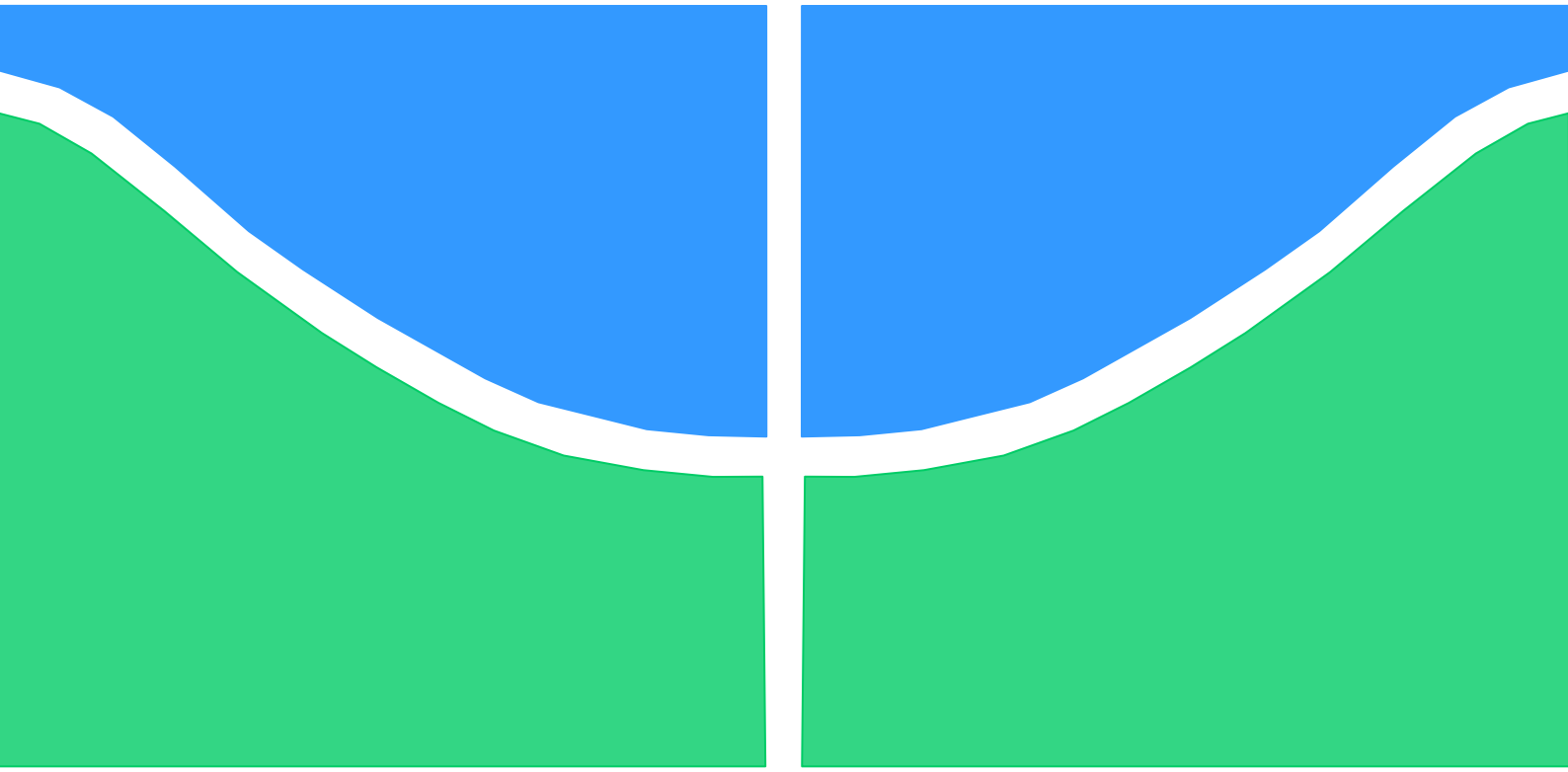


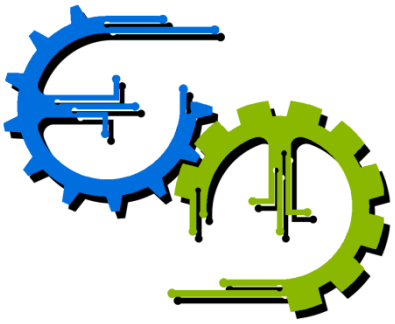
TRABALHO DE GRADUAÇÃO

DESENVOLVIMENTO DO FIRMWARE PARA GUIAGEM NAVEGAÇÃO E CONTROLE DA PLATAFORMA LAICAnSat

Por,

Marina Andrade Lucena Holanda

Brasília, Dezembro de 2016



UNIVERSIDADE DE BRASILIA

Faculdade de Tecnologia

Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

DESENVOLVIMENTO DO FIRMWARE PARA GUIAGEM NAVEGAÇÃO E CONTROLE DA PLATAFORMA LAICAnSat

POR,

Marina Andrade Lucena Holanda

Relatório submetido como requisito parcial para obtenção
Do grau de Engenheiro de Controle e Automação.

Banca Examinadora

Prof. Renato Alves Borges, UnB/ ENE
(Orientador) _____

Prof. Simone Battistini, UnB/ FGA
(Co-orientador) _____

Prof. Geovany Araújo Borges UnB/ENE _____

Brasília, Dezembro de 2016

FICHA CATALOGRÁFICA

MARINA, ANDRADE LUCENA HOLANDA

Desenvolvimento do Firmware para Guiagem, Navegação e Controle da Plataforma LAICAnSat,

[Distrito Federal] 2016.

xvii, 83p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2016). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Aeroespecial
2. CanSat
3. *Firmware*
4. Arduino

I. Mecatrônica/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

ANDRADE LUCENA HOLANDA, M., (2016). Desenvolvimento do Firmware para Guiagem, Navegação e Controle da Plataforma LAICAnSat. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-nº 20/2016, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 86p.

CESSÃO DE DIREITOS

AUTOR: Marina Andrade Lucena Holanda.

TÍTULO DO TRABALHO DE GRADUAÇÃO: Desenvolvimento do Firmware para Guiagem, Navegação e Controle da Plataforma LAICAnSat.

GRAU: Engenheiro ANO: 2016

_____

Marina Andrade

SQN 113 Bloco D ap 307 – Asa Norte.

70763-040 Brasília – DF – Brasil.

RESUMO

Neste trabalho é mostrado o desenvolvimento e implementação em *hardware* do *firmware* que fará parte da terceira missão LAICAnSat, que tem por objetivo fornecer uma solução para a configuração do módulo GNSS e captação de dados dos sensores, além de implementar o controle PID para reproduzir uma trajetória específica na descida da plataforma com o auxílio de paraquedas. A meta é proporcionar à estrutura um pouso em uma área específica de modo autônomo com uma margem de erro aceitável. Além disso são apresentados os projetos nos quais a plataforma está inserida, bem como suas características.

Simulações dos módulos do *firmware*, implementação na placa de circuito impresso LAICAnSat v1.1, e testes do *hardware* ilustram a eficiência e correto funcionamento do *firmware* proposto.

.

Palavras-chaves: PID. Controlador Proporcional Integral Derivativo. Parapente. Teensy. Firmware.

ABSTRACT

This work presents and implementation and development of the in the *hardware* that will be part of the third LAICAnSat mission, which aims to provide a solution for the configuration of the GNSS module and capture of data from sensors, as well as implement the PID control to guide the platform in the descent phase of the flight. The goal is to provide for the structure a secure landing in a specific area in an autonomous way with an acceptable margin of error. In addition, are presented the projects in which the platform is inserted, as well as its characteristics.

Simulations of the firmware modules, implementation on the LAICAnSat v1.1 printed circuit board, and hardware tests illustrate the efficiency and correct operation of the proposed firmware.

Keywords: PID; CubeSat; Firmware; Proportional Integral Derivative;

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Siglas

APRS	Automatic Packet Reporting System
IMU	Inertial Measurement Unit
MCU	Microcontroler
GPS	Global Position System
GNSS	Global Navigation Satellite System
IDE	Integrated Development Environment
UART	Universal Asynchronous Reiceiver/Transmitter
I2C	Inter-Integrated Circuit
SCL	Clock Signal
SDA	Data Signal
SPI	Serial Peripheral Interface
SS	Slave Select
MOSI	Master Out Slave In
MISO	Master In Slave Out
SCK	Clock Signal
PCB	Printed Circuit Board
ARHAB	Amateur Radio High Altitude Ballooning
PID	Proportional Integral Derivative Controller
AEB	Agência Espacial Brasileira
CINDACTA I	1◦ Centro Integrado de Defesa Aérea e Controle de Tráfego Aéreo
NOTAM	Notice to Airmen
MUTUM	Grupo de rádio amadores
DCEA	Departamento de Controle do Espaço Aéreo
CAsB	Clube Astronômico de Brasília
PID	Proportional Integral Derivative Controller
ECMWF	European Centre for Medium-Range Weather Forecasts
GLONASS	Globalnaya navigatsionnaya sputnikovaya sistema ou Sistema de Navegação Global por Satélite
QZSS	Quasi-Zenith Satellite System
Beidou	Navigation Satellite System
ISS	International Space Station
USB	Universal Serial Bus

CAPÍTULO 1 – INTRODUÇÃO

Neste capítulo são apresentadas as características da plataforma e da missão LAICAnSat e os projetos e parcerias das quais a plataforma faz parte, a fim de, justificar a definição do problema que motiva o presente trabalho.

1.1 A PLATAFORMA

A plataforma LAICAnSat está sendo desenvolvida pelo Laboratório de Aplicação e Inovação em Ciências Aeroespaciais (LAICA) da Universidade de Brasília (UnB) desde 2013 [1] para a realização contínua de pesquisas científicas em altas e baixas altitudes, utilizando o conceito CanSat e BalloonSat para simular uma missão de satélite.

Devido ao seu conteúdo interdisciplinar e a possibilidade de trabalhar com artigos eletrônicos de baixo custo, as plataformas CanSat/BalloonSat se adaptam melhor ao ambiente educacional e científico, além de possuir grande potencial para aplicações comerciais [2], [44], [45], [46], [47]. O projeto consiste no primeiro balão para altitudes elevadas auto recuperável desenvolvido pela Universidade de Brasília com foco em aplicações de sensoriamento remoto.

CanSat, palavra de origem inglesa da união Can (lata) e Sat (abreviação de satélite), como o próprio nome sugere, refere-se a um satélite equipado com diversos dispositivos de sensoriamento miniaturizado ao tamanho de uma lata como visto na Figura 1.

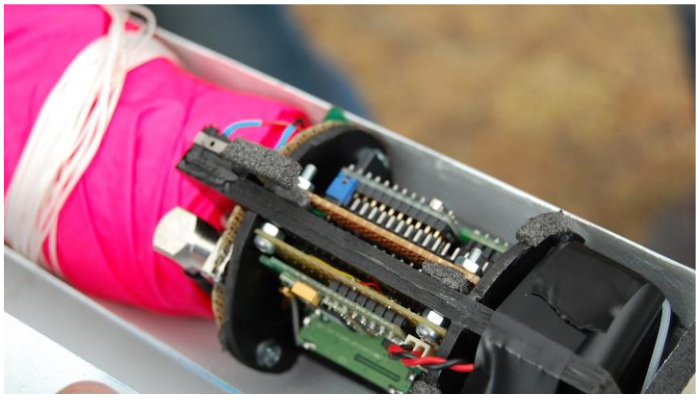


Figura 1 - CanSat montado dentro de um foguete (Fonte: [49]).

O tamanho reduzido, tem por finalidade possibilitar ao satélite ser embarcado em foguetes de sondagem. Também chamados de foguetes de pesquisa, os foguetes de sondagem são projetados para fazer medições e executar experimentos científicos em voos suborbitais.

Já os BalloonSat (Figura 2) se caracterizam por usar um balão para altas altitudes capaz de levar uma carga útil até a estratosfera. Não apresentando tantas restrições de dimensão quanto o CanSat, os BalloonSats também possibilitam a realização de experimentos científicos, de baixo custo, em altas altitudes. Essas plataformas são classificadas pela legislação radio amadora como Balões Amadores de Alta Altitude (ARHAB, do inglês *Amateur Radio High Altitude Ballooning*), possibilitando a utilização de sistemas de rastreamento de longa distância e a infraestrutura existente da comunidade radioamadora [5].

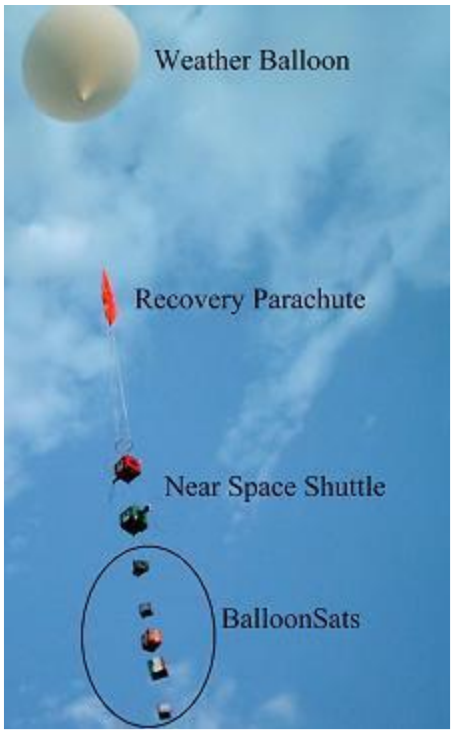


Figura 2 - Múltiplos BalloonSats em um único lançamento (Fonte: [49]).

Sendo assim, o LAICAnSat aproxima seu formato de uma combinação de BalloonSat com CanSat, possibilitando o balonismo de alta altitude com as características compactas do CanSat.

1.2 PROJETO LAICAnSat

O projeto tem como alvo desenvolver uma plataforma de missões de alta altitude que possibilite a integração dos experimentos BalloonSat e CanSat.
Abaixo são apresentados os objetivos introdutórios dos dois lançamentos já realizados, correspondentes as missões LAICAnSat-1 e LAICAnSat-2;

- Obter dados atmosféricos;
- Obter dados relativos à altitude da carga útil;
- Obter imagens da superfície terrestre;
- Obter a telemetria dos sensores embarcados;
- Manter a temperatura ideal dos componentes eletrônicos;
- Manter autonomia do sistema rastreamento da carga útil;

1.2.1 MISSÃO LAICAnSat-1

Após consulta ao 1º Centro de Integrado de Defesa Aérea e Controle de Tráfego Aéreo (CINDACTA I) foi definida a data de lançamento e o local de onde partiria o BalloonSat da missão. Conforme orientação dos oficiais do CINDACTA I, a cidade de Padre Bernardo, GO e o dia 02 de maio de 2014 eram os mais favoráveis ao lançamento. Vale salientar que a determinação desta região se deu em função da sua favorável cobertura de repetidores de sinal APRS, ficando esta orientação disponível a próximos lançamentos que venham a ocorrer. Outro obstáculo a ser resolvido era o rastreamento e recuperação do balão após pouso, pois, devido as intempéries físicas e climáticas em altura próxima ao solo é natural a perda de sinal de localização. Com isto o Grupo MUTUM constituído de radioamadores orientou a equipe de lançamento da Missão LAICAnSat-1, determinando quais ferramentas e técnicas deveriam ser utilizadas para recuperação do BalloonSat após pouso. Com

isto o carro da equipe de lançamento foi equipado com um rádio Kenwood TM710A, que possui transmissão 50W, conforme apresentado na Figura 3.

Figura 3 - Rádio Kenwood TM710A (Fonte: [38]).

Para interação entre o BalloonSat e o rádio foi construído um refletor de radar, conforme orientação Departamento de Controle do Espaço Aéreo (DCEA), pois este equipamento é detectável para radares que operam em uma frequência de 200MHz a 2.1700MHz.

O voo durou aproximadamente 3h20 minutos e o BalloonSat pousou em um local, aproximadamente 9,5km distante do local de lançamento. As imagens requeridas da superfície terrestre foram obtidas, conforme Figura 4. Bem como o rastreamento da trajetória do BalloonSat, conforme Figura 5.

Figura 4 - Imagens da superfície terrestre (Fonte: [38]).

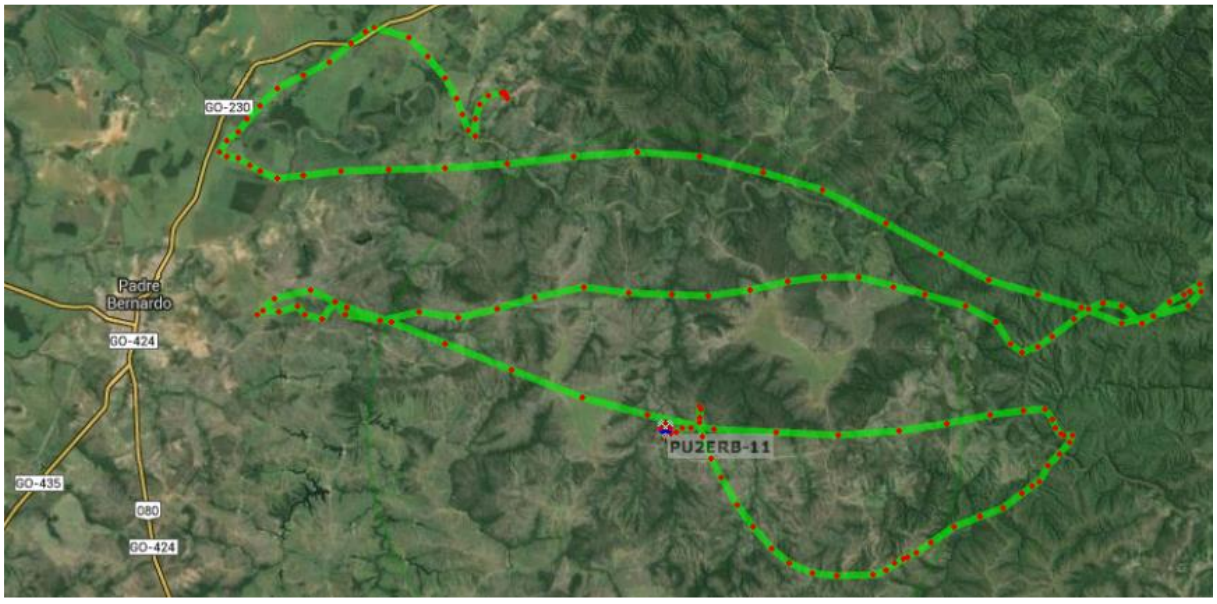


Figura 5 - Trajetória do módulo LAICAnSat durante a missão (Fonte: [38]).

Com isto, os objetivos da missão foram alcançados, tornando-a assim efetiva e de grande valor para experiências futuras.

1.2.2 MISSÃO LAICAnSat-2

Baseado no sucesso das experiências colhidas com a Missão LAICAnSat-1 foi determinado que a região de Padre Bernardo, GO, seria mantida para este novo lançamento, ocorrido em, 24 de maio de 2014. Este segundo lançamento teve novos objetivos, demonstrados abaixo:

- Testar sensores de infravermelho;
- Testar sensores de temperatura e umidade;
- Revisar os procedimentos de lançamento e preparação;

Além disto foi realizado um novo experimento que visava observar a expansão do ar dentro de uma bola de futebol murcha (Figura 6) conforme diminuição da pressão externa.

Conforme realizado na primeira missão, o lançamento do LAICAnSat-2 ocorreu após orientação do 1º Centro de Integrado de Defesa Aérea e Controle de Tráfego Aéreo (CINDACTA I).

Em função da assimetria na estrutura do balão por causa da bola de futebol instalada, foi incluído um contrapeso objetivando reestabelecer o centro de gravidade do balão no mesmo local do seu centro de massa.

Figura 6 – Imagem capturada pelo módulo durante a missão LAICAnSAT-2 (Fonte:[50]).

Iniciado o lançamento notou-se que o balão retornou ao solo em função do seu contrapeso, o que fez com que a equipe de lançamento o removesse e refizesse o lançamento. O peso da bola de futebol permitiu ao balão uma taxa de ascensão menor do que a taxa registrada no primeiro voo, o que consentiu a equipe um teste mais assertivo do sistema embarcado, já que proporcionava uma maior exposição as situações atmosféricas encontradas. Com isto, o tempo total de voo foi de aproximadamente 5 horas.

Como o balão não possuía tecnologia de direcionamento, a queda da carga útil foi na região de Agua Fria, em um local de difícil acesso, pois conta com vários declives e aclives, além de uma mata muito fechada, somado a isto, o horário da queda da carga útil não permitia fácil localização, pois, tratava-se do entardecer. Assim a equipe de lançamento tomou a decisão de retornar no dia seguinte para efetuar novas buscas. No dia 25 de maio de 2014 a equipe retornou a região de Agua Fria acompanhados de alguns radioamadores do Grupo MUTUM, que auxiliaram no monitoramento do balão de forma remota, em Brasília, DF. Mesmo assim o grupo não obteve sucesso, pois novamente a ausência de ferramentas de leitura geográfica, como mapas de relevo topográfico dificultou a busca. Após solicitação da Universidade de Brasília, um grupo de bombeiros da 10ª Companha Independente de Bombeiro Militar, de Planaltina de Goiás, GO, em 31 de maio de 2014, realizou o resgate da carga útil. Após resgate, os dados do microSD foram processados e as imagens da câmera recuperadas, conforme Figura 7.

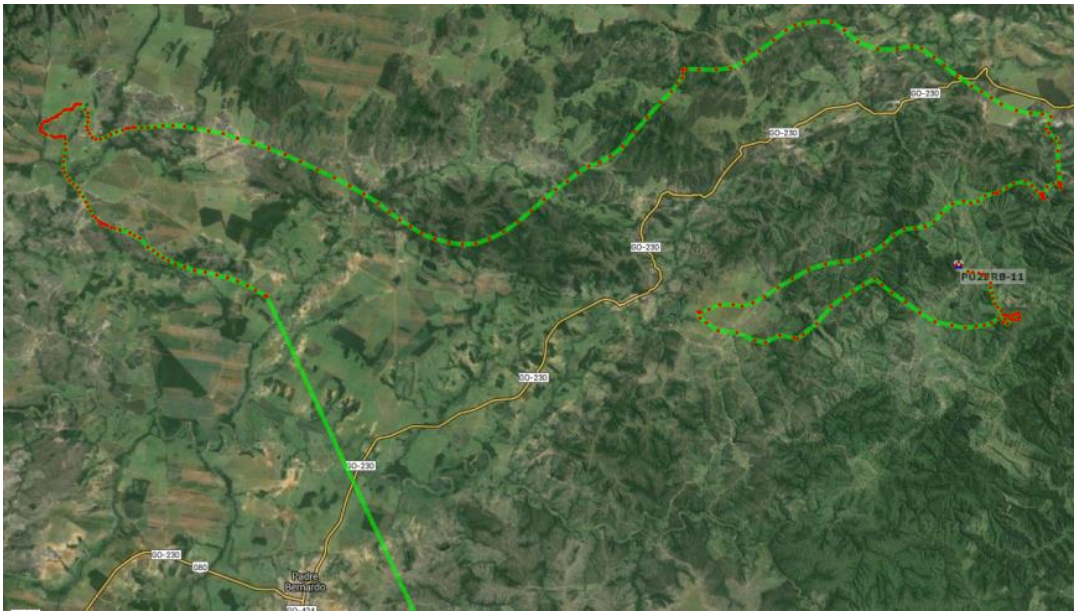


Figura 7 - Trajetória BalloonSat dada pelo sistema APRS (Fonte: [38]).

O experimento foi considerado um sucesso sendo inclusive foi noticiado em um grande meio de comunicação televisiva [50].

Umas das principais lições aprendidas com o Projeto LAINCaSat-2 foi que se o balão dispusesse de um sistema de controle de direção não haveria risco de perda do módulo e consequentemente dos dados e equipamentos desenvolvidos.

1.2.3 MISSÃO LAICAnSat-3

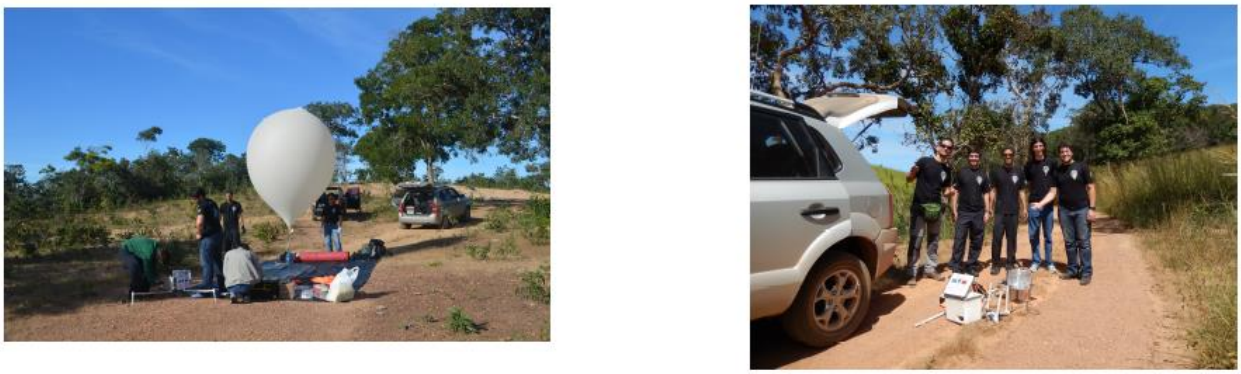


Figura 8 - Equipe do LAICAnSat no primeiro lançamento em 2 de maio de 2014 (Fonte: [8]).

Após uma fase de testes, vide Figura 7, em 2014 com dois lançamentos para testar o sistema preliminar projetado, e fazer estudos na área de modelagem e identificação aerodinâmica do parapente utilizado durante a descida da carga útil [8], o padrão CubeSat 3U foi adotado para a versão atual da estrutura.

O padrão CubeSat é caracterizado por dar ao satélite o formato de um cubo (Figura 9) de 10 centímetros de aresta. Esse tipo de satélite miniaturizado vem se tornando popular nos últimos tempos devido a fácil manufatura. As aplicações são vastas e multifuncionais. Os CubeSats estão sendo usados tanto para finalidades relacionadas à pesquisa quanto a propósitos comerciais. As empresas que desenvolvem pequenos sistemas de sensoriamento remoto por satélite podem oferecer um conjunto de dados único, como imagens de toda a Terra atualizadas diariamente, em uma frequência muito maior do que as fornecidas por sistemas de satélites tradicionais. Essas imagens podem então ser analisadas por computadores usando algoritmos de aprendizagem de máquina que podem extrair informações para

uso em várias aplicações como por exemplo, melhorar a produção agrícola [44], [45], [46], [47].

Figura 9 - CubeSat brasileiro (Fonte: [51]).

Este padrão escolhido para ser implementado como estrutura da plataforma será composto de três cubos, com 10 cm de aresta, impressos através de uma máquina de prototipagem 3D, utilizando filamentos PLA (Poliácido Lático), e posteriormente empilhados longitudinalmente para compor a estrutura LAICAnSat observada na Figura 10.

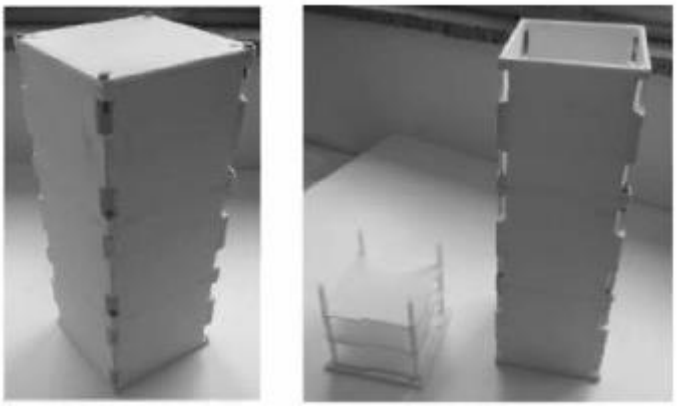


Figura 10 - Estrutura LAICAnSat CubeSat 3U (Fonte: [6]).

Como mostrado na Figura 11, a estrutura será dividida em três seções, a primeira armazena uma unidade para o sistema de atuação, a segunda para o computador de bordo e a terceira para a carga útil que define a missão.

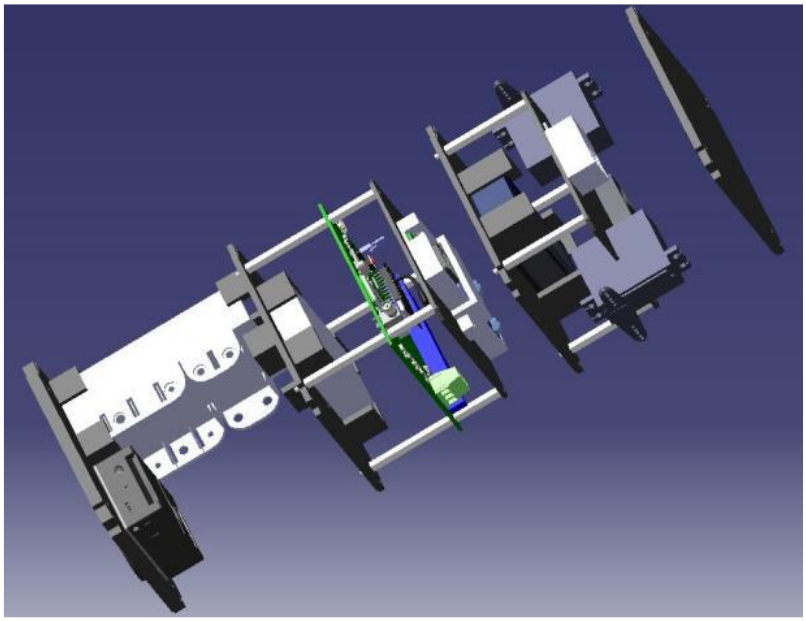


Figura 11 - LAICAnSat desenhado usando CATIA: Visão interna (Fonte: [6]).

O sistema de reentrada LAICAnSat será constituído por um paraquedas, responsável pela redução da velocidade do veículo durante a descida e pela capacidade de manobra durante a fase de pouso.

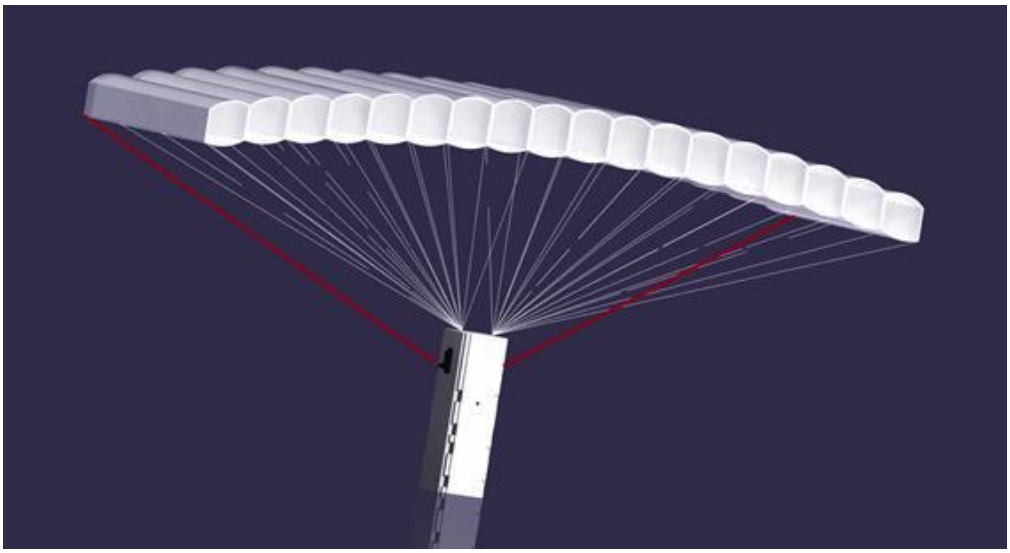


Figura 12 - Estrutura do parapente (Fonte: [6]).

Conforme mostrado [6], uma armadura foi projetada para ser anexada ao velame com o objetivo de evitar que o parapente não se abra ou torça no início da reentrada. Implementada com barras transversais no centro do velame, esta estrutura

iniciará parcialmente aberta na fase ascendente, estando totalmente aberta durante a fase de descida. As linhas do parapente são ligadas à parte superior da estrutura. Em particular, as duas linhas externas do bordo de arrasto são utilizadas para manobrar, sendo ativadas por dois motores de passo distintos.

Projetado no formato PC104, exigindo que a placa tenha dimensões de 90x96mm, com consumo limitado de cerca de 1W~2W, limitando a corrente em 4mA, o computador ajusta-se à estrutura CubeSat perfeitamente. Sendo programável via USB e capaz de coletar dados científicos ao longo do voo, bem como obter informações suficientes para calcular sua trajetória para fins de navegação. Outra característica interessante desse padrão é a possibilidade de conectar um módulo PC104 com outro.

Além disso, possui um sistema de rastreamento e telemetria baseado no sistema APRS e um transmissor xBee que permite a comunicação half-duplex para manter o controle da localização da plataforma e dos sensores, e também permite monitorar as condições de funcionamento do sistema em todos os momentos durante o voo.

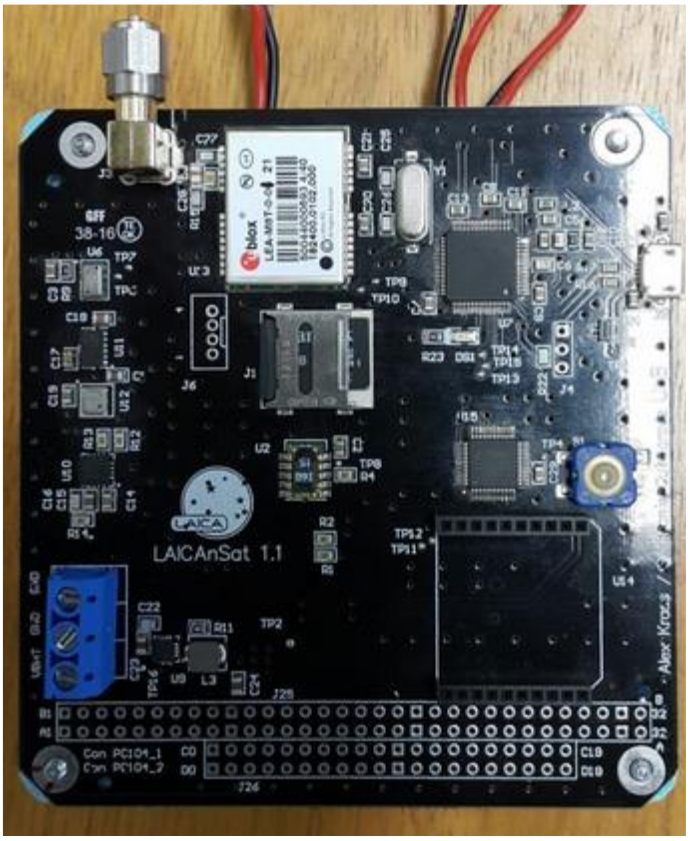


Figura 13 - Computador de bordo (Fonte: [9]).

O computador de bordo (Figura 13) foi implementado com um conjunto de sensores para adquirir dados meteorológicos, um módulo GNSS, um giroscópio e uma unidade de medida inercial. O armazenamento de dados é feito em um cartão SD tornando-o adequado para uma ampla variedade de aplicações.

Em relação ao microcontrolador, o LAICAnSat utiliza uma placa Teensy 3.1 por suas características de processamento superiores em relação a outras placas como o Arduino. A possibilidade de trabalhar com o mesmo ambiente de desenvolvimento integrado e linguagem usados pelo Arduino contribuiu fortemente na escolha [7].

1.3 MOTIVAÇÃO

O projeto LAICAnSat é um esforço realizado dentro da Universidade de Brasília para aprimorar o conhecimento no campo de tecnologia aeroespacial no Brasil. O desenvolvimento desse projeto depende da evolução de diferentes frentes de pesquisa como mostrado na seção 1.1 sendo elas referentes ao desenvolvimento da estrutura física da plataforma, bem como do sistema computacional.

Em seu primeiro lançamento a plataforma contava com um protótipo do sistema embarcado constituído de placas de desenvolvimento dentro de uma caixa de material isolante [7]. Hoje o sistema conta com uma estrutura física e computador de bordo que obedecem a padrões utilizados mundialmente. Isso permite não só a manufatura da placa em larga escala, como também a torna mais próxima de um produto que pode ser lançado no mercado.

O *firmware* desenvolvido durante esse trabalho é uma peça importante para a evolução do projeto LAICAnSat, pois implementa o cerne da plataforma, dando propósito às pesquisas desenvolvidas no projeto até este momento, já que torna possível a realização das missões futuras. Com a estrutura mecânica e de *hardware* bem definidas, o *firmware* implementa a captura de dados, o sistema de posicionamento e a estratégia de controle, que são de suma importância para o acompanhamento das missões e para o pouso da plataforma.

Além disso, o desenvolvimento desse trabalho também contribui para a evolução dos projetos descritos a seguir.

1.3.1 PROJETO GLONASS

A região em torno de Brasília é fortemente afetada pela Anomalia Equatorial Ionosférica, caracterizada por uma ionosfera altamente variável. Dentro do projeto GLONASS estão previstas investigações dos efeitos do clima espacial na região de Brasília, cálculo do posicionamento via satélite assim como o teste de diferentes modelos para correção ionosférica para receptores com uma única banda de frequência. Nesse contexto o *firmware* a ser desenvolvido terá a função de configurar o módulo GNSS para a obtenção de medições de dados brutos na banda L1 que serão usados na obtenção de arquivos RINEX para pós processamento.

1.3.2 PROJETO MICROGRAVIDADE DA AEB

O projeto Plataforma Sensorial para Medidas Fisiológicas em Voo Suborbital da Agência Espacial Brasileira (AEB), tem por objetivo realizar uma série de medições fisiológicas em um espaçonauta durante o seu voo suborbital, em micro gravidade, no veículo espacial Lynx MarkII.

A plataforma será equipada com sensores inerciais e registrará dados de voo em ambiente interno, a fim de realizar uma análise do pós voo de todas as medições coletadas confrontando os dados fisiológicos e de trajetória. Para o sucesso da missão o *firmware* tem que estar apto a configurar e captar os dados dos sensores disponíveis na plataforma guardando os resultados em um arquivo para um estudo posterior.

1.3.3 PROJETO KUARAY

Uma parceria entre a UnB, o CAsB (Clube de Astronomia de Brasília) e Grupo Mutum de Rádio Expedição, pretende realizar o lançamento de um balão estratosférico de coleta de dados com uma câmera especialmente preparada para capturar o eclipse solar de 2017 nos EUA.

O fenômeno vai ter seu ponto máximo no centro dos Estados Unidos, onde o projeto Kuaray fará o lançamento de um balão de alta altitude possibilitando a coleta

dados da estratosfera, como pressão, temperatura, entre outros, durante a ocorrência do fenômeno.

Nessa missão é exigido do *firmware* as funcionalidades de configuração e captura de dados por parte do sensores e módulo GNSS, também exigirá a implementação do sistema de controle que permitirá que a plataforma pouse numa área pré-determinada.

1.4 OBJETIVOS DO TRABALHO

O objetivo do presente trabalho é o desenvolvimento do firmware para a missão LAICAnSat, compatível com a placa multifuncional desenvolvida a partir de projetos anteriores. Além da implementação de um controle autônomo que permita que a plataforma LAICAnSat pouse em uma área pré-estabelecida pela equipe de solo da missão.

No próximo capítulo serão apresentadas com detalhes as definições do problema e as preliminares dentro de cada projeto usado como motivação para a implementação do *firmware*. Serão apresentadas também, as ferramentas, o software e a metodologia usada no projeto de uma maneira que seu resultado possa ser reproduzido e usado como referência para continuidade do projeto LAICAnSat.

No capítulo 3 serão apresentadas simulações e discussões dos resultados. No último capítulo faz-se uma breve conclusão acerca dos resultados e algumas sugestões para trabalhos futuros.

CAPÍTULO 2 – PRELIMINARES E DEFINIÇÃO DO PROBLEMA

Nessa seção são apresentados os elementos iniciais para o entendimento da metodologia e escolhas feitas durante para a implementação. Para o cumprimento dos objetivos desse trabalho, precisa-se entender a configuração do *hardware* desenvolvido nas missões anteriores, bem como os requisitos de cada projeto parceiro da missão LAICAnSat.

2.1 PRELIMINARES

2.1.1 ESPECIFICAÇÕES DO *HARDWARE*

A placa do sistema computacional embarcado, usada na missão LAICAnSat, foi desenvolvida segundo o padrão PC104 se adequando ao padrão CubeSat. A placa possui um microcontrolador, cinco sensores e um transmissor Xbee. Esse sistema mínimo foi projetado a partir do estudo sobre a funcionalidade de cada placa de desenvolvimento que compunha o protótipo utilizado nas primeiras missões descritas em [9].

Para o seu melhor entendimento, a placa pode ser dividida, segundo suas funcionalidades em três módulos: módulo de processamento e armazenamento de dados, composto por Teensy 3.1 e cartão SD, módulo de coleta de dados, composto por sensores e módulo GNSS, e módulo de comunicação, composto por Xbee e saída USB.

O Módulo de Processamento e armazenamento, conta com uma placa Teensy 3.1 (Figura 14) que possui um microcontrolador central MK20DX128VLHS, ARM Cortex M4 de 32 bits, com um *clock* de 16 MHz, 64 Kb de RAM e 256 Kb de memória Flash.

Figura 14 - Teensy 3.1 (Fonte: [52]).

O Teensy pode ser programado via USB usando a plataforma de desenvolvimento do Arduino e tem suporte para os protocolos de comunicação USB, Serial, I2C, SPI entre outros. Todos os seus pinos digitais têm uma tolerância de 5 V, já os pinos analógicos (A10-A14), AREF, Program e Reset operam em 3.3V.

O módulo de coleta de dados possui uma série de sensores embutidos. O BMP180 mostrado na Figura 15, apesar de medir temperatura foi desenvolvido com o objetivo de medir a pressão atmosférica. Os dados fornecidos por esse sensor permitem determinar a altitude e realizar previsões do tempo com grande precisão.

O seu baixo consumo de energia o torna ideal para o uso em equipamentos compactos como GPS.

Figura 15 - BMP 180 (Fonte: [53]).

O L3GD20H (Figura 16) é um sensor de baixa potência, que mede variação angular em três eixos. Ele inclui um elemento sensor e uma interface I2C capaz de fornecer a taxa de variação angular medida ao mundo externo através da interface digital (I2C / SPI).

O sensor tem uma escala completa de ± 245 / ± 500 / ± 2000 dps e é capaz de medir taxas com uma largura de banda selecionável pelo usuário.

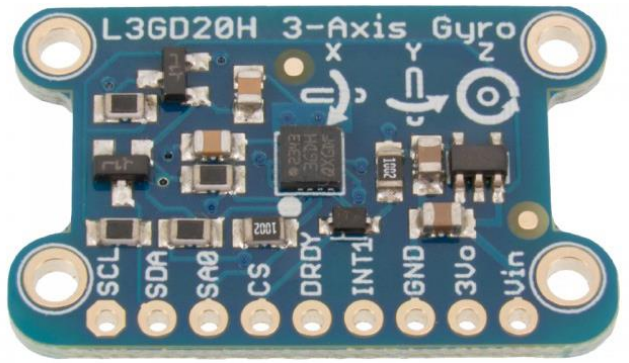


Figura 16 - LSM303 (Fonte: [54]).

O LSM303DLHC (Figura 17) é um módulo composto por um sensor digital de aceleração linear em 3 dimensões e por um sensor digital magnético em 3 dimensões.

O módulo inclui uma interface de barramento serial I2C que suporta modo padrão e rápido de 100 kHz e 400 kHz. O sistema pode ser configurado para gerar sinais de interrupção por eventos inerciais de despertar / queda livre, bem como pela posição do próprio dispositivo. Os limiares e o tempo dos geradores de interrupção são programáveis pelo usuário final. Os blocos magnéticos e de acelerômetro podem ser ativados ou colocados no modo de desligamento separadamente.

Figura 17 - LSM303DLHC (Fonte: [55]).

O MS5611 (Figura 18) proporciona um valor preciso de pressão e temperatura digital de 24 Bit e diferentes modos de operação que permitem ao usuário otimizar da velocidade de conversão ao consumo de corrente. Este sensor de pressão barométrica é optimizado para altímetros e variômetros com uma resolução de altitude de 10 cm. O módulo de sensores inclui um sensor de pressão de alta linearidade e um Conversor analógico digital de 24 bit, de potência ultrabaixa com coeficientes internos calibrados de fábrica.

Figura 18 - MS5611 (Fonte: [56]).

O SHT15 é um sensor de temperatura e umidade digital altamente preciso. A tecnologia digital CMOSens® integra dois sensores e circuitos de leitura em um único chip. Os dois sensores integrados no SHT15 foram acoplados sem interrupção a um conversor analógico a digital de 14bit e um circuito de interface serial, resultando em qualidade de sinal superior, tempo de resposta rápido e uma forte resistência a distúrbios externos. Além disso, o SHT15 a bordo apresenta uma faixa de medição de 0 a 100% de HR com uma precisão de temperatura de +/- 0,3 ° C a 25 ° C. Há apenas quatro pinos que precisam ser conectados para começar a usar este sensor em um projeto. Uma para VCC, uma para GND e as duas linhas de dados SDA e SCL.

Figura 19 - SHT15 (Fonte: [57]).

A placa vem com um módulo GNSS da família LAE-M8T Ublox, mais recente [10], que oferece a recepção simultânea de sinais GPS, QZSS, GLONASS, BeiDou. O módulo LEAM8T é um dispositivo Multi-GNSS capaz de fornecer dados brutos de posicionamento, que são usados para direcionar e posicionar a plataforma LaicanSat, usados tanto no solo para acompanhar o voo, como a bordo da plataforma para fornecer o feedback ao controlador PID.

Figura 20 - LEA-M8T (Fonte: [10]).

O hardware também conta com reguladores de corrente e tensão para que a alimentação da placa seja fornecida adequadamente.

2.1.2 PROTOCOLOS DE COMUNICAÇÃO

O primeiro aspecto que deve ser entendido no projeto do *firmware* são os protocolos de comunicação. O conhecimento dos protocolos de comunicação se faz necessário para entender como os sensores irão interagir entre si e assim evitar conflitos entre eles. Na Tabela 2.1 são mostrados os protocolos de comunicação que cada elemento usa ao se comunicar com o microcontrolador.

Tabela 2.1 Protocolos de comunicação.

Componente	Protocolo	Fabricante
BMP180	I²c	Bosch
L3GD20H	I²c	ST Microletronics
LSM303DLHC	I²c	ST Microletronics
MS5611	I²c	Measurement Specialties
SHT15	2-Wire	Sensirion
Módulo GNSS M8T	UART	Ublox
Cartão de memória	SPI	----------

O computador de bordo faz uso de apenas quatro protocolos para comunicar o microcontrolador e seus dispositivos.

O protocolo UART (universal asynchronous receiver/transmitter) é um protocolo assíncrono, ou seja, a transmissão é dada de forma serial sem a necessidade que um sinal de *clock* seja usado na comunicação.

Os dados são enviados em pacotes de 10 a 12 bits, divididos em bits de dados, bits de sincronização e bit de paridade. O requisito para que os dispositivos se comuniquem é que ambos precisam ter a mesma frequência de envio dos bits do pacote, ou seja, o mesmo *Baud Rate* [11].

No protocolo I²C (Inter-integrated Circuit), diferente do UART, são permitidos conectar até 1008 dispositivos no mesmo canal de comunicação. Funcionando em uma dinâmica mestre-escravo, o protocolo I²C exige um canal para o sinal de *clock* (SCL) e outro para os dados (SDA) propriamente ditos. O microcontrolador sendo o mestre, quando o mesmo deseja saber alguma informação sobre um certo sensor, envia pelo canal SDA o endereço desse sensor e o mesmo envia as informações requisitadas. Cada sensor deve possuir um endereço único para que a comunicação seja efetiva. Esse endereço é decidido pelo fabricante e cabe ao projetista atentar para esse detalhe na escolha dos sensores que irão compor seu sistema de coleta de dados, para que não haja conflito, pois não é possível alterar o endereço de um sensor por meio de programação.

O endereço de identificação dos sensores usados é apresentado na Tabela 2.2 [12].

Tabela 2.2 Endereço dos sensores usados.

Componente	Endereço I²C
BMP180	0x77
L3GD20H	0x6B
LSM303DLHC	0x19 & 0x1E
MS5611	0x76 & 0x77

Quando se tem mais de uma opção para o endereço, como é o caso do LSM303 e MS5611, é possível via programação escolher usar um dos dois e assim evitar conflitos de sensores.

Para a comunicação com o cartão de memória, usa-se o protocolo SPI, onde são necessários quatros canais distintos para comunicação, *clock*, *MOSI* (master out slave in), *MISO* (master in slave out), *Slave Select* (SS).

Da mesma maneira que o I²C, o microcontrolador seleciona o dispositivo com o qual deseja se comunicar, mas ao invés de selecionar um endereço, o mesmo seleciona o pino *SS* ao qual o escravo está conectado através do canal *MOSI* e o escravo responde através do canal *MISO* enviando os dados requeridos [13].

Sendo utilizado apenas pelo sensor SHT15, o 2-wire bus usa o mesmo *clock* que a comunicação SPI, enquanto a transmissão de dados ocorre por um canal de dados exclusivo para esse sensor, como pode ser observado na Tabela 2.3 [14].

Tabela 2.3 Conexão de cada sensor.

Componente	Pino
BMP180, L3GD20H, LSM303, MS5611 (I²C)	SCL – 19 / SDA – 18
Módulo GNSS M8T (UART)	0 e 1 (serial 1)
Cartão SD (SPI)	MOSI - 7/ MISO - 8 / SCK - 14 / SS - 20
SHT15 (2-Wire)	SCL – 14 / SDA - 15

2.2 DEFINIÇÃO DO PROBLEMA

Apresentados os aspectos do *hardware*, é importante contextualizá-lo com os objetivos de cada missão em que a plataforma LAICAnSat está envolvida, para que se possa traçar os objetivos a serem alcançados pelo *firmware* a ser desenvolvido.

O problema a ser resolvido nesse trabalho tem como requisitos:

- Capturar dados dos sensores;
- Usar módulo GNSS para capturar a posição e direção;
- Capturar dados brutos na banda L1;
- Armazenar as informações obtidas em arquivos no cartão de memória;
- Implementar o controle PID desenvolvido a partir de análise e simulação matemática;

CAPÍTULO 3 - O *FIRMWARE*

Nesse capítulo serão mostrados a metodologia e ferramentas usadas para a implementação do *firmware* para a plataforma LAICAnSat.

3.1 CONFIGURAÇÕES INICIAIS

Para assegurar o correto funcionamento e o sucesso dos testes que serão explicados na próxima seção, serão mostradas as configurações iniciais e um teste simples para checar o funcionamento do *hardware*. Seguindo as orientações mostradas a seguir, evita-se conflitos de arquivo e erros de compilação, facilitando a reprodução dos resultados.

Como dito anteriormente, o sistema embarcado permite que sua programação seja feita via USB usando o ambiente de desenvolvimento do Arduino e sua linguagem de programação.

A programação foi feita usando Sistema Operacional Windows 10 e a Arduino IDE versão 1.6.4. Como a Arduino IDE não vem com suporte para Teensy 3.1, foi necessário instalar a extensão Teensyduino à IDE do Arduino e o aplicativo Teensyloader para poder programar a placa [15].

Depois de baixados e instalados os aplicativos citados, o primeiro passo é selecionar na IDE o modelo “Teensy 3.1” como mostrado na Figura 21:

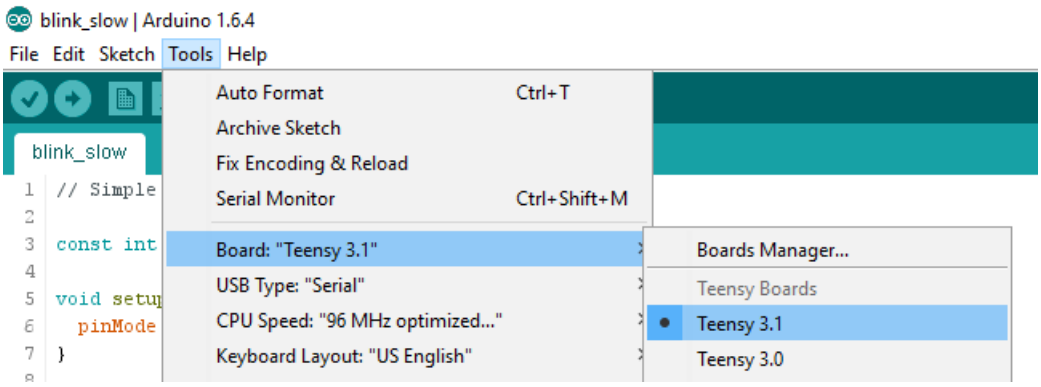


Figura 21 - Seleção do modelo Teensy 3.1.

Na Figura 22 é selecionado o exemplo *blink* para realizar um teste simples:

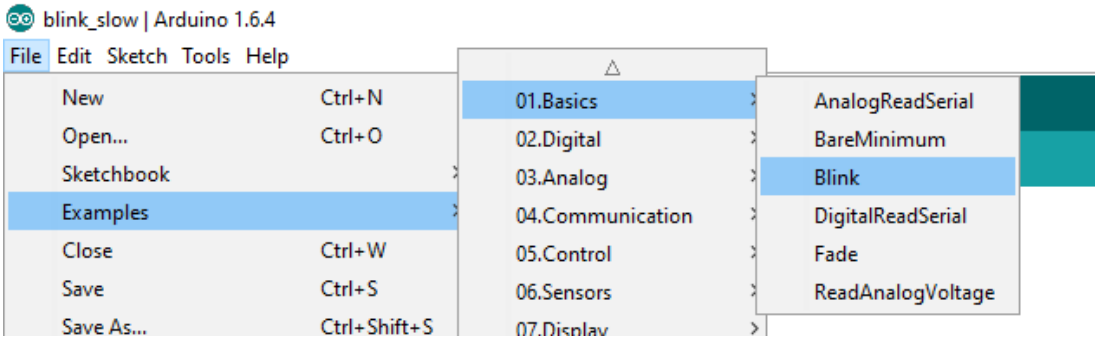


Figura 22 - Exemplo para teste simples.

Muda-se o valor da variável ‘led’ de 13 para 30, que é o pino onde o *led* está conectado. Como mostrado na Figura 23.

```
// Pin 13 has the LED
// give it a name:
int led = 30;

// the setup routine
void setup() {
  // initialize the d
```

Figura 23 - Variável ‘led’.

Para compilar o programa clica-se em “*Verify*”, sinalizado em amarelo na Figura 24:

Figura 24 - Compilando o exemplo.

O *Teensy Loader*, mostrado na Figura 25, automaticamente irá abrir e atualizar o novo arquivo a ser gravado mostrando o tamanho que deve ser ocupado na memória da placa uma vez que o programa for gravado.

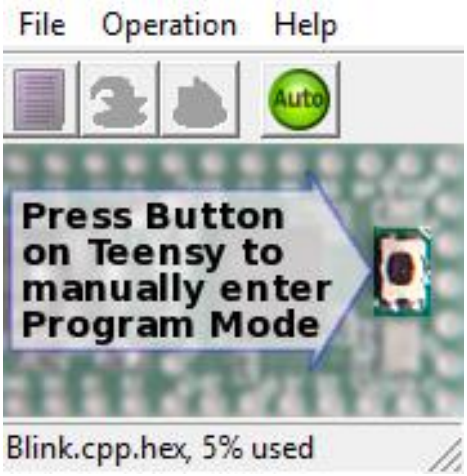


Figura 25 - Teensy Loader.

Para gravar o programa basta apertar o botão da placa. O *led* deve piscar em seguida.

Todas as bibliotecas instaladas se encontram no diretório mostrado na Figura 26. É importante notar que a IDE Arduino possui três locais possíveis de instalação de bibliotecas figuras 26, 27 e 28.

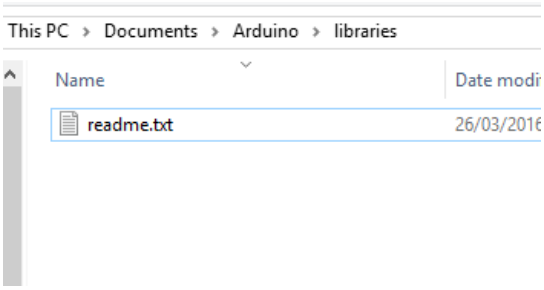


Figura 26 - Diretório \Documents\Arduino\libraries.

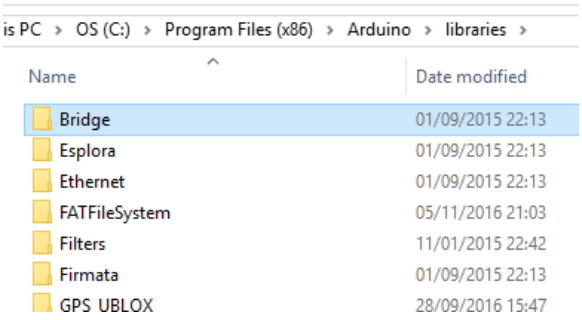


Figura 27 - Diretório C:\Program Files (x86)\Arduino\libraries.

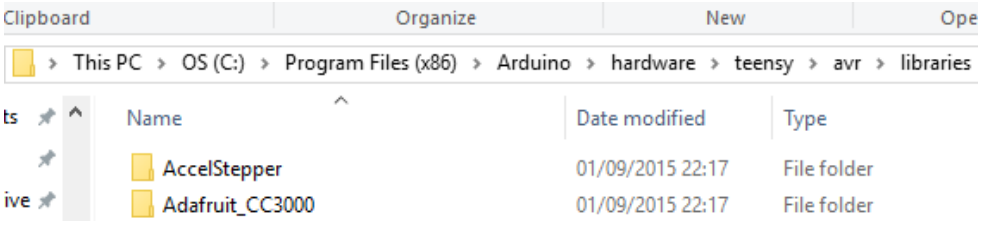


Figura 28 - Diretório C:\Program Files (x86)\Arduino\hardware\teensy\avr\libraries.

É necessário certificar-se que as bibliotecas usadas se encontram somente em umas dessas três pastas para evitar duplicações e erros de compilação.

3.2 BIBLIOTECA LAICANSAT

Existem vários aspectos a considerar na concepção do *firmware* projetado. Levando em consideração que se trata de um projeto open *hardware* feito a partir de tecnologias acessíveis, tem-se a vantagem de que os conhecimentos necessários para o desenvolvimento das aplicações computacionais já estão disponíveis a partir de plataformas que permitem o compartilhamento de conhecimento.

No caso do sistema embarcado da plataforma LAICAnSat, essa foi uma grande vantagem na maioria dos casos como será mostrado em seguida.

O *firmware* projetado para o sistema embarcado não só cumpre os requisitos para missão mas proporciona um código mais simples e conciso permitindo um entendimento rápido e a facilidade de poder se adaptar a qualquer mudança no hardware que possa vir a acontecer.

O firmware faz uso de conceitos de programação orientada a objeto na criação de uma biblioteca exclusiva para a plataforma responsável pela inicialização e chamada dos sensores.

Abaixo serão mostradas como são atingidos cada objetivo requerido de cada missão citada anteriormente.

3.2.1 CONFIGURANDO OS SENSORES

A primeira versão *firmware* que foi usada no protótipo durante a missão LAICAnSat-1, [7], fazia uso dos códigos disponíveis para os sensores sem nenhum refinamento o que resultou em um código complexo e poluído. Essa seção mostrará o processo de otimização desse código com a criação de uma biblioteca exclusiva para a plataforma LAICAnSat de maneira a fazer uso das características do *hardware* de forma otimizada.

O sistema embarcado é composto por cinco sensores, sendo eles:

- L3GD20 - Giroscópio
- LSM303DLHC - Acelerômetro
- BMP180 – Barômetro e termômetro

- SHT15 - Termômetro e higrômetro
- MS5611 – Termômetro

Foi possível encontrar para todos esses sensores, bibliotecas já prontas de código aberto para serem usadas como base para a criação da biblioteca LAICAnSat [16], [17], [18], [19], [20].

Como visto na seção 2.1.1, o computador de bordo faz uso de quatro protocolos de comunicação, sendo eles UART, SPI, I²C e 2-Wire. Analisando as tabelas 2.1, 2.2 e 2.3 é possível notar que existem quatro sensores conectados ao mesmo protocolo na mesma porta. Como o protocolo I²C faz uso de endereços para diferenciar os sensores na hora da comunicação é necessário certificar-se que não haja conflito entre eles.

O ponto crítico são os sensores BMP180 e MS5611, que possuem endereços em comum, mas com há a possibilidade de escolha entre dois endereços para o sensor MS5611 pode-se evitar o conflito. Nesse caso foi necessário verificar o código fonte que controla a operação de cada um destes dois sensores [16], [18] para garantir que os sensores BMP180 e MS5611 possuíssem endereços de protocolo I²C diferentes. Para tal, abre-se primeiramente os arquivos cabeçalhos BMP180.h e MS5611.h. Neles foram encontrados os valores da Figura 29, resultando num conflito de sensores.

```
#define BMP180_ADDR 0x77 // 7-bit address

#define MS5611_ADDRESS                (0x77)
```

Figura 29 - Definição dos endereços I²C.

Como mostrado na Tabela 2.2 o endereço do BMP180 é 0x77, no entanto o sensor MS5611 possui duas opções de endereçamento, 0x76 e 0x77, para evitar um conflito de comunicação entre os sensores resultando na incapacidade do microcontrolador diferenciar-los, o endereço 0x77 da variável MS5611_ADDRESS, foi alterado para 0x76. Esse ajuste do endereço I²C via *software* só foi possível porque as conexões de hardware do sensor MS5611, que habilitam a troca de

endereçamento, foram conectadas quando o computador de bordo foi projetado em [9].

As duas opções de endereçamento disponíveis para o sensor MS5611 são ajustáveis através do seu circuito de alimentação. A Figura 30 mostra o diagrama de blocos que representa o diagrama de blocos do sensor MS5611.

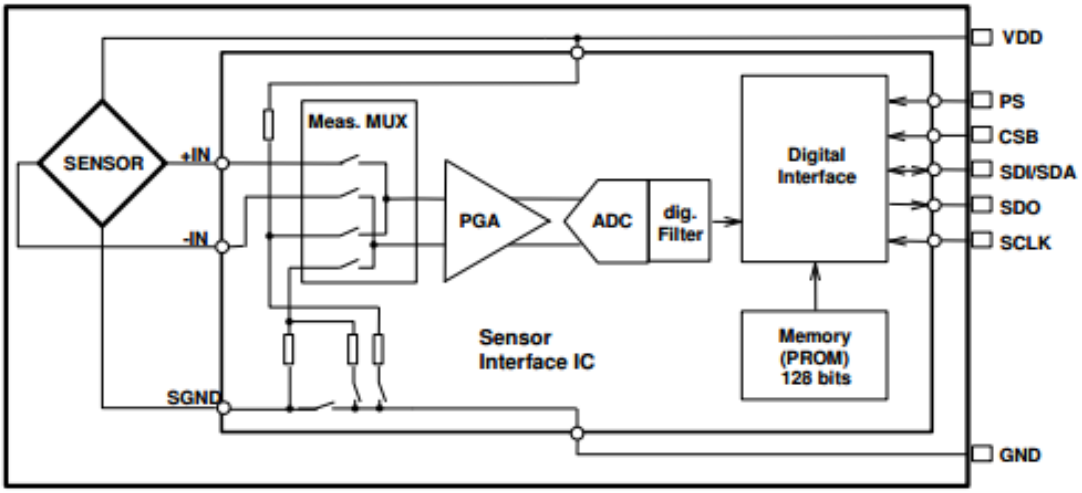


Figura 30 - Diagrama de blocos MS5611 (Fonte: [58]).

O endereço do MS5611 é 111011Cx, onde C é o valor complementar do pino CSB sendo C, o LSB do endereço I²C. É possível utilizar dois sensores com dois endereços diferentes no barramento I²C, 0x77 ou 0x76. Para isto basta que o pino CSB seja conectado a VDD ou GND. A Figura 31 mostra o circuito de comunicação entre um microcontrolador e o MS5611 [58].

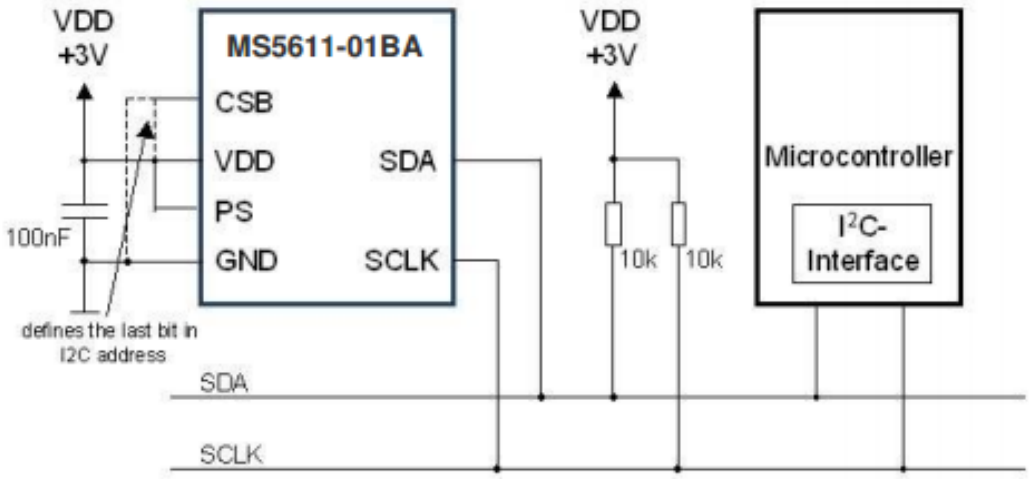


Figura 31 - Circuito típico de aplicação do MS5611 (Fonte: [58]).

As linhas pontilhadas na parte esquerda da Figura 31, definem o último *bit* para o endereço I²C. Se o pino CSB for ligado ao VDD, obtém-se o endereço 0x76. Se conectado ao GND é obtido o valor de endereço 0x77.

Existe somente mais um conflito, mas este será mostrado na próxima seção pois não ocorria entre sensores, mas entre um sensor o cartão de memória.

Depois de garantido que não existia mais nenhum outro conflito partiu-se para a a otimização do código que faz o acionamento dos sensores.

Usando as bibliotecas disponíveis como referência, foi possível construir uma biblioteca própria que englobasse todos os sensores que fazem parte da placa. Nas figuras 32, 33 e 34 é mostrado o resultado final da criação da classe *laicansat*.

laicansat1_2

```
1
2  #include <laicansat.h>
```

Figura 32 - Definição dos endereços I²C.

```
laicansat.bar->begin();
laicansat.thermo->begin(BMP180_THERMOMODE);
laicansat.thermo->begin(MS5611_THERMOMODE);
laicansat.thermo->begin(SHT15_THERMOMODE);
laicansat.accel->begin();
laicansat.gyro->begin();
laicansat.hygro->begin();
```

Figura 33 - Inicialização dos sensores.

```
double temperatureBMP180 = laicansat.thermo->getTemperatureBMP180();
double temperatureMS5611 = laicansat.thermo->getTemperatureMS5611();
double temperatureSHT15 = laicansat.thermo->getTemperatureSHT15();
double pressureBMP180 = laicansat.bar->getPressure();
double humiditySHT15 = laicansat.hygro->getHumidity();
laicansat.accel->getAcceleration(arrayAccel);
laicansat.gyro->getAngularSpeed(angularSpeeds);
```

Figura 34 - Chamada dos sensores.

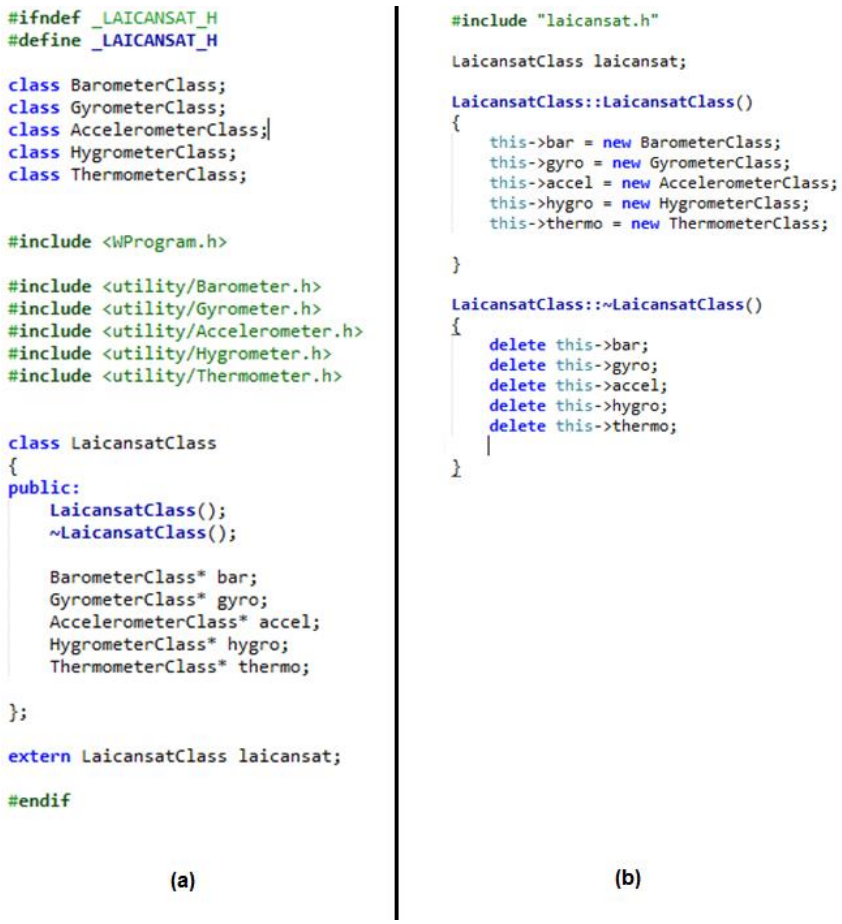
Nas figuras 33 e 34 são mostradas a inicialização e chamada de cada sensor. Nota-se que todo o processo de acionamento se resume a poucas linhas de código.

Na Figura 33, a linha de comando laicansat.XXX->begin() é responsável por ativar e checar o funcionamento dos componentes da placa, gerando uma mensagem de confirmação que pode ser vista no terminal serial. As palavras que aparecem depois do ponto em laicansat.XXX, fazem referência a que tipo de sensor está sendo chamado. Para fazer a captura de dados dos sensores é usado o comando laicansat.XXX->getXXXXXXX(), onde também foram escolhidos termos reservados para cada sensor. Na Tabela 3.1 são vistas as palavras reservadas usadas.

Tabela 3.1 Classes da biblioteca laicansat.

Palavra Reservada	**Tipo de sensor**
bar	BMP180 – barômetro
thermo	BMP180, MS5611, SHT15 – termômetro
hygro	SHT15 – higrômetro
accel	LSM303DLHC – acelerômetro
gyro	L3GD20 – giroscópio

A ideia central é que a classe *laicansat* herde as características de outras cinco classes que por sua vez herdam características de outras cinco bibliotecas, uma para cada tipo de sensor. Encapsulando o código das bibliotecas base em um é possível obter um código mais limpo conforme visto nas figuras 32, 33 e 34. Na Figura 35 é mostrada como foi feita a declaração da biblioteca mostrando seus arquivos cabeçalho.



```h
#ifndef _LAICANSAT_H
#define _LAICANSAT_H

class BarometerClass;
class GyrometerClass;
class AccelerometerClass;
class HygrometerClass;
class ThermometerClass;


#include <WProgram.h>

#include <utility/Barometer.h>
#include <utility/Gyrometer.h>
#include <utility/Accelerometer.h>
#include <utility/Hygrometer.h>
#include <utility/Thermometer.h>


class LaicansatClass
{
public:
    LaicansatClass();
    ~LaicansatClass();

    BarometerClass* bar;
    GyrometerClass* gyro;
    AccelerometerClass* accel;
    HygrometerClass* hygro;
    ThermometerClass* thermo;

};

extern LaicansatClass laicansat;

#endif
```

(a)

```cpp
#include "laicansat.h"

LaicansatClass laicansat;

LaicansatClass::LaicansatClass()
{
    this->bar = new BarometerClass;
    this->gyro = new GyrometerClass;
    this->accel = new AccelerometerClass;
    this->hygro = new HygrometerClass;
    this->thermo = new ThermometerClass;

}

LaicansatClass::~LaicansatClass()
{
    delete this->bar;
    delete this->gyro;
    delete this->accel;
    delete this->hygro;
    delete this->thermo;

}
```

(b)

Figura 35 - Declaração da interface (a) e Implementação da classe (b).

Na Figura 35 tem-se a declaração da classe *laicansat* e sua implementação. É possível notar a referência às classes:

- BarometerClass;
- GyrometerClass;
- AccelerometerClass;
- HygrometerClass;
- ThermometerClass;

Essas por sua vez herdam as características criadas pelas classes usadas para definir cada sensor. Olhando a declaração da classe *BarometerClass*, fez-se o uso do arquivo BMP180.h da biblioteca já disponível para o sensor BMP180. A função da classe *BarometerClass,* é somente inicializar o sensor BMP180 e fazer a chamada quando necessário, vide Figura 36.

```h
#ifndef _BAROMETER_H
#define _BAROMETER_H

#include "BMP180.h"


#include "../laicansat.h"


class BarometerClass
{
public:
    SFE_BMP180 *bmp180;

    int pressureBaseline;

    void begin();

    double getPressure();



};

#endif
```

(a)

```cpp
#include "Barometer.h"

void BarometerClass::begin()
{

  this->bmp180 = new SFE_BMP180();

  while(!this->bmp180->begin())
  {
    Serial.println("BMP180 could not start. Trying aga
    delay(500);
  }

  Serial.println("BMP180 initiation successful!");

  // Get the baseline pressure:
  this->pressureBaseline = getPressure();

}

double BarometerClass::getPressure()
{

    char status;
  double P, dummyT;

  status = this->bmp180->startPressure(3);
  if (status != 0)
  {
    // Wait for the measurement to complete:
    delay(status);


    status = this->bmp180->getPressure(P,dummyT);

    return P;
  }
  return -1;

}
```

(b)

Figura 36 - Barometer.h (a) e Barometer.cpp (b).

Usando o conceito de herança de classes da programação orientada a objetos foi possível permitir uma classe herdar todo o comportamento e atributos de outra classe. A classe que herda é chamada de subclasse e a que fornece é chamada de superclasse. No construtor da subclasse passa-se parâmetros para o construtor da classe pai [21], [22], [23]. No caso do *Barometer.h* a variável SFE_BMP180 *bmp180, passa parâmetros para a superclasse SFE_BMP180. O objeto bmp180 herda todos os atributos e métodos que vem da classe original SFE_BMP180, permitindo que as funções da classe pai sejam acessadas através da classe filha. Assim é possível realizar o encapsulamento de uma classe mais complexa em uma classe simples dedicada à uma tarefa específica.

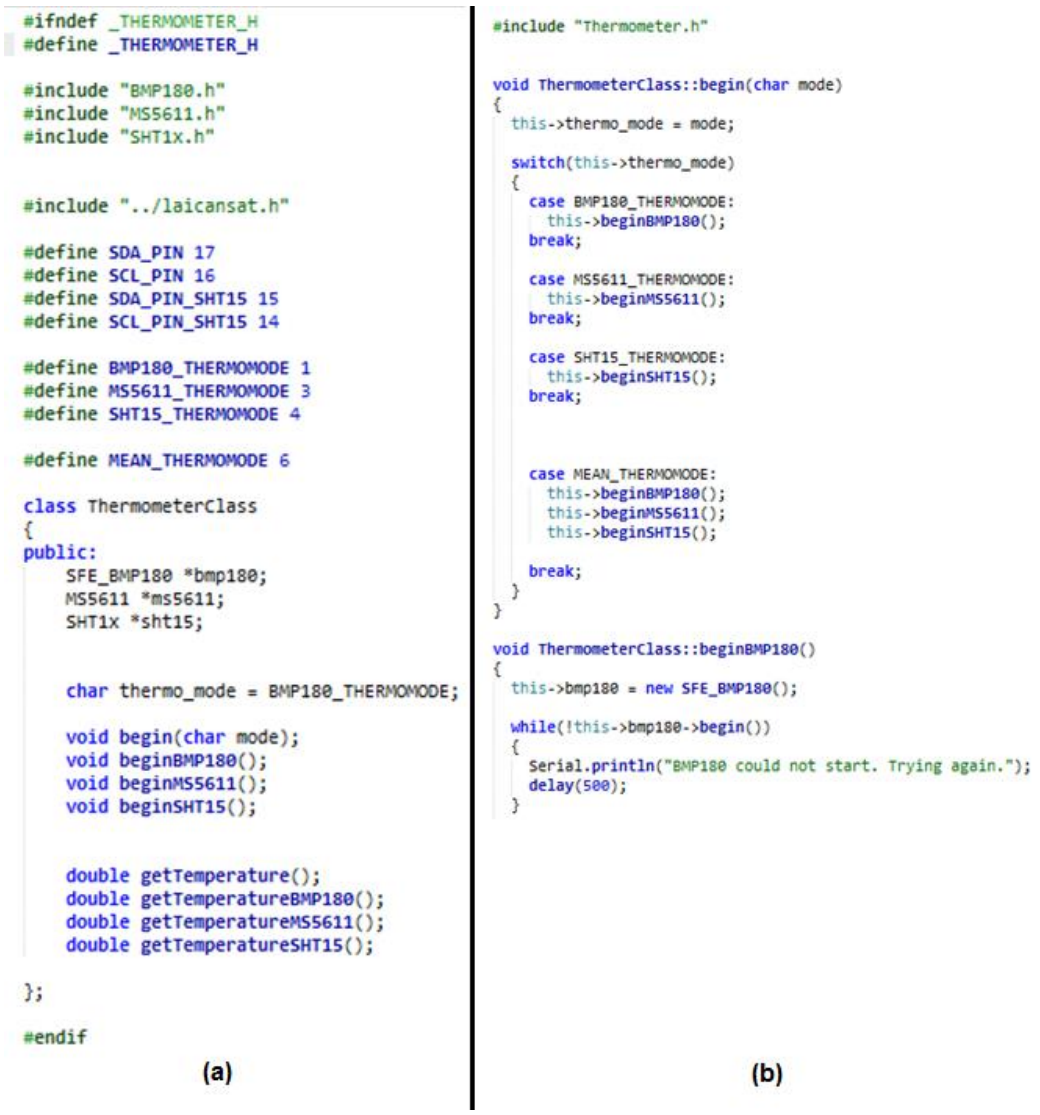
```
#ifndef SFE_BMP180_h
#define SFE_BMP180_h

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

class SFE_BMP180
{
    public:
        SFE_BMP180(); // base type

        char begin();
            // call pressure.begin() to
            // returns 1 if success, 0 i

        char startTemperature(void);
```

Figura 37 - Classe pai arquivo BMP180.h

Portanto, a classe SFE_BMP180, ilustrada na Figura 37, com seus atributos e métodos, foram encapsuladas na classe *BarometerClass* que por sua vez foi encapsulada na classe *laicansat.*

Esta mesma lógica se repete na criação das classes *AccelerometerClass, GyrometerClass, HygrometerClass, ThermometerClass.* A vantagem do encapsulamento nesse caso é a otimização do código mantendo visíveis somente os métodos importantes para a aplicação em questão, e também a possibilidade de englobar sensores diferentes, mas com características semelhantes, em um mesmo tipo de classe como acontece no caso dos termômetros.

Na classe *ThemometerClass* são usadas três superclasses (Figura 38). Como haviam três sensores sendo usados como termômetro, foi necessário implantar o funcionamento dos três na mesma subclasse para tornar o código mais conciso. Uma de suas características que permite que o código das figuras 31 e 32 seja conciso é a possibilidade de usar as variáveis BMP180_THERMOMODE, MS5611_THERMOMODE, SHT15_THERMOMODE sendo possível selecionar especificamente o sensor o qual deseja ativar usando o comando laicansat.thermo->begin(), inserindo a variável entre os parênteses.



```h
#ifndef _THERMOMETER_H
#define _THERMOMETER_H

#include "BMP180.h"
#include "MS5611.h"
#include "SHT1x.h"


#include "../laicansat.h"

#define SDA_PIN 17
#define SCL_PIN 16
#define SDA_PIN_SHT15 15
#define SCL_PIN_SHT15 14

#define BMP180_THERMOMODE 1
#define MS5611_THERMOMODE 3
#define SHT15_THERMOMODE 4

#define MEAN_THERMOMODE 6

class ThermometerClass
{
public:
    SFE_BMP180 *bmp180;
    MS5611 *ms5611;
    SHT1x *sht15;


    char thermo_mode = BMP180_THERMOMODE;

    void begin(char mode);
    void beginBMP180();
    void beginMS5611();
    void beginSHT15();


    double getTemperature();
    double getTemperatureBMP180();
    double getTemperatureMS5611();
    double getTemperatureSHT15();

};

#endif
```

(a)

```cpp
#include "Thermometer.h"


void ThermometerClass::begin(char mode)
{
  this->thermo_mode = mode;

  switch(this->thermo_mode)
  {
    case BMP180_THERMOMODE:
      this->beginBMP180();
    break;

    case MS5611_THERMOMODE:
      this->beginMS5611();
    break;

    case SHT15_THERMOMODE:
      this->beginSHT15();
    break;



    case MEAN_THERMOMODE:
      this->beginBMP180();
      this->beginMS5611();
      this->beginSHT15();

    break;
  }
}

void ThermometerClass::beginBMP180()
{
  this->bmp180 = new SFE_BMP180();

  while(!this->bmp180->begin())
  {
    Serial.println("BMP180 could not start. Trying again.");
    delay(500);
  }
```

(b)

Figura 38 - Thermometer.h(a) e Thermometer.cpp(b).

Os da Figura 38 acima mostram a implementação da classe *ThermometerClass* que permite o encapsulamento das classes SHTX, MS5611 e SE_BMP180.

3.2.2 CONFIGURANDO CARTÃO DE MEMÓRIA

Para a configuração correta do cartão de memória foram necessárias a instalação das bibliotecas mostradas na Figura 39 [24], [25], [26]:

```
#include <SD.h>
#include <SD_t3.h>
#include <SPI.h>
```

Figura 39 - Bibliotecas necessárias.

Declarou-se as variáveis para sinalizar para o protocolo SPI, onde o cartão SD está conectado e também para inicializar os arquivos em que são salvos os dados coletados como mostrados na Figura 40.

```
const int chipSelect = 20;
File dataFileGNSS;
File dataFile;
```

Figura 40 - Variáveis SS e de arquivo.

O protocolo SPI usa quatro canais diferentes para a comunicação com o cartão de memória. A variável *chipSelect* na Figura 40 corresponde canal *SS* (*Slave Select*) que é usado para selecionar o dispositivo que o microcontrolador deseja acionar.

Foi preciso ajustar manualmente os pinos *MOSI, MISO, SCK* (*clock*) [27], [28], pois a placa faz uso de pinos diferentes dos que são usados como padrão por outras placas. Na Figura 41 é mostrado como esse ajuste deve foi feito.

```
SPI.setMOSI(7);
SPI.setMISO(8);
SPI.setSCK(14);
```

Figura 41 - Configuração do Protocolo SPI.

```
Serial.print("Initializing SD card...");
pinMode(20, OUTPUT);
pinMode(led, OUTPUT);

if (!SD.begin(chipSelect)) {
   Serial.println("Card failed, or not present");
   while (1) ;
 }
Serial.println("card initialized.");
```

Figura 42 - Inicialização do cartão de memória.

Depois de inicializado o cartão de memória (Figura 42), os arquivos para serem salvos os dados dos sensores e do módulo GNSS são criados. Como mostra a Figura 43.

```
dataFileGNSS = SD.open("gnss11.txt", FILE_WRITE);
          if (! dataFileGNSS) {
            Serial.println("error opening datalog.txt");
            while (1) ;
          }
//CRIA E ABRE ARQUIVO DE DADOS
 dataFile = SD.open("datalog.csv", FILE_WRITE);
  if (!dataFile) {
    Serial.println("error opening datalog.txt");
    while (1) ;
  }
```

Figura 43 - Criação dos arquivos de dados.

As variáveis *dataFileGNSS* e *dataFile* correspondem às variáveis que receberem os dados a serem passados para os arquivos *gnss.txt* e *datalog.txt* respectivamente. Arquivos estes que guardaram os dados do módulo GNSS e sensores.

A biblioteca SD é responsável por abrir e fechar os arquivos dentro do cartão de memória, permitindo a criação de arquivos com nomes de até seis caracteres e com extensão de até três caracteres [27]. A última versão dessa biblioteca permite trabalhar com múltiplos arquivos [24] tornando possível evitar a mistura dos dados brutos vindo do módulo GNSS com os dados vindos dos sensores em formato ASCII.

No que se refere aos sensores, foi necessário a formatação dos dados utilizando a estrutura mostrada na Figura 44, antes de salva-los no arquivo para que obedecessem ao padrão .csv (*comma separated values*) facilitando sua manipulação posterior em algum *software* para plotagem de gráficos.

```
int count = 0;
dataSensors += String (humiditySHT15);
dataSensors += ",";
dataSensors += String (temperatureBMP180);
dataSensors += ",";
dataSensors += String (temperatureMS5611);
dataSensors += ",";
dataSensors += String (temperatureSHT15);
dataSensors += ",";
dataSensors += String (pressureBMP180);
dataSensors += ",";

  for (count = 0; count < 3; count++) {

      dataSensors += String(arrayAccel[count]);
      if (count < 3)
       dataSensors += ",";
     }
   for (count = 0; count < 3; count++) {

  dataSensors += String(angularSpeeds[count]);
  if (count < 2)
  dataSensors += ",";
 }
```

Figura 44 - Formatação dos dados dos sensores.

Com os dados organizados prontos parem serem salvos no arquivo, os seguintes comandos escrevem os dados e os salvam na memória como mostrado na Figura 45 [29]. O comando *flush* o responsável pela escrita definitiva em arquivo, ao contrário do comando *println* que faz somente a transferência dos dados para um *buffer.*

```
dataFile.println(dataSensors);//ESCREVE OS DADOS
dataFile.flush();//SALVA OS DADOS
```

Figura 45 - Escrevendo e salvando os dados no arquivo.

Tanto o protocolo SPI, que lida com o cartão SD, quanto o protocolo 2-Wire que lida com o sensor de temperatura e umidade SHT15 estão conectados no mesmo pino para receber o sinal do *clock*, por essa razão é necessário reiniciar o cartão SD logo depois de cada chamada do sensor SHT15, como mostrado na Figura 46.

```
pinMode(chipSelect, OUTPUT);
if (!SD.begin(chipSelect)) {
  digitalWrite(led, LOW);
  while (1) ;
    }

 dataFile.println(dataSensors);//ESCREVE OS DADOS
 dataFile.flush();//SALVA OS DADOS
```

Figura 46 - Reinicialização do cartão SD.

Para isso, foi necessário fazer uma adaptação na classe SD no arquivo SD.cpp. Na linha 338 onde se lê a implementação do método boolean SDClass::begin(uint8_t csPin) foram adicionadas as linhas de código como exemplificado na Figura 47 [30]:

```
boolean SDClass::begin(uint8_t csPin) {

  if(root.isOpen()) {
    root.close();
  }

  return card.init(SPI_HALF_SPEED, csPin) &&
         volume.init(card) &&
         root.openRoot(volume);
}
```

Figura 47 - Modificação da Classe SD.

3.3.3 CONFIGURANDO MÓDULO GNSS

O módulo GNSS escolhido para ser usado na plataforma pertence à família LEA-M8T, modelo mais recente da companhia Ublox. É um módulo que permite trabalhar com multiconstelações, suportando uma taxa de até 14Hz de atualização de posição, EEPROM para salvar configurações, entre outras vantagens. A configuração é feita a partir do envio de mensagens da classe UBX-CFG-XXX do protocolo UBX. Onde XXX corresponde a subseção da mensagem de configuração que deve ser enviada ao receptor. Na Tabela 3.2 são mostrados os números que correspondem a cada subseção.

Tabela 3.2 Subseção das classes.

Número	Nome	Mensagens CFG	Descrição
0	PRT	UBX-CFG-PRT UBX-CFG-USB	Configuração de porta e USB
1	MSG	UBX-CFG-MSG	Configuração de mensagens (ativar/desativar)
2	INF	UBX-CFG-INF	Configuração das informações de saídas
3	NAV	UBX-CFG-NAV5 UBX-CFG-NAVX5 UBX-CFG-DAT UBX-CFG-RATE UBX-CFG-SBAS UBX-CFG-NMEA UBX-CFG-TMODE2	Configurações para Parâmetros de Navegação, Datum do Receptor, Medição e Taxa de Navegação, SBAS, protocolo NMEA e Modo de tempo (Apenas variantes de temporização e de produto FTS)
4	RXM	UBX-CFG-GNSS UBX-CFG-TP5 UBX-CFG-RXM UBX-CFG-PM2 UBX-CFG-ITFM	Configurações do GNSS, Configurações do Modo de Energia, Configurações de Pulso de Tempo, Configurações do Jamming / Interference Monitor
9	RINV	UBX-CFG-RINV	Configuração de inventário remoto
10	ANT	UBX-CFG-ANT	Configuração da antena
11	LOG	UBX-CFG-LOGFILTER	Configuração do logging

Para o propósito da missão LAICAnSat o receptor foi programado de modo a proporcionar o acesso às constelações GPS e GLONASS à uma taxa de 1Hz. Todas as mensagens do protocolo NMEA padrões na configuração de fábrica foram desabilitadas, deixando somente mensagens do protocolo UBX. Na Tabela 3.3 são apresentadas as mensagens que o receptor deve fornecer e seus respectivos significados [31].

Tabela 3.3 Configuração do receptor em formato decimal.

Mensagem	Significado
!UBX CFG-RATE 1000 1 1	Taxa de atualização em 1Hz
!UBX CFG-MSG 2 15 0 1 0 0 0 0	Habilita mensagens UBX RXM-RAWX na UART-1
!UBX CFG-MSG 2 13 0 1 0 0 0 0	Habilita UBX RXM-SFRBX na UART-1
!UBX CFG-MSG 13 3 0 1 0 0 0 0	Habilita mensagens TIM-TIM2
!UBX CFG-GNSS 0 32 32 1 0 8 16 0 1 0 1 1	Configura GPS 8-16 canais
!UBX CFG-GNSS 0 32 32 1 1 1 3 0 0 0 1 1	Desabilita SBAS
!UBX CFG-GNSS 0 32 32 1 2 0 0 0 0 0 0 1	Desabilita Galileo
!UBX CFG-GNSS 0 32 32 1 3 8 16 0 0 0 1 1	Desabilita BeiDou
!UBX CFG-GNSS 0 32 32 1 4 0 8 0 0 0 1 1	Desabilita IMES
!UBX CFG-GNSS 0 32 32 1 5 0 3 0 0 0 1 1	Desabilita QZSS
!UBX CFG-GNSS 0 32 32 1 6 8 14 0 1 0 1 1	Configura 8-14 canais GLONASS
!UBX CFG-MSG 240 0 0 0 0 0 0 0 !UBX CFG-MSG 240 1 0 0 0 0 0 0 !UBX CFG-MSG 240 2 0 0 0 0 0 0 !UBX CFG-MSG 240 3 0 0 0 0 0 0 !UBX CFG-MSG 240 4 0 0 0 0 0 0 !UBX CFG-MSG 240 5 0 0 0 0 0 0 !UBX CFG-MSG 240 8 0 0 0 0 0 0	Desabilita mensagens NMEA padrão
!UBX CFG-MSG 1 2 0 1 0 0 0 0 !UBX CFG-MSG 1 3 0 1 0 0 0 0 !UBX CFG-MSG 1 6 0 1 0 0 0 0 !UBX CFG-MSG 1 12 0 1 0 0 0 0	Habilita NAV-STATUS, NAV-SOL, NAV-VELNED, NAV-POLLSH
!UBX CFG-MSG 1 48 0 0 0 0 0 0 !UBX CFG-MSG 1 34 0 0 0 0 0 0	Desabilita NAV-CLOCK, NAV-SVINFO
!UBX CFG-NAV5 255 255 3 6 0 0 0 0 16 39 0 0 5 0 250 0 250 0 100 0 44 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Plataforma Dinâmica de voo <1g

A tecnologia de posicionamento Ublox suporta diferentes modelos de plataforma dinâmica mostradas na Tabela 3.4, para ajustar o mecanismo de navegação ao ambiente de aplicação esperado. Essas configurações de plataforma

podem ser alteradas dinamicamente sem precisar reiniciar o sistema. As configurações melhoram a interpretação das medições do receptor e assim fornecem uma saída de posição mais precisa. Configurar o receptor para um modelo de plataforma inadequado para o ambiente de aplicação é suscetível a perda de desempenho do receptor e precisão [10].

Tabela 3.4 Plataforma dinâmica.

Plataforma	**Descrição**
Portável	As aplicações com baixa aceleração, isto é, dispositivos portáteis. Adequado para a maioria das situações.
Estacionária	Usado em aplicações de temporização (antena deve ser estacionária) ou outras aplicações estacionárias. Velocidade limitada a 0 m / s.
Pedestre	Aplicações com baixa aceleração e velocidade, ou seja, como um pedestre se moveria. Aceleração baixa assumida.
Automotiva	Utilizado para aplicações com dinâmicas equivalentes às de um automóvel. Assumida baixa vertical aceleração.
Marítima	Recomendado para aplicações no mar, com velocidade vertical zero. Assumido velocidade vertical e altitude zero.
Voo <1g	Utilizado para aplicações com maior faixa dinâmica e maior aceleração vertical do que a de um carro. Correção 2D não suportada.
Voo <2g	Recomendado para ambientes aéreos típicos. Nenhuma correção de posição 2D suportada.
Voo <4g	Recomendado apenas para ambientes extremamente dinâmicos. Nenhuma correção de posição 2D suportada.

Tabela 3.5 Detalhes da Plataforma Dinâmica.

Plataforma	**Altitude Max**	**Velocidade horizontal max (m/s)**	**Velocidade vertical max (m/s)**
Portável	12000	310	50
Estacionária	9000	10	6
Pedestre	9000	30	20
Automotiva	6000	100	15
Marítima	500	25	5
Voo >1g	50000	100	100
Voo >2g	50000	250	100
Voo >4g	50000	500	100

Apesar da configuração ser mostrada em notação decimal na Tabela 3.2, a configuração do módulo é feita através de mensagens em formato hexadecimal enviadas do microcontrolador para o receptor GNSS. As mensagens do protocolo UBX obedecem ao formato mostrado na Figura 48.

Figura 48 - Estrutura da mensagem UBX (Fonte: [10]).

Os primeiros dois bytes compõem a estrutura, são os bytes de sincronização que são 0xB5 0x62 para todas as mensagens. Os dois bytes seguintes correspondem a Classe e ao ID da mensagem, respectivamente. Os próximos dois, dão o tamanho da carga útil da mensagem, escrito em *little endian*. Em seguida vem a carga útil de tamanho variável seguida por dois bytes de *checksum*. As mensagens mostradas na Tabela 3.2 fornecem os valores da carga útil para as mensagens que devem ser enviadas. A Classe e ID podem ser deduzidos a partir do nome de cada uma das mensagens, e o *checksum* é calculado automaticamente, como será mostrado a seguir.

A mensagem da classe CFG-RATE configura a taxa de atualização dos dados fornecidos pelo receptor. O exemplo da Figura 49 demonstra como é feito o envio das

mensagens em hexadecimal ao receptor. Cada mensagem é guardada em um vetor e esse vetor é transmitido ao receptor *byte* por *byte* através da função *sendUBX* [32], [33], [34], mostrada na Figura 50.

```
203   //--------------------------------------------------------------------------
204     //dados atualizados em 1Hz
205 Serial.println("Setting uBlox set CFG-RATE ");
206   uint8_t setCFG_RATE[] = {
207
208     0xB5,0x62,  0x06,0x08,  0x06,0x00,     0xE8,0x03,    0x01,0x00,0x01,0x00,
209
210   };
211   while(!gps_set_sucess)
212   {
213     sendUBX(setCFG_RATE, 12);
214     gps_set_sucess=getUBX_ACK(setCFG_RATE);
215   }
216   gps_set_sucess=0;
217
218   //--------------------------------------------------------------------------
```

Figura 49 - Criação e envio da mensagem CGF-RATE.

```
// Send a byte array of UBX protocol to the GPS
void sendUBX(uint8_t *MSG, uint8_t len) {
  for(int i=0; i<len; i++) {
    gpsSerial.write(MSG[i]);
    Serial.print(MSG[i], HEX);
    Serial.print(" ");
  }
  sendCheckSum(MSG+2/*remove header*/, len-2/*remove header*/);
  Serial.println();
}
```

Figura 50 - Função sendUBX.

Por sua vez, logo depois que todos os *bytes* foram enviados, o *checksum* é enviado através da função *sendCheckSum* Figura 51.

```
void sendCheckSum(uint8_t *MSG,uint8_t len ){
  uint8_t ckA = 0, ckB = 0;
  for (int i=0;i<len;i++){
    ckA = ckA + MSG[i];
    ckB = ckB + ckA;
  }

  gpsSerial.write(ckA);
  Serial.print(ckA, HEX);
  Serial.print(" ");

  gpsSerial.write(ckB);
  Serial.print(ckB, HEX);
  Serial.print(" ");
}
```

Figura 51 - Função sendCheckSum.

A mensagem da classe CFG-MSG é responsável por habilitar todas as mensagens que o receptor deve retornar, por isso o entendimento de como essa mensagem funciona é importante. Tomando o exemplo a mensagem da Figura 52:

```
0xB5,0x62,0x06,0x01,0x08,0x00,    0x02, 0x15,    0x00,0x01,0x00,0x00,0x00,0x00,
```

Figura 52 - CFG-MSG-RXM-RAWX.

Na mensagem os dois primeiros *bytes* são os *bytes* de sincronização 0xB5 e 0x62. Os bytes 0x06 e 0x01 são Classe e ID da mensagem CFG-MSG. 0x08 0x00 sinaliza que se tem uma carga útil de 8 *bytes*. Os próximos dois são responsáveis por enviar a classe e ID da mensagem que se deseja que o receptor transmita, nesse caso 0x02 0x015 corresponde à RXM-RAWX que contém informações necessárias para gerar arquivos de observação RINEX. Os últimos seis *bytes* correspondem a seis portas de entrada/saída disponíveis no receptor. Como mostrado na Figura 52 somente a UART-1 está habilitada para receber as mensagens RXM-RAWX. Como pode ser notado na Tabela 3.6 [10] pois o décimo *byte* tem o valor 0x01, correspondente à UART-1.

Tabela 3.6 Estrutura da mensagem CFG-MSG.

Byte	**Significado**
1 e 2	*Bytes* de sincronização
3	Classe
4	ID
5 e 6	Tamanho da carga útil
7 e 8	Classe e ID da mensagem a ser habilitada
9	Porta DDC
10	UART 1
11	UART 2
12	USB
13	SPI
14	Reservado

Outras duas mensagens importantes a serem habilitadas são NAV-POSLLH e NAV-VELNED. Ambas são necessárias para fornecer os dados de posição e direção que serão usadas no controle de trajetória da carga útil.

Depois de habilitadas as mensagens, a biblioteca UBX_parser [35] é responsável por fazer a tradução dos dados binários para dados que possam ser usados no controle. Esse processo é demostrado nas figuras 53 e 54.

```
class MyParser : public UBX_Parser {

   void handle_NAV_POSLLH(unsigned long iTOW,
            long lon,
            long lat,
            long height,
            long hMSL,
            unsigned long hAcc,
            unsigned long vAcc) {

        Serial.print("lat=");
        Serial.print(lat/1e7,7);
        LAT = lat;
        Serial.print(" deg lon=");
        Serial.print(lon/1e7,7);
        Serial.print(" deg");
        LON = lon;
```

Figura 53 - Manipulação da mensagem NAV_POSLLH.

```
   virtual void handle_NAV_VELNED(unsigned long iTOW,
        long velN,
        long velE,
        long velD,
        unsigned long speed,
        unsigned long gSpeed,
        long heading,
        unsigned long sAcc,
        unsigned long cAcc)
{
```

Figura 54 - Manipulação da mensagem NAV_VELNED.

No laço principal do código é necessário capturar os dados fornecidos pelo receptor durante um segundo sem interrupção para que a mensagem possa ser transmitida sem que seja corrompida. Para tal foi necessário a ajuda da seguinte estrutura mostrada na Figura 55:

```
int start = millis();
while (millis() - start < 1000)
   {
        if(gpsSerial.available() > 0){
            inByte = gpsSerial.read();
            parser.parse(inByte);
            dataStringGNSS  = String(inByte);
            Serial.print(dataStringGNSS);
            dataFileGNSS.print(dataStringGNSS);
```

Figura 55 - Leitura do canal serial UART-1.

O laço *while* mantém a leitura do canal serial durante um segundo, enquanto caractere por caractere são salvos numa *String* para depois serem salvos em um arquivo txt. A linha *parse.parse(inByte)* é responsável por passar os *bytes* recebidos para as funções manipuladoras das figuras 53 e 54.

3.3.4 HISTÓRICO DA ESTRATÉGIA DE CONTROLE

Nesta subseção vem trazer a implementação no *firmware* de uma estratégia de controle para a plataforma LAICAnSat desenvolvida em [38], que se baseia em alguns trabalhos pesquisados na literatura sobre o controle e guiamento de parapentes tanto de forma manual, quanto de forma autônoma. O histórico mostrado a seguir aborda esse tema buscando a experimentação de novas soluções ou validação de soluções mais clássicas.

O trabalho de [59] investiga a utilização de flaps como mecanismos para aumentar o arrasto nas pontas do perfil aerodinâmico do parapente. Usando esse mecanismo gera-se um sistema simples e eficiente de guiamento autônomo, que permite mediante o acionamento de freios aerodinâmicos, realizar curvas com diferentes raios de curvatura e modificar o ângulo de ataque do sistema.

Já o trabalho em [60] aborda uma estratégia para controlar tanto a curvatura do sistema quando a velocidade vertical. Este resultado é obtido através do controle lateral e longitudinal de um paramente utilizando deflexão assimétrica da superfície superior do paramente em contraste com o controle tradicional que usa deflexão das bordas de arrasto do parapente.

Trazendo uma alternativa para sistemas autônomos dependentes de sistemas de localização multi GNSS, o estudo apresentado em [61] vem propor a utilização de um rádio *beacon* como referencial para realizar a missão autônoma de guiamento para um ponto de pouso especifico, no caso do sinal do módulo muilt GNSS ser bloqueado.

O trabalho em [62] possui maior influência nos resultados desenvolvidos na identificação de uma estratégia de controle para a plataforma LAICAnSat desenvolvida em [38] e implementada do *firmware.* O trabalho em [62] utiliza-se a teoria dos *Dubing Paths* para criar o parâmetro de margem de altitude, parâmetro que busca quantificar a quantidade de energia disponível para a realização de manobras durante a missão. É apresentada uma estratégia híbrida que utiliza dois métodos para

gerar trajetórias, em que o método utilizado é escolhido de acordo com a margem de altitude atual do sistema. A trajetória pode ser redefinida em cada ciclo de operação.

Utilizando os trabalhos descritos acima como base o estudo feito em [38] analisou e desenvolveu, através de *softwares* matemáticos uma estratégia de controle autônomo para a descida da plataforma LAICAnSat. Foi feito o equacionamento do modelo dinâmico do sistema e os passos para a simplificação do modelo dinâmico para o modelo cinemático, considerando um sistema de coordenadas fixado no vento. Também foi descrita em [38] a estratégia para elaboração da trajetória de descida, e o algoritmo de controle desenvolvido para a missão LAICAnSat-3 além de terem sido feitas diversas simulações para a validação do sistema de controle.

Na próxima sessão será mostrado o resultado obtido no estudo feito em [38], no qual o controle PID implementado no *firmware* é baseado.

3.3.5 IMPLEMENTAÇÃO DO CONTROLE PID NO *FIRMWARE*

O projeto do *firmware* foi desenvolvido em paralelo ao projeto da estratégia de controle de pouso do módulo LAICAnSat que foi tema do Trabalho de graduação do Yago Henrique Melo Honda intitulado “Análise e controle da trajetória do LAICAnSat-3” [38].

Para o sucesso da missão LAICAnSat é imprescindível que a carga útil possa ser recuperada após o pouso. Como a plataforma atinge altitudes elevadas, é necessário que sua trajetória de descida seja guiada, de forma que, o pouso seja efetuado em uma área próxima à estação base. Assim, a estratégia de descida da plataforma precisa conter algumas etapas básicas de forma que o resgate da carga útil seja sempre possível.

As etapas da trajetória de pouso são divididas em *inicialization, loiter* e *final approach*.

A fase *inicialization*, que corresponde a fase de inicialização, busca aproximar a plataforma de uma área circular 10 vezes o tamanho do raio mínimo de curvatura do sistema, que corresponde à área onde deve ser realizado o pouso. O sistema de

navegação, usando os sensores da placa para recolher as informações necessárias de velocidade e posição, aciona o sistema de controle para que este comece a atuar na plataforma corrigindo seu posicionamento. Até então a plataforma desce de forma livre.

Entrando na fase *loiter,* o sistema faz a aproximação tentando reduzir a altitude usando uma série de manobras em espiral baseadas em Dubins Paths [36], [37] ou seja, uma manobra que visa conectar dois pontos pela menor curva possível, a fim de evitar que se deixe a área de pouso especificada.

Já na fase *final approach*, o sistema começa a se preparar para a aproximação final, o sistema de controle começa a atuar de forma mais intensa, onde a taxa de curvatura e o esforço de controle são levados ao máximo. De forma que o erro entre a posição desejada e a posição real seja mínimo. A Figura 56 exemplifica como o sistema de controle age para atingir os objetivos das etapas de pouso mostrando a trajetória da descida.

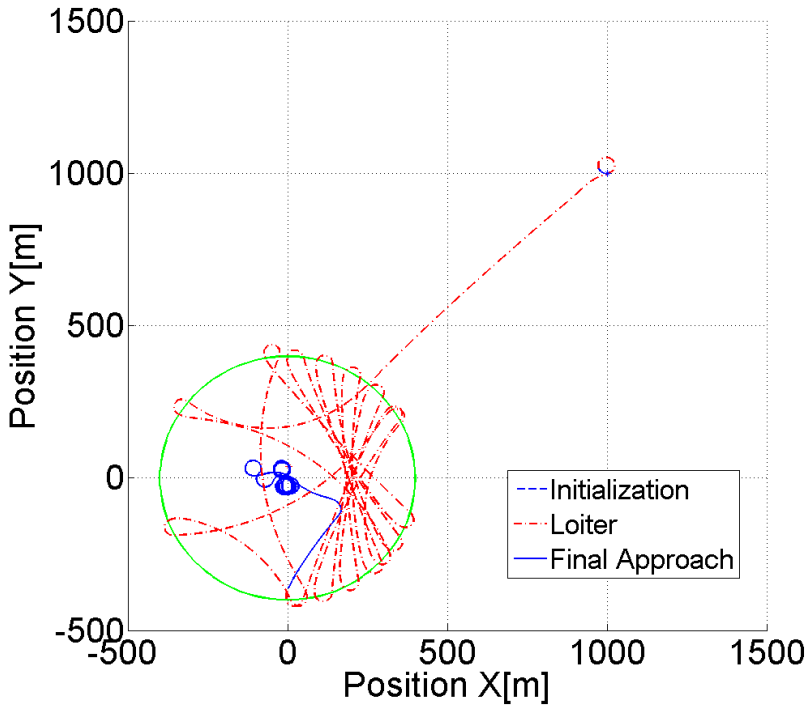


Figura 56 - Trajetória em 3D sem o efeito do vento (Fonte: [38]).

Para o modelo do sistema de controle [38] foi adotado um referencial cinemático que busca observar a posição do sistema em um plano XY. A partir de coordenadas de um ponto qualquer nesse plano, é possível determinar o ângulo ψ_{ref} que esse ponto faz com a posição atual do sistema. Sendo assim é possível

descrever a estratégia de controle para a descida, fazendo a comparação do ângulo ψ da direção da velocidade com o ângulo ψ_{ref} da reta que liga a plataforma ao ponto de pouso. O objetivo da estratégia de controle é anular a diferença entre esses dois ângulos.

A partir da análise das equações cinemáticas que descrevem o sistema definidas em [38], pode-se assumir ψ como a variável controlável. Então é necessário definir o sinal de referência que se adeque à estratégia de controle.

Para isso, utiliza-se o a comparação da posição desejada com a posição final, gerando um ângulo de saída ref que é usado como referência para minimizar o erro do controlador. Na Equação (3.1) pode ser visto como calcular matematicamente o sinal de referência, onde y e x são as posições atuais e yf e xf são as posições finais de pouso previamente definidas:

$$ref = tan^{-1}\frac{(y - y_f)}{(x - x_f)} \tag{3.1}$$

O tipo de controle escolhido foi do tipo PID. Esse tipo de controle calcula a saída do atuador através da soma de cálculos proporcional, integral e derivativo. Ele é inserido em um sistema com realimentação que busca continuamente calcular o valor do erro entre a referência e o sinal de saída do sistema, buscando minimiza-lo ajustando a variável de controle de modo que ela convirja para o sinal de referência. [48].

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{3.2}$$

Onde:

- · Kp é o ganho proporcional;
- · Ki é o ganho integral;
- · Kd é o ganho derivativo;
- · e é o erro;
- · t é o tempo;
- · τ é o tempo de integração;

Foi escolhido utilizar a ferramenta *Simulink* do MATLAB para modelagem, simulação e análise de sistemas dinâmicos envolvidos no projeto, usando os resultados mostrados em [8], a fim de propor uma estratégia para uma lei de controle que atuasse na trajetória da descida. O diagrama de blocos, Figura 57, representa o esquemático do sistema utilizado para a simulação do controle.

O diagrama do sistema de controle consiste em um gerador de referência, um controlador PID, o atuador eletromecânico e a planta. O atuador linear vai aplicar o esforço de controle no sistema, em seguida os dados do sistema são realimentados para gerar o sinal de referência baseado nas coordenadas do local de pouso para controlar a direção do parapente.

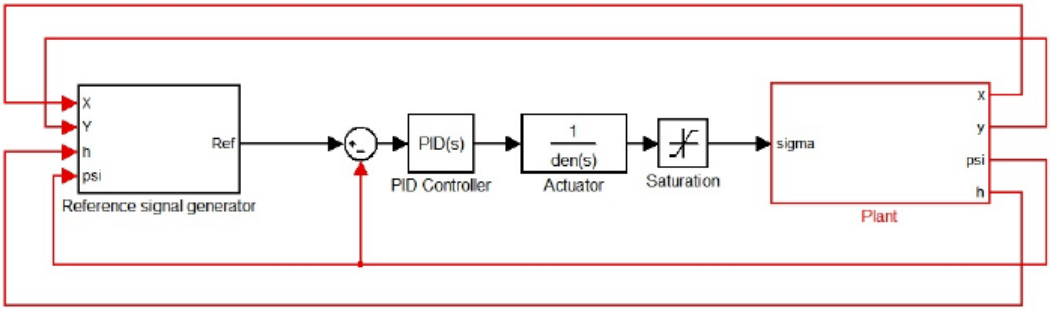


Figura 57 - Esquemático para simulação (Fonte: [38]).

Para a determinação dos ganhos do controlador e avaliar o seu comportamento foi feita uma simulação sem considerar o efeito dos ventos. Na Figura 58 é possível observar como o controle de age nas fases da descida explicadas anteriormente. Atuando mais levemente da fase *loiter* e mais agressivamente no *final approach,* a fim de garantir que o pouso convirja para o ponto final.

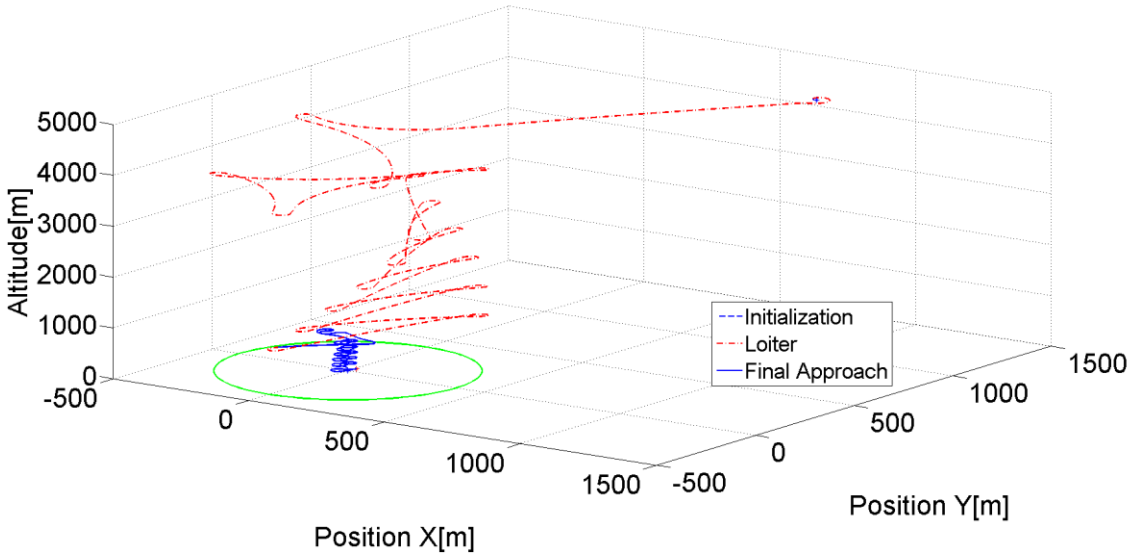


Figura 58 - Simulação do controle sobre efeito do vento. (Fonte: [38])

A Simulação do sistema com o auxílio do MATLAB, permitiu a determinação dos valores das constantes Kp, Ki e Kd do controlador e do coeficiente N do filtro passa-baixa, implementado junto à componente derivativa do controlador, para reduzir o efeito de ruídos.

Tabela 3.7 Ganhos do controlador PID [38].

Tipos de Grandeza	**Valores**
Ganho $\boldsymbol{K_P}$	-0.33572198
Ganho $\boldsymbol{K_i}$	-0.00003604
Ganho $\boldsymbol{K_d}$	1.30119804
Coeficiente de filtro N	0.25800991

Já na simulação que considera o efeito do vento (Figura 59), foram utilizados os mesmos parâmetros iniciais e os mesmo ganhos do controlador PID, para avaliar a robustez do sistema de controle projetado.

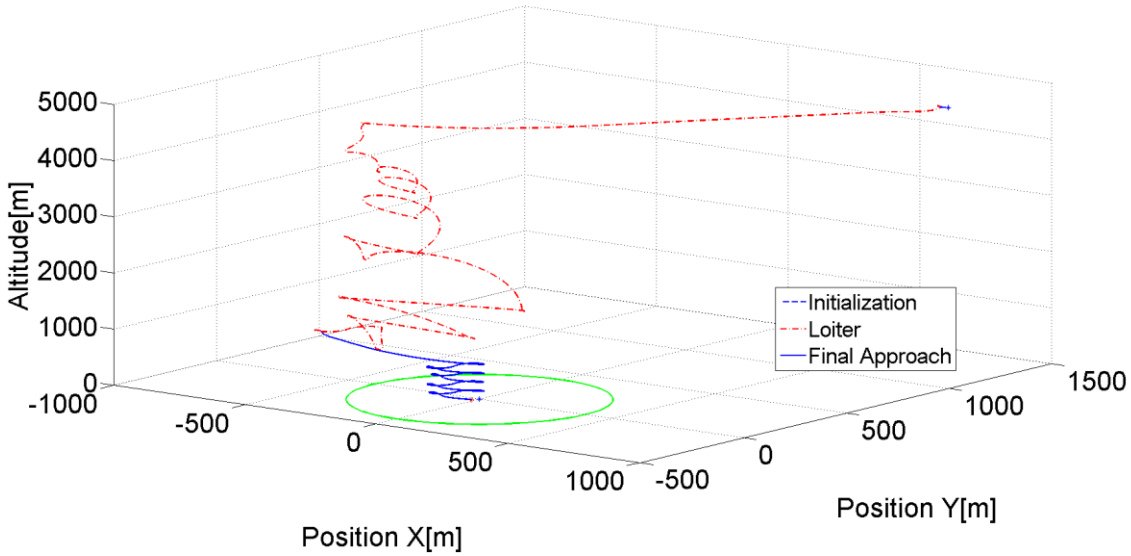


Figura 59 - Simulção da trajetória com efeito do vento (Fonte: [38]).

Como pode-se observar a trajetória tem um comportamento mais irregular devido ao efeito do vento, mas mantendo o ponto final dentro da área desejada.

Investigando o bloco do controlador PID no *Simulink* (Figura 60), nota-se que o algoritmo usado para o cálculo do controle, usa a Equação (3.1), na qual o diferenciador vem em série com um filtro.

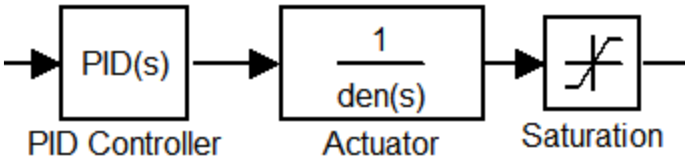


Figura 60 - Bloco PID *Simulink* (Fonte: [38]).

$$P + I\frac{1}{s} + Ds\,\frac{N}{s+N} \tag{3.1}$$

O uso de um diferenciador puro não é apropriado, pois quando ocorre uma mudança brusca na referência ou uma perturbação, a diferenciação resulta num sinal de controle teoricamente infinito.

Para evitar este sinal de controle errático, a maioria dos pacotes de software PID e módulos de hardware adicionam um filtro ao diferenciador. A filtragem é

particularmente útil num ambiente ruidoso [39]. Nas figuras 61 e 62 observa-se a influiência do fildro no sinal da derivada. A Figura 59 mostra o sinal de controle obtido através da implementação do controle no *firmware* utilizando um algoritmo que implementava o PID sem a adição de um filtro. A Figura 62, mostra o sinal de controle obtido na simulação do MATLAB onde há filtragem da componente derivativa do controlador.

Este é o sinal ideal que deve ser usado como parâmetro para a implementação do controle no *firmware.*

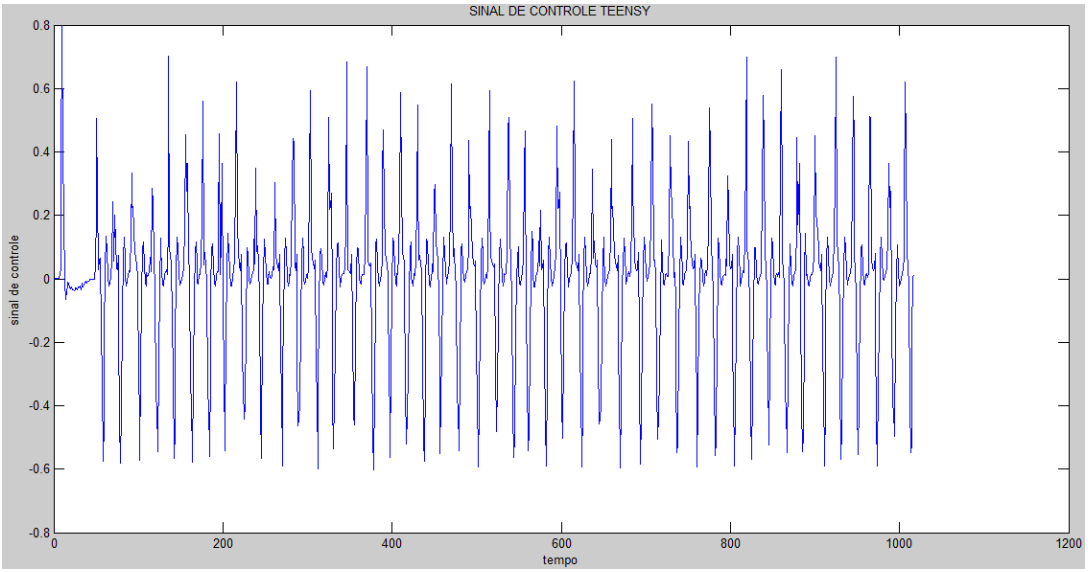


Figura 61 - Sinal de controle ruidoso implementado do firmware.

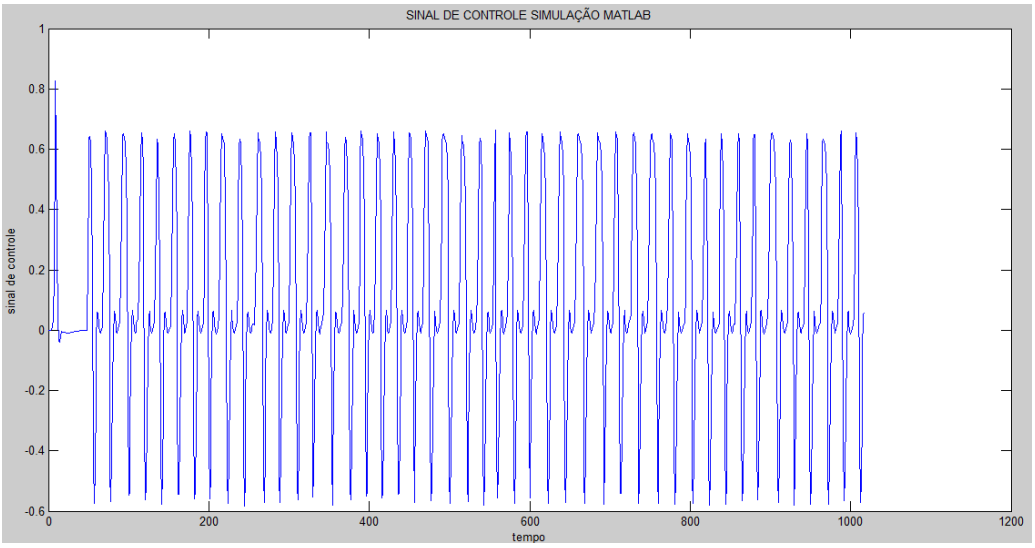


Figura 62 - Sinal de controle MATLAB.

Para permitir que o sistema embarcado realizasse o controle de trajetória do pouso, um algoritmo que implementa o controle PID foi adicionado ao *firmware*. O algoritmo tem como dados de entrada a longitude e latitude da posição atual fornecidas pelo receptor GNSS, o valor do ângulo da velocidade (*heading)*, e as coordenadas do ponto de pouso que devem ser estabelecidas antes do voo.

```
double reference(double Xi, double Yi) {

  double Xf=0; // coordenadas finais são escolhidas previamente
  double Yf=0;

  double y = Yi - Yf;
  double x = Xi- Xf;


  double ref = atan2(y, x); // arc tangent of y/x


  return ref;
}
```

Figura 63 - Calculo da referência.

Assim como nas simulações do MATLAB, o código que implementa o PID calcula o sinal de referência (Figura 63) como mostra a Equação (3.1), sendo o resultado final um ângulo ψ_{ref} cujo valor deve ser subtraído do ângulo da velocidade ψ para que se chegue ao valor do erro.

A implementação digital do controle PID, propriamente dito, é baseada na Equação (3.2). A componente proporcional é o erro multiplicado pela constante proporcional, K_p. A componente integral é a soma dos erros multiplicados pela constante integral vezes um intervalo de tempo [40]. A Figura 64 mostra como essas componentes são calculadas.

```
double erro = REF - Sample; //Sample = Heading

double deltaTime = (millis() - lastProcess) / 1000.0;
lastProcess = millis();

//P
P = erro * kP;

//I
I = I + (erro * kI) * deltaTime;
```

Figura 64 - Componente P e I.

O intervalo de tempo *deltaTime* corresponde ao intervalo de integração e é determinado pelo tempo gasto desde a última execução do cálculo do controle até o momento presente.

A componente derivativa pode ser calculada sem o filtro como mostrado na Figura 65 [41]. A variação do ângulo de velocidade multiplicada pela constante derivativa e dividindo pelo intervalo de tempo.

```
D = (lastSample - H)* kD/ deltaTime;
lastSample = H;

// Soma tudo
pid = P + I + D;
```

Figura 65 - Componente derivativa D sem filtro.

Por fim, sinal de controle vem da soma das componentes P, I e D. O resultado do sinal do controle gerado por esses cálculos pode ser visto novamente na Figura 61 que mostra um sinal ruidoso.

Contudo, para a obtenção do melhor resultado possível se fez necessária também a implementação de um filtro para a componente derivativa. Foram trabalhadas duas versões para o algoritmo do filtro. A primeira utiliza a biblioteca *Filters* [42] que cria um filtro de um polo. A função *lowpassFilter.input()* vista na Figura 66 trata os valores da variável D. Para a criação do filtro passa baixa nesse código, o único argumento exigido pela biblioteca *Filters* foi o coeficiente de filtro N mostrado na Tabela 3.7.

```
D = (lastSample - H)* kD/ deltaTime;
lastSample = H;

DF = lowpassFilter.input(D);
```

Figura 66 - Filtro passa-baixa de um polo.

A variável DF irá receber o no valor da componente D depois de passada pelo filtro. Sendo assim, o sinal de controle se torna a soma de P, I e DF.

A segunda opção considerada para a obtenção do filtro passa baixa foi realizar uma discretização de um filtro de primeira ordem utilizando MATLAB [43]. Com o valor do coeficiente de filtro N tem-se a função de transferência para o filtro passa-baixa:

$$G(s) = \frac{0.2580}{s + 0.2580} \tag{3.2}$$

A discretização foi feita usando o tempo de amostragem de 0.2 segundos obtendo o resultado da Equação (3.3).

$$G(z) = \frac{0.05029}{z - 0.9497}\ , \tag{3.3}$$

A Equação (3.4) representa a equação de diferenças obtida a partir de $G(z)$:

$$G(z) = \frac{Y(z)}{U(z)} = \frac{0.05029}{z - 0.9497}$$

$$Y(z)z - Y(z)0.9497 = U(z)0.05029$$

$$Y(z)z = U(z)0.05029 + Y(z)0.9497$$

Aplicando a transforma z inversa, obtém-se:

$$y(n+1) = 0.9497y(n) + 0.05029u(n) \tag{3.4}$$

A Equação (3.4) é usada para filtrar a componente derivativa do controle PID, em que $u(n)$ corresponde à variável D, que deve ser filtrada.

```
double erro = REF - HEADING;

double deltaTime = (millis() - lastProcess)/1000.0;
lastProcess = millis();

//P
P = erro * kP;

//I
I = I + (erro * kI) * deltaTime;

D = (lastSample - H)* kD/ deltaTime;
lastSample = H;

y = 0.9499*y + 0.05014*D;

pid = P + I + y;
```

Figura 67 - Filtro digital.

Na Figura 67 é vista a implementação completa do algoritmo para o PID com o filtro digital. Como a variável y irá assumir os valores filtrados, o sinal de controle nesse caso passa ser da soma das componentes P, I e y.

Para fins de comparação, o sinal de controle calculado pelo *firmware* foi comparado com o sinal produzido em simulação pelo MATLAB. O objetivo foi fazer com que o *firmware* reproduzisse os resultados obtidos na simulação, conforme será mostrado no próximo capítulo.

CAPÍTULO 4 – RESULTADOS E DISCUSSÕES

Esse capítulo apresenta os resultados obtidos a partir das simulações dos experimentos feitos com o *firmware* na plataforma mais recente para o projeto LAICAnSat.

4.1 COLETA DE DADOS

Com relação à coleta de dados o *firmware* faz a inicialização completa de todos os sensores e módulo GNSS. Ele é capaz de gerar leituras sem um atraso significativo, que venha a prejudicar os resultados, apesar de ter sido notado uma maior demora de processamento quando se combina a chamada dos sensores, o módulo GNSS e as operações de escrita em arquivo. Observa-se que as operações de escrita em arquivo são as mais críticas para serem executadas quando levado em consideração o tempo de processamento.

Foi criada uma rotina para assegurar a inicialização correta da placa. Os sensores e receptor GNSS só são inicializados se o cartão de memória puder ser acessado para a escrita dos arquivos, dessa maneira impedirá que a aquisição de dados seja iniciada sem ter como salvá-los. Caso o cartão de memória não possa ser acessado é gerada uma mensagem de erro como mostrada na Figura 68.

Depois de inicializado o cartão de memória é verificado se todos os sensores estão operacionais (Figura 69).

```
Initializing SD card...Card failed, or not present
```

Figura 68 - Mensagem indicando ausência do cartão de memória.

```
Initializing SD card...card initialized.
 GPS Initialising....
BMP180 initiation successful!
BMP180 initiation successful!
MS5611 initiation successful!
Wire
LSM303 initiation successful!
L3GD20 initiation successful!
```

Figura 69 - Mensagens de inicialização dos sensores.

Depois de inicializados os sensores ocorre a programação do módulo GNSS (Figura 70) e se tudo correr corretamente o LED deve acender indicando que a coleta de dados já iniciou.

```
 * Reading ACK response: B562512061F38 (SUCCESS!)
Setting uBlox set TIM_TIM2
B5 62 6 1 8 0 D 3 0 1 0 0 0 0 20 25
 * Reading ACK response: B562512061F38 (SUCCESS!)
Setting GPS 8-16 channels CFG_GNSS1
B5 62 6 3E C 0 0 20 20 1 0 8 10 0 1 0 1 1 AC F2
 * Reading ACK response: B562512063E4C75 (SUCCESS!)
Setting SBAS 1-3 channels off CFG_GNSS2
B5 62 6 3E C 0 0 20 20 1 1 1 3 0 0 0 1 1 98 77
 * Reading ACK response: B562B562B562512063E4C75 (SUCCESS!)
Setting Galileo 0 channels off CFG_GNSS3
B5 62 6 3E C 0 0 20 20 1 2 0 0 0 0 0 0 1 94 64
 * Reading ACK response: B562512063E4C75 (SUCCESS!)
Setting BeiDou 0 channels off CFG_GNSS4
B5 62 6 3E C 0 0 20 20 1 3 0 0 0 0 0 1 1 96 6E
 * Reading ACK response: B562512063E4C75 (SUCCESS!)
Setting IMES 0-8 channels off CFG_GNSS5
B5 62 6 3E C 0 0 20 20 1 4 0 8 0 0 0 1 1 9F A6
 * Reading ACK response: B562512063E4C75 (SUCCESS!)
Setting off QZSS 0-3 channels CFG_GNSS6
B5 62 6 3E C 0 0 20 20 1 5 0 3 0 0 0 1 1 9B 90
 * Reading ACK response: B562512063E4C75 (SUCCESS!)
Setting GLONASS 8-14 channels CFG_GNSS7
B5 62 6 3E C 0 0 20 20 1 6 8 E 0 1 0 1 1 B0 16
 * Reading ACK response: B562B562B562512063E4C75 (SUCCESS!)
Setting uBlox nav5 dynamic model:
B5 62 6 24 24 0 FF FF 6 3 0 0 0 0 10 27 0 0 5 0 FA 0 FA 0 64 0 2C 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 16 DC
```

Figura 70 - Programação do módulo.

Depois da coleta iniciada são criados dois arquivos separados. Um que guarda os dados dos sensores e outro que guarda dados brutos de observação que serão convertidos em RINEX no pós processamento. É importante que as informações dos sensores (Figura 71) e do módulo GNSS sejam salvas em arquivos diferentes pois como mostrado na Figura 72, os dados fornecidos pelo módulo GNSS são dados binários. A mistura desses dados com os dos sensores dificultaria seu estudo.

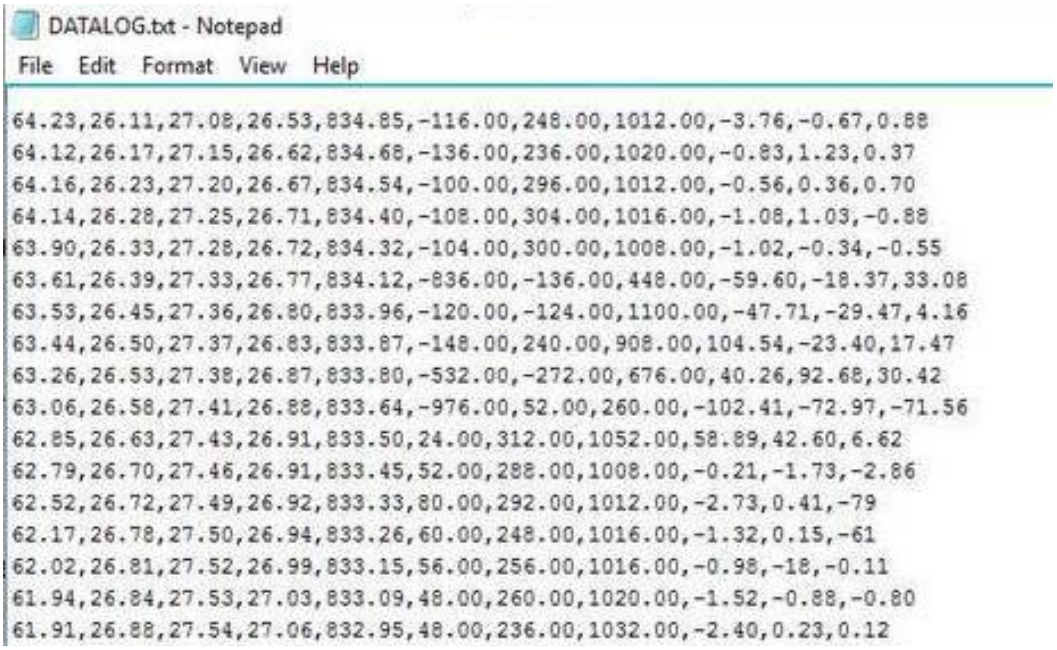


Figura 71 - *Datalog* sensores.

Salvo em formato .csv, o arquivo de dados dos sensores se torna compatível com editores de planilha.

Figura 72 - Dados brutos binários.

Mudando a extensão do arquivo de dados brutos de .txt para .ubx é possível fazer a conversão para RINEX dos dados binários através do aplicativo RTKCONV. Usando o RTKCONV versão 2.3.4 para conversão de dados brutos em dados de navegação e observação, utiliza-se a seguinte configuração mostrada na Figura 73.

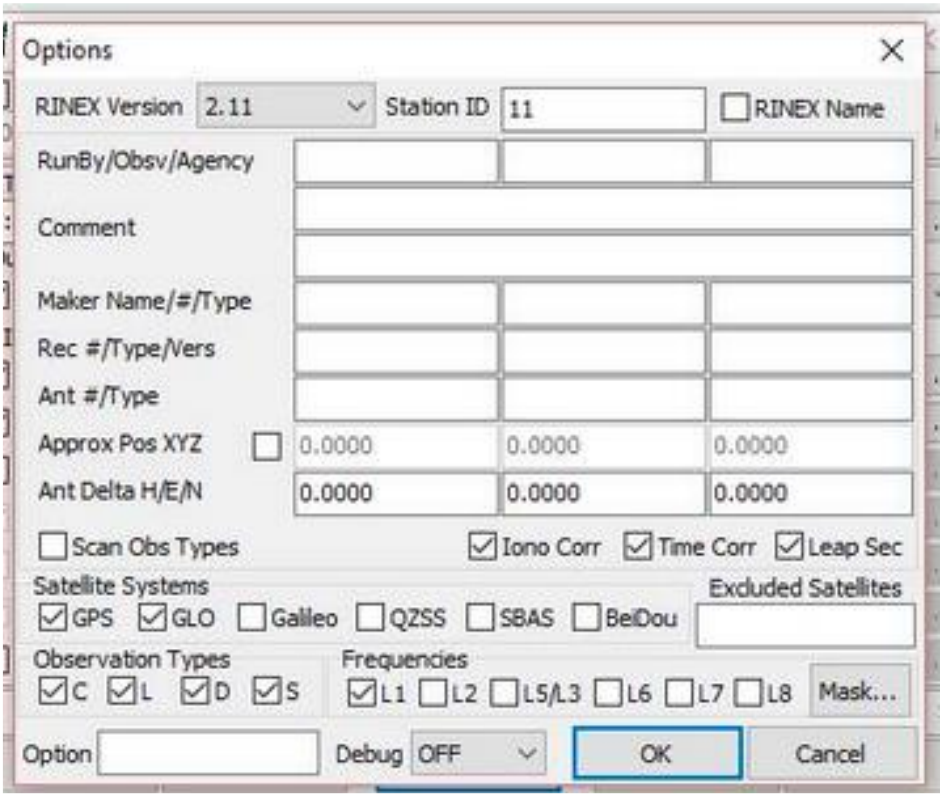


Figura 73 - Configuração RTKCONV.

Depois de escolhido o diretório onde devem ser salvos os resultados da conversão, é esperado que sejam gerados dois arquivos. Um arquivo de extensão .nav, com dados de navegação, e outro de extensão .obs, com dados de observação. Mudando a extensão dos arquivos gerados para .txt é observada nas figuras 74 e 75 as características dos arquivos esperados em RINEX.

gnss11NAV.txt - Notepad

File Edit Format View Help

```
     2.11           N: GPS NAV DATA                         RINEX VERSION / TYPE
RTKCONV 2.4.3 b8                         20160706 003103 UTC PGM / RUN BY / DATE
log: C:\Users\MA\Desktop\raw data\gnss11.ubx                 COMMENT
format: u-blox                                               COMMENT
    0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00          ION ALPHA
    0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00          ION BETA
     .000000000000E+00  .000000000000E+00        0     1792 DELTA-UTC: A0,A1,T,W
     0                                                       LEAP SECONDS
                                                             END OF HEADER
 1 16  6  9 20  0  0.0  .231075100601E-04  .125055521494E-11  .000000000000E+00
     .102000000000E+03 -.817500000000E+02  .438196824085E-08  .412206228238E+00
    -.414438545704E-05  .555496045854E-02  .960379838943E-05  .515364889908E+04
     .417600000000E+06  .447034835815E-07 -.241125690683E+01  .707805156708E-07
     .964619103046E+00  .190656250000E+03  .467437633854E+00 -.802533428739E-08
    -.192508018732E-09  .100000000000E+01  .190000000000E+04  .000000000000E+00
     .240000000000E+01  .000000000000E+00  .512227416039E-08  .102000000000E+03
     .411756000000E+06  .400000000000E+01
```

Figura 74 - Arquivo .nav.

gnss11OBS.txt - Notepad

File Edit Format View Help

```
     2.11           OBSERVATION DATA    M (MIXED)           RINEX VERSION / TYPE
RTKCONV 2.4.3 b8                         20160706 003103 UTC PGM / RUN BY / DATE
log: C:\Users\MA\Desktop\raw data\gnss11.ubx                 COMMENT
format: u-blox                                               COMMENT
                                                             MARKER NAME
                                                             MARKER NUMBER
                                                             OBSERVER / AGENCY
                                                             REC # / TYPE / VERS
                                                             ANT # / TYPE
  4117966.6746 -4556056.7521 -1720167.5121                   APPROX POSITION XYZ
        0.0000        0.0000        0.0000                   ANTENNA: DELTA H/E/N
     1     1                                                 WAVELENGTH FACT L1/2
     4    C1    L1    D1    S1                               # / TYPES OF OBSERV
  2016     6     9    18    22    8.9995587     GPS         TIME OF FIRST OBS
  2016     6     9    18    23   21.9995587     GPS         TIME OF LAST OBS
                                                             END OF HEADER

 16  6  9 18 22  8.9995587  0  6R 6R21G 3G23G 9R16
  21496242.239   114708072.2982       1720.192          40.000
  21990787.509   117677203.2302      -1439.979          50.000
  21403500.055   112476155.3352       2213.003          48.000
  21673406.604   113894515.5462        480.649          51.000
  23147065.957   121638645.5192       1396.733          45.000
  20155423.725                        -836.706          26.000
```

Figura 75 - Arquivo .obs.

4.2 CONTROLE PID

A pesar de o MCU utilizado dar suporte à comunicação serial, o *design* da atual versão da placa não fornece suporte para conexão de servo motores. A estrutura que vai permitir a atuação de servo motores é um passo futuro a ser implementado no projeto.

Para o teste com a placa foram usados os valores de coordenadas e ângulos de direção da velocidade gerados pela simulação do MATLAB. Esses valores foram guardados em um arquivo de texto para serem usados nos testes.

Os testes decorreram de forma que o *firmware* lesse os valores desse arquivo e calculasse o sinal de controle a partir deles. Dessa forma o resultado esperado para a saída do controle teria que ser igual a saída gerada pela simulação.

A escolha da melhor solução para o controle PID partiu da análise de duas abordagens de como deveria se fazer a filtragem da componente derivativa. A implementação do filtro foi o que possibilitou que os resultados obtidos com o sistema real fossem semelhantes ao sistema simulado. Na Figura 76 é mostrado o sinal de controle do sistema simulado.

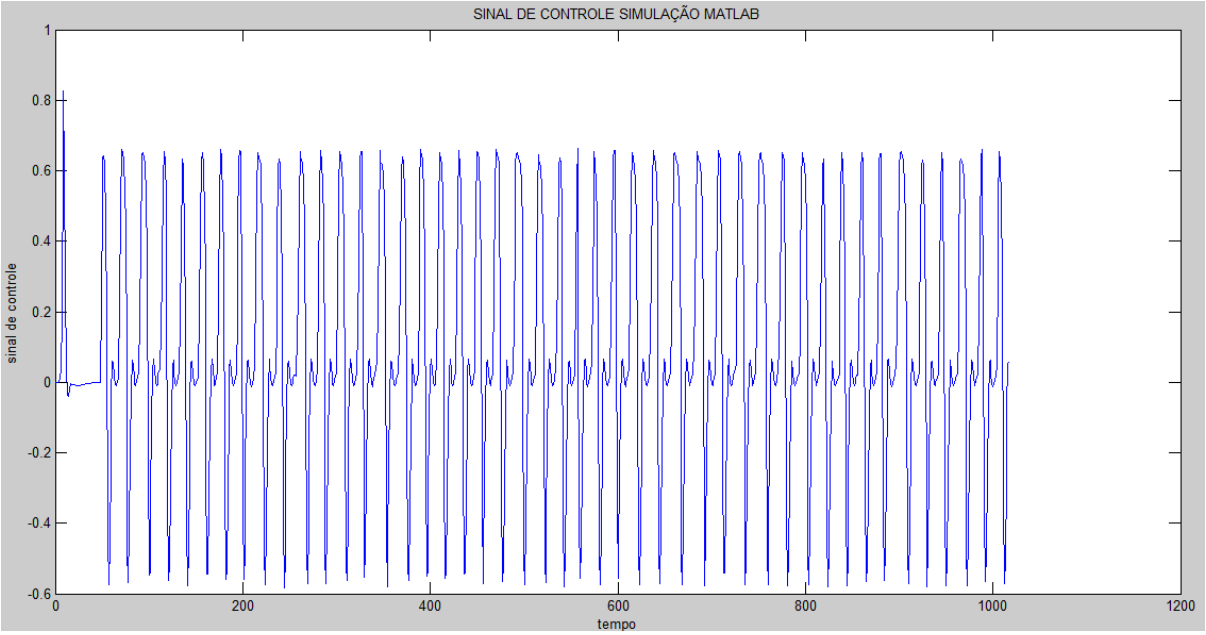


Figura 76 - Sinal de controle MATLAB.

Usando algoritmo disponibilizado pela biblioteca *Filters.* [42] o resultado para o sinal de controle gerou a Figura 77. Nota-se a discrepância entre esse resultado e do da Figura 76.

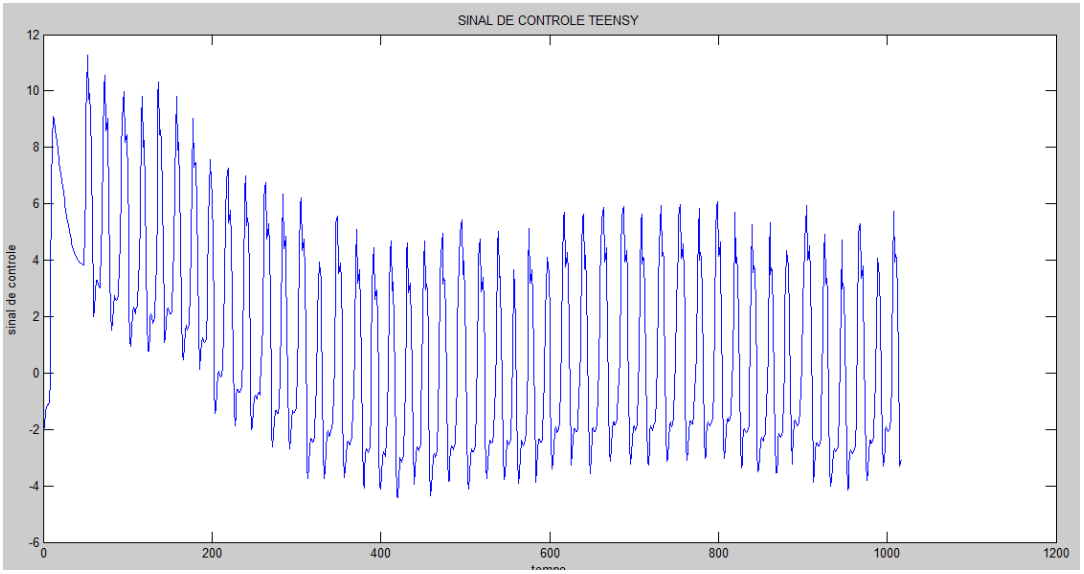


Figura 77 - PID usando biblioteca *Filters*.

Utilizando o filtro digital da Equação (3.4) para a implementação do controle PID foi obtida a Figura 78.

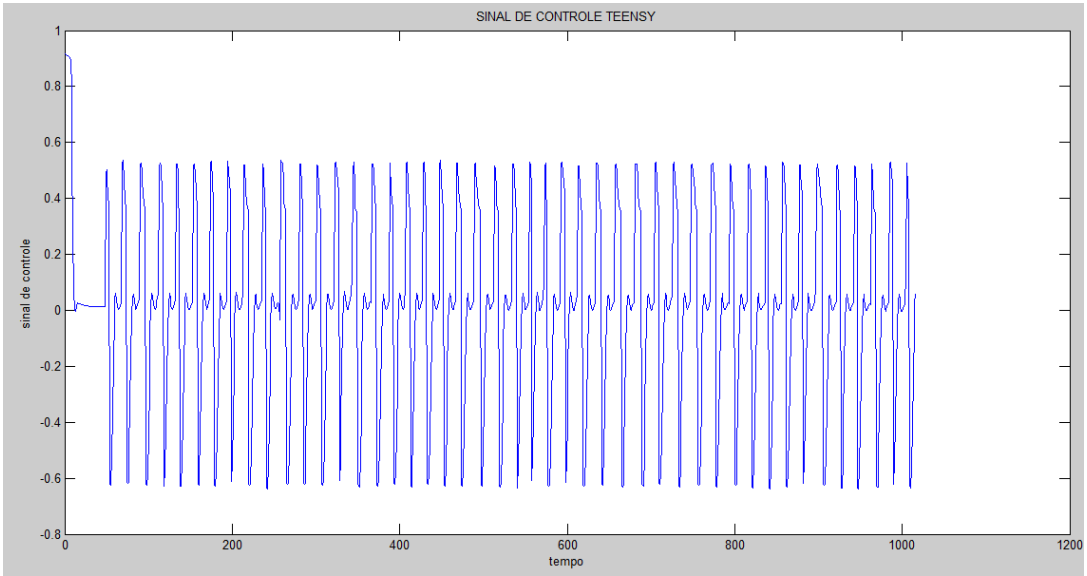


Figura 78 - PID usando filtro no canal derivativo.

O resultado que melhor se aproxima do sinal de controle simulado no MATLAB é o apresentado na Figura 78, onde é implementado o filtro digital.

Testes em voo são necessários para o melhor entendimento das diferenças nas amplitudes do sinal de controle, e estão previstas para trabalhos futuros.

5 CONCLUSÃO

Esse trabalho apresentou o desenvolvimento do firmware para a última versão da plataforma LAICAnSat que tem por objetivo coletar dados de sensores, dados brutos de um receptor GNSS e fazer o controle da trajetória da descida de uma carga útil para uma região que se possa realizar o resgate.

Foram mostrados os contextos das missões e projetos no qual a plataforma está inserida a fim de apresentar a motivação para este trabalho.

Para o alcance dos objetivos da plataforma, foi projetado o *firmware* utilizando o ambiente de desenvolvimento Arduino. O sistema de bordo projetado é constituído por componentes acessíveis e populares o que proporciona a praticidade de se ter uma literatura vasta sobre projetos que tem uma arquitetura similar.

É importante observar que aspectos de *hardware* também influenciaram no *firmware* criado, como por exemplo o endereço dos sensores no protocolo I²C. Por conta disto, o conhecimento de aspectos computacionais se mostram importantes no projeto do um sistema embarcado.

O ambiente de desenvolvimento Arduino tornou o trabalho muito prático apesar da oportunidade de trabalhar com o modelo de um receptor GNSS de nível profissional ter se mostrado um desafio. A placa projetada é versátil, podendo ser usada em diferentes tipos de aplicações, como BalloonSats e vants necessitando apenas de algumas adaptações para a inclusão de outros recursos. O módulo GNSS utilizado é uma ferramenta poderosa que pode levar a estudos mais complexos de controle de trajetória com alta precisão.

Baseado na linguagem Arduino, o *firmware* projetado tem sua própria biblioteca desenvolvida com princípios de programação orientada a objeto para dar uma imagem limpa e objetiva. Ele dá ao módulo GNSS a configuração correta para a missão, coleta dados de sensores e armazena cada uma dessas informações em dois arquivos de texto distintos no cartão de memória para que essas informações sejam utilizadas no pós processamento.

Na análise do controle PID, os resultados apresentados representam a implementação para controle de trajetória que será implantado na plataforma permitindo a realização de missões como as do projeto Kuaray, que lançará um balão estratosférico carregando a plataforma. Nesse tipo de missão o controle implementado nesse trabalho é imprescindível pois facilita a etapa de resgate da

carga útil no término da missão. O *firmware* está disponível na página https://github.com/laicansat.

PERSPECTIVAS FUTURAS

Este trabalho faz parte de um esforço para construir uma plataforma CubeSat auto recuperável de baixo custo de possibilite experimentos em altas altitudes. Nesse sentido já foi feito o estudo e modelagem aerodinâmica bem como a identificação de um sistema mínimo para o funcionamento do computador de bordo e a padronização da estrutura que compõe a carga útil.

Dessa maneira o desenvolvimento de um *firmware* que torne o sistema funcional é de extrema importância para a evolução do projeto. Portanto, como trabalhos futuros para a plataforma LAICAnSat destacam-se:

- Validação em voo do controle PID implementado;
- Implementação sistema de comunicação via Xbee;
- Implementação do sistema de atuação via servo motores;
- Validação em voo do *firmware* implementado nos diversos projetos mencionados;
- Implementação de coleta de *time Stamps* junto com a captura de dados dos sensores, para que os dados possam ser analisados de forma mais completa;
- Adaptação do código para Linguagem C, principalmente a parte que implementa a estratégia de controle, para dar maior robustez ao sistema;
- Separar a coleta do módulo GNSS dos demais sensores. Como o GNSS captura dados em uma frequência inferior à dos outros sensores, estes ficam operando abaixo de sua capacidade de trabalho;

Sugere-se que haja uma adaptação para a linguagem C por motivos de otimização e eficiência. Aplicações em C são mais robustas e possuem maior velocidade de execução. Elas também permitem um gerenciamento melhor da memória alocada. Outra vantagem significativa da Linguagem C, é que ela é altamente versátil, compatível com uma grande variedade de sistemas operacionais e plataformas, com uma quantidade mínima de alterações necessárias no código fonte.

O projeto do *firmware* gerou algumas reflexões acerca da estrutura de *hardware* da plataforma LAICAnSat. Então aqui serão sugeridas algumas melhorias para serem implementadas no sistema embarcado

Apesar de estar no padrão PC104, foi percebido que o barramento ISA está sendo subutilizado e por consequência o microcontrolador também.

O microcontrolador faz uso de somente seis dispositivos, sendo eles cinco sensores, um módulo GNSS e um cartão de memória. Os pinos não conectados a nenhum desses dispositivos simplesmente estão inacessíveis pois o barramento ISA não possui conexão com estes pinos.

A primeira sugestão para trabalhos futuros com o projeto da placa, é fazer o acesso do barramento ISA a todos os pinos disponíveis no microcontrolador. Inclusive isso permitirá o suporte para o sistema de acionamento dos servos motores que serão responsáveis pela atuação no sistema de controle de trajetória. Além de permitir a inclusão de quaisquer adaptações necessárias às missões futuras.

Quanto aos sensores, o ponto crítico de sua performance se encontra no sensor SHT15, que por dividir o sinal de *clock* com o cartão de memória cria um conflito, exigindo que seja feita a reinicialização do cartão sempre que se faça nele a escrita de algum dado. Para a melhoria nessa área, se faz necessário escolher outro sensor de temperatura e umidade que faça uso de um barramento I²C. Como barramento I²C faz uso de apenas dois pinos, isso pouparia espaço para que sejam adicionados outros tipos de dispositivos ao microcontrolador como por exemplo, outras portas do módulo GNSS.

O módulo LEA-M8T possui cinco portas I/O independentes disponíveis para saída de informações. Sendo elas de protocolos DDC, UART1, UART2, I²C e SPI, mas atualmente somente a porta UART1 está sendo usada. Estas portas independentes podem gerar dados de protocolos NMEA ou UBX, a taxas de atualizações diferentes, o que habilita uma maior gama de recursos para serem utilizados, aumentando a riqueza dos resultados que placa pode gerar.

PUBLICAÇÕES

Durante o desenvolvimento deste trabalho foi aceito um artigo [3] na Aerospace Conference 2017, do Institute of Eletrical and Eletronics Engineers (IEEE), que será publicado em março de 2017. O título do trabalho a ser publicado é " Trajectory Control System for the LAICAnSat-3 Mission" e o *firmware* apresentado nesse trabalho de graduação constitui parte do artigo em questão.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] R. R. Dias, A. K. de Castro, , & C. Cappelletti, LAICAnSat-3: A Mission for Testing a New Electronic and Electronic and Telemetry and Tracking System, 2014.

[2] P. H. Doria Nehme, R. Alves Borges, C. Cappelletti, S. Battistini, Development of a meteorology and remote sensing experimental platform: The laicansat-1, in: Aerospace Conference, 2014 IEEE, IEEE, 2014, pp. 1–7.

[3] M. A. L. Holanda, Y. M. Honda, S. Battistini, & R. A. Borges, Trajectory Control System for the LAICAnSat-3 Mission, in: Aerospace Conference, 2017 IEEE, IEEE, 2017, pp. 1–7.

[4] C. Koehler, “Balloonsat: Missions to the edge of space,” in 16th Annual/USU Conference on Small Satellites,2004.

[5] R.R. Dias, Desenvolvimento de um Sistema de Comunicação para Rastreamento e Telemetria da Plataforma LAICAnSat, 2014, Publicação FT.TG-n 12/2014, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 71p.

[6] M. F. S. Alves, A. P. Wernke, F. C. Pereira, D. H. Gomes, G. S. Lionc¸o, C. H. F. L. Domingos, D. B. d. Trindade, C. Cappelletti, M. N. D. B. Junior, S. Battistini, and R. A. Borges, “Design of the structure and reentry system for the laicansat-3 platform.”, 2014.

[7] P. H. Nehme, LAICAnSat: Uma Plataforma Experimental para Balões de Pequeno Porte, 2014.

[8] SILVA, A. V. S.; NORONHA, B. H. A.; Identificação Aerodinâmica de um Sistema Paraquedas-Carga Útil para a Plataforma LAICAnSat, 2014, Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-n 11/2014, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 52p.

[9] A. K. de Castro, Desenvolvimento de uma Placa Eletrônica do Sistema Mínimo da plataforma LAICANSAT, 2015.

[10] Ublox M8T, [Online]. Disponível:https://www.ublox.com/en/product/neoleam8t/ [Acesso em 20 Outubro 2016].

[11] UART, Serial comunication [Online]. Disponível em: https://learn.sparkfun.com/tutorials/serial-communication [Acesso em 20 Outubro 2016].

[12] I2C, Protocol [Online]. Disponível em: https://learn.sparkfun.com/tutorials/i2c [Acesso em 20 Outubro 2016].

[13] SPI, Serial Peripheral Interfacel [Online]. Disponível em: https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi [Acesso em 20 Outubro 2016].

[14] 2-Wire-Bus, [Online]. Disponível em: https://www.sparkfun.com/products/13683 [Acesso em 20 Outubro 2016].

[15] Configuração Inicial Teensy 3.1. Disponível em: https://www.pjrc.com/teensy/teensyduino.html [Acesso em 20 de Outubro 2016].

[16] Biblioteca BMP180. Disponível em: https://github.com/sparkfun/BMP180_Breakout [Acesso em 20 de Outubro 2016].

[17] Biblioteca SHT15. Disponível em: https://github.com/practicalarduino/SHT1x [Acesso em 20 de Outubro 2016].

[18] Biblioteca MS5611. Disponível em: https://github.com/RobTillaart/Arduino/tree/master/libraries/MS5611 [Acesso em 20 de outubro 2016].

[19] Biblioteca L3G. Disponível em: https://github.com/pololu/l3g-arduino [Acesso em 20 de Outubro 2016].

[20] Biblioteca LSM303. Disponível em: https://github.com/adafruit/Adafruit_LSM303 [Acesso em 20 de Outubro 2016].

[21] C. T. Pozzer, Introdução a Programação Orientada à Objetos na Linguagem C++, 2009.

[22] Orientação a Objetos em C++. Disponível em: http://www.inf.pucrs.br/~pinho/PRGSWB/OO/oocpp.html [Acesso em 18 de Outubro 2016].

[23] Programação Orientada a Objetos em C++ Disponível em: http://www.cpdee.ufmg.br/~jramirez/disciplinas/cdtn/cap4c++.pdf [Acesso em 20 de Outubro 2016].

[24] Biblioteca SD. Disponível em: https://github.com/PaulStoffregen/SD [Acesso em 20 de Outubro 2016].

[25] Biblioteca SD_t3. Disponível em: https://github.com/PaulStoffregen/SD [Acesso em 20 de Outubro 2016].

[26] Biblioteca SD_t3. Disponível em: https://github.com/PaulStoffregen/SPI [Acesso em 20 de Outubro 2016].

[27] Configuração cartão SD. Disponível em: https://www.arduino.cc/en/Reference/SD [Acesso em 20 de Outubro 2016]

[28] Interface SD/SPI. Disponível em: http://playground.arduino.cc/Learning/SDMMC [Acesso em 20 de Outubro 2016]

[29] Arduino DataLogger. Disponível em: https://www.paulotrentin.com.br/eletronica/data-logger-sdmmc-com-arduino/ [Acesso em 18 de novembro 2016]

[30] Modificação Classe SD. Disponível em https://github.com/adafruit/SD/issues/7 [Acesso em 18 de novembro 2016]

[31] Configuração módulo GNSS. Disponível em https://github.com/emlid/ReachView/blob/master/rtklib_configs/GPS_GLONASS_1Hz.cmd [Acesso 15 de outubro 2016]

[32] Getting Started with Ublox. Disponível em http://ava.upuaut.net/?p=738 [Acesso 10 de outubro 2016]

[33] ERGO-GEN3-WORKING-ARDUINO. Disponível em: https://github.com/simonfrfr/ERGO-GEN3-WORKING-ARDUINO/tree/master/GPS_UBLOX [Acesso 10 de outubro 2016]

[34] uBlox NEO-6Q / uBlox MAX-6Q. Disponível em https://ukhas.org.uk/guides:ublox6 [Acesso 10 de outubro 2016]

[35] UBX_parser_master. https://github.com/simondlevy/UBX_Parser Disponível em [Acesso 10 de novembro 2016]

[36] L.E. Dubins, “On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents”. American Journal of mathematics, v. 79, n. 3, p. 497-516, 1957.

[37] J.A Reeds, L.A. Shepp. Optimal paths for a car that goes both forwards and backwards, Pacific J. Math., 145 (1990), pp. 367–393.

[38] Y. H. M. Honda. Análise e controle da trajetória do LAICAnSat-3 – Brasília, DF, 2016- 68 p. : il.

[39] Li, Y. and Ang, K.H. and Chong, G.C.Y. (2006) PID control system analysis and design. IEEE Control Systems Magazine 26(1):pp. 32-41.

[40] Arduino PID Library. Disponível em: http://playground.arduino.cc/Code/PIDLibrary [Acesso 15 de novembro 2016]

[41] Arduino + PID. Disponível em: https://www.youtube.com/watch?v=txftR4TqKYA [Acesso 15 de novembro 2016]

[42] Arduino Filters Library. Disponível em: http://playground.arduino.cc/Code/Filters [Acesso em 15 de novembro 2016]

[43] Digital filter Low-pass filter Using Arduino. Disponível em: https://www.youtube.com/watch?v=CPpOJsHuMsM [Acesso em 15 de novembro 2016]

[44] Big data a big market for small satellites. Disponível em : http://spacenews.com/big-data-a-big-market-for-small-satellites/ [Acesso em 28 de Novembro 2016]

[45] Small Satellites Are The Next Big Thing. Disponível em: http://www.eatglobe.com/news/environment/2832-small-satellites-are-the-next-big-thing.html [Acesso em 28 de Novembro 2016]

[46] Here's why small satellites are so big right now. Disponível em: http://fortune.com/2015/08/04/small-satellites-newspace/ [Acesso em 28 de Novembro 2016]

[47] MarCO! CubeSats set to support NASA InSight mission. Disponível em: http://www.spaceflightinsider.com/missions/solar-system/marco-cubesats-set-support-nasa-insight-mission/ [Acesso em 28 de Novembro 2016].

[48] OGATA, K.; Modern control engineering. Prentice Hall PTR, 2001.

[49] Wikipédia. Disponível em: https://pt.wikipedia.org/ [Acesso em 28 de Novembro 2016]

[50] Disponível em: https://sites.google.com/a/aerospace.unb.br/renato/project [Acesso em 28 de Novembro 2016]

[51] Primeiro CubeSat Brasileiro. Disponível em: http://www.qsl.net/py4zbz/aesp.htm [Acesso em 29 de Novembro 2016].

[52] Teensy 3.1. Disponível em: https://www.pjrc.com/teensy/ [Acesso em 29 de Novembro 2016].

[53] Sensor BMP180. Disponível em: http://blog.filipeflop.com/ [Acesso em 29 de Novembro 2016].

[54] Sensor L3GD20H. Disponível em: http://www.electrokit.com/ [Acesso em 29 de Novembro 2016].

[55] Sensor L3M303. Disponível em: http://www.waveshare.com/ [Acesso em 29 de Novembro 2016].

[56] Sensor MS5611. Disponível em: http://www.hobbytronics.co.uk/ [Acesso em 29 de Novembro 2016].

[57] Sensor SHT15. Disponível em: http://www.sparkfun.com/ [Acesso em 29 de Novembro 2016].

[58] Datasheet sensor MS5611. Disponível em: http://www.te.com/usa-en/product-CAT-BLPS0036.html [Acesso em 06 de Dezembro 2016]

[59] N. Slegers, M. Costello. Aspects of control for a parafoil and payload system. Journal of Guidance, Control, and Dynamics, v. 26, n. 6, p. 898–905, 2003.

[60] E. Scheuermann. Combined lateral and longitudinal control of parafoils using upper-surface canopy spoilers. Journal of Guidance, Control, and Dynamics, American Institute of Aeronautics and Astronautics, v. 38, n. 11, p. 2122–2131, 2015.

[61] M. R. Cacan. Autonomous control of gps denied guided airdrop systems using radio beacon feedback. In: AIAA Guidance, Navigation, and Control Conference. [S.l.: s.n.], 2016. p. 1143.

[62] B. J. Rademacher. In-flight trajectory planning and guidance for autonomous parafoils. Journal of guidance, control, and dynamics, v. 32, n. 6, p. 1697–1712, 2009.