino devido aos seus recursos limitados de edição de código-fonte e organização de arquivos. Em seu lugar, foi utilizada a IDE Atmel Studio 7, que é distribuída pela Atmel para projetos utilizando os seus microprocessadores. Essa IDE é baseada no Microsoft Visual Studio, possui ferramentas avançadas de edição de código-fonte e permite uma boa organização de arquivos para elaboração de um projeto de *software* complexo.

3.2.2 Implementação de arquitetura de *software*

Uma preocupação latente no desenvolvimento do *software* era uma organização que permitisse adotar microprocessadores mais avançados a medida que o projeto amadurecesse sem que fosse necessário reescrever todo o código-fonte novamente. Assim, foi proposta a divisão do *software* em três camadas, como exposto a seguir.

Figura 29: Arquitetura de *software* proposta no trabalho.



Fonte: Autor.

Essa divisão de camadas permitiu uma melhor organização do sistema e facilitará a migração para processadores mais avançados no futuro.

A camada inferior é a de abstração de *hardware* que foi aproveitada da IDE do Arduino. Essa camada é a que possui menor nível de abstração e faz a conexão entre o *hardware* do processador e o *software* a ser executado, criando as definições básicas do programa através da configuração dos registradores, memória de dados e memória de programa. Como a maior parte das placas de desenvolvimento já possuem a camada de abstração de *hardware* implementada

em suas IDEs, o trabalho de migração para outras plataformas é facilitado. Já que essa camada não requer mudanças durante o desenvolvimento do projeto, ela é pré-compilada e referenciada no projeto através de um arquivo de saída, diminuindo o tempo de compilação do projeto.

Já a camada intermediária utiliza serviços da camada de abstração de *hardware* para fazer a ligação entre a camada inferior e superior do sistema. Assim, ela fornece serviços básicos a camada de aplicação como operação de *timers* e protocolos de comunicação implementados em *hardware*. Em muitas plataformas de desenvolvimento boa parte dessa camada também se encontra disponibilizada.

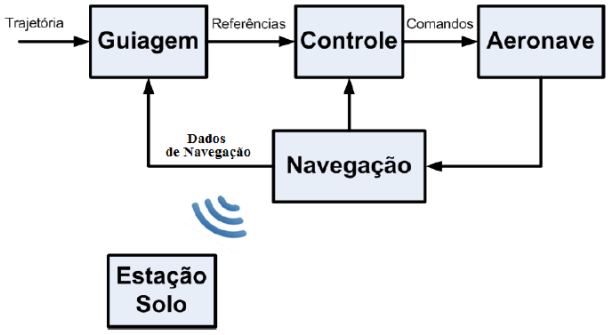
Por fim, a camada de aplicação é a que possui maior nível de abstração e é responsável por implementar a execução de tarefas em nível de usuário. Nessa camada se encontra o escalonamento das tarefas do microprocessador, as classes principais e a implementação de protocolos de comunicação em *software*.

Com essa arquitetura de *software* é possível realizar a portabilidade do sistema para outras plataformas de forma simples, sem necessidade de modificações na camada de aplicação. A camada de abstração de *hardware* deverá ser trocada pela camada do novo processador, que geralmente encontra-se disponibilizada. Assim, os ajustes devem ser feitos apenas na camada intermediária para garantir a ligação entre as camadas superior e inferior.

3.2.3 Modelagem da camada de aplicação

Com a arquitetura de *software* definida, foi realizada a modelagem da camada de aplicação. A estrutura da camada de aplicação desse trabalho baseou-se no sistema de piloto automático proposto em (BITTAR, 2012) e exposto abaixo.

Figura 30: Diagrama de blocos de um sistema de pilotagem automática.



Fonte: (BITTAR, 2012).

Cada um dos blocos propostos por (BITTAR, 2012) e sua correlação com a estrutura desenvolvida no trabalho são:

- Aeronave: Representa fisicamente a aeronave. Não corresponde a uma classe em *software*.
- Estação Solo: Também chamada de GCS, corresponde à interface gráfica do operador. Este aplicativo para computadores pessoais não será desenvolvido pela empresa, pois será utilizado o aplicativo *open-source* QGroundControl, que faz uso do protocolo de comunicação MAVLink, ou *Micro Air Vehicle Link*. Esse protocolo também se encontra disponível livremente e será implementado na aeronave. Assim, foi criada a classe Interface, que abrange todo nível de comunicação direta com o usuário. Ela fica responsável pela comunicação com a GCS através do protocolo MAVLink, recebimento dos dados do rádio controlador e pelo armazenamento de informações no cartão SD.
- Navegação: É formado pelo conjunto de sensores utilizados durante o voo. Neste trabalho foi criada a classe Sensores para representar esse subsistema, que deve ser responsável pela configuração e aquisição de dados dos sensores, bem como pela implementação de um filtro estimador que realize a fusão de dados.
- Guiagem: Consiste do sistema que gera os valores de referência para a navegação e estabilidade da aeronave. No código-fonte foi representada pela classe Controle. Essa classe recebe os sinais de trajetória recebidos do usuário através de *waypoints* e os compara com os dados dos sensores para atualizar as malhas de controle que garantem a estabilidade e a navegação correta da aeronave.
- Controle: Sistema que gera os sinais de comando para os atuadores. Esse subsistema é representado em *software* através da classe Atuadores cuja função é gerar os sinais elétricos para atuação de motores e outras cargas que interferem no movimento da aeronave.

As principais classes implementadas no código-fonte e suas respectivas funções estão resumidas na tabela a seguir.

Tabela 3: Relação das classes e tarefas da aplicação.

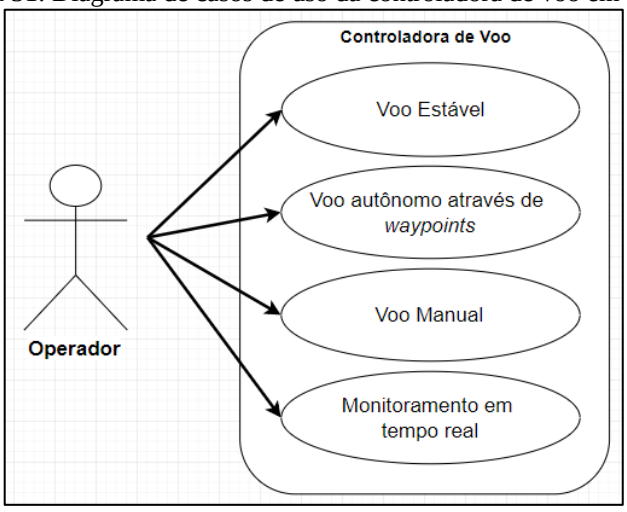
Classes	Tarefas
Interface	Gerenciar comunicação com GCS via MAVLink
	Armazenamento de informações no cartão SD
	Comunicação com o rádio controlador
Sensores	Implementação do filtro estimador
	Leitura e configuração de sensores
Controle	Interpretar comandos do usuário
	Monitorar falhas do sistema
	Gerenciar malhas de controle
Atuadores	Gerar sinais de excitação para atuadores

3.2.4 Diagramas em UML

Com as principais classes da camada de aplicação idealizadas, iniciou-se a construção de diagramas em UML através da ferramenta *online* para confecção de diagramas draw.

O primeiro diagrama construído foi o de casos de uso. Como representado abaixo, o cliente da controladora de voo é o operador do VANT, que espera obter da placa um voo estável, possibilidade de programação de *waypoints* para voo autônomo, voo manual e monitoramento de dados em tempo real.

Figura 31: Diagrama de casos de uso da controladora de voo em UML.



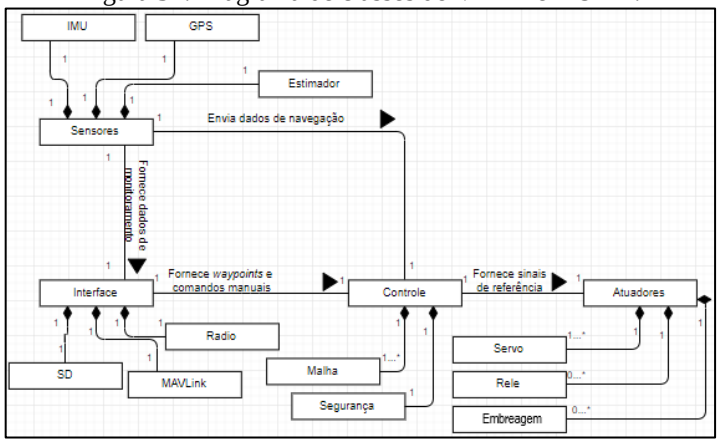
Fonte: Autor.

Em seguida foi esquematizado o diagrama de classes, na qual se omitiu os atributos e funções das mesmas. Neste diagrama foram representadas as quatro classes principais, a saber, Interface, Controle, Sensores e Atuadores, bem como as principais subclasses que as compõe, como pode ser visto a seguir. Tais subclasses principais são:

- IMU, GPS e Estimador em relação à classe Sensores,

- Radio, SD e MAVLink em relação à classe Interface,
- Segurança, que tem função de checar erros do sistema e Malha, que representa uma malha de controle genérica relativas à classe Controle e
- Servo, Rele e Embreagem que compõem a classe Atuadores.

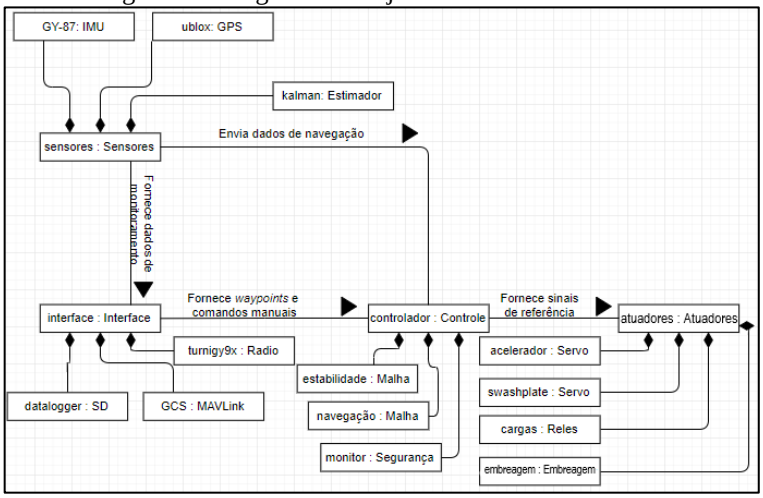
Figura 32: Diagrama de classes do VANT em UML.



Fonte: Autor.

Com o diagrama de classes, foi possível desenhar o diagrama de objetos, que está exposto abaixo.

Figura 33: Diagrama de objetos do VANT em UML.

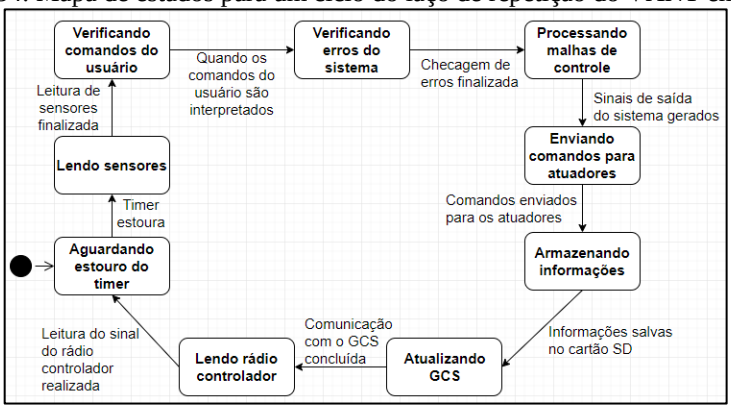


Fonte: Autor.

Com as classes e suas relações definidas pelos diagramas anteriores, foi elaborado um mapa de estados para um ciclo do laço de repetição do programa, que é exibido a seguir. O ciclo começa com o sistema aguardando o estouro do *timer*. Quando o *timer* estoura, inicia-se a leitura de sensores. Com a leitura finalizada, o controlador verifica os comandos do usuário e

os erros do sistema, processa as malhas de controle e por fim envia o sinal aos atuadores. A partir daí a interface armazena as informações no SD, lê os sinais do rádio controlador e atualiza as informações da GCS.

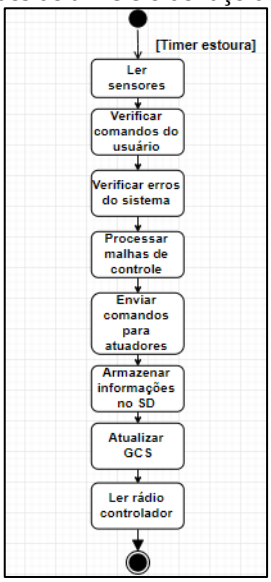
Figura 34: Mapa de estados para um ciclo do laço de repetição do VANT em UML.



Fonte: Autor.

Mudando o foco do mapa de estados para as atividades a serem realizadas durante um ciclo do programa, foi obtido o diagrama de atividades que está ilustrado abaixo.

Figura 35: Diagrama de atividades de um ciclo do laço de repetição do VANT em UML.

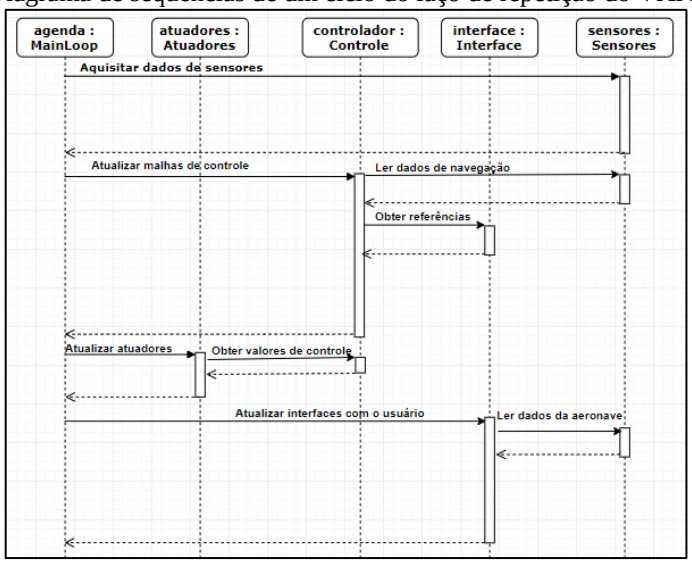


Fonte: Autor.

Foi elaborado em seguida um diagrama de sequências para um ciclo do laço de repetição do programa. Nesse diagrama o laço principal do programa foi representado como um objeto apenas para indicar o gerenciamento e chamada das funções. Primeiramente, o objeto sensores realiza a leitura dos sensores. Em seguida, o controlador é acionado para atualizar as malhas de

controle e para isso precisa coletar os dados do objeto sensores e os comandos do usuário do objeto interface. Depois, o objeto atuadores atualiza seus membros através da leitura dos valores de referência gerados pelo controlador. Por fim, o objeto interface atualiza os meios de comunicação com o usuário e para isso lê os dados dos sensores. O diagrama completo está exposto abaixo.

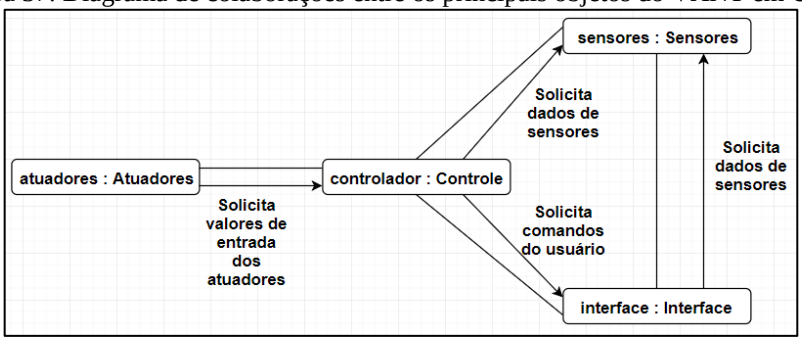
Figura 36: Diagrama de sequências de um ciclo do laço de repetição do VANT em UML.



Fonte: Autor.

Com base no diagrama de sequências foi elaborado um diagrama de colaborações entre os quatro objetos principais do programa exibido abaixo.

Figura 37: Diagrama de colaborações entre os principais objetos do VANT em UML.



Fonte: Autor.

Com base nos diagramas elaborados em UML foi escrito o código-fonte do *software* a ser embarcado na aeronave.

3.3 Transceptor RF

Conforme citado anteriormente neste capítulo, o transceptor RF utilizado nesse trabalho foi o RFD 900+ que está ilustrado na figura abaixo.

Figura 38: Transceptor RF RFD 900+.

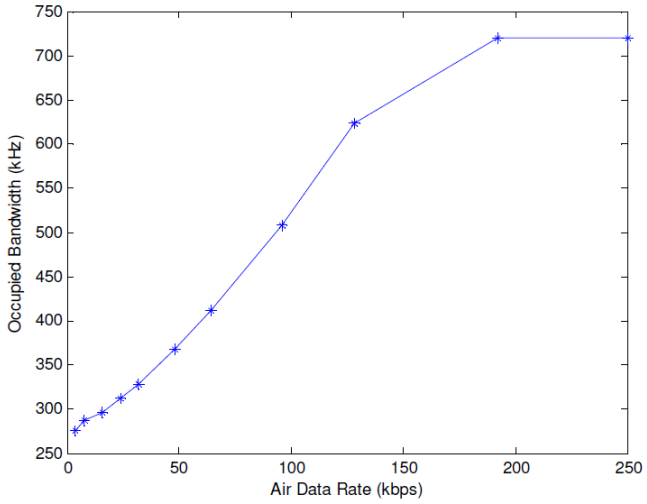
Fonte: (RFD 900+ Modem, 2018).

Esse dispositivo possui como pontos fortes: faixa de frequência de operação customizável entre 902 e 928 MHz, potência de transmissão de até um 1 Watt, taxa de transmissão de dados de até 250kbps, tecnologia de espalhamento espectral e utilização de duas antenas.

Como esse aparelho é importado, ele precisa ser corretamente ajustado para que possa cumprir com os requisitos da legislação brasileira, que o define como um equipamento de radiocomunicação de radiação restrita. Tais requisitos para esse tipo de equipamento se encontram em dois documentos: Regulamento sobre Equipamentos de Radiocomunicação de Radiação Restrita (ANATEL, 2018) e Requisitos técnicos para a avaliação da conformidade de equipamentos de radiocomunicação de radiação restrita (ANATEL, 2017).

A primeira configuração realizada foi restringir a faixa de operação do dispositivo entre 915 e 928 MHz, uma vez que a faixa entre 907,5 e 915 MHz é destinada ao Serviço Móvel Pessoal (SMP) em caráter primário (ANATEL, 2018). Para permitir a operação com transmissão de 1W, a legislação exige pelo menos 35 canais de salto no espalhamento espectral. Além disso, é exigida uma separação mínima entre os canais de salto de 25kHz ou da largura de faixa do canal de salto a 20dB, sendo considerado o valor mais alto. O gráfico a seguir correlaciona a largura de faixa do canal de salto com a taxa de transmissão de dados.

Figura 39: Largura de banda ocupada pelo canal de salto em função da taxa de transmissão de dados no RFD 900+.

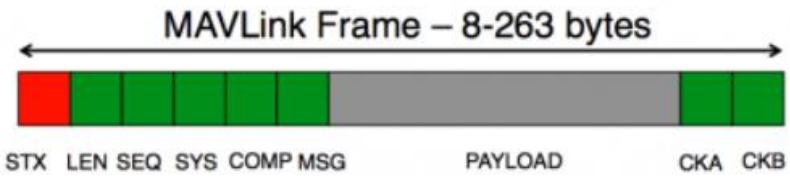


Fonte: (RFDESIGN, 2013).

De posse das informações do gráfico e da quantidade de canais de salto necessária, foram configurados 35 canais de salto. Com isso, obteve-se uma separação de canais de salto de pouco mais de 370kHz, sendo necessário então adotar uma taxa de transmissão de dados de 48kbps. Com esses parâmetros configurados foi possível cumprir com as obrigações da legislação brasileira.

Finalizada a configuração do rádio, foi possível iniciar a implementação do protocolo de comunicação entre VANT e GCS. O protocolo utilizado foi o MAVLink, protocolo *open-source* criado para aplicações em pequenas aeronaves. Foi utilizada a versão 1.0 do protocolo, que possui estrutura e tamanho de pacotes de dados fixos para cada uma das mensagens especificadas. A estrutura de uma mensagem MAVLink 1.0 está exposta abaixo.

Figura 40: Estrutura de uma mensagem do protocolo MAVLink 1.0.



Fonte: (MAVLink Micro Air Vehicle Communication Protocol, 2018).

Cada um dos componentes que formam uma mensagem MAVLink são descritos na tabela a seguir.

Tabela 4: Explicação de cada componente de uma mensagem MAVLink.

Campo	Descrição
STX	É formado pelo *byte* 0xFE e indica o início de uma nova mensagem
LEN	*Byte* que indica o tamanho do PAYLOAD
SEQ	Número de sequência incrementado a cada nova mensagem e usado para detectar perdas
SYS	Identificador do sistema que envia a mensagem, exemplo: GCS ou aeronave
COMP	Identificador de um componente do sistema que envia a mensagem. Exemplo: Rádio do VANT
MSG	Identificador da mensagem, indica que tipo de informações são transportadas
PAYLOAD	Informações propriamente ditas a serem enviadas pela mensagem. Pode conter de 0 a 255 *bytes*
CKA	Dois *bytes* que compõem o *checksum*, utilizado para identificar erros de transmissão e codificação da mensagem. Segue o padrão SAE AS-4
CKB	

O protocolo de comunicação MAVLink comunica-se com o transceptor RF através do protocolo UART. O dispositivo utilizado também possui o MAVLink integrado em seu processador e é capaz de se comunicar com a placa controladora, informando em uma mensagem de *status* a força do sinal local e remoto, porcentagem do preenchimento do *buffer* de transmissão e níveis de ruído local e remoto. A GCS QGroundControl também utiliza o mesmo protocolo, possibilitando a comunicação com o VANT sem grandes dificuldades.

Como subconjuntos de mensagens a serem utilizadas no sistema, optou-se pelo emprego do subconjunto comum do protocolo (MAVLINK Common Message Set) aliado ao subconjunto específico da comunidade Ardupilot (MAVLINK ArduPilotMega Message Set). Além do mais, ainda é possível adicionar mensagens customizadas pelo usuário para atender a necessidades específicas.

3.4 Receptor GPS

O receptor GPS utilizado no trabalho foi o here fabricado pela ProfiCNC, ilustrado na figura a seguir. Esse receptor emprega o *chip* Neo-M8 da u-blox, cujas principais pontos fortes são a alta sensibilidade e suporte para múltiplos sistemas de navegação além do GPS, como o GLONASS, BeiDou e GALILEO.

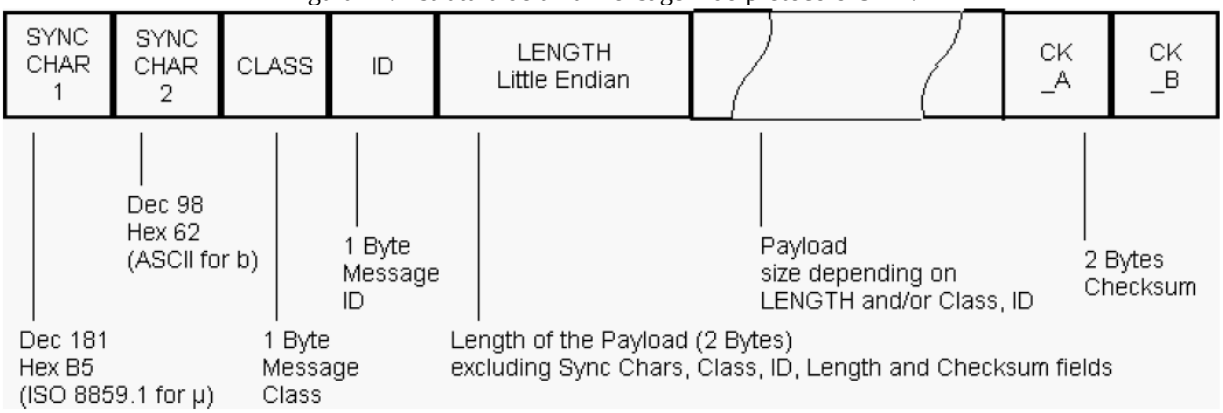
Figura 41: Receptor GPS here da ProfiCNC.

Fonte: (Here GNSS (M8N)).

A possibilidade de utilizar vários sistemas GNSS no mesmo módulo traz consigo a vantagem de possibilitar a fusão de dados e pós-processamento de sistemas de satélite diferentes para se obter uma melhor precisão da leitura de tempo e posição. Nesse trabalho foi empregada a fusão dos sistemas GPS e GLONASS.

O receptor GPS se comunica com a placa controladora através do protocolo de comunicação UART. As mensagens do receptor à controladora são enviadas através do protocolo UBX, propriedade da fabricante u-blox. Esse protocolo possui a seguinte estrutura: dois *bytes* iniciais indicam o início de uma nova mensagem, os dois *bytes* seguintes indicam a que classe de informações essa mensagem pertence e qual o seu identificador. Na sequência aparecem dois *bytes* que indicam o tamanho do vetor de informações. Por fim, é enviado o vetor de informações seguido de dois *bytes* de checksum, que são utilizados para verificar se há algum erro de leitura da mensagem. A estrutura completa é apresentada a seguir.

Figura 42: Estrutura de uma mensagem do protocolo UBX.



Fonte: (U-BLOX, 2017).

Na placa controladora de voo foi implementada uma biblioteca para fazer a leitura desse protocolo. A configuração inicial do *chip* é toda realizada antes da montagem do equipamento através do aplicativo u-center distribuído gratuitamente pela fabricante. A mensagem do protocolo utilizada para navegação é a NAV-PVT, que passa informações básicas sobre posição, velocidade e tempo, como data, hora, tipo de correção que está sendo empregada, latitude, longitude, altitude, diluição da precisão, componentes da velocidade, direção do movimento entre outros. Essa mensagem é enviada a 1 Hz, que é a taxa de atualização padrão do sistema GPS.

Adicionalmente são enviadas as mensagens MON-HW e MON-HW2 a cada cinco segundos. Essas mensagens têm por função apresentar alguns dados a respeito do funcionamento do sistema, como informações da onda portadora do sinal GPS, nível de ruído externo, valor do controle automático de ganho, *status* da antena e indicador de *jamming*.

4 TESTES, RESULTADOS E DISCUSSÕES

Este capítulo aborda os testes realizados com o sistema, expõe os resultados obtidos e apresenta uma breve discussão a respeito dos mesmos. Ele será dividido em três partes: testes de performance de software, teste de alcance do transceptor RF e teste de precisão do receptor GPS. Não será abordado nenhum tipo de teste de hardware, uma vez que foram realizados apenas testes preliminares para certificar o funcionamento do sistema assim que a placa acabou de ser soldada.

4.1 Testes de performance do *software*

Uma das preocupações iniciais durante o desenvolvimento da arquitetura de *software* era se a utilização de uma camada intermediária entre a aplicação e a camada de abstração de *hardware* acarretaria em prejuízos na performance do sistema em termos de tempo de processamento de funções básicas. Desejava-se manter uma boa organização do *software* em objetos sem prejuízo significativo da performance. Resolveu-se então realizar testes utilizando um osciloscópio 2190D da B&K Precision e um dos LEDs da placa controladora para gerar um pulso cujo tempo pudesse ser medido. O LED era aceso antes da chamada da função e apagado depois de finalizada a função. O tempo entre o acender e apagar do LED foi aferido e descontado do tempo total do pulso. Depois da primeira iteração do desenvolvimento da camada intermediária foram realizados testes com algumas funções, primeiramente chamando-as diretamente da camada de abstração de *hardware* e em seguida foram chamadas as mesmas funções através da camada intermediária de *software*. Uma imagem que exemplifica a saída do osciloscópio durante os testes e uma tabela compilando os tempos de processamento obtidas estão expostas a seguir.

Figura 43: Exemplo de saída do osciloscópio para os testes realizados em software.

Fonte: Autor.

Tabela 5: Comparação entre os tempos de processamento de funções em diferentes camadas.

Função	**Tempo de processamento na camada de abstração de hardware (µs)**	**Tempo de processamento na camada intermediária (µs)**
Atribuição de valor a saída digital	2,32	3,36
Inversão de valor da saída digital	8,58	9,48
Leitura de entrada digital	4,38	5,78
Leitura de entrada analógica	112	112,6
Escrita de *byte* na UART	4,44	4,68

A partir das cinco operações básicas descritas na tabela acima, buscou-se otimizá-las para aproximar os tempos de execução da camada intermediária da camada de abstração de hardware. Para isso foram utilizados alguns atributos de C++, como funções *inline*, que evitam o tempo perdido com chamadas de funções aninhadas através da cópia direta de um trecho de código-fonte de função dentro de um setor específico da memória de programa. Foram realizadas também algumas modificações nas classes criadas, permitindo, por exemplo, uma grande economia de tempo na inversão de valor da saída digital com a criação de uma variável que sinaliza o estado da saída, evitando a necessidade de leitura do valor via *hardware* para permitir a inversão do nível lógico de maneira correta. Os valores de tempos de processamento de cada função antes e depois das modificações, bem como a redução porcentual no tempo de processamento estão compilados na tabela a seguir.

Tabela 6: Comparação no tempo de processamento de funções básicas antes e depois das modificações.

Função	**Tempo de processamento na camada intermediária original (µs)**	**Tempo de processamento na camada intermediária após modificações (µs)**	**Redução porcentual no tempo de processamento (%)**
Atribuição de valor a saída digital	3,36	2,46	26,79
Inversão de valor da saída digital	9,48	4,78	49,58
Leitura de entrada digital	5,78	4,38	24,22
Leitura de entrada analógica	112,6	112,6	0
Escrita de *byte* na UART	4,68	4,44	5,13

Através da análise das duas tabelas anteriores, nota-se que houve melhora significativa na performance das funções da camada intermediária após as modificações, aproximando-a daquela obtida com a camada de abstração de hardware.

Um dado que chama a atenção nas tabelas anteriores é o tempo demasiadamente grande que o processador leva para realizar a leitura do conversor analógico/digital. Portanto, será preferível utilizar no projeto sensores com saídas digitais em relação a sensores analógicos. Outra opção seria utilizar um *chip* dedicado, que possua algum conversor mais rápido e forneça a leitura na forma de um protocolo digital, tirando do microprocessador a responsabilidade por uma tarefa que demanda muito tempo de processamento.

Outra observação que chamou a atenção durante o desenvolvimento do trabalho foi a diferença entre os tempos de processamento de equações utilizando variáveis de ponto fixo e ponto flutuante. A operação em ponto flutuante chega a tomar um tempo cinco vezes maior para ser executada. Logo, optou-se no trabalho em utilizar variáveis inteiras sempre que possível, otimizando dessa maneira a utilização do microprocessador.

Devido ao fato das principais classes da camada de aplicação não estarem totalmente implementadas quando da entrega desse trabalho, não foi possível realizar a aferição dos tempos de processamento das funções principais do programa para inferir a respeito do tempo total de execução de um ciclo completo do programa exposto no capítulo anterior.

4.2 Teste de alcance do transceptor RF

O teste de alcance do transceptor RF foi realizado nas dependências do Campus 1 da Escola de Engenharia de São Carlos. O procedimento foi realizado através do envio de mensagens MAVLink da placa controladora de voo para um terminal de recepção em um computador pessoal. Em cada um dos dispositivos foi utilizada uma antena dipolo meia onda.

A placa controladora foi posicionada próxima às dependências do NETeF e o autor foi descendo a rua em sentido à portaria da USP na Av. Trabalhador São-Carlense até que houvesse a perda de comunicação. O mapa mostrando o alcance obtido do aparelho é apresentado abaixo.

Figura 44: Máximo alcance obtido no teste realizado com o transceptor RF.

Fonte: Autor.

A partir da imagem acima, nota-se que foi obtido um alcance de pouco mais de 320 metros. Esperava-se obter um alcance maior devido ao fato de o teste ter sido realizado em linha de visada. Para a operação do VANT é necessário um alcance bem maior. Um fator que pode ter levado a um alcance relativamente baixo foram as antenas utilizadas. A folha de dados do fabricante recomenda a utilização de duas antenas em cada dispositivo para obter sua melhor performance. Uma possibilidade de otimização seria a utilização de uma antena Yagi, que permite o dobro de ganho em relação a antena dipolo meia-onda utilizada.

4.3 Teste de precisão do receptor GPS

O teste de precisão do receptor GPS foi realizado no campo de futebol do campus 1 da EESC. O receptor foi posicionado em um dos vértices do campo de futebol conforme a Figura 45 a seguir.

Figura 45: Fotografia do teste realizado com o receptor GPS.

Fonte: Autor.

Foi desenvolvida uma função no código-fonte da controladora para fornecer uma saída via UART com as informações recebidas em cada mensagem de posição disponibilizada pelo receptor, conforme a figura abaixo.

Figura 46: Saída esquematizada para cada mensagem de posição recebida no teste do GPS.

```
Msg Recebida:
Latitude: -22.0067844 graus
Longitude: -47.8980865 graus
Altitude: 834.166 m
Fix: 3D/DGNSS
Numero de Satelites: 14
Velocidade em relacao ao chao: 0.04 m/s
Diluicao Precisao Horizontal: 0.89
Diluicao Precisao Vertical: 1.08
Nivel de Ruido: 100
Nivel de Jamming: 15
```

Fonte: Autor.

Na saída exposta anteriormente são apresentadas a latitude e longitude em graus decimais, altitude em metros e o *fix* indica o tipo de correção que o receptor está realizando, nesse caso está sendo utilizada correção via fusão de dados do GPS e GLONASS. São apresentados também o número de satélites, a velocidade em relação ao chão e as diluições de precisão horizontal e vertical, que são figuras de mérito que indicam a quantidade de erro associado a cada ponto. Quanto menor esse valor, melhor é a precisão. Por fim, é apresentado os níveis de ruído e *jamming* no núcleo do receptor.

O teste iniciou-se com uma espera de aproximadamente cinco minutos para que o receptor iniciasse as correções utilizando a fusão de dados GPS e GLONASS, obtendo dessa forma seu melhor desempenho. Em seguida, foi salvo um arquivo contendo um minuto de aquisições de dados do receptor. Posteriormente os 60 pontos adquiridos foram utilizados como

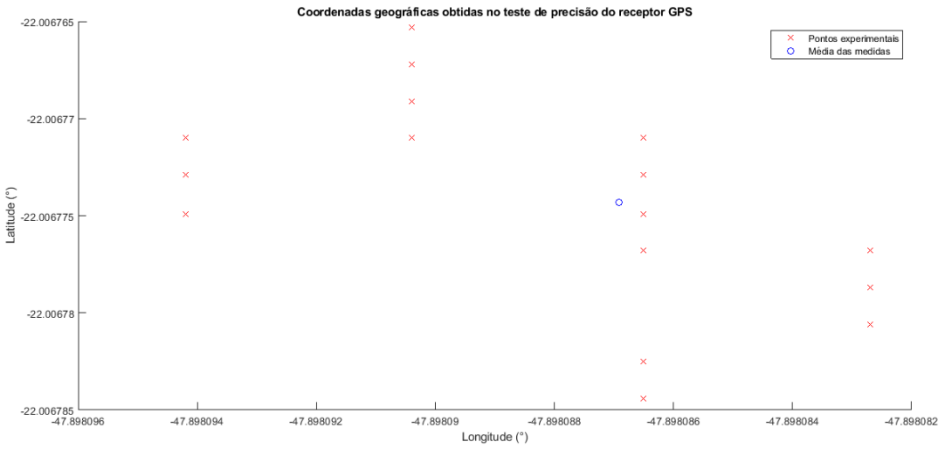
input no aplicativo Google My Maps. O conjunto de coordenadas recebidas pelo dispositivo está representado pelo círculo azul com contorno branco na parte inferior da figura abaixo.

Figura 47: Pontos geográficos obtidos do receptor GPS.

Fonte: Autor.

A primeira observação pertinente é que há um desvio das coordenadas em relação ao exposto no aplicativo utilizado, que pode não ser tão preciso em aplicações mais sensíveis. Outro dado que chama a atenção é que os dados obtidos possuem uma dispersão bastante pequena. Foi então desenvolvida um *script* em Matlab para analisar o resultado. O código-fonte é apresentado no Apêndice A deste documento. Primeiramente foi confeccionado um gráfico contendo todas as coordenadas geográficas obtidas e a média das medidas, conforme pode-se observar abaixo.

Figura 48: Gráfico das coordenadas geográficas obtidas no teste de precisão do receptor GPS.



Fonte: Autor.

De posse das coordenadas geográficas, foi utilizada uma ferramenta de cálculo de distâncias geográficas (Calculadora Geográfica) para obter a distância de cada ponto em relação ao ponto médio observado. Com tais distâncias, foi possível obter o valor médio da distâncias dos pontos experimentais em relação à média, sendo encontrado um valor de 0,7 metros. Ainda há a necessidade de realização de novos testes com o VANT em movimento, para estimar o erro cometido em uma situação dinâmica. Se o erro permanecer dentro de um metro, é possível utilizar este receptor em aplicações que não exijam precisão tão acurada.

5 CONCLUSÃO

O trabalho apresentado nesse documento se propôs a realizar os primeiros passos da implementação de um sistema embarcado para controle e instrumentação de um VANT. Nesse sentido, foram obtidos avanços consideráveis como a manufatura de uma placa de circuito impresso, estruturação de *software* e implementação de um transceptor RF para comunicação com a estação de solo e de um receptor GPS para navegação.

O trabalho aqui exposto teve como foco um VANT para pulverização agrícola, mas pode ser muito bem aplicado a qualquer veículo autônomo, seja ele aéreo, terrestre ou marítimo.

Os testes expostos nesse documento ainda são preliminares e exigirão testes mais complexos à medida que o projeto atingir um grau mais avançado de desenvolvimento.

A opção por manufaturar uma PCB se mostrou uma decisão acertada, dado que durante o desenvolvimento do trabalho e realização dos testes não foram enfrentadas dificuldades relacionadas a *hardware*, como problemas de conexão dos dispositivos e conectores.

Os testes feitos em *software* a respeito do tempo de processamento de funções básicas ainda não são suficientes para inferir a respeito da capacidade do processador ATmega2560 de processar todas as tarefas do sistema em um tempo hábil. Ao se desenvolver as rotinas mais avançadas de malhas de controle e do filtro estimador, que exigem cálculos mais complexos, mais testes como este exposto no trabalho serão necessários.

Quanto ao teste de alcance do transceptor RF, a folha de dados do dispositivo indica um alcance bem maior do que o obtido na prática. Os fatores que levaram ao baixo desempenho foram: utilização de apenas uma antena em cada transceptor, escolha inadequada das antenas e área de teste urbana, com maior nível de ruído do que o encontrado nas áreas em que o sistema irá operar.

Já o receptor GPS ainda carece de testes em situação dinâmica do VANT, porém esse teste só será possível com a construção do VANT finalizada. Seria muito interessante testar o sistema simultaneamente com um produto comercial de conhecida acurácia para comparação. Algumas opções são possíveis para melhorar a precisão dos dados de posição e velocidade, como a utilização de dois receptores GPS em conjunto ou até mesmo a substituição do receptor atual por um sistema mais preciso, como o sistema GPS RTK.

Alguns dos trabalhos principais a serem desenvolvidos posteriormente:

- Desenvolvimento dos algoritmos de leitura e calibração do IMU;
- Implementação das malhas de controle para estabilização e navegação;
- Implementação dos algoritmos de interpretação de comandos do rádio controlador;

- Implementação das rotinas de armazenamento de informações no cartão SD;
- Escolha de servos motores e atuadores adequados para cada aplicação do sistema;
- Desenvolvimento de um filtro estimador para os dados de navegação;
- Desenvolvimento do sistema de pulverização.

REFERÊNCIAS

ANATEL. Setor Regulado : Apresentação. **ANATEL**, 2015. Disponivel em: <http://www.anatel.gov.br/setorregulado/coordenacao-nacional>. Acesso em: 21 maio 2018.

ANATEL. **REQUISITOS TÉCNICOS PARA A AVALIAÇÃO DA CONFORMIDADE DE EQUIPAMENTOS DE RADIOCOMUNICAÇÃO DE RADIAÇÃO RESTRITA**. ANATEL. [S.l.]. 2017.

ANATEL. ATRIBUIÇÃO DE FAIXAS DE FREQUÊNCIAS NO BRASIL. **ANATEL**, 2018. Disponivel em: <http://www.anatel.gov.br/Portal/verificaDocumentos/documento.asp?numeroPublicacao=348213>. Acesso em: 21 maio 2018.

ANATEL. Resolução nº 680, de 27 de junho de 2017. **ANATEL**, 2018. Disponivel em: <http://www.anatel.gov.br/legislacao/resolucoes/2017/936-resolucao-680>. Acesso em: 21 maio 2018.

ARCHIVED:APM 2.5 and 2.6 Overview. **Ardupilot Copter**, 2016. Disponivel em: <http://ardupilot.org/copter/docs/common-apm25-and-26-overview.html>. Acesso em: 22 maio 2018.

ARDUINO MEGA 2560 & Genuino MEGA 2560. **ARDUINO**. Disponivel em: <https://www.arduino.cc/en/Main/ArduinoBoardMega2560?setlang=en>. Acesso em: 22 maio 2018.

ARDUPILOT Home. **Ardupilot**, 2018. Disponivel em: <http://ardupilot.org/>. Acesso em: 14 maio 2018.

BEAGLEBOARD:BEAGLEBONEBLACK. **elinux**, 2017. Disponivel em: <https://elinux.org/Beagleboard:BeagleBoneBlack>. Acesso em: 22 maio 2018.

BENSON, T. **NASA**. Disponivel em: <https://www.grc.nasa.gov/www/k-12/VirtualAero/BottleRocket/airplane/rotations.html>. Acesso em: 16 maio 2018.

BITTAR, A. **Arquitetura da Unidade Central de Processamento do Pegasus Autopilot: da concepção à implementação de um sistema de tempo real em Hardware-In-The-Loop**. Instituto Tecnológico de Aeronáutica. São José dos Campos. 2012.

CALCULADORA Geográfica. **INPE**. Disponivel em: <http://www.dpi.inpe.br/calcula/>. Acesso em: 27 maio 2018.

CARMO, J. P.; CORREIA, J. H. **Introdução às microtecnologias no silício**. 1ª. ed. [S.l.]: Lidel, 2010.

CAVALLO, A. et al. **Experimental Comparison of Sensor Fusion Algorithms for Attitude Estimation**. Proceedings of the 19th World Congress The International Federation of Automatic Control. Cape Town: [s.n.]. 2014. p. 7585-7591.

DEITEL, H. M.; DEITEL, P. J. **C++:** Como Programar. 3ª. ed. [S.l.]: Bookman, 2001.

DIAGRAMA de caso de uso. **Wikipedia**. Disponivel em: <https://pt.wikipedia.org/wiki/Diagrama_de_caso_de_uso>. Acesso em: 20 maio 2018.

DRONES Classe 3 (RPA com peso máximo de decolagem maior que 250g e até 25 kg). **ANAC Agência Nacional de Aviação Civil**, 2017. Disponivel em: <http://www.anac.gov.br/assuntos/paginas-tematicas/drones/aeronaves-nao-tripuladas-da-classe-3-com-peso-maximo-de-deco-lagem-maior-que-250g-e-ate-25-kg>. Acesso em: 14 maio 2018.

FILIZOLA, H. F. et al. Monitoramento e avaliação do risco de contaminação por pesticidas em água superficial e subterrânea na região de Guaíra. **Pesquisa Agropecuária Brasileira**, Jaguariuna, 37, Maio 2002.

FUTABA T12K 12 Channel 2.4GHz Combo with R3008SB (Mode 2). **Nexus Modelling Supplies**. Disponivel em: <http://www.nexusmodels.co.uk/futaba-t12k-12-channel-2-4ghz-combo-with-r3008sb-mode-2.html>. Acesso em: 19 maio 2018.

HERE GNSS (M8N). **ProfiCNC**. Disponivel em: <http://www.proficnc.com/gps/76-gps-module.html>. Acesso em: 26 maio 2018.

LIANG, O. RC TX RX PROTOCOLS EXPLAINED: PWM, PPM, SBUS, DSM2, DSMX, SUMD. **Oscar Liang**, 2018. Disponivel em: <https://oscarliang.com/pwm-ppm-sbus-dsm2-dsmx-sumd-difference/>. Acesso em: 19 maio 2018.

MAVLINK ArduPilotMega Message Set. **MAVLink**. Disponivel em: <http://mavlink.org/messages/ardupilotmega>. Acesso em: 26 maio 2018.

MAVLINK Common Message Set. **MAVLink**. Disponivel em: <http://mavlink.org/messages/common>. Acesso em: 26 maio 2018.

MAVLINK Micro Air Vehicle Communication Protocol. **QGROUNDCONTROL**, 2018. Disponivel em: <http://qgroundcontrol.org/mavlink/start>. Acesso em: 26 maio 2018.

MISSION Planner Home. **Ardupilot**, 2016. Disponivel em: <http://ardupilot.org/planner/>. Acesso em: 19 maio 2018.

MÓDULO MEDIÇÃO INERCIAL GY-87. **HETPRO STORE**, 2018. Disponivel em: <https://hetpro-store.com/modulo-medicion-inercial-gy-87-mpu6050-hmc5883l-bmp180/>. Acesso em: 22 maio 2018.

NMEA 0183. **Wikipedia**, 2017. Disponivel em: <https://pt.wikipedia.org/wiki/NMEA_0183>. Acesso em: 21 maio 2018.

PACÍFICO-DA-SILVA, I.; MELO, M. M.; SOTO-BLANCO, B. Efeitos tóxicos dos praguicidas para abelhas. **Revista Brasileira de Higiene e Sanidade Animal**, 10, 2016. 142-157.

PIXHAWK 2.1, The Cube. **jDrones**. Disponivel em: <http://store.jdrones.com/pixhawk2_p/pixhawkv2.htm>. Acesso em: 15 maio 2018.

RASPBERRY PI 3 IS OUT NOW! SPECS, BENCHMARKS & MORE. **The MagPi Magazine**, 2016. Disponivel em: <https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/>. Acesso em: 22 maio 2018.

REIS, E. et al. Qualidade da Aplicação Aérea Líquida Com Uma Aeronave Agrícola. **Engenharia Agrícola**, Jaboticabal, 30, Setembro 2010. 958-966.

RFD 900+ Modem. **RFDesign Pty Ltd**, 2018. Disponivel em: <http://store.rfdesign.com.au/rfd-900p-modem/>. Acesso em: 26 maio 2018.

RFDESIGN. **RFD900 Radio Modem Data Sheet**. [S.l.]. 2013.

SECCO, D. et al. Influência da altura e vazão da eficiência de deposição de gotas em pulverização aérea. **Acta Iguazu**, Cascavel, 2, 2013. 40-45.

SERVO Motor - 5Kg Torque. **PotentialLabs**, 2018. Disponivel em: <https://potentiallabs.com/cart/servo-motor-5kg-india>. Acesso em: 2018 maio 20.

SILVEIRA, C. B. Servo Motor: Veja como Funciona e Quais os Tipos. **Citisystems**. Disponivel em: <https://www.citisystems.com.br/servo-motor/>. Acesso em: 20 maio 2018.

TURNIGY 9X REVIEW – 9 CHANNEL RC TRANSMITTER. **Oscar Liang**, 2013. Disponivel em: <https://oscarliang.com/turnigy-9x-review-9-channel-rc-transmitter/>. Acesso em: 22 maio 2018.

U-BLOX. **u-blox 8 / u-blox M8 Receiver Description**. [S.l.]. 2017.

U-BLOX AG. **GPS Essentials of Satellite Navigation**. [S.l.]: [s.n.], 2009.

VARIAÇÃO com a altitude. **Física UFPR**. Disponivel em: <https://fisica.ufpr.br/grimm/aposmeteo/cap4/cap4-3.html>. Acesso em: 19 maio 2018.

WECONTROL History. **weControl**, 2018. Disponivel em: <http://www.wecontrol.ch/>. Acesso em: 14 maio 2018.

WECONTROL Home. **weControl**, 2018. Disponivel em: <http://www.wecontrol.ch/>. Acesso em: 14 maio 2018.

WECONTROL Products. **weControl**, 2018. Disponivel em: <http://www.wecontrol.ch/>. Acesso em: 15 maio 2018.

APÊNDICE A – SCRIPT EM MATLAB PARA ANÁLISE DOS RESULTADOS DO TESTE COM O RECEPTOR GPS

```
clear all;
close all;
clc;

format long;

% criando os vetores de latitude e longitude
latitudes = [-22.0067844, -22.0067825, -22.0067825, -22.0067825, -
22.0067806, -22.0067806, -22.0067806, -22.0067806, -22.0067806, -
22.0067806, -22.0067806, -22.0067806, -22.0067806, -22.0067806, -
22.0067806, -22.0067806, -22.0067787, -22.0067806, -22.0067787, -
22.0067787, -22.0067787, -22.0067787, -22.0067787, -22.0067768, -
22.0067768, -22.0067768, -22.0067768, -22.0067768, -22.0067749, -
22.0067749, -22.0067729, -22.0067729, -22.0067729, -22.0067710, -
22.0067710, -22.0067710, -22.0067691, -22.0067691, -22.0067691, -
22.0067691, -22.0067672, -22.0067672, -22.0067653, -22.0067653, -
22.0067653, -22.0067653, -22.0067653, -22.0067672, -22.0067672, -
22.0067672, -22.0067672, -22.0067691, -22.0067691, -22.0067691, -
22.0067691, -22.0067691, -22.0067710, -22.0067710, -22.0067729, -
22.0067749];
longitudes = [-47.8980865, -47.8980865, -47.8980865, -47.8980865, -
47.8980827, -47.8980827, -47.8980827, -47.8980827, -47.8980827, -
47.8980827, -47.8980827, -47.8980827, -47.8980827, -47.8980827, -
47.8980827, -47.8980827, -47.8980827, -47.8980827, -47.8980827, -
47.8980827, -47.8980827, -47.8980827, -47.8980827, -47.8980865, -
47.8980865, -47.8980865, -47.8980865, -47.8980865, -47.8980865, -
47.8980865, -47.8980904, -47.8980904, -47.8980904, -47.8980904, -
47.8980904, -47.8980904, -47.8980904, -47.8980904, -47.8980904, -
47.8980904, -47.8980904, -47.8980904, -47.8980904, -47.8980904, -
47.8980904, -47.8980904, -47.8980904, -47.8980904, -47.8980904, -
47.8980904, -47.8980904, -47.8980942, -47.8980942, -47.8980942, -
47.8980942];

% Plotando os dados experimentais
scatter(longitudes, latitudes, 'xr');
hold on;

% calculando e plotando a media
latitudes_soma = 0;
longitudes_soma = 0;
for indice = 1:60
    latitudes_soma = latitudes_soma + latitudes(indice);
    longitudes_soma = longitudes_soma + longitudes(indice);
end
latitude_media = latitudes_soma/60
longitude_media = longitudes_soma/60

scatter(longitude_media, latitude_media, 'ob');

% Descrição das coordenadas do ponto médio e dos pontos experimentais

% Ponto Medio: (-47.8980869,-22.0067743)
% Ponto 1: (-47.8980865,-22.0067844) x 1
% Ponto 2: (-47.8980865,-22.0067825) x 3
```

```
% Ponto 3: (-47.8980827,-22.0067806) x 13
% Ponto 4: (-47.8980827,-22.0067787) x 6
% Ponto 5: (-47.8980827,-22.0067768) x 4
% Ponto 6: (-47.8980865,-22.0067768) x 1
% Ponto 7: (-47.8980865,-22.0067749) x 2
% Ponto 8: (-47.8980865,-22.0067729) x 3
% Ponto 9: (-47.8980865,-22.0067710) x 2
% Ponto 10: (-47.8980904,-22.0067710) x 1
% Ponto 11: (-47.8980904,-22.0067691) x 9
% Ponto 12: (-47.8980904,-22.0067672) x 6
% Ponto 13: (-47.8980904,-22.0067653) x 5
% Ponto 14: (-47.8980942,-22.0067710) x 2
% Ponto 15: (-47.8980942,-22.0067729) x 1
% Ponto 16: (-47.8980942,-22.0067749) x 1

% Distância de cada ponto em relação ao valor médio obtidos em
http://www.dpi.inpe.br/calcula/

% Distancias em metros:
% Distancia 1: 1.140
% Distancia 2: 0.925
% Distancia 3: 0.828
% Distancia 4: 0.654
% Distancia 5: 0.512
% Distancia 6: 0.283
% Distancia 7: 0.109
% Distancia 8: 0.164
% Distancia 9: 0.343
% Distancia 10: 0.483
% Distancia 11: 0.652
% Distancia 12: 0.842
% Distancia 13: 1.043
% Distancia 14: 0.819
% Distancia 15: 0.761
% Distancia 16: 0.751

% Declarando o vetor de distancias
distancias = [1.14, 0.925, 0.925, 0.925, 0.828, 0.828, 0.828, 0.828, 0.828,
0.828, 0.828, 0.828, 0.828, 0.828, 0.828, 0.828, 0.828, 0.654, 0.654,
0.654, 0.654, 0.654, 0.654, 0.512, 0.512, 0.512, 0.512, 0.283, 0.109,
0.109, 0.164, 0.164, 0.164, 0.343, 0.343, 0.483, 0.652, 0.652, 0.652,
0.652, 0.652, 0.652, 0.652, 0.652, 0.652, 0.842, 0.842, 0.842, 0.842,
0.842, 0.842, 1.043, 1.043, 1.043, 1.043, 1.043, 0.819, 0.819, 0.761,
0.751];

% Obtendo o valor médio e desvio padrão das distancias
soma_distancias = 0;
for indice = 1:60
    soma_distancias = soma_distancias + distancias(indice);
end
distancia_media = soma_distancias/60

soma_desvio_padrao = 0;
for indice = 1:60
    soma_desvio_padrao = soma_desvio_padrao + ((distancias(indice) -
distancia_media)^2);
end
desvio_padrao_distancia = sqrt(soma_desvio_padrao/59)
```